Universidade Federal Fluminense
Escola de Engenharia
Curso de Graduação em Engenharia de Telecomunicações

Leonardo Mesentieri Burle

Um estudo de caso em cadeias de blocos: principais mecanismos de consenso e a plataforma Hyperledger Fabric

Niterói – RJ

Dezembro de 2019

Leonardo Mesentieri Burle

Um estudo de caso em cadeias de blocos: principais mecanismos de consenso e a plataforma Hyperledger Fabric

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Engenheiro de Telecomunicações.

Orientador: Prof. Dr. Diogo Menezes Ferrazani Mattos

Niterói – RJ

2019

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

B961e Burle, Leonardo Mesentieri
Um estudo de caso em cadeia de blocos : principais mecanismos de consenso e a plataforma Hyperledger Fabric / Leonardo Mesentieri Burle ; Diogo Menezes Ferrazani Mattos, orientador. Niterói, 2019.
70 f.

Trabalho de Conclusão de Curso (Graduação em Engenharia de Telecomunicações)-Universidade Federal Fluminense, Escola de Engenharia, Niterói, 2019.

1. Cadeia de Blocos. 2. Mecanismos de Consenso. 3. Hyperledger Fabric. 4. Blockchain. 5. Produção intelectual. I. Mattos, Diogo Menezes Ferrazani, orientador. II. Universidade Federal Fluminense. Escola de Engenharia. III. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

Leonardo Mesentieri Burle

Um estudo de caso em cadeias de blocos:
principais mecanismos de consenso e a plataforma Hyperledger Fabric

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Engenheiro de Telecomunicações.

Aprovada em 10 de Dezembro de 2019.

BANCA EXAMINADORA

Prof. Dr. Diogo Menezes Ferrazani Mattos - Orientador
Universidade Federal Fluminese - UFF

Profª. Drª Dianne Scherly Varela de Medeiros
Univesidade Federal Fluminense - UFF

Prof. Dr. Célio Vinicius Neves de Albuquerque
Universidade Federal Fluminese - UFF

Niterói – RJ
2019

Resumo

Cadeia de blocos (blockchain) é uma tecnologia disruptiva cujo objetivo é criar um registro distribuído em uma rede par-a-par. A fim de garantir o caráter descentralizado do registro, se fazem fundamentais mecanismos de consenso, ou de acordo, que atuarão na obtenção da consistência do sistema. Esses mecanismos são, portanto, parte crítica no estudo de cadeia de blocos e por isso têm sido um importante alvo de pesquisa na área. Este trabalho tem como objetivo prover uma visão geral da tecnologia de cadeia de blocos, com enfoques em mecanismos de consenso e na plataforma de desenvolvimento Hyperledger Fabric. São abordados os alicerces da tecnologia, como o Problema dos Generais Bizantinos [1] e as propriedades de imutabilidade, descentralização e integridade, assim como a estrutura de blocos, como se dá o encadeamento entre eles, e suas aplicações em diversos setores. Alguns dos principais mecanismos de consenso probabilísticos e determinísticos existentes, como o prova de trabalho [2] e o PBFT [3], respectivamente, são avaliados à luz dos conceitos de segurança e vivacidade e das limitações impostas pelo Teorema CAP [4], expondo os pontos positivos e negativos de cada algoritmo. Além disso, é descrito o modo de funcionamento do Hyperledger Fabric junto de seus mecanismos de consenso plugáveis e sua arquitetura modular.

Palavras-chave: Cadeias de bloco, mecanismos de consenso, Hyperledger Fabric.

Abstract

Blockchain is a disruptive technology whose goal is to create a distributed record on a peer-to-peer network. In order to ensure the decentralized nature of the registry, mechanisms that provide consensus, or agreement, are fundamental for achieving consistency in the system. These mechanisms are, therefore, a critical part in the study of blockchain and have, for this reason, been an important target of research in the area. The purpose of this work is to provide an overview of the blochain technology, focusing on its consensus mechanisms and the blockchian development platform Hyperledger Fabric. The foundations of this technology are discussed, approaching topics such as Byzantine Generals Problem [1], the properties of immutability, decentralization and integrity, as well as the structure of blocks and how they are linked to each other, besides applications in several sectors. Some of the main existing probabilistic and deterministic consensus mechanisms, like Proof of Work [2] and PBFT [3], respectively, will be evaluated in light of the safety and liveness concepts and the limitations imposed by the CAP Theorem [4], exposing the positive and negative points of each algorithm. In addition, it will be described how Hyperledger Fabric works alongside with the pluggable consensus mechanisms and its modular architecture.

Keywords:Blockchain, Conesensus Mechanisms, Hyperledger Fabric

“A imaginação muitas vezes nos conduz a mundos a que nunca fomos, mas sem ela não iremos a nenhum lugar".

-Carl Sagan

Agradecimentos

Agradeço imensamente a todos professores que fizeram parte da minha vida acadêmica, incluindo os que tive prazer de conhecer na Escola Pequeno Príncipe, assim como os que encontrei no Colégio Ruy Barbosa, no Liceu Franco Brasileiro além de, mais recentemente, na Universidade Federal Fluminense. Não seria a pessoa que sou se não fosse a contribuição de cada desses importantes profissionais. Sou grato também a meus familiares, em especial meu pai Augusto, minha mãe Marina e minha avó Maria Lúcia, que sempre investiram incondicionalmente nos meus estudos.

Lista de Figuras

Conteúdo

Capítulo 1

Introdução

A tecnologia de cadeia de blocos (blockchain) promete revolucionar diversos setores como finanças, saúde, cadeias de suprimentos e, também, estruturar áreas ainda emergentes, como a Internet das Coisas. Contudo, o caráter disruptivo dessa tecnologia já pode ser notado. As principais plataformas de criptomoedas, cujo funcionamento é baseado em cadeias de blocos, tiveram em 2018 um valor de mercado global avaliado em mais de 850 bilhões de dólares[1] e com fortes tendências de crescimento.

Apesar da difusão relativamente recente, iniciada a partir de 2008, através do icônico artigo de Satoshi Nakamoto [2], a tecnologia de cadeia de blocos é uma colcha de retalhos [5] feita de estudos realizados no século XX, principalmente nas áreas de criptografia e computação distribuída. Através de aplicações práticas desses conceitos se tornou possível realizar o papel até então desempenhado por entidades centralizadoras, como bancos. Essas entidades funcionam como âncoras de confiança, isto é, uma entidade na qual todos os membros da rede confiam para manter a consistência e o correto funcionamento do sistema, mas que pode impor dispêndios, custos, aos participantes pelo trabalho realizado, conforme ocorre no setor financeiro. Já a tecnologia de cadeia de bloco propõe uma quebra de paradigma, transferindo a âncora de confiança de uma autoridade central para um algoritmo distribuído, assim eliminando custos inerentes à existência dessas entidades.

Uma cadeia de bloco pode ser definida como uma máquina de estado replicada [6] que consegue, através de diversos algoritmos, sustentar um estado consistente em todo o sistema distribuído. Em termos práticos, cada um dos membros dessa rede, denominado

[1] https://www.fortunebusinessinsights.com/industry-reports/cryptocurrency-market-10014

nó, tem uma cópia do livro-razão, que deve sempre estar sincronizada com as demais. Em outras palavras, todas devem ter o mesmo estado, que pode ser consultado ou auditado sempre que necessário. Os blocos novos são registrados contendo as transações mais recentes submetidas por clientes e também com um hash, resumo criptográfico, da bloco anterior. Portanto, o livro-razão fica estruturado de modo similar a listas encadeadas, cada livro contém uma referência ao anterior, formando uma cadeia de blocos, ou *ledger*.

Devido a propriedades das funções de resumos criptográficos, qualquer alteração feita em um registro gerará uma mudança no valor do *hash*. Isso aliado ao encadeamento faz com que alterações em certo bloco da cadeia sejam facilmente auditáveis, bastando recalcular o *hash* e comparará-lo ao do bloco seguinte. Por esse motivo, alterar registros em uma cadeia de blocos de maneira consistente se torna inviável em termos práticos, já que para isso deve-se ajustar o *hash* de todos os blocos seguintes ao alterado. Dessa forma é garantida uma das propriedades fundamentais de cadeias de blocos, a imutabilidade. Por isso, os membros de uma cadeia devem concordar com cada registro novo, visto que eles não poderão ser editados após a inserção na livro-razão, o que muitas vezes não é uma tarefa fácil.

Mecanismos de consenso são sistemas compostos por algoritmos capazes de prover confiança a um serviço em que seus participantes não necessariamente confiam uns nos outros. A palavra consenso tem origem etimológica do Latim *consensus* e significa acordo, conformidade, unanimidade, a obtenção dessas propriedades é importante para sustentar o caráter descentralizado de uma cadeia de blocos. Em termos práticos, os algoritmos de consenso têm duas principais atribuições: garantir que todas as transações registrada na cadeia de blocos estejam corretas e também definir a ordem em que essas transações vão estar dentro de um bloco. Entretanto, as formas de desempenhar essas funções são diversas e têm sido o principal alvo de estudo de pesquisadores na área de cadeia de blocos. Devido ao grande potencial dessa tecnologia, há um esforço conjunto de centros de pesquisas de universidades, consórcios *open source* e empresas no desenvolvimento de algoritmos de consenso mais eficientes. Entretanto, percebe-se que a eficácia dos mecanismos de consenso está diretamente relacionada com o tipo de ambiente no qual a cadeia de blocos se propõe a atuar.

Um modo de classificar plataformas de cadeias de blocos é quanto à permissão de participação na cadeia e no mecanismo de consenso. Cadeias privadas são normalmente

gerenciadas por uma única instituição, ou grupo de instituições, que vai determinar quais nós tem permissão para participar da rede [7]. Já as cadeias não permissionadas (ou públicas), por outro lado, são de participação livre. Nesse tipo de sistema alcançar o consenso tende a ser mais custoso, fazendo com que o desempenho dessas plataformas seja pior do que as permissionadas em métricas como o tempo de inserção de bloco.

O mecanismo de consenso por Prova de Trabalho (*Proof of Work* ou PoW), utilizado na criptomoeda Bitcoin, é o mais clássico exemplo de algoritmo usado no contexto de cadeias de blocos não permissionadas. Seu funcionamento é baseado em uma competição computacional entre os chamados mineradores, membros da cadeia que fazem parte do mecanismo de consenso, para resolver um desafio criptográfico. O minerador vencedor desse desafio é o responsável por organizar as transações no bloco e disseminá-lo para o restante da rede, e recebe um prêmio em Bitcoins por isso. Apesar de servir bem a seu propósito no contexto em que se propõe atuar, o PoW possui problemas de eficiência, principalmente em relação ao gasto de energia para se minerar um bloco.

Por outro lado, há mecanismos de consenso idealizados para funcionar em cadeias permissionadas, como o caso do Raft, que é implementado seguindo o modelo líder-seguidor. Esse mecanismo, apesar de ser significativamente mais eficiente do que o por Prova de Trabalho, não é apropriado para cenários em que o grau de confiança entre os participantes seja baixo. Todavia, uma análise mais cautelosa e detalhada desses mecanismos requer outros parâmetros além da confiabilidade entre os membros da cadeia.

Os conceitos de segurança e vivacidade (*safety and liveness*) descritos por Leslie Lamport [8] e o Teorema CAP de Eric Brewer [4] são ferramentas teóricas importantes na avaliação de qualquer sistemas de cadeia de blocos, sobretudo dos mecanismos de consenso. O primeiro explicita que qualquer parte de um sistema pode ser entendido como a intersecção de *safety* e *liveness*. Esses dois conceitos podem ser descritos de modo simplificado, respectivamente, como propriedade cuja intenção é fazer com que algo de ruim não aconteça, e propriedade cuja intenção é fazer com que algo de bom aconteça. O Teorema de Brewer, por sua vez, estabelece limitações para sistemas distribuídos, postulando que é impossível garantir ao mesmo tempo consistência, disponibilidade e tolerância a partições, sendo apenas possível assegurar duas dessas três propriedades. Consistência, de modo simplificado, está relacionada com a aptidão de um sistema para apresentar, quando requisitado, o dado salvo mais recente ou mensagem de erro; disponibilidade pode ser defi-

nida como a capacidade de um sistema apresentar uma mensagem válida, diferente de erro quando recebida uma requisição; Tolerância a partições diz respeito ao sistema continuar a operar independentemente de falhas na rede.

