UNIVERSIDADE D
COIMBRA

Rui Pedro das Neves Dias

# ANÁLISE DE PLATAFORMAS BLOCKCHAIN

**Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas de Informação orientada pelo Professor Doutor Paulo Rupino da Cunha da Faculdade de Ciências e Tecnologia, e pelo Professor Doutor Manuel Paulo de Albuquerque Melo da Faculdade de Economia e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.**

Setembro de 2019

Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

# ANÁLISE DE PLATAFORMAS BLOCKCHAIN

Rui Pedro das Neves Dias

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas de Informação orientada pelo Professor Doutor Paulo Rupino da Cunha, da Faculdade de Ciências e Tecnologia, e pelo Professor Doutor Manuel Paulo de Albuquerque Melo da Faculdade de Economia e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática

Setembro de 2019

UNIVERSIDADE D
COIMBRA

# Resumo

Atualmente é possível encontrar várias plataformas que implementam a tecnologia *Blockchain*. No entanto, e para poder fazer uma escolha fundamentada, é necessário estudar a sua performance e características em variados ambientes de execução, através do estudo de testes de *benchmark*.

Ao longo deste documento é feito um estudo da tecnologia *Blockchain*, e das várias plataformas que a implementam. Estas são posteriormente comparadas tendo em conta as suas características e funcionalidades, recorrendo para isso, a um conjunto de parâmetros apresentados.

Estudos no âmbito de *benchmarking* a plataformas *blockchain* permitiram que fosse desenvolvido um sistema de *benckmark*, através de uma adaptação a uma ferramenta existente, *Gauge*. O sistema foi implementado, e executado sobre a plataforma *Hyperledger Fabric*, uma plataforma *Blockchain*, implementada pela *Linux Foundation.* Dos testes foi possível a recolha de métricas relativas ao *throughput*, latência, número de transações bem-sucedidas e dados do consumo dos recursos dos componentes da plataforma.

De uma análise dos resultados obtidos, é possível concluir que o tipo de operação realizada sobre a *blockchain*, o tamanho do bloco inserido, e a política de aprovação de uma transação, influenciam diretamente a performance da plataforma. Outros testes também foram executados, sem, no entanto, apresentarem conclusões significantes.

# Palavras-Chave

Plataformas *blockchain*, Tecnologia de *Ledgers* Distribuídos, *Smart Contracts,* Aplicações Descentralizadas, *Benchmark*, *Hyperledger Fabric*

# Abstract

Currently, it is possible to find several platforms that implement Blockchain technology. However, in order to make an informed choice, it is necessary to study its performance and characteristics in various execution environments, through the study of benchmark tests.

Throughout this document a study is made of the Blockchain technology, and the various platforms that implement it. These are then compared considering their characteristics and functionalities, using a set of parameters presented.

Benchmarking studies of blockchain platforms allowed the development of a benckmark system, through an adaptation to an existing tool, Gauge. The system was implemented and executed on the Hyperledger Fabric platform, a Blockchain platform, implemented by the Linux Foundation. From the tests it was possible to collect metrics related to throughput, latency, number of successful transactions and resource consumption data from the platform components.

From an analysis of the results obtained, it is possible to conclude that the type of operation performed on the blockchain, the size of the block inserted, and the endorsment policy of a transaction, directly influence the performance of the platform. Other tests were also performed, without presenting significant conclusions.

# Keywords

Blockchain platforms, Distributed Ledger Technology, Smart Contracts, Descentralized Applications, Benchmark, Hyperledger Fabric

# Agradecimentos

Ao Professor Doutor Paulo Rupino da Cunha, orientador da dissertação, pelo saber, interesse, espírito crítico e rigor académico, e por todas as valiosas contribuíções para o trabalho. Um agradecimento especial, por me ter apresentado ao tema que originou a realização da dissertação.

Ao Professor Doutor Manuel Paulo de Albuquerque Melo, co-orientatador da dissertação, pelo saber e experiência prática, e, de igual forma, pelas suas contribuições para o desenvolvimento do trabalho.

A mais sincera gratidão à minha familia, pelo apoio e motivação que sempre me deram, sem nunca desistir de mim. Aos meus pais, pela inspiração que todos os dias me passam, e por terem feito tudo ao seu alcance para me tornarem na pessoa que sou. Ao meu irmão, pela sua compreensão e habitual motivação com que sempre posso contar.

Aos meus amigos que conheci no meio académico e que tenho o prazer de os levar para a vida.

A todos, um muito obrigado.

# Índice

# Tabela de acrónimos

| Acrónimo | Descrição |
|---|---|
| **ASIC** | *Application-Specific Integrated Circuits* |
| **BFT** | *Bizantine Fault-Tolerant* |
| **CA** | *Contract Account* |
| **CFT** | *Crash Fautl-Tolerant* |
| **DApp** | *Decentralized Application* |
| **DLT** | *Distributed Ledger Technology* |
| **EOA** | *External Owned Account* |
| **EVM** | *Ethereum Virtual Machine* |
| **HSM** | *Hardware Security Model* |
| **JVM** | *Java Virtual Machine* |
| **KYC** | *Know Your Costumer* |
| **MSP** | ***Membership Service Provider*** |
| **NIST** | *National Institute of Standards and Technology* |
| **PBZT** | *Practical Byzantine Fault Tolerance* |
| **PKI** | *Public Key Infrastructure* |
| **PoAC** | *Proof of Activity* |
| **PoAU** | *Proof of Authority* |
| **PoB** | *Proof of Burn* |
| **PoS** | *Proof of Stake* |
| **PoW** | *Proof of Work* |
| **PoI** | *Proof of Identity* |
| **PoP** | *Proof of Publication* |
| **PTM** | *Peer Transaction Manager* |
| **RPC** | *Remote Procedure Call* |
| **SGX** | *Software Guard eXtensions* |
| **SHA** | *Secure Hash Algorithm* |
| **SLA** | *Service Level Agreement* |
| **TEE** | *Trusted Execution Environment* |
| **TLS** | *Transport Layer Security* |
| **TPS** | *Transações Por Segundo* |

| | |
|---|---|
| **UML** | ***U**nified **M**odeling **L**anguage* |
| **UNL** | ***U**nique **N**ode **L**ist* |
| **UTXO** | ***U**nspent **T**ransaction **O**utput* |

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1
# Introdução

O presente documento nasce do trabalho realizado pelo aluno Rui Pedro das Neves Dias na Dissertação do Mestrado de Engenharia Informática, do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra, inserida no ramo de Sistemas de Informação, referente ao ano letivo 2018/2019.

A Dissertação decorre no Departamento de Engenharia Informática sob a orientação do Professor Doutor Paulo Rupino da Cunha do Departamento de Engenharia Informática da Universidade de Coimbra, e coorientação do Professor Doutor Manuel Paulo de Albuquerque Melo da Faculdade de Economia da Universidade de Coimbra.

No decorrer deste capítulo são apresentadas a motivação e objetivos para a realização da dissertação, seguido da apresentação dos contributos do aluno, terminando com a apresentação da estrutura do documento em causa.

## 1.1 Motivação e objetivos

A tecnologia *Blockchain* surge como uma solução ao problema do *double-spending*, encontrado em todas as cripto moedas, que consiste na possibilidade de gastar a mesma moeda virtual mais que uma vez, por parte do mesmo utilizador (Sudhir Khatwani, 2018). A sua primeira aplicação prática surgiu com a implementação da *Bitcoin*, apresentada por Nakamoto (2008). O evoluir da tecnologia traz consigo novas plataformas que a implementam, cada uma com diferentes características e limitações. Certas plataformas passam a suportar o uso de aplicações descentralizadas e *smart contracts* – *scripts* alocados na *blockchain* que define um conjunto de regras que governam uma transação (Gupta, 2017). As suas características únicas têm vindo a suscitar um crescente interesse da indústria, demonstrando que o seu domínio de aplicação se estende para além da cripto moeda, encontrando-se exemplos de aplicações em imensas indústrias, como é o caso do controlo do comércio de diamantes de sangue (Minkenberg, 2015), das melhorias na cadeia de valor (Marr, 2018) e na logística marítima (Gronholt-Pedersen, 2018) ou até mesmo a revolução da indústria musical (Heap, 2017).

Ao longo do desenvolvimento desta dissertação foi, por parte do aluno, realizado um trabalho de identificação e ensaio das possibilidades práticas desta tecnologia. O primeiro momento da dissertação compreendeu o estudo dos conceitos envolventes da tecnologia *Blockchain*. No segundo momento, foi feito um estudo das plataformas que implementam a tecnologia e dos trabalhos existentes na área de *benchmarking* a estas. Posteriormente, foi desenhado um sistema de *benckmark*, que utilizou uma plataforma de testes de *benchmark* já existente, *Gauge* (Persistent Systems, 2019c), tendo sido feita uma adaptação do código desta pelo aluno. A proposta inicial para o desenvolvimento da dissertação compreendia a realização de testes de *benchmark* a duas plataformas *Blockchain* de implementação privada, nomeadamente a plaforma *Quorum* (JPMorgan Chase & Co, 2019) e *Hyperledger Fabric* (The Linux Foundation, 2019f). No entanto, no espaço temporal em que foi realizada, e de acordo com o trabalho desenvolvido pelo aluno, os testes foram executados apenas sobre a

plataforma *Hyperledger Fabric*. A execução e análise dos resultados dos testes deu ao aluno um entendimento sobre os componentes, arquitetura e *workflow* da plataforma.

## 1.2 Contributos

Feita a apresentação da motivação e objetivos da dissertação, são agora apresentados na Tabela 1, os contributos do aluno ao longo do desenvolvimento do trabalho.

Tabela 1 - Contributos do aluno

| Contributo | Análise de Plataformas Blockchain |
|---|---|
| ID | 1 |
| Descrição | Este contributo resulta da recolha teórica dos conceitos relacionados com a Tecnologia Blockchain encontrados na literatura atual, tendo como resultado desse trabalho o Capítulo 3. |
| Contributo | *Scripts* de automação |
| ID | 2 |
| Descrição | Deste contributo fazem parte um conjunto de três *scripts* desenvolvidos na linguagem de programação Python, cujo objetivo é o auxílio na automação da execução dos testes de *benchmark*. Estes encontram-se na diretoria *"scripts"*. |
| Contributo | *Scripts* de *parsing* dos resultados obtidos |
| ID | 3 |
| Descrição | Este contributo é composto por um conjunto de dois *scripts* desenvolvidos na linguagem de programação Python, cujo objetivo é fazer o *parsing* dos resultados obtidos da realização dos testes de *benchmark*. Estes encontram-se na diretoria *"scripts"*. |
| Contributo | *Script* para geração de artefactos de uma rede *Hyperledger Fabric* |
| ID | 4 |
| Descrição | Este contributo é o *script* utilizado para gerar os artefactos criptográficos da rede *Hyperledger Fabric*, tais como os certificados de identidade, o *genesis block* e o canal ou canais da rede, sendo utilizado sempre que são realizadas alterações na rede. Este encontra-se na diretoria *"scripts"*. |
| Contributo | Resultados obtidos |
| ID | 5 |
| Descrição | Este contributo é constituído por todos os ficheiros *.xlsx* com os dados dos resultados tratados e organizados, juntamente com todos os gráficos gerados a partir desses resultados. |

Os contributos de código gerado podem ser encontrados no repositório: https://github.com/ruipedrodias94/analise_plataformas_blockchain. Este é privado, e apenas acessível por *password* entregue aos orientadores.

## 1.3 Estrutura do documento

Terminada a introdução, o restante documento encontra-se estruturado da seguinte forma: o Capítulo 2 compreende o planeamento da dissertação para os dois semestres em que esta decorreu, apresentando as datas para a realização de cada tarefa, e as respetivas justificações para os desvios temporais verificados. No Capítulo 3 é descrito o estado da arte da tecnologia *Blockchain*, dando ao leitor uma introdução sobre os conceitos base desta, usados ao longo da dissertação. No Capítulo 4 são apresentados quatro exemplos de plataformas *Blockchain*, duas pertencendo ao domínio de *Blockchain* sem permissões e as outras duas ao domínio de *Blockchain* com permissões, tendo sido dado maior foco às duas últimas uma vez que estas vão estar na base do desenvolvimento do trabalho apresentado. No Capítulo 5 é realizado um estudo e detalhe dos trabalhos existentes ao momento da realização deste documento na área de *benchmarking* a plataformas *Blockchain*. Na continuação deste capítulo é apresentada a ferramenta utilizada pelo aluno para a execução dos testes de *benckmark* e as modificações que esta foi alvo, realizadas pelo aluno, necessárias para a adaptação da plataforma existente ao modelo de execução pretendido. O capítulo termina com a ilustração da arquitetura do sistema de testes. Os testes a executar sob as plataformas são adaptados dos testes já existentes para a plataforma apresentada no capítulo anterior. Estes são definidos no Capítulo 6. Os resultados da sua execução e possíveis justificações para os comportamentos observados são apresentados no Capítulo 7. O documento termina com o Capítulo 8, onde são apresentadas as conclusões gerais do trabalho realizado, um breve sumário deste, e a apresentação de trabalho futuro.

# Capítulo 2
# Planeamento e execução da dissertação

Neste capítulo é apresentado o planeamento e execução da dissertação. É dividida em duas secções, correspondentes aos dois semestres em que a dissertação teve lugar. Em cada secção são apresentados dois diagramas de Gantt. Um representa o diagrama do planeamento do semestre, e o outro representa o diagrama executado. Para ambos os diagramas, são apresentadas e detalhadas as tarefas que os constituem, referindo as datas planeadas, datas executadas e as respetivas justificações para os desvios verificados.

## 2.1 Primeiro Semestre

Esta secção compreende a apresentação do planeamento e execução do primeiro semestre, bem como as devidas justificação para os desvios temporais verificados. O primeiro semestre, seguiu um modelo de tempo parcial, correspondendo a 16 horas semanais, tendo início a 10 de setembro de 2018 e fim a 21de janeiro de 2019. No entanto, a data para a entrega do relatório intermédio sofreu uma alteração à sua data inicialmente apresentada, estendendo-se até dia 23 de janeiro de 2019. O aluno não teve influência neste processo, uma vez que a decisão foi tomada pelo departamento. Na Figura 1 e Figura 2 podemos observar os diagramas de *Gantt* correspondentes ao trabalho planeado e trabalho executado para o primeiro semestre, respetivamente. De seguida é feita uma apresentação das tarefas que os compreendem.

Tabela 2 - Tarefa 1 do primeiro semestre

| Tarefa 1 | Estudo da teoria subjacente à Blockchain | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 09-09-2018 | 05-11-2018 |
| Executado | 09-09-2018 | 19-11-2018 |

A primeira tarefa, "Estudo da teoria subjacente à Blockchain", teve como objetivo proporcionar ao aluno a possibilidade de este se familiarizar com os conceitos envolventes da teoria *Blockchain* e com os conceitos necessários para a realização da dissertação. Essa familiarização de conceitos é obtida através da leitura de *papers*, pesquisa de testes existentes na área de *benchmarking* a plataformas *blockchain* e através da pesquisa das próprias plataformas *blockchain*.

Da Figura 2 podemos concluir que a data final para esta tarefa se estendeu para além do previsto. Atraso justificado pela demora na sub-tarefa "Pesquisa de plataformas *blockchain*".

Tabela 3 - Tarefa 2 do primeiro semestre

| Tarefa 2 | Estudo das plataformas a usar | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 06-11-2018 | 04-12-2018 |
| Executado | 19-11-2018 | 24-12-2018 |

A segunda tarefa compreendeu o estudo das plataformas *blockchain* existentes, através da leitura da sua documentação oficial e da identificação das dimensões de comparação. Esta tarefa deu continuação à tarefa 1, funcionando como uma extensão à sub-tarefa "Pesquisa de plataformas".

A segunda tarefa apresenta um grande desvio relativamente à data inicialmente planeada, uma vez que o estudo das plataformas se demonstrou ser um processo demoroso, causado pelo volume da documentação das plataformas em estudo, atrasando assim a sua leitura e interpretação.

Tabela 4 - Tarefa 3 do primeiro semestre

| Tarefa 3 | Setup do ambiente de testes | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 06-11-2018 | 27-11-2018 |
| Executado | 03-12-2018 | 22-01-2019 |

Da terceira tarefa compreenderam-se as atividades de identificação dos requisitos mínimos necessários para a execução de uma instância da plataforma *blockchain*. O desvio na data correspondente à execução justifica-se pelo facto de a documentação oficial não apresentar qualquer referência para os requisitos mínimos necessários. Este apresentou o primeiro problema identificado pelo aluno, sendo que a solução escolhida foi basear-se em trabalhos previamente desenvolvidos para a definição desses mesmos requisitos.

Tabela 5 - Tarefa 4 do primeiro semestre

| Tarefa 4 | Escrita do relatório final | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 06-11-2018 | 27-11-2018 |
| Executado | 03-12-2018 | 22-01-2019 |

Por fim, a quarta tarefa apresenta-se como a tarefa final a ser desempenhada no primeiro semestre. Esta teve como foco principal o desenvolvimento do relatório final. Das sub-tarefas planeadas, a "Escrita do capítulo do estado da arte" demorou muito mais do que o previsto, causada pelo planeamento irrealista do aluno e por este apresentar alguma dificuldade na escrita do mesmo. Na Figura 2 é possível observar que, a sub-tarefa "Escrita do artigo científico", se encontra a vermelho, uma vez que não foi cumprida.

| 1º SEMESTRE - DIAGRAMA PLANEADO | | | | Setembro 2018 | | | | Outubro 2018 | | | | Novembro 2018 | | | | Dezembro 2018 | | | | | Janeiro 2019 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 09-09-2018 | 16-09-2018 | 23-09-2018 | 30-09-2018 | 07-10-2018 | 14-10-2018 | 21-10-2018 | 28-10-2018 | 04-11-2018 | 11-11-2018 | 18-11-2018 | 25-11-2018 | 02-12-2018 | 09-12-2018 | 16-12-2018 | 23-12-2018 | 30-12-2018 | 06-01-2019 | 13-01-2019 | 20-01-2019 | 27-01-2019 | 31-01-2019 |
| Estudo da teoria subjacente à Blockchain | 8 | 09-09-2018 | 05-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Leitura de artigos | 6 | 09-09-2018 | 21-10-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Pesquisa de testes | 1 | 22-10-2018 | 29-10-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Pesquisa de plataformas | 2 | 22-10-2018 | 05-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Estudo das plataformas a usar | 4 | 06-11-2018 | 04-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Estudo da documentação official | 2 | 06-11-2018 | 20-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Identificação das dimensões de comparação | 2 | 20-11-2018 | 04-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Setup do ambiente de testes | 3 | 06-11-2018 | 27-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Identificação dos requisitos | 2 | 06-11-2018 | 20-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Aprovisionamento dos recursos | 1 | 20-11-2018 | 27-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do relatório final | 15 | 19-10-2018 | 31-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Definição da estrutura | 2 | 19-10-2018 | 02-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do capítulo do estado da arte | 1 | 22-10-2018 | 29-10-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do restante relatório | 8 | 03-12-2018 | 28-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do artigo científico | 8 | 03-12-2018 | 28-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Entrega do relatório intermédio | 0 | 31-01-2019 | 31-01-2019 | | | | | | | | | | | | | | | | | | | | | | |

Figura 1 - Diagrama de Gantt planeado para o primeiro semestre

| 1º SEMESTRE - DIAGRAMA EXECUTADO | | | | Setembro 2018 | | | | Outubro 2018 | | | | Novembro 2018 | | | | Dezembro 2018 | | | | | Janeiro 2019 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 09-09-2018 | 16-09-2018 | 23-09-2018 | 30-09-2018 | 07-10-2018 | 14-10-2018 | 21-10-2018 | 28-10-2018 | 04-11-2018 | 11-11-2018 | 18-11-2018 | 25-11-2018 | 02-12-2018 | 09-12-2018 | 16-12-2018 | 23-12-2018 | 30-12-2018 | 06-01-2019 | 13-01-2019 | 20-01-2019 | 23-01-2019 | 27-01-2019 |
| Estudo da teoria subjacente à Blockchain | 10 | 09-09-2018 | 19-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Leitura de artigos | 6 | 09-09-2018 | 21-10-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Pesquisa de testes | 1 | 22-10-2018 | 29-10-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Pesquisa de plataformas | 4 | 22-10-2018 | 19-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Estudo das plataformas a usar | 5 | 19-11-2018 | 24-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Estudo da documentação official | 2 | 19-11-2018 | 03-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Identificação das dimensões de comparação | 3 | 03-12-2018 | 24-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Setup do ambiente de testes | 7 | 03-12-2018 | 22-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Identificação dos requisitos | 3 | 03-12-2018 | 24-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Aprovisionamento dos recursos | 3 | 01-01-2019 | 22-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do relatório final | 14 | 19-10-2018 | 28-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Definição da estrutura | 2 | 19-10-2018 | 02-11-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do capítulo do estado da arte | 6 | 22-10-2018 | 03-12-2018 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do restante relatório | 7 | 03-12-2018 | 21-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Escrita do artigo científico | 8 | 03-12-2018 | 28-01-2019 | | | | | | | | | | | | | | | | | | | | | | |
| Entrega do relatório intermédio | 0 | 23-01-2019 | 23-01-2019 | | | | | | | | | | | | | | | | | | | | | | |

Figura 2 - Diagrama de Gantt executado para o primeiro semestre

## 2.2 Segundo Semestre

Esta secção compreende a apresentação do planeamento e execução do segundo semestre, bem como as devidas justificação para os desvios temporais verificados. O segundo semestre seguiu um modelo de tempo integral, correspondendo a 40 horas semanais, tendo início a 2 de fevereiro de 2019 e fim a 14 de julho de 2019. A apresentação dos diagramas dividiu-se em quatro figuras, nomeadamente a Figura 3 e Figura 4 para os diagramas planeados, e a Figura 5 e Figura 6 para os diagramas executados. A utilização de quatro imagens foi necessária, uma vez que os diagramas eram muito grandes e assim dificultada a sua observação no documento. É, de seguida, feita a apresentação das tarefas que os compreendem.

Tabela 6 - Tarefa 1 do segundo semestre

| Tarefa 1 | Estudo detalhado de trabalhos existentes | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 11-02-2019 | 28-02-2019 |
| Executado | 11-02-2019 | 28-02-2019 |

Na primeira tarefa, "Estudo detalhado de trabalhos existentes", foi dada continuação à - Tarefa 1 do primeiro semestre, dirigindo o foco para os trabalhos existentes na área de benchmarking a plataformas *blockchain*, identificando as diversas dimensões de comparação dessas plataformas. Esta tarefa foi cumprida no tempo previsto.

Tabela 7 - Tarefa 2 do segundo semestre

| Tarefa 2 | Setup do ambiente de testes | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 18-02-2019 | 06-05-2019 |
| Executado | 18-02-2019 | 26-07-2019 |

A segunda tarefa compreendeu a identificação de requisitos mínimos para instalar as plataformas com sucesso, o estudo detalhado e instalação das plataformas *blockchain* e das plataformas de execução dos testes, e por fim a validação de trabalhos já existentes. A tarefa 2 apresenta um grande desvio temporal, uma vez que foi a executar esta tarefa que o aluno se começou a deparar com determinados desafios. O primeiro desafio apareceu quando, após o estudo das plataformas a usar, não foi encontrada nenhuma informação que indicasse quais os requisitos mínimos para a instalação das mesmas. Uma vez que o aluno se encontrava dependente dessa informação para lhe serem fornecidos os recursos computacionais, o atraso na obtenção de tal informação atrasou a instalação das plataformas. A solução encontrada foi a utilização de recursos computacionais idênticos aos utilizados nos trabalhos estudados na tarefa anterior. Por fim, a tarefa da instalação das plataformas

*blockchain* e plataformas de execução dos testes revelou não ser um processo trivial, e como consequência, originou um novo atraso no planeamento temporal.

Tabela 8 - Tarefa 3 do segundo semestre

| Tarefa 3 | Benchmarking a plataformas Blockchain | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 06-05-2019 | 30-06-2019 |
| Executado | 22-04-2019 | 12-08-2019 |

A tarefa 3, "Benchmarking a plataformas Blockchain", apresentou o segundo duas modificações à tarefa planeada. A primeira, foi a substituição da sub-tarefa "Desenvolvimento de workloads" para "Adaptação dos workloads já existentes". A segunda modificação foi a antecipação da sua execução, tendo esta começado logo a seguir ao término da tarefa 2. Contudo, o processo de adaptação dos testes existentes, e posterior execução revelou-se um processo mais demorado que o previsto, fazendo com que a tarefa se desenrolasse até ao mês de agosto. O atraso nesta levou a que fosse tomada a decisão de o aluno entregar a dissertação em Época Especial.

Tabela 9 - Tarefa 4 do segundo semestre

| Tarefa 4 | Escrita do relatório final | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 18-02-2019 | 06-08-2019 |
| Executado | 18-02-2019 | 05-09-2019 |

A tarefa 4, "Escrita do relatório final", foi sendo desenvolvida ao longo do semestre, começando pela revisão do relatório intermédio, escrita do capítulo referente aos trabalhos identificados na tarefa 1, escrita do trabalho desenvolvido pelo aluno, e a escrita dos resultados da execução dos testes de *benchmark*. A tarefa termina com a entrega do relatório, no dia 5 de setembro de 2019.

Tabela 10 - Tarefa 5 do segundo semestre

| Tarefa 5 | Escrita do relatório final | |
|---|---|---|
| | Data de início | Data de fim |
| Planeado | 18-03-2019 | 17-09-2019 |
| Executado | - | - |

A tarefa 5, que constava do planeamento da escrita de dois *papers*, um na forma de *survey* aos conceitos da *blockchain* e outro com a apresentação dos resultados obtidos com a realização desta dissertação, não foi executada.

| 2º SEMESTRE - DIAGRAMA PLANEADO (FEVEREIRO - JUNHO) | | | | Fevereiro 2019 | | | Março 2019 | | | | Abril 2019 | | | | | Maio 2019 | | | | Junho 2019 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 11-02-2019 | 18-02-2019 | 25-02-2019 | 04-03-2019 | 11-03-2019 | 18-03-2019 | 25-03-2019 | 01-04-2019 | 08-04-2019 | 15-04-2019 | 22-04-2019 | 29-04-2019 | 06-05-2019 | 13-05-2019 | 20-05-2019 | 27-05-2019 | 03-06-2019 | 10-06-2019 | 17-06-2019 | 24-06-2019 |
| Estudo detalhado de trabalhos existentes | 3 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação dos trabalhos | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação das dimensões de análise para a comparação das plataformas | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Setup do ambiente de testes | 11 | 18-02-2019 | 06-05-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação dos requisitos minimos para o setup | 1 | 18-02-2019 | 25-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Estudo das plataformas a testar | 2 | 18-02-2019 | 04-03-2019 | | | | | | | | | | | | | | | | | | | | |
| Instalação das plataformas e do sistema de testes | 8 | 01-03-2019 | 26-04-2019 | | | | | | | | | | | | | | | | | | | | |
| Execução e validação de trabalho já desenvolvido | 1 | 29-04-2019 | 06-05-2019 | | | | | | | | | | | | | | | | | | | | |
| Benchmarking das plataformas blockchain | 8 | 06-05-2019 | 30-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento de workloads | 4.5 | 06-05-2019 | 06-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Execução dos workloads e recolha dos dados | 3.5 | 06-06-2019 | 30-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do relatório final | 24 | 18-02-2019 | 06-08-2019 | | | | | | | | | | | | | | | | | | | | |
| Revisão do documento do primeiro semestre | 1.5 | 18-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo do trabalho relacionado | 2.5 | 01-03-2019 | 18-03-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo do trabalho desenvolvido | 2 | 06-06-2019 | 20-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo de resultados e conclusões | 2.5 | 01-07-2019 | 18-07-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do restante relatório | 3 | 16-07-2019 | 06-08-2019 | | | | | | | | | | | | | | | | | | | | |
| Entrega do relatório final | 0 | 05-08-2019 | 05-08-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita de artigos | 26 | 18-03-2019 | 17-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Survey | 9 | 18-03-2019 | 20-05-2019 | | | | | | | | | | | | | | | | | | | | |
| Resultados obtidos | 9 | 16-07-2019 | 17-09-2019 | | | | | | | | | | | | | | | | | | | | |

Figura 3 - Diagrama de Gantt planeado para o segundo semestre – Parte 1

| 2º SEMESTRE - DIAGRAMA PLANEADO (JULHO - SETEMBRO) | | | | Julho 2019 | | | | | Agosto 2019 | | | | Setembro 2019 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 01-07-2019 | 08-07-2019 | 15-07-2019 | 22-07-2019 | 29-07-2019 | 05-08-2019 | 12-08-2019 | 19-08-2019 | 26-08-2019 | 02-09-2019 | 09-09-2019 | 16-09-2019 |
| Estudo detalhado de trabalhos existentes | 3 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Identificação dos trabalhos | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Identificação das dimensões de análise para a comparação das plataformas | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Setup do ambiente de testes | 11 | 18-02-2019 | 06-05-2019 | | | | | | | | | | | | |
| Identificação dos requisitos minimos para o setup | 1 | 18-02-2019 | 25-02-2019 | | | | | | | | | | | | |
| Estudo das plataformas a testar | 2 | 18-02-2019 | 04-03-2019 | | | | | | | | | | | | |
| Instalação das plataformas e do sistema de testes | 8 | 01-03-2019 | 26-04-2019 | | | | | | | | | | | | |
| Execução e validação de trabalho já desenvolvido | 1 | 29-04-2019 | 06-05-2019 | | | | | | | | | | | | |
| Benchmarking das plataformas blockchain | 8 | 06-05-2019 | 30-06-2019 | | | | | | | | | | | | |
| Desenvolvimento de workloads | 4.5 | 06-05-2019 | 06-06-2019 | | | | | | | | | | | | |
| Execução dos workloads e recolha dos dados | 3.5 | 06-06-2019 | 30-06-2019 | | | | | | | | | | | | |
| Escrita do relatório final | 24 | 18-02-2019 | 06-08-2019 | | | | | | | | | | | | |
| Revisão do documento do primeiro semestre | 1.5 | 18-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Escrita do capitulo do trabalho relacionado | 2.5 | 01-03-2019 | 18-03-2019 | | | | | | | | | | | | |
| Escrita do capitulo do trabalho desenvolvido | 2 | 06-06-2019 | 20-06-2019 | | | | | | | | | | | | |
| Escrita do capitulo de resultados e conclusões | 2.5 | 01-07-2019 | 18-07-2019 | | | | | | | | | | | | |
| Escrita do restante relatório | 3 | 16-07-2019 | 06-08-2019 | | | | | | | | | | | | |
| Entrega do relatório final | 0 | 05-08-2019 | 05-08-2019 | | | | | | | | | | | | |
| Escrita de artigos | 26 | 18-03-2019 | 17-09-2019 | | | | | | | | | | | | |
| Survey | 9 | 18-03-2019 | 20-05-2019 | | | | | | | | | | | | |
| Resultados obtidos | 9 | 16-07-2019 | 17-09-2019 | | | | | | | | | | | | |

Figura 4 - Diagrama de Gantt planeado para o segundo semestre – Parte 2

| 2º SEMESTRE - DIAGRAMA EXECUTADO (FEVEREIRO - JUNHO) | | | | Fevereiro 2019 | | | Março 2019 | | | | Abril 2019 | | | | | Maio 2019 | | | | Junho 2019 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 11-02-2019 | 18-02-2019 | 25-02-2019 | 04-03-2019 | 11-03-2019 | 18-03-2019 | 25-03-2019 | 01-04-2019 | 08-04-2019 | 15-04-2019 | 22-04-2019 | 29-04-2019 | 06-05-2019 | 13-05-2019 | 20-05-2019 | 27-05-2019 | 03-06-2019 | 10-06-2019 | 17-06-2019 | 24-06-2019 |
| Estudo detalhado de trabalhos existentes | 3 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação dos trabalhos | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação das dimensões de análise para a comparação das plataformas | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Setup do ambiente de testes | 23 | 18-02-2019 | 26-07-2019 | | | | | | | | | | | | | | | | | | | | |
| Identificação dos requisitos minimos para o setup | 5 | 18-02-2019 | 25-03-2019 | | | | | | | | | | | | | | | | | | | | |
| Estudo das plataformas a testar | 8 | 18-02-2019 | 15-04-2019 | | | | | | | | | | | | | | | | | | | | |
| Instalação das plataformas e do sistema de testes | 8 | 01-03-2019 | 26-04-2019 | | | | | | | | | | | | | | | | | | | | |
| Execução e validação de trabalho já desenvolvido | 11 | 10-05-2019 | 26-07-2019 | | | | | | | | | | | | | | | | | | | | |
| Benchmarking das plataformas blockchain | 16 | 22-04-2019 | 12-08-2019 | | | | | | | | | | | | | | | | | | | | |
| Adaptação dos workloads já existentes | 16 | 22-04-2019 | 12-08-2019 | | | | | | | | | | | | | | | | | | | | |
| Execução dos workloads e recolha dos dados | 3.5 | 06-06-2019 | 30-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do relatório final | 28 | 18-02-2019 | 05-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Revisão do documento do primeiro semestre | 1.5 | 18-02-2019 | 28-02-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo do trabalho relacionado | 4 | 01-03-2019 | 29-03-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo do trabalho desenvolvido | 2 | 06-06-2019 | 20-06-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do capitulo de resultados e conclusões | 2 | 20-08-2019 | 03-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita do restante relatório | 3 | 13-08-2019 | 03-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Entrega do relatório final | 0 | 05-09-2019 | 05-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Escrita de artigos | 26 | 18-03-2019 | 17-09-2019 | | | | | | | | | | | | | | | | | | | | |
| Survey | 9 | 18-03-2019 | 20-05-2019 | | | | | | | | | | | | | | | | | | | | |
| Resultados obtidos | 9 | 16-07-2019 | 17-09-2019 | | | | | | | | | | | | | | | | | | | | |

Figura 5 - Diagrama de Gantt executado para o segundo semestre – Parte 1

| 2º SEMESTRE - DIAGRAMA PLANEADO (JULHO - SETEMBRO) | | | | Julho 2019 | | | | | Agosto 2019 | | | | Setembro 2019 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TAREFA | DURAÇÃO | START | END | 01-07-2019 | 08-07-2019 | 15-07-2019 | 22-07-2019 | 29-07-2019 | 05-08-2019 | 12-08-2019 | 19-08-2019 | 26-08-2019 | 02-09-2019 | 09-09-2019 | 16-09-2019 |
| Estudo detalhado de trabalhos existentes | 3 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Identificação dos trabalhos | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Identificação das dimensões de análise para a comparação das plataformas | 2.5 | 11-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Setup do ambiente de testes | 23 | 18-02-2019 | 26-07-2019 | | | | | | | | | | | | |
| Identificação dos requisitos minimos para o setup | 5 | 18-02-2019 | 25-03-2019 | | | | | | | | | | | | |
| Estudo das plataformas a testar | 8 | 18-02-2019 | 15-04-2019 | | | | | | | | | | | | |
| Instalação das plataformas e do sistema de testes | 8 | 01-03-2019 | 26-04-2019 | | | | | | | | | | | | |
| Execução e validação de trabalho já desenvolvido | 11 | 10-05-2019 | 26-07-2019 | | | | | | | | | | | | |
| Benchmarking das plataformas blockchain | 16 | 06-05-2019 | 26-08-2019 | | | | | | | | | | | | |
| Adaptação dos workloads já existentes | 16 | 06-05-2019 | 26-08-2019 | | | | | | | | | | | | |
| Execução dos workloads e recolha dos dados | 3.5 | 06-06-2019 | 30-06-2019 | | | | | | | | | | | | |
| Escrita do relatório final | 28 | 18-02-2019 | 05-09-2019 | | | | | | | | | | | | |
| Revisão do documento do primeiro semestre | 1.5 | 18-02-2019 | 28-02-2019 | | | | | | | | | | | | |
| Escrita do capitulo do trabalho relacionado | 4 | 01-03-2019 | 29-03-2019 | | | | | | | | | | | | |
| Escrita do capitulo do trabalho desenvolvido | 2 | 06-06-2019 | 20-06-2019 | | | | | | | | | | | | |
| Escrita do capitulo de resultados e conclusões | 2 | 20-08-2019 | 03-09-2019 | | | | | | | | | | | | |
| Escrita do restante relatório | 3 | 13-08-2019 | 03-09-2019 | | | | | | | | | | | | |
| Entrega do relatório final | 0 | 05-09-2019 | 05-09-2019 | | | | | | | | | | | | |
| Escrita de artigos | 26 | 18-03-2019 | 17-09-2019 | | | | | | | | | | | | |
| Survey | 9 | 18-03-2019 | 20-05-2019 | | | | | | | | | | | | |
| Resultados obtidos | 9 | 16-07-2019 | 17-09-2019 | | | | | | | | | | | | |

Figura 6 - Diagrama de Gantt executado para o segundo semestre – Parte 2

# Capítulo 3
# Tecnologia Blockchain

O presente capítulo serve como uma introdução à tecnologia *Blockchain*. Este começa na secção 3.1, pela apresentação dos seus conceitos base. O capítulo continua na secção 3.2, onde é feita a distinção entre *Blockchains*, e apresentada a definição para *Blockchain* com permissões e *Blockchain* sem permissões. A secção termina com uma comparação direta entre estas. Na secção 3.3, é abordado o tema dos mecanismos de consenso, seguido pelas apresentações de diversas implementações destes. De igual forma à secção anterior, esta termina com uma comparação direta entre os diversos mecanismos de consenso apresentados. De seguida, a secção 3.4, introduz o conceito de *smart contract*, complementando-o com um exemplo de uma possível aplicação deste. O capítulo termina com a secção 3.5, onde são discutidos os possíveis riscos e limitações da tecnologia *Blockchain*.

## 3.1 Conceitos base

O registo de transações não é algo recente, sejam elas transações de bens ou até mesmo transações financeiras. Tal é feito com recurso a um *ledger* (livro-razão), um livro no qual são realizados registos, especialmente, atividades de negócio e dinheiro pago ou recebido (Cambridge University Press, 2019). O *ledger* mantém um estado e um conjunto ordenado de transações que o determinam (The Linux Foundation, 2018a). Este, no entanto, foi sempre mantido por organizações centrais, como é o exemplo de sistemas bancários (Gupta, 2017), empresas, ou o governo (Yaga, Mell, Roby, & Scarfone, 2018).

Apresentando-se como a base da *Bitcoin*, a tecnologia *Blockchain* foi proposta em 2008 por uma pessoa ou grupo de pessoas sob o pseudónimo de Satoshi Nakamoto, como solução para um sistema de pagamentos eletrónicos livre de entidades provedoras de confiança (Nakamoto, 2008). Com mecanismos dedicados de verificação e validação de cada transação, este sistema surge também como solução para o conhecido *double spending problem* – problema comum a todas as cripto moedas, que consiste na possibilidade de gastar o mesmo *token* virtual mais que uma vez, por parte do mesmo utilizador (Sudhir Khatwani, 2018).

A introdução da tecnologia *Blockchain* permite que apareçam novos conceitos de *ledger*, como é o caso do *distributed ledger* (livro-razão distribuído). Para este são encontradas várias definições. Os autores Brakeville & Perepa (2018), definem *distributed ledger* como:

> *"a type of database that is shared, replicated, and synchronized among the members of a decentralized network."*

Outra definição é proposta por Rutland (2018), que define o *distributed ledger* como:

> *"a record of consensus with a cryptographic audit trail which is maintained and validated by several separate nodes*."*

Comparando as definições anteriormente apresentadas, notamos que Rutland (2018) introduz o conceito de registos de consensos. No entanto, seguindo a estrutura do documento em causa, este tema só é abordado na secção 3.3.

Com o crescente interesse na tecnologia, em muito devido às suas características únicas, diversos autores sugerem aplicações da mesma em diversos setores, como é o exemplo do controlo do comércio de diamantes de sangue (Minkenberg, 2015), melhoramento dos processos nas cadeias de valor (Marr, 2018) e na logística marítima (Gronholt-Pedersen, 2018), e revolucionar a industria musical (Heap, 2017).

É importante perceber contudo que, a *blockchain* não é um *distributed ledger* mas sim um caso particular do mesmo (Rutland, 2018). Existindo múltiplas definições para a tecnologia *Blockchain,* são de seguida apresentadas duas definições, sendo que a sua escolha foi feita com base na reputação dos seus autores.

De acordo com o *National Institute of Standards and Technology* (NIST) (Yaga et al., 2018), *blockchains* são:

> *"tamper evident and tamper resistant digital ledgers implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority (i.e., a bank, company or government). At their basic level, they enable a community of users to record transactions in a shared ledger within that community, such that under normal operation of the blockchain network no transaction can be changed once published".*

Já a The Linux Foundation (2018a), define a *blockchain* como:

> *"an immutable transaction ledger, maintained within a distributed network of peer nodes. These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that bind each block to the preceding block."*

A segunda definição, pela *Linux Foundation*, complementa a apresentada pelo *NIST*, introduzindo detalhes sobre o processo de interligação dos blocos da *blockchain*, consistindo em integrar no bloco atual, uma *string hash* do bloco anterior. A *string* é obtida após a aplicação de uma função de *hashing* – função criptográfica que garante a integridade de determinada informação, garantindo dessa forma que a mesma não sofreu alterações, seja de forma propositada ou até mesmo de forma acidental (Granjal, 2017). A tecnologia *Blockchain* usa a função *Secure Hash Algorithm* (*SHA*), com um output de 256 *bits* (*SHA-256*) (Yaga et al., 2018). A partir deste momento e no decorrer do documento, como forma de simplificar a leitura, a *string hash* passa a ser descrita apenas como *hash*.

De seguida, são apresentados os componentes constituintes de uma *blockchain*, no entanto, a constituição de cada *blockchain*, bem como dos seus blocos, varia de acordo com a implementação em causa. Assim, os componentes apresentados referem-se a uma implementação genérica de uma *blockchain*.

Na Figura 7 é apresentada uma *blockchain*, B, compreendida pelos blocos B0, B1, B2 e B3. O bloco B0 apresenta-se na sua definição como o *genesis block* – bloco inicial de uma *blockchain*, sem transações, sem bloco precedente e com o *hash* do bloco anterior preenchido a zeros (Yaga et al., 2018). A *blockchain* interliga os seus blocos armazenando no seu *header* o *hash* das suas transações e uma cópia do *hash* do bloco anterior, capacitando-a com a característica de virtual imutabilidade, uma vez que a menor alteração feita num bloco é facilmente detetável, através de uma comparação do *hash* do bloco original e do *hash* do bloco alterado. Uma tentativa de falsificação dos conteúdos de um bloco implicaria que o atacante tivesse de

refazer todo o trabalho computacional necessário para o cálculo das funções de *hash* do bloco a ser alterado, e de todos os blocos subsequentes (Yaga et al., 2018).

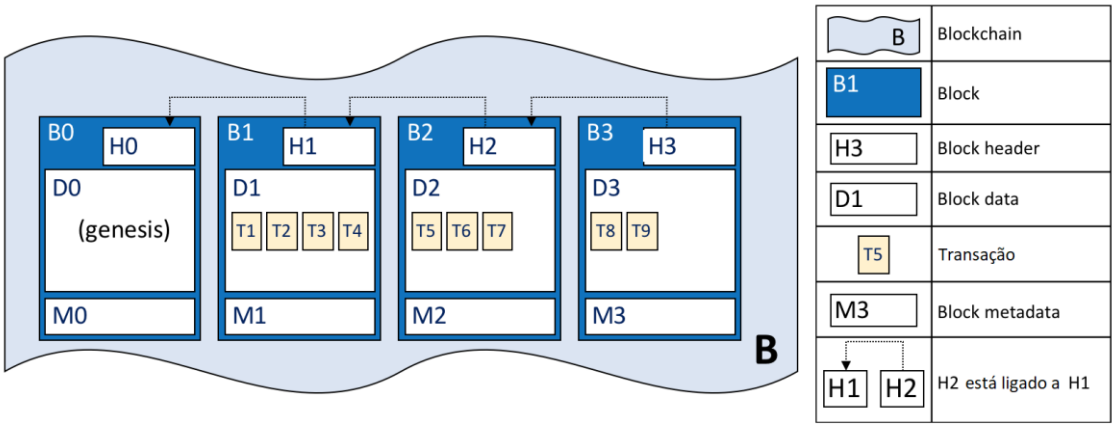


Figura 7 - Blockchain, adaptada de (The Linux Foundation, 2018b)

As transações inseridas na *blockchain* são agrupadas em blocos. Um bloco é divido em três secções: *block header*, *block data* e *block metada*. Da Figura 7, observamos que o bloco B1, é constituído por um *header* H1, um *block data* D1, compreendido pelas transações T1, T2, T3 e T4, e um *block metada* M1. Na Figura 8 é apresentado em detalhe a composição de um bloco.

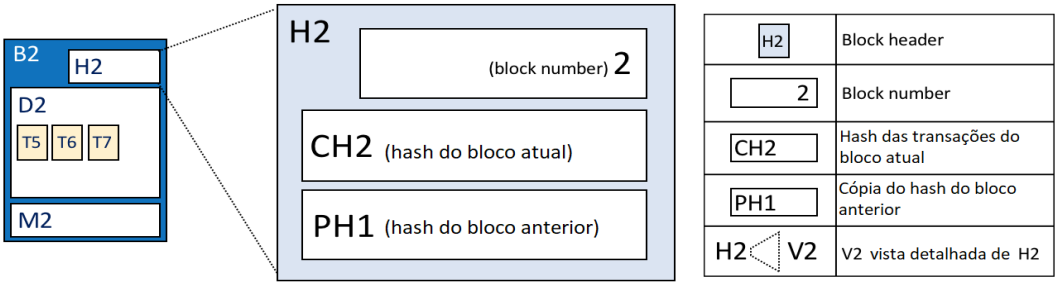


Figura 8 - Bloco de uma Blockchain, adaptada de (The Linux Foundation, 2019g)

Da Figura 8 é possível concluir que do *header* H2 do bloco apresentado, fazem parte:

- *Block number* – Normalmente definido por um inteiro, começando no 0, sendo incrementado a cada adição de um novo bloco à blockchain.
- *Hash* do bloco atual, CH2 – Resultado *hash* das transações desse bloco, conseguido através de uma árvore de *Merkle* (estrutura de dados em árvore onde cada nó não folha é etiquetado com o *hash* do seu filho (Curran, 2018)).
- *Hash* do bloco anterior, PH1.

Do *block data* fazem parte a lista de transações, nomeadamente T5, T6 e T7, inseridas de forma sequencial após aprovação e validação dos nós. Por fim, o *block metadata* é constituído com informação adicional que se queira adicionar ao bloco, como por exemplo uma mensagem associada às transações que o constituem (The Linux Foundation, 2018a; Walker, 2018; Yaga et al., 2018)

## 3.2 Categorização de Blockchains

As *Blockchains* podem ser categorizadas relativamente ao seu modelo de permissões. De acordo com esse modelo, é possível determinar quem está autorizado a participar, manter uma cópia do *ledger* atualizado na sua posse, e a executar os mecanismos de consenso implementados por essa *Blockchain* (Jayachandran, 2017). A presente secção apresenta as definições para dois tipos de modelos de *blockchain*, *blockchains* publicas e *blockchains* privadas. A secção termina com a apresentação de uma comparação das duas sob a forma de uma tabela.

Considera-se uma *blockchain* pública, uma *blockchain* aberta à participação de qualquer utilizador, permitindo que este mantenha uma cópia do *ledger* atualizado na sua posse, consiga enviar transações para outros utilizadores, e consiga executar os seus mecanismos de consenso, desde que este possua o equipamento necessário para tal (Yaga et al., 2018). Os mecanismos de consenso são mecanismos que permitem decidir qual o próximo nó a ser inserido na *blockchain* (Zheng, Xie, Dai, Chen, & Wang, 2017). Exemplos detalhados de mecanismos de consenso são apresentados na secção 3.3. Não tendo nenhum mecanismo para a associação de identidades aos endereços públicos dos seus utilizadores, estes são considerados como anónimos na rede. Tal característica pode originar comportamentos de desconfiança entre os utilizadores. Nos casos em que o domínio da *blockchain* é a cripto-moeda, há tentativas de mitigação de tal comportamento com recurso a processos de cripto-economia – processo que combina incentivos económicos e mecanismos de consenso de trabalho elevado trabalho computacional. Para este tipo de mecanismos de consenso, um utilizador é recompensado com *tokens* do sistema, sempre que cria e adiciona um novo bloco à *blockchain* (Buterin, 2015).

As *blockchains* privadas, apresentam um modelo onde a participação nesta já não é aberta a qualquer utilizador, precisando-se de um convite por parte da entidade responsável para a participação nesta. Este modelo contrasta com o modelo de *blockchain* públicas apresentado anteriormente (Jayachandran, 2017). Um utilizador, ao necessitar de convite, garante que este seja conhecido por parte da entidade responsável, dessa forma, permite que os restantes participantes da rede sejam conhecidos entre si. Esta característica proporciona um maior controlo sobre a *blockchain*, uma vez que sobre um utilizador passa a ser possível a aplicação de restrições de operações de leitura e escrita na *blockchain* (Yaga et al., 2018). Associado à possibilidade de os utilizadores serem conhecidos entre si, este modelo de *blockchain* dispensa os processos derivados de cripto-economia (observados no modelo de *blockchain* pública), que, por sua vez, dispensam os mecanismos de consenso de elevado trabalho computacional. Por sua vez, o risco de inserção de informação maligna por parte dos utilizadores na *blockchain*, é reduzido, uma vez que a sua assinatura digital acompanha todas as operações realizadas na rede, tornando mais fácil a possibilidade de esse utilizador ser rastreado e expulso da mesma (The Linux Foundation, 2018a).

Conhecidos os dois modelos de *blockchain*, é relevante perceber as suas diferenças. De forma a realizar uma comparação fundamentada, foram tidos em conta os parâmetros de

comparação propostos por Jaeger (2018) e Zheng (2017). São eles, a identidade dos nós participantes, o seu modelo de restrições, o tipo de mecanismo de consenso implementado e a sua performance no geral. A comparação é feita sob a forma da Tabela 11.

Tabela 11 - Comparação entre os vários tipos de Blockchain, adaptada de (Jaeger, 2018; Zheng et al., 2017)

| Parâmetros de comparação | Blockchain pública | Blockchain privada |
|---|---|---|
| Identidade dos nós | Não existe uma associação entre a identidade virtual e a identidade real de um utilizador. Na rede, os nós são anónimos entre si. | Existe uma associação entre uma identidade virtual e uma identidade real. Essa identidade é conhecida pelos restantes nós. |
| Restrições a operações de leitura e escrita | Não | Sim |
| Mecanismos de consenso implementados | Mecanismos de consenso que requerem um trabalho computacional elevado. | Mecanismos de consenso que não requerem um trabalho computacional elevado. |
| Performance | Ao utilizarem mecanismos de consenso de trabalho computacional elevado, considera-se que no geral a sua performance é baixa. | Apresenta uma performance melhor quando comparada com a blockchain pública. |

Da tabela acima apresentada são retiradas as seguintes conclusões:

- Em ambos os modelos de *blockchains*, usam uma identidade virtual. No entanto, apenas na *blockchain* privada existe uma ligação dessa identidade a uma identidade do mundo real. No caso da *blockchain* pública, os nós são conhecidos apenas pelo seu endereço virtual, fazendo com que estes sejam anónimos entre si;
- Na *blockchain* pública, qualquer utilizador pode participar na blockchain. De igual forma, não são aplicados quaisquer tipos de restrições aos utilizadores para operações de leitura e escrita na blockchain. O mesmo não se verifica nas *blockchains* privadas, onde para participar, um utilizador necessita de um convite. Neste modelo podem ser aplicadas restrições a operações de leitura e escrita a um utilizador;
- Derivado do facto dos seus utilizadores serem conhecidos entre si, as *blockchains* privadas implementam mecanismos de consenso que não requerem trabalho computacional elevado, contrastando com o modelo de *blockchains* públicas;
- Como consequência do ponto anteriormente referido, é expectável que a *blockchain* privada tenha um melhor desempenho quando comparada com a *blockchain* pública;

## 3.3 Mecanismos de consenso

Na *Blockchain*, a tomada de decisão sobre qual o próximo nó a publicar um bloco é conseguida com recurso a mecanismos de consenso (Yaga et al., 2018). A definição original para este problema, inicialmente apresentado pelas *blockchains* públicas, surge como uma adaptação ao problema dos Generais Bizantinos – problema em que *n* generais tentam concordar mutuamente sobre um plano de batalha através de mensageiros, todavia, *f* generais são traidores tentando boicotar o ataque. Este problema é comparável ao que temos numa *blockchain* (Tschorsch & Scheuermann, 2016), onde se tenta atingir um consenso num sistema distribuído, ausente de confiança entre os seus participantes. Este processo pode tornar-se complicado uma vez que não existe uma entidade responsável pela tomada de decisões (Zheng et al., 2017).

Com o principal objetivo de não tornar o documento e o capítulo em causa muito extenso, vão ser detalhados apenas três modelos de implementações de mecanismos de consenso, tais como o *Proof-of-Work*, *Proof-of-Stake* e *Raft*. A escolha dos mesmos baseou-se na sua importância, sendo que os dois primeiros são implementados em *blockchains* públicas e o último em *blockchains* privadas. No final da secção é apresentada uma tabela de comparação entre os vários mecanismos de consenso. Nessa tabela são apresentados mecanismos de consenso que não foram detalhados anteriormente, no entanto, é deixada a referência para o leitor caso este pretenda conhecer os seus detalhes.

### 3.3.1. Proof-of-Work

O primeiro modelo que é apresentado é o modelo de *Proof-of-Work* (PoW), e é conhecido como o mecanismo de consenso usado na *blockchain Bitcoin.* Neste modelo, um novo bloco é adicionado por um utilizador caso seja o primeiro a resolver um *puzzle* criptográfico computacionalmente intensivo. Quando um nó cria um novo bloco, é chamado de nó *miner*.

O *puzzle*, que de entre as suas imensas características se destacam a dificuldade de resolução e trivialidade na verificação da solução, consiste em encontrar um *hash* para o *header* do bloco, que seja menor que um valor alvo – valor que corresponde ao número de zeros que precede esse *hash*. Para o cálculo desse *hash* é adicionado ao *header* um *nounce* – número único, incrementado até se encontrar a solução pretendida (Zheng et al., 2017). A solução desse puzzle, serve como prova de que foram investidos tempo e recursos computacionais numa tentativa de resolução. O puzzle é, contudo, probabilístico, uma vez que há uma hipótese de serem gerados mais que um bloco ao mesmo tempo. Esses eventos são designados como *forks* (Yaga et al., 2018). A *blockchain* lida com os *forks* considerando um bloco definitivamente confirmado após este ser seguido por um número arbitrário de blocos (Dinh et al., 2017).

Seguindo o paradigma inicial do *PoW* proposto por Nakamoto (2008), qualquer utilizador com um simples CPU poderia efetuar operações de mineração – processo de executar o mecanismo de consenso *PoW* (Zheng et al., 2017). Com o aumento dos utilizadores da rede, aumenta a dificuldade dos *puzzles*, impossibilitando a resolução destes com recurso a um simples CPU, e, inevitavelmente começam a surgir soluções à base de hardware especifico para executar este tipo de operações, denominados de *ASICs – application-specific integrated circuits* (Tschorsch & Scheuermann, 2017). De igual forma, surgem também *pools* ou comunidades de *miners*, onde o trabalho computacional é distribuído de forma a aumentar a eficácia na resolução dos puzzles, repartindo a recompensa de acordo com o poder computacional de cada nó nessa *pool* (Yaga et al., 2018).

Surgem, no entanto, algumas preocupações relativas ao uso deste mecanismo de consenso. A primeira sendo o gasto excessivo de recursos computacionais e de energia ao manter um nó *online* e a publicar blocos. A segunda sendo a criação de *pools* de mineração, uma vez que ao serem criadas estas *pools*, a rede *blockchain* passa a ser centralizada, indo contra o paradigma inicial de descentralização da rede. Esta medida pode ainda levar a que seja possível a realização do ataque dos 51% – onde um utilizador passa a ter na sua posse 51% do total dos recursos computacionais da rede, e dessa forma tomar o controlo da *blockchain*. (Zheng et al., 2017). Por fim, a última preocupação surge do mau sistema de recompensas, pois um utilizador que tenha maior poder computacional, tem maior probabilidade de resolver o puzzle primeiro que outro com menos poder computacional, ficando com a recompensa da resolução. Isto cria um sistema desequilibrado, que não dá hipóteses a que utilizadores mais pequenos possam crescer na rede (Tschorsch & Scheuermann, 2017).

### 3.3.2. Proof-of-Stake

O modelo de *Proof-of-Stake* (*PoS*) surge de uma tentativa de resolução dos problemas encontrados pelo modelo de *PoW*. Este, no entanto, difere no método de escolha para o próximo nó *miner*. Enquanto que, no modelo de *PoW*, a decisão era suportada pelo investimento de recursos computacionais, no modelo de *PoS*, essa decisão é determinada pelo *stake* de cada utilizador. O *stake*, ou quota, traduz o investimento feito pelo o utilizador em *tokens* do sistema, sendo que, após investido, esse *stake* não pode utilizado pelo o utilizador que o detenha (Yaga et al., 2018). Há, no entanto, diversas abordagens à forma como o *stake* é tido em conta na rede *blockchain*. Essa abordagem está dependente da implementação do modelo *PoS* em causa. No documento publicado pelo *NIST* (Yaga et al., 2018), são apresentadas algumas dessas abordagens, detalhadas de seguida.

- ***Random choice***

Num sistema que use a abordagem de *random choice*, também referenciado como *chain-based proof-of-stake*, a *blockchain* percorre todos os utilizadores participantes na rede, escolhendo um para a publicação do próximo bloco com base no seu *stake*. Assim, se um utilizador tiver na sua posse 42% do *stake* total da rede, este tem uma probabilidade de 42% de ser selecionado para minerar o próximo bloco (Christidis & Devetsikiotis, 2016). Após a apresentação desta definição, reparamos que o seu nome não corresponde ao que realmente faz, uma vez que a escolha do utilizador não é aleatória.

- ***Multi-round voting***

Numa abordagem de *multi-round voting*, a *blockchain* seleciona vários utilizadores. Estes vão gerar propostas de novos blocos para serem adicionados à *blockchain*. Os blocos acabados de gerar são, posteriormente, alvo de uma votação por parte dos restantes utilizadores da *blockchain*. Dessa votação resulta a decisão sobre se esse bloco deve ou não ser adicionado à *blockchain*, podendo haver várias rondas até que se atinja essa decisão. Esta abordagem tem uma componente democrática associada, uma vez que todos os utilizadores podem expressar a sua vontade em adicionar ou rejeitar um novo bloco à *blockchain*.

- ***Coin aging***

A abordagem de *coin aging* é possivelmente a implementação mais conhecida do modelo de *PoS*. Nesta, o *token* utilizado no *stake* faz uso de uma propriedade de *aging*, que é usada como um contador, sendo incrementada durante o tempo em que o *stake* não é usado. Assim que um utilizador publica um novo bloco, a propriedade de *aging* dos *tokens* sofre um *reset.* Esse *reset* impede que um utilizador volte a publicar blocos enquanto a propriedade de *aging* atingir o requisito de tempo necessário para tal. Desta forma, é garantido que utilizadores com *tokens* mais antigos têm uma maior probabilidade de serem selecionados para publicar blocos, mitigando a possibilidade de o sistema ser monipolizado por um único utilizador (Tschorsch & Scheuermann, 2017).

- ***Delegate sytems***

A última abordagem apresentada pelos autores tem o nome de *delegate systems.* Nesta, os utilizadores da rede votam no utilizador que pretendem que seja o próximo a publicar um bloco. A votação é um processo constante, podendo ser usada nas duas vertentes disponíveis, votar a favor de um utilizador, e votar contra esse utilizador. Ganha o direito a publicar um bloco, o utilizador que tenha mais votos quando terminar a votação. O peso do voto de cada utilizador está diretamente associado ao seu *stake*, ou seja, um utilizador com maior *stake* tem um voto com maior peso. Esta abordagem permite melhorias na segurança da rede, uma vez que associada a cada utilizador está a sua reputação de *miner.* Ao praticar operações maliciosas na rede, um utilizador perde essa reputação, e com ela as recompensas obtidas pela mineração de novos blocos.

Com o detalhe deste mecanismo de consenso conseguimos observar as melhorias quando comparado ao modelo anterior, *PoW*. Ao não necessitar de operações com trabalho computacional exigente, os custos de manutenção de uma *blockchain* que implemente este modelo são reduzidos em relação ao anterior

### 3.3.3. Raft

O último modelo a ser detalhado neste documento é o mecanismo de consenso *Raft*. A este mecanismo é dada especial atenção, uma vez que é o mecanismo de consenso implementado nas plataformas *Blockchain* que vão ser alvo dos testes de *benchmark*, detalhados no Capítulo 6, sendo que o texto seguinte se foca principalmente na sua implementação e funcionamento.

O *Raft* surge de uma adaptação do mecanismo de consenso *Paxos* – modelo que dominou o tema de mecanismos de consenso na última década (Ongaro & Ousterhout, 2014). No entanto, o modelo *Paxos* apresenta algumas desvantagens, como a sua difícil compreensão e implementação. O mecanismo de consenso *Raft* pretende endereçar e resolver essas desvantagens, mantendo as características de boa performance e tolerância a falhas, oferecidas pelo *Paxos* (Huang, Ma, & Zhang, 2018). De entre as diversas modificações que o *Raft* sofreu ao longo do seu desenvolvimento, destacam-se a subdivisão em problemas mais pequenos e independentes, como por exemplo a eleição de um líder, a replicação de *logs* e segurança do mecanismo, e a redução do grau de não determinismo dos servidores dedicados à execução do *Raft* (Ongaro & Ousterhout, 2014). Nos próximos parágrafos é detalhado o funcionamento deste, adaptado de (Ongaro & Ousterhout, 2014).

O cliente faz um novo pedido a um servidor. O servidor é compreendido por uma *state machine*, um *log* e um *consensus module.* Um exemplo dessa arquitetura é visível na Figura 9. Após a receção do pedido, é da responsabilidade do *consensus module*, inserir a ordem recebida no seu *log*, e paralelamente, comunicar com os *consensus modules* dos restantes servidores, garantindo a sincronia entre estes. Resolvidos quaisquer conflitos que possam haver, os comandos guardados no *log* são enviados para a *state machine*, para a respetiva execução. O resultado desta execução é depois retornado para o cliente.

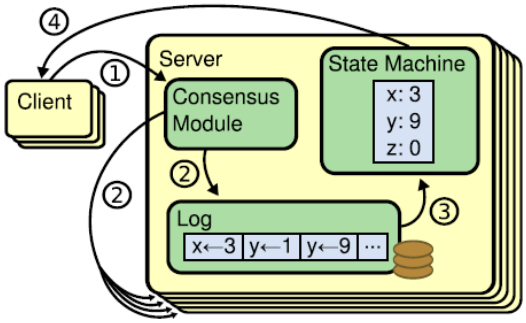


Figura 9 - Arquitetura de um servidor, adaptada de (Ongaro & Ousterhout, 2014)

A uma dada altura na execução, cada servidor pode tomar um de três estados, sendo eles líder, candidato e seguidor. Em condições normais de execução, o algoritmo suporta apenas um líder, com os restantes servidores no estado seguidor. Todos os servidores começam no estado seguidor. Caso não recebam resposta de um líder, assumem o papel de candidato e iniciam uma nova eleição para se tornarem líderes. Se um candidato vencer a eleição torna-se líder, operando até se desconectar ou sofrer alguma falha, com os restantes candidatos a assumirem o papel de seguidores. Uma ilustração deste fluxo pode ser observada na Figura 10.

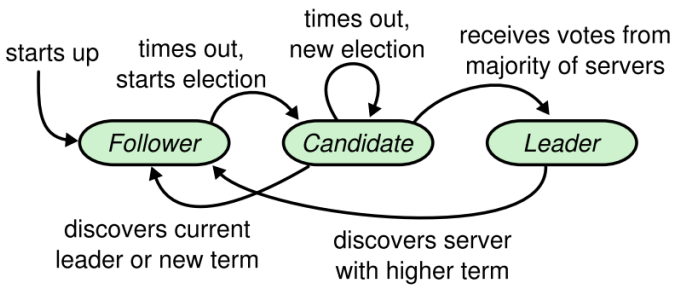


Figura 10 - Estados de um servidor, adaptada de (Ongaro & Ousterhout, 2014)

O tempo de execução é divido em *terms* de duração finita. Um *term* é iniciado com uma eleição, onde o líder é responsável por operar durante o período do mesmo. Se um *term* terminar sem que seja eleito um líder, um novo *term* é iniciado com uma nova eleição.

A comunicação entre servidores é feita com recurso a *remote procedure calls* (*RPCs*), sendo que estes necessitam apenas de duas chamadas durante o seu processo de execução. A primeira, *RequestVote RPC*, é executada apenas por servidores no estado candidato, sendo que a sua função é requerer votos dos outros servidores durante uma eleição. A segunda chamada, *AppendEntries*, é executada apenas por servidores líderes, e tem como função a replicação

dos comandos enviados para os *logs*, e funciona como um *heartbeat* – mecanismo utilizado pelos líderes com o intuito de prevenir novas eleições.

Uma eleição é iniciada caso um servidor no estado seguidor deixe de receber comunicações do seu líder (*heartbeats*), durante o período de *election timeout* – período de tempo atribuído a um servidor no estado seguidor, necessário para este se tornar candidato, assumindo que não há nenhum líder viável nesse momento. Ao iniciar uma eleição, um servidor incrementa o seu *term* e atualiza o seu estado para candidato. De seguida, o servidor candidato vota nele mesmo, e envia *RequestVote RPCs* para os restantes servidores. Deste processo podem resultar três possíveis resultados. No primeiro resultado, uma eleição termina com sucesso, e o servidor é eleito como líder. No segundo caso, um servidor candidato em espera pela receção dos votos pode receber uma *AppendEntries RPC*, significando que nesse intervalo de tempo, outro servidor foi eleito como líder. Há, no entanto, uma exceção onde o servidor candidato pode rejeitar o novo líder, caso o *term* do novo líder seja inferior ao seu. Por fim, no último caso, uma eleição pode terminar sem que seja eleito um líder. Tal evento é raro, uma vez que a implementação dos tempos de *election timeout* é feita de forma aleatória. No entanto, caso este aconteça, todos os servidores dão *timeout* e é iniciada uma nova eleição de seguida. O processo de replicação de *logs* pelos outros servidores é iniciado após terminar uma eleição.

Um pedido de um cliente é compreendido por comandos que vão ser executados pela *state machine* do servidor. Para prevenir inconsistências nos *logs*, à chamada *AppendEntries RPC*, o líder adiciona os campos correspondentes ao índice da chamada, e do *term* da chamada que a precede. Tal ação permite que um servidor recuse uma entrada, caso a entrada precedente a esta não corresponda a nenhuma das entradas constituintes do seu *log*. Caso se verifiquem à mesma tais inconsistências nos *logs*, o líder deve encontrar a última entrada em que todos os servidores estão sincronizados, apagar as entradas subsequentes a essa ultima entrada, e reenviar novamente as entradas que geraram as inconsistências, permitindo desta forma que os servidores se mantenham sincronizados até ao final do *term*.

### 3.3.4. Visão geral dos mecanismos de consenso

Ao longo das sub-secções anteriores foram apresentados três dos mecanismos de consenso mais conhecidos. O *Proof-of-Work*, que se apresenta como sendo o mecanismo de consenso implementado na *Blockchain Bitcoin*, o *Proof-of-Stake*, mecanismo de consenso implementado na *Blockchain Ethereum* e o mecanismo de consenso *Raft*, desenhado principalmente para *Blockchains* privadas. No decorrer do texto que se segue é apresentado um resumo dos mesmos, com a adição de alguns mecanismos que não foram apresentados anteriormente. A secção termina com a Tabela 12, de forma a permitir ao leitor uma melhor comparação entre estes.

De uma observação preliminar à Tabela 12, é possível destacar a principal divisão entre os vários mecanismos de consenso apresentados. A divisão é então feita através do domínio de cada um, sendo que este pode tomar os valores de *Blockchain com permissões* e *Blockchain sem permissões*.

São características de todos os mecanismos de consenso cujo domínio são as *Blockchains* sem permissões, a capacidade de estes conseguirem atingir consenso numa rede governada pela desconfiança entre os seus utilizadores. Estes conseguem-no com recurso a um gasto elevado por parte do utilizador, no caso do *Proof-of-Work* em recursos computacionais, e no caso do *Proof-of-Stake* em investimento de *tokens* do sistema. Alternativas que pretendam acabar com

esse gasto vão surgindo, como é o caso do mecanismo de consenso *Proof-of-Burn* (*PoB*) (Frankenfield, 2018). O *PoB* apresenta-se como sendo uma alternativa ao *PoW*, de igual performance, sem os gastos em recursos computacionais que este exige, e segue o princípio de que um utilizador deve queimar os seus *tokens*, como garantia de que estes foram gastos. Essa garantia, atribui ao utilizador uma possibilidade de ser selecionado para a publicação de um novo bloco. Ainda que na sua definição esta diga que o utilizador deve queimar os seus *tokens*, estes não são realmente destruídos, sendo ao invés enviados para um *unspendable address*, apenas identificado por uma chave pública (Smith, 2019).

Associado ao facto de os seus utilizadores serem conhecidos entre si, e dessa forma apresentarem uma maior confiança uns nos outros, os mecanismos de consenso implementados nas *Blockchains* com permissões usam da característica de que não necessitam de um elevado trabalho computacional, contrariamente ao observado nos mecanismos de consenso abordados anteriormente. Oferecendo uma alternativa ao *PoS*, é apresentado o *Proof-of-Activity* (*PoAc*) (Tschorsch & Scheuermann, 2017), pretendendo resolver os problemas comportamentais de ausência dos seus utilizadores encontrados no anterior. Modelos como o *Proof-of-Authority* (*PoAu*) ou como o *Proof-of-Elapsed-Time* (*PoEt*), oferecem uma boa performance, no entanto encontram-se dependentes de uma entidade responsável (Yaga et al., 2018). No caso do *PoAu*, os utilizadores vão estar sempre dependentes da confiança depositada no nó publicador e no caso do *PoEt*, os utilizadores vão estar dependentes dos servidores dedicados à execução deste. O modelo *Raft* foi desenhado com inspiração no *Paxos*, oferecendo a mesma performance e eficiência, apresentando melhorias na sua compreensão e implementação (Huang et al., 2018). Por fim, são apresentados modelos como o *Ripple* ou *Tendermint*, desenhados e otimizados para executar nas plataformas *Blockchain* que lhe dão o nome (Zheng et al., 2017).

Percebemos então que cada mecanismo de consenso é diferente, e, consequentemente, o seu domínio de aplicação e características. Resta ao leitor uma análise profunda a estes, de forma a que seja possível determinar qual o que melhor se adequa às necessidades da sua implementação de *blockchain*.

Tabela 12 - Comparação de mecanismos de consenso, adaptada de (Christidis & Devetsikiotis, 2016; Frankenfield, 2017; Ongaro & Ousterhout, 2014; Tschorsch & Scheuermann, 2017; Yaga et al., 2018)

| Nome | Objetivo | Pontos fortes | Pontos fracos | Domínio |
|---|---|---|---|---|
| *Proof-of-Work* (Yaga et al., 2018) | Permitir a publicação de blocos usando um puzzle criptográfico de extrema dificuldade de resolução, permitindo realizar transações entre participantes que não confiem uns nos outros | Difícil executar um ataque de *denial of service*, ao inserir dados fraudulentos na *blockchain;*<br><br>Aberto a todos os praticantes que o queiram executar; | Requer muito trabalho computacional;<br><br>Consumo elevado de energia;<br><br>Possível ataque dos 51%, obtendo o poder computacional suficiente; | *Blockchains* sem permissões |
| *Proof-of-Stake* (Yaga et al., 2018) | Permitir a publicação de novos blocos com usando recursos computacionais pouco exigentes, permitindo na mesma realizar transações entre participantes que não têm confiança entre si | Requer menos trabalho computacional que o modelo PoW;<br><br>Aberto a qualquer participante que deseje ter um *stake* no sistema; | Não existência de um mecanismo que evite a criação de uma *pool* de *Stakholders*, evitando desta forma a centralização do sistema;<br><br>Possível ataque dos 51%, obtendo *stake* suficiente; | *Blockchains* sem permissões |

| | | | | |
|---|---|---|---|---|
| *Proof-of-Burn* (Frankenfield, 2018) | Desenvolver um mecanismo de consenso com as mesmas características do PoW, sem, no entanto, ser tão dispendioso quanto este | Mais económico quando comparado com o modelo PoW | O utilizador tem de gastar *tokens* de forma a garantir uma possibilidade de executar o consenso | *Blockchains* sem permissões |
| *Proof-of-Publication* (Tschorsch & Scheuermann, 2017) | Criar um mecanismo de consenso que garanta o ordenamento de mensagens recorrendo a um serviço de *timestamping* | Possibilidade de garantir a integridade da informação encriptada com recurso a *timestamps* | Este mecanismo de consenso está dependente da confiança depositada no serviço de *timestamping* | *Blockchains* sem permissões |
| *Proof-of-Activity* (Tschorsch & Scheuermann, 2017) | Usufruir de um sistema descentralizado que premeia utilizadores ativos | A propriedade de *aging* de um *token* do sistema não incrementa caso um utilizador se encontre *offline* | Requer um grande trabalho computacional | *Blockchains* com permissões |
| *Proof-of-Authority* (Yaga et al., 2018) | Criar um processo de consenso centralizado, de forma a minimizar a taxa de criação e de verificação de blocos | Rápida taxa de confirmação de blocos; | Assume que o nó que publica não foi comprometido;<br><br>Leva à centralização dos nós que publicam blocos, tendo assim um ponto de falha central; | *Blockchains* com permissões |

| | | | | |
|---|---|---|---|---|
| | | | Baseia-se na reputação dos nós para a escolha de qual vai ser o próximo a publicar blocos; | |
| *Proof-of-Elapsed Time* (Yaga et al., 2018) | Criar um mecanismo de consenso que oferecesse as mesmas garantias de segurança oferecidas pelo modelo PoW, sendo, no entanto, mais económico que este. | Requer menos trabalho computacional que o modelo PoW; | Necessita de hardware específico para obtenção dos tempos de espera;<br><br>Assume que o hardware não foi comprometido;<br><br>É praticamente impossível obter sincronização de relógios em sistemas distribuídos (Lamport, 1978); | *Blockchains* com permissões |
| *Pratical Byzantine Fault Tolerance* (Christidis & Devetsikiotis, 2016) | Providenciar uma solução para o problema dos Generais Bizantinos funcional em sistemas assíncronos | Possibilidade de resolver o problema dos Generais Bizantinos usando menos recursos computacionais, quando comparado com os modelos de PoW e PoS | Propicio a falhas | *Blockchains* com permissões |
| *Raft* (Ongaro & Ousterhout, 2014) | Gerir *logs* replicados por servidores distribuídos. Desenhado de forma a facilitar a sua | Fácil de perceber e implementar; | Propicio a falhas | *Blockchains* com permissões |

| | | | | |
|---|---|---|---|---|
| | implementação, mantendo a performance e eficiência do *Paxos*. | Performance e eficiência tão boa como no *Paxos*;<br><br>Não requer trabalho computacional exigente; | | |
| *Ripple* (Zheng et al., 2017) | Atingir um consenso, tolerando falhas bizantinas até (n-1)/5, sendo n o número de utilizadores na rede | Usa uma *Unique Node List*, o que permite um aumento na eficiência durante a consulta dos servidores no processo de consenso | Um *ledger* é considerado como *closed* caso 80% da *Unique Node List* concorde com a transação a ser adicionada a este | *Blockchains* com permissões |
| *Tendermint* (Zheng et al., 2017) | Criar uma adaptação do problema Bizantino, aplicando-a à validação de blocos | Um bloco só é considerado como validado caso receba a maioria dos votos dos utilizadores, nas três fases pelo qual tem de passar durante o processo de consenso | Sem informação suficiente para desenhar uma conclusão sobre os seus pontos fracos | *Blockchains* com permissões |

## 3.4 Smart contracts

O conceito inicial de *smart contract* foi apresentado por Nick Szabo (1997), definindo-o como:

> *"a computerized transaction protocol that executes the terms of a contract".*

Com os constantes avanços no desenvolvimento da tecnologia *Blockchain*, esta começou a adaptar o conceito de *smart contract* às suas funcionalidades. Surgem então novas definições para *smart contract* na literatura atual. São, de seguida, apresentadas algumas dessas definições. Não podendo apresentar todas, são escolhidas apenas algumas, de acordo com a reputação dos seus autores.

Assim, Gupta (2017), define um *smart contract* como:

> *"an agreement or set of rules that govern a business transaction".*

Já Christidis & Devetsikiotis (2016), cujo foco de trabalho recai sobre *smart contracts* e as suas aplicações na *Internet of things*, define um *smart contract* como:

> *"scripts stored on the blockchain".*

Por fim, o *NIST* (Yaga et al., 2018), documento de referencia literária para a tecnologia *Blockchain*, define um *smart contract* como:

> *"a collection of code and data (…) that is deployed using cryptographically signed transactions on the blockchain network".*

Das diversas definições apresentadas, é possível observar que estas partilham todas do mesmo conceito, um pedaço de código que está guardado na rede *blockchain* e que executa múltiplos acordos de uma transação. Todavia, nem todas as blockchains conseguem armazenar ou executar *smart contracts* (Yaga et al., 2018). São exemplos desta, a *Blockchain Bitcoin.*

Dos *smart contracts*, é expectável um comportamento determinístico, de forma a permitir a obtenção de consenso após a sua execução (The Linux Foundation, 2018a). Ao terem um comportamento determinístico, é possível considerar os *smart contracts* como atores autónomos na *blockchain*, onde todos os resultados da sua execução são conhecidos (Christidis & Devetsikiotis, 2016).

Por estarem alocados na *blockchain*, os *smart contracts* são visíveis e executados por todos os participantes desta, que, assim, podem analisar o seu código e todos os resultados possíveis da sua execução (Christidis & Devetsikiotis, 2016).

Os *smart contracts*, no entanto, apresentam ainda algumas ressalvas, nomeadamente no que toca à escalabilidade e performance destes (The Linux Foundation, 2018a), ou até mesmo quando o *smart contract* tem interações que vão além do digital – problema do Oraculo (Yaga et al., 2018), abordado em detalhe na secção 3.5.3.

A Figura 11 apresenta um exemplo de um *smart contract* implementado em *Solidity* – linguagem de implementação de *smart contracts* para a plataforma *Ethereum* (Ethereum, 2018). Este simples contrato vai executar duas funções. A primeira função, *open*, é responsável por guardar numa estrutura de dados os valores passados por argumento. A segunda função, *query*, é responsável por retornar esses valores.

```
pragma solidity ^0.5.1;
contract simple {
    mapping(string => string) private map;

    function open(string memory _key, string memory _value) public {
        map[_key]= _value;
    }

    function query(string memory _key) public view returns(string memory)  {
        return map[_key];
    }
}
```

Figura 11 - Exemplo de smart contract implementado em Solidity, adaptado de (Persistent Systems, 2019e)

Ainda que as suas imensas aplicações ainda estejam num estado teórico, é de seguida apresentado um exemplo de uma possível aplicação de um *smart contract*. Este surge no âmbito de *Service Level Agreements* (*SLA*), que se apresenta pela sua definição como uma representação dos requisitos que devem ser cumpridos pelos fornecedores do serviço contratado. No caso desses requisitos não serem cumpridos, o cliente passa a ter direito a uma indeminização. O processo de indemnizar o cliente é, no entanto, um processo demorado e que necessita de pessoas dedicadas à execução dessas tarefas. Graças à sua automatização, segundo (Scheid & Stiller, 2018), um *smart contract* pode facilitar e apressar esse mesmo processo.

## 3.5 Riscos e limitações da tecnologia

Com o aparecer de novas tecnologias, há uma tendência de as sobrevalorizar, sendo pelas novidades que esta apresenta (Iansiti & Lakhani, 2017) ou pelo conhecido *fear of missing out* (Yaga et al., 2018), levando a que a que haja uma incorporação forçada em certos projetos, sem que sejam estudados os riscos ou limitações associados. A tecnologia *Blockchain* não é diferente das outras (Yaga et al., 2018). Ao longo desta secção são apresentados diversos riscos e limitações desta. À semelhança da secção 3.3, serão apresentados apenas alguns temas, de forma a que o texto não seja extensivo. Caso o leitor pretenda ler um pouco mais sobre o tema, pode consultar o Capítulo 7 do documento fornecido pelo *NIST* (Yaga et al., 2018).

### 3.5.1. Imutabilidade

Na literatura disponível encontramos a palavra imutável associada à tecnologia *Blockchain*. Isso não é completamente verdade, uma vez que em certas situações esse mesmo conceito de imutabilidade pode ser violado (Yaga et al., 2018).

Nas *Blockchains* sem permissões, ao abrigo do que foi afirmado na secção 3.3, ao sofrer um *fork*, a *blockchain* considera como verdadeira a maior cadeia de blocos, por consequência de ter maior trabalho computacional despendido. A adoção da cadeia de blocos maior pode, no entanto, apresentar-se sob a forma de um ataque dos 51% - ataque em que o atacante reúne recursos computacionais suficientes para superar a taxa de geração de novos blocos (normalmente corresponde a 51% dos recursos computacionais totais da rede) (Learn Cryptocurrency, 2019). Este ataque não é difícil, uma vez que basta replicar o processo de

criação de blocos, desde o bloco modificado até ao último bloco inserido. É, no entanto, muito dispendioso (Yaga et al., 2018).

As *Blockchains* com permissões também sofrem deste possível ataque, uma vez que estas são maioritariamente controladas por um proprietário ou por um consorcio de proprietários, que se assim o desejarem, podem adicionar um número arbitrário de novos blocos à *blockchain* (Yaga et al., 2018).

### 3.5.2. Utilizadores envolvidos no controlo da blockchain

Dizer que a *blockchain* é um sistema sem controlo e propriedade, e que não é controlada por nenhuma entidade, é um equívoco. Uma *blockchain*, independentemente do seu tipo (sem permissões ou com permissões), é sempre controlada por alguém. No caso das *blockchains* sem permissões, quem as controla são os desenvolvedores, utilizadores comuns e nós publicadores de blocos. Já as *blockchains* com permissões vão ser sempre controladas pela entidade proprietária ou por um consorcio (Yaga et al., 2018). Esta subsecção pretende demonstrar o papel que cada entidade tem no controlo da *blockchain*.

Nas *blockchains* sem permissões, os desenvolvedores desempenham um papel importantíssimo no controlo destas, uma vez que são os responsáveis pela sua criação, gestão do código fonte e desenvolvimento dos *smart contracts.* Este nível de controlo permite-lhes agir em interesse de certas partes, manipulando assim a *blockchain*. Os nós publicadores desempenham de igual forma um importante papel no controlo da *blockchain*. Ao ser realizado um *update* na *blockchain*, podem originar-se *forks*, mudando eventualmente a cadeia de blocos verdadeira e em certas situações, tornar transações inválidas. São os nós publicadores, que após uma discussão, tomam uma decisão sobre aceitar ou não esse *update*, influenciando os restantes utilizadores com a sua decisão (Yaga et al., 2018).

As *blockchains* com permissões, como referido na subsecção anterior, são controladas por um proprietário ou por um consórcio de proprietários. Esse controlo, embora mitigue comportamentos ilícitos na rede, torna os utilizadores da *blockchain* dependentes destes, e esperar que desempenhem as suas funções de forma honesta (Yaga et al., 2018).

### 3.5.3. Além do digital - Problema do Oráculo

A *blockchain* trabalha muito bem com informação transmitida dentro dos seus próprios sistemas. No entanto, apresenta algumas desafios quando necessita da interação com dados externos (Yaga et al., 2018). Tal problema é abordado na literatura como o problema do Oráculo (Buck, 2017). No contexto da tecnologia *Blockchain*, é possível definir um Oráculo como um agente intermediário, que encontra, valida e verifica dados para serem inseridos na *blockchain* e eventualmente usados por um *smart contract* (BlockchainHub, 2018). Os dados podem ser obtidos através de *input* humano ou de *input* por sensores.

O risco aparece no momento da validação dos dados anteriormente recolhidos. No caso destes resultarem de *input* humano, o utilizador pode inserir de forma deliberada, ou por engano, informação incorreta. No caso em que os dados provêm de *input* por sensores, pode acontecer que estes apresentem um funcionamento defeituoso, levando a que dessa forma seja recolhida informação que não corresponde à realidade (Yaga et al., 2018).

### 3.5.4. Vulnerabilidades nos smart contracts

Os *smart contracts* são desenvolvidos por humanos. Como tal, é expectável que estes apresentem vulnerabilidades. Essas vulnerabilidades podem, no entanto, surgir do mau design, mau desenvolvimento e falta de testes dos *smart contracts* (Li, Jiang, Chen, Luo, & Wen, 2017).

Num estudo conduzido por Luu, Chu, Olickel, Saxena, & Hobor (2016) à plataforma *Ethereum*, concluiu-se que 8833 dos 19,366 *smart contracts* que nesta corriam no momento da realização do estudo, apresentavam vulnerabilidades, podendo destacar a dependência de ordem nas transações, a dependência de *timestamps* e exceções mal geridas. Por sua vez, Chen, Li, Luo, & Zhang (2017) apresentam também um estudo à plataforma *Ethereum*, onde identifica sete padrões de *smart contracts* consumidores de *gas* – quantia de *Ether* (moeda nativa da plataforma *Ethereum*) que um utilizador deve pagar pela execução de operações na plataforma (Ethereum Community, 2016). Dos padrões identificados, os autores realçam código morto – código que nunca é executado, mas que se mantém na *blockchain*, operações opacas, execução de operações desnecessárias pela *Ethereum Virtual Machine* (*EVM*), e por fim, operações a executar em *loop* infinito (Li et al., 2017).

### 3.5.5. Consumo de recursos

A atividade de participar numa *blockchain* pode exigir um gasto considerável de recursos por parte de um utilizador, quer seja para publicar blocos ou simplesmente para manter uma cópia do *ledger* atualizado.

É conhecido que para *blockchains* públicas, particularmente as que implementam o *PoW* como o seu mecanismo de consenso, que o processo da sua execução é extremamente exigente no que toca ao consumo de recursos, quer sejam computacionais, elétricos e até mesmo de tempo (Yaga et al., 2018). Um estudo publicado pelo autor de Vries (2018) estima que o consumo elétrico anual atual da *Bitcoin* seja idêntico ao consumo feito pela Irlanda no mesmo período. Quanto ao processo de manter uma cópia atualizada do *ledger*, são identificados gastos relacionados com o consumo da largura de banda da ligação à internet de um utilizador, uma vez que é necessário o download de dados sempre que um novo bloco é adicionado à *blockchain* (Yaga et al., 2018).

### 3.5.6. Fim da Blockchain

Contrariamente aos sistemas de arquitetura centralizada, que com regularidade são criados e destruídos, nos sistemas *Blockchain* tal tarefa não pode ser desempenhada com a mesma facilidade. Por ser um sistema distribuído, é assim fácil de afirmar que este sistema pode nunca vir a ser destruído na sua totalidade, uma vez que poderão haver nós a manter uma cópia do *ledger* no momento dessa destruição, uma vez que não há mecanismos que permitam verificar e forçar a que todos os utilizadores destruam essa cópia (Yaga et al., 2018).

# Capítulo 4
# Plataformas Blockchain

Neste capítulo são apresentadas algumas plataformas *Blockchain*, com especial atenção ao detalhe do seu funcionamento. Neste, à semelhança das secções anteriormente apresentadas, secção 3.3 e 3.5, serão apresentadas apenas quatro plataformas *Blockchain*, de forma a não tornar o texto muito extenso, sendo que a sua escolha foi feita com base na maturidade do seu código, e na base de utilizadores destas, condições apresentadas por Dinh (2017).

As duas primeiras plataformas a serem apresentadas, são do domínio de *Blocskchains* sem permissões. Elas são a *Bitcoin* (Bitcoin Project, 2019), conhecida como sendo a primeira implementação da tecnologia *Blockchain* (Iansiti & Lakhani, 2017), e a *Ethereum* (Ethereum Foundation, 2019a), a primeira plataforma a implementar *smart contracts* programáveis (Ethereum Foundation, 2019b). Seguem-se duas plataformas do domínio de *Blockchains* com permissões, sendo estas o foco para o desenvolvimento do trabalho da presente dissertação. São elas *Quorum* (JPMorgan Chase & Co, 2019) e *Hyperledger Fabric* (The Linux Foundation, 2019f), respetivamente.

O capitulo termina com uma tabela comparativa entre diversas plataformas *Blockchain*, recorrendo para isso a métricas de comparação propostas por Beyer (2016); Dinh (2017); Fersht (2018) e LeewayHertz (2018). De igual forma, serão apresentadas plataformas que não foram detalhadas no correr do capítulo, referenciando-as caso o leitor pretenda saber mais sobre as mesmas.

## 4.1 Bitcoin

A *Bitcoin* surgiu em 2008, sob a forma do *paper* "*Bitcoin: A Peer-to-Peer Electronic Cash System*", publicado por Nakamoto (2008). Com o seu primeiro bloco publicado em 2009 (Blockchain Luxembourg, 2017), esta plataforma é considerada como sendo a primeira real aplicação da tecnologia *Blockchain* (Iansiti & Lakhani, 2017).

A plataforma é providenciada com um *ledger* público e partilhado por todos os participantes na rede, onde é mantido um registo ordenado de transações enviadas entre estes. Cada transação, consiste no processo de transferência de valor entre duas carteiras *Bitcoin*, onde após a realização de uma transação, é atualizado o saldo das carteiras. As transações seguem o modelo de *Unspent Transaction Output* (*UTXO*). Este modelo pode ser observado na Figura 12, e parte do princípio que a cada transação feita, a rede divide o total enviado em várias partes, como se dinheiro em papel se tratasse. O restante, é considerado como o troco da transação, e é reenviado para o utilizador que criou a transação (Hertig, 2018).

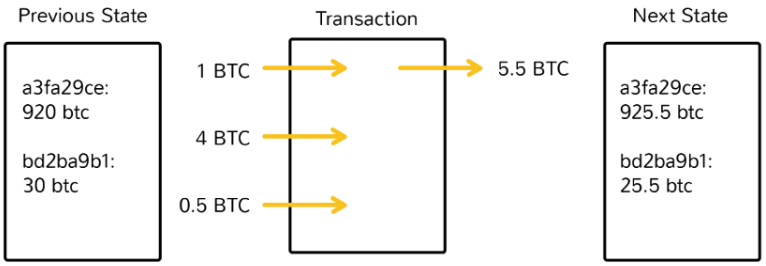


Figura 12 - Modelo de transação UTXO, adaptada de (Hertig, 2018)

Faz parte de uma transação a assinatura de utilizador que a criou e o conteúdo dessa transação. A assinatura da transação é gerada pela chave privada da carteira do utilizador que a criou, garantindo a integridade e proveniência da transação que esta assina. Por fim, as transações são verificadas e validadas pelo mecanismo de consenso *Proof-of-Work*, e agrupadas em blocos, constituídos por um *header* e os dados deste (Bitcoin Project, 2018a).

Ao implementar o mecanismo de consenso *PoW*, significa que qualquer utilizador pode participar neste, desde que, à partida, tenha na sua posse equipamento com requisitos suficientes para desempenhar esse processo. Fazem parte desses requisitos mínimos, uma máquina com 200 GB de espaço em disco, 1 GB de RAM, um processador com velocidade superior a GHz e um sistema operativo entre *Windows 7/8.x/10*, *Mac Os X* ou *Linux*. Ainda que não seja requisito obrigatório, o utilizador deverá também ter uma ligação à internet com velocidades de transmissão de 5 GB/dia de upload e 500 MB/dia de download. Mais detalhes sobre os requisitos podem ser encontrados em (Bitcoin Project, 2018b). Para um utilizador cuja utilização da plataforma se baseia no envio e receção de transações, tal equipamento não é necessário. Em tais casos, o utilizador deve apenas escolher a carteira que mais se adequa às suas necessidades, instalá-la no seu terminal (*Desktop* ou *Mobile*) e adquirir *bitcoins* – cripto moeda nativa da plataforma *Bitcoin*, de forma a poder realizar o envio de transações. Caso o utilizador pretenda apenas receber transações, um processo perfeitamente válido, o passo de aquisição de *bitcoins* passa a ser redundante.

## 4.2 Ethereum

A *Ethereum* é uma plataforma de *blockchain* pública e *open-source*, que permite que qualquer utilizador possa desenvolver e usufruir de aplicações descentralizadas (*DApps*), alocadas na *blockchain* (Ethereum Foundation, 2019a), sendo dessa forma uma plataforma adequada para aplicações de automatização de interações entre os nós participantes da rede.

Esta plataforma, implementa conceitos conhecidos aos utilizadores da *Bitcoin*, apresentando, contudo, melhorias na sua adaptabilidade e flexibilidade, permitindo que seja reprogramada para a realização de tarefas com certa complexidade computacional. Enquanto que na plataforma *Bitcoin* é usado o modelo *UTXO* para a realização de transações, na plataforma *Ethereum* é usado o modelo de estados, significando assim que cada nó mantém na sua posse o estado mais recente de todos os *smart contracts* e de todas as transações realizadas na *blockchain* (Hertig, 2018). Um exemplo deste modelo pode ser observado na Figura 13.

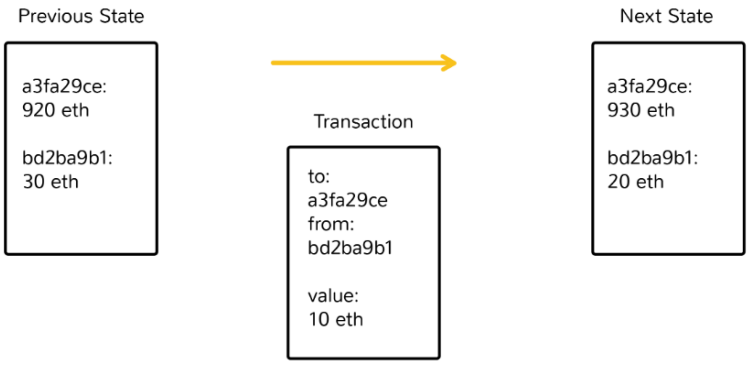


Figura 13 - Modelo de estados da plataforma Ethereum, adaptada de (Hertig, 2018)

A unidade central da plataforma passa então a ser as contas de utilizadores, que por sua vez vão realizar transações entre si. Isto permite que os fundos de uma conta possam ser portados para outra conta, caso um utilizador assim o deseje. As contas, podem tomar dois possíveis valores, *Externally Owned Accounts* (*EOAs*) e *Contract Accounts* (*CAs*). As *EOAs* são controladas por utilizadores, que por sua vez controlam as chaves privadas que lhe dão o acesso a essas contas. É possível afirmar que a maioria das transações são efetuadas entre este tipo de conta. As *CAs* são controladas pelo código que dela faz parte, os *smart contracts* – código executável alocado na *blockchain*, que executa ao ser enviado para o endereço dessa conta uma transação (Ethereum Community, 2017).

Os *smart contracts* são escritos em *Solidity* (Ethereum, 2018), compilados em *bytecode*, e interpretados pela *Ethereum Virtual Machine* (*EVM*). A *EVM*, é o núcleo da plataforma *Ethereum*, e é responsável pela execução dos *smart contracts* que sejam depositados na *blockchain*. Como forma de manter o consenso entre os nós, todas as operações realizadas na *blockchain* e por consequência executadas pela *EVM*, são executadas por todos os nós participantes da rede, levando a que a plataforma *Ethereum* seja equivocamente apelidada de "computador universal (Ethereum Foundation, 2019b).

A plataforma *Ethereum* faz uso de uma cripto moeda – o *Ether*, sendo para esta aplicados em duas vertentes. A primeira, para o simples uso na realização das transações e a segunda para o pagamento de *gas* – pequenas taxas aplicadas aos utilizadores pela execução de tarefas na *blockchain*, como por exemplo a execução de *smart contracts.* A implementação desta taxa permite que seja diminuída a possibilidade de ataques maliciosos, como por exemplo, ataques sob a forma de código a correr infinitamente, que dessa forma esgotaria os recursos do sistema (Ethereum Community, 2017).

De igual forma ao observado na plataforma *Bitcoin*, na plataforma *Ethereum*, um utilizador também pode instalar no seu terminal uma carteira e começar a usufruir da *blockchain Ethereum*. Para essas carteiras, estão disponíveis diversas opções, como por exemplo carteiras sob a forma de software, tais como a carteira *Mist* (Ethereum Foundation, 2018) ou a *Atomic Ethereum Wallet* (Atomic Wallet, 2019), ou carteiras de hardware, como por exemplo o *Ledger Nano S* (Ledger SAS, 2019). No entanto, para que um utilizador consiga participar no processo de *mining*, uma simples carteira não é suficiente. Assim, o utilizador deve correr na sua máquina o cliente *Geth* – implementação em *Go* de uma interface de linha de comandos de um nó completo da plataforma *Ethereum* (*go-ethereum*) (The go-ethereum Authors, 2016).

## 4.3 Quorum

A plataforma *Quorum* (JPMorgan Chase & Co, 2019) foi desenhada com o intuito de providenciar a industria com uma implementação com permissões do protocolo *Ethereum*, que, de igual forma, suporte a privacidade nas transações e contratos da *blockchain*. Por ser uma plataforma que tem como base o código *Ethereum*, herda todas as características desse mesmo códgio, e aproveitando todas as vantagens da pesquisa e desenvolvimento feita pela Ethereum Community (2017). Foram, no entanto, feitas algumas extensões a esse código, podendo destacar-se:

- A privacidade nas transações e contratos da *blockchain;*
- Mecanismos de consenso baseados em múltiplas rondas de votos;
- Uma gestão de permissões da rede *blockchain;*
- Uma melhor performance relativamente a este;

Como forma de atingir a privacidade nas transações e nos contratos alocados na *blockchain*, a plataforma *Quorum* passou a suportar dois estados, nomeadamente, o estado público, acessível a todos os nós participantes da rede, e o estado privado, acessível apenas a nós com permissões para tal. Dessa forma, todos os nós participantes partilham o estado público, e mantêm na sua posse o estado privado. No entanto, há uma dificuldade acrescida quando os nós tentam obter consenso relativamente ao seu estado privado, uma vez que as transações privadas podem apenas ser acedidas pelos nós envolvidos nestas. Para esses casos, foi implementada uma chamada *RPC* onde é passada a raiz do último bloco privado processado. Para obtenção do consenso para o estado público, é passado no bloco o *hash* da raiz do estado público.

Na Figura 14 observamos a arquitetura lógica da plataforma *Quorum* e a constituição dos seus componentes (JPMorgan Chase & Co, 2019). Uma explicação sobre os mesmos é apresentada logo de seguida.

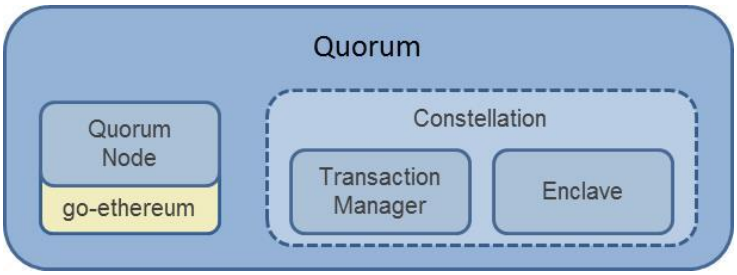


Figura 14 - Arquitetura lógica do Quorum, adaptada de (JPMorgan Chase & Co, 2019)

Ao *Quorum Node*, que é uma leve implementação do *Geth* (The go-ethereum Authors, 2016), foram feitas as seguintes modificações:

- A possibilidade de usar mecanismos de consenso tais como o *Raft* ou *Istanbul BFT*, em oposição ao *Proof-of-Work* implementado pela plataforma *Ethereum*;
- A camada P2P foi modificada de forma a permitir apenas conexões entre nós com permissões;

- A lógica de geração de blocos foi modificada, trocando a verificaça do '*global state root*' pela verificação de '*global public state root*';
- A lógica de validação de blocos foi também modificada, trocando o '*global state root*' no *header* do bloco por '*global public state root*';
- A lógica de validação de blocos foi modificada de forma a permitir transações privadas;
- A '*State Patricia*' – estrutura de dados utilizada para guardar as chaves dos seus utilizadores, foi dividida em dois: '*public state trie*' e '*private state trie*';
- A criação de transações foi modificada de forma a que pudessem alterar os dados da transação pelo *hash* encriptado destes, preservando a sua privacidade;
- Ainda que o parâmetro de *gas* se mantenha na plataforma, o seu valor foi removido;

O módulo *Transaction Manager* é responsável pela privacidade das transações. Este permite a realização de transações com dados encriptados sem que tenha que aceder às chaves privadas que os encriptam. Uma transação pode ter um de dois tipos, pública ou privada. É, no entanto, importante perceber que a plataforma *Quorum* não desenhou novas topologias de transações, adaptando o modelo oferecido pela plataforma *Ethereum*. Como forma de identificar o tipo de transação, foi adicionada a esta o campo *privateFor*. Caso esse campo esteja preenchido, com a chave publica do recetor, uma transação toma o valor de transação privada. A criação de transações publicas é feita de igual forma que na plataforma *Ethereum*.