Vale notar que aplicações para cadeia de blocos necessitam de balanceamentos distintos das propriedades definidas no Teorema CAP, o que por outro lado, limita a atuação ótima da cadeia em outros cenários. Assim, conclui-se que uma solução única para o problema de consenso em sistemas distribuídos é inviável e que mecanismos de consenso eficientes tendem a ser desenvolvidos para atuarem em cenários específicos.

A plataforma de desenvolvimento de cadeia de blocos Hyperledger Fabric, que é mantida pela Linux Foundation, parte desse princípio de que não existe um algoritmo *"one size fits all"*, isto é, uma solução otimizada para todos os cenários em que uma cadeia de blocos pode ser implementada. Por isso, o Fabric implementa mecanismos de consenso adaptáveis, através dos quais o desenvolvedor consegue compor a cadeia de blocos mais apropriada para sua necessidade. Essa modularidade aliada à flexibilidade e escalabilidade é o que diferencia o Hyperledger Fabric das demais plataformas.

Esse trabalho visa prover uma visão geral sobre cadeias de blocos com enfoque em mecanismos de consenso, sendo apresentados no Capítulo 2 conceitos que fundamentam o estudo dessa tecnologia, como o Problema dos Generais Bizantinos, o Teorema CAP e a Infraestrutura de Chave Pública . Outros atributos que constituem o alicerce para o desenvolvimento de cadeias de blocos, como a integridade e a anonimidade serão estudados no Capítulo 3. Tópicos abordando a estrutura de um bloco, como é feito o encadeamento entre esses, além de algumas das principais aplicações de cadeia de blocos também serão abordados nesse capítulo. O Capítulo 4 foca o estudo nos principais mecanismos de consenso existentes, como o PoW, PoS, Ripple, Paxos, levando em conta as plataformas nas quais eles estão inseridos e avaliando seus pontos fortes e fracos. No Capítulo 5, por sua vez, é apresentada a arquitetura da plataforma Hyperledger Fabric, incluindo seus mecanismos de consenso Kafka com Zookeeper, Solo e Raft, junto de um exemplo prático de funcionamento de cadeia de blocos. Por fim, o Capítulo 6 apresenta as conclusões obtidas e também sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

As cadeias de blocos são primordialmente redes par a par. Houve diversas tentativas que procuraram definir esse tipo de arquitetura de rede, podendo ser descrita como um conjunto de recursos distribuídos que estão conectados através de uma rede [9].

Uma arquitetura distribuída de rede pode ser chamada de rede par a par (P-to-P, P2P, etc.) caso os participantes compartilhem seus próprios recursos de hardware tais como, poder de processamento, capacidade de armazenamento, capacidade de link de rede, impressoras. Esses recursos compartilhados são necessários para prover o serviço e o conteúdo oferecido pela rede, como compartilhamento de *workspaces* para colaboração. Eles são acessíveis por outros nós diretamente, sem passar por entidades intermediárias. Os participantes de tal rede são tanto provedores de recurso de serviço ou conteúdo assim como solicitantes desses recursos.

Outras definições classificam as redes par a par de uma maneira mais abrangente, como sendo a arquitetura oposta a denominada cliente-servidor. Essa última é definida como uma rede distribuída heterogênea, possuindo um sistema de desempenho superior, servidor, e diversos outros sistemas cujos desempenhos são menores, clientes. Esses clientes estão ligados diretamente ao servidor, que seria o único provedor de conteúdo e de serviços da rede. Sendo assim, os clientes conseguem apenas requisitar a utilização de recursos e serviços.

A literatura de sistemas distribuídos e redes par a par é vasta e anterior ao surgimento das cadeias de blocos. Assim, a teoria por trás das cadeias de blocos usa diversos conceitos provenientes dessa literatura. Neste capítulo serão apresentados alguns conceitos e também outras definições fundamentais para o estudo de cadeias de blocos.

2.1 Teorema CAP

O Teorema CAP (*Consistency Availabitity Partition tolerance*), também conhecido como teorema de Brewer [4] identifica três características como sendo críticas para qualquer sistema distribuído: consistência, disponibilidade e tolerância a particionamentos.

- Um sistema é dito consistente quando garante que toda leitura seja a mesma em qualquer nó. No caso de não haver essa garantia, o resultado deverá ser uma mensagem de erro.
- Um sistema possui disponibilidade caso sempre dê uma resposta válida, diferente de uma mensagem de erro, a uma requisição feita por um nó que não esteja em falha.
- Já um sistema é considerado tolerante a partições caso continue a operar mesmo quando haja um número de mensagens perdidas ou que tenham sofrido atrasos devido à rede.

Este teorema está relacionado com a confiabilidade de um sistema distribuído e como esse se comporta em casos de falhas. Brewer postulou que é impossível garantir essas três propriedades para um sistema, podendo-se obter no máximo duas delas. Portanto, caso um sistema seja construído fornecendo duas dessas três propriedades, a terceira necessariamente não será atendida.

Em caso de o sistema sempre garantir que toda leitura feita estará correta e nunca apresentará uma mensagem de erro, o mesmo não será tolerante a partições. Isso ocorre pois em casos de falha na rede é impossível garantir que não houve nenhuma atualização nos dados, o que viola uma das duas premissas.

Caso o sistema seja tolerante a partições e consistente, é impossível ele garantir o retorno somente de mensagens válidas. Em caso de falha na comunicação entre um dos componentes com o restante do sistema e este continue funcionando, sempre apresentando a leitura mais recente, não há outra opção senão apresentar uma mensagem de erro, não possuindo assim a disponibilidade.

Se um sistema for tolerante a partições e disponível, ele não conseguirá garantir que todas as leituras estejam corretas. A premissa é que o sistema sempre apresentará uma resposta válida em caso de requisição, ainda que esta não seja a versão mais recente da contida no sistema, o que, em caso de falhas na rede, viola o princípio da consistência.

De modo geral, incluindo no caso das cadeias de blocos, os sistemas distribuídos prezam pela tolerância a partições, pois falhas e atrasos na rede ocorrem com certa frequência e os sistemas devem conseguir lidar com elas. Assim, a consistência e a disponibilidade devem ser balanceadas conforme as necessidades das aplicações.

2.2 Problema dos Generais Bizantinos

Além de conseguirem ajustar suas características de tolerância a partições, consistência e disponibilidade de forma satisfatória, sistemas confiáveis devem ter formas para lidar com componentes maliciosos ou que estão apresentando um mau funcionamento, como no caso de apresentarem informações conflitantes para partes distintas do sistema [1].

Lamport apresenta o problema da computação entre nós não confiáveis através de uma analogia, em que várias divisões do exército bizantino, cada qual comandada por seu general, fazem um cerco em uma cidade inimiga e planejam um ataque. Devido à distância, os generais conseguem apenas fazer contato uns com os outros através de um mensageiro e, dessa forma, elaboram um plano de invasão. Contudo, alguns generais podem ser traidores e tentar boicotar o acordo, logo, para que a missão tenha sucesso, deve haver um algoritmo garantindo que:

1. Todos os generais leais entrem em acordo sobre a mesma estratégia de invasão. Os generais comprometidos com a invasão da cidade devem entrar em consenso sobre qual e melhor plano a ser seguido e, após a decisão, devem segui-lo à risca, independentemente dos traidores. Assim, o algoritmo acordado entre os generais deve ter sucesso, não importando o que os traidores façam.

2. Um pequeno número de traidores não influenciem generais leais ao império bizantino a adotarem um plano ruim. Essa condição é mais difícil de ser descrita, já que o termo plano ruim pode não ser fácil de identificar dependendo do contexto. Caso as alternativas do problema citado sejam apenas atacar ou recuar, o acordo explicado em 1, pode ser feito através de voto, baseado em maioria simples. Contudo, se houver um número quase igual de generais traidores e generais leais é impossível definir qual das opções pode ser chamada de ruim. As Figuras 2.1 e 2.2 abaixo ilustram esse problema de computação distribuída com três generais.

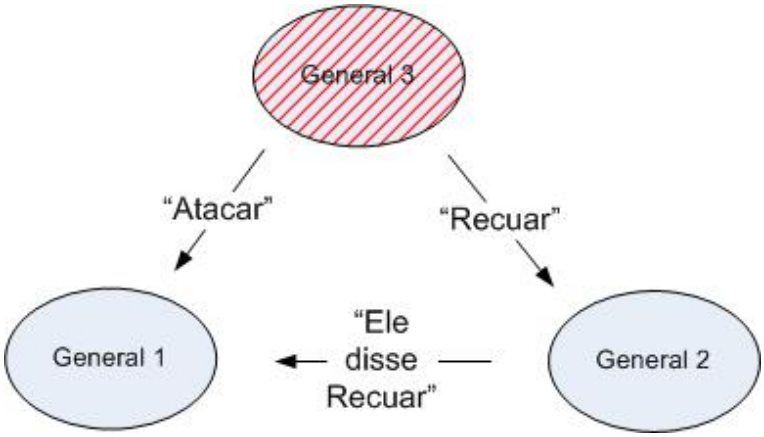

Figura 2.1: General 3 é traidor

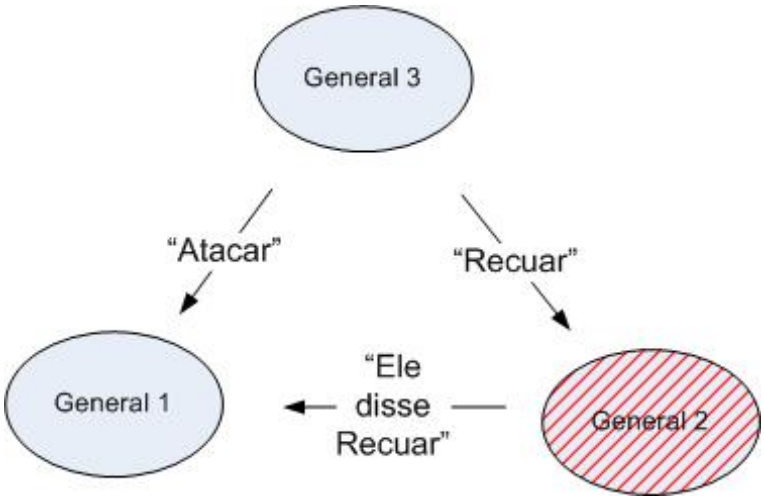

Figura 2.2: General 2 é traidor

Em nenhum dos casos apresentados na Figura 2.1 e na Figura 2.2, o General 1 consegue definir qual das opções seguir, já que não sabe qual dos seus companheiros é o traidor. Assim, percebe-se que nesse caso o Problema dos Generais Bizantinos não possui solução, sendo portanto impossível atingir o consenso desejado.

Pode-se mostrar que não é possível existir um sistema tolerante a esse tipo de falhas (denominadas falhas bizantinas), com um número total de generais menor do que 3f + 1, onde f indica o número de generais traidores [10].

O problema apresentado consiste em uma analogia onde os generais representam os computadores em uma rede e a invasão à cidade representa uma tarefa que deve ser realizada em conjunto por esses computadores. As plataformas de cadeias de blocos devem saber lidar com tipo de problema e é papel de seus respectivos mecanismos de consenso

encontrar a solução mais apropriada para cada ambiente em que a plataforma se propõe a atuar.

2.3 Segurança e Vivacidade

Um programa pode ser entendido como uma máquina de estados infinita, e a execução desse como uma sequência também infinita de estados, denominada histórico. Caso a execução do programa termine, seu estado permanecerá fixo infinitamente e será definido pelo último valor apresentado na execução do programa.

Leslie Lamport define informalmente o conceito de segurança (*safety*) em seu artigo [8], como sendo uma propriedade de um programa impedir que algo ruim aconteça. Para Lamport, vivacidade (*liveness*) é, nesse mesmo contexto, propriedades que trabalham para que algo de bom aconteça no programa. O termo algo ruim corresponde na prática a defeitos que devem ser evitados no software, como *deadlock*s, por exemplo. Já algo bom refere-se a propriedades de um software que contribuem para o funcionamento correto do mesmo.

Esses conceitos ajudam a classificar determinadas propriedades de um mecanismo, facilitando assim seu entendimento, pois todo programa pode ser caracterizado por uma combinação dessas características. A distinção entre vivacidade e segurança se torna relevante na avaliação do modo de funcionamento de cadeias de blocos, sobretudo no estudo dos mecanismos de consenso. A existência de artifícios que inviabilizem a possibilidade de duas versões do mesmo bloco na cadeia está relacionada com a propriedade de segurança. Um exemplo de como o conceito de vivacidade pode ser aplicado em um contexto de cadeia de blocos são os mecanismos probabilísticos, que garantem a terminação do protocolo e assumem que eventualmente o consenso é atingido.