Por fim, o trabalho criptográfico de encriptação dos dados de uma transação, é delegado ao módulo *Enclave*. Este, em conjunto com o *Transaction Manager*, reforça a privacidade, gerindo a encriptação e desencriptação dos dados transacionados, guardando as chaves privadas dos utilizadores, e funcionando como um *Hardware Security Module* (*HSM*), isolado dos outros componentes.

## 4.4 Hyperledger Fabric

A plataforma *Hyperledger Fabric* apresenta-se pela sua definição, em (The Linux Foundation, 2019k), como:

> "*an open source enterprise-grade permissioned distributed ledger technology (DLT) platform, designed for use in enterprise contexts, that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms*."

No correr deste texto são apresentados a arquitetura da plataforma e os conceitos dos principais componentes que a compreendem, seguindo a ordem encontrada em (Androulaki et al., 2018).

Com a plataforma é possível a execução de aplicações descentralizadas, desenvolvidas em linguagens de programação de uso comum, como por exemplo *Go*, *Java* ou *Node.js.* Uma aplicação descentralizada é por sua vez compreendida por um *smart contract*, ou *chaincode* no contexto da plataforma *Hyperledger Fabric*, compreendido pelo código que implementa a lógica da aplicação e pela política de aprovação – sistema de avaliação de transações, executada na fase de validação. Ambas as fases apresentadas serão abordadas em detalhe nos seguintes parágrafos. Começando pela sua arquitetura, esta segue o modelo de *execute-order-validate*, contrariamente ao modelo *order-execute* implementado nas plataformas *Bitcoin*

e *Ethereum*. Segue uma explicação do curso de execução do modelo implementado pela arquitetura da plataforma:

1. Um cliente envia propostas de transações para os *peer nodes*, indicados na política de aprovação, para execução por parte destes;
2. Os *endorser nodes* (nós que fazem parte da política de aprovação), simulam as propostas de transações recebidas, executando o *chaincode* correspondente;
3. Da simulação realizada pelos *endorser nodes* resultam dois conjuntos, *readset* – conjunto de *chave-valor* lido durante a simulação, e o *writeset* – conjunto de *chave-valor* modificado com o resultado da simulação;
4. Terminada a simulação, esta é assinada pelos *endorsers nodes*, e reenviada para o cliente sob a forma de *proposal response*;
5. Recebida uma *proposal response*, o cliente vai recolher as respostas dos restantes *endorser nodes*, até satisfazer o número definido na sua política de aprovação. Nesse processo, o cliente deve certificar-se de que todas as *proposal responses* têm conjuntos de *readset* e *writeset* idênticos. Realizada essa verificação, o cliente cria uma nova transação e envia-a para o serviço de ordenamento de transações.
6. O serviço de ordenamento é apenas responsável por ordenar as transações recebidas, não mantendo um estado das mesmas, e sem as executar e validar. Tal conceito, atribui à plataforma *Fabric* a capacidade de poder separar a camada de consenso da camada de execução e validação. Ordenadas as transações, são agrupadas em blocos e enviadas para os *peers* para posterior validação;
7. Recebido um novo bloco, este entra no processo de validação, que por sua vez consiste em três distintos passos:
   - Primeiro, é realizada uma verificação da política de avaliação, em paralelo a todas as transações. No caso uma verificação não ser satisfeita, a transação correspondente é marcada como inválida;
   - De seguida é realizada uma verificação aos conjuntos de *read-write*, gerados no processo de simulação da proposta de transação. No caso de as versões destes conjuntos não coincidirem, a transação é marcada como inválida;
   - Por fim, é feito um *update* ao *ledger*, adicionando o novo bloco à blockchain, e atualizando o seu estado. A atualização do estado é feita escrevendo no estado local o resultado do conjunto *writeset*, gerado nos passos anteriores;

Uma interpretação gráfica do processo anteriormente descrito pode ser observada na Figura 15.

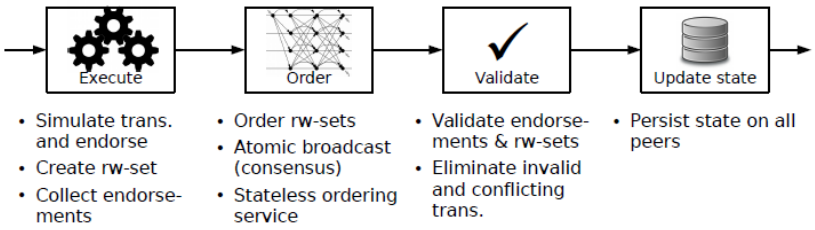


Figura 15 - Modelo execute-order-validate, adaptada de (Androulaki et al., 2018)

Conhecida a arquitetura da plataforma e o fluxo de execução de uma transação, são de seguida apresentados os componentes constituintes da plataforma. São eles:

- ***Membership Service***

O *membership service provider* (*MSP*) é o componente da plataforma responsável pela atribuição de identidades aos nós participantes na rede. Este, gera certificados criptográficos que servem como forma de autenticação para os nós participantes. Tal processo é necessário, uma vez que ao ser uma *blockchain* com permissões, todas as interações entre os participantes necessitam de ser autenticadas. A implementação do *MSP* oferecida pela plataforma permite o uso de metodologias de autenticação *public key infrastructure* (*PKI*), e adicionalmente também consegue acomodar autoridades de certificação (*CAs*) comerciais. Caso o utilizador necessite, a plataforma *Fabric* oferece uma implementação de uma *CA*, apelidada de *Fabric-CA*. Por fim, é da responsabilidade do *MSP*, garantir que todos os participantes reconhecem e aceitam como válidas todas as identidades e autenticações dos restantes.

- ***Ordering Service***

O *ordering service* é o componente responsável pelo ordenamento das transações recebidas. Implementado com recurso ao serviço de mensagens *Apache Kafka*, este oferece um serviço de transmissão de mensagens independentemente de falhas que possam acontecer nos seus clientes. Além da organização de transações, destacam-se das suas funções:

1. A transmissão atómica da ordem das transações recebidas;
2. A reconfiguração de um canal, quando recebida uma *configuration update transaction*;
3. E, opcionalmente, o desempenho de funções de controlo de acesso à *blockchain*;

No momento da sua criação, o serviço recebe o *genesis block* da *blockchain*, compreendido por uma *configuration transaction*, que dessa forma define as suas propriedades. Destas propriedades, é possível que este possa ser configurado de duas formas:

1. A primeira forma, o *Orderin Service* é configurado como um serviço de ordenamento central, com a definição de *Solo orderer*;
2. A segunda, o *Orderin Service* é configurado como um serviço de ordenamento baseado em *Bizantine Fault-Tolerant* (*BFT*) *State Machine Replication* (*SMaRt*).

- ***Peers***

Ao serem separadas as fases de execução, ordenamento e execução, permite que estas possam ser escaladas de melhor forma. No entanto, como é característico dos mecanismos de consenso implementados pela plataforma, estes, ao serem limitados pela largura de banda utilizada, o *throughput* apresentado pelo serviço de ordenamento é também limitado por esta nos nós que o implementam. Com o intuito de resolver esses problemas de largura de banda, foi introduzida a componente *gossip* dos *peers*. Entre estes, a comunicação é feita com recurso a *gRPC* e usa *TLS* para a autenticação mútua dos *peers* que estão a comunicar. Assim, o principal propósito da introdução deste componente é a transmissão de mensagens ordenadas tirando vantagem do protocolo *push-pull.* Este protocolo desenrola-se em duas fases:

1. Na fase de *push*, cada *peer* seleciona um conjunto *random* de outros *peers*, para lhes enviar as mensagens pretendidas;

2. Na fase *pull*, cada *peer* seleciona um conjunto *random* de outros *peers*, e realiza sobre estes pedidos de mensagens perdidas no processo de *push*;

O processo de *push-pull* sofreu, no entanto, algumas melhorias, como por exemplo, a eleição de um líder que, sozinho faz *pull* das mensagens ao serviço de ordenamento, e posteriormente realiza a sua distribuição pelos outros *peers*, reduzindo a carga enviada por cada mensagem.

- ***Ledger***

O *ledger* da plataforma é um componente mantido por todos os *peers* participantes da rede. Este permite a persistência do estado da *blockchain* e permite a execução das fases de simulação, validação e atualização do *ledger*, abordadas anteriormente. O *ledger* consiste então em duas partes, o *ledger block store* e o *peer transaction manager* (*PTM*).

O *ledger block store* mantém a persistência dos dados dos blocos adicionados à *blockchain*, implementado como *append-only*. O *PTM* mantém o último estado da *blockchain* sob a forma de um tuplo *chave-valor*. Este pode ser implementado em *LevelDB*[1] ou *Apache CouchDB*[2]. Na Figura 16, podemos observar a constituição do *ledger* da plataforma *Fabric*. Assim, como referido anteriormente, o *ledger* (*L*) é composto pelo *block store* (*B*) (ou na figura como *Blockchain*) e pelo *PTM* (ou na figura como *world state*), sendo que este é determinado pelo anterior.

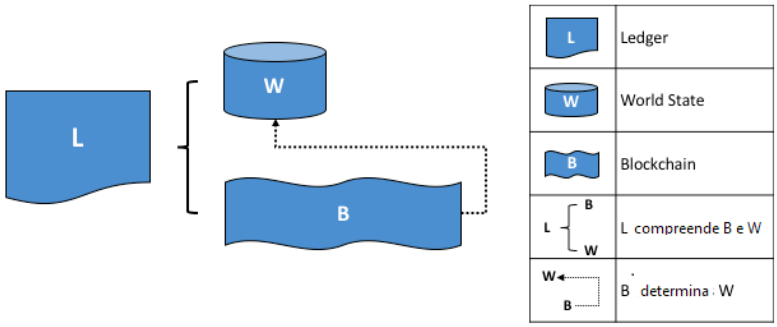


Figura 16 - Composição do *ledger* do *Hyperledger Fabric*, adaptada de (The Linux Foundation, 2018c)

Por fim, é ainda oferecida pela plataforma *Fabric* a possibilidade de criação e gestão de canais dentro da mesma rede. Tal característica atribui à plataforma uma maior capacidade de realizar transações num ambiente privado, uma vez que as transações realizadas dentro de um canal, estão disponíveis apenas para os *peers* participantes desses canais.

## 4.5 Visão geral das plataformas

Ao longo das secções anteriores foram descritas quatro das principais plataformas *blockchain* que existem ao momento, sendo que duas se aplicam no domínio de *blockchain* sem

[1] https://github.com/syndtr/goleveldb
[2] https://couchdb.apache.org

permissões e as restantes duas ao domínio de *blockchain* com permissões. Nesta seção será apresentada uma visão geral sob variadas plataformas. Serão apresentadas algumas que não foram detalhadas anteriormente, no entanto, é feita uma referência caso o utilizador pretenda saber mais acerca das mesmas. A comparação é apresentada sob a forma da Tabela 13, considerando os parâmetros propostos por Beyer (2016), Fersht (2018), LeewayHertz (2018) e Dinh (2017). A seguir à apresentação da tabela, é feita uma análise detalhada desta.

Os parâmetros propostos pelos autores acima referidos compreendem:

- Domínio – O domínio sobre o qual a plataforma opera;
- Tipo de *blockchain* – Se a plataforma implementa uma *blockchain* com permissões ou não;
- Mecanismos de consenso – Os mecanismos de consenso que esta implementa;
- *Smart contracts* – Se a plataforma suporta ou não o uso de *smart contracts.* No caso de esta suportar, é indicada a linguagem de programação no qual estes podem ser implementados;
- O ambiente de execução dos *smart contracts*;
- Modelo de dados utilizado – O modelo de dados sobre o qual a *blockchain* atua. Este pode ser *UTXO* (*Unspent Transaction Output)*, ou a conta de utilizador
- Cripto moeda – No caso de a plataforma utilizar uma cripto moeda, é apresentada o nome desta;

Tabela 13 - Visão geral das plataformas Blockchain, adaptada de (Beyer, 2016; Dinh et al., 2017; Fersht, 2018; LeewayHertz, 2018)

| Plataforma | Domínio | Tipo de *blockchain* | Mecanismos de consenso | *Smart contracts* | Ambiente de execução de *smart contracts* | Modelo de dados | Cripto moeda |
|---|---|---|---|---|---|---|---|
| *Bitcoin* | Cripto moeda | Sem permissões | PoW | – | – | UTXO | BTC |
| *Ethereum* | Aplicações descentralizadas<br>Cripto moeda | Sem permissões | PoW | Solidity | EVM | Conta de utilizador | ETH |
| *Quorum* | Múltiplos setores | Com permissões | Raft<br>IBFT | Solidity | EVM | Conta de utilizador | – |
| *Hyperledger Fabric* | Múltiplos setores | Com permissões | Framework conectável | Java<br>Node.js<br>Go | Dockers | Conta de utilizador | – |
| *Corda* | Serviços financeiros | Com permissões | Framework conectável | Java<br>Kotlin | JVM | UTXO | – |
| *Ripple* | Cripto moeda | Com permissões | Sistema de votos probabilísticos | – | – | UTXO | XRP |
| *Tezos* | Aplicações descentralizadas<br>Cripto moeda | Sem permissões | PoS | Michelson | Dockers | Conta de utilizador | Tezos |
| *BigchainDB* | Múltiplos setores | Com permissões | Tendermint | – | – | Conta de utilizador | – |

Analisando a tabela acima apresentada, podemos destacar os seguintes pontos:

- A principal comparação que se destaca é o tipo de *blockchain* implementada. Assim, é fácil fazer a distinção entre as *blockchains* com e sem permissões;
- De igual forma reparamos que o tipo de *blockchain* influencia diretamente o tipo de mecanismo de consenso que a plataforma implementa. Nos casos em que a *blockchain* é sem permissões, encontramos implementações de mecanismos de consenso com elevado trabalho computacional necessário para a sua execução, como é o exemplo do *PoW*, ou mecanismos de consenso que requerem um grande investimento do utilizador para terem maior probabilidade de serem selecionados para publicar um novo bloco, como é o exemplo do *PoS.* Para as *blockchains* com permissões o mesmo não se verifica, onde encontramos implementações do mecanismo *Raft* ou de um sistema de votos probabilístico;
- As plataformas que usam de uma cripto moeda foram desenvolvidas primariamente para o setor financeiro. No caso das plataformas *Bitcoin* (Bitcoin Project, 2019) e *Ripple* (Ripple, 2019), o seu uso é limitado a este setor. No entanto, as plataformas *Ethereum* (Ethereum Foundation, 2019a) e *Tezos* (Nomadic Labs, 2018) é possível a execução de aplicações descentralizadas;
- Das plataformas que permitem o desenvolvimento e execução de *smart contracts*, apenas a *Hyperledger Fabric* (The Linux Foundation, 2019f) e *Corda* (R3, 2018) o permitem fazer com recurso a linguagens de programação do uso comum, como por exemplo *Java* ou *Node.js*. A *Ethereum, Quorum* (JPMorgan Chase & Co, 2019) *e Tezos* implementam a sua própria linguagem de programação para o desenvolvimento dos *smart contracts*, utilizando *Solidity* e *Michelson* , respetivamente;
- Plataformas *open-source* como é o exemplo da *BigchainDB* (BigchainDB, 2018), apresenta-se como uma *blockchain database*, derivado das suas características apresentarem um misto de sistemas *blockchain* e de sistemas de bases de dados, utilizando *MongoDB* para a persistência de dados, associada ao mecanismo de consenso *Tendermint*.
- Outros projetos como *Openchain* (Coinprism, 2015), *HydraChain* (HydraChain, 2017), ou *MultiChain* (Coin Sciences, 2019) são deixados de fora deste estudo, uma vez que, ou pela falta de documentação, ou pelo facto dos seus repositórios públicos não serem recentes, dificultam assim o seu estudo.

# Capítulo 5
# Benchmarking de Plataformas Blockchain

Diferentes plataformas apresentam diferentes performances quando submetidas a casos de uso específicos. De forma a auxiliar o utilizador a exercer uma escolha fundamentada sobre qual a plataforma que melhor satisfaz as suas necessidades, podem, sobre estas, ser realizados testes de *benchmark*. Neste capítulo são apresentados trabalhos e avanços feitos nesta área, com especial foco sob a metodologia dos mesmos. O capítulo termina com uma comparação dos trabalhos anteriormente apresentados, permitindo ao leitor ter uma visão ampla sobre estes.

## 5.1 Benchmarkings existentes

Dinh e colegas (2018, 2017), apresentam-nos o seu trabalho, *BLOCKBENCH*[3], como sendo a primeira ferramenta de benchmarking para o estudo e comparação da performance de *blockchains* com permissões. Com este trabalho, os autores conduziram um estudo abrangente sobre as plataformas *Ehtereum* (*geth v1.4.18*), *Parity* (*release v1.6.0*) e *Hyperledger Fabric* (*release v0.6.0-preview*). Os autores terminam apresentando uma comparação dos sistemas de *blockchain* com os sistemas atuais de bases de dados, concluindo que os primeiros apresentam uma performance muito baixa em relação aos últimos. A ferramenta *BLOCKBENCH* quantifica a performance das plataformas *blockchain* relativamente ao seu *throughput*, latência, escalabilidade e tolerância a falhas. É ainda apresentada uma avaliação adicional à segurança das plataformas, simulando-se, para isso, ataques ao nível da rede. A arquitetura modular da ferramenta *BLOCKBENCH*, que pode ser observada na Figura 17, permite que o seu uso seja possível por parte por parte de qualquer utilizador. Dessa figura, é possível destacar três principais componentes, entre eles, o *IWorkloadConnector*, permite a adição de novos *workloads* para teste, a *Assynchronous Driver*, permite a recolha de informação relativa aos testes executados na *blockchain* e, o *IBlockchainConnector*, permite a adição de novas plataformas blockchain para teste.

Os autores consideraram a divisão da *blockchain* em quatro camadas de abstração. A razão para a escolha das mesmas deve-se ao facto de estas serem comuns a todos os três sistemas em teste. Estas são, a camada de consenso, a camada de modelo de dados, a camada de execução e por fim, a camada de aplicação. Os *workloads* implementados foram divididos em duas categorias, *Macro Benchmarks* e *Micro Benchmarks*, com o objetivo de testarem as diferentes camadas. Para cada *workload* foi implementado um *smart contract* em *Solidity* para as plataformas *Parity* e *Ethereum* e em *Golang* para a plataforma *Hyperledger Fabric*.

[3] https://github.com/ooibc88/blockbench

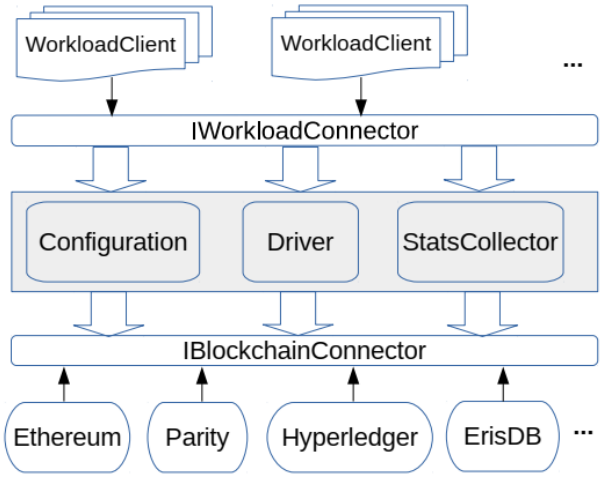


Figura 17 - Arquitetura da ferramenta BLOCKBENCH, adaptada de (Dinh et al., 2017)

Os autores Thakkar, Nathan, & Vishwanathan (2018) desenvolveram um estudo sobre a plataforma *Hyperledger Fabric* (*v1.0*). No seu trabalho, avaliam a performance da plataforma *Hyperledger Fabric* relativamente ao seu *throughput* e latência, variando para isso os vários parâmetros de configuração que a plataforma apresenta, tais como o tamanho dos blocos, a política de aprovação de uma transação, a alocação de recursos e a base de dados escolhida. No final do trabalho, os autores apresentam diversas sugestões de otimização que acabaram por ser adotadas em versões posteriores da plataforma.

A ferramenta *Hyperledger Caliper*[4] fornecida pela The Linux Foundation (2019c), apresenta-se como uma ferramenta de *benchmark* que permite ao utilizador testar blockchains com casos de uso predefinidos. No entanto, de momento, esta ferramenta suporta apenas as implementações de *blockchain* abrangidas pelo projeto *Hyperledger*, nomeadamente, *Hyperledger Burrow*, *Composer*, *Fabric* e *Sawtooth*. Com a ferramenta é possível recolher informação relativamente à taxa de sucesso de transações, o *throughput* para operações de escrita e leitura, latência de operações de escrita e leitura e o consumo de recursos dos nós constituintes da rede. A sua arquitetura, representada na Figura 18, é composta por três principais componentes. A camada de adaptação (*Adaptation Layer*) é usada para integrar os sistemas *blockchain* na ferramenta *Caliper* com recurso ao *SDK* ou à *REST API* da *blockchain* em teste. A camada de interface (*Interface & Core*) é responsável por executar as funções principais da ferramenta, tais como a recolha da informação resultante da execução dos testes e a geração de relatórios no formato *HMTL*. Por fim, a camada de aplicação (*Application Layer*) está equipada com testes previamente implementados para cenários típicos de implementações *blockchain*. Adicionalmente, é implementado um motor de *benchmark* padrão, permitindo desta forma um melhor entendimento do funcionamento da plataforma por parte dos desenvolvedores para que estes possam desenhar os seus próprios testes de forma mais rápida e eficaz. A ferramenta *Hyperledger Caliper* não tem, contudo, disponibilizados resultados públicos dos testes fornecidos. Para tal, um utilizador deve executar esses mesmos testes numa instância local.

[4] https://github.com/hyperledger/caliper

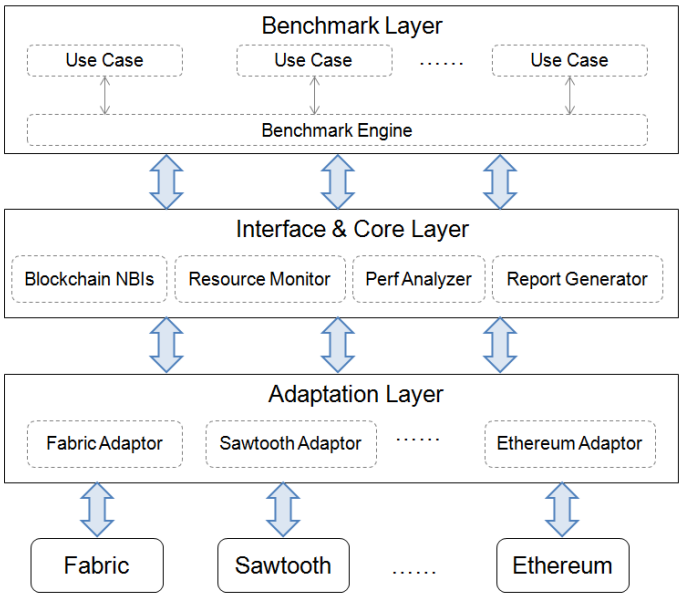


Figura 18 - Arquitetura da ferramenta Caliper, adaptada de (The Linux Foundation, 2019d)

Já Nasir, Qasse, Abu Talib, & Nassif (2018), realizaram uma análise à performance da plataforma *Hyperledger Fabric* comparando duas versões da mesma, nomeadamente a *0.6* e a *1.0*. A metodologia da usada para avaliar as plataformas dividiu-se em duas fases. Na primeira, foi feita uma recolha de métricas como o tempo de execução, latência e *throughput* para posterior análise da performance da plataforma. Na segunda fase, foi avaliada a escalabilidade das plataformas, variando o número de nós até um total de vinte, repetindo o processo realizado na primeira fase. Neste trabalho, os autores utilizaram uma versão adaptada da ferramenta *Hyperledger Caliper*.

O autor Leppelsack (2018) apresenta a sua tese de mestrado tendo como principal objetivo a avaliação da performance de DLT num ambiente que represente um caso de uso real. Ao longo deste trabalho, o autor pretendeu responder às seguintes questões: Que métricas são relevantes para medir a performance de DLT? Quais são as falhas dos *ledgers* distribuídos? Quais são os limites da avaliação da performance dentro de uma simulação? É possível prever níveis de performance tendo como base parâmetros da rede e do *ledger*?

Este estudo utilizou a ferramenta *Hyperledger Caliper* para a definição, geração e execução dos *workloads*. Na definição dos *workloads*, o autor baseou-se nas definições apresentadas por Dinh (2017). As métricas recolhidas pelo autor assentam no *throughput*, latência para operações de leitura e escrita, consumo de recursos, taxa de sucesso e tempo de execução.

A ferramenta *Gauge*[5] apresenta-se como uma ferramenta de *benchmarking* para a categorização da performance das plataformas *Hyperledger Fabric* (*v1.1*) e *Quorum* (*v2.0.0*). Desenvolvida pela Persistent Systems (2019d), esta plataforma é um *fork* da ferramenta *Hyperledger Caliper*, apresentando, contudo, algumas modificações em relação a esta. Entre elas destacam-se a possibilidade no envio de transações a um elevado ritmo. Os seus testes são divididos em duas categorias, nomeadamente *Controlled workloads*, responsáveis pela recolha de informação relativamente ao *throughput* e latência na confirmação das transações, e *Micro benckmarks*, que através da alteração de parâmetros a nível das transações e dos *smart contracts*, permitem estudar o impacto dessas mesmas alterações na latência das transações.

[5] https://github.com/persistentsystems/gauge

Da sua implementação é ainda possível recolher informação relativamente ao consumo de recursos nos nós, e realização de *benchmarks* que avaliem a escalabilidade da plataforma. No entanto, os testes de escalabilidade estão implementados apenas para a plataforma *Hyperledger Fabric*.

O trabalho apresentado por Pongnumkul, Siripanpornchana, & Thajchayapong (2017) foca-se na avaliação de duas plataformas *blockchain*. A plataforma *Ethereum* (*geth v1.5.8*) e *Hyperledger Fabric* (*v0.6*). Neste trabalho pretendeu-se avaliar a performance das duas plataformas quando submetidas a um cenário cujo número de transações invocadas pelos utilizadores é elevado. Para tal, foi simulado um cenário de uma aplicação de transferência de bens. Neste, contas de utilizadores podem ser criadas, bens podem ser atribuídos e transferidos entre as contas criadas. Foi, no entanto, tomada a decisão, por parte dos autores, de desligar os mecanismos de consenso e não os considerar nesta experiência. A justificação dos mesmos assenta no facto de as duas plataformas utilizarem dois mecanismos de consenso diferentes, e dada a sua natureza, estes afetariam diretamente a performance das plataformas em teste. Pela justificação apresentada, e por o aluno considerar que os mecanismos de consenso como um componente imprescindível dos sistemas *blockchain*, os resultados deste estudo não são considerados neste documento.

De seguida é apresentada a Tabela 14, cujo objetivo é facilitar ao leitor a comparação dos trabalhos anteriormente apresentados. A tabela encontra-se divida em duas partes. Na primeira parte são apresentadas as métricas, identificando os trabalhos que recolhem dados sobre estas, e na segunda parte, são apresentados os tipos de *workloads*, elementares ou realistas, identificando novamente os trabalhos que os implementam. As definições de *workloads* elementares e realistas foram apresentadas por Dinh (2017), e correspondem a *workloads* que não acontecem num ambiente de execução normal, e *workloads* de implementações de casos de uso reais respetivamente. A apresentação dos trabalhos na tabela encontra-se na mesma ordem pelo qual foram discutidos no texto acima.

Embora seja possível criar a tabela abaixo apresentada, os resultados gerados pelos trabalhos desenvolvidos não podem ser diretamente comparados, uma vez que, ainda que os trabalhos recolham informação relativamente às mesmas métricas, ou até mesmo às mesmas plataformas, como é possível observar no trabalho apresentado por Thakkar (2018) e Nasir (2018) (ambos testam a plataforma *Hyperledger Fabric v1.0*), a metodologia utilizada para a recolha desses resultados difere, podendo dessa forma influenciar os mesmos.

Tabela 14 - Comparação direta do trabalho desenvolvido na área de benckmarking

| Trabalho | Métricas | | | | | | | | Workloads | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Throughput | Latência | Consumo de Recursos | Escalabilidade | Taxa de sucesso | Tolerância a falhas | Tempo de execução | Métricas de segurança | Elementares | Realistas |
| BLOCKBENCH (Dinh et al., 2017) | | | | | | | | | | |
| Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform (Thakkar et al., 2018) | | | | | | | | | | |
| Hyperledger Caliper (The Linux Foundation, 2019c) | | | | | | | | | | |
| Performance Analysis of Hyperledger Fabric Platforms (Nasir et al., 2018) | | | | | | | | | | |
| Experimental Performance Evaluation of Private Distributed Ledger Implementations (Leppelsack, 2018) | | | | | | | | | | |
| Gauge (Persistent Systems, 2019d) | | | | [6] | | | | | | |
| Performance Analysis of Private Blockchain Platforms in Varying Workloads (Pongnumkul et al., 2017) | | | | | | | | | | |

[6] Implementado apenas na plataforma *Hyperledger Fabric*

## 5.2 Plataforma de Benchmark

Dos trabalhos apresentados e estudados anteriormente, seguiu-se uma discussão com os orientadores de forma tomar uma decisão sobre o caminho a adotar, neste caso, se deveríamos implementar uma plataforma de testes *benchmark* de raiz, ou dar continuidade a um trabalho existente. Para a tomada de decisão foram tidas em conta quatro características, nomeadamente a idade do projeto, a facilidade de perceção do seu código e funcionamento, as métricas que este recolhia, e o facto do projeto ser *OpenSource*. Foram considerados para comparação três dos trabalhos estudados. *BLOCKBENCH*, o primeiro trabalho a ser considerado, acabou por ser abandonado uma vez que o seu código era difícil de perceber, não se encontrava comentado, e já não era atualizado no período de um ano. O trabalho, Caliper, foi comparado diretamente com o trabalho *Gauge*, uma vez que o último apresenta uma implementação do anterior, e ambos eram projetos relativamente recentes. O fator que determinou a decisão da adoção do trabalho *Gauge* foi o fato de este ter implementado os adaptadores para a plataforma *Hyperledger Fabric* e *Quorum*, contrariamente ao observado no trabalho Caliper, que só tinha implementado o adaptador para a plataforma *Hyperledger Fabric.*

No decorrer desta secção é apresentada a plataforma *Gauge* e os seus componentes em detalhe. De seguida são apresentadas as modificações e contribuições que o aluno fez ao código da mesma. A secção termina com a apresentação do sistema para a execução dos testes de *benchmark*, e respetivas redes de teste.

### 5.2.1. Plataforma Gauge

A Figura 19 apresenta uma visão detalhada da plataforma *Gauge*. A plataforma é composta por dois principais componentes, que são o *Benchmark Engine* e a Fila de mensagens *Apache Kafka + Zookeeper.*

Percorrendo o seu curso de execução, o *Benchmark Engine* começa por consumir três ficheiros de configuração, o ficheiro de configuração do *benchmark*, da *blockchain* em teste, e da fila de mensagens. De seguida o componente Gerador de carga gera transações a um ritmo de envio designado no ficheiro de configuração do *benchmark.* O componente *Blockchain SDK* envia as transações para a *blockchain* em testes, utilizando o *Hyperledger Fabric Client SDK* em *Node.js* para a comunicação com a plataforma *Hyperledger Fabric*, e a biblioteca *web3js* para a comunicação com a plataforma *Quorum.* Após o envio das transações para a rede *blockchain*, o componente Monitor de recursos recolhe informação sobre os consumos de CPU e memoria RAM dos nós da rede. Paralelamente, o componente Monitor de eventos de blocos fica à escuta de novos eventos na *blockchain.* Um evento é caracterizado pela confirmação de um novo bloco na rede. Após a receção de um novo evento, o componente Monitor de eventos atribuí uma *timestamp* a esse bloco, representativa da data de confirmação de todas as transações inseridas nesse bloco. Esses eventos são posteriormente enviados para a fila de mensagens à espera de serem consumidos pelo componente Calculador de TPS e Latência. Por fim, o componente Calculador de TPS e Latência lê os eventos da fila de mensagens, e calcula o valor de transações por segundo e latência das transações desse evento.

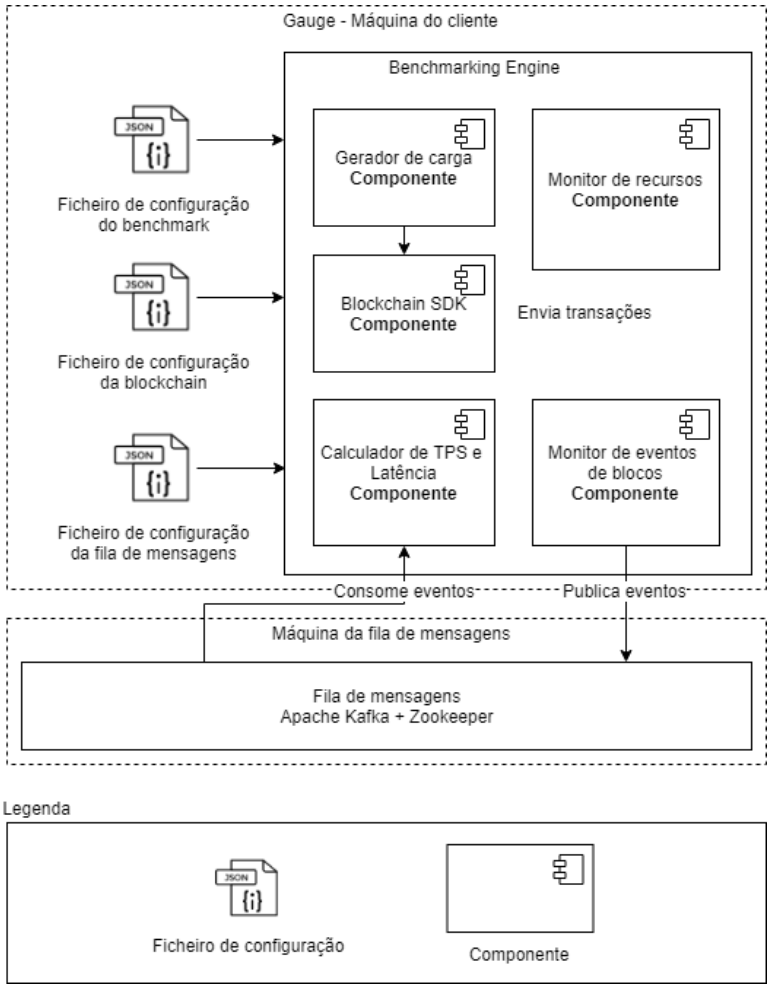


Figura 19 - Arquitetura da ferramenta *Gauge*, adaptada de (Persistent Systems, 2019b)

De seguida são apresentadas as modificações feitas pelo aluno à plataforma *Gauge.*

## 5.2.2. Modificações feitas à plataforma Gauge

Ainda que a plataforma *Gauge* já se apresentasse como uma ferramenta completa para a realização dos testes de *benchmarking*, foram realizadas 5 modificações ao seu código, para que esta se tornasse compatível com o modelo de execução pretendido pelo aluno. Nesta secção são apresentadas essas modificações.

- **Modificação 1 *Output* gerado pela ferramenta *Gauge***

A primeira modificação, reside na alteração do método de geração de *output* dos testes executados. Usando o seu código original, a plataforma gera um ficheiro *csv* por cada teste executado, escrevendo por cima de outro ficheiro de *output*, caso este exista. Como era intenção do aluno realizar os testes ao longo de dez corridas, foi alterada a forma como a plataforma gera o *output*, passando a gerar um ficheiro novo para cada ronda de testes executados. Mais detalhes sobre as rondas são fornecidos no Capítulo 6.

- **Modificação 2 *Script* de compilação e *deployment* de *smart contracts* na plataforma *Quorum***

Foi modificado o *script* original que acompanhava a ferramenta, para a compilação e *deployment* de *smart contracts* na plataforma *Quorum*, uma vez que este terá sido desenvolvido utilizando a biblioteca *web3.js v0.2.0.* No presente, essa biblioteca encontra-se na *v1.0.0*, que por sua vez é incompatível com a anterior, justificando dessa forma o seu *update* no contexto do desenvolvimento da dissertação (Furter, 2019).

- **Modificação 3 *Update* do componente *Blockchain SDK***

Foi iniciado o processo de *update* do componente *Blockchain SDK* para a comunicação com a plataforma *Quorum*, à semelhança da Modificação 2, passando este componente a usar a biblioteca *web3.js v1.0.0.* Este processo não foi, no entanto, concluído por questões temporais.

- **Modificação 4 Componentes da rede *Hyperledger Fabric***

Os componentes utilizados pela rede criada para testar a plataforma *Hyperledger Fabric* foram atualizados para a sua versão mais recente. Destes fazem parte o *Orderer* e o *Peer*. Detalhes sobre o seu funcionamento são apresentados na secção 4.4.

- **Modificação 5 *Script* para a geração dos artefactos da rede da plataforma *Hyperledger Fabric***

Em adição ao update feito aos componentes utilizados pela rede *Hyperledger Fabric*, foi desenvolvido um *script* para a geração dos artefactos da rede *Hyperledger Fabric*, tais como os certificados das organizações participantes, o bloco *genesis* e o canal da rede. Após concluído o processo de geração dos artefactos, o *script* renomeia as chaves geradas, para que o utilizador não tenha problemas com as diretorias das mesmas.

### 5.2.3. Sistema de execução dos testes de benchmark

Nesta secção é apresentado o *setup* do sistema para a execução dos testes de *benchmark.* O sistema montado vai executar um conjunto de testes, descritos na secção 6.1, com o objetivo de avaliar e comparar a performance entre as plataformas *Hyperledger Fabric* e *Quorum*. Em adição à plataforma existente, foi desenhado pelo aluno um *script* para automação da execução e recolha dos resultados dos testes. Um diagrama de contextualização pode ser observado na Figura 20.

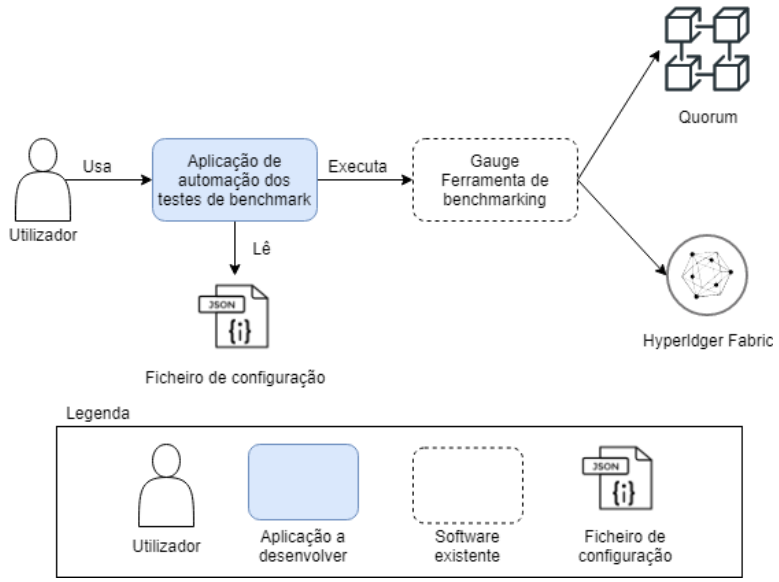


Figura 20 - Diagrama de contextualização do sistema de execução dos testes de benchmark

O utilizador usa o *script* de automação, que por sua vez executa o teste pretendido na plataforma *Gauge*. Por sua vez, o *Gauge* conecta-se às plataformas *Hyperledger Fabric* e *Quorum* para a execução dos testes. O *script* de automação foi desenvolvido em *Python* e interage com o utilizador através da linha de comandos. Este consome um ficheiro *json* de configuração, "*config.json*", presente na diretoria *"scripts"* da plataforma *Gauge*, com os parâmetros necessários para a execução dos testes. Desses parâmetros fazem parte:

"***virtual_machine***" – Informação relativa à máquina virtual que contém o cliente *Gauge*;

- *"host"* – *Host* da máquina virtual;
- *"port"* – Porto da máquina virtual, para a ligação *ssh*;
- *"user"* – *Username* da máquina virtual onde se tem a rede de testes;
- *"pass"* – *Password* da máquina virtual;

"***benchmark***" – Informação relativa ao teste que se quer executar;

- "*blockchain*" – O tipo da *blockchain* em teste. Pode tomar os valores de "*fabric*" e "*quorum*";
- "*workload*" – O nome do teste a executar. Este parâmetro deve ter o mesmo nome da diretoria do teste que se pretende executar;
- "*limit*" – O limite máximo para a geração das transações por segundo a enviar para a plataforma em teste;
- "*executions*" – O número total de execuções para o teste dado;
- "*type*" – O tipo de *benchmark* a executar. Pode tomar os valores de "*controlled*", "*micro*" e "*scalability*".
- "*n_chaincodes*" – Define o número de chaincodes a serem instanciados na rede de teste, para os testes de escalabilidade à plataforma *Hyperledger Fabric*. De igual forma, este parâmetro também serve para o número total de canais na

rede, igualmente usado nos testes te escalabilidade à plataforma *Hyperledger Fabric;*

Um exemplo desse ficheiro pode ser encontrado na Figura 21. O diagrama *UML* do *script* desenvolvido pode ser encontrado no Apêndice A Diagrama UML do script de automação dos testes de benchmark.

```
{
  "virtual_machine": {
    "host": "10.0.60.211",
    "port": 22223,
    "user": "ruidias",
    "pass": "1234"
  },
  "benchmark": {
    "blockchain": "fabric",
    "workload": "write",
    "limit": 3000,
    "executions": 10,
    "type": "controlled",
    "n_chaincodes": 50
  }
}
```

Figura 21 - Exemplo de ficheiro de configuração para o script de automação

Interpretando o exemplo acima, é expectável que o *script* execute o teste *write*, com um limite total de 3000 transações por segundo, sob a plataforma *Hyperledger Fabric* presente na máquina virtual com o *ip "10.0.60.211"*, num total de dez corridas.

### 5.2.4. Rede Blockchain utilizada para o benchmark

Apresentadas a plataforma usada para a execução dos testes, e as modificações de que esta foi alvo, são de seguida apresentadas as arquiteturas das redes utilizadas pelas plataformas *Hyperledger Fabric* e *Quorum*, respetivamente.

- ***Hyperledger Fabric***

Na Figura 22 é possível observar a arquitetura da rede utilizada para testar a plataforma *Hyperledger Fabric.* Esta é composta por duas organizações, na imagem representadas pelos triângulos a roxo e amarelo, cada uma compreendida por dois *peer nodes*. Cada nó tem uma cópia do *ledger* atualizado, e faz pedidos de transações para o canal disponível. No caso da rede em teste, existe apenas um canal. A ordenação das transações propostas fica a cargo do nó *Orderer*, que por sua vez, é do tipo *Solo.* A escolha deste tipo de *Orderer* baseia-se no facto de este processar as transações da mesma forma que um *Orderer* do tipo *Kafka* ou *Raft*, diminuindo o trabalho de manutenção e atualização desses nós. É, no entanto, aconselhável que o tipo *Solo* seja usado apenas para ambientes de desenvolvimento, uma vez que não são tolerantes a falhas (The Linux Foundation, 2019j). A organização da rede foi adaptada da documentação providenciada pelo *Hyperledger Caliper*, disponível em (The Linux Foundation, 2019e).

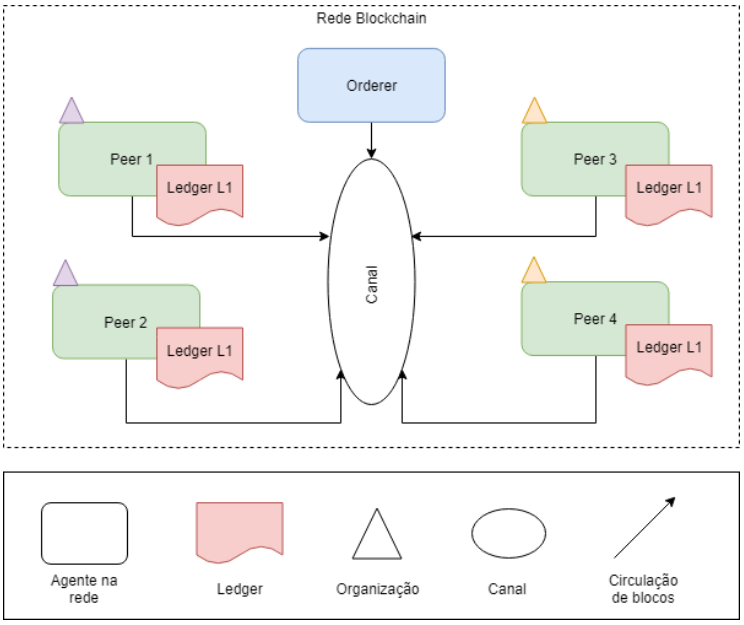


Figura 22 - Rede Hyperledger Fabric, adaptada de (The Linux Foundation, 2019h)

Caso o utilizador pretenda construir a rede com componentes à sua escolha, pode fazê-lo editando o ficheiro *"crypto-config.yaml"*, presente na diretoria da rede *"fabric-v1.4"*, com a nova topologia da rede. Logo de seguida deve gerar os artefactos da rede, como o *genesis block* e o canal da rede e os certificados de identificação dos participantes. Para simplificar este processo, o utilizador deve executar o *script "generate-crypto-materials.sh"*, presente na diretoria *"scripts"* da plataforma *Gauge*. Quando conectar a nova rede à plataforma *Gauge*, o utilizador deve ainda verificar o caminho dos artefactos dos participantes da rede, presente no ficheiro de configuração *"fabric.json"*, na diretoria do teste que se pretende executar.

- ***Quorum***

Contrariamente à plataforma *Hyperledger Fabric*, na plataforma *Quorum*, os nós da rede são todos do mesmo tipo, desempenhando funções de acordo com o papel que lhe é atribuído pelo mecanismo de consenso *Raft*. Um exemplo da rede utilizada para a realização dos testes pode ser observado na Figura 23. Esta é composta por cinco nós.

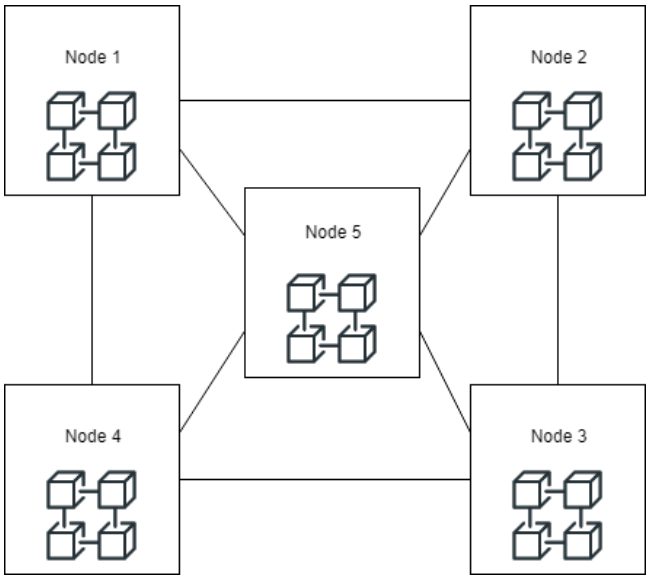


Figura 23 - Rede Quorum, adaptada de (JPMorgan Chase & Co, 2016)

O processo da criação da rede *Quorum* apresentou o primeiro desafio para o aluno. Após tentativas de criação da rede *From Scratch*[7], sem sucesso, o aluno optou pela criação da rede com recurso à ferramenta *Quorum Maker*[8], ferramenta sugerida pelos desenvolvedores da plataforma *Quorum*, após serem realizadas comunicações entre estes e o aluno via *Slack*. Esta ferramenta providencia o utilizador com uma interface que permite a criação e manipulação de redes *Quorum*.

## 5.3 Resumo do capítulo

Este capítulo apresentou trabalhos desenvolvidos na área de *benchmarking* a plataformas *blockchain*. Do estudo dos mesmos, foi possível a construção de uma comparação pela Tabela 14. Esta comparação foi importante, pois permitiu ao aluno identificar qual a melhor estratégia para o desenvolvimento do seu trabalho. Da estratégia, foi decidido, por parte do aluno e dos seus orientadores, dar continuidade ao trabalho apresentado pela (Persistent Systems, 2019c).

Continuando, foi apresentada a arquitetura da plataforma *Gauge*, incluindo os seus componentes. Foi também apresentado um diagrama de contextualização do sistema utilizado para o desenvolvimento dos testes. Este compreende um *script* de automação dos testes que usa a plataforma *Gauge*, e executa os testes sob as plataformas *Hyperledger Fabric* e *Quorum*. O diagrama atualizado pode ser observado na Figura 24.

[7] https://docs.goquorum.com/en/latest/Getting%20Started/Creating-A-Network-From-Scratch/
[8] https://github.com/synechron-finlabs/quorum-maker/wiki

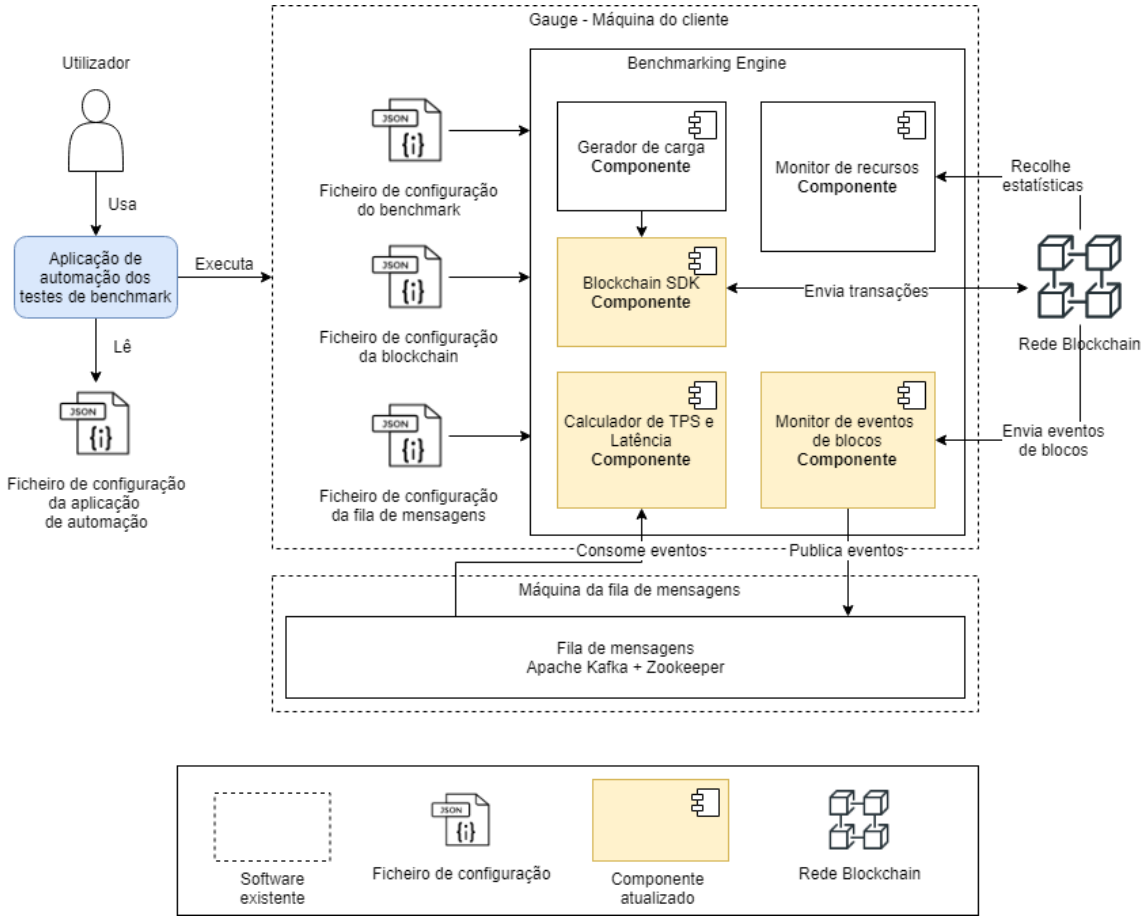


Figura 24 - Diagrama completo do sistema de benchmarking a plataformas blockchain

Da figura acima apresentada, podemos observar o diagrama completo do sistema de *benchmarking* utilizado pelo aluno. São também observáveis as modificações apresentadas na secção A, no diagrama representadas a amarelo.

O capítulo termina com a descrição das redes de teste utilizadas pelas plataformas *blockchain*, apresentando as suas arquiteturas, e detalhando os componentes que as compreendem.

# Capítulo 6
# Testes de Benchmark

Neste capítulo são descritos os testes de *benchmark* realizados às plataformas *Hyperledger Fabric* e *Quorum*. Os testes dividem-se em *Controlled Workloads*, *Micro-benchmarks* e *Scalability experiments*, no entanto, estes últimos encontram-se disponíveis apenas para a plataforma *Hyperledger Fabric*.

Para a execução de cada teste de *benchmark* a plataforma *Gauge* necessita de ter acesso ao ficheiro *json* de configuração do *benchmark*, presente na pasta do *workload* que se pretende executar. Esse ficheiro tem o nome da plataforma em teste, "*fabric.json*" ou "*quorum.json*", e a sua estrutura é igual para todos os testes executados, ainda que este sofra ligeiras alterações de acordo com o teste a ser executado no momento. De seguida é apresentado na Figura 25 um exemplo desse mesmo ficheiro.

```
{
  "test": {
    "clients": 1,
    "type": "ONE_CHANNEL",
    "rounds": [
      {
        "txNumbAndTps": [[1, 1]],
        "callback": "benchmark/fabric/write/open-write-only.js",
        "cmd": "open",
        "arguments": [
          {
            "args": [
              { "fname": "open" },
              { "key": "accountno_" },
              { "value": "1000" }
            ],
            "chaincodeid": "simple"
          }
        ]
      }
    ]
  },
  "blockchain": {
    "type": "fabric",
    "config": "./fabric.json"
  },
  "monitor": {
    "docker": {
      "name": ["http://10.0.60.211:2375/peer0.org1.example.com"]
    },
    "interval": 1,
    "type": "docker"
  }
}
```

Figura 25 - Excerto do ficheiro de configuração do benchmark

Da figura acima apresentada é possível destacar três parâmetros principais, que são *test*, *blockchain* e *monitor*. Segue a sua definição:

***test:*** definição do teste a executar pela plataforma *Gauge*;

- ***clients***: define o número de clientes *Gauge.* De forma a não reduzir a performance da plataforma *Gauge*, os autores da mesma aconselham apenas o uso de um cliente por instância;
- ***type***: define o tipo da experiência a ser executada. Este parâmetro só está disponível apenas para os ficheiros de configuração da plataforma *Hyperledger Fabric* e pode ter os seguintes valores:
  - **ONE CHANNEL** – para executar os *controlled workloads*;
  - **MULTI CHANNEL** – para executar os *channel scalability*;
  - **MICRO BENCHMARK** – para executar os *micro-benchmarks*;
- ***txNumbAndTps:*** define o array de sub-rondas [i, j], a executar em cada teste. Desse array, o parâmetro *i* define o total de transações enviadas para a plataforma em teste, e o parâmetro *j* o ritmo a que essas transações são enviadas;
- ***cmd***: corresponde ao tipo de operação a ser executada pelo *smart contract*. Este parâmetro pode tomar dois valores, nomeadamente:
  - ***open****:* para a realização de operações de invocação, como por exemplo a realização de transações;
  - ***query***: a realização de operações de consulta de dados na *blockchain*;
- ***arguments***: argumentos definidos pelo utilizador passados diretamente ao módulo de execução do teste. Estes devem conter:
  - ***chaincodeid***: id do *smart contract* instanciado na *blockchain*;
  - ***args***: argumentos recebidos pelo *smart contract* invocado. Este parâmetro deve conter duas partes. Uma parte deve ter o nome da função a executar dentro do *smart contract*, e o segundo parte a chave do mesmo;
- ***callback***: especifica o módulo usado em cada ronda do teste. Este módulo é responsável pelo início, execução e término do teste em causa;

***blockchain:*** define o tipo de *blockchain* sob o qual vai ser realizado o teste, podendo tomar os valores de "*fabric" e "quorum"*. Igualmente, é definido o caminho para o ficheiro de configuração dessa *blockchain*;
***monitor:*** define o *Docker container* sob o qual se vão recolher estatísticas dos seus recursos durante a execução dos testes;

Uma definição mais extensa sobre o ficheiro de configuração do *benchmark* está disponível em (Persistent Systems, 2019a).

## 6.1 Definição dos testes de benchmark

Nesta secção são apresentadas as definições dos testes de *benchmark.* Estes encontram-se divididos em *Controlled Workloads*, *Micro Benchmarks* e *Scalability Experiments.* A definição dos

testes é feita seguindo o modelo da Tabela 15, sendo estes adaptados dos testes disponíveis em (Persistent Systems, 2018). No entanto, após um processo de avaliação critica aos testes disponíveis, foi tomada a decisão de não incluir todos neste documento, procedendo-se à seleção dos que são apresentados de seguida.

Tabela 15 - Modelo de definição de um teste de benchmark

| | |
|---|---|
| Nome do teste | Nome do teste |
| ID | Número de identificação do teste |
| Descrição | Breve descrição do teste |
| Parâmetros de input | Parâmetros de input que o teste recebe para a sua execução |

### 6.1.1. Controlled Workloads

Os *Controlled Workloads* apresentam-se como testes de *benchmark* simples, cujo objetivo é medir o *througput* e latência das plataformas. Destes fazem parte:

- Testes de operações de escrita;
- Testes de operações de leitura;
- Testes de operações nulas;
- Modificação dos parâmetros do *Orderer*;
- Modificação da política de aprovação de uma transação;

Tabela 16 - Teste T1F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Write Only - Hyperledger Fabric |
| ID | T1F |
| Descrição | Este teste mede o Throughput e a Latência da plataforma, quando submetida a operações de escrita na Blockchain. Utiliza uma função simples, responsável por guardar numa estrutura de dados os valores passados por argumento. |
| Parâmetros de input | ***type*:** ONE_CHANNEL;<br>***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *simple;*<br>▪ ***args:*** *{*<br>*{**fname**: open},*<br>*{**key:** accountno},*<br>*{**value:** 1000}*<br>*}*<br>***callback:*** *benchmark/fabric/write/open-write-only.js* |

Tabela 17 - Teste T1Q - Quorum

| | |
|---|---|
| Nome do teste | Write Only - Quorum |
| ID | T1Q |
| Descrição | Este teste vai medir o Throughput e a Latência das plataformas, quando submetidas a operações de escrita na Blockchain. Utiliza uma função simples, responsável por guardar numa estrutura de dados os valores passados por argumento. |
| Parâmetros de input | ***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***money:*** *1000;*<br>***callback:*** *benchmark/quorum/write/open.js* |

Tabela 18 - Teste T2F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Read Only - Hyperledger Fabric |
| ID | T2F |
| Descrição | Este teste mede o Throughput e a Latência da plataforma, quando submetida a operações de leitura na Blockchain. Utiliza uma função simples, responsável por retornar os dados presentes na blockchain. |
| Parâmetros de input | O utilizador deve executar o script para popular a blockchain com dados antes de executar este teste. O script encontra-se na pasta do *workload* em causa, com o nome *"pre-populateKeys.js"*;<br>***type*:** ONE_CHANNEL;<br>***cmd*:** *query;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *simple;*<br>▪ ***args:*** *{*<br>*{**fname**: query},*<br>*{**key:** accountno},*<br>*}*<br>***callback:*** *benchmark/fabric/read/query-read-only.js* |

Tabela 19 - Teste T2Q - Quorum

| | |
|---|---|
| Nome do teste | Read Only - Quorum |
| ID | T2Q |
| Descrição | Este teste vai medir o Throughput e a Latência das plataformas, quando submetidas a operações de leitura na Blockchain. Utiliza uma função simples, responsável por retornar os dados presentes na blockchain. |
| Parâmetros de input | O utilizador deve executar o script para popular a blockchain com dados antes de executar este teste;<br>***cmd*:** *query;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***args:*** *{*<br>*{**accounts**: 100},*<br>*{**query:** true},*<br>*}*<br>***callback:*** *benchmark/quorum/read/query.js* |

Tabela 20 - Teste T3F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Null - Hyperledger Fabric |
| ID | T3F |
| Descrição | Este teste mede o Throughput e a Latência da plataforma, quando submetida a operações nulas. É semelhante ao teste Write-Only, com a única diferença que não efetua nenhuma operação, retornando apenas o resultado da chamada. |
| Parâmetros de input | ***type*:** ONE_CHANNEL;<br>***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *simple;*<br>▪ ***args:*** *{*<br>*{**fname**: donothing}*<br>*}*<br>***callback:*** *benchmark/fabric/null/donothing-null-workload.js* |

Tabela 21 - Teste T3Q - Quorum

| | |
|---|---|
| Nome do teste | Read Only – Quorum |
| ID | T3Q |
| Descrição | Este teste vai medir o Throughput e a Latência das plataformas, quando submetidas a operações nulas. É semelhante ao teste Write-Only, com a única diferença que não efetua nenhuma operação, retornando apenas o resultado da chamada. |
| Parâmetros de input | ***cmd*: *null;***<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***args:*** *{*<br>*{**money**: 100}*<br>*}*<br>***callback:*** *benchmark/quorum/null/open.js* |

Tabela 22 - Teste T4F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Modificação dos parâmetros do Orderer - Hyperledger Fabric |
| ID | T4F |
| Descrição | Este teste mede o impacto que a modificação dos parâmetros do Orderer tem no Throughput e a Latência da plataforma Hyperledger Fabric. |
| Parâmetros de input | O utilizador deve modificar os valores de:<br>▪ *BatchSize.MessageCount;*<br>▪ *BatchTimeout;*<br>▪ *BatchSize.PreferredMaxBytes;*<br>▪ *BatchSize.AbsoluteMaxBytes;*<br><br>Estes valores podem ser encontrados no ficheiro *configtx.yaml*, presente na diretoria da rede de testes. Após a modificação dos mesmos, o utilizador deve correr novamente o teste T1F. |

Tabela 23 - Teste T5F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Modificação da política de aprovação de uma transação - Hyperledger Fabric |
| ID | T5F |
| Descrição | Este teste mede de que forma a modificação da política de aprovação influencia o Throughput e a Latência da plataforma Hyperledger Fabric. |
| Parâmetros de input | O utilizador deve modificar os valores da política de aprovação (*Endorsment Policy*), presente no ficheiro *fabric.json*:<br>▪ ***signed-by:*** *0*;<br><br>Após a modificação destes valores, o utilizador deve correr novamente o teste T1F. |

### 6.1.2. Micro-Benchmarks

Os *Micro-Benchmarks* compreendem testes que modificam os parâmetros das transações e dos *chaincodes*, medindo o impacto que essas modificações têm na latência das transações. Deste conjunto de testes fazem parte:

- Testes de variação do tamanho da carga enviada para os *chaincodes* e *smart contracts;*
- Testes de uso de eventos nos *chaincodes;*
- Testes de variação dos níveis de chamadas dos *chaincodes* e *smart contracts;*

Para a execução dos testes acima descritos , a rede *Hyperledger Fabric* deve ser composta apenas por uma organização (com um *peer*) e um *solo Orderer.*

Tabela 24 - Teste T6F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Variação do tamanho da carga enviada para os chaincodes - Hyperledger Fabric |
| ID | T6F |
| Descrição | Este teste mede o efeito que o tamanho da carga passada como argumento a um chaincode tem no Throughput e Latência da plataforma Hyperledger Fabric. |
| Parâmetros de input | ***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *chaicode-payload;*<br>▪ ***args:*** *{*<br>*{**fname**: invokewithoutevents},*<br>*{**key:** accountno_},*<br>*{**payloadSize:** string* variável, a definir pelo utilizador*}*<br>*}*<br>***callback:*** *benchmark/fabric/chaicode-payload-size/open-payload-size.js* |

Tabela 25 - Teste T4Q - Quorum

| | |
|---|---|
| Nome do teste | Variação do tamanho da carga enviada para os smart contracts - Quorum |
| ID | T4Q |
| Descrição | Este teste mede o efeito que o tamanho da carga passada como argumento a um smart contract tem no Throughput e Latência da plataforma Quorum. |
| Parâmetros de input | ***cmd*:** *keyValStore;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***key:*** *inteiro de valor variável;*<br>***callback:*** *benchmark/quorum/key-val-store/open.js* |

Tabela 26 - Teste T7F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Uso de eventos nos chaincodes – Hyperledger Fabric |
| ID | T7F |
| Descrição | Este teste mede o efeito que o uso de eventos num chaincode tem no Throughput e Latência da plataforma Hyperledger Fabric. Para a execução deste teste, são utilizadas as mesmas cargas que em T6F, com a adição que após inserida a carga na blockchain, o chaincode retorna um evento. |
| Parâmetros de input | ***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *chaicode-payload;*<br>▪ ***args:*** *{*<br>*{**fname**: invokewithevents},*<br>*{**key:** accountno_},*<br>*{**payloadSize:** string* variável, a definir pelo utilizador*}*<br>*}*<br>***callback:*** *benchmark/fabric/chaicode-payload-size/open-payload-size.js* |

Tabela 27 - Teste T8F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Comparação de chamadas a chaincodes inter e intra-channel - Hyperledger Fabric |
| ID | T8F |
| Descrição | Este teste compara a performance da plataforma quando são realizadas invocações a chaincodes instanciados no mesmo canal e em canais diferentes, apresentando como comparação os resultados obtidos pela execução dos testes T8.1F e T8.2F. |

Tabela 28 - Teste T8.1F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Chamadas inter-channel a chaincodes - Hyperledger Fabric |
| ID | T8.1F |
| Descrição | Este teste compara a performance da plataforma Hyperledger Fabric quando são realizadas operações de leitura a chaincodes instanciados em canais diferentes. |
| Parâmetros de input | ***cmd*:** *query;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *caller_inter;*<br>▪ ***args:*** *{*<br>*{**fname**: query},*<br>*{**fname**: query},*<br>*{**key:** accountno_},*<br>*{**chaicodeid:** called_inter},*<br>*{**channel:** mychannel1}*<br>*}*<br>***callback:*** *benchmark/fabric/inter-channel-read/query-CC.js* |

Tabela 29 - Teste T8.2F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Chamadas intra-channel a chaincodes - Hyperledger Fabric |
| ID | T8.2F |
| Descrição | Este teste compara a performance da plataforma Hyperledger Fabric quando são realizadas operações de leitura a chaincodes instanciados no mesmo canal. |
| Parâmetros de input | ***cmd*:** *query;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *caller_inter;*<br>▪ ***args:*** *{*<br>*{**fname**: query},*<br>*{**fname**: query},*<br>*{**key:** accountno_},*<br>*{**chaicodeid:** called_inter}*<br>*}*<br>***callback:*** *benchmark/fabric/inter-channel-read/query-CC.js* |

Tabela 30 - Teste T9F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Operações de escrita sobre chaincodes instanciados no mesmo canal - Hyperledger Fabric |
| ID | T9F |
| Descrição | Este teste mede a performance da plataforma quando são realizadas operações de escrita sobre chaincodes instanciados no mesmo canal, com diversos níveis de profundidade de invocação. |
| Parâmetros de input | ***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *caller1;*<br>▪ ***args:*** *{*<br>*{**fname**: open},*<br>*{**fname**: open},*<br>*{**key:** accountno_},*<br>*{**caller2:** called_cc},*<br>*{**value:** 1000}*<br>*}*<br>***callback:*** *benchmark/fabric/inter-channel-write/open-intra-CC.js*<br><br>O utilizador deve usar o ficheiro de configuração *config-intra-CC-write-depth(i).json*, presente na diretoria do teste em causa, sendo que *i* toma o valor da profundidade que se pretende executar. Este aceita valores compreendidos entre 2 e 4. |

Tabela 31 - Teste T10F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Operações de escrita sobre o mesmo chaincode com niveis de profundidade de invocação variaveis - Hyperledger Fabric |
| ID | T10F |
| Descrição | Este teste mede a performance da plataforma quando sobre o mesmo chaincode são realizadas operações de escrita, com níveis de profundidade de invocação variaveis. |
| Parâmetros de input | ***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***chaicodeid:*** *nested-call-depth1;*<br>▪ ***args:*** *{*<br>*{**fname**: open},*<br>*{**key:** accountno_},*<br>*{**value:** 1000}*<br>*}*<br>***callback:*** *benchmark/fabric/single-chaicode-nested-calls/open.js*<br>O utilizador deve usar o ficheiro de configuração *config-nested-write-depth(i).json*, sendo que *i* toma o valor da profundidade que se pretende executar. Este aceita valores compreendidos entre 1 e 4. |

Tabela 32 - Teste T5Q - Quorum

| | |
|---|---|
| Nome do teste | Operações de escrita sobre o mesmo smart contract com niveis de profundidade de invocação variaveis - Quorum |
| ID | T5Q |
| Descrição | Este teste mede a performance da plataforma quando sobre o mesmo smart contract são realizadas operações de escrita, com níveis de profundidade de invocação variáveis. O smart contract deve ser atualizado de acordo com o número de níveis para as invocações desejadas. |
| Parâmetros de input | ***cmd*:** *nestedInvocation;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, em que *i* define o número total de transações a enviar para a plataforma e *j* o ritmo a que as transações são enviadas;<br>***arguments:***<br>▪ ***money:*** *inteiro de valor variavel;*<br>***callback:*** *benchmark/quorum/nested-invocation/open.js* |

### 6.1.3. Scalability Experiments

Das *Scalabilty Experiments*, cujo objetivo é medir a performance da plataforma ao nível da sua escalabilidade, fazem parte os seguintes testes:

- Teste de escalabilidade ao nível dos *chaincodes;*
- Teste de escalabilidade ao nível dos canais da rede;
- Teste de escalabilidade ao nível dos participantes da rede;

Tabela 33 - Teste T11F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Escalabilidade ao nível dos chaincodes - Hyperledger Fabric |
| ID | T11F |
| Descrição | Este teste mede a performance do Throughput e a Latência da plataforma Hyperledger Fabric, quando na sua rede são instanciados múltiplos chaincodes, e a todos é feita uma invocação em paralelo simultaneamente. |
| Parâmetros de input | Para o teste em causa, o *script* de instanciação de chaincodes, instância por defeito um total de dez chaincodes no mesmo canal.<br>***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos *[i, j]*, onde:<br>▪ ***i***: número total de chaincodes sobre o qual se pretendem enviar as transações definidas em *j*.<br>▪ ***j*:** número total de transações enviadas à blockchain;<br>***arguments:*** Os argumentos contêm uma lista dos chaincodes sob o qual se vão fazer as invocações. Caso um utilizador deseje adicionar mais chaincodes, deve fazê-lo adicionando os mesmos a essa lista.<br>***callback:*** *benchmark/fabric/chaicode-scalability/open-chaicode-scalability.js*<br><br>**Nota:** Caso o utilizador pretenda enviar 50 transações para 20 chaincodes, o array de ***txNumbAndTps*** deve ser igual a [20, 50] e assim sucessivamente. |

Tabela 34 - Teste T12F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Escalabilidade ao nível dos canais da rede - Hyperledger Fabric |
| ID | T12F |
| Descrição | Este teste mede a performance do Throughput e a Latência da plataforma Hyperledger Fabric, quando na sua rede são criados *n* canais, e em cada um é instanciado um chaincode. |
| Parâmetros de input | O teste garante que a carga distribuída de igual forma pelos canais.<br>***cmd*:** *open;*<br>***txNumbAndTps*:** array de tuplos [i, j], onde:<br>▪ ***i***: número total de canais sobre o qual se pretendem enviar as transações definidas em *j*.<br>▪ ***j*:** número total de transações enviadas à blockchain;<br>***arguments:*** Os argumentos contêm uma lista dos chaincodes sob o qual se vão fazer as invocações. Caso um utilizador deseje adicionar mais chaincodes, deve fazê-lo adicionando os mesmos a essa lista.<br>***callback:*** *benchmark/fabric/channel-scalability/open-channel-scalability.js* |

Tabela 35 - Teste T13F - Hyperledger Fabric

| | |
|---|---|
| Nome do teste | Escalabilidade ao nível dos participantes da rede - Hyperledger Fabric |
| ID | T13F |
| Descrição | Este teste mede a performance do Throughput e a Latência da plataforma Hyperledger Fabric, quando na sua rede é aumentado o número de organizações participantes. |
| Parâmetros de input | Para iniciar o teste em causa, o utilizador deve criar a rede com apenas uma organização (com um *peer node).*<br>O utilizador deve modificar os parâmetros da rede, como por exemplo, a adição ou remoção de participantes, ou a modificação da política de aprovação.<br>Após as modificações estarem concluídas, o utilizador deve reexecutar os testes T1F e comparar os resultados obtidos. |

Tabela 36 - Teste T6Q - Quorum

| | |
|---|---|
| Nome do teste | Escalabilidade ao nível dos participantes da rede - Quorum |
| ID | T6Q |
| Descrição | Este teste mede a performance do Throughput e a Latência da plataforma Quorum, quando na sua rede é aumentado o número de participantes. |
| Parâmetros de input | Para a execução do teste, o utilizador deve criar mais nós participantes e adicioná-los à rede *blockchain*. Após o processo de criação estar concluído, o utilizador deve modificar o ficheiro *"quorum.json"*, adicionando os novos participantes, e atualizando o campo *"targetParty"* dos restantes participantes com as chaves publicas dos participantes recém criados.<br>Após as modificações estarem concluídas, o utilizador deve reexecutar os testes T1Q e comparar os resultados obtidos. |

## 6.2 Interpretação do output gerado

Da execução dos testes anteriormente descritos, por cada ronda executada, são gerados na pasta do teste em causa, vários ficheiros *csv* com os resultados desta. Um excerto de um desses ficheiros pode ser observado na Figura 26.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | Name | Succ | Fail | Send_Rat | Max_Dela | Min_Dela | Avg_Dela | Throughp | Delay_c2e | Delay_e2c | Delay_o2 | T0_time | T1_time | Tn_time | ratio | Time Taken |
| 2 | 0 | open | 500 | 0 | 50 tps | 1.1700 s | 0.1290 s | 0.6562 s | 54 tps | 0.0048 s | 0.0033 s | 0.6482 s | 1.57E+09 | 1.57E+09 | 1.57E+09 | 1.125937 | 13.688 |

Figura 26 - Exemplo do ficheiro cvs de resultados de uma execução de um workload

De seguida segue uma explicação do seu significado.

- ***Succ*** - Número total de transacções submetidas com sucesso, apresentado como um número inteiro;
- ***Fail*** - Número total de transacções submetidas sem sucesso, apresentado como um número inteiro;
- ***Send Rate*** - Obtido pela divisão do valor das transacções bem sucedidas pela diferença entre o tempo de envio da última transacção submetida (*Tn*) e o tempo de envio da primeira transacção submetida (*T0*). O resultado é apresentado em transações por segundo (*TPS)*.

---

- ***Delay*** - O *delay* para cada transacção é obtido pela subtração do tempo da envio de uma transacção ao tempo de confirmação dessa mesma transacção. O resultado é apresentado em segundos.

Do valor de *delay* podemos destacar três categorias, nomeadamente:

***Delay_c2e*** - Como sendo o tempo passado entre o envio de uma transação e a recepção por parte de um *endorser peer*;

***Delay_e2o*** - Representando o tempo de envio de uma transação de um *endorser peer* para o *ordering service*;

***Delay_o2v*** - Tempo necessário para as transações serem ordenadas, agrupadas em blocos e confirmadas pelo *peer* que as submeteu na rede;

- ***Throughput*** - Obtido pela divisão do número de transações bem-sucedidas pela diferença entre o tempo de confirmação da última transação submetida (*Tn*) e o tempo de confirmação da primeira transação submetida (*T0*). O resultado é apresentado em transações por segundo.

---

No próximo capítulo procede-se à execução dos testes definidos, e à respetiva apresentação e análise dos resultados da sua execução.

# Capítulo 7
# Execução e resultados dos testes de benchmark

Neste Capítulo são apresentados os resultados da execução dos testes definidos no Capítulo 6. O ambiente de execução compreende um total de três máquinas virtuais, localizadas nas infraestruturas do Instituto Pedro Nunes (IPN), cada uma equipada com 100 GB de memória em disco, 8 GB de memória RAM e 4vCPU's, correndo o sistema operativo Debian 9, e 8 máquinas virtuais na plataforma *Cloud* do Departamento de Engenharia Informática (DEI), com as mesmas especificações, correndo o sistema operativo Ubuntu 16.04. A execução dos testes à plataforma *Hyperledger Fabric* foi distribuída pelas máquinas do IPN e a execução dos testes à plataforma *Quorum* distribuída pelas máquinas do DEI.

Com o objetivo de eliminar *outilers* que, de alguma forma, pudessem comprometer os resultados obtidos, cada teste foi executado num total de dez corridas. A escolha do número total de execuções foi limitada pela duração temporal de cada teste. De igual forma, antes de iniciar cada teste, a plataforma passou por um período de "aquecimento", de um total de cinco rondas com 500 transações enviadas para a plataforma a um ritmo de 50 transações por segundo. As rondas de aquecimento foram incorporadas no processo de execução dos testes, de forma a permitir que a plataforma atingisse um estado estável, eliminando possíveis desvios nos resultados gerados. Os resultados das rondas de aquecimento não são, no entanto, tidos em conta nos resultados finais. Para todos os resultados finais obtidos, foi calculado um intervalo de confiança através função do *Microsoft Excel "CONFIDENCE.T"*, com o valor de alfa igual a 0.05. Esse intervalo de confiança é representado nos gráficos pelas barras de erro.

Da configuração base da rede para a execução dos testes, fazem parte um *Solo Orderer*, duas organizações (*Org1*) e (*Org2*) com dois *peers* cada uma, a comunicarem através de um canal único, e *LevelDB* para a persistência de dados. No entanto, sempre que seja necessário modificar a configuração da rede para executar determinado teste, essa nova configuração é detalhada na secção de resultados do teste em causa.

Os testes foram executados com recurso ao *script* de automação desenvolvido, presente na diretoria *Scripts*. Após modificar o ficheiro de configuração, presente na mesma diretoria, de acordo com as imagens presentes no início de cada uma das secções seguintes, o utilizador deve executar o seguinte comando: *python run-benchmark.py*.

Durante o processo de adaptação do código base da plataforma *Gauge* para a execução dos testes à plataforma *Quorum*, o aluno deparou-se com diversas dificuldades que impossibilitaram que as execuções dos testes prosseguissem. Dessas dificuldades destacam-se a impossibilidade de manter a rede *Quorum* estável e a comunicar entre si sem problemas, e a impossibilidade de terminar o processo mencionado na *Modificação 3* da secção 5.2.2. Como consequência, os resultados para a plataforma *Quorum* não são apresentados neste documento. De igual forma, por o aluno não ter conseguido executar o teste T13F, este também não consta do capítulo de apresentação de resultados.

## 7.1 Teste T1F

O teste T1F foi desenhado com o objetivo de medir os valores máximos de *Throughput* e Latência da plataforma quando esta é submetida a operações de escrita sobre a *blockchain*. Este faz uso do *chaincode Simple*, que por sua vez insere os dados passados por argumento na *blockchain*. Com o objetivo de executar o teste sobre uma *blockchain* sem dados, antes de se dar início a uma nova corrida, a rede *blockchain* foi destruída e gerada uma nova.

De forma a atingir um tempo de execução para cada ronda de pelo menos 10 segundos, o intervalo de amostras para os resultados recolhidos foi divido em quatro partes. Da primeira parte, compreendem as rondas de "aquecimento". A segunda parte, é composta pelo intervalo de 0 a 4000 transações, com intervalos de 500 transações. A terceira parte, composta pelo intervalo de 4000 a 6000 transações, com intervalos de 100 transações. A quarta e final parte, composta pelo intervalo de transações de 6000 até 29000, com intervalos de 1000 transações. No entanto, o tempo de execução para cada ronda revelou-se ser muito mais demorado que os 10 segundos inicialmente planeados.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 27. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "write",
  "limit": 3000,
  "executions": 10,
  "type": "controlled",
  "n_chaincodes": 1
}
```

Figura 27 - Excerto do ficheiro de configuração para a execução do teste T1F

Seguindo a estrutura desta secção, começamos pela apresentação da comparação das grandezas de *Send Rate* e *Throughput* na Figura 28. O *Send Rate* corresponde ao número de transações por segundo enviadas pela plataforma *Gauge* à plataforma *Hyperledger Fabric*, e o *Throughput* corresponde ao número de transações por segundo que a plataforma *Hyperledger Fabric* processa. Esta comparação surgiu da necessidade de perceber se os valores apresentados de *Throughput* estariam ou não limitados pelos valores de *Send Rate*, uma vez que a plataforma *Hyperledger Fabric* não teria hipóteses de processar as transações caso não as recebesse.

Na Figura 28, é possível observar que os valores de *Send Rate* são limitados pela plataforma *Gauge*, onde, após atingir o seu valor máximo de 366 *TPS* é visível uma ligeira descida dos mesmos. Foi retirada esta conclusão pois, uma vez comparados os valores gerados de *Send Rate* com os valores de configuração para envio à plataforma *Hyperledger Fabric*, estes não coincidiam. O *Throughput* atinge o seu valor máximo de aproximadamente 355 *TPS* no mesmo intervalo de transações que o *Send Rate*, entre as 3000 e 4000 transações. Após atingir esse valor, o *Throughput* decresce, aproximando-se do seu valor médio de aproximadamente 259 *TPS*.

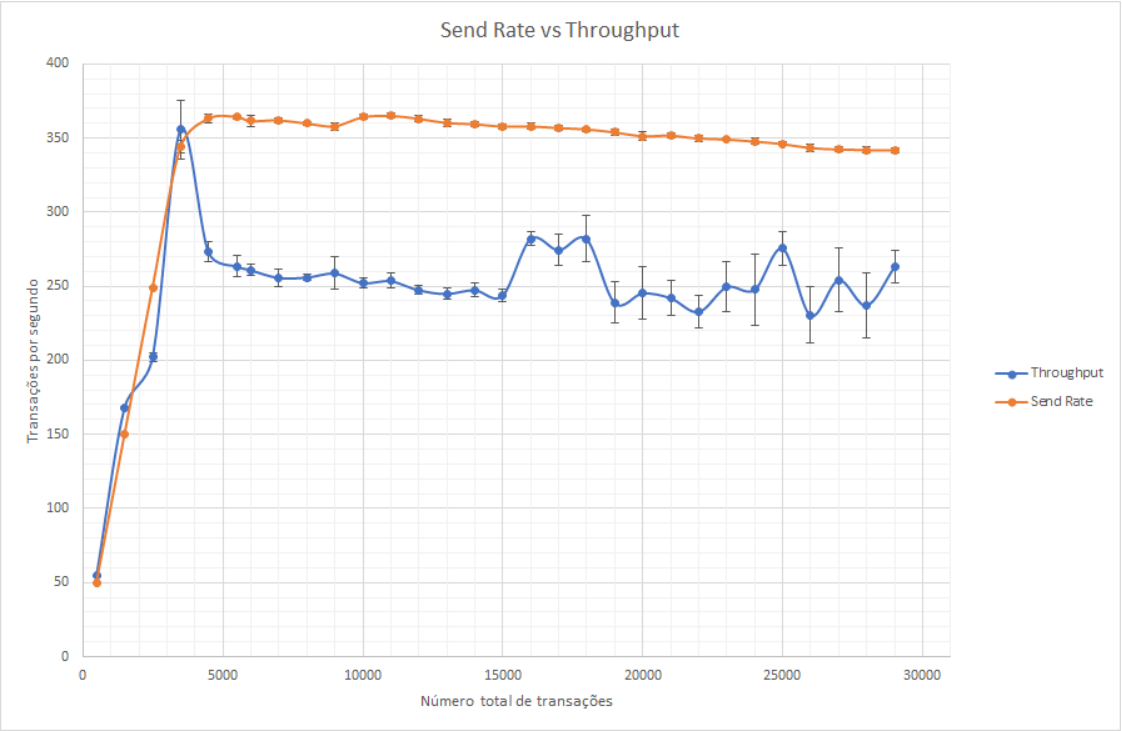


Figura 28 - Send Rate vs Throughput – T1F

Na Figura acima apresentada, podemos observar a diferença nos intervalos de confiança para cada grandeza. Enquanto que, o *Send Rate* apresenta valores para os intervalos de confiança pequenos, os valores para intervalos de confiança do *Throughput* são um pouco maiores, principalmente nos intervalos de entre as 3000 e 5000 transações e a partir das 15000 transações até ao final. Neste último intervalo, o *Throughput* apresenta um comportamento variável, com os seus valores a oscilarem entre as 230 e 290 *TPS*.

De forma a ser possível observar com mais clareza o intervalo de transações onde as duas grandezas atingem os seus valores máximos, foi realizado um *zoom* à Figura 28, considerando-se para tal o intervalo de 0 a 6000 transações. Esse *zoom* pode ser observado na Figura 29.

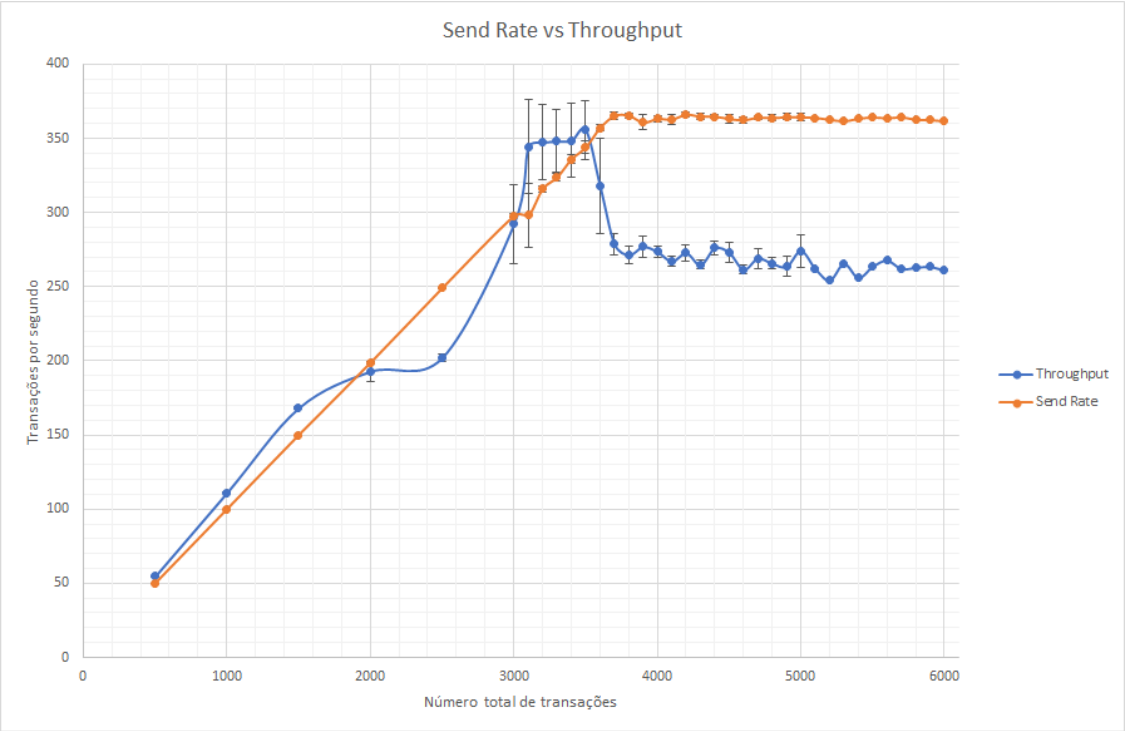


Figura 29 - Send Rate vs Throughput – T1F - Intervalo de 0 a 6000 transações

De uma análise mais cuidada à Figura 29, identificamos três pontos de interesse. O primeiro apresenta-se nos intervalos de 0 a 1500 transações e o segundo no intervalo de 3000 a 3500 transações. Em ambos os pontos de interesse, os valores de *Throughput* são superiores aos valores de *Send Rate*, não tendo sido encontrada uma justificação para tal comportamento. O terceiro ponto de interesse surge no intervalo de 3500 transações, onde é observável uma descida abrupta nos valores de *Throughput*.

A Figura 30 compara as grandezas de *Throughput* e Transações Bem-Sucedidas. Considera-se bem-sucedida, uma transação que é enviada para a plataforma *Hyperledger Fabric* e posteriormente processada por esta. A necessidade desta comparação surgiu como forma para identificar a origem da descida abrupta dos valores de *Throughput*, abordado no paragrafo anterior.

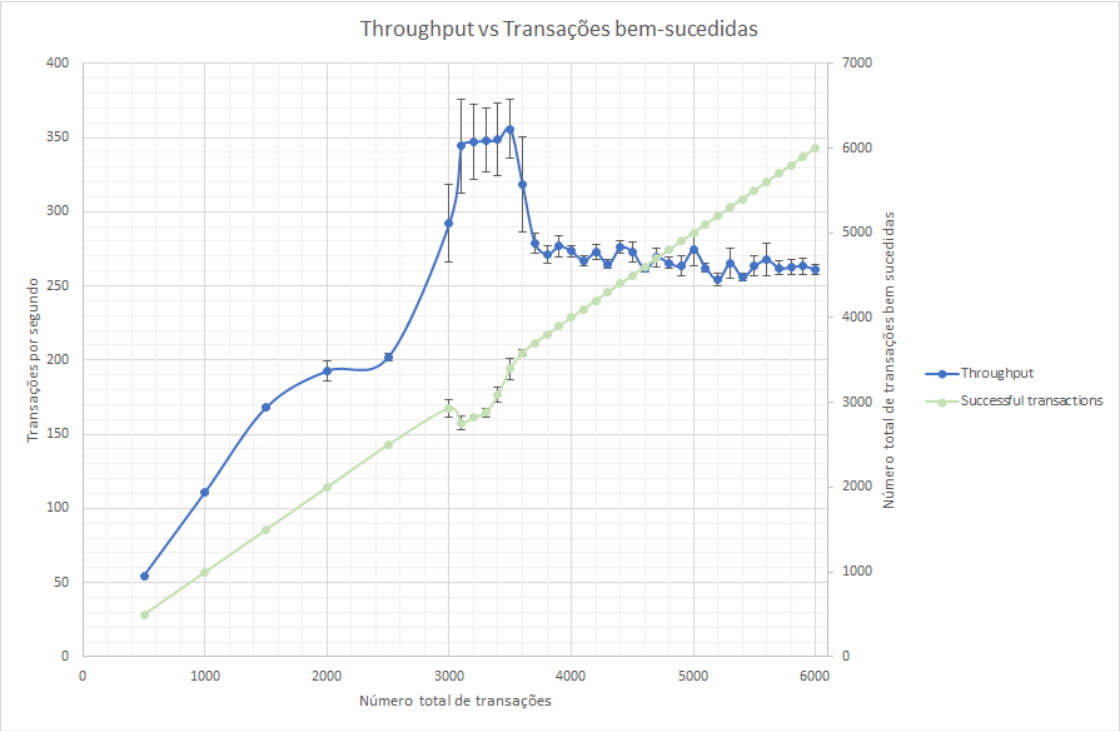


Figura 30 - Throughput vs Transações bem-sucedidas – T1F - Intervalo de 0 a 6000 transações

Na Figura 30, observamos que no intervalo entre as 3000 e 3500 transações, o número total de transações bem-sucedidas deixa de corresponder à totalidade de transações enviadas pela plataforma *Gauge*. Desta observação podemos concluir que, ao não serem enviadas todas as transações para a plataforma *Hyperledger Fabric*, esta tem uma maior facilidade em processar as transações enviadas, apresentando por isso maiores valores de *Throughput*. Esses valores descem abruptamente quando, no intervalo a partir das 3500 transações, o número de transações bem-sucedidas estabiliza e volta a corresponder à totalidade de transações enviadas.

Na Figura 31 e Figura 32 são apresentados os valores da Latência. A primeira figura apresenta o intervalo completo de amostras, e a segunda apresenta o intervalo utilizado anteriormente das 0 às 6000 transações.

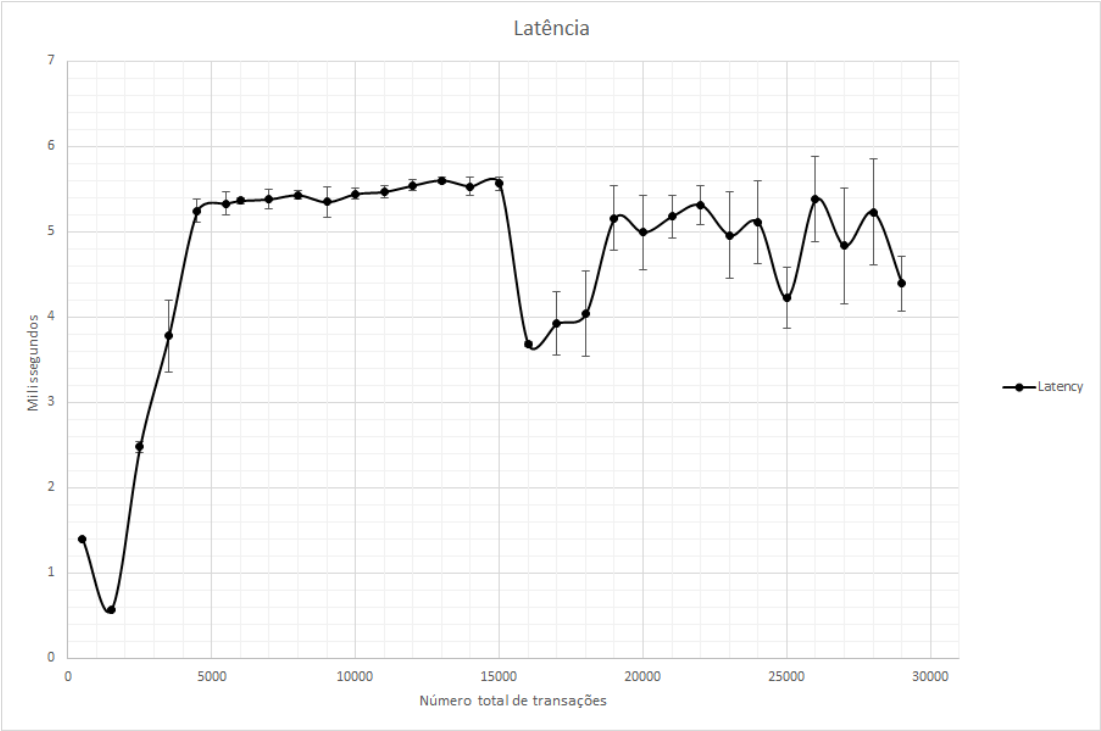


Figura 31 - Latência – T1F

Da análise à Figura 31, é possível destacar três momentos. O primeiro momento surge no intervalo inicial das 0 as 1500 transações, onde é observável um decréscimo nos valores de Latência, apresentando assim um comportamento contrário ao esperado. Uma possível explicação para isso, pode residir no facto de a plataforma ainda ter por processar algumas das transações das rondas de "aquecimento". O segundo momento compreende o intervalo entre as 3000 e 3500 transações. Este intervalo corresponde ao intervalo referido na Figura 30, onde o número de transações bem-sucedidas reduz. Esse momento pode ser observado em detalhe na Figura 32. O último momento encontra-se entre o intervalo de 15000 a 20000 transações, onde a Latência sofre uma descida abrupta após atingir o seu valor máximo de aproximadamente 6 milissegundos. Do último momento referido, até ao final, os seus valores oscilam entre os 4 e 5.5 milissegundos, e apresentam intervalos de confiança maiores. Não foi, no entanto, encontrada uma justificação para este comportamento.

A Figura 32 apresenta o *zoom* no intervalo de amostras de 0 a 6000 transações e reforça a ideia apresentada no paragrafo anterior, relativamente ao segundo momento referido.

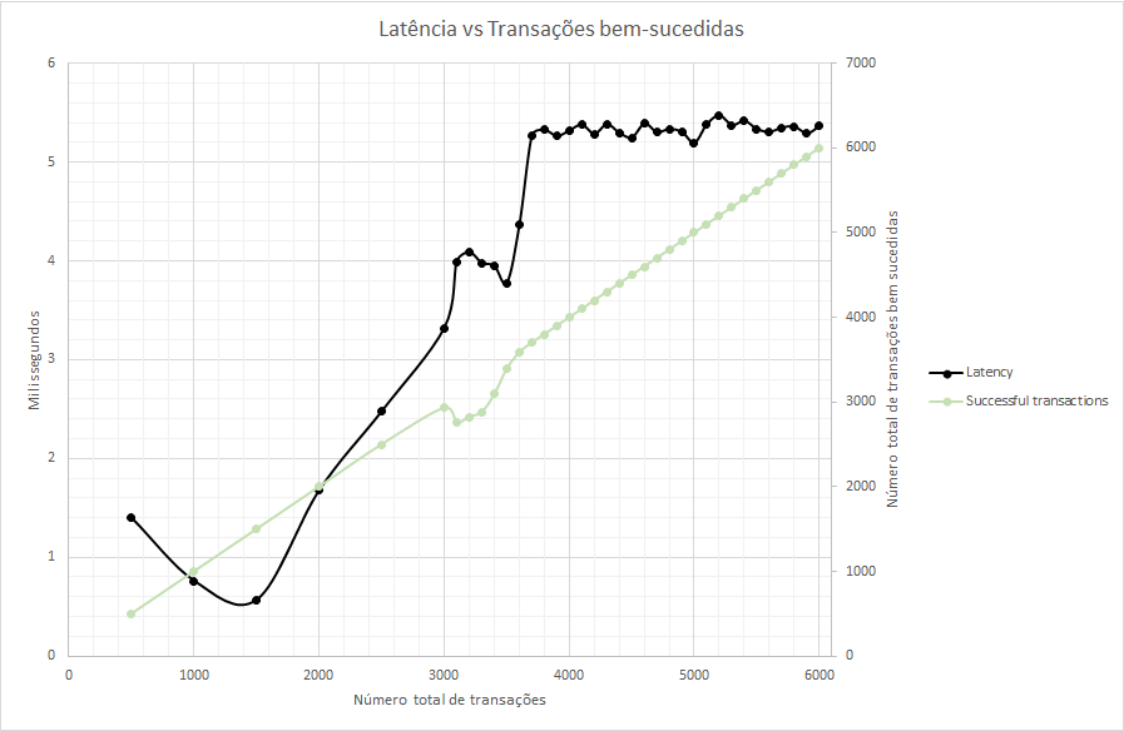


Figura 32 - Latência vs Transações bem-sucedidas – T1F - Intervalo de 0 a 6000 transações

A Figura 33 apresenta a comparação entre o consumo de memória RAM e o consumo de CPU de um *peer* da rede. Desta Figura é possível concluir que só a partir das 10000 transações é que as duas grandezas em comparação estabilizam, ainda que o consumo de CPU apresente valores bastante variáveis. Este comportamento contraria o que acontece quando comparamos estas grandezas com o *Throughput* e Latência, onde os seus valores começam as estabilizar a partir das 4000 transações. Esta comparação permitiu-nos concluir que as duas grandezas de consumo de memória RAM e consumo de CPU apresentam um comportamento normal, uma vez que à medida que o número de transações recebidas aumenta, os seus valores também aumentam como expectável.

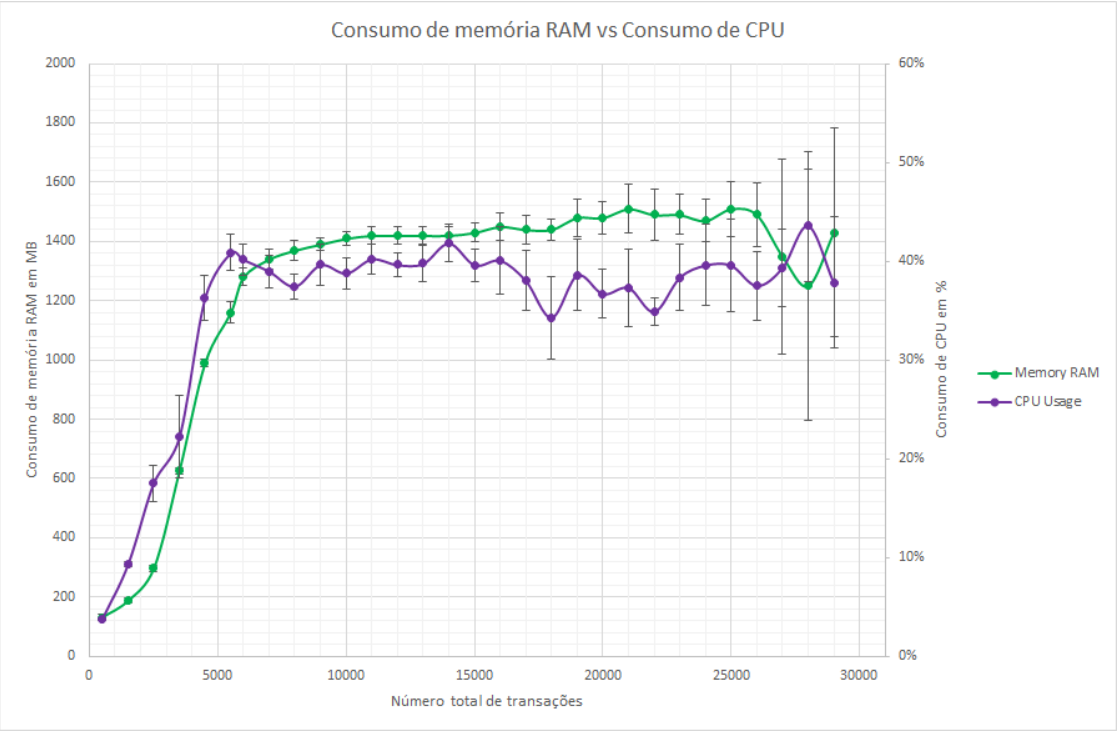


Figura 33 - Consumo de memória RAM vs Consumo de CPU – T1F

A Figura 84 apresenta um *zoom* à Figura 33, considerando o intervalo de 0 a 6000 transações. Nesta é possível reforçar que o consumo de memória RAM e o consumo de CPU sofrem um crescimento abrupto, no mesmo intervalo observado na Figura 30, onde o número de transações bem-sucedidas não corresponde à totalidade de transações enviadas.

Em suma, os resultados acima apresentados permitem-nos concluir que a plataforma *Hyperledger Fabric* apresenta um valor máximo de 355 *TPS* para o *Throughput*. Após atingir o seu valor máximo, o *Throughput* desce para os valores médios de 259 *TPS*. A Latência da plataforma apresenta um valor máximo de 6 milissegundos e uma média de 4 milissegundos. O intervalo entre as 3400 e 4000 transações revelou-se bastante importante, uma vez que é nesse intervalo que observamos um crescimento significativo das grandezas de *Throughput*, Latência e consumo de CPU. Nesse mesmo intervalo, o número de transações bem-sucedidas não corresponde à totalidade de transações enviadas, o que por sua vez influencia o crescimento das grandezas anteriormente referidas.

## 7.2 Teste T2F

O teste T2F foi desenhado com o objetivo de medir os valores máximos de *Throughput* e Latência da plataforma quando esta é submetida a operações de leitura sobre a *blockchain*. Este faz uso do *chaincode Simple*, que por sua vez lê e retorna os dados alocados na *blockchain*. Contrariamente ao que aconteceu durante a execução do T1F, para este teste não foi

necessária a destruição e criação de novas redes entre as execuções. O teste T2F seguiu o mesmo intervalo de amostras que o apresentado no T1F, e, de igual forma, seguirá a mesma ordem de apresentação de resultados.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 34. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "read",
  "limit": 3000,
  "executions": 10,
  "type": "controlled",
  "n_chaincodes": 1
}
```

Figura 34 - Excerto do ficheiro de configuração para a execução do teste T2F

A Figura 35 apresenta a comparação entre *Send Rate* e *Throughput*. Para esta comparação foi necessária a representação com dois eixos verticais, de forma a tornar mais simples a visualização dos dados. Desta Figura, podemos observar que o comportamento do *Send Rate* é idêntico ao comportamento apresentado na secção 7.1. Contrariamente, o *Throughput* apresenta valores muito superiores, com um máximo de aproximadamente 3070 *TPS* e uma média de aproximadamente 2072 *TPS*. É também possível destacar o comportamento anómalo do *Throughput* no intervalo entre as 15000 e 16000 transações, onde os valores deste sofrem uma queda abrupta antes de tornarem a crescer novamente. Para esse comportamento não foi encontrada uma justificação.