2.4 Assinaturas Digitais

As assinaturas digitais têm como objetivo garantir a autenticidade e a não repúdio de informações trafegadas em uma rede par a par. Assim como assinaturas manuais, as assinaturas digitais devem funcionar como uma prova irrefutável de que determinada entidade assinou o documento, sendo impossível fraudar a assinatura. Essas são implementadas no contexto digital com o auxílio de algoritmos de criptografia e garantem o não repúdio da

informação trocada, isto é, impedindo que o remetente da mensagem possar negar o seu envio.

A tecnologia usada em assinaturas digitais é a criptografia de chave assimétrica. O seu funcionamento é baseado no uso de chaves públicas e chaves privadas: cada usuário da rede deve possuir o par de chaves mencionado, de forma que a chaves públicas, conforme o próprio nome indica, devem estar disponíveis para outros usuários consultarem quando necessário, e as chaves privadas não devem nunca ser reveladas para ninguém. A implementação dessa tecnologia é feita através dos três métodos explicitados abaixo:

- O método de gerar chaves, que após ser acionado deve fornecer ao solicitante o par de chaves, pública e privada;
- O método de assinatura, que marca a mensagem com um selo único e inconfundível através da chave privada e realiza a criptografia;
- O método de validação, que recebe como entrada a chave pública do assinante e a mensagem assinada e retorna um valor booleano.

É importante que o método de assinatura seja unidirecional, sendo impossível através da mensagem assinada a obtenção da assinatura privada. Já o método de validação deve ser tal que qualquer alteração em um de seus valores de entrada (mensagem assinada ou chave pública) resulte em um valor falso, o que em termos práticos significa que o remetente não conseguiu desvendar a mensagem. Dessa forma, são garantidas a autenticidade das mensagens, visto que qualquer alteração nelas é facilmente perceptível, e o não repúdio, dado que as chaves privadas são únicas e só o remetente da mensagem tem acesso a elas.

Capítulo 3

Cadeias de Blocos

O surgimento do que posteriormente passou a se chamar de cadeia de blocos remonta à última década do século XIX, em 1991, quando os cientistas norte-americanos S. Haber e WS Stornetta publicaram o artigo [11] propondo um novo modo de se registrar datas de criação e alteração em documentos. No artigo já se notava a ideia de uma rede de blocos ligados e protegidos criptograficamente.

Contudo, foi apenas em 2008, com o protocolo da Bitcoin, proposto por Satoshi Nakamoto [2], que a tecnologia da cadeia de blocos começou a se tonar um grande alvo de pesquisas nos meios acadêmicos e corporativos. Desde então, tem-se observado diversas evoluções nessa tecnologia, que podem ser caracterizadas em gerações: *Blockchain 1.0*, *Blockchain 2.0* e *Blockchain 3.0* [12].

A primeira geração das cadeias de blocos é representada pelos sistemas par a par de dinheiro eletrônico, como o Bitcoin, cujo principal feito é eliminar uma terceira parte, uma âncora de confiança até então indispensável em sistemas financeiros. Já a segunda geração tem como seu principal expoente a plataforma Ethereum proposta por Vitalik Buterin em seu artigo [13], através do qual notou-se a possibilidade de registrar outros ativos, além de criptomoedas, como contratos inteligentes, códigos auto executáveis feitos para facilitar negociações em redes par a par. A terceira geração, por sua vez, é a mais recente das evoluções e empenha-se em reduzir, ou até mesmo eliminar, os problemas de escalabilidade enfrentados pelas cadeias de blocos.

Enquanto as gerações descritas anteriormente identificam os sistemas de cadeias de blocos no que diz respeito a avanços na tecnologia, outro critério classifica as cadeias de blocos com base na visibilidade de registros e permissão de participação. As cadeias

públicas são aquelas em que não há restrições de participação e na visualização dos registros nos livros-razão. O contrário ocorre com as cadeias privadas, em que apenas certos grupos de determinadas organizações têm permissão para participar daquele sistema, e essas permissões estão relacionadas em geral a chaves públicas e privadas, fornecidas por uma autoridade certificadora.

Apesar das diferenças entre todas as plataformas de cadeia de blocos desenvolvidas até então, o objetivo permanece: funcionar como histórico imutável de transações, implementando uma estrutura de dados distribuída em ambientes de rede par a par. Assim, existem propriedades que são inerentes à tecnologia de cadeia de blocos [5]. São elas:

- Descentralização e Desintermediação - Nos sistemas de transações convencionais, cada transferência de ativos tem a necessidade de ser validada por uma autoridade central, como um banco, de forma que a responsabilidade pelo funcionamento correto do sistema recai sobre uma única entidade. Isso resulta em uma limitação de desempenho, já que essa única entidade deve concentrar um enorme poder computacional para validar transações, e em custo aos participantes do sistema, pois a autoridade central impõe custos aos participantes da rede para que ela possa desempenhar seu papel. As cadeias de blocos surgem como uma alternativa para esse tipo de sistema, ao descentralizar a validação das transações e ao mudar o ponto focal de confiança responsável por assegurar a consistência dos dados. Desse modo, a âncora de confiança passa de uma autoridade central para um algoritmo de consenso e, com isso, a cadeia de bloco torna possível integrar de modo eficiente diversos sistemas.
- Integridade e Imutabilidade - A integridade diz respeito à garantia de que certa informação apresentada mediante consulta esteja correta. Assim, para que uma cadeia seja considerada íntegra, no instante da inserção de novos blocos, transações inválidas devem ser descobertas imediatamente e transações corretas após registradas podem ser refutadas. Além disso, deve ser impossível, ao menos em termos práticos, a exclusão de blocos da cadeia após estes serem inseridos. Atualizações só são possíveis através de novos registros e, consequentemente, novo consenso.
- Auditabilidade e Privacidade – As transações registradas no livro-razão devem poder ser auditadas. Contudo, deve haver privacidade com relação aos dados de cada usuário, sendo improvável que terceiros tenham acesso e controle dessas informações, o que é feito com ajuda de chaves públicas e privadas.

3.1 Encadeamento de Blocos

O encadeamento de blocos consiste basicamente em incluir em determinado bloco uma referência ao bloco anterior, usando o *hash*. Esse nada mais é do que um valor obtido através de uma função criptográfica homônima, que recebe uma quantidade de dados e os mapeia em dados de comprimento fixo, funcionando como um resumo. Além disso, a função criptográfica *hash* deve seguir os seguintes critérios:

- Ser computacionalmente pouco custosa de ser resolvida;
- Garantir que uma pequena variação nos dados da mensagem gere um resultado, *hash*, completamente distinto;
- Ser determinística, isto é, dada uma mesma entrada, a função deve sempre produzir o mesmo resultado;
- Assegurar que a partir do *hash* a obtenção dos dados de entrada seja em termos práticos inviável.

Assim, no instante em que o valor do *hash* de um bloco N-1 é inserido no cabeçalho do bloco seguinte (bloco N) consegue-se estabelecer uma relação, uma ligação entre ambos os blocos. Esta relação é denominada encadeamento e a Figura 3.1 ilustra esse processo.

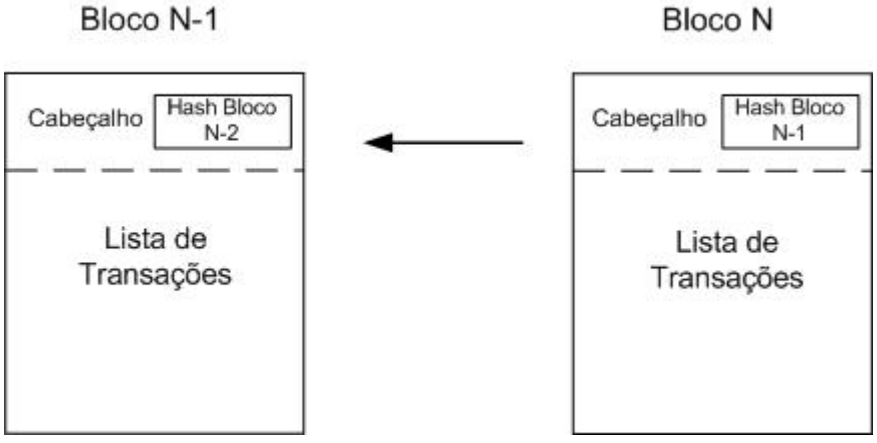

Figura 3.1: Encadeamento de Blocos - A lista de transações é secção onde estão salvas as transações no bloco. O bloco pai de um bloco é aquele cujo hash está salvo nesse bloco.

O encadeamento dificulta a atuação de nós maliciosos pois qualquer mudança feita no conteúdo de determinado bloco implicará uma alteração nos valores hash de todos os

blocos seguintes ao alterado, o que é facilmente perceptível já que esses valores de hash seguem determinados padrões, os quais serão explicitados mais adiante.

Dois dos algoritmos de *hashing* mais conhecidos são o MD5 (*Message-Digest algorithm 5*), que resulta em um valor de 128 bits, SHA-256 (*Secure Hash Algorithm*), que resulta em um valor de 256 bits, ambos comumente utilizados em um contexto de redes par a par, sendo este último o usado por Nakamoto no desenvolvimento do Bitcoin.

3.2 Estrutura dos Blocos

Nessa seção é feito um estudo de caso, apresentando estrutura de bloco do Bitcoin. Vale ressaltar que a estrutura não é comum a todas as aplicações de cadeia de blocos, mas serve de exemplo base para o estudo das demais estruturas usadas em outras plataformas, que em geral possuem diversos dos campos apresentados.

A estrutura básica de um bloco está exemplificada na Figura 3.2. O bloco está dividido em duas partes, o cabeçalho e o corpo. De modo simplificado, o corpo contém as transações enquanto o cabeçalho contém as demais informações.

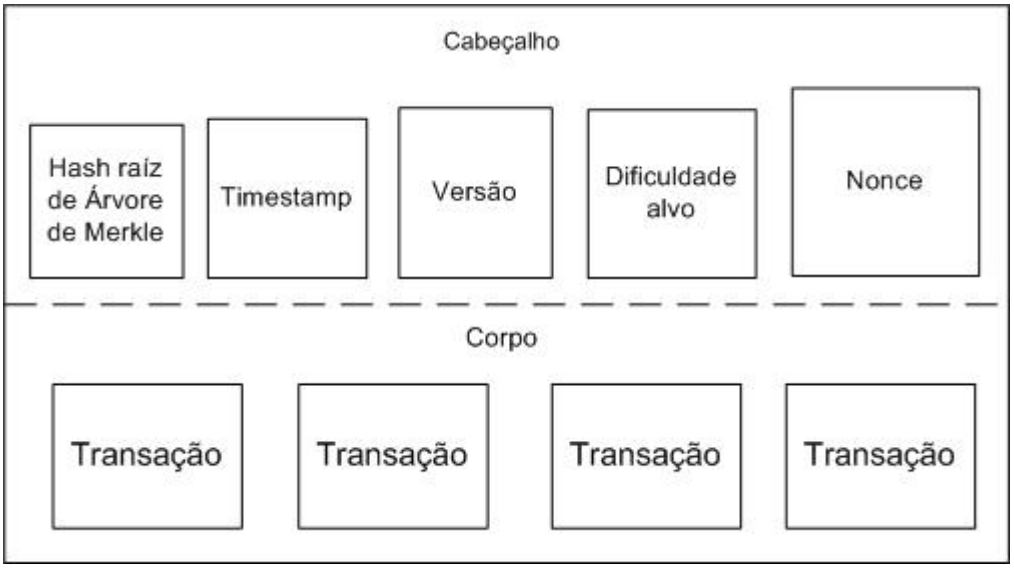

Figura 3.2: Estrutura do bloco - Na figura podemos ver as principais informações contidas em um bloco.

Em seguida, são definidas as informações contidas no bloco, explicitando qual a importância de cada uma delas.

Hash Raiz – As Árvores de Merkle são tipos de árvores binárias, nas quais cada

folha é marcada com um hash de um bloco de dados e cada não folha é marcada com o hash criptográfico de seus nós-filhos.