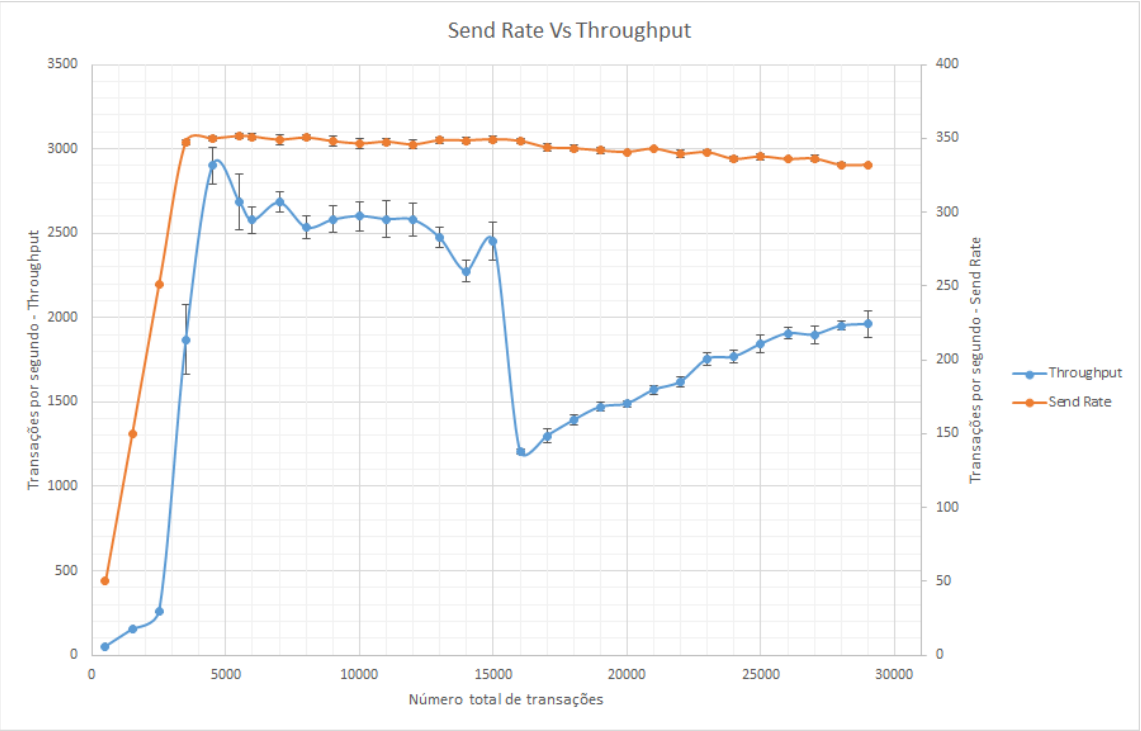


Figura 35 - Send Rate vs Throughput – T2F

Foi também feito um *zoom* ao gráfico da Figura 35, utilizando para tal o intervalo entre 0 e 6000 transações, permitindo dessa forma uma melhor observação dos resultados. O gráfico resultante pode ser observado na Figura 85.

É possível justificar os elevados valores de *Througput* gerados pelo T2F quando comparados com os valores gerados por T1F quando recordamos o modelo implementado pela plataforma *Hyperledger Fabric*, apresentado na secção 4.4. Enquanto que para operações de escrita na *blockchain*, uma transação deve passar pelas três fases que a plataforma implementa, o mesmo não se verifica para operações de leitura sobre a blockchain, sendo que estas são quase imediatas.

Para o teste T2F não foi necessária a análise e comparação do *Throughput* com o número total de transações bem-sucedidas, uma vez que estas corresponderam sempre à totalidade de transações enviadas.

Os valores de Latência são apresentados na Figura 36. À semelhança do que foi observado na Figura 85, também na Figura 36 reparamos que é no intervalo entre as 3500 e 4000 transações, que os valores de Latência apresentam um crescimento repentino. O seu comportamento corresponde ao esperado, uma vez que os seus valores mantêm um crescimento regular, atingindo o seu valor máximo de aproximadamente 2.4 milissegundos, perto do maior número total de transações (28000 transações). Já o valor médio de Latência é de aproximadamente 1.9 milissegundos.

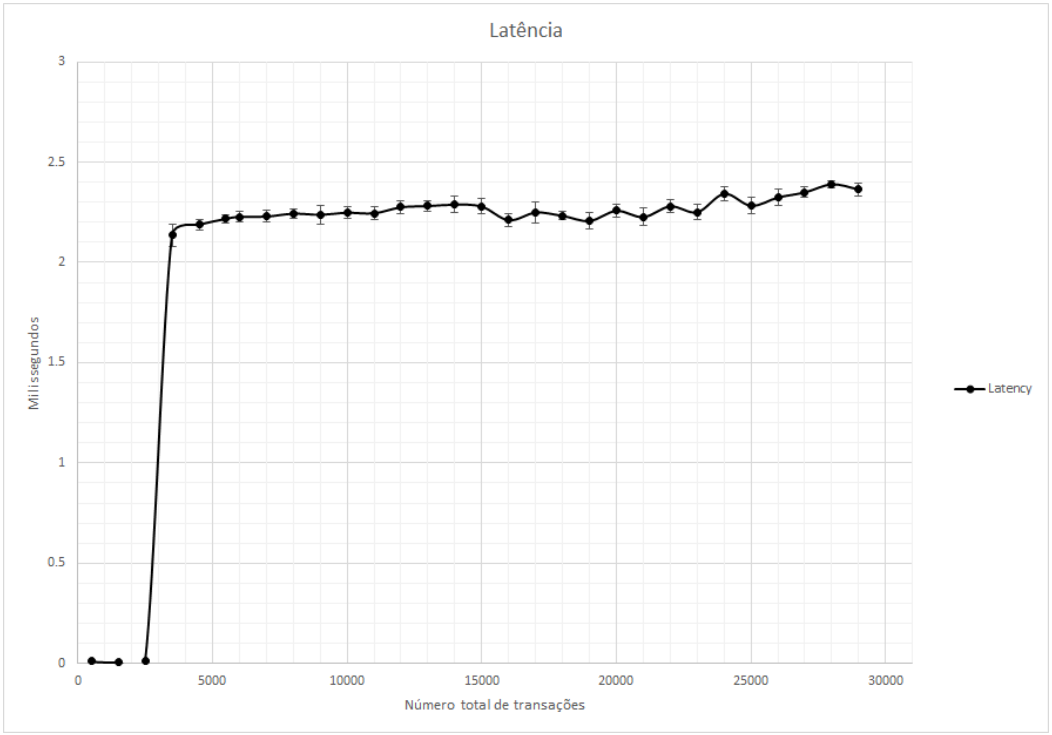


Figura 36 - Latência – T2F

Na Figura 37, observamos a comparação entre os valores de consumo de memória RAM e consumo de CPU. De destacar o facto de os valores iniciais do consumo de memória RAM descerem dos 700MB até aos 400MB. Uma possível justificação para este comportamento reside na arquitetura implementada pela plataforma *Hyperledger Fabric*, apresentada na secção 4.4, com especial atenção para a Figura 16. Nessa Figura observamos a presença do componente *World State*, cuja função é guardar a *cache* dos estados dos *ledgers* da *blockchain*. Essa decisão de implementação permite que o acesso ao estado do *ledger* seja imediato sem que o utilizador que pretende consultar a *blockchain* tenha que percorrer o *log* de transações inteiro (The Linux Foundation, 2019i). Dos valores apresentados para os intervalos de confiança para o consumo de memória RAM, é apenas possível concluir que estes apresentam uma grande variação. O consumo de CPU apresenta novamente um comportamento expectável, onde é observável que o seu crescimento acompanha o aumento das transações enviadas para a plataforma.

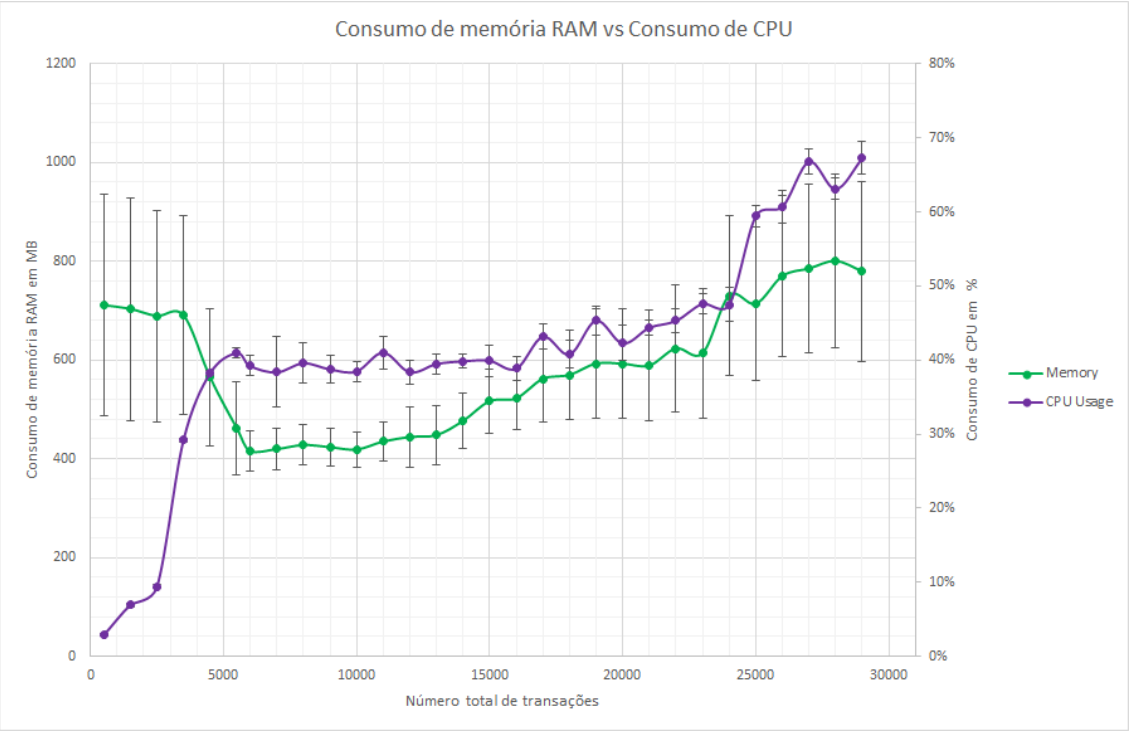


Figura 37 - Consumo de memória RAM vs Consumo de CPU – T2F

Em suma, concluímos que a plataforma *Hyperledger Fabric* apresenta uma melhor performance quando sobre ela são realizadas operações de leitura, com valores médios de aproximadamente 2072 *TPS* para o *Throughput* e 1.9 milissegundos para a Latência. É importante notar que, de igual forma ao que foi observado na secção 7.1, nos resultados gerados pelo teste T2F foi possível destacar o intervalo de 3400 a 4000 como sendo um intervalo de interesse, onde observamos novamente um crescimento significativo para as grandezas de *Throughput*, Latência e consumo de CPU.

## 7.3 Teste T3F

O teste T3F foi desenhado com o objetivo de medir os valores máximos de *Throughput* e Latência da plataforma quando esta é submetida a operações nulas sob a *blockchain*. Na sua definição, este teste comporta-se de igual forma ao - Teste T1F - Hyperledger Fabric, com a diferença de não desempenhar nenhuma operação sobre a *blockchain*, retornando apenas o resultado da chamada da função. Dessa forma, o teste foi executado da mesma maneira que o teste T1F, onde entre execuções é destruída e criada uma nova rede *blockchain* e, igualmente, seguiu o mesmo intervalo de amostras. Para facilitar a visualização e interpretação dos resultados, estes vão ser apresentados em comparação com os resultados obtidos por T1F.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 38. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "null",
  "limit": 3000,
  "executions": 10,
  "type": "controlled",
  "n_chaincodes": 1
}
```

Figura 38 - Excerto do ficheiro de configuração para a execução do teste T3F

Na Figura 39 observamos a comparação dos valores de *Throughput*. Dessa, concluímos que o comportamento entre os mesmos é idêntico, no entanto, com T3F a ter um ligeiro melhor desempenho, apresentando assim um valor médio de *Throughput* de aproximadamente 278 *TPS*, superior às 259 *TPS* apresentadas por T1F. Consequentemente, o valor máximo é também superior, de aproximadamente 369 *TPS*.

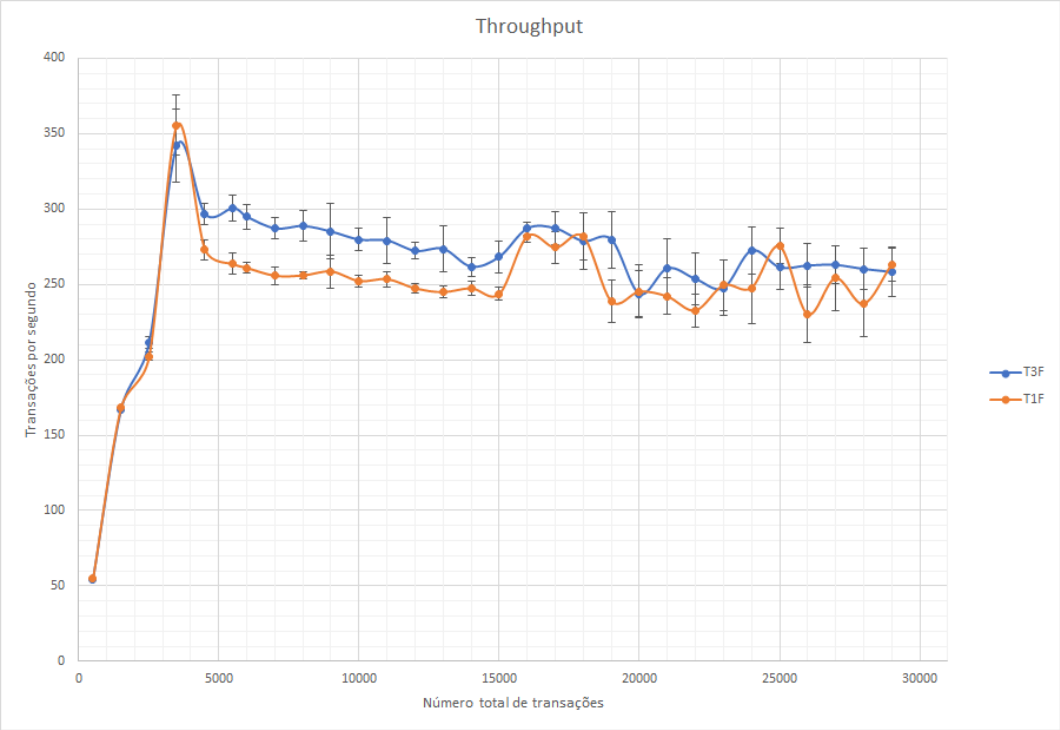


Figura 39 - Comparação dos valores de Throughput entre os testes T1F e T3F

Na Figura 40 são apresentados os valores da Latência. Novamente, e como observado anteriormente na Figura 39, o comportamento da Latência entre os testes T1F e T3F é muito idêntico, com, no entanto, o teste T3F a ter um melhor desempenho. Para o teste T3F, a Latência atinge valores médios de aproximadamente 4.4 milissegundos, inferior aos 5 milissegundos apresentados por T1F.

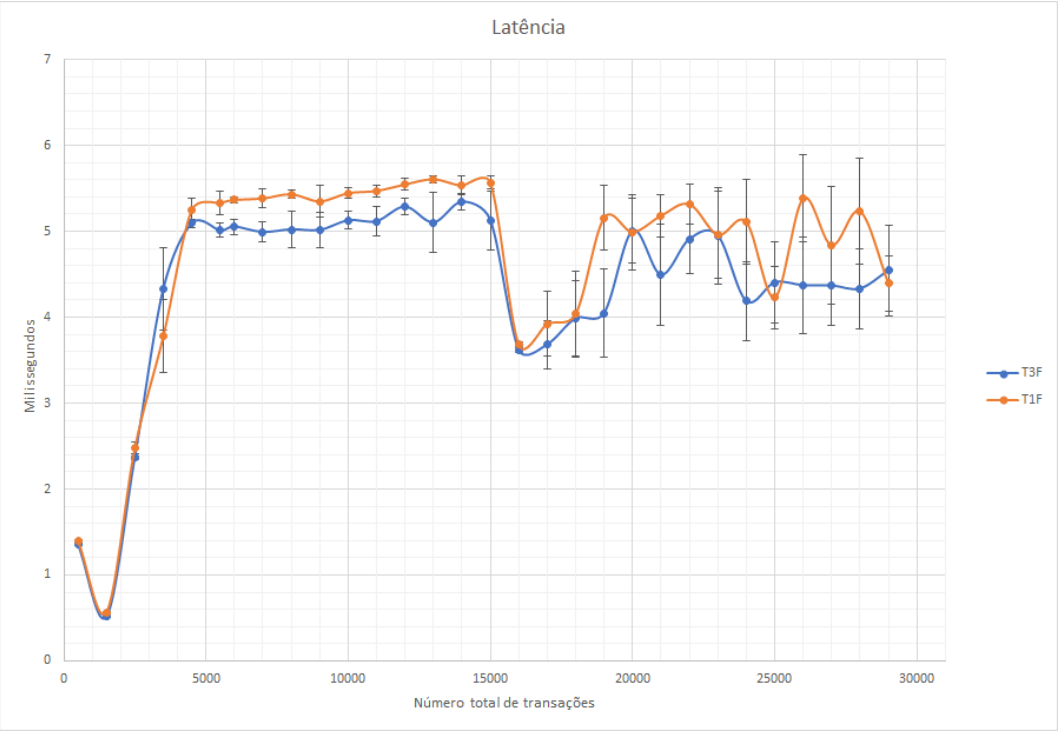


Figura 40 - Comparação dos valores de Latência entre os testes T1F e T3F

Na Figura 41 e Figura 42 são apresentadas as comparações entre o consumo de CPU e o consumo de memória RAM, respetivamente. Continuando o comportamento observado nas Figuras anteriores, nomeadamente na Figura 39 e Figura 40, o teste T3F continua com um melhor desempenho. O desempenho no consumo de CPU, ainda que leve, destaca-se mais a partir das 15000 transações, onde os valores para este decrescem no T3F, contráriamente ao que acontece no T1F. Uma vez mais, os valores para os intervalos de confiança apresentados para o intervalo entre as 15000 transações até ao final, permitem-nos apenas concluir que os valores de consumo de CPU apresentam uma variação de valores que segue uma média de 3%.

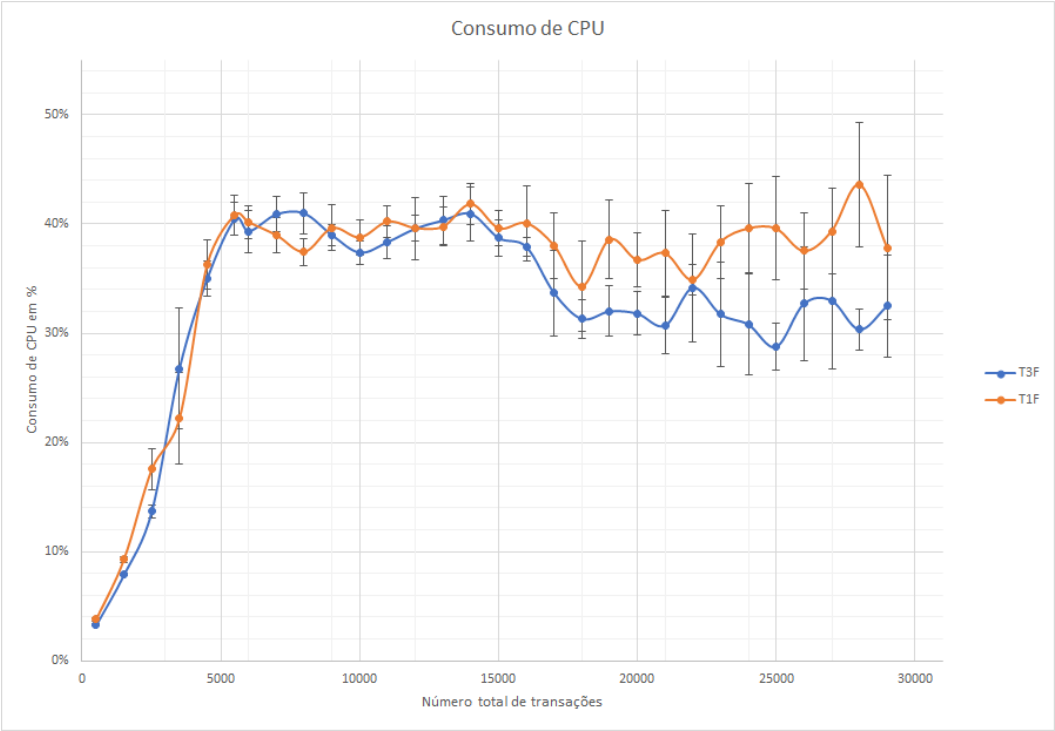


Figura 41 - Comparação dos valores do consumo de CPU entre os testes T1F e T3F

Da Figura 42, à exceção do intervalo entre as 25000 e 29000 transações, onde o T1F apresenta um menor consumo de memória RAM que o T3F, não há mais observações que mereçam destaque, uma vez que desta comparação, observamos que os dois testes apresentam um desempenho semelhante. Assim, é possível concluir que de acordo com os testes realizados, o tipo de operações que se realiza sobre a *blockchain* não afeta muito o consumo de memória RAM.

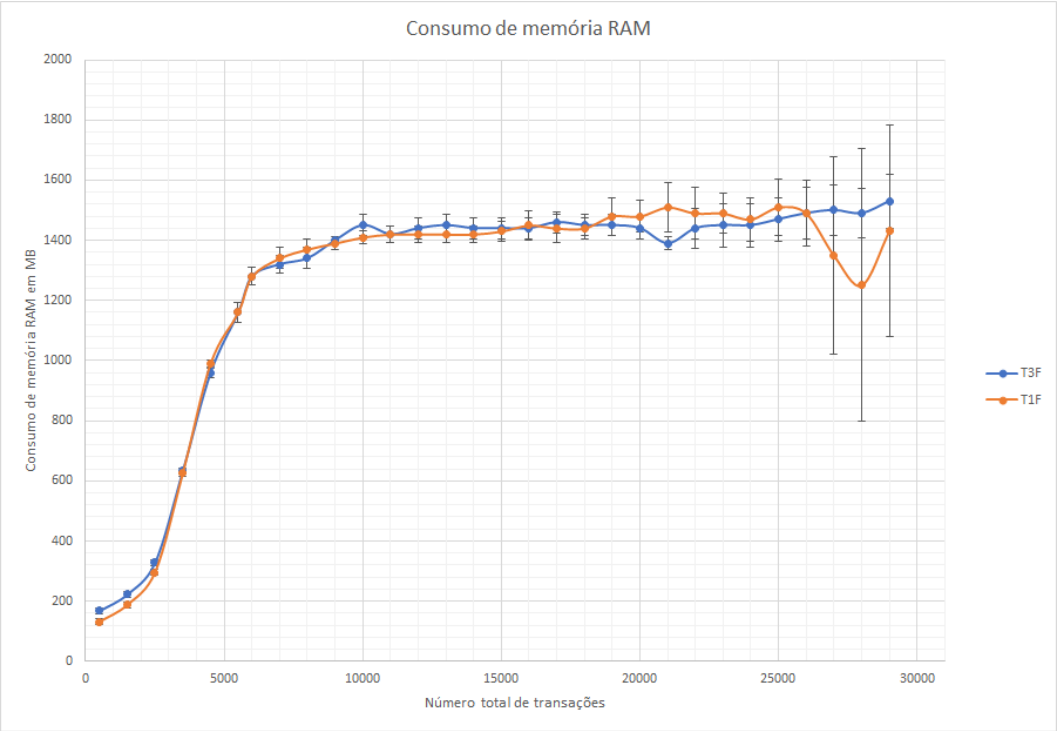


Figura 42 - Comparação dos valores do consumo de memória RAM entre os testes T1F e T3F

Em suma, quando comparamos o desempenho entre os testes T1F e T3F, notamos que no geral o teste T3F apresenta um melhor desempenho que o teste T1F. Este comportamento é, uma vez mais, expectável uma vez que no teste T3F não estão a ser executadas operações sobre a *blockchain*, no entanto, observamos que, mesmo não estando a executar operações, a diferença entre os valores de T1F e T3F não é muito significativa.

## 7.4 Teste T4F

O teste T4F foi desenhado com o objetivo de medir o impacto que a modificação dos valores do *Orderer* têm nos valores máximos de *Throughput* e Latência da plataforma. O teste seguiu o modelo de execução do teste T1F, onde entre execuções é destruída e criada uma nova rede *blockchain* e, utilizando de igual forma o mesmo intervalo de amostras. A realização do teste T4F foi motivada pela afirmação disponível em (Androulaki et al., 2018), onde é dito que:

> *"Grouping or batching transactions into blocks improves the throughput of the broadcast protocol, which is a well-known technique in the context of fault-tolerant broadcasts."*

Para a execução deste teste foram usadas quatro distintas configurações dos parâmetros do *Orderer*. Esses parâmetros devem ser modificados no ficheiro *"configtx.yaml"*, presente na diretoria *"network/fabric-v1.4/simplenetwork"*. Assim, do ficheiro *"configtx.yaml"* foi modificado o parâmetro *MaxMessageCount*. Esse parâmetro define o número de mensagens que são permitidas por *batch* ou bloco. Os valores utilizados para esse parâmetro são os seguintes:

- *MaxMessageCount: 10;*
- *MaxMessageCount: 250;*
- *MaxMessageCount: 500;*
- *MaxMessageCount: 1000;*

A escolha para estes valores seguiu a seguinte ordem. O primeiro valor considerado, 10, foi adaptado do ficheiro providenciado pela plataforma *Hyperledger Caliper*, em (The Linux Foundation, 2019e). O valor 250, foi utilizado para servir de intermediário entre o primeiro valor de 10 e o valor 500, valor que vinha por *default* com a plataforma *Gauge*. O último valor, 1000, foi utilizado como um valor máximo para esse parâmetro, visto ser o dobro do anterior.

Os restantes parâmetros do *Orderer* mantêm-se iguais para todas as configurações, e são os seguintes:

- *OrdererType: solo* – parâmetro que define o tipo de *Orderer*;
- *AbsoluteMaxBytes: 999 MB* – o número máximo absoluto de *bytes* que é permitido por *batch*;
- *PreferedMaxBytes: 20 MB* – o número pretendido de *bytes* por *batch*;

Após a modificação dos parâmetros anteriormente referidos, o utilizador deve executar o *script "generate-crypto-materials.sh"* na mesma diretoria, de forma a gerar os artefactos criptográficos da rede, como por exemplo os certificados de autenticidade, chaves privadas dos utilizadores e os respetivos canais da rede.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 43. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "write",
  "limit": 3000,
  "executions": 10,
  "type": "controlled",
  "n_chaincodes": 1
}
```

Figura 43 - Excerto do ficheiro de configuração para a execução do teste T4F

Na Figura 44 observamos a comparação dos valores de *Throughput* para as diversas configurações do *Orderer*. Podemos concordar com a afirmação exposta no início da secção, uma vez que observamos que a configuração de apenas 10 mensagens por *batch* apresenta os piores valores de *Throughput*, com um valor máximo de apenas 77 *TPS* e um valor médio de apenas 69 *TPS*. Ainda que o maior valor médio de *Throughput* pertença à configuração de *Message Count* = 1000, com aproximadamente 264 *TPS*, o valor máximo para o *Throughput* pertence à configuração de *Message Count* = 500, com aproximadamente 370 *TPS*.

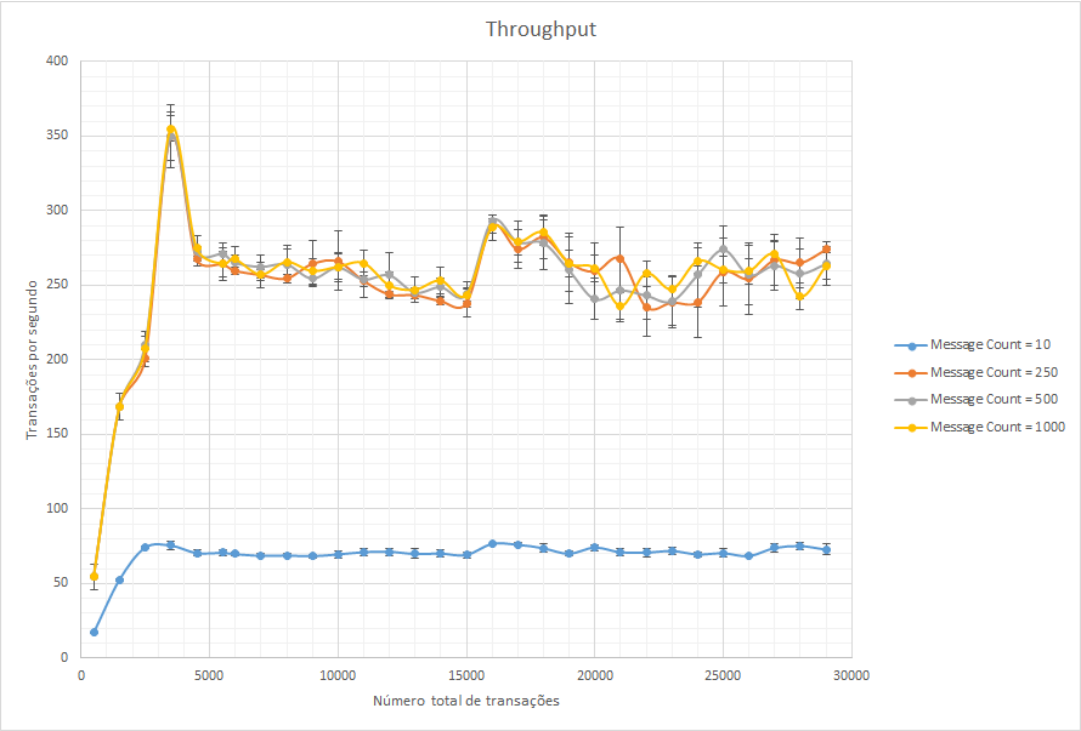


Figura 44 - Comparação dos valores de Throughput – T4F

Na Figura 86, encontramos o *zoom* gráfico anterior, no mesmo intervalo utilizado nas secções anteriores. Desta Figura, podemos observamos novamente o intervalo de destaque referenciado nas secções 7.1 e 7.3, onde é observável o pico nos valos de *Throughput* seguido pela descida dos mesmos. Reparamos ainda com mais detalhe que os valores de *Throughput* para as configurações de *Message Count* = 250, 500 e 1000 são muito idênticos.

Na Figura 45, observamos os valores de Latência para as diversas configurações do *Orderer*. Uma vez mais, reparamos que a configuração de *Message Count* = 10 é a que apresenta uma pior performance, com o maior valor médio de Latência, de aproximadamente 10 milissegundos. Observa-se também um comportamento idêntico ao que foi observado nas secções 7.1 e 7.3, onde nos momentos iniciais entre as 0 e 1500 transações, observa-se uma descida nos valores de Latência, que uma vez mais poderá ser criada pelos resíduos das rondas de "aquecimento". De notar, que essa descida é muito mais acentuada na configuração de *Message Count* = 10, uma vez que o seu *Throughput* é muito inferior ao *Throughput* apresentado pelas outras configurações.

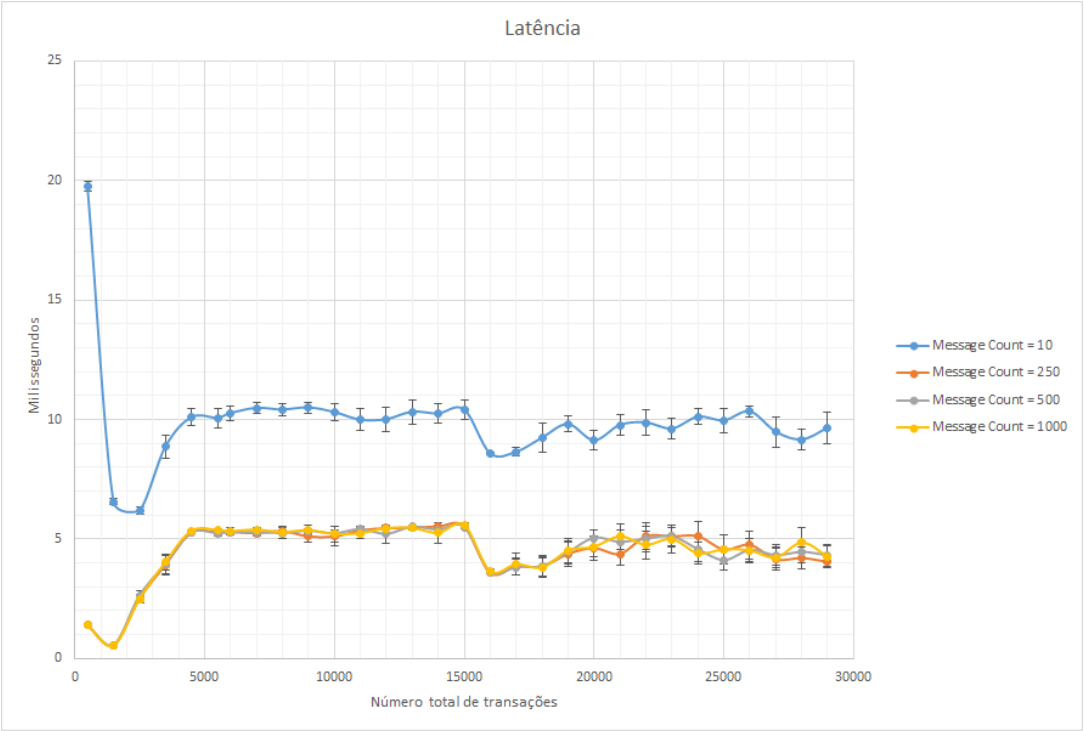


Figura 45 - Comparação dos valores de Latência – T4F

Na Figura 46 e Figura 47, observamos a comparação dos valores de consumo de CPU e consumo de memória RAM para as diversas configurações do *Orderer*, respetivamente. Uma vez mais, pela Figura 46, observamos que a performance entre as diversas configurações é idêntica, apresentando o mesmo comportamento e valores médios muito próximos de 33%, 34%, 34% e 34%, para as configurações de *Message Count* = 10, 250, 500 e 1000 respetivamente. Concluímos dessa forma que a modificação do parâmetro *Message Count* não afeta muito o consumo de CPU.

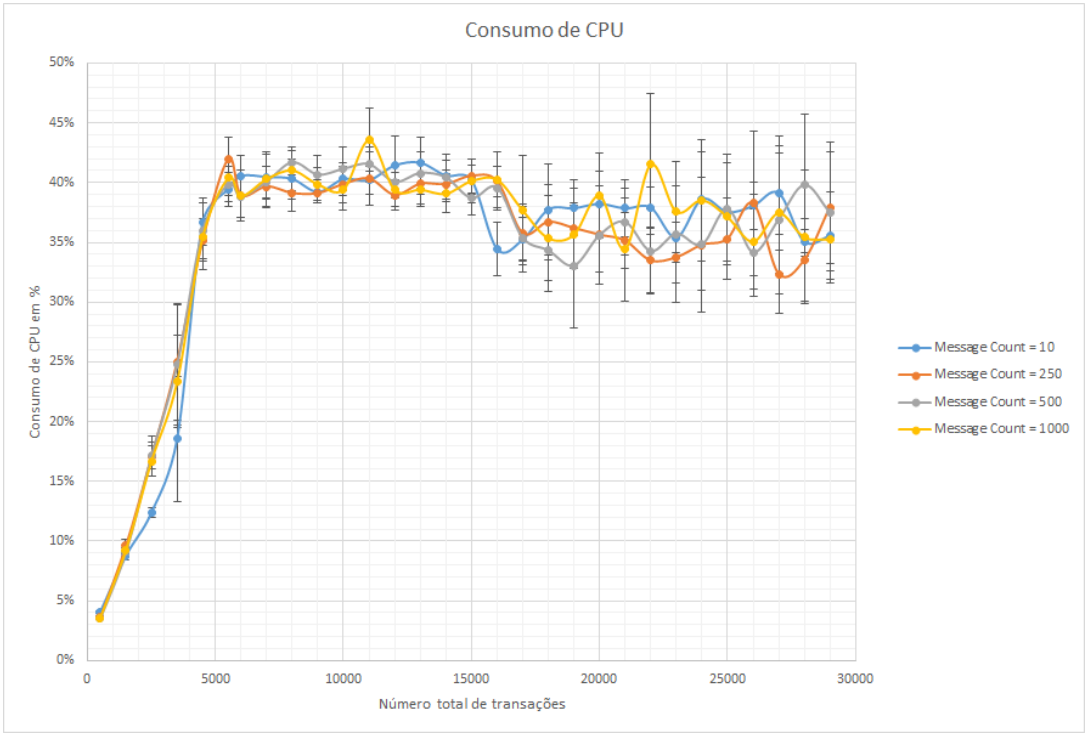


Figura 46 - Comparação dos valores do consumo de CPU – T4F

Na Figura 47, encontramos os valores de consumo de memória RAM. Nesta Figura observamos um comportamento expectável, onde o consumo de memória RAM aumenta com o aumento do tamanho do *batch*, uma vez que com o aumento deste, aumenta a complexidade do processo de criação de um bloco.

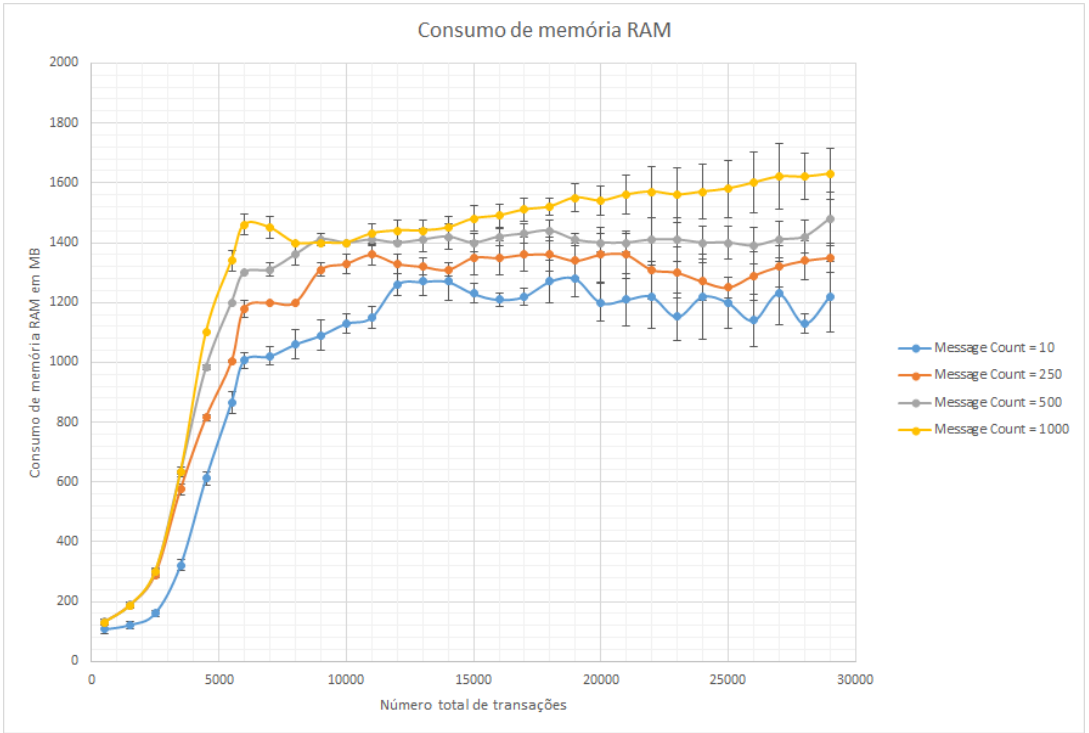


Figura 47 - Comparação dos valores do consumo de memória RAM – T4F

Em suma, são várias as conclusões retiradas da execução do teste T4F. Ao modificarmos o parâmetro *MaxMessageCount* do *Orderer*, concluímos que quanto maior o valor do parâmetro, maior o *Throughput* e consequentemente menor a Latência da plataforma, validando dessa forma a afirmação apresentada no início da secção. Foi também possível concluir que a mudança do parâmetro não influencia significativamente o consumo de CPU por parte de um *peer*. Contrariamente, o consumo de memória RAM aumenta quando o valor do parâmetro também é aumentado.

## 7.5 Teste T5F

O teste T5F foi desenhado com o objetivo de medir o impacto que a modificação da política de aprovação de transações tem nos valores máximos de *Throughput* e Latência da plataforma. O teste seguiu o modelo de execução do teste T1F, onde entre execuções é destruída e criada uma nova rede *blockchain* e, de igual forma utiliza o mesmo intervalo de amostras.

A politica de aprovação está presente no ficheiro *"fabric.json"*, na diretoria do *workload* em causa, sobre o parâmetro *"endorsement-policy"*. Considerando a política de aprovação representada na Figura 48, a interpretação da mesma segue o modelo: a transação pode ser assinada por qualquer membro das organizações pelo valor de *"signed-by"*. Esse valor vai corresponder ao índice no *array* de *identities*. No caso da Figura 48, esta é interpretada como: a transação pode ser assinada por qualquer um dos membros das duas organizações da rede, que neste caso são a *Org1MSP* e *Org2MSP*.

```
"endorsement-policy": {
  "identities": [
    { "role": { "name": "member", "mspId": "Org1MSP" } },
    { "role": { "name": "member", "mspId": "Org2MSP" } },
    { "role": { "name": "admin",  "mspId": "Org1MSP" } } ],
  "policy": {
    "1-of": [
      { "signed-by": 0 },
      { "signed-by": 1 } ]
  }
},
```

Figura 48 - Configuração original da política de aprovação

A segunda política de aprovação foi modificada para a execução do teste em causa. Esta está representada na Figura 49, e interpreta-se como: a transação pode ser assinada por qualquer membro da organização *Org1MSP*.

```
"endorsement-policy": {
  "identities": [
    { "role": { "name": "member", "mspId": "Org1MSP" } },
    { "role": { "name": "admin", "mspId": "Org1MSP" } } ],
  "policy": {
    "1-of": [
      { "signed-by": 0 }
    ]
  }
},
```

Figura 49 - Configuração modificada da política de aprovação

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 50. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "write",
  "limit": 3000,
  "executions": 10,
  "type": "controlled",
  "n_chaincodes": 1
}
```

Figura 50 - Excerto do ficheiro de configuração para a execução do teste T5F

Na Figura 51, encontramos a comparação entre as duas políticas de aprovação. Traduzindo a legenda, é utilizado o termo E. P. para *Endorsment Policy*, ou política de aprovação. Da observação desta, reparamos que a E. P. modificada apresenta melhores valores de *Throughput* quando comparados com os valores da E. P. original, com um valor médio de 316 *TPS* para a E. P. modificada e 259 *TPS* para a original, respetivamente. Este comportamento foi observado e relatado por (Thakkar et al., 2018) na *Observation 6*, ainda que a versão da plataforma *Hyperledger Fabric* usada pelos autores seja diferente da versão usada pelo aluno, *v1.0* e *v1.4* respetivamente.

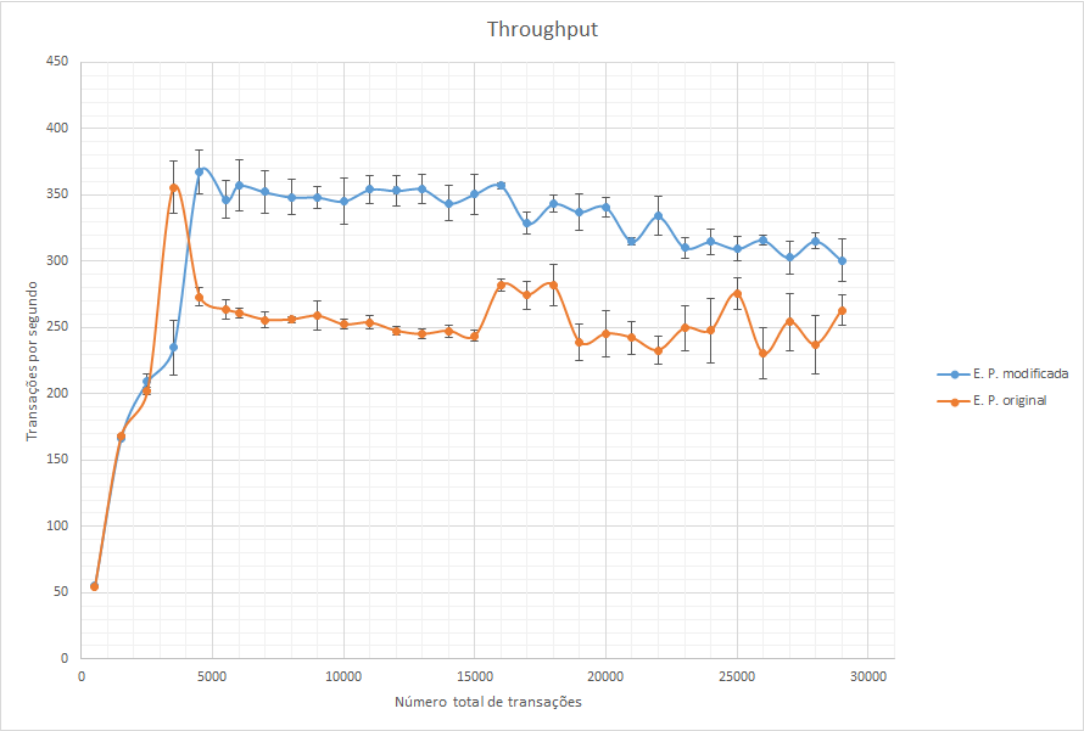


Figura 51 - Comparação dos valores de Throughput – T5F

Na Figura 87 encontramos um *zoom* ao gráfico da Figura 51, tendo como intervalo de amostras entre as 0 e 6000 transações. Desta Figura, observamos no intervalo compreendido entre as 3000 e 4000 transações, o comportamento distinto das duas E. P. em comparação. Neste caso específico, no momento em que os valores de *Throughput* para a E. P. original decrescem, momento analisado na secção 7.1, os valores de *Throughput* para a E. P. modificada aumentam, atingindo o seu valor máximo no intervalo das 3800 transações.

A estranha mudança de comportamento levou a que fosse feita uma nova análise aos valores das transações bem-sucedidas. Dessa análise resultou o gráfico da Figura 52. Nesta, observamos o mesmo comportamento encontrado e analisado na Figura 30, apresentando, no entanto, duas pequenas diferenças. A primeira diferença é que o número de transações bem-sucedidas é superior, significando que para a E. P. modificada o número de transações falhadas diminuiu. A segunda diferença encontra-se encontra-se mudança do intervalo em que a quebra das transações bem-sucedidas acontece, passando agora a ser entre as 3600 e 4000 transações.

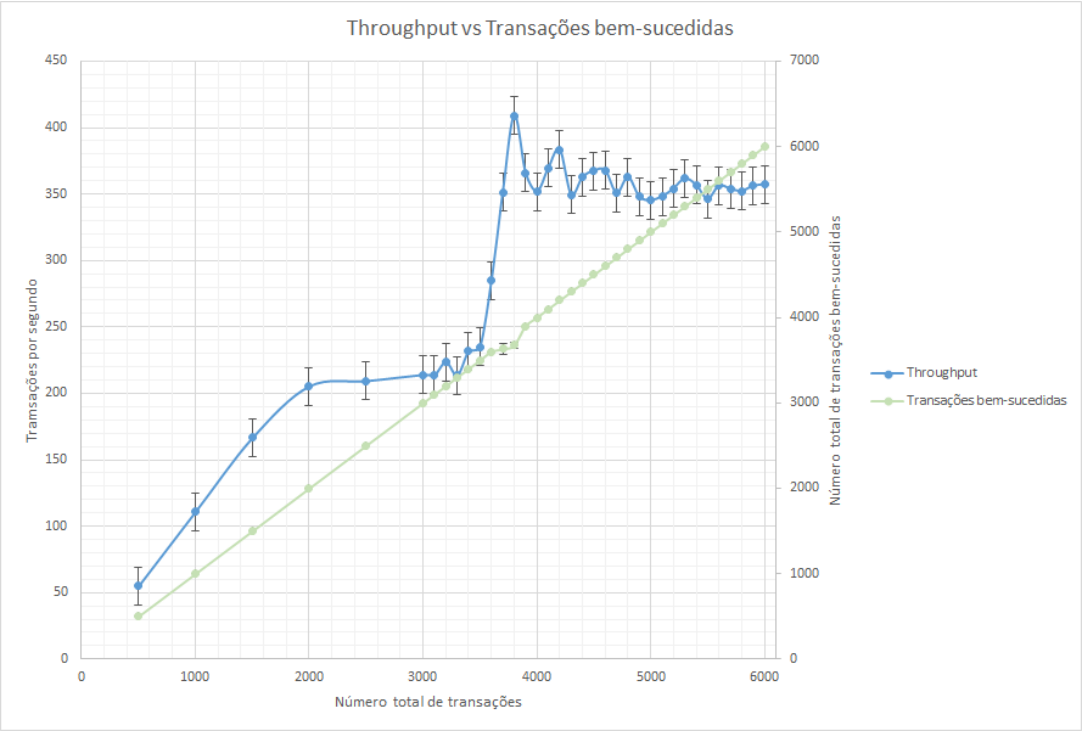


Figura 52 - Comparação entre o Throughput e as transações bem-sucedidas - Política de aprovação modificada – T5F

Na Figura 53 observamos a comparação dos valores da Latência entre as duas políticas de aprovação. Desta comparação, concluímos novamente que a E. P. modificada apresenta valores mais baixos de Latência, com um valor médio de 3 milissegundos. É ainda possível observar que a Latência para a E. P. modificada não apresenta a mesma descida abrupta verificada no intervalo entre as 15000 e 20000 transações para os valores da E. P. original. De igual forma, os intervalos de confiança para os valores da Latência da E. P. modificada diminuem no intervalo entre as 15000 e 29000 transações.

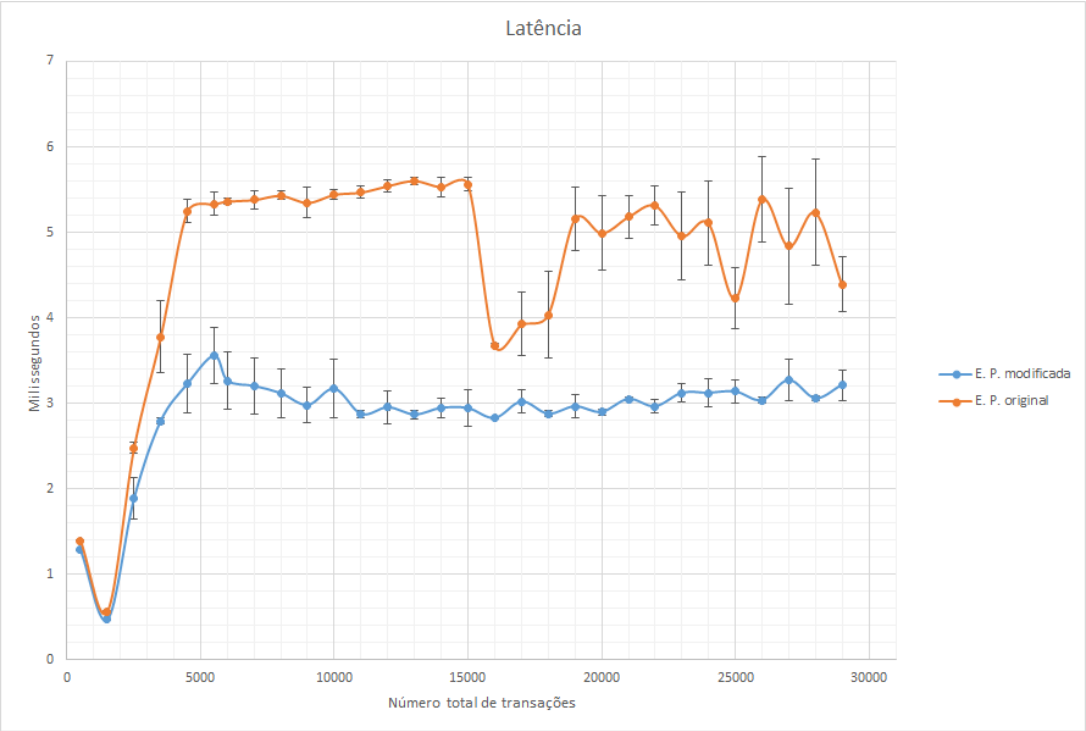


Figura 53 - Comparação dos valores de Latência – T5F

Na Figura 54 observamos a comparação entre o consumo de CPU para as duas políticas de aprovação. Uma vez mais, é observado uma melhor performance pela E. P. modificada. Os autores (Thakkar et al., 2018), justificam este comportamento afirmando que durante a fase de validação das politicas de aprovação são desempenhadas três grandes operações de consumo de CPU intensivo. São elas:

1. Deserialização da identidade de um utilizador, convertendo os certificados de autoridade dos mesmos;
2. Validação dessa identidade com o MSP da organização;
3. Verificação da assinatura nos dados da transação;

Assim, com o aumento de subpolíticas de aprovação, aumenta o número de identidades que um *peer* tem que gerir e assinaturas que tem de validar, aumentando dessa forma o consumo de CPU. No caso da comparação utilizada no teste T5F, a política de aprovação original (Figura 48) tem duas subpolíticas, contrastando com apenas uma na política de aprovação modificada (Figura 49).

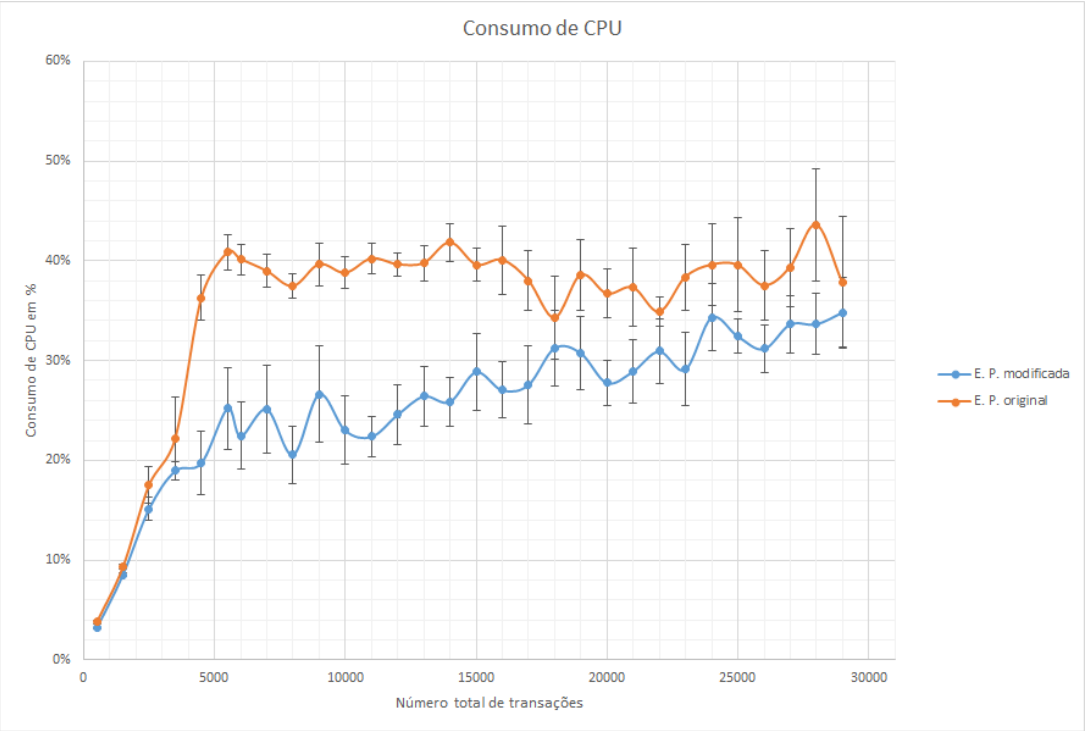


Figura 54 - Comparação dos valores do consumo de CPU – T5F

A terminar, na Figura 55 observamos a comparação do consumo de memória RAM. Este, no entanto, apresenta um resultado diferente do observado até agora, onde é o único caso em que a E. P. original apresenta um melhor resultado que a E. P. modificada. Foi realizada uma investigação para encontrar o motivo dos valores do consumo de memória RAM serem tão altos para a E. P. modificada, e, ainda que não tenha sido encontrado nada de relevante na literatura presente, o aluno sugere que estes valores podem dever-se ao facto de o *peer* da rede estar a desempenhar as funções de assinar as transações sozinho, fazendo com que os valores do consumo de memória RAM aumentem com o aumento no número total de transações.

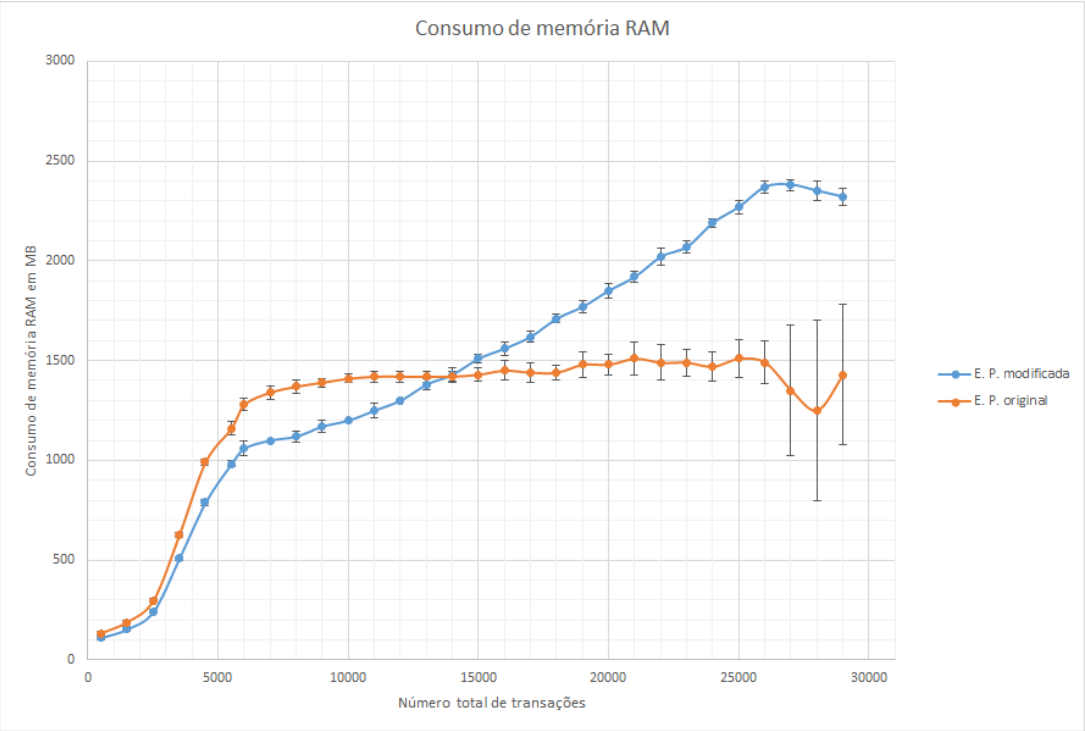


Figura 55 - Comparação dos valores do consumo de memória RAM – T5F

Em suma, é possível concluir com a execução deste teste que quanto menor for o número de subpolíticas presentes na política de aprovação, maior será o *Throughput* e menor a Latência da plataforma. Há, no entanto, que perceber que esta não faz parte dos parâmetros de modificação para melhoramento da performance da plataforma, uma vez que é definida pelas regras do negócio que implementa a plataforma *blockchain*. De igual forma, é recomendado que não seja usada em ambientes de produção, uma vez que é possível que uma aplicação ignore a E. P., e envie ao *Orderer* uma transação manualmente construída, com um conjunto *writeset* arbitrário. O resultado deste processo permite que o contexto do utilizador que assina as transações adicione essa transação ao *ledger*, ainda que esta seja fraudulenta (The Linux Foundation, 2019b).

## 7.6 Teste T6F

O teste T6F foi desenhado com o objetivo de medir o impacto que vários tamanhos de carga enviados para os *chaincodes* têm nos valores máximos de *Throughput* e Latência da plataforma. O teste seguiu o modelo de execução do teste T1F, onde entre execuções é destruída e criada uma nova rede *blockchain.*

Este teste é o primeiro teste da categoria *Micro Benckmarks*, e é também o primeiro teste onde a rede *blockchain* sofre alterações, passando esta a ser composta apenas por uma Organização (*Org1*) e por um *peer* (*peer1*). Para utilizar a nova configuração, basta ao utilizador definir no ficheiro *"fabric.json"*, presente na diretoria do teste, apenas uma organização e um *peer*. Igualmente, este é o primeiro teste em que o intervalo de amostras é reduzido, passando a compreender-se entre as 0 e 15000 transações. Este valor foi encontrado por tentativa e erro, uma vez que a partir desse valor, a plataforma *Gauge* apresentava erros e não permitia a conclusão da execução do teste.

Para a execução deste teste, foram consideradas três *strings* de tamanho 34, 38 e 41 *bytes*, de valor *"1"*, *"12121"* e *"12121121"* respetivamente. Foi considerada a escolha destes três valores numa tentativa de encontrar um valor mínimo, um intermédio e um valor máximo. Dos testes realizados, a *string* de 41 *bytes* demonstrou ser o valor máximo aceite pelo *chaincode* específico do teste, não significando que este valor se verifique para *chaincodes* no geral.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 56. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "chaincode-payload-size",
  "limit": 1500,
  "executions": 10,
  "type": "micro",
  "n_chaincodes": 1
}
```

Figura 56 - Excerto do ficheiro de configuração para a execução do teste T6F

Na Figura 57 observamos a comparação dos valores de *Throughput* para os diferentes tamanhos de carga enviados para o *chaincode.* Concluímos, assim, que o tamanho da carga não influencia diretamente o valor de *Throughput*. No entanto, se analisarmos com detalhe os valores médios das três séries em comparação, concluímos que é a carga de de 38 *bytes* que apresenta o menor valor médio de aproximadamente 314 *TPS*, contra o valor de aproximadamente 315 *TPS* das cargas de 34 e 38 *bytes* respetivamente.

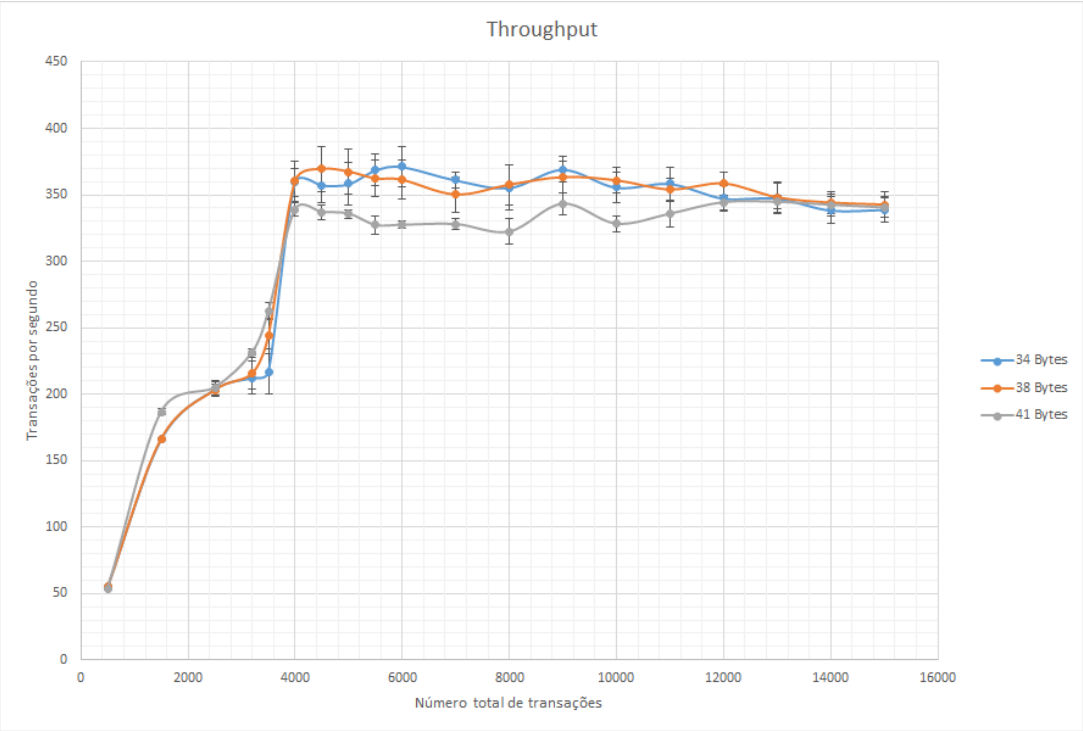


Figura 57 - Comparação dos valores de Throughput – T6F

Na Figura 58 é feita a comparação para os valores da Latência. Concluímos igualmente, que o tamanho da carga enviada por argumento não influencia os valores da Latência. No entanto, para os valores de Latência, observamos que a melhor performance pertence agora à carga de 41 *bytes*, com um valor médio de 2.6 milissegundos, contra os valores médios de 2.9 milissegundos, da carga de 34 e 38 *bytes* respetivamente.

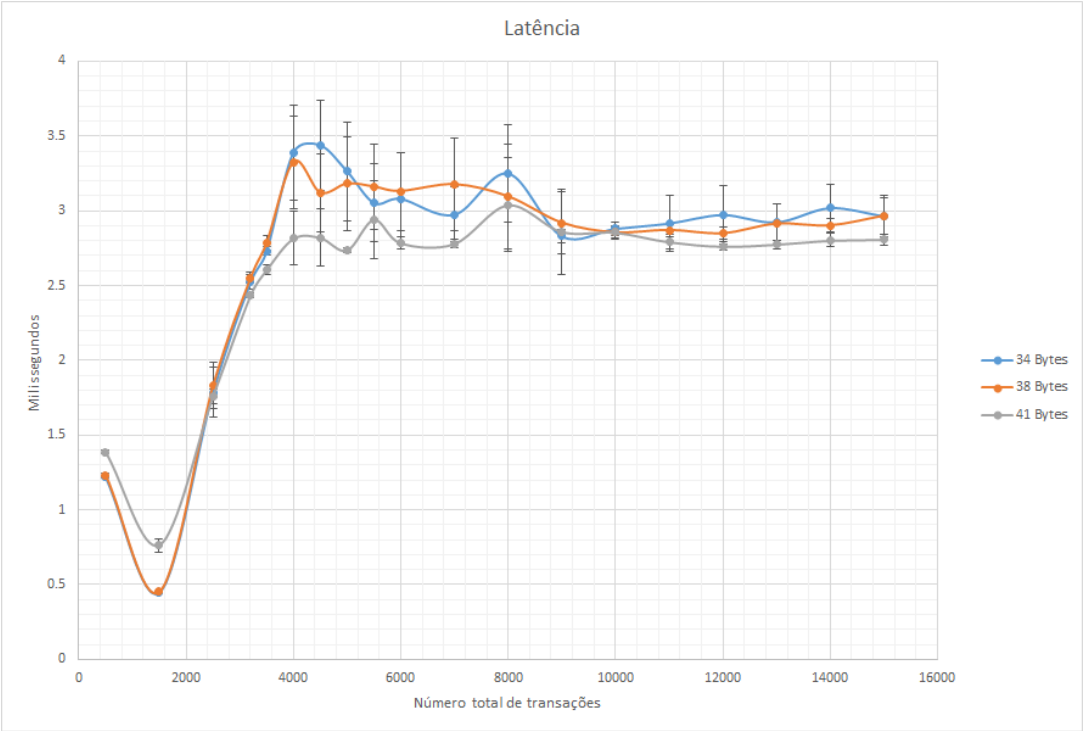


Figura 58 - Comparação dos valores de Latência – T6F

É na Figura 59 e Figura 60, correspondentes ao consumo de CPU e consumo de memória RAM, respetivamente. Para a primeira Figura, observamos que a variação do tamanho da carga enviada a um *chaincode* tem não tem um grande impacto nos valores de consumo de CPU, uma vez que os diversos tamanhos de carga apresentam valores semelhantes. Uma vez mais, os intervalos de confiança apresentados apenas nos permitem concluir que os valores de consumo de CPU apresentam uma grande variação.

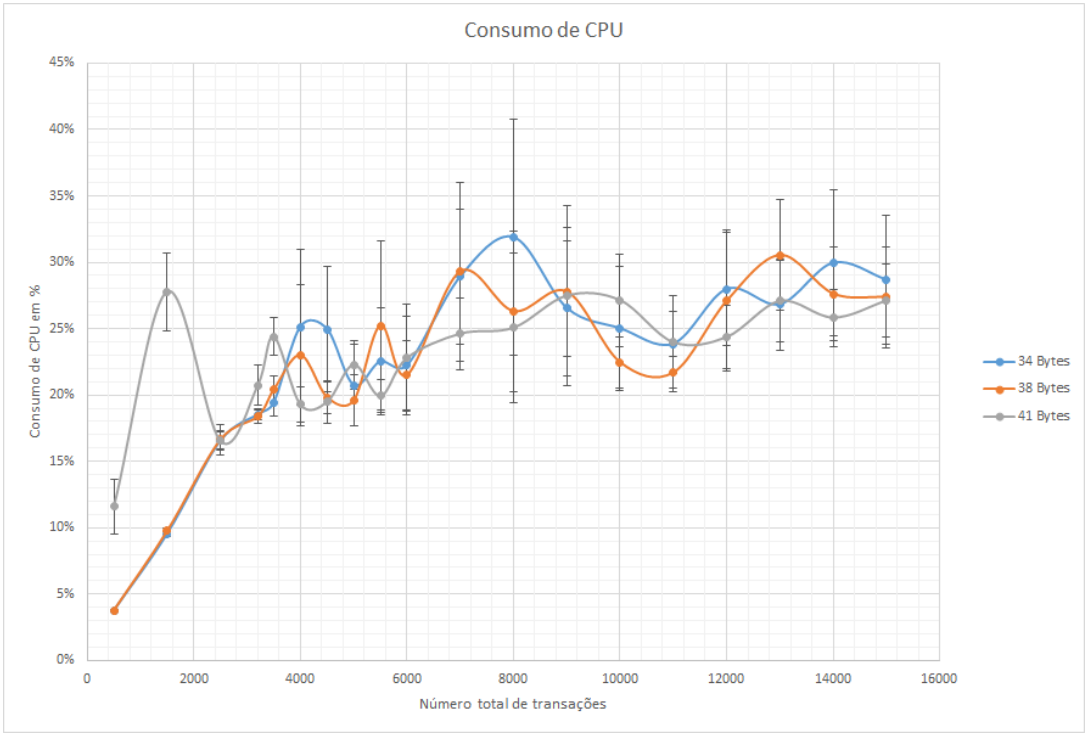


Figura 59 - Comparação dos valores do consumo de CPU – T6F

Na Figura 60 observamos que os valores do consumo de memória RAM para a carga de 41 *bytes* são aproximadamente 4.3 vezes superiores aos valores para a carga de 34 e 38 *bytes*, que por sua vez apresentam valores iguais.

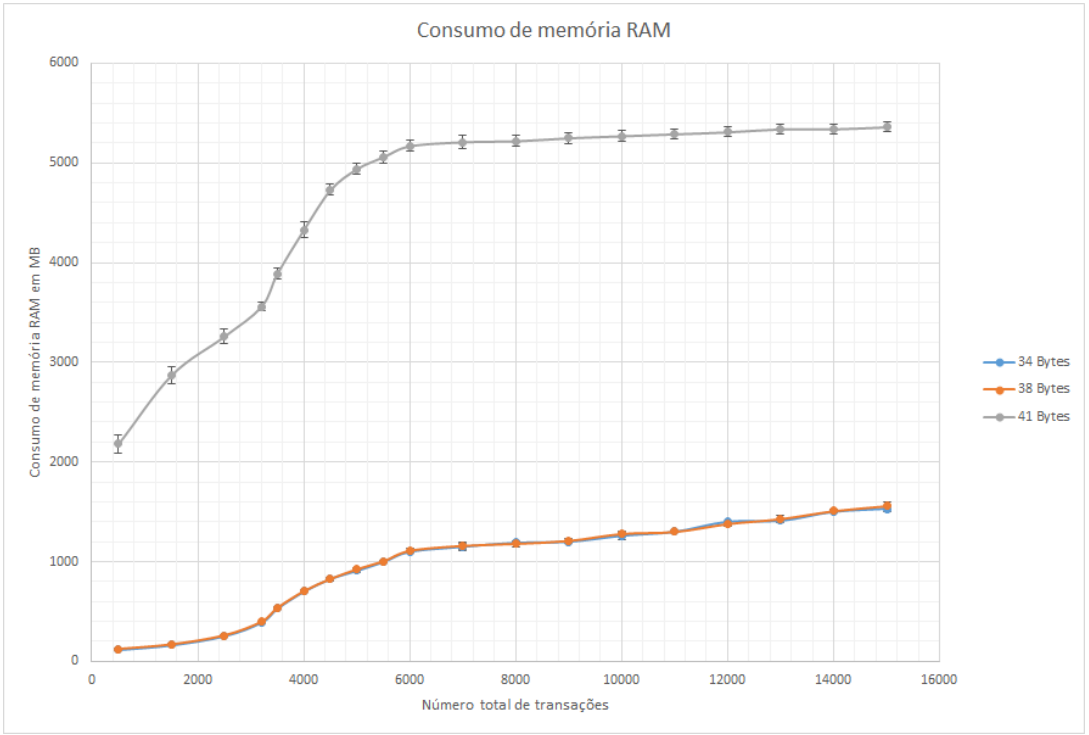


Figura 60 - Comparação dos valores do consumo de memória RAM – T6F

Ao relembrar que no início do teste foi dito que a rede passaria a ser composta apenas por uma organização e um *peer*, podemos dessa forma justificar os valores altos para o consumo de CPU e consumo de memória RAM da carga com 41 *bytes*, uma vez que o único *peer* da rede passou a desempenhar as funções conjuntas de simulação das propostas de transação e de assinar as mesmas.

Em suma podemos concluir que a o tamanho da carga enviada por argumento a um *chaincode* não afeta diretamente os valores de *Throughput*, Latência e consumo de CPU da plataforma. No entanto, a mesma condição influencia o consumo memória RAM por parte de um *peer*, fazendo com que os valores destes aumentem juntamente com o aumento do tamanho da carga enviada por argumento a um *chaincode*.

## 7.7 Teste T7F

O teste T7F foi desenhado com o objetivo de medir o impacto que o uso de eventos nos *chaincodes* tem nos valores máximos de *Throughput* e Latência da plataforma. Os eventos podem ser usados no contexto de um *chaincode* como uma alternativa a uma busca intensiva ao estado do *ledger*. Assim, um evento pode ser disparado para notificar um utilizador sempre que há uma nova interação com o *chaincode* em causa (Kuhnert, 2019). O teste T7F adotou o mesmo modelo de execução que o teste T6F e, dessa forma, utilizou a mesma configuração da rede *blockchain*, o mesmo intervalo de amostras e os mesmos tamanhos de cargas enviadas para o *chaincode* como argumento. Da apresentação dos resultados é esperada a comparação dos valores do teste T7F com o teste T6F.

De forma a não tornar esta secção demasiado extensa, são apresentados apenas os resultados da comparação para a carga de tamanho de 34 *bytes*, sendo que os resultados para os restantes tamanhos de carga podem ser consultados no Apêndice B Resultados adicionais da execução dos testes de benchmark.

Para a execução deste teste, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 56. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

Na Figura 61, Figura 88 e Figura 89 observamos a comparação dos valores de *Throughput* para os tamanhos de carga de 34, 38 e 41 *bytes* respetivamente, com e sem o uso de eventos no *chaincode*. De uma análise a estas figuras é possível concluir que, com os tamanhos de carga de 34 e 38 *bytes*, o uso de eventos no *chaincode* não influencia os valores de *Throughput*. Na Figura 89 é, no entanto, possível observar que no intervalo entre as 2000 e as 10000 transações, os valores de *Throughput* para a carga de 41 *bytes* são superiores quando não são usados eventos no *chaincode*.

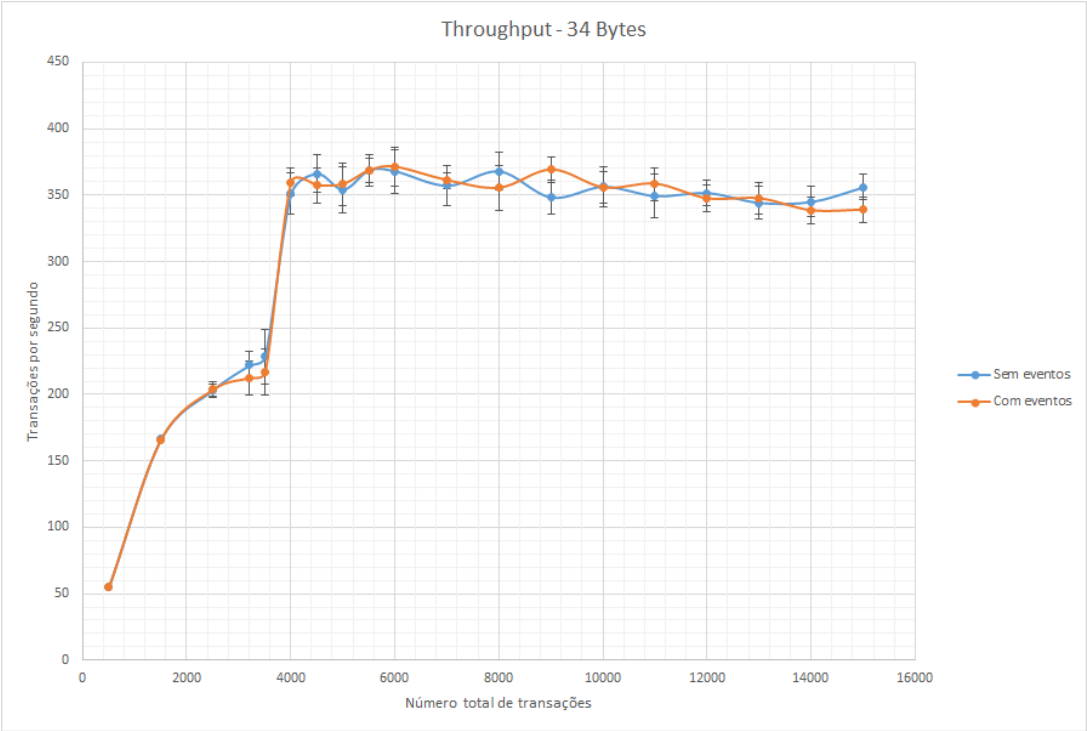


Figura 61 - Comparação dos valores de Throughput para a carga de tamanho 34 bytes – T7F

Quanto à Latência da plataforma, analisada na Figura 62, Figura 90 e Figura 91, podemos concluir que os seus valores não são afetados pelo uso de eventos, onde a diferença de valores apresentados nestas figuras é mínima.

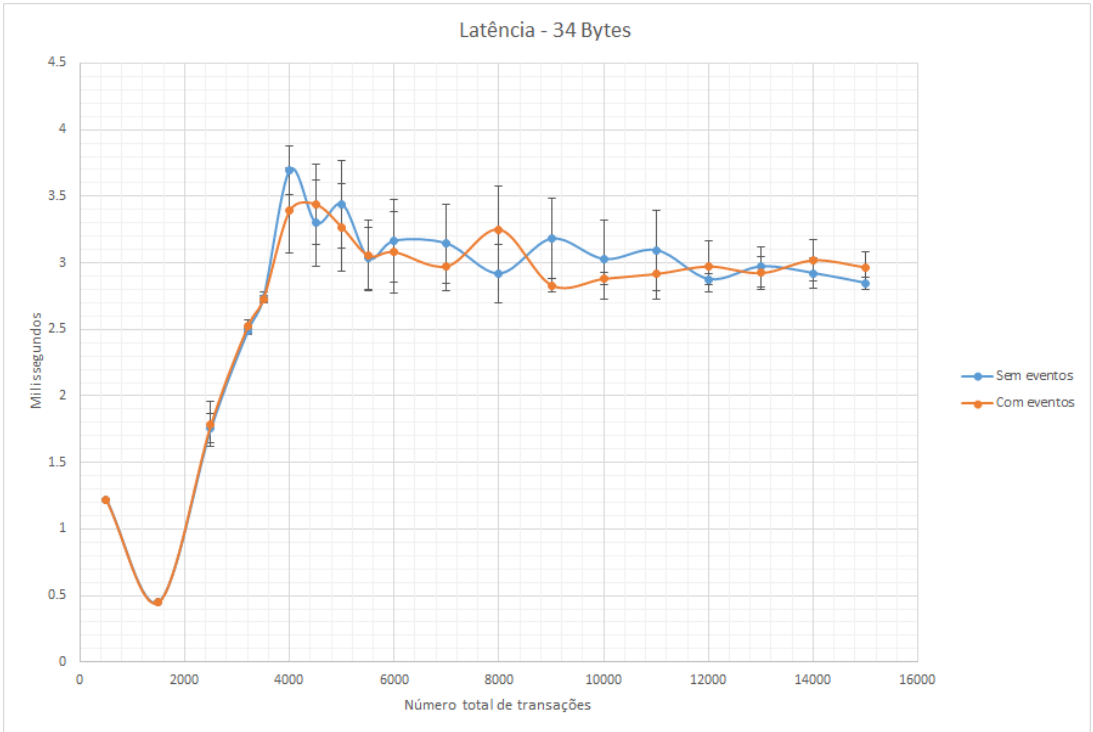


Figura 62 - Comparação dos valores de Latência para a carga de tamanho 34 bytes – T7F

Para os valores de consumo de CPU, a tendência observada anteriormente mantém-se. Assim na Figura 63, Figura 94 e Figura 95, observamos novamente que o uso de eventos nos *chaincodes* faz aumentar os valores consumo de CPU. De igual forma ao observado na Figura 89, a diferença de valores é mínima. Os altos valores dos intervalos de confiança apresentados para os valores de uso de eventos nos *chaincodes* permitem concluir que há uma grande variação destes.

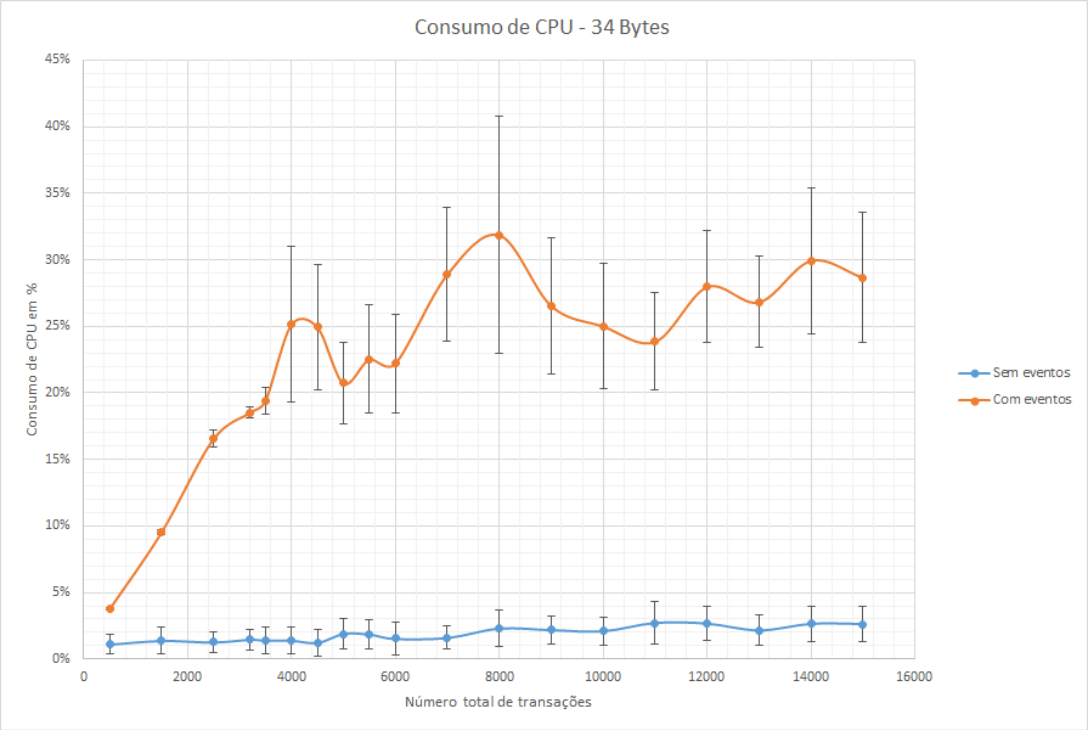


Figura 63 - Comparação dos valores de consumo de CPU para a carga de tamanho 34 bytes – T7F

Na Figura 64 e Figura 93 observamos a tendência observada na Figura anterior, onde há um aumento dos valores de consumo de memória RAM com o uso de eventos no *chaincode*. Na Figura 92 também é possível observar essa diferença, ainda que mínima.

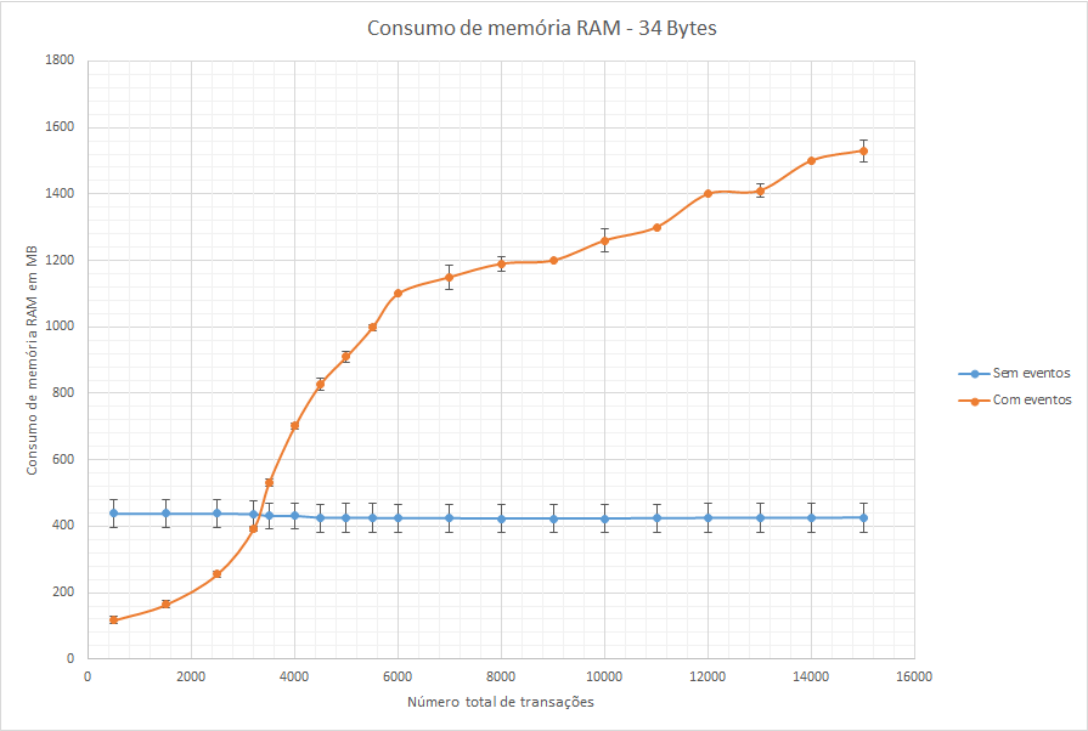


Figura 64 - Comparação dos valores de consumo de memória RAM para a carga de tamanho 34 bytes – T7F

Em suma concluímos que o uso de eventos nos *chaincodes* não afeta diretamente os valores de *Throughput* e Latência. A diferença nos valores passa a ser mais obvia quando são comparados os valores de consumo de memória RAM e de consumo de CPU para as cargas de tamanho 34 e 41 *bytes*. Para a carga de tamanho 38 *bytes*, a diferença de valores não é significativa.

## 7.8 Teste T8F

O teste T8F compara os resultados da execução do- Teste T8.1F - Hyperledger Fabric e do - Teste T8.2F - Hyperledger Fabric. O teste T8.1F foi desenhado com o principal objetivo de recolher os valores de *Throughput* e Latência da plataforma, quando são invocadas operações de leitura a *chaincodes* distribuídos por diversos canais (inter-channel), e o teste T8.2F desenhado para avaliar os valores de *Throughput* e Latência, quando as invocações aos diversos *chaincodes* são feitas no mesmo canal (intra-channel). Assim, para o T8.1F, a rede foi composta por dois canais com um *chaincode* instanciado em cada um, enquanto que para o T8.2F a rede foi composta por apenas um canal, com dois *chaincodes* instanciados. Ambas as redes compreendiam apenas uma organização (*Org1*) e um *peer* (*peer1*). Semelhante ao observado na secção 7.6, o intervalo de amostras para este teste compreende-se entre as 0 e as 15000 transações. O nível de profundidade de chamadas para correr os dois testes foi de um.