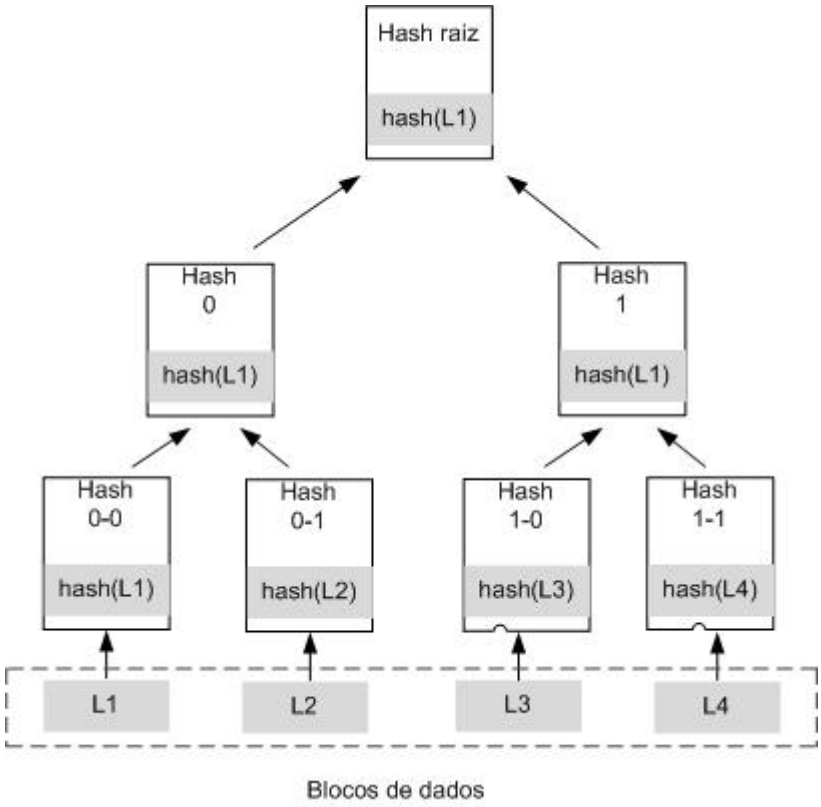

Figura 3.3: Arvore de Merkle - Na figura pode-se ver como pode ser obtido um *hash* raiz.

O processo explicitado na Figura 3.3 é usado no cálculo do *hash* de um bloco em uma cadeia. São calculados *hashes* de diferentes conjuntos transações, representada na Figura 3.3 pelos blocos de dados, e então repete-se o processo, reaplicando a função nos *hashes* resultantes da operação anterior até que reste apenas um *hash*. Desta forma, cada nó da árvore contém um resumo de informações referente a todos os seus descendentes e, portanto, o *hash* raiz é um valor, geralmente representado em hexadecimal, que resume toda a informação contida no bloco.

Timestamp – É um valor relacionado à medição de tempo Unix, que marca o número de segundos que se passaram desde 0:00:00 quinta-feira, 1º de janeiro de 1970. Atua também como uma fonte de variação para o hash do bloco.

Versão – Descreve a estrutura dos dados dentro do bloco. É usado para que membros da cadeia consigam ler a informação de cada bloco corretamente.

Dificuldade alvo – É um número que determina a rapidez com que blocos novos serão inseridos na cadeia. A dificuldade alvo estabelece uma quantidade mínima de alga-

rismos zero que o hash raiz deve conter para poder ser inserido na cadeia.

Nonce – É o último valor a ser inserido no bloco, ele deve ser escolhido de tal forma que o hash do bloco resultante após sua inserção obedeça ao critério imposto pela dificuldade alvo. O método usado para descobrir um valor apropriado para o *Nonce* é a força bruta, tentativa e erro, usado no processo de mineração do PoW.

3.3 Contratos Inteligentes

Os contratos inteligentes (*Smart Contracts*) podem ser definidos como um protocolo de transações que executa termos de uma contrato [14]. Eles são usados para forçar os envolvidos em transações a seguirem protocolos pré-determinados, aumentando a confiança no algoritmo e também a flexibilidade. Os contratos inteligentes vão além da ideia inicial de um sistema monetário em uma rede par a par, ao introduzir diversas outras funcionalidades ao sistema como, por exemplo, a possibilidade de transações de outros ativos além de criptomoeadas.

A implementação dos contratos inteligentes se dá através da inserção de códigos no livro-razão da cadeia, que funcionam como cláusulas de um contrato, sendo todas estas respeitadas em quaisquer cenários. Os contratos inteligentes podem ser invocados através de uma transação que aponta para seu endereço único, contido na cadeia de blocos. Feito isso, eles são executados automaticamente e seu resultado é salvo em cada nó da rede, assim como a transação que o invocou. Vale notar que, para que os nós possam implementar os contratos inteligentes, eles devem ter uma máquina virtual ativa, e portanto a cadeia de blocos pode ser vista como uma máquina virtual distribuída.

Uma premissa dos contratos inteligentes é que os códigos que os constituem sejam determinísticos, isto é, os seus resultados gerados a partir de uma mesma entrada devem ser sempre iguais. Isso é fundamental pois a consistência é um dos pilares das cadeias de blocos e, caso os contratos inteligentes fossem não-determinísticos, ou ela não seria respeitada em todos os casos ou seria impossível atingir um consenso entre os nós sobre as transações ocorridas na rede. Outra premissa de contratos inteligentes é que todas os possíveis resultados da transação devem ser por ele descritos, evitando assim casos em que não há um algorítimo a ser seguido.

Dessa forma, ao funcionar de modo completamente previsível, os contratos inteli-

gentes possibilitam que partes de uma negociação verifiquem o seu resultado antes mesmo de ela ser executada, podendo assim decidir a *priori* se vão participar, ou não, da transação. Outra vantagem é que todas as interações com um contrato ocorrem via mensagens assinadas na cadeia de bloco com as chaves privadas das partes. Desse modo, todos os participantes têm um rastreio criptográfico verificável das operações do contrato [15].

3.4 Aplicações

O caráter disruptivo das cadeias de blocos já vem provocando grandes mudanças em áreas como mercado financeiro, *supply chain*, saúde, entre outros. Abaixo são apresentadas brevemente algumas de suas aplicações.

- Mercado Financeiro - Os mercados de capital, área responsável por gerar liquidez aos títulos, reconhecidos por terem uma infraestrutura lenta e possuírem processos lentos e burocráticos, é uma das inúmeras áreas do mercado financeiro que está sendo impactada positivamente com as cadeias de blocos, tendo custos reduzidos, melhoras significativas na eficiência dos serviços e também na sua segurança e privacidade. Outra área na qual a cadeia de blocos irá trazer benefícios é a de transações internacionais, cujos intermediários, geralmente bancos, devem ter seu pepel reduzido ou até substituído pela tecnologia.

- Internet das Coisas - Uma alternativa para armazenar o alto consumo de dados gerado pelas dezenas de bilhões de novos aparelhos que estarão conectados na internet até o ano de 2020 são as redes descentralizadas, oferecendo mais segurança e confiabilidade do que um armazenamento centralizado. A tecnologia de contratos inteligentes pode prover maneiras de trocar informação e tornar possíveis cooperações entre dispositivos que não necessariamente confiam uns nos outros, sem a necessidade de agentes intermediadores.

- *Supply Chain* - Este é provavelmente o setor em que a adoção da tecnologia de cadeia de blocos têm-se mostrado mais promissora. Os principais motivos por trás disso estão ligados à grande quantidade de intermediários, que fazem as operações serem mais custosas e também mais lentas, já que deve haver um processo rigoroso de autenticação cada vez que um produto troca de custódia. Conforme mostrado,

as cadeias de blocos são uma alternativa a sistemas centralizados e com muitos intermediários e por isso ajudariam a solucionar grande parte dos desafios enfrentados pelo setor, além de agregar mais rastreabilidade e transparência para sistemas de logística pois toda a mudança na localização de um produto será documentada na cadeia, podendo ser acessada a qualquer momento.

- Saúde - As cadeias de blocos podem facilitar os tratamentos e planos personalizados para cada cliente de um plano de saúde. A criação de um histórico médico pessoal e familiar fácil de se verificar e compartilhar pode abrir o caminho para que os planos de saúde consigam desenvolver pacotes e exames rotineiros mais personalizados, o que pode ser atingido através de uma cadeia de bloco. Outro uso em potencial da tecnologia é com os registros eletrônicos médicos, que podem conter informações como histórico de vacinas e resultados de exames, por exemplo, com a possibilidade de serem acessados em tempo real por médicos e profissionais da saúde.

Ainda que as cadeias de blocos possam ter aplicações nos mais variados setores existentes hoje, talvez no futuro sua maior contribuição seja possibilitar a criação de áreas até então impensadas. Um exemplo são os conceitos de empresas independentes com funcionamento todo baseado em contratos inteligentes, o que era inconcebível, ao menos em termos de implementação, até poucos anos atrás.

Capítulo 4

Mecanismos de Consenso

Os mecanismos de consenso podem ser binários, multi-valorados, vetoriais e baseados em Paxos [16]. Um algoritmo de consenso binário visa obter consenso sobre um valor binário, verdadeiro ou falso. Cada processo propõe um valor binário inicial e decide sobre o valor v. O problema do consenso é formalmente definido em termos de três propriedades, $\mathbf{validade}$ (*validity*), $\mathbf{acordo}$ (*agreement*) e $\mathbf{terminação}$ (*terminations*). Validade assegura que se todos os processos corretos propõem o mesmo valor v, qualquer processo correto converge para v. A propriedade de acordo define que não há dois processos corretos que decidem diferentemente. terminação garante que cada processo correto eventualmente decida. As duas primeiras propriedades são propriedades de segurança, de acordo com a definição de Lamport, enquanto a última é a propriedade vivacidade. O consenso multi-valorado é semelhante ao consenso binário, contudo ele permite que os nós decidam sobre um conjunto de valores de tamanho arbitrário, enquanto o binário tem um domínio de saída de apenas um dígito. No entanto, a definição de validade neste tipo de consenso introduz a possibilidade de decidir sobre um valor nulo. O consenso vetorial propõe uma propriedade de validade diferente em comparação com as versões anteriores do problema. A decisão não é mais um valor único, mas um vetor com alguns dos valores iniciais dos processos e, pelo menos $f+1$, a maioria, são processos corretos. Um tipo de definição ligeiramente diferente é a utilizada nos algoritmos Paxos. A ideia é que os processos desempenhem um ou mais dos seguintes papéis: proponentes que propõem valores; aceitantes que escolhem o valor a ser decidido; e aprendizes que aprendem o valor escolhido. A propriedade segurança do consenso é definida em termos de que apenas um valor proposto é escolhido; apenas um valor único é escolhido; um aprendiz correto aprende apenas um valor proposto. A

vivacidade é definida em termos de algum valor proposto ser finalmente escolhido e, após a escolha de um valor, os aprendizes corretos aprendem-no. Embora as definições conduzam a modelos de sistemas práticos, a ocorrência de falhas bizantinas, caracterizadas em casos em que não é possível definir se um sistema está apresentando comportamento normal ou defeituoso, dificulta a implementação de mecanismos de consenso, já que os nós se desviam do comportamento esperado das mais diversas formas possíveis [17].

4.1 Mecanismos de Consenso Probabilísticos

Fischer *et al*. provam a impossibilidade de resolver consenso deterministicamente em um sistema assíncrono se um único processo pode falhar. Este resultado é frequentemente referenciado como impossibilidade FLP, nomeada em reconhecimento a seus autores, Fischer, Lynch e Paterson [18, 19]. A prova se baseia em uma definição enfraquecida do problema do consenso, em que a validade é flexibilizada, e a terminação é necessária apenas para um único processo. A incerteza em relação ao tempo da resposta de resposta, por se tratar de um sistema assíncrono, somada à incerteza em termos de falhas, leva à conclusão de que nenhum algoritmo determinístico pode garantir a tomada de uma decisão. Há uma impossibilidade de um sistema assíncrono diferenciar um processo travado de outro que é meramente lento e que eventualmente terminará.

Por um lado, a impossibilidade FLP indica quando o consenso não pode ser alcançado. Por outro lado, Dolev *et al*. identificam cinco parâmetros que afetam a solução do problema do consenso: sincronização entre processos, sincronização de comunicação, ordenação de mensagens, *broadcast* ou comunicação *peer-to-peer*, e atomicidade do envio e da recepção [20]. Assim, para resolver o problema de consenso, o algoritmo deve contornar o resultado da impossibilidade de FLP, em outras palavras, ele só pode ser alcançado se uma das três propriedades, validade, acordo e terminação, não for satisfeita. Como resultado, os algoritmos podem resolver consensos se os processos seguirem fielmente o algoritmo ou, se os processos escaparem ao comportamento esperado, o algoritmo sempre satisfará as propriedades de segurança, mas pode sacrificar a vivacidade. A ideia é que, em caso de falha, não há suposição de tempo adicional, e o algoritmo pode não terminar, mas as propriedades de validade e acordo serão sempre satisfeitas. Algumas técnicas para contornar a limitação FLP sacrificam o determinismo, levando a algoritmos probabilís-

ticos; técnicas que adicionam sincronização ao modelo do problema; técnicas que usam hibridação; técnicas que enriquecem a definição do problema [16]. Portanto, mecanismos de consenso determinísticos criam modelos de sincronização, muitas vezes baseados em protocolos de rede e transporte [21, 22], ou usam soluções híbridas [23].

Os mecanismos probabilísticos de consenso são uma classe de mecanismos de consenso que sacrificam o determinismo como forma de contornar a impossibilidade FLP [16]. Dessa forma, eles trocam sua convergência determinística por convergência probabilística. Assim, uma rodada de mecanismo de consenso probabilístico pode chegar a um consenso com uma certa probabilidade. Considerando o Teorema CAP, plataformas de cadeia de blocos baseadas em um mecanismo de consenso probabilístico tipicamente sacrificam a consistência em favor da tolerância à partição e da disponibilidade.

4.1.1 Proof of Work

A ideia por trás desse mecanismo surgiu na primeira metade da década de 1990 em um artigo de Cynthia Dwork e Moni Naor como uma forma para combater e-mails *spam* [24]. *Proof of Work* foi o mecanismo de consenso escolhido por Satoshi Nakamoto para o Bitcoin [2] e por isso ganhou grande popularidade. Apesar de servir bem a seu propósito no contexto do Bitcoin, esse mecanismo possui algumas desvantagens. A principal delas está relacionada ao desempenho, isto é, a rapidez com que se consegue inserir um bloco na cadeia.

Esse mecanismo é usado em um contexto de cadeias de blocos não permissionadas, isto é, cadeias abertas ao público. Em um cenário desse tipo o número de nós é geralmente grande e cada um deles é anônimo, assim, não há nenhum tipo de confiança entre os participantes da cadeia, o que dificulta o trabalho do mecanismo de consenso. Além disso, deve-se considerar a possibilidade de existência de nós maliciosos e também ataques como o Sybil, no qual um usuário pode ter duas ou mais identidades, o que pode influenciar ou até manipular o processo de atingimento do consenso.

O funcionamento do *Proof of Work*, cuja motivação é evitar o problema do duplo gasto, baseia-se na resolução de um desafio criptográfico. Os nós responsáveis por resolver tal desafio são denominados mineradores e o processo de resolução do problema é chamado de mineração, que é basicamente um método de tentativa e erro, isto é, força bruta.

A mineração consiste em encontrar um determinado valor numérico, *nonce* crip-

tográfico, a ser inserido no fim do bloco que se está tentando introduzir na cadeia. Isso deve ser feito de tal forma que o valor gerado pela função de resumo criptográfica *hash* do bloco em questão tenha uma certa quantidade pré-definida de números zero em seu início. Uma vez solucionado o problema, o valor encontrado é difundido através da rede para ser validado pelos pares e, após mais da metade da cadeia conferir, o bloco ser inserido de fato na cadeia. A fim de incentivar a mineração, é oferecida uma recompensa ao minerador, ou grupo de mineradores, que primeiro encontrar a resposta do desafio.

É importante que o processo de validação, que nesse caso se trata de calcular o hash do bloco e verificar se ele tem a quantidade de zeros demandada em seu início, seja relativamente rápido. Isso deve ocorrer para fins de desempenho da cadeia e também para evitar possíveis ataques de DDoS, ataques distribuídos de negação de serviço, (*Distributed Denial of Service*), em que inúmeros nós maliciosos poderiam falsamente clamar ter resolvido o desafio criptográfico, e, com um processo de validação lento, causar congestionamento ou até queda de serviço em certos nós da cadeia.

Por outro lado, o tempo para um bloco ser minerado também não deve ser muito curto para evitar que ramificações surjam com mais frequência na cadeia. Ao surgirem ramificações, a cadeia deve entrar em consenso sobre qual caminho irá seguir e qual caminho será descartado, o que demanda certo tempo. Caso o período de surgimento dos ramificações seja menor do que o tempo de resolução deles, o sistema seria inviável. O mecanismo de PoW consegue contornar essa questão ajustando o nível de dificuldade do desafio criptográfico.

A probabilidade de dois mineradores conseguirem resolver o desafio criptográfico e difundir suas distintas versões do bloco a ser inserido na cadeia aumenta conforme diminui o tempo de resolução do problema. No PoW o tempo médio que um bloco leva para ser minerado pode ser ajustado de acordo com a quantidade demandada de zeros no hash do bloco: quanto maior for o número de algarismos nulos, mais difícil será o problema e, consequentemente, mais tempo levará para ser resolvido.

A contrapartida da atuação do *Proof of Work* em cadeias de blocos não permissionadas é o desempenho. Isso ocorre, dentre outros motivos, pois a verificação de resolução do desafio deve ser feita por, no mínimo, mais da metade da rede, e também pelo fato de o tempo para solucionar o desafio criptográfico não poder ser curto, para evitar ramificações.

Além disso, outra desvantagem do protocolo são os custos de energia associados à

mineração, devido ao fato de essa atividade demandar um grande poder computacional. Na prática, o que ocorre é que a maioria dos blocos são minerados por conglomerados de mineradores, denominados *pools*, que trabalham em paralelo para resolver o desafio. Isso é um problema já que a formação desses *pools* pode ir de encontro, dependendo de seu tamanho, a uma das premissas fundamentais de uma cadeia de blocos, que é a descentralização.

4.1.2 Proof of Stake

O protocolo *Proof of Stake* (PoS) foi criado para superar as dificuldades de eficiência que o protocolo *Proof of Work* (PoW) gera. A necessidade energética para a execução de um protocolo PoW é significativamente alta, uma vez que o número de operações de *hashing* realizadas para gerar um único bloco pode ultrapassar 2^{60} na sua maior plataforma, o Bitcoin.

A ideia por trás do mecanismo de consenso do *Proof of Work* é que um líder minerador emergirá através da competição e será responsável por gerar o próximo bloco a ser inserido na blockchain. A fim de estimular a competição e manter a geração dos blocos, uma recompensa é dada ao minerador que completou primeiro o desafio de computação estabelecido. As tensões aparecem com a escalabilidade da rede, mais pares tentando resolver o quebra-cabeça criptográfico implicam um aumento na dificuldade do problema, a fim de evitar ramificações, e, como o tempo médio para extrair um bloco deve permanecer aproximadamente o mesmo, o aumento no uso de energia para extrair um bloco é inevitável. Nas plataformas cadeia de blocos baseadas em *Proof of Work*, o crescimento da rede é diretamente proporcional ao aumento no uso de energia para chegar a um consenso.

Ao invés de fazer a eleição do líder minerador como uma corrida computacional, o mecanismo de *Proof of Stake* sugere que um líder deve ser escolhido de forma aleatória, mas ao invés de recursos computacionais a eleição deve levar em conta a quantidade de posses (*stakes*) de cada minerador, de acordo com o atual livro-razão da blockchain. Esta mudança de paradigma tornou o PoS consideravelmente mais eficiente do que seu predecessor.

A aplicação mais conhecida que usa um mecanismo de consenso baseado na *Proof of Stake* é a moeda criptográfica Ethereum. Nesse caso, os validadores, nós que participam do protocolo de consenso (análogo aos minerador), depositam uma certa quantidade de

moedas (aposta) na rede, então um algoritmo seleciona o validador responsável por forjar o novo bloco de forma parcialmente aleatória. A quantidade de moedas que cada validador compartilhou é proporcional à chance de ser selecionado para montar as transações de forma linear, mas, por outro lado, a aleatoriedade impede que nós ricos monopolizem o processo. Além disso, as apostas compartilhadas na rede podem ser entendidas como depósitos de segurança e, se um validador é notado não se comportando honestamente, ou seja, incluindo viés no processo de validação, suas apostas são confiscadas. Em contraste, se o nó selecionado se comportar como esperado, ele receberá uma recompensa em moedas, da mesma forma que o PoW.

O ataque de 51% é uma fragilidade bem conhecida em sistemas baseados em prova de trabalho e que se torna evidente quando um grupo de mineradores opera em conjunto, um *pool* de minerador, controlando mais da metade do poder computacional da rede. Quando isso ocorre, não há garantia de que todo o processo de geração de novos blocos esteja sendo feito de forma imparcial porque, mesmo que os outros 49% da rede discordem sobre a validade de determinado bloco, o pool maior ainda poderá inseri-lo na cadeia. O *Proof of Stake* reduz drasticamente o risco deste tipo de ataque, já que o indivíduo ou grupo deve deter a maioria das moedas na rede, o que é caro e pode ser facilmente detectado e evitado pelos outros nós.

4.1.3 Delegated Proof of Stake

Delegated Proof of Stake (DPoS) prosseguiu com os aumentos de velocidade e escalabilidade proporcionados pelo mecanismo PoS. Os objetivos deste sistema são melhorar o desempenho, diminuir o tempo de transação e o tempo de geração de blocos, e ganhar flexibilidade, sem comprometer a estrutura descentralizada. Algumas das plataformas de cadeia de blocos que utilizam um mecanismo de consenso baseado em DPoS são Bitshares, Ark, EOS, Lisk e Steem.

Os trabalhos do DPoS são baseados na votação de aprovação, onde cada eleitor pode selecionar qualquer número de candidatos, e o vencedor é o mais aprovado (votado). Diferentemente do PoS, que seleciona pseudo-aleatoriamente um nó para ser responsável pela geração de um novo bloco e outros ficam responsáveis pela verificação de transações, os *stakeholders* de um sistema DPoS, primeiro elegem um forjador e depois concordam com um número de nós, denominados testemunhas, que são responsáveis pela validação

das atividades.

O número de testemunhas é definido de forma que pelo menos 50% dos participantes votantes concordem no nível de descentralização e, uma vez concluída essa tarefa, a votação ocorre a fim de eleger o forjador, que será o responsável por ordenar as transações e construir o bloco. Cada voto tem um peso que é proporcional à quantidade de participação que o eleitor tem. Portanto um usuário não precisa ter uma quantidade significativa de participação para ter grandes chances de ser o gerador do bloco, bastando ter uma votação expressiva.

Os únicos nós responsáveis pela validação das transações são as testemunhas eleitas pelos *stakeholders*. Reduzir a quantidade de validações feitas por bloco inserido, sem perder o fator de segurança que envolve a votação por aprovação, é o elemento chave por trás do aumento substancial do desempenho da DPoS quando comparado ao PoS.

Por outro lado, os sistemas de *Delegated Proof of Stake* proporcionam flexibilidade à rede. As regras e parâmetros, como o tamanho das transações e a tabela de taxas, podem ser ajustados democraticamente mesmo após a criação do bloco de gênese. Isso pode ser feito por meio de comitês, que são compostos por delegados eleitos que têm o papel de manter e gerenciar a rede, incluindo a organização do processo de seleção de testemunhas.

Usar um sistema eleitoral ponderado com base na participação do eleitor pode levar a problemas, como usuários menores que não participam da votação depois de considerar seu voto insignificante. Outra desvantagem do DPoS é que todos os delegados permanecem bem informados e interessados, a fim de selecionar testemunhas adequadas e proporcionar suficiente descentralização, o que pode ser difícil de garantir.

4.1.4 Proof of Elapsed Time

O objetivo do *Proof of Elapsed Time* (PoET) é substituir o alto desperdício de recursos imposto pelo PoW através da inclusão de um tempo de espera específico gerado por hardware confiável [25]. O hardware proposto para esta tarefa é o *Intel Software Guard Extension* (SGX) disponível em muitos processadores Intel modernos. O Hyperledger Sawtooth Lake [1] plataforma de cadeia de blocos implementa PoET.