Para a execução do teste T8.1F, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 65. O parâmetro *"n_chaincodes"* não é utilizado no

contexto deste teste. Para a execução do teste T8.2F, o utilizador deve modificar o parâmetro *"workload"* para *"intra-channel-read"*.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "inter-channel-read",
  "limit": 1500,
  "executions": 10,
  "type": "micro",
  "n_chaincodes": 1
}
```

Figura 65 - Excerto do ficheiro de configuração para a execução do teste T8F

Na Figura 66 observamos a comparação dos valores de *Throughput*. Desta concluímos que no intervalo inicial entre as 0 e 4000 transações, a performance para as chamadas inter-channel e intra-channel é idêntica. A partir do intervalo das 4000 transações é onde se observa a diferença nos valores de *Throughput*, com as chamadas intra-channel a terem uma melhor performance, com um valor médio de 1187 *TPS*, contrastando com o valor médio de 1131 *TPS* das chamadas inter-channel. Quando comparamos os valores gerados pelas chamadas intra-channel com os valores gerados por T2F, observamos que os primeiros são menores que os segundos, podendo tal comportamento justificar-se pela diminuição dos componentes constituintes da rede.

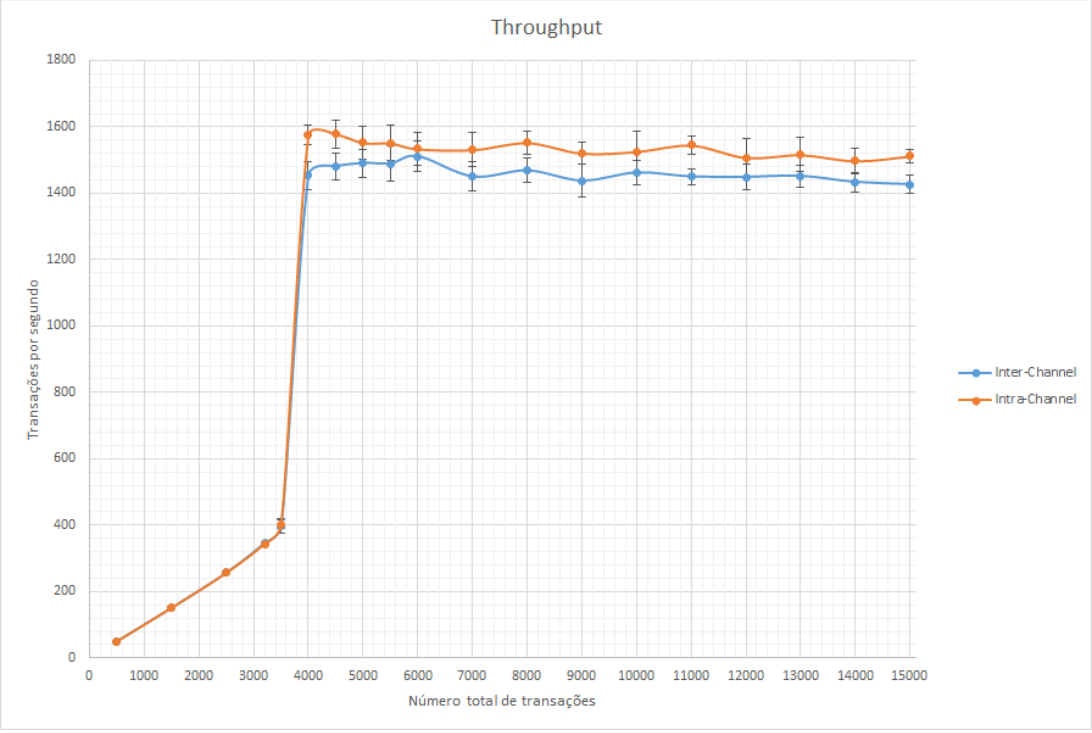


Figura 66 - Comparação dos valores de Throughput entre operações de leitura a chaincodes inter e intra-channel – T8F

Na Figura 67 mantém-se a tendência observada na Figura 66, onde só a partir do intervalo das 4000 transações é que a diferença nos valores é mais visível, com as chamadas intra-channel a terem um melhor desempenho. No entanto, a diferença observada para os valores de Latência é mais subtil que a diferença observada para os valores de *Throughput*.

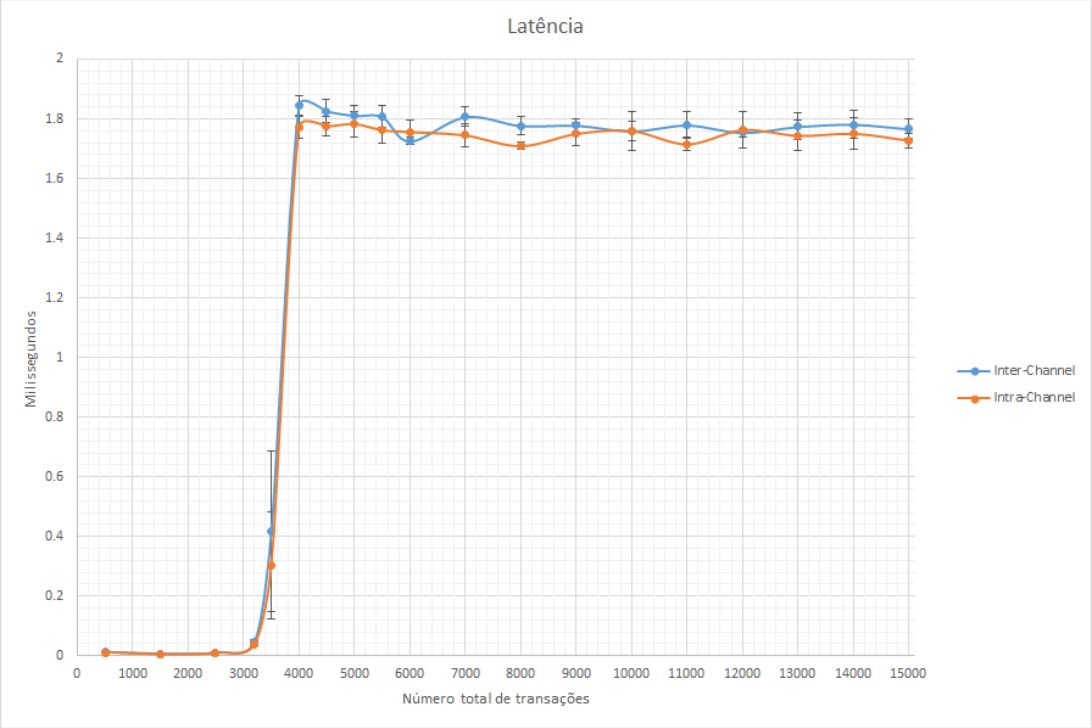


Figura 67 - Comparação dos valores de Latência entre operações de leitura a chaincodes inter e intra-channel – T8F

Os valores do consumo de CPU, representados na Figura 68, são novamente muito semelhantes, passando a haver uma diferença significativa entre os mesmos a partir do intervalo de 8000 transações, onde o consumo de CPU das chamadas intra-channel é superior ao consumo das chamadas inter-channel.

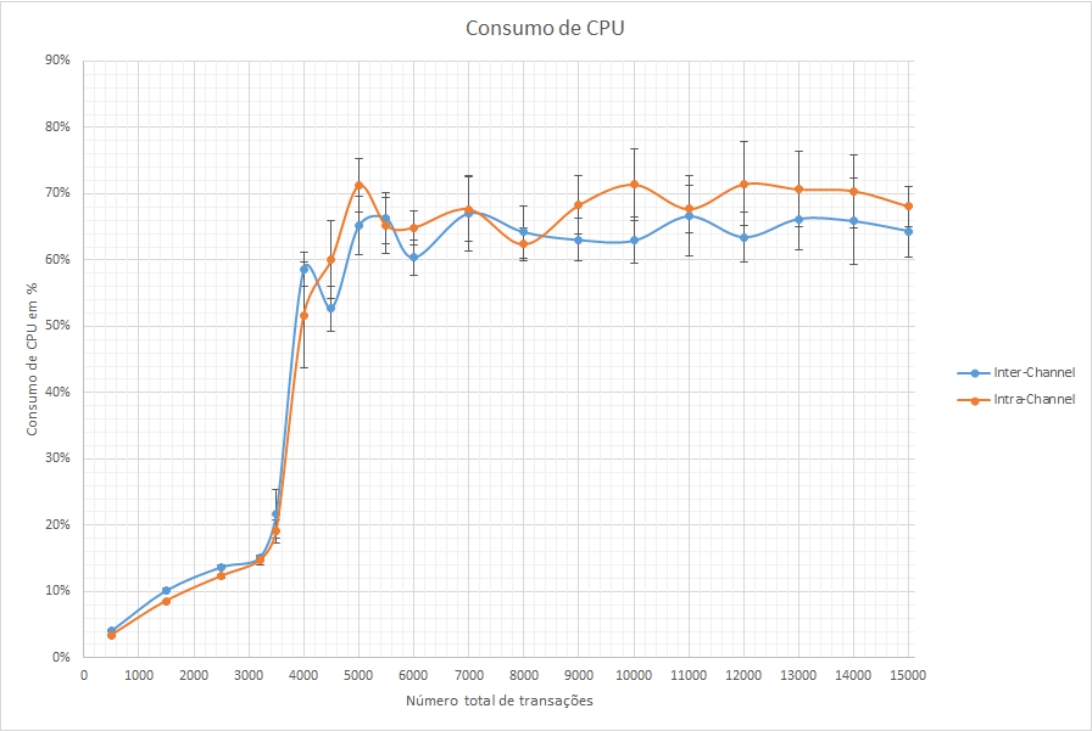


Figura 68 - Comparação dos valores de consumo de CPU entre operações de leitura a chaincodes inter e intra-channel – T8F

A maior diferença entre os testes T8.1F e T8.2F é observada na Figura 69, onde se comparam os valores do consumo de memória RAM. Nessa Figura, observamos o padrão encontrado na Figura 37, onde os valores para esta grandeza decrescem no intervalo inicial de transações, de igual forma, os valores para os intervalos de confiança são muito grandes, concluindo assim estes vão ter uma grande variação. Por o teste executar uma operação de leitura no *chaincode*, a justificação para este comportamento é a mesma que foi apresentada na secção 7.2.

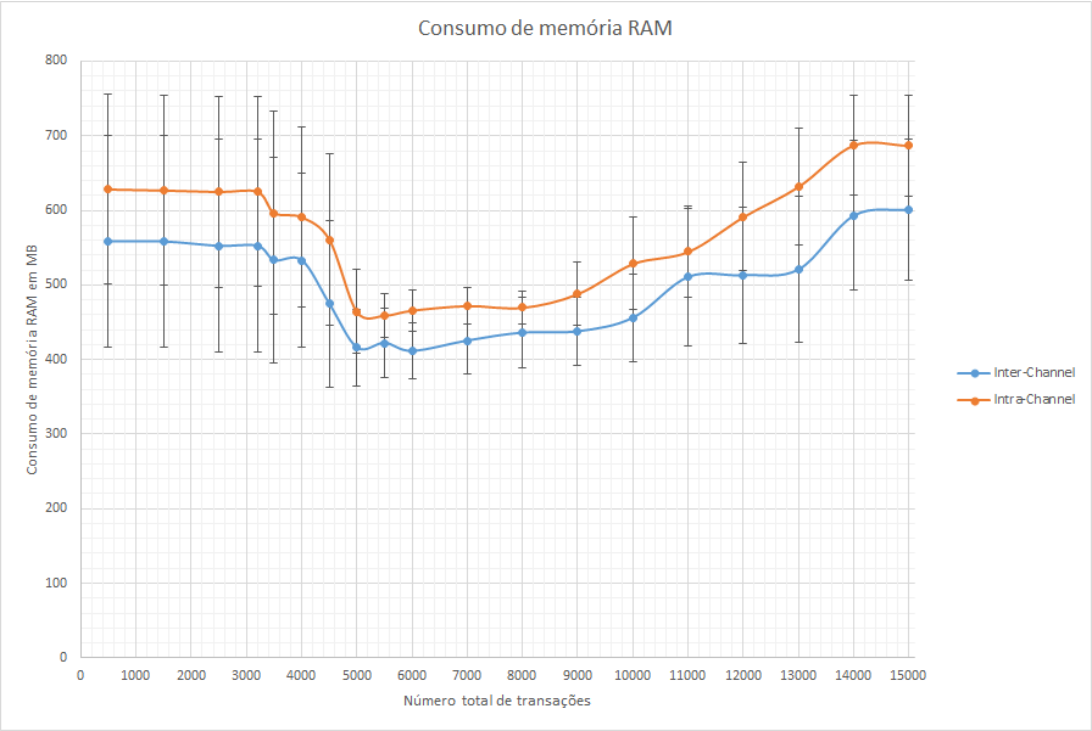


Figura 69 - Comparação dos valores de consumo de memória RAM entre operações de leitura a chaincodes inter e intra-channel – T8F

Em suma, após comparação dos valores de *Throughput*, Latência e consumo de CPU, para invocações de operações de leitura a *chaincodes* inter e intra-channel, concluímos que a diferença entre estes não é significativa, não afetando muito os seus valores finais. Há uma diferença nos valores de consumo de memória RAM, ainda que esta não seja significativa.

## 7.9 Teste T9F

O teste T9F foi desenhado com o principal objetivo de avaliar os valores de *Throughput* e Latência da plataforma, quando são invocadas operações de escrita a *chaincodes* dentro do mesmo canal. Para operações de escrita na entre diversos *chaincodes*, o *chaincode* invocado tem de estar instanciado no mesmo canal que o *chaincode* que invoca (The Linux Foundation, 2019a). De forma semelhante ao observado na secção 7.6, o intervalo de amostras para este teste compreende-se entre as 0 e as 15000 transações. A rede *blockchain* utilizada compreende uma organização (*Org1*), um *peer* (*peer1*), um canal e quatro *chaincodes* instanciados nesse canal. Os valores utilizados para a profundidade das chamadas dos *chaincodes* compreendem-se entre 2 e 4, sendo que 1 corresponde a uma chamada do *chaincode* a ele mesmo.

Para a execução do teste T9F, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 70. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "intra-channel-write",
  "limit": 1500,
  "executions": 10,
  "type": "micro",
  "n_chaincodes": 1
}
```

Figura 70 - Excerto do ficheiro de configuração para a execução do teste T9F

Na Figura 71, observamos a comparação dos valores de *Throughput* para os níveis de profundidade de invocação de 2, 3 e 4. Para estes níveis, o *Throughput* atinge o seu valor máximo perto do intervalo de 4000 transações, correspondente à invocação de nível 4, com aproximadamente 411 *TPS*. Já o nível que apresenta um melhor desempenho geral é o nível 2, com um valor médio de 297 *TPS*, contra as 287 e 280 *TPS* dos níveis 3 e 4, respetivamente.

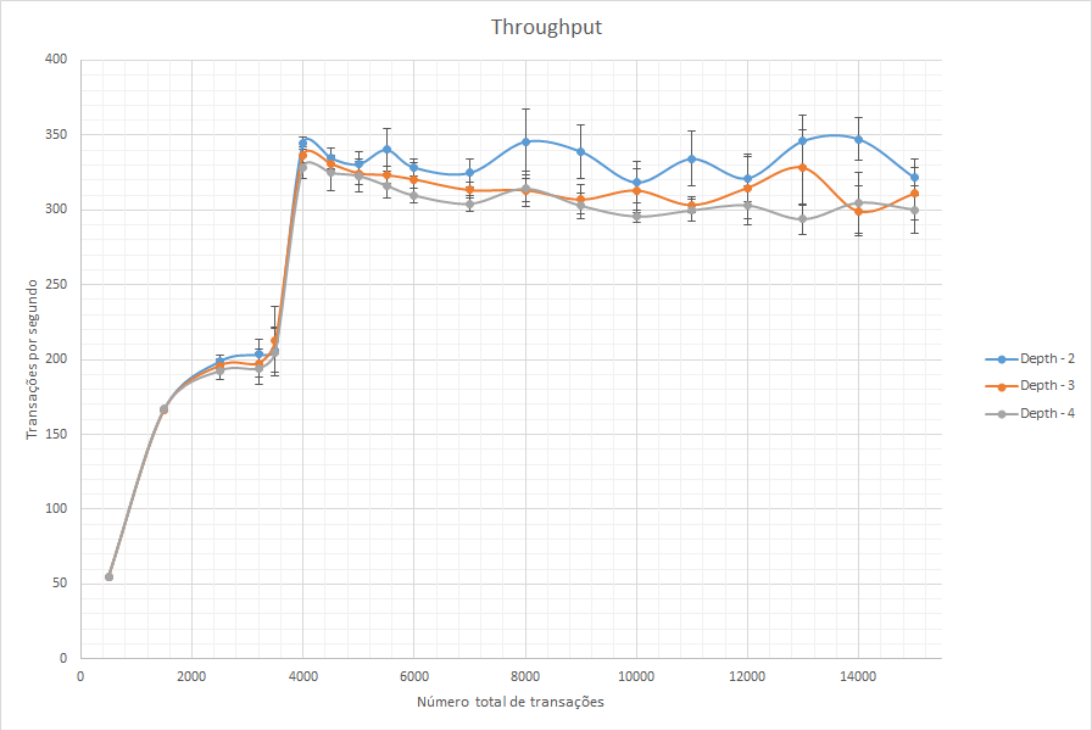


Figura 71 - Comparação dos valores de Throughput para operações de escrita em chaincodes no mesmo canal, com níveis de profundidade 2, 3 e 4 – T9F

A Latência, apresentada na Figura 72, mostra novamente a tendência observada na Figura 71, onde o nível de profundidade 2, apresenta o valor médio mais baixo de aproximadamente 3.2 milissegundos, contrastando com os valores de 3.4 e 3.5 milissegundos para os níveis 3 e 4, respetivamente.

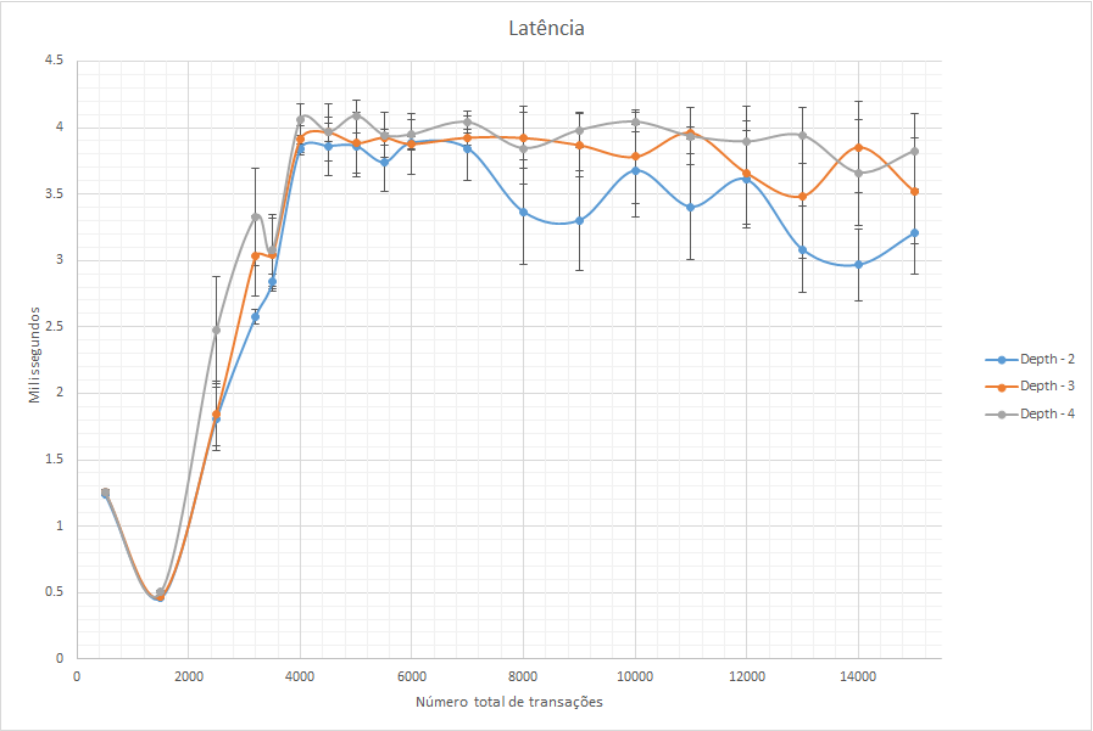


Figura 72 - Comparação dos valores de Latência para operações de escrita em chaincodes no mesmo canal, com níveis de profundidade 2, 3 e 4 – T9F

Na Figura 73 observamos a comparação dos valores do consumo de CPU, e é aqui que os diferentes níveis de invocação apresentam diferenças significativas, com o nível 4 a ser o que mais consome CPU, um comportamento expectável uma vez que, à medida que a profundidade aumenta, a complexidade de criação de uma transação também aumenta, uma vez que o *peer* tem de percorrer todos os *chaincodes* até ao nível a em teste.

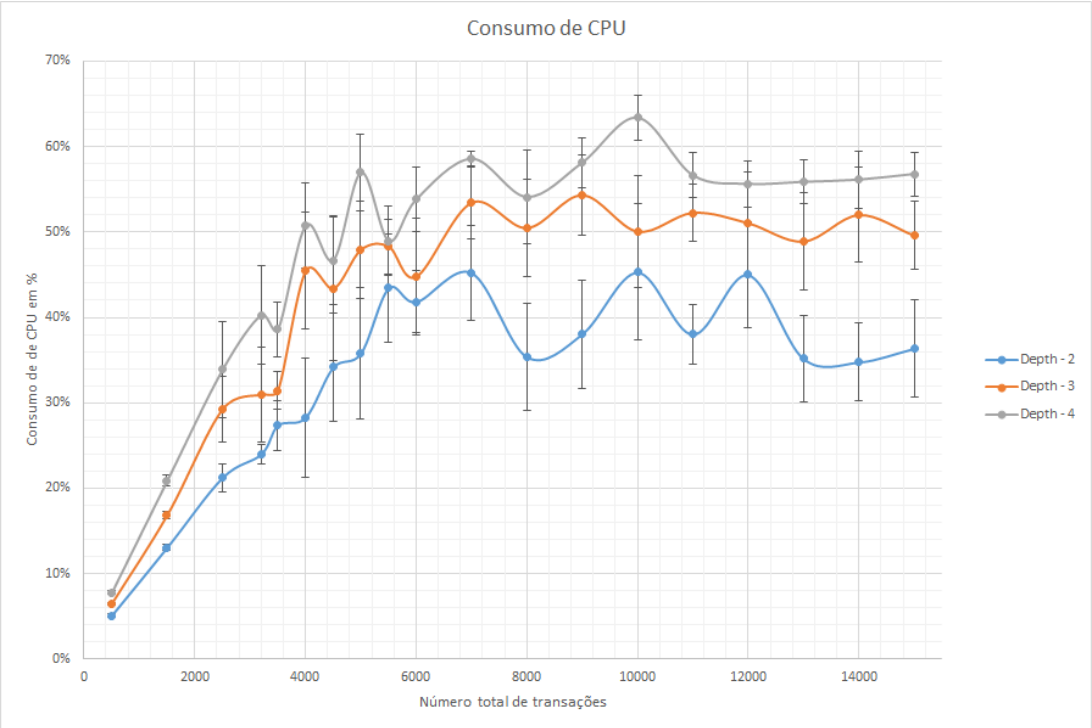


Figura 73 - Comparação dos valores de consumo de CPU para operações de escrita em chaincodes no mesmo canal, com níveis de profundidade 2, 3 e 4 – T9F

Na Figura 74 observamos a comparação dos valores do consumo de memória RAM, e concluímos que para os níveis 2 e 3, estes estão bastante próximos. No entanto, o consumo de memória RAM para o nível de invocação 4 passa a ser maior a partir do intervalo de 6000 transações.

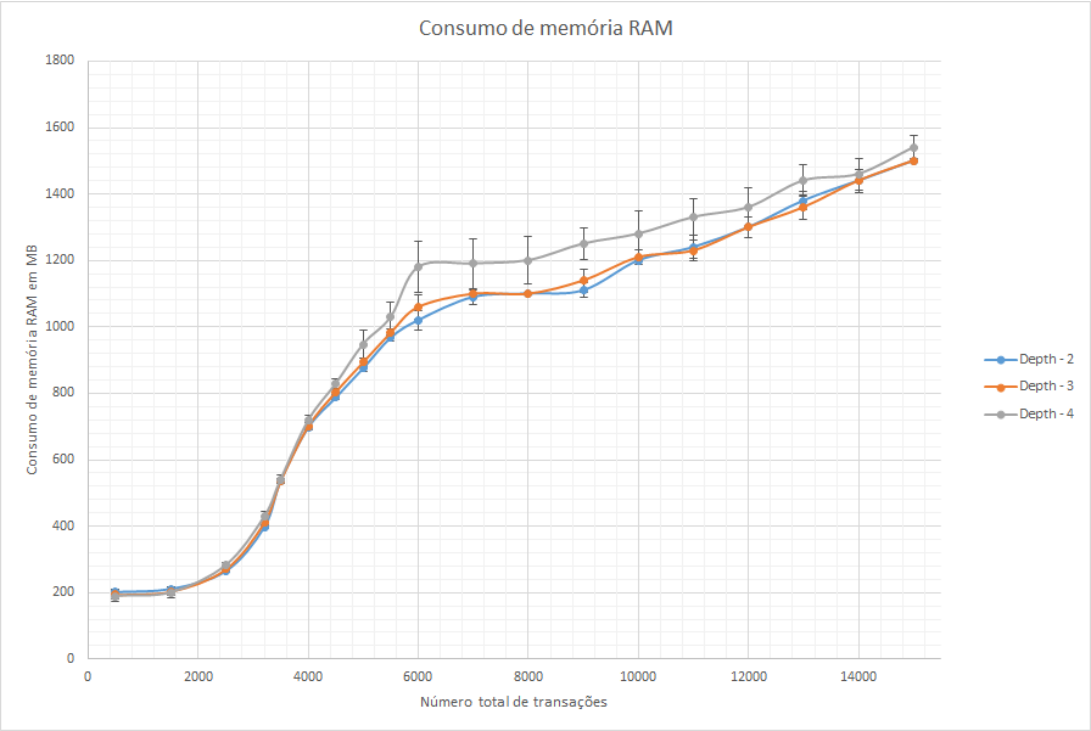


Figura 74 - Comparação dos valores de consumo de memória RAM para operações de escrita em chaincodes no mesmo canal, com níveis de profundidade 2, 3 e 4 – T9F

Em suma, após a conclusão da execução do teste T9F, concluímos que o nível de invocação para operações de escrita a diversos *chaincodes* instanciados no mesmo canal, as chamadas de profundidade 2 apresentam os melhores valores de *Throughput* e Latência. De igual forma, é observado que à medida que o nível de profundidade aumenta, a complexidade de gerar uma transação também aumenta, e consequentemente o consumo de CPU. Para o consumo de memória RAM, a diferença entre os valores não é significativa.

## 7.10 Teste T10F

O teste T10F foi desenhado com o principal objetivo de avaliar os valores de *Throughput* e Latência da plataforma *Hyperledger Fabric*, quando ao mesmo *chaincode* são invocadas operações de escrita com diversos níveis de profundidade. De forma semelhante ao observado na secção 7.6, o intervalo de amostras para este teste compreende-se entre as 0 e as 15000 transações. A rede *blockchain* utilizada compreende uma organização (*Org1*), um *peer* (*peer1*), um canal e um *chaincode* instanciado nesse canal. Os valores utilizados para a profundidade das chamadas do *chaincode* compreendem-se entre 1 e 4.

Para a execução do teste T10F, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 75. O parâmetro *"n_chaincodes"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "single-chaincode-nested-calls",
  "limit": 1500,
  "executions": 10,
  "type": "micro",
  "n_chaincodes": 1
}
```

Figura 75 - Excerto do ficheiro de configuração para a execução do teste T10F

Na Figura 76, observamos a comparação dos valores de *Throughput* para os diferentes níveis de profundidade. Dessa comparação, concluímos que estes valores se encontram bastante próximos uns dos outros, apresentando um crescimento no intervalo inicial até às 200 *TPS*, voltando a crescer repentinamente no intervalo de 4000 transações, onde acabam por estabilizar por volta das 350 *TPS*.

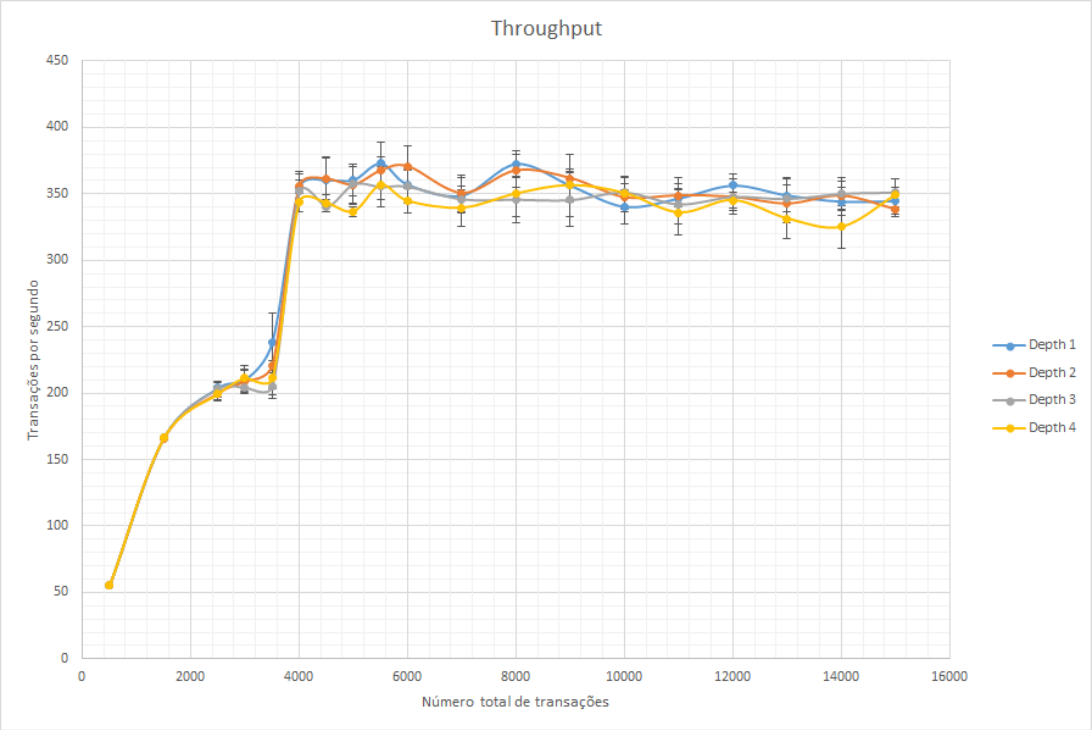


Figura 76 - Comparação dos valores de Throughput em chamadas dentro do mesmo chaincode, com níveis de profundidade 1, 2, 3 e 4 – T10F

Na Figura 77 observamos a comparação dos valores de Latência. Ainda que a diferença entre valores seja mínima, é possível observar que a Latência aumenta de acordo com o aumento da profundidade da invocação.

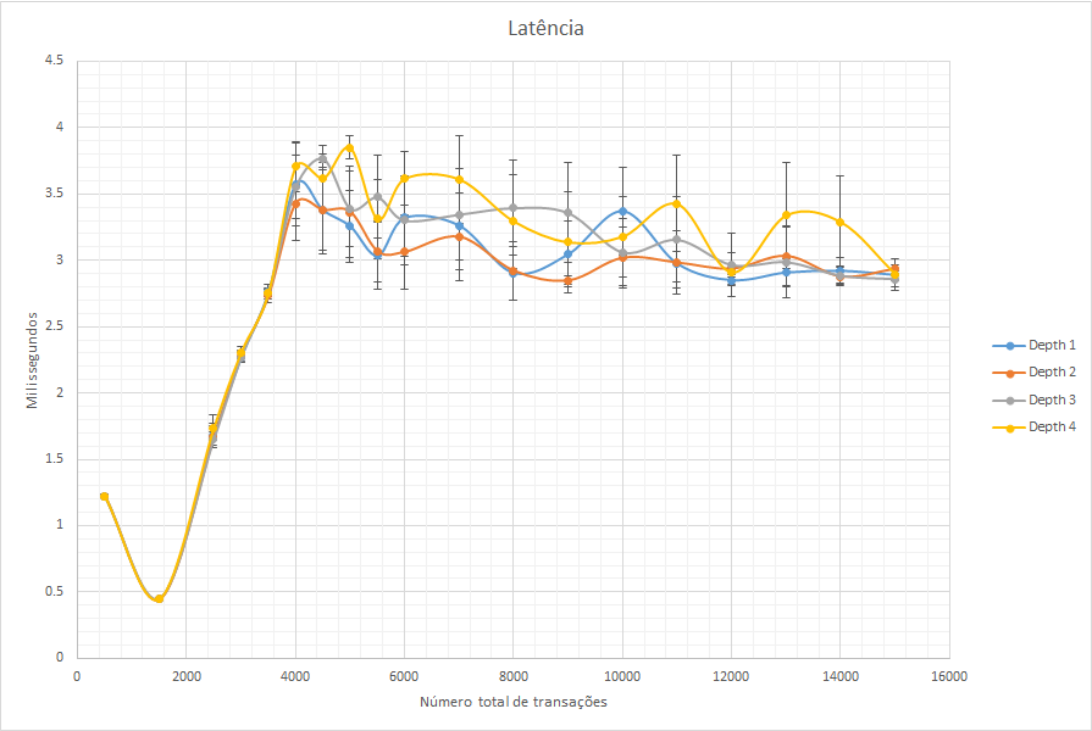


Figura 77 - Comparação dos valores de Latência em chamadas dentro do mesmo chaincode, com níveis de profundidade 1, 2, 3 e 4 – T10F

Uma vez mais, é na comparação dos valores de consumo de CPU, representada na Figura 78, que se observa a maior diferença entre valores, com a invocação de nível 2 a apresentar um melhor desempenho.

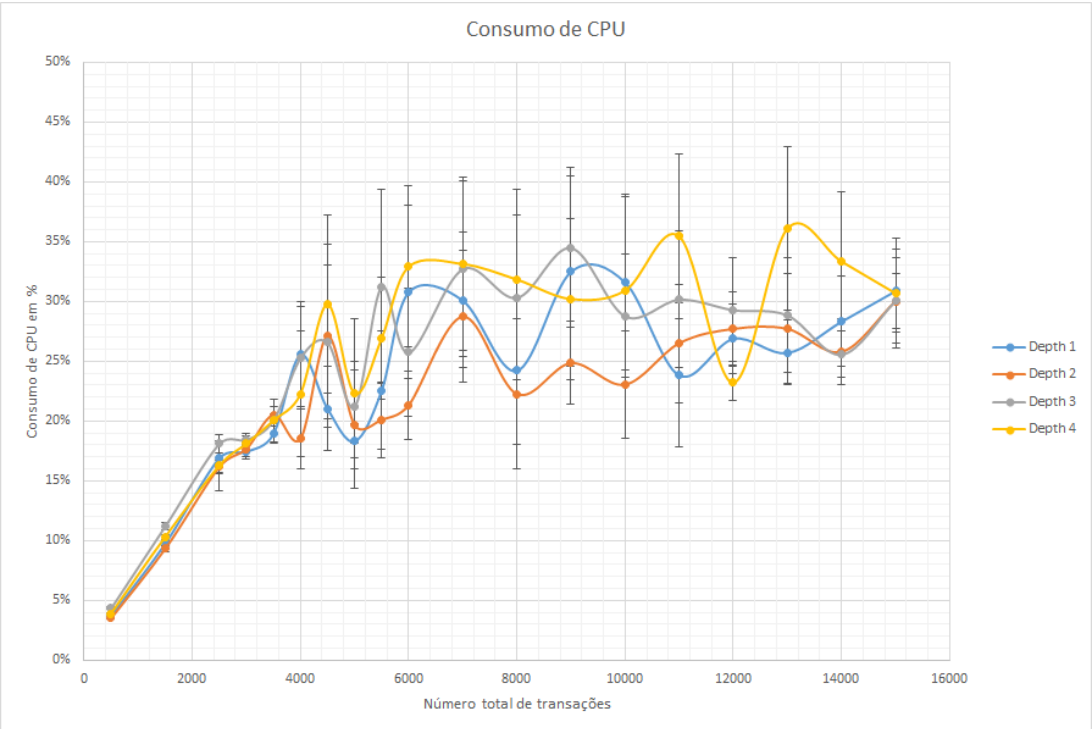


Figura 78 - Comparação dos valores de consumo de CPU em chamadas dentro do mesmo chaincode, com níveis de profundidade 1, 2, 3 e 4 – T10F

A comparação para os valores de consumo de memória RAM foi realizada na Figura 79, não apresentando, no entanto, esta não apresenta nenhum ponto de destaque, uma vez que os valores para os diferentes níveis de invocação são praticamente idênticos.

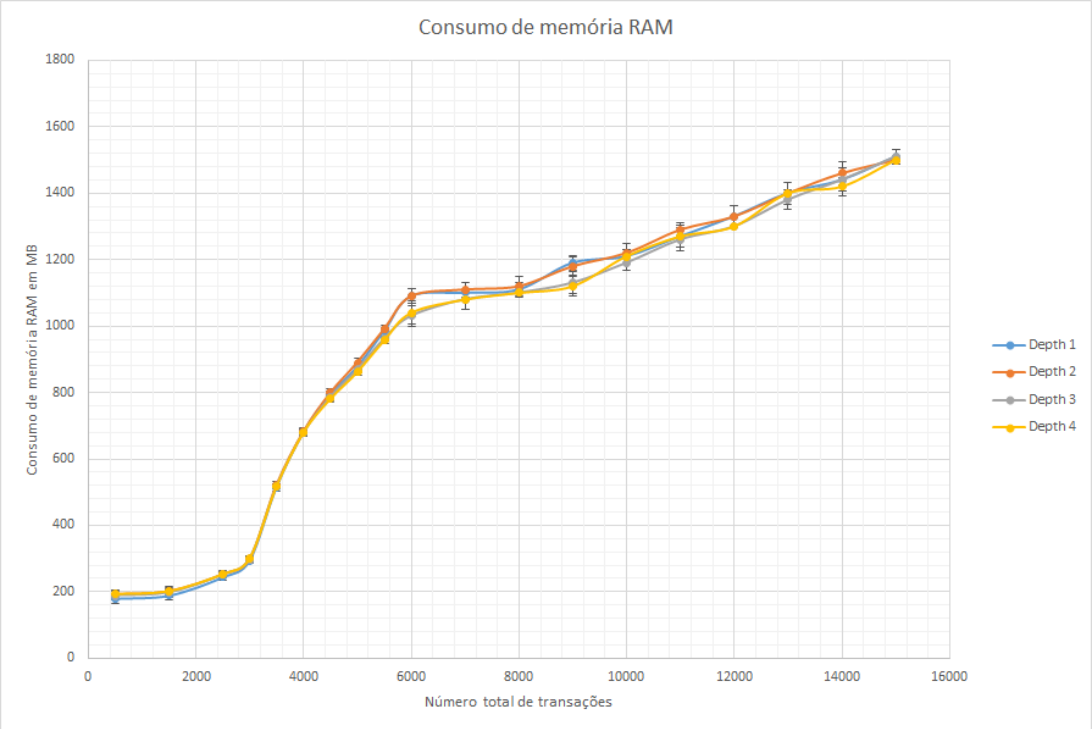


Figura 79 - Comparação dos valores de consumo de memória RAM em chamadas dentro do mesmo chaincode, com níveis de profundidade 1, 2, 3 e 4 – T10F

Da execução do teste T10F foi possível concluir que ainda que não seja muito grande, a diferença nos valores de *Throughput* e Latência para os níveis de invocação de 1 a 4 existe.

## 7.11 Teste T11F

O teste T11F foi desenhado com o principal objetivo de avaliar o impacto que o aumento dos *chaincodes* tem nos valores de *Throughput* e Latência da plataforma *Hyperledger Fabric*. A rede *blockchain* utilizada compreende duas organizações (*Org1* e *Org2*), com dois *peers* cada (*peer1* e *peer2*), um canal e um total de 20 *chaincodes* instanciado nesse canal. Após instanciados os *chaincodes* é enviada uma transação para todos em simultâneo. O intervalo de amostras compreendeu-se entre um total de 0 e 500 transações, com intervalos de 50 transações. O número total de *chaincodes* instanciados no canal foi imposto por limitação da rede *blockchain* em teste ou da plataforma *Gauge*, uma vez que acima desse valor, o *script* de instanciação dos *chaincodes* apresentava erros e não permitia finalizar o processo de execução do teste. Do ponto de vista crítico do aluno, o limite de 20 *chaincodes* por canal é uma ideia que não faz sentido e que pode ser descartada uma vez que, na documentação oficial da plataforma *Hyperledger Fabric* não são conhecidos limites teóricos para o tamanho de uma rede *blockchain* e dos seus componentes (The Linux Foundation, 2019l), levando a concluir que a limitação é imposta pela ferramenta *Gauge*. Por essa razão, os resultados deste teste são descartados.

## 7.12 Teste T12F

O teste T12F foi desenhado com o principal objetivo de avaliar o impacto que o aumento dos canais tem nos valores de *Throughput* e Latência da plataforma *Hyperledger Fabric*. A rede *blockchain* utilizada compreende duas organizações (*Org1* e *Org2*), com dois *peers* cada (*peer1* e *peer2*), 50 canais e um um *chaincode* instanciado por cada canal. Após criados os canais e instanciados os *chaincodes* em cada um desses canais, é enviada uma transação com uma operação de escrita na *blockchain* para todos em simultâneo. O intervalo de amostras compreendeu-se entre 0 e 500 transações, com intervalos de 50 transações.

Para a execução do teste T12F, o utilizador deve modificar o ficheiro de configuração com os parâmetros apresentados na Figura 80. O parâmetro *"limit"* não é utilizado no contexto deste teste.