No consenso do PoET, em cada rodada, cada nó executa uma etapa de espera

[1] Disponível em `https://www.hyperledger.org/projects/sawtooth`

cujo valor é gerado pelo SGX. Cada nó chama um codificador dentro do SGX para gerar um atraso aleatório. Após esperar o tempo necessário, um nó pode se declarar líder em uma rodada de consenso e gerar um novo bloco para a blockchain. O módulo também cria um certificado que pode ser usado por qualquer nó para verificar se o líder esperou corretamente pelo tempo aleatório apropriado. Assumindo que um atacante não pode subverter o módulo de hardware, o PoET fornece um mecanismo de consenso com as mesmas propriedades do PoW, mas sem o desperdício de recursos causado pela mineração. No entanto, o investimento econômico ainda influencia o protocolo porque a probabilidade de um nó se tornar líder é proporcional ao número de módulos de hardware sob seu controle.

Apesar de apresentarem as mesmas propriedades que o PoW, os resultados experimentais mostram que o PoET proporciona menor tolerância [25]. Se um atacante pode subverter pelo menos $\Theta(\frac{\log\log n}{logn})$ nós, em que n é o número de nós na rede, ele pode simular o comportamento do nó honesto mais rápido de maneira indetectável.

4.1.5 Ripple Consensus Mechanism - RPCA

O Ripple é um projeto é um projeto que objetiva funcionar como uma moeda criptográfica (XRP) e como uma rede de pagamento para transações financeiras. Ambas as aplicações têm subjacente o Algoritmo de Consenso do Protocolo de Ripple, que pode suportar falhas $(n-1)/5$, em que n é o número de servidores na rede. As principais vantagens do Protocolo Ripple são as melhorias fornecidas em utilidade, conceito definido por Brad Chase e Ethan MacBrough [26] que inclui aspectos como poder computacional envolvido na participação do consenso, mas que, de modo simplificado, por ser entendido como latência do sistema.

Os principais componentes de uma cadeia que faz uso do RPCA são descritos a seguir.

- *Server*: todas as entidades que participam ativamente no processo de consenso são chamadas de servidores. Eles executam o software Ripple Server;
- Livro-razão: livro-razão é um registro das transações válidas que passaram pelo processo de consenso. O livro-razão também é atualizado com a quantidade de moeda criptográfica que cada usuário possui;

- *Open Ledger*: as novas operações propostas pelos utilizadores finais são armazenadas em livros-razão abertos. Assim, os livros-razão abertos têm transações que ainda não foram validadas pelo mecanismo de consenso. Cada nó tem sua própria versão de livros-razão abertos;

- *Last-Closed Ledger*: quando um livro-razão aberto passa pelo consenso, torna-se o último livro-razão fechado. Consequentemente, este é o livro-razão mais recente possuído por todos os nós da rede.

- *Unique Node List*: a *Unique Node List* (UNL) é um registro mantido por um servidor que contém uma lista de outros servidores confiáveis que podem não conluiar contra transações propostas justas. Somente membros da UNL do servidor que propôs a transação podem ter votos válidos no processo de consenso. Vale ressaltar que não é necessário que todos os membros da rede tenham um comportamento imparcial, mas a UNL como um todo deve ter.

- *Proposer*: Os proponentes são servidores que transmitem transações para serem incluídas no processo de consenso. O consenso é dividido em rodadas. Em cada rodada, qualquer servidor pode se tornar um proponente, tentando incluir transações no *ledger*.

Cada *round* do consenso tem as mesmas fases. Primeiro, cada servidor reúne todas as transações armazenadas em livros-razão abertos, formando uma lista. A lista é o conjunto de candidatos a ser validado na fase seguinte pelos membros da UNL. A segunda fase é a votação pelos membros da UNL. Com base nos votos acumulados, cada nó refina seu conjunto de candidatos, e as transações que recebem o maior número de votos são passadas para a próxima rodada [27]. As transações que recebem 80% ou mais de votos concordando com sua validade são então aplicadas ao livro-razão. Caso contrário, as transações são descartadas ou incluídas na lista de candidatos para a próxima rodada de consenso.

4.2 Mecanismos de Consenso Determinísticos

Os sistemas de computadores confiáveis devem lidar com componentes maliciosos ou avariados, que causam informação contraditória em diferentes partes do sistema. Lamport

define este problema como o Problema Geral Bizantino [1], já explicado no Capítulo 2. Na analogia apresentada, os generais representam os computadores em uma rede, e a invasão da cidade representa uma tarefa que deve ser executada em conjunto por esses computadores. As plataformas cadeia de blocos devem ser capazes de lidar com esse tipo de problema, e é papel dos mecanismos de consenso encontrar a solução mais adequada para cada ambiente em que a plataforma atua. A formulação do problema é fundamental para o estudo dos mecanismos de consenso em sistemas distribuídos, pois estabelece um limite mínimo teórico do número de nós bem comportados que devem existir em uma rede com um número fixo de nós maliciosos sem comprometer seu correto funcionamento. Mostra-se que não é possível ter um sistema tolerante a este tipo de falhas, conhecidas como falhas bizantinas, se o número total de nós for inferior a $3f + 1$, onde f indica o número de nós maliciosos [23].

As atuais plataformas de cadeia de blocos baseadas em mecanismos de consenso determinístico tolerante a falhas bizantinas falhas tendem a sacrificar a disponibilidade em favor de uma forte consistência [28, 3, 29]. Mecanismos de consenso determinísticos que não toleram falhas bizantinas tendem a adotar um modelo de consistência eventual [30, 17]. Vale a pena notar que as recentes propostas de consenso em cadeia de bloco introduzem as ideias de (i) executar o consenso BFT em uma janela deslizante entre nós [31] e (ii) adicionar um caminho rápido e assinaturas de limite baseadas em grupos para reduzir a carga de rede dos protocolos BFT [32, 33]. Estas propostas sacrificam parcialmente a tolerância à partição das plataformas.

4.2.1 Paxos

O protocolo Paxos [34] tem o nome de um sistema de consenso fictício idealizado na ilha grega de Paxos. É um mecanismo tolerante a falhas, uma vez que suporta uma determinada fração (menos de metade) dos nós com erros de rede. Contudo, ele assume a inexistência de conluio ou mensagens corrompidas, que tentam subverter o protocolo e, portanto, não suporta falhas bizantinas.

O Paxos é composto por cinco tipos de nó com diferentes funções no protocolo. Eles são:

- Clientes - responsáveis por escrever os pedidos no sistema distribuído.

- Aceitante (votantes) - estes nós atuam como memória do sistema, garantindo a tolerância a falhas do protocolo. São organizados em grupo denominado *quorum*, sendo que qualquer mensagem recebida por um membro aceitante de um quorum deverá ser recebida também por todos os demais membros do *quorum*, caso contrário, será descartada. O mesmo funciona para as mensagens recebidas pelos membros do *quorum*: se não forem recebidas por todos eles, serão ignoradas.

- Proponente - o proponente atua ativamente para buscar consenso. Estes nós pressionam os aceitantes a concordar com uma solicitação e tentam resolver quaisquer conflitos quando eles ocorrerem, atuando como um coordenador.

- Aprendiz - Os aprendizes são responsáveis por responder às solicitações dos clientes. Uma vez que todos os aceitantes tenham validado uma solicitação, os aprendizes a executam e devolvem os resultados aos clientes.

- Líder - Os líderes são tipos especiais de proponentes, que tentam garantir a propriedade de segurança do sistema. Sua presença é necessária para o correto funcionamento do Paxos e, se mais de um proponente atuar como líder, o protocolo possivelmente terá atualizações conflitantes, de modo que só garantirá o progresso se um único líder for escolhido.

O consenso ocorre em duas fases: a fase 1 consiste na preparação do pedido e a fase 2 está relacionada com a aceitação do mesmo.

Na primeira fase, o proponente envia ao *quorum* uma mensagem preparatória, com um identificador. Então cada aceitante verifica o número do identificador, e se ele for maior que o último valor, o aceitante retorna uma mensagem de promessa ao proponente com o valor do identificador da última requisição submetida. Caso o identificador tenha um valor menor, na versão original de Paxos, o aceitante não enviará nenhuma mensagem, não validando a requisição.

Quando um proponente recebe uma mensagem de promessa de todos os membros do *quorum*, ocorre a fase dois. Em seguida, o proponente prepara e envia uma mensagem de aceitação, com um número superior ao do identificador do último pedido submetido enviado pelos aceitantes, forçando-os a concordar com a submissão. Assim, o *quorum* valida a mensagem e responde aceito ao aceitante.

4.2.2 Practical Bizantine Fault Tolerance - PBFT

Este mecanismo de consenso atua em um cenário padrão em sistemas distribuídos e considera a existência de nós conectados por uma rede, que podem ou não entregar todas as mensagens, atrasar a sua entrega, duplicá-las, ou entregá-las fora de ordem. O PBFT [3] também parte do princípio de que as falhas são independentes e que os nós confiam parcialmente uns nos outros, configurando um ambiente de cadeias privadas. É implementado em plataformas de desenvolvimento de cadeias de blocos como a Zylliqa e o Hyperledger Fabric.

Seu objetivo é, conforme indicado pela sua denominação, atuar como uma solução prática para o Problema dos Generais Bizantinos apresentado anteriormente. Portanto, o PBFT garante segurança e vivacidade mesmo com a existência de (n-1)/3 nós maliciosos de um total de n nós. Destarte, ele procura evitar falhas gerais, mitigando o efeito da atuação dos nós maliciosos no sistema.

O algoritmo consiste em quatro partes, que seguem o formato General-Tenente, uma variação do modelo original, onde existiam apenas Generais. Nesse contexto, os nós seguem uma hierarquia representada pela existência de um nó líder. As quatro etapas são:

1. Um cliente envia um requerimento de transação para o nó líder.
2. O líder então transmite o pedido através da rede para outros nós.
3. Esses outros nós então executam o pedido e enviam uma resposta ao cliente.
4. O cliente espera a chegada de f+1 respostas com o mesmo resultado, onde f representa o número máximo de nós que podem ser faltosos.

Os nós devem ser determinísticos e partirem todos do mesmo estado inicial. Os resultados desse algoritmo são que os nós honestos chegam a um acordo quanto à ordem das transações e se eles devem ou não aceitá-las. Assim, a maioria de nós honestos impediria a ação de faltosos através da comunicação estabelecida entre eles.

O PBFT resolve o problema de nós líderes faltosos com um modelo de troca baseado em *Round-Robin*. Essa troca é acionada uma vez que o líder passa determinado período sem requisições. Outro modo de troca de líderes ocorre quando a grande maioria dos nós

decide que o líder é faltoso e derruba-o e, então, o próximo da fila assume a posição do líder.

Algumas das vantagens do PBFT estão relacionadas ao fato de ele possuir baixo custo energético, e, também pelo fato de esse mecanismo de consenso possuir tempo de execução muito abaixo da média para sistemas assíncronos, com apenas um pequeno incremento em latência. Contudo, em cadeias com um número de nós maior ficam evidentes as limitações do mecanismo. A grande quantidade de mensagens trocadas para se obter o consenso implica uma significativa perda de desempenho em redes maiores. Outra desvantagem é que o PBFT é suscetível a ataques Sibyl.

4.2.3 BFT-SMaRt

O BFT-SMaRt é um mecanismo bizantino de replicação de máquina de estado tolerante a falhas usado para alcançar consenso sobre plataformas de cadeias de blocos permitidas [28]. Ele baseia sua operação na eleição de um líder responsável por validar as solicitações de transações recebidas de outros nós. Não há processo de eleição neste mecanismo. Uma lista identifica os nós responsáveis pelo consenso durante a criação da rede. Sempre que um líder não consegue se comunicar com o resto da rede, o próximo nó da lista assume automaticamente a liderança.

No BFT-SMaRt caso um nó que queira propor uma transação, ele deve enviá-la para todos os outros nós da rede. No início de uma rodada de votação, o líder envia uma lista de transações para cada nó da rede. Ao receber esta lista, os nós verificam se tem a transação em sua base local e a validam enviando seu voto para os outros nós. O envio de votos para todos os nós permite verificar se um nó enviou dois votos diferentes para a mesma transação.

Ao receber todos os votos, o líder informa os nós que foram aprovados, e eles devem ser anexados ao histórico local. O líder também envia a lista de votos que recebeu para todos os nós da rede, permitindo-lhes verificar se outros nós votaram de forma diferente para o mesmo conjunto de transações, garantindo a segurança do mecanismo. Se uma transação permanece por muito tempo no banco de dados local de um nó sem validação, ou se o nó recebe uma solicitação de votação para uma transação que ele não tem, ele imediatamente solicita a última versão da informação da rede do líder.

Na presença de partições de rede, o BFT-SMaRt pode continuar recebendo solici-

tações desde que as partições resultantes tenham nós suficientes para realizar a votação. Neste caso, cada partição irá eventualmente convergir para diferentes versões da rede, comprometendo a consistência de toda a rede em troca de manter sua disponibilidade.

4.2.4 Tendermint

O Tendermint [23] propõe resolver problemas encontrados em PoW adicionando uma forma de punição aos nós maliciosos e prevenindo mais efetivamente o ataque de gasto duplo. Essa punição evita a criação de ramificações e o problema conhecido como “nothing-at-stake", em que os participantes não têm nada a perder ao agirem maliciosamente. Tendermint não requer o uso de alto poder computacional, impedindo que a validação de transações seja restrita a grupos específicos na rede. Outro fator que permite uma melhor distribuição do lucro é a divisão das taxas de validação entre os participantes de uma rodada de consenso.

O Tendermint é um protocolo de consenso capaz de punir os participantes que contribuem para ramificações da cadeia de bloco. Ele pode tolerar falhas de travamento de até $\frac{N}{3}$ nós. Os nós responsáveis pela validação das transações são chamados de validadores. Um nó precisa submeter uma parte de seu saldo atual para se tornar um validador. Enquanto ele quiser participar do processo de validação, esse valor deve permanecer comprometido.

A operação do protocolo consiste em três etapas primárias e duas adicionais. Os passos primários são: proposta, evocar e pré-comprometer. Nessas etapas, um validador, escolhido por meio de um esquema de *round-robin* ponderado pela capacidade de voto dos candidatos, é responsável por criar e propor um novo bloco a ser validado, ou concluir a validação de um bloco anterior. Se a rede aceitar um bloco durante a fase de pré-comprometimento, todos os validadores que o aceitaram ficam bloqueados até à sua confirmação. O processo de bloqueio de cada nó garante a segurança do consenso. Se a rede rejeitar um bloco, uma prova de rejeição é criada e utilizada para desbloquear os validadores.

A prova de rejeição evita que os nós sejam bloqueados indefinidamente, garantindo assim vivacidade da rede. Durante a etapa de validação, se um nó detectar a proposição de dois blocos distintos, é possível descobrir quais validadores são responsáveis por tentar ramificar a cadeia de bloco com base na chave pública presente no bloco proposto. O

protocolo então pune esses nós com a perda de um terço dos valores comprometidos com a participação no consenso. Cada validador monitora o comportamento um do outro para lidar com partições na rede. Um validador que não participa de um determinado número de rodadas de consenso é considerado desconectado da rede. Um validador desconectado tem seu *status* revogado e seu saldo não comprometido. Ao retornar à rede, o nó pode se tornar um validador novamente, comprometendo parte de seu saldo.

Capítulo 5

Hyperledger Fabric

O Hyperledger Fabric é uma das principais plataformas de desenvolvimento de cadeias de blocos existentes atualmente. O projeto do qual faz parte tem como objetivo difundir o uso dessa tecnologia e também acelerar sua adesão no mercado. Hoje, o Fabric é a espinha dorsal diversos serviços de cadeia de blocos oferecidos pelo setor privado, como a plataforma de cadeia de blocos da IBM. O caráter inovador, implementando uma arquitetura modular e mecanismos de consenso plugáveis, também faz do Hyperledger Fabric um importante alvo de pesquisas no meio acadêmico.

A Linux Foundation, tradicional organização dedicada a criação de projetos de código aberto, visando acelerar o desenvolvimento da tecnologia de cadeia de blocos, fundou o projeto Hyperledger no ano de 2015, em conjunto com diversas empresas, incluindo IBM, Intel, Fujitsu e JP Morgan. O Hyperledger Fabric é um dos projetos de cadeia de blocos dentro do escopo do Hyperledger e, assim como outras tecnologias, também faz uso de livro-razão, contratos inteligentes e tem a vantagem de ser um sistema descentralizado.

O Fabric é voltado para a construção de aplicações em cadeia de bloco no contexto de cadeias privadas e permissionadas e seu principal diferencial é o seu ambiente modular. Tal modularidade confere ao desenvolvedor a possibilidade de moldar sua cadeia de blocos de um modo mais apropriado para o caso de uso específico. Além disso, o Fabric é o primeiro sistema de cadeia de blocos que tem aplicações distribuídas executando em linguagens de uso geral, como Python, Java, Go, o que aumenta a adesão de desenvolvedores à plataforma.

Para adquirir tamanha flexibilidade, o Hyperledger Fabric adotou um tipo diferente de arquitetura do usado anteriormente em outros sistemas, conhecido como *Order-Execute*,

que primeiro ordena as transações, depois as executa sequencialmente em todos os nós da cadeia, e então as grava em ordem no livro-razão. Esse modo de funcionamento possui diversas desvantagens, como:

- Desempenho - Em aplicações com alta carga de trabalho e grande número de transações conflitantes simultâneas, resulta em um problema de desempenho e escalabilidade, já que os nós de toda a rede podem executar as mesmas transações simultaneamente.

- Confidencialidade - Como os contratos inteligentes estarão visíveis a todos, há fragilidade no que diz respeito à confidencialidade das transações. Esta poderia ser alcançada através de criptografia das transações salvas no livro-razão, mas que vem com o custo de um cabeçalho considerável e não viável na prática.

- Fragilidade - Pouca robustez contra ataques de negação de serviço (DoS, *Denial of Service*), bastando criar um *chaincode* (contrato inteligente) com tempo de execução demasiado alto, para que todos os nós da cadeia sejam impactados.

- Existência de transações não determinísticas - A existência de transações não determinísticas implica a probabilidade de existência de ramificações na cadeia, o que viola a premissa de cadeias de blocos de que todos os nós devem possuir o mesmo estado. Em geral, tecnologias de cadeias de blocos fazem o uso de linguagens de programação próprias para evitarem esse problema, como é o caso da Solidity nativa da plataforma Ethereum. Contudo, em muitos casos essas linguagens não são de fácil utilização e impõem dificuldades e necessidade de tempo de estudo ao programador.

Assim, com a intenção de contornar essas adversidades, o Hyperledger Fabric propõe um modelo de execução de contratos inteligentes em paralelo, denominado *execute-order-validate* (executar-ordenar-validar). Nele nós podem executar transações distintas em certo instante, o que implica um ganho de desempenho, já que há uma maior eficiência introduzida pelo paralelismo.

Contudo, a ordenação das transações em determinado bloco deve ser a mesma em toda a rede. Para garantir essa premissa, foram criados nós especiais denominados ordenadores (*orderers*). Esses são responsáveis pela correta ordenação das transações que serão inseridas no livro-razão. Assim, após feita a execução em paralelo das transações,

o resultado obtido é repassado para os ordenadores, validado e então é inserido no livro-razão.

Esse tipo de arquitetura contorna a falha de confidencialidade enfrentada pelo modelo *Order-Execute*. O *execute-order-validate* restringe o acesso a contratos inteligentes quando necessário, isto é, fazendo com que apenas alguns nós confiáveis o executem e validem o resultado da execução.

5.1 Componentes e Funcionalidades

Até o surgimento do Hyperledger Fabric cadeias de blocos permissionadas possuíam várias limitações e muitas dessas advinham de características herdadas de arquiteturas *order-execute*, conforme já explicitado, ou de cadeias permissionadas (públicas). Exemplos disso são o mecanismos de consenso fixos e embutidos na plataforma, e os modelos de validação de transações que não conseguem adaptar-se conforme o tipo de contratos inteligentes. Isso até mesmo contradiz o conceito bem estabelecido de que não existe um único protocolo de consenso que seja otimizado a todos os casos (“*one size fits all*”).

A arquitetura modular proposta pelo Fabric objetivava uma maior escalabilidade, flexibilidade e confidencialidade do que as encontradas em tecnologias de blockchains até então. A Figura 5.1 ilustra como o Hyperledger Fabric está estruturado.

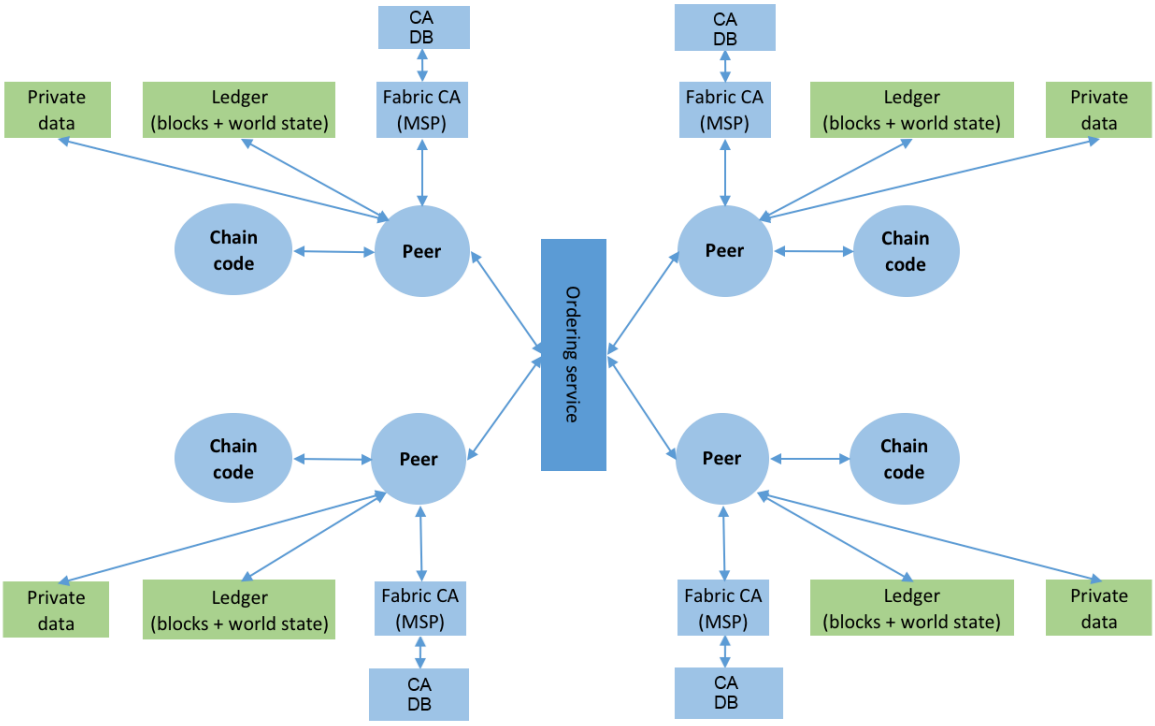

Figura 5.1: Arquitetura Hyperledger Fabric[2]- Na figura pode-se notar os principais integrantes da arquitetura do Fabric e como a relação entres eles é estruturada.

5.1.1 Peer Nodes

Os *peers* são elementos centrais da arquitetura da cadeia de blocos do Hyperledger Fabric, sendo representantes de uma organização, que pode ser definida com um grupo gerenciado de membros que participam da rede. Nos *peers* estão salvas as principais informações da cadeia, livro-razão e contratos inteligentes, garantindo o caráter distribuído de qualquer cadeia de blocos. De modo geral. um *peer* é responsável por endossar e atualizar o livro-razão, além de fazer operações.

[2]Disponível em: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781838649982/11/ch11lvl1sec76/hyperledger-fabric-architecture

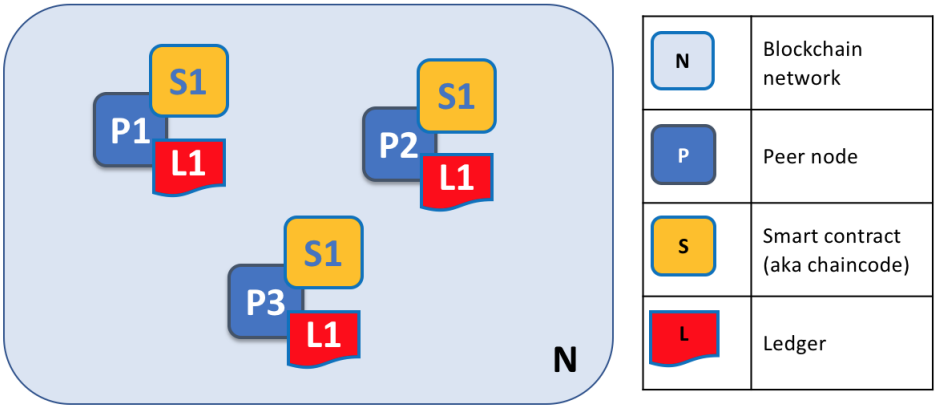

Figura 5.2: *Peers*[4]- A figura apresenta como os *peers* estão estruturados no domínio de uma cadeia

A Figura 5.2 representa o estado base em que uma cadeia de blocos pode se encontrar, em que cada um dos *peers*, P1, P2 e P3, possui um contrato inteligente S1 e um livro-razão L1. Apesar de no exemplo citado haver apenas um livro-razão associado, cada *peer* pode possuir mais de uma e também mais de um contrato inteligente (*chaincode*). Vale ressaltar que os *peers* devem possuir ao menos um contrato inteligente que os permita ter acesso ao conteúdo do livros-razão. Normalmente, o número de contratos inteligentes é igual ou maior do que a quantidade de livros-razão. Contudo, é possível a existência de *peers* que possuem apenas livros-razão, sem nenhum contrato inteligente.