```
"benchmark": {
  "blockchain": "fabric",
  "workload": "channel-scalability",
  "limit": 1500,
  "executions": 10,
  "type": "scalability",
  "n_chaincodes": 50
}
```

Figura 80 - Excerto do ficheiro de configuração para a execução do teste T12F

Na Figura 81 observamos a comparação dos valores de *Throughput* quando são realizadas transações em diversos canais em simultâneo. Para este teste em específico são realizadas chamadas no intervalo de 5 a 50 canais, com intervalos de 5 canais. Podemos então concluir que, para chamadas abaixo dos 25 canais em simultâneo, o *Throughput* é baixo, oscilando entre as 0 e as 60 *TPS*. Os maiores valores de *Throughput* verificaram-se entre as chamadas a 30 e 45 canais em simultâneo, estabilizando por valores perto das 300 *TPS*. São exceção os valores para 25 e 50 canais em simultâneo, que por sua vez apresentam uma performance intermédia quando comparada com os restantes valores.

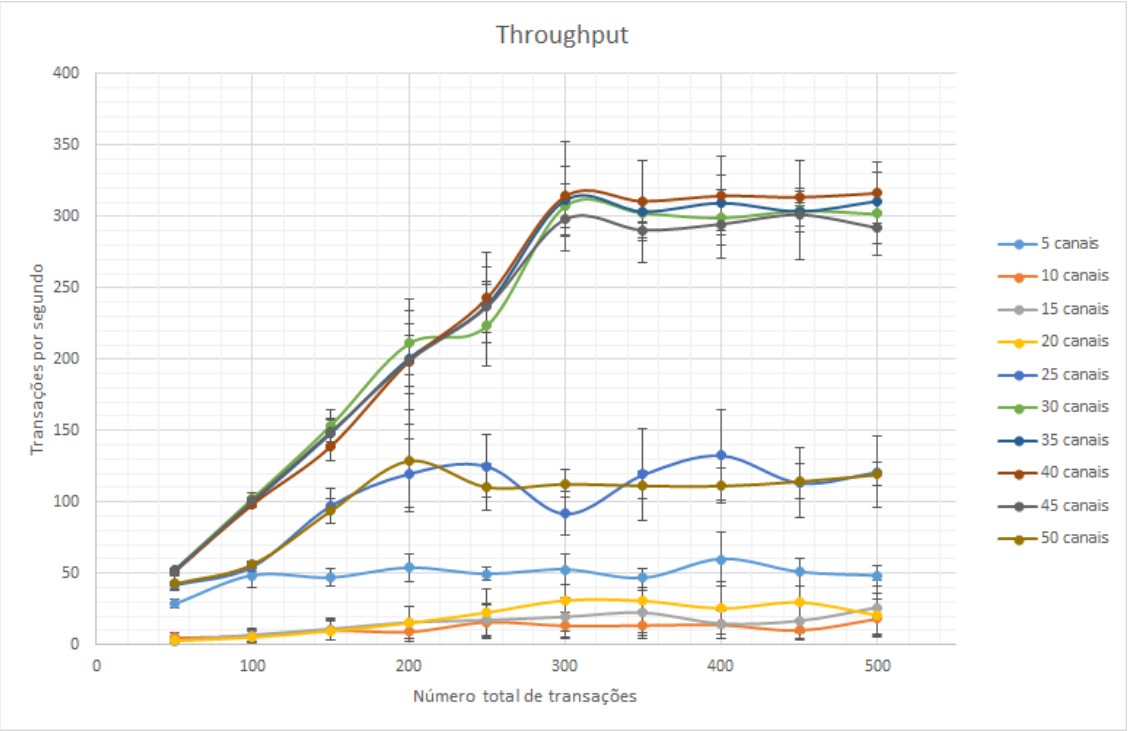


Figura 81 - Comparação dos valores de Throughput para chamadas no intervalo de 5 a 50 canais em simultâneo – T12F

A Latência, representada na Figura 82, apresenta valores baixos para chamadas a 10, 15 e 20 canais em simultâneo. Os valores aumentam quando nos encontramos no intervalo entre os 25 e 50 canais. Uma vez mais, surge uma exceção, para o intervalo compreendido por 5 canais em simultâneo, onde a Latência apresenta os valores mais altos de todos os testes executados, com um valor médio de aproximadamente 260 milissegundos.

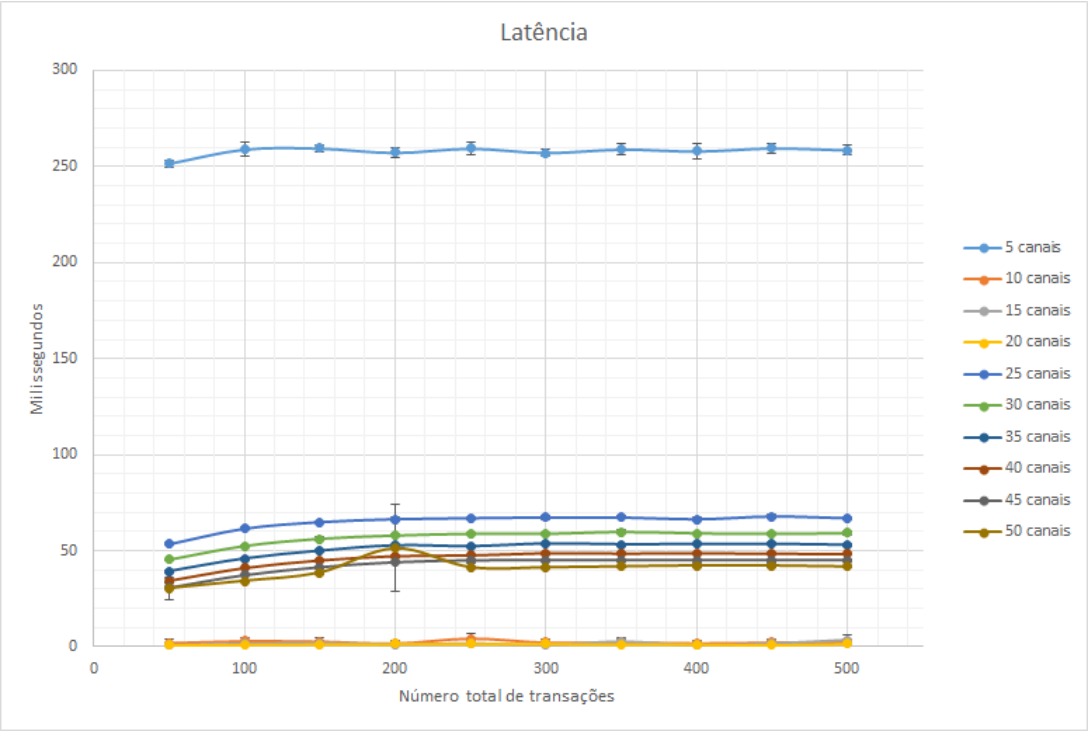


Figura 82 - Comparação dos valores de Latência para chamadas no intervalo de 5 a 50 canais em simultâneo – T12F

O autor (Ferris, 2019), publicou recentemente resultados de testes de escalabilidade da plataforma *Hyperledger Fabric*. No entanto, o autor não especifica a metodologia usada, referenciando apenas a composição da rede utilizada. A rede foi então composta por 128 *peers* e 325 canais, obtendo valores máximos de aproximadamente 13000 *TPS*. Não sendo conhecidos mais pormenores sobre quais os testes realizados e o processo da sua execução, torna-se impraticável a comparação desses valores com os valores obtidos pela execução de T12F.

## 7.13 Conclusões gerais da execução do benchmark

Da execução dos testes de *benchmark* definidos no Capítulo 6, é possível retirar algumas conclusões. A arquitetura e modelo de transações da plataforma *Hyperledger Fabric origina* a que esta apresente um melhor desempenho para operações de leitura sobre a *blockchain* quando comparadas com operações de escrita. Enquanto que as operações de leitura na *blockchain* são praticamente imediatas, quando é realizada uma operação de escrita, esta tem que passar por cada uma das três fases de *execute-order-validate*, reduzindo assim o *Throughput* e aumentando a Latência da plataforma. Operações nulas, ainda que tenham que passar pelo mesmo processo que as operações de escrita, apresentam uma melhor performance relativamente a estas, o que seria de esperar, uma vez que não são realizadas operações na *blockchain*.

Para os testes T1F e T3F, observou-se um ponto de interesse compreendido pelo intervalo entre as 3500 e 4000 transações, onde os valores de *Throughput* para estes testes atingiram os

seus valores máximos. Uma possível causa para esse comportamento pode estar na origem de no mesmo intervalo haver uma falha no número total de transações bem-sucedidas.

A modificação dos parâmetros do *Orderer*, com especial foco no tamanho do bloco gerado, revelou aumentar a performance da plataforma *Hyperledger*, como já tinha sido estudado por outros autores, nomeadamente (Androulaki et al., 2018). De igual forma, foi verificada uma melhoria na performance quando modificada a política de aprovação de uma transação. No entanto, esta não deve ser utilizada para esse propósito, apresentando-se como um ponto de falha no sistema.

Usar diferentes tamanhos para cargas enviadas como argumento a um *chaincode*, demonstrou não ter grande efeito nos valores de *Throughput* e Latência, com estes a apresentar valores muito semelhantes para os diferentes tamanhos de carga. Essa diferença de valores observa-se quando comparados os valores de consumo de memória RAM, justificando-se pelo facto de a rede sobre qual foi feita a execução, ser composta por apenas uma organização e um *peer.* Aproveitando os diferentes tamanhos de cargas passadas por argumento, a utilização de eventos por parte de um *chaincode* revelou não influenciar os valores de *Throughput* e Latência. Contrariamente, a diferença nos valores de consumo de memória RAM e consumo de CPU voltou a ser observada.

Quando comparadas as operações de leitura a *chaincodes* intra-channel e inter-channel, observamos que no geral as chamadas intra-channel apresentam melhores valores de *Throughput* e Latência. Nas operações de escrita para *chaincodes* instanciados no mesmo canal, as invocações de nível 2 apresentaram uma melhor performance geral, comportamento expectável, uma vez que ao aumentar o nível de invocação, aumenta a complexidade da chamada. Esta tendência volta a ser observada no teste T10F, onde as chamadas passam a ser feitas a apenas um *chaincode*.

Ao avaliar a performance da plataforma relativamente à sua escalabilidade, considerou-se o teste T11F, como teste falhado, uma vez que este foi limitado pelo número de *chaincodes* instanciados no canal, característica que não pode ser justificada uma vez que não são conhecidos limites para o tamanho da rede (The Linux Foundation, 2019l). Nos testes de escalabilidade dos canais, observou-se que a rede atinge valores mais altos de *Throughput* quando são feitas chamadas em simultâneo ao intervalo de *chaincodes* de 30 a 45. Neste teste, a Latência apresentou um comportamento diferente, com o número de chamadas em simultâneo mais baixo, 5, a ter um maior valor quando comparado com os restantes. Os melhores valores de Latência são observados quando são realizadas chamadas em simultâneo no intervalo entre 10 e 20 chamadas.

# Capítulo 8
# Conclusão

O presente documento descreve o trabalho realizado pelo aluno no âmbito da dissertação *"Análise de Plataformas Blockchain"*. Nos parágrafos seguintes, é apresentado em jeito de resumo as tarefas desempenhadas ao longo dos dois semestres, período que compreendeu a realização da dissertação.

Assim, no primeiro momento, pretendeu-se familiarizar o leitor com os conceitos abrangentes da tecnologia *Blockchain*, sendo apresentadas definições para os seus componentes integrantes, como por exemplo o *ledger* e o bloco, passando para a distinção entre o tipo de *blockchains* públicas e privadas. De seguida, foram apresentados em detalhe três mecanismos de consenso, dois implementados por *blockchains* públicas e um por *blockchains* privadas, terminando com uma comparação geral sobre os mesmos, motivada por (Yaga et al., 2018). O estudo termina com a apresentação do conceito de *smart contract*, e com a apresentação de diversos riscos e limitações associados à tecnologia *Blockchain*.

Continuando, foram estudadas quatro plataformas que implementam a tecnologia *Blockchain*, nomeadamente a *Bitcoin*, *Ethereum*, *Quorum*, e *Hyperledger Fabric*, tendo sido apresentadas a sua arquitetura, funcionamento e componentes, com especial detalhe para as duas últimas, que, são plataformas do domínio de *blockchain* privada, e representam o foco do trabalho do aluno. Do estudo foi possível a comparação entre as diversas plataformas apresentadas, tendo em conta os diversos parâmetros propostos por (Beyer, 2016; Dinh et al., 2017; Fersht, 2018; LeewayHertz, 2018).

O segundo momento da dissertação foi desenvolvido ao longo do segundo semestre. Este, começou pela análise aos trabalhos existentes na área de *benchmarking* a plataformas *Blockchain*. Essa análise revelou-se importante, uma vez que permitiu ao aluno saber o que já tinha sido desenvolvido, e tomar uma decisão sobre o trabalho a desenvolver. Foi tomada a decisão de dar continuidade ao trabalho *Gauge*, pois de entre os trabalhos estudados anteriormente, este era o que mais se adequava aos requisitos procurados pelo aluno, nomeadamente, a idade do projeto, a facilidade de perceção do seu código e funcionamento, as métricas que este recolhia, e o facto do projeto ser *OpenSource*. A plataforma foi, posteriormente, alvo de modificações por parte do aluno, podendo destacar-se a mudança no *output* de geração de resultados, a criação de um novo *script* de compilação e *deployment* na rede *Quorum*, um *update* nos componentes da rede *Hyperledger*, a criação de um *script* para a geração dos componentes criptográficos da rede *Hyperledger* e um *update* ao componente *Blockchain SDK*. No entanto, embora tenha sido iniciada, a última modificação não foi terminada.

Foi desenhado e implementado um sistema de *benchmark*, para a automação da execução dos testes, definidos através da adaptação dos testes apresentados pela plataforma *Gauge.* Com a utilização desse sistema, foi possível realizar o *benchmarking* à plataforma *Hyperledger Fabric*, terminando com uma comparação da sua performance tendo em conta diversas configurações. Dos resultados obtidos retiraram-se algumas conclusões, apoiadas pela arquitetura e modo de operação da plataforma *Hyperledger Fabric*.

Como conclusão, considera-se que não foram cumpridos na totalidade os objetivos iniciais da dissertação, uma vez que os testes de *benchmark* foram apenas realizados à plataforma *Hyperledger Fabric*. Do trabalho desenvolvido, foi possível ao aluno familiarizar-se com a tecnologia *Blockchain* e práticas de testes de *benchmarking*, permitindo que fossem aplicados diversos conhecimentos adquiridos ao longo do seu percurso académico. Uma das principais dificuldades encontradas pelo aluno prendeu-se na questão temporal, uma vez que todo o processo desde à instalação das plataformas, adaptação dos testes, e a sua execução, demorou muito mais tempo que o planeado, influenciando dessa forma muitas das decisões tomadas ao longo do ano. Outra dificuldade encontrada, é que ao ser uma tecnologia recente, a escassez de utilizadores que estão a desenvolver aplicações sobre a mesma é evidente, dificultando a possibilidade de discussão e resolução de possíveis erros encontrados ao longo do caminho.

## 8.1 Trabalho futuro

Na continuação do trabalho desempenhado ao longo desta dissertação, há vários aspetos que podem ser abordados para trabalho futuro. São eles:

- **Redesenhar os *smart contracts* de operações de escrita na *blockchain***

Numa primeira abordagem ao trabalho desenvolvido, poderão ser redesenhados os *smart contracts* que implementam os testes de operações de escrita na *blockchain*, fazendo-lhes um *update* para funções que requeiram um maior trabalho computacional. Recordo que para os testes executados, estes desempenhavam uma simples inserção numa estrutura de dados. Com a utilização de funções mais complexas será possível realizar novamente a comparação entre dos novos resultados com os resultados obtidos com este trabalho. De igual modo seria possível comparar e observar as diferenças quando as novas operações de escrita fossem novamente comparadas com as operações nulas.

- **Reexecutar os testes utilizando diversas combinações da rede *Hyperledger Fabric***

Outra sugestão, que por questões temporais não pôde ser explorada é a combinação das diversas configurações para a plataforma *Hyperledger Fabric*. Nos testes realizados, o aluno apenas modificou um parâmetro do componente *Orderer* da rede. Este, no entanto, é altamente modificável, e utilizando uma combinação dos diversos parâmetros que este oferece seria possível analisar o impacto destes na performance da plataforma.

- **Uso de *CouchDB* para persistência de dados**

Os testes executados no presente documento utilizaram uma rede que implementava *LevelDB* para a persistência de dados no *ledger*. Uma sugestão para trabalho futuro, é a re-execução dos testes à plataforma *Hyperledger Fabric*, optando por usar uma rede que implemente *CouchDB*, e, comparar de que forma a base de dados utilizada para a persistência de dados influencia a performance da plataforma.

- **Terminar a modificação 3**

Por fim, a última sugestão é a possibilidade de terminar o processo da modificação 3 à plataforma *Gauge*, iniciado pelo aluno. Terminada essa modificação, será possível a execução dos testes à plataforma *Quorum*, e dessa forma realizar uma comparação da performance desta com a plataforma *Hyperledger Fabric.*

# Referências


Androulaki, E., Barger, A., Bortnikov, V., Muralidharan, S., Cachin, C., Christidis, K., … Yellick, J. (2018). Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, *2018-Janua*(1). https://doi.org/10.1145/3190508.3190538

Atomic Wallet. (2019). Atomic Wallet. Retrieved January 19, 2019, from https://atomicwallet.io/

Beyer, S. (2016). Enterprise Blockchain Platforms — A Comparison. Retrieved December 29, 2018, from Enterprise Blockchain Platforms — A Comparison website: https://medium.com/blackinsurance/enterprise-blockchain-platforms-a-comparison-d58f1227ce70

BigchainDB. (2018). *BigchainDB*.

Bitcoin Project. (2018a). Developer Guide - Bitcoin. Retrieved December 19, 2018, from https://bitcoin.org/en/developer-guide#stratum

Bitcoin Project. (2018b). System Requirements. Retrieved December 19, 2018, from Bitcoin Core Requirements and Warnings website: https://bitcoin.org/en/bitcoin-core/features/requirements

Bitcoin Project. (2019). Bitcoin - Open source P2P money. Retrieved January 19, 2019, from https://bitcoin.org/en/

Blockchain Luxembourg. (2017). *Block #1*.

BlockchainHub. (2018). What's a blockchain oracle? Information oracles. Retrieved December 7, 2018, from https://blockchainhub.net/blockchain-oracles/

Brakeville, S., & Perepa, B. (2018). Blockchain basics: Introduction to distributed ledgers – IBM Developer. Retrieved November 12, 2018, from https://developer.ibm.com/tutorials/cl-blockchain-basics-intro-bluemix-trs/

Buck, J. (2017). Blockchain Oracles, Explained | Cointelegraph. Retrieved December 7, 2018, from https://cointelegraph.com/explained/blockchain-oracles-explained

Buterin, V. (2015). On Public and Private Blockchains. Retrieved November 2, 2018, from On Public and Private Blockchains website: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/

Cambridge University Press. (2019). LEDGER. *LEDGER | Meaning in the Cambridge English Dictionary*.

Chen, T., Li, X., Luo, X., & Zhang, X. (2017). Under-optimized smart contracts devour your money. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 442–446. https://doi.org/10.1109/SANER.2017.7884650

Christidis, K., & Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, *4*, 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339

Coin Sciences. (2019). MultiChain | Open source blockchain platform. Retrieved January 19, 2019, from https://www.multichain.com/

Coinprism. (2015). Openchain - Blockchain technology for the enterprise. Retrieved January 19,

2019, from https://www.openchain.org/

Curran, B. (2018). BLOCKONOMI. Retrieved November 16, 2018, from What is a Merkle Tree? Beginner's Guide to this Blockchain Component website: https://blockonomi.com/merkle-tree/

de Vries, A. (2018). Bitcoin's Growing Energy Problem. *Joule*, *2*(5), 801–805. https://doi.org/10.1016/j.joule.2018.04.016

Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., & Wang, J. (2018). Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Transactions on Knowledge and Data Engineering*, *30*(7), 1366–1385. https://doi.org/10.1109/TKDE.2017.2781227

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., & Tan, K.-L. (2017). BLOCKBENCH. *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, 1085–1100. https://doi.org/10.1145/3035918.3064033

Ethereum. (2018). Solidity — Solidity 0.5.2 documentation. Retrieved December 4, 2018, from Ethereum Revision 8a9e0127 website: https://solidity.readthedocs.io/en/latest/index.html

Ethereum Community. (2016). Ethereum Homestead Documentation — Ethereum Homestead 0.1 documentation. *Ethereum Homestead Documentation, Revision 41fc2c03*. Retrieved from http://www.ethdocs.org/en/latest/index.html

Ethereum Community. (2017). Ethereum Homestead Documentation. *Github*, https://www.ethereum.org/. https://doi.org/10.1002/asia.201200834

Ethereum Foundation. (2018). Mist Browser. Retrieved from https://github.com/ethereum/mist

Ethereum Foundation. (2019a). Ethereum. Retrieved from Ethereum website: https://www.ethereum.org/

Ethereum Foundation. (2019b). Ethereum White Paper. Retrieved from White Paper website: https://github.com/ethereum/wiki/wiki/White-Paper

Ferris, C. (2019). Does Hyperledger Fabric perform at scale? Retrieved from https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale/

Fersht, P. (2018). The top 5 enterprise blockchain platforms you need to know about. Retrieved December 29, 2018, from The top 5 enterprise blockchain platforms you need to know about website: https://www.horsesforsources.com/top-5-blockchain-platforms_031618

Frankenfield, J. (2017). Proof of Activity. Retrieved November 28, 2018, from Proof of Activity (Cryptocurrency) website: https://www.investopedia.com/terms/p/proof-activity-cryptocurrency.asp

Frankenfield, J. (2018). Proof of Burn. Retrieved January 9, 2019, from Proof of Burn (Cryptocurrency) website: https://www.investopedia.com/terms/p/proof-burn-cryptocurrency.asp

Furter, S. (2019). Web3.js v1.0.0-beta.38. Retrieved from Web3.js v1.0.0-beta.38 website: https://medium.com/@samuel_91690/web3-js-v1-0-0-beta-38-8d2bb6e73d0b

Granjal, J. (2017). *SEGURANÇA PRÁTICA EM SISTEMAS E REDES COM LINUX* (1st ed.; L. FCA - Editora de Informática, Ed.). Lidel - Edições Técnicas, Lda.

Gronholt-Pedersen, J. (2018). Maersk, IBM to launch blockchain-based platform for global trade. Retrieved from Reuters website: https://www.reuters.com/article/us-maersk-blockchain-ibm/maersk-ibm-to-launch-blockchain-based-platform-for-global-trade-idUSKBN1F51DE

Gupta, M. (2017). *Blockchain for Dummies, IBM Limited Edition* (Vol. 102).

Heap, I. (2017). Blockchain Could Help Musicians Make Money Again. *Harvard Business Review, Harvard Business School Publishing Corportation*, 2–7. Retrieved from https://hbr.org/2017/06/blockchain-could-help-musicians-make-money-again

Hertig, A. (2018). How Ethereum Works. Retrieved from How Ethereum Works website: https://www.coindesk.com/information/how-ethereum-works

Huang, D., Ma, X., & Zhang, S. (2018). *Performance Analysis of the Raft Consensus Algorithm for Private Blockchains*. 1–7.

HydraChain. (2017). *HydraChain*.

Iansiti, M., & Lakhani, K. R. (2017). The truth about blockchain. *Harvard Business Review*. https://doi.org/10.1016/j.annals.2005.11.001

Jaeger, L. G. (2018). Blockchain Explained. *Public versus Private: What to Know before Getting Started with Blockchain*.

Jayachandran, P. (2017). Blockchain Explained. Retrieved November 22, 2018, from The difference between public and private blockchain website: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/

JPMorgan Chase & Co. (2016). Quorum Network. Retrieved from A permissioned implementation of Ethereum supporting data privacy website: https://drive.google.com/file/d/0B8rVouOzG7cOeHo0M2ZBejZTdGs/view

JPMorgan Chase & Co. (2019). Quorum.

Kuhnert, D. (2019). Hyperledger Fabric Events. Retrieved from Subscribe to Hyperledger Fabric Chaincode Events website: https://developers.sap.com/tutorials/blockchain-hlf-chaincode-events.html#

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21*(7), 558–565. https://doi.org/10.1145/359545.359563

Learn Cryptocurrency. (2019). 51% Attack - Learn Cryptocurrency. Retrieved December 7, 2018, from 51% Attack website: https://learncryptography.com/cryptocurrency/51-attack

Ledger SAS. (2019). Ledger. Retrieved January 19, 2019, from Ledger - Hardware wallets - Securing your crypto assets | Ledger website: https://www.ledger.com/?r=c06d

LeewayHertz. (2018). Best Blockchain Platforms. Retrieved December 29, 2018, from Best Blockchain Platforms website: https://www.leewayhertz.com/blockchain-platforms-for-top-blockchain-companies/

Leppelsack, H. F. (2018). *Experimental Performance Evaluation of Private Distributed Ledger Implementations*. Technical University of Munich.

Li, X., Jiang, P., Chen, T., Luo, X., & Wen, Q. (2017). A survey on the security of blockchain systems. *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2017.08.020

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making Smart Contracts Smarter. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 254–269. https://doi.org/10.1145/2976749.2978309

Marr, B. (2018). How Blockchain Will Transform The Supply Chain And Logistics Industry. *Forbes*, 3–7.

Minkenberg, M. (2015). Transforming the Transformation? In M. Minkenberg (Ed.), *Supply & Demand Chain Executive*. https://doi.org/10.4324/9781315730578

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*. https://doi.org/10.1007/s10838-008-9062-0

Nasir, Q., Qasse, I. A., Abu Talib, M., & Nassif, A. B. (2018). Performance Analysis of Hyperledger Fabric Platforms. *Security and Communication Networks*, *2018*, 1–14. https://doi.org/10.1155/2018/3976093

Nomadic Labs. (2018). Tezos Developer Documentation. Retrieved December 30, 2018, from Welcome to the Tezos Developer Documentation! website: https://tezos.gitlab.io/master/#

Ongaro, D., & Ousterhout, J. (2014). In Search of an Understandable Consenus Algorithm (Extended Version). *Proceedings of USENIX ATC '14*, 305–319. https://doi.org/10.1145/1529974.1529978

Persistent Systems. (2018). Workloads Gauge. Retrieved from Run Benchmarks For Hyperledger Fabric website: https://github.com/ruipedrodias94/gauge/blob/master/docs/running-expts.md

Persistent Systems. (2019a). Anatomy of Gauge. Retrieved from Tool Anatomy website: https://github.com/ruipedrodias94/gauge/blob/master/docs/tool-anatomy.md

Persistent Systems. (2019b). Gauge Architecture. Retrieved from https://github.com/persistentsystems/gauge/blob/master/docs/architecture.png

Persistent Systems. (2019c). Persistent Systems. Retrieved from https://github.com/persistentsystems/gauge

Persistent Systems. (2019d). *Persistent Systems*.

Persistent Systems. (2019e). Simple Solidity. Retrieved from https://github.com/ruipedrodias94/gauge/blob/master/deployment-script/quorum/contracts/simple/simple.sol

Pongnumkul, S., Siripanpornchana, C., & Thajchayapong, S. (2017). Performance Analysis of Private Blockchain Platforms in Varying Workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 1–6. https://doi.org/10.1109/ICCCN.2017.8038517

R3. (2018). R3 - Corda Platform. Retrieved January 19, 2019, from https://www.r3.com/corda-platform/

Ripple. (2019). *Ripple*.

Rutland, E. (2018). The Blockchain Byte features a question from the distributed ledger space. *Blockchain Byte*.

Scheid, E. J., & Stiller, B. (2018). *Automatic SLA Compensation based on Smart Contracts*.

Smith, R. (2019). Proof of Burn. Retrieved January 9, 2019, from Proof of Burn | Consensus Through Coin Destruction website: https://coincentral.com/proof-of-burn/

Sudhir Khatwani. (2018). What is Double Spending & How Does Bitcoin Handle It? Retrieved November 12, 2018, from Bitcoin website: https://coinsutra.com/bitcoin-double-spending/

Szabo, N. (1997). The idea of smart contracts.

Thakkar, P., Nathan, S., & Vishwanathan, B. (2018). Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform. *CoRR*, *abs/1805.1*. Retrieved from http://arxiv.org/abs/1805.11390

The go-ethereum Authors. (2016). *Go Ethereum*.

The Linux Foundation. (2018a). *Hyperledger Fabric Documentation, Release Master*.

The Linux Foundation. (2018b). Hyperledger Fabric Ledger Diagram 2. Retrieved November 20, 2018, from Ledger, Revision e3353f51 website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/_images/ledger.diagram.2.png

The Linux Foundation. (2018c). Hyperledger Fabric Ledger Image. Retrieved December 24, 2018, from Ledger, Revision e3353f51 website: https://hyperledger-fabric.readthedocs.io/en/latest/_images/ledger.diagram.1.png

The Linux Foundation. (2019a). Chaincode Namespace. Retrieved from Considerations website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/developapps/chaincodenamespace.html#scenario

The Linux Foundation. (2019b). Endorsement Policy. Retrieved from ChaincodeInstantiateUpgradeRequest website: https://fabric-sdk-node.github.io/release-1.4/global.html#PolicySpec__anchor

The Linux Foundation. (2019c). Hyperledger Caliper. Retrieved January 21, 2019, from https://hyperledger.github.io/caliper/

The Linux Foundation. (2019d). Hyperledger Caliper Architecture. Retrieved January 21, 2019, from Architecture website: https://hyperledger.github.io/caliper/assets/img/architecture.png

The Linux Foundation. (2019e). Hyperledger Caliper Network Example. Retrieved from caliper website: https://github.com/hyperledger/caliper/tree/master/packages/caliper-samples/network

The Linux Foundation. (2019f). Hyperledger Fabric. Retrieved from Fabric website: https://www.hyperledger.org/projects/fabric

The Linux Foundation. (2019g). Hyperledger Fabric Block. Retrieved from Blocks, Revision 0163d56f website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/_images/ledger.diagram.4.png

The Linux Foundation. (2019h). Hyperledger Fabric Network. Retrieved from https://hyperledger-fabric.readthedocs.io/en/release-1.4/network/network.html

The Linux Foundation. (2019i). Hyperledger Ledger. Retrieved from Ledger website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html

The Linux Foundation. (2019j). Hyperledger Ordering Service. Retrieved from Ordering service implementations website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html

The Linux Foundation. (2019k). Introduction - Hyperledger Fabric. Retrieved from release-1.4 website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html

The Linux Foundation. (2019l). Network Completed. Retrieved from Install not instantiate website: https://hyperledger-fabric.readthedocs.io/en/release-1.4/network/network.html?highlight=limit#network-completed

Tschorsch, F., & Scheuermann, B. (2016). Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys & Tutorials*, *18*(3), 2084–2123. https://doi.org/10.1109/COMST.2016.2535718

Tschorsch, F., & Scheuermann, B. (2017). *Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies* . *18*(2 January 2018), 1–5. https://doi.org/2 January 2018

Walker, G. (2018). What's inside a Block on the Blockchain? Retrieved November 16, 2018, from

Blocks website: http://learnmeabitcoin.com/guide/blocks

Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2018). *Blockchain technology overview*. https://doi.org/10.6028/NIST.IR.8202

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, 557–564. https://doi.org/10.1109/BigDataCongress.2017.85

# Apêndices

# Apêndice A Diagrama UML do script de automação dos testes de benchmark

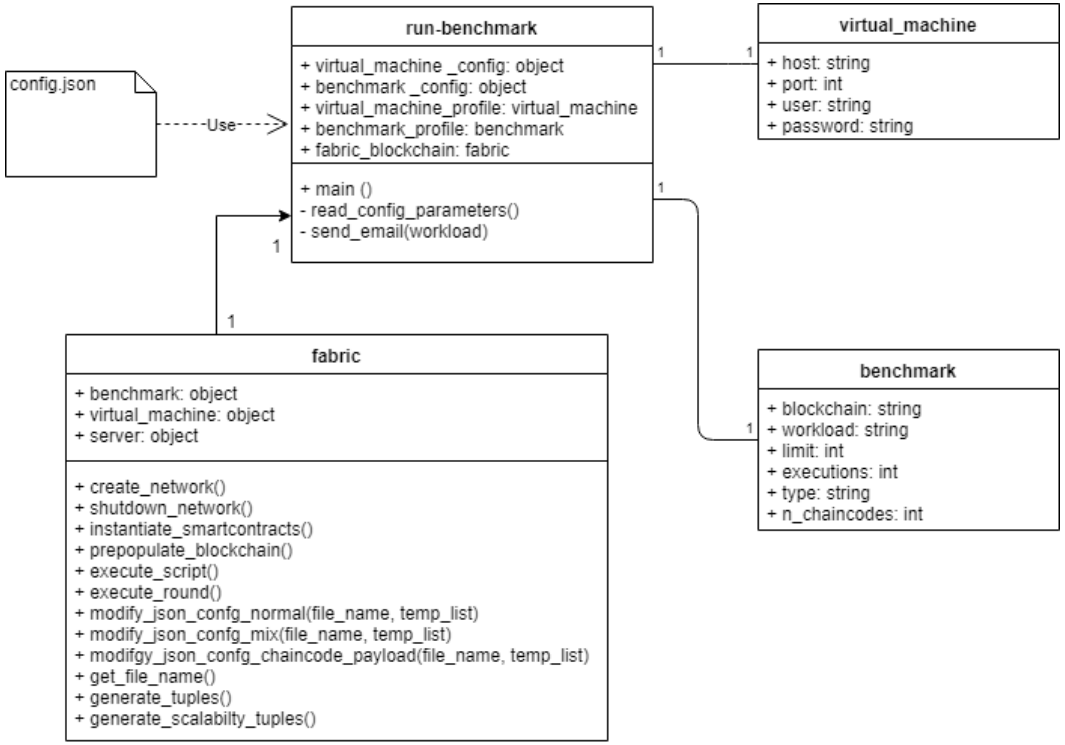


Figura 83 - Diagrama UML do script de automação da execução dos testes de benchmark

# Apêndice B
# Resultados adicionais da execução dos testes de benchmark

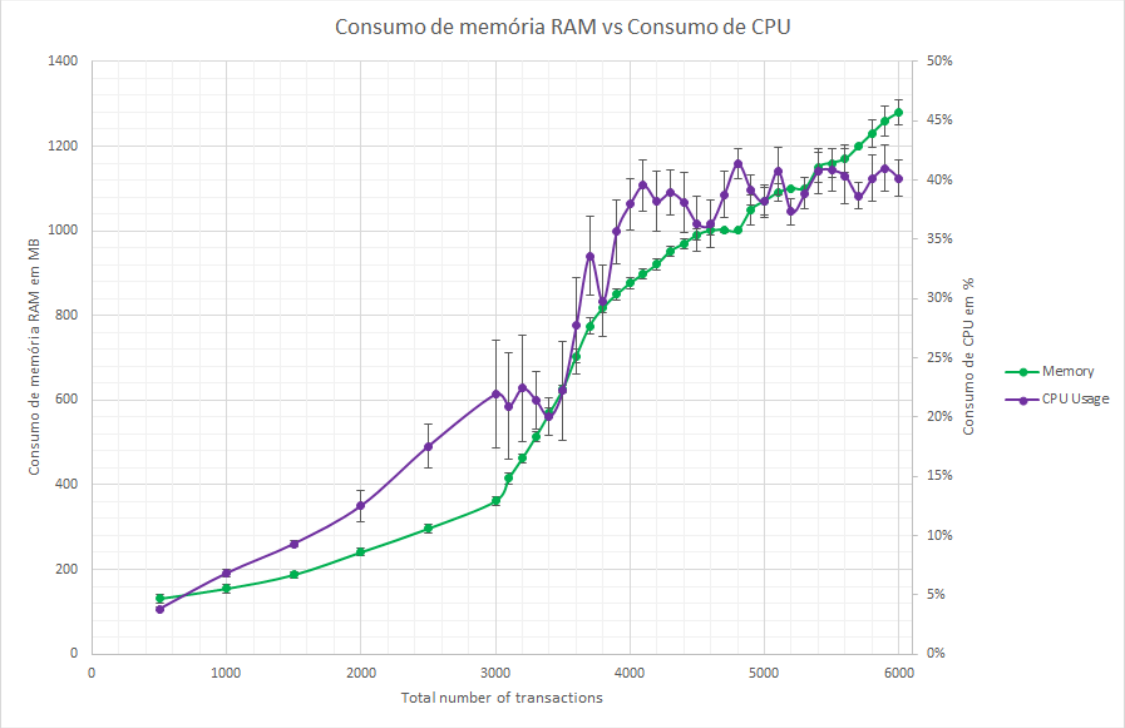


Figura 84 - Consumo de memória RAM vs Consumo de CPU - T1F – Intervalo de 0 a 6000 transações

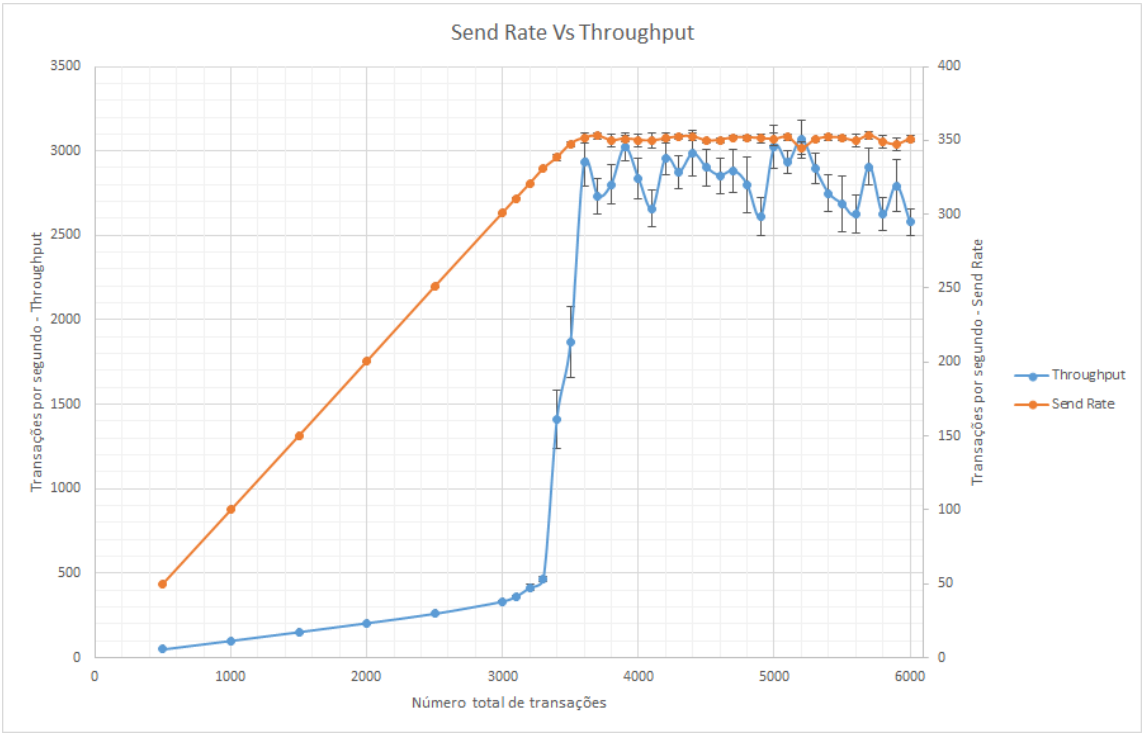


Figura 85 - Send Rate vs Throughput - T2F – Intervalo de 0 a 6000 transações

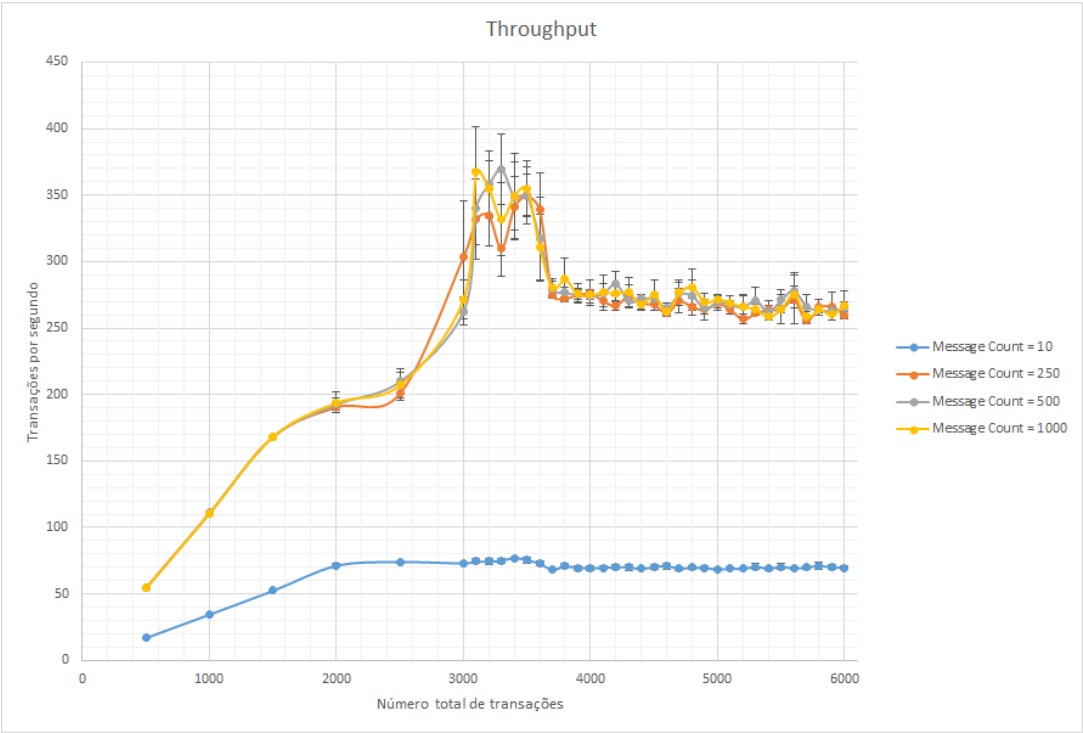


Figura 86 - Comparação dos valores de Throughput - T4F – Intervalo de 0 a 6000 transações

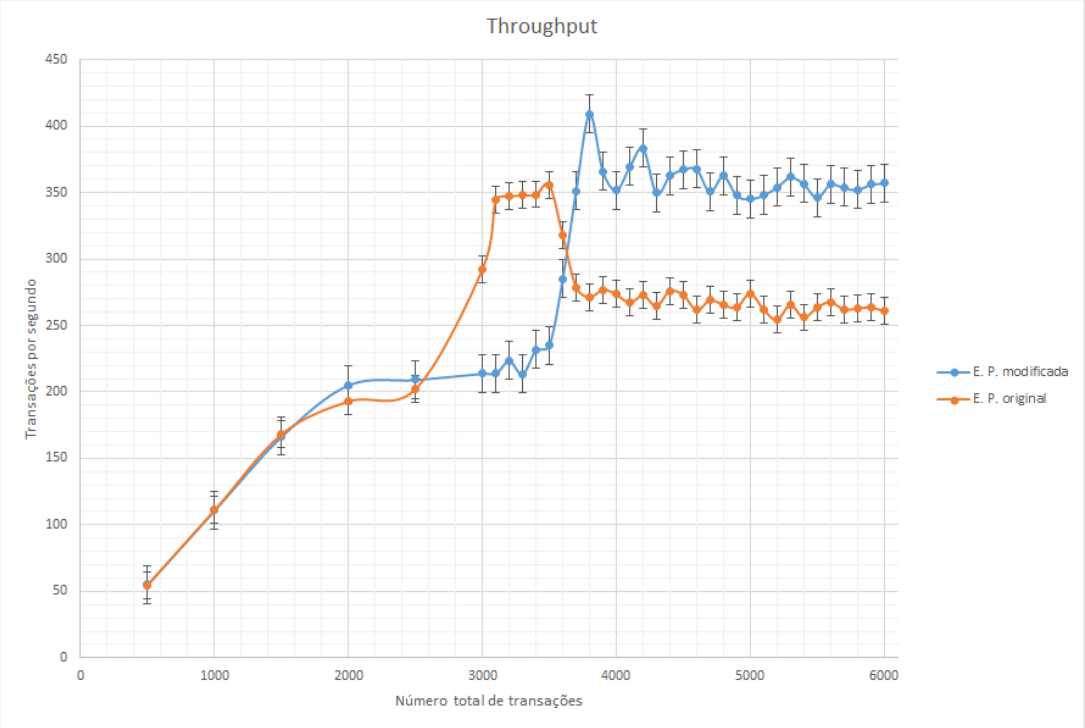


Figura 87 - Comparação dos valores de Throughput - T5F – Intervalo de 0 a 6000 transações

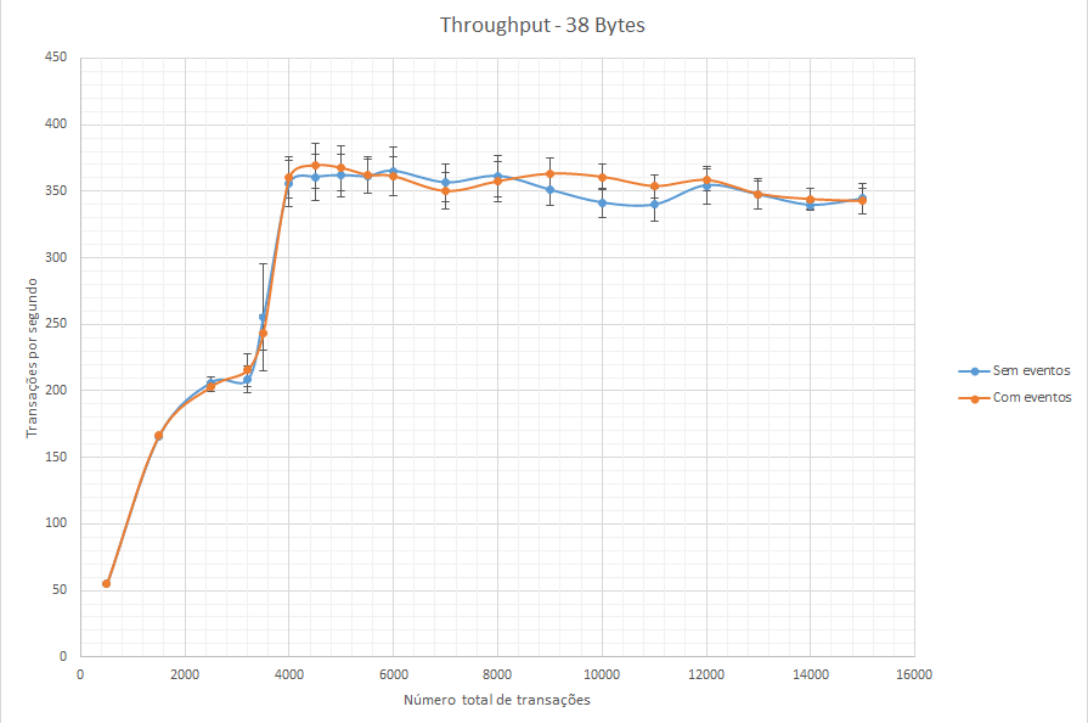


Figura 88 - Comparação dos valores de Throughput para a carga de tamanho 38 bytes – T7F

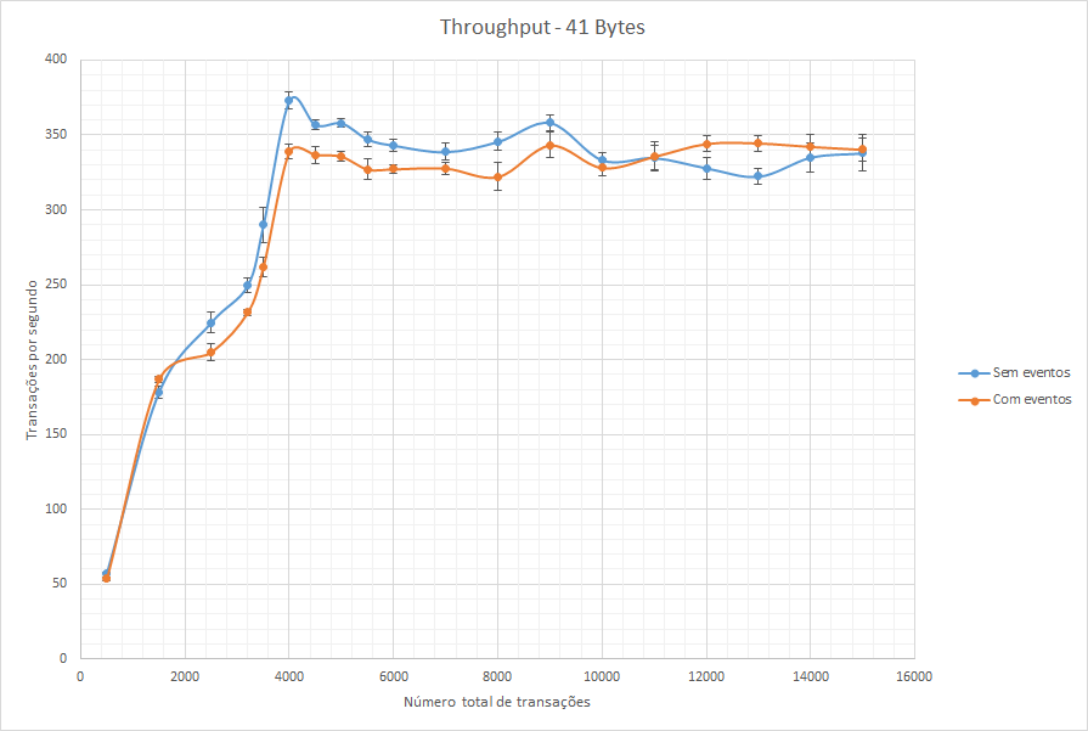


Figura 89 - Comparação dos valores de Throughput para a carga de tamanho 41 bytes – T7F

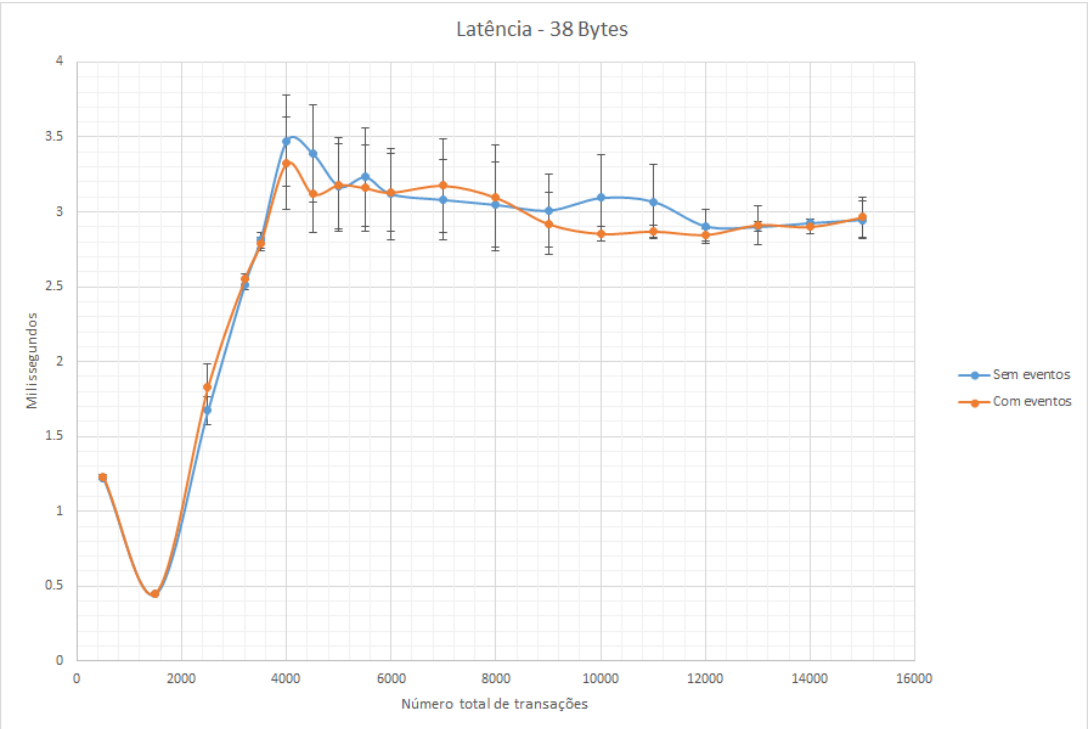


Figura 90 - Comparação dos valores de Latência para a carga de tamanho 38 bytes – T7F

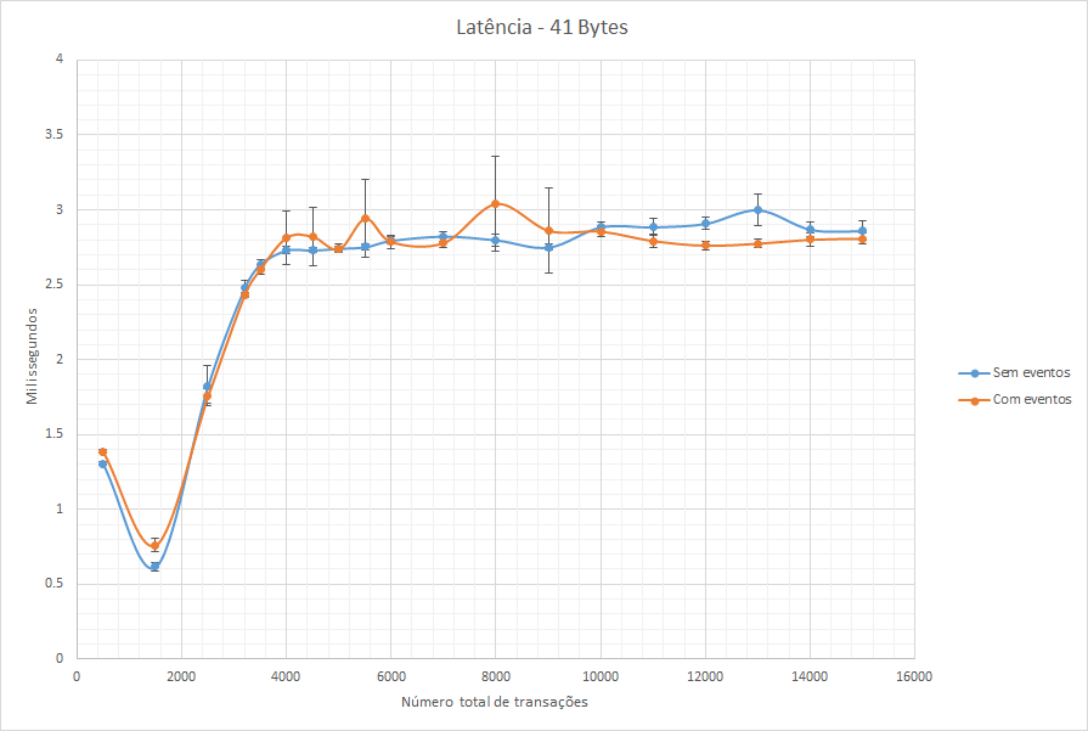


Figura 91 - Comparação dos valores de Latência para a carga de tamanho 41 bytes – T7F

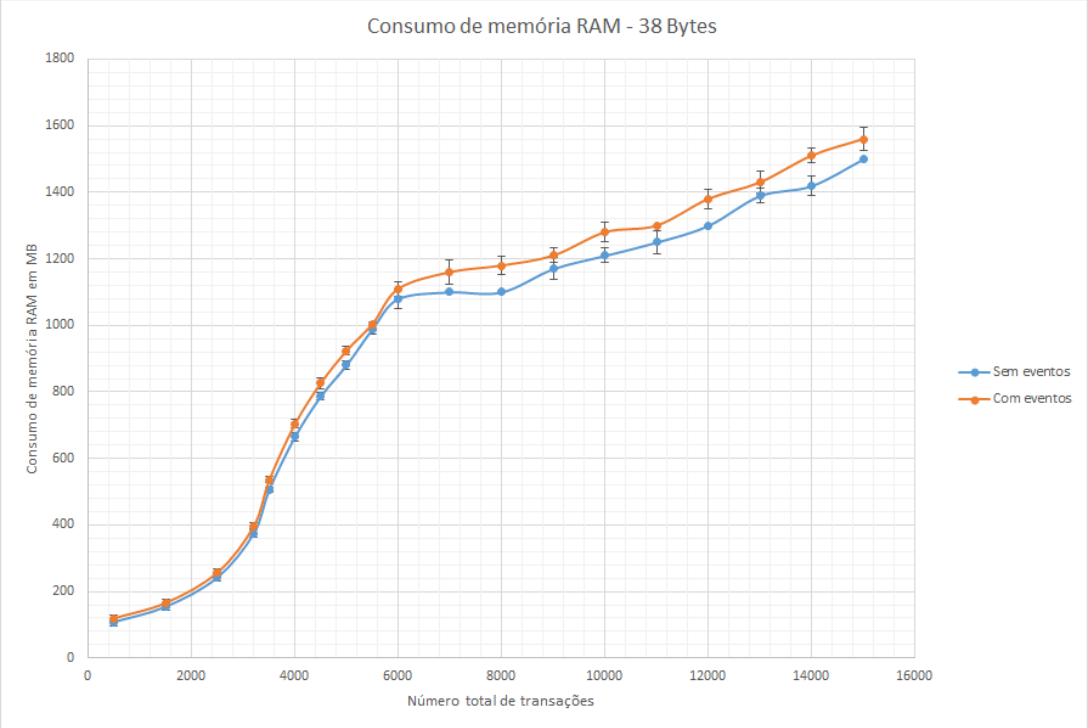


Figura 92 - Comparação dos valores de consumo de memória RAM para a carga de tamanho 38 bytes – T7F

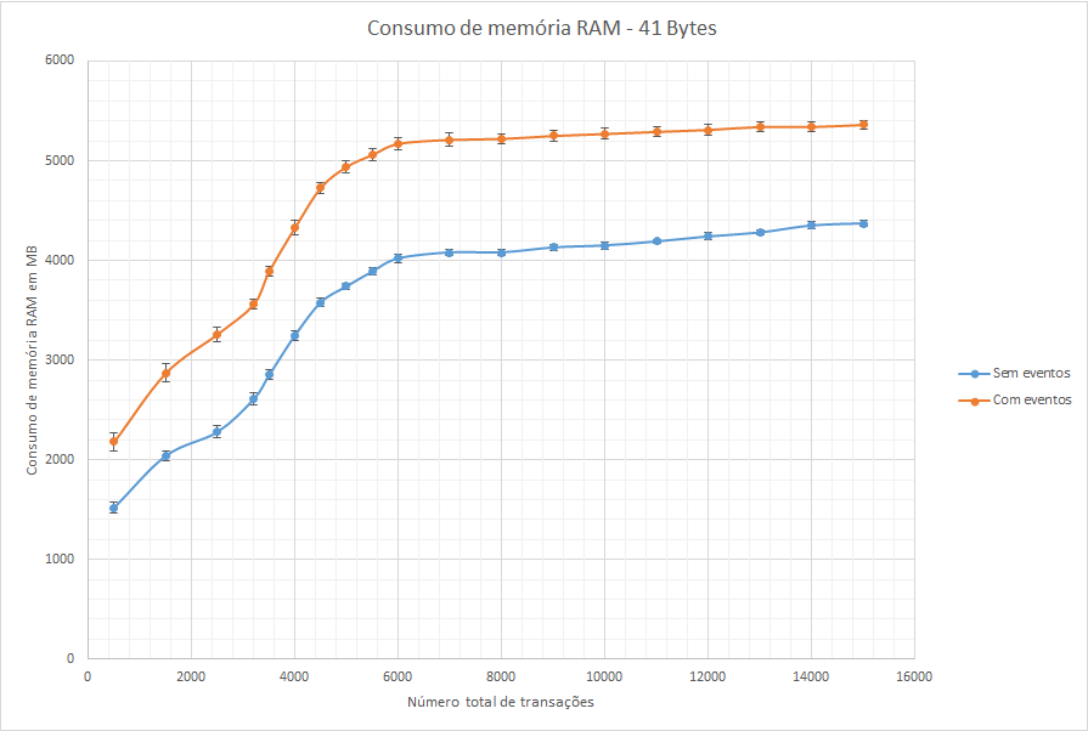


Figura 93 - Comparação dos valores de consumo de memória RAM para a carga de tamanho 41 bytes – T7F

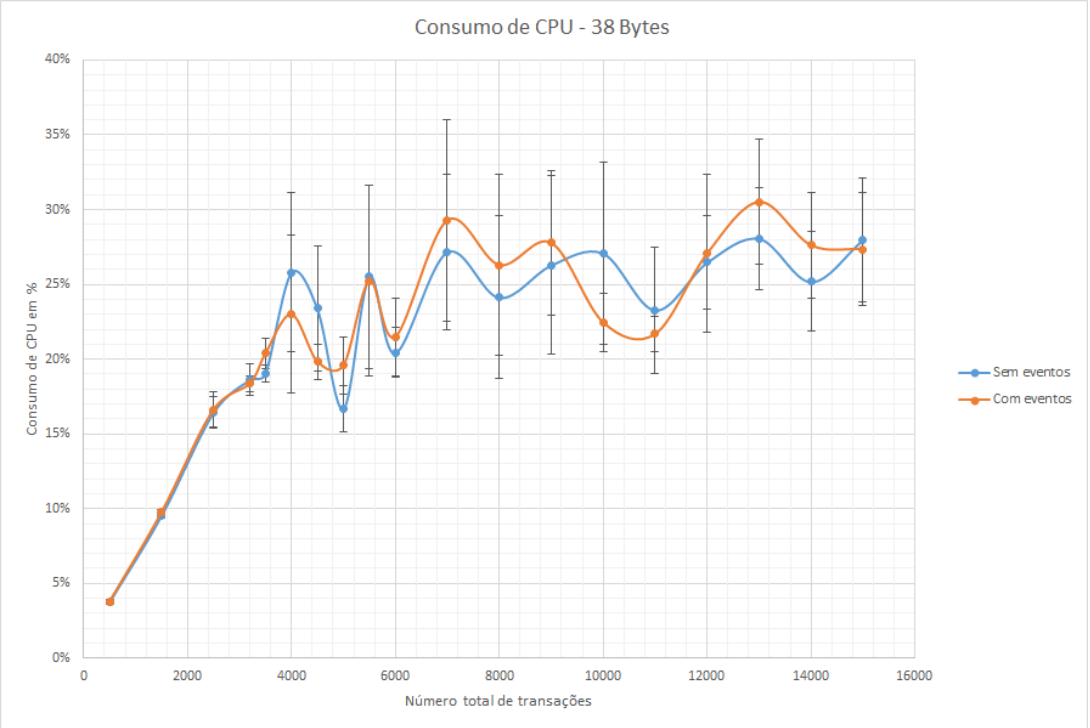


Figura 94 - Comparação dos valores de consumo de CPU para a carga de tamanho 38 bytes – T7F

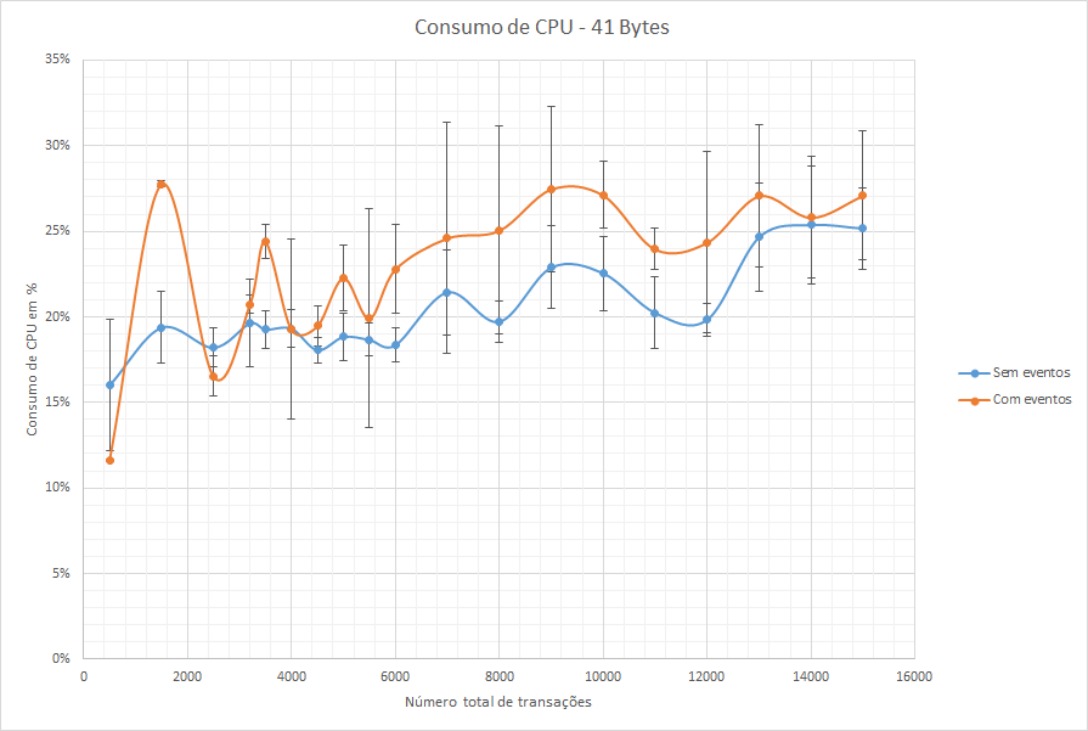


Figura 95 - Comparação dos valores de consumo de CPU para a carga de tamanho 41 bytes – T7F