Os *peers* podem assumir diversas funções em uma rede e eles podem ser classificados de acordo com cada uma delas. Abaixo estão descritas essas classificações.

- *Peers* Âncora - São conhecidos de fora de cada organização e registrados no bloco gênese da cadeia.
- *Peers* Comuns - Sua atribuição é guardar o livro-razão e endossar contratos inteligentes.
- *Peers Bootnodes* - São os responsáveis por descobrir novos *peers* e iniciar o *gossip*, que é o nome da comunicação que se dá entre nós da rede para troca de informações.

[4]Disponível em: `https://hyperledger-fabric.readthedocs.io/en/release-1.4/peers/peers.html`

- *Peers* Líderes - São os responsáveis por receber dados de outros *peers* através de protocolos *gossip*.
- *Endorsing Peers* - São responsáveis por fazer a validação de transações ocorridas na cadeia.

5.1.2 Orderers - Ordering Service Nodes - OSNs

Diversas plataformas não permissionadas, como Bitcoin e Ethereum, usam de mecanismos de consenso probabilísticos devido à natureza de suas redes e, com isso, surgem problemas como o de ramificações na cadeia. A arquitetura permissionada, em que apenas um grupo da rede está apto a atuar no algoritmo de consenso, do Hyperledger Fabric permite que sejam usados mecanismos determinísticos para se atingir a consistência dos livros-razão e com isso todo novo bloco gerado é garantidamente correto.

O Fabric separa o consenso em duas etapas, que inclusive ocorrem em tipos de nós distintos: ordenamento das transações, feito pelos nós ordenadores, e endosso dos contratos inteligentes, realizados pelos *peers*. Essa separação traz à plataforma ganhos em desempenho e escalabilidade, pois remove ineficiências, como a impossibilidade de fazer a validação e o ordenamento em paralelo, relacionadas à realização dessas duas atividades em um mesmo nó.

Os nós ordenadores, assim como todos que interagem com uma rede de cadeia de blocos, recebem suas identificações através de certificados digitais providos por Autoridades Certificadoras e domínios de confiança identificados por MSPs. Sua principal função é atuar na fase de ordenamento e agrupamento de transações em forma de blocos.

Assim, ordenadores fazem a interface entre os clientes, que vão propor atualizações, e os *peers* que vão validá-las e, de fato, realizá-las. A transmissão dessas atualizações é feita através de um canal do qual os ordenadores também realizam um controle de acesso. Portanto, após receber transações de diversas aplicações de clientes, os ordenadores, que em conjunto formam o serviço de ordenamento, definem uma ordem com o auxílio de um algoritmo de consenso e então agrupam essas transações em formas de blocos. Vale notar que a sequência em que essas transações estão dispostas no bloco não é necessariamente a mesma da ordem em que foram recebidas as transações.

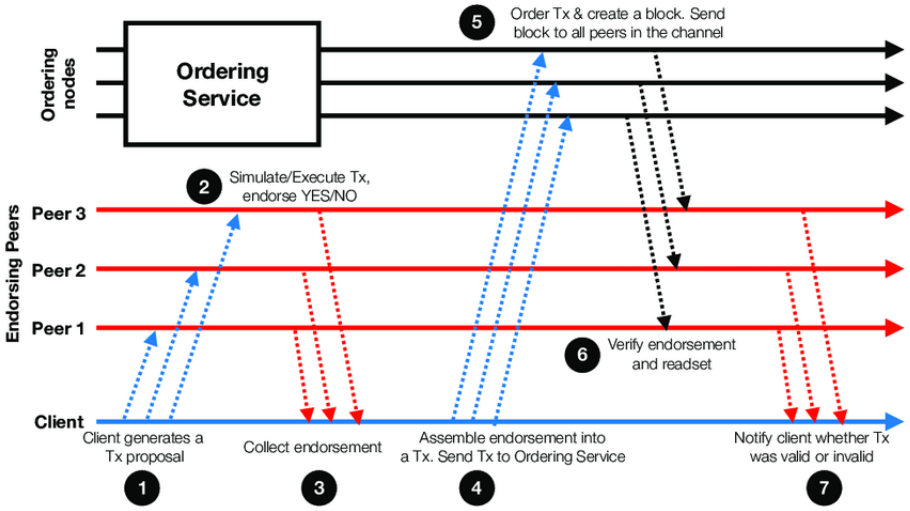

Figura 5.3: Fluxo de inserção de um novo bloco[6]- Na figura podem ser vistas todos as etapas para se inserir um bloco na cadeia.

A Figura 5.3 demonstra o fluxo completo no qual uma transação é submetida até o momento de sua inserção. Em 1, o cliente gera a proposta de transação, que é validada pelos *peers* em 2 e retornada aos clientes em 3. Em 4, o cliente envia novamente a transação válida para os ordenadores que a ordenam e inserem em um bloco e, em 5, reenviam para os *peers*, que por sua vez validam o bloco formado em 6 e enviam a resposta final para os clientes em 7. O fluxo descrito acima é a base de funcionamento de toda inserção de transações na cadeia de blocos. Contudo, há variações conforme as implementações de consenso (e serviços de ordenamento), Raft e Kafka com Zookeepér, que serão descritas na seção de mecanismos de consenso.

5.1.3 Canais

Os canais funcionam como sub-redes, no sentido de que possibilitam a existência de meios pelos quais pode haver transações entre dois ou mais membros da rede de modo totalmente separado da rede principal da cadeia de blocos. Isso confere aos participantes da cadeia a confidencialidade necessária para realizar transações com qualquer membro desse canal, já que apenas os presentes tem acesso às informações trocadas nele. Há, portanto, isolamento das informações contidas nas *ledgers* do canal, isto é, não se pode passar informações de um canal para outro.

[6]Disponível em: https://www.researchgate.net/figure/Hyperledger-Fabric-Protocol_fig1_328899592

Em um canal, os membros da rede são representados pelos *peers* âncora, *peers* líderes, e também pelos *peers* comuns. Outras partes constituintes do canal são: o chamado bloco gênesis, nos quais estão salvas as informações sobre *peers* âncora, membros e políticas do canal; o livro-razão compartilhado, onde serão salvas todas as alterações; os *chaincodes*, contratos inteligentes responsáveis pelas transações e pelo acesso a cada livro-razão; nós de ordenação, que farão a ordenação das transações a serem validadas e inseridas no livro-razão.

Para se criar um canal é necessário referenciar contratos inteligentes especiais denominados *chaincodes* de configuração de sistema. Na referência deve-se informar quem serão os *peers* âncora e os membros do canal e estes dados ficarão guardados no bloco gênese. Caso um membro novo se junte a um canal já existente, o bloco gênese ou um bloco de reconfiguração será compartilhado com ele. Todos os membros do canal têm sua identidade gerenciada pelos chamados Provedores de Serviços de Associação (*Membership Service Provider* - MSP), que serão responsáveis pelo controle de acesso ao canal, validação de permissões, autenticação.

Após todos os membros das diferentes organizações estarem presentes no canal, é necessário eleger um *peer* líder para cada organização. Caso as organizações não definam um nó em especial, um algoritmo poderá ser usado para definir tal líder. Ele será o responsável por receber um bloco do serviço de consenso e depois repassá-lo para os *peers* a ele subordinados, através de protocolos de *gossip*.

5.1.4 Certificate Authority

A identidade digital é atribuída a cada nó por meio de certificados digitais, documentos contendo atributos referentes ao possuidor do certificado incluindo sua chave pública. Esses certificados são públicos e a comunicação entre nós se dá através da assinatura da mensagem com a chave pública, que pode ser adquirida através da consulta em certificados, do nó para o qual a mensagem é destinada. A relação única entre a chave pública e a chave privada, única e secreta para cada membro da rede, é o que garante que apenas o nó de destino consiga ter acesso ao conteúdo da mensagem transmitida.

A entidade responsável pela geração dos certificados que atribuem identidades digitais aos membros da rede da cadeia de blocos é denominada Autoridade Certificadora (*Certificate Authority*- CA) e ela também possui um certificado e assina com sua chave

pública cada um dos certificados concedidos. O seu funcionamento está baseado em uma cadeia de confiança: caso a autoridade certificadora seja confiável então identidades e atributos contidos em um certificado por ela conferidos também serão confiáveis. A validação de que uma autoridade certificadora forneceu dado certificado pode ser feita através de sua assinatura em cada um deles e sua chave pública.

Existe também a possibilidade de se criar uma cadeia de certificadores (certificators chain), e com isso haveria certificadores intermediários subordinados à CA raiz (RCA - *Root Central Authority*. Em redes maiores, esse tipo de estrutura tem maior relevância, já que provê dinamismo à cadeia, distribuindo as requisições antes concentradas apenas na autoridade certificadora principal. A Figura 5.4 ilustra o funcionamento desse mecanismo.

Na Figura 5.4 pode-se ver a relação entre a RCA, autoridades certificadoras raiz, as ICA (*Intermediate Central Authority*), autoridades certificadoras intermediárias, e os certificados emitidos por cada uma delas.

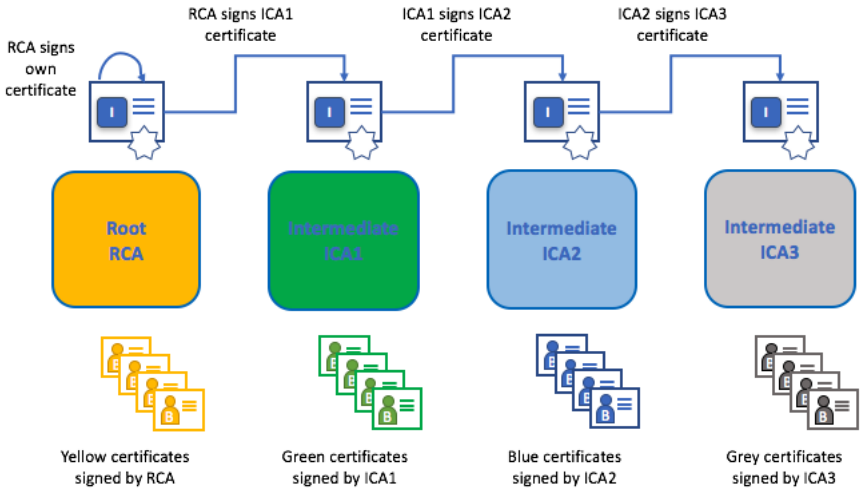

Figura 5.4: Cadeia de autoridades certificadoras[8]- A RCA gera certificados para que outras Autoridades, ICAs, possam fazer parte do processo, diminuindo a centralização em da cadeia em uma única entidade.

5.1.5 Membership Service Providers - MSPs

Os MSPs são Provedores de Serviço de Associação cuja função é identificar um domínio de confiança, isto é, um conjunto de ICAs e RCAs confiáveis que serão responsáveis por gerar

[8]Disponível em: https://hyperledger-fabric.readthedocs.io/en/release-1.4/identity/identity.html

certificados necessários. Além disso, também através de uma listagem de identidades, é responsável definir organizações e funções para cada participante listado da rede.

Ademais, o *Membership Service Provider* tem a função de distribuir credenciais, definindo privilégios. Ele contém uma lista de quem participa de cada rede e inclusive registra as funções que um nó tem em cada um de seus canais tais como: administrador de canal, permissão de leitura e escrita, etc. O gerenciamento dos membros de uma organização, independente do seu porte, é feito por um único MSP, contudo esse pode estar separado em grupos como mostra a Figura 5.5.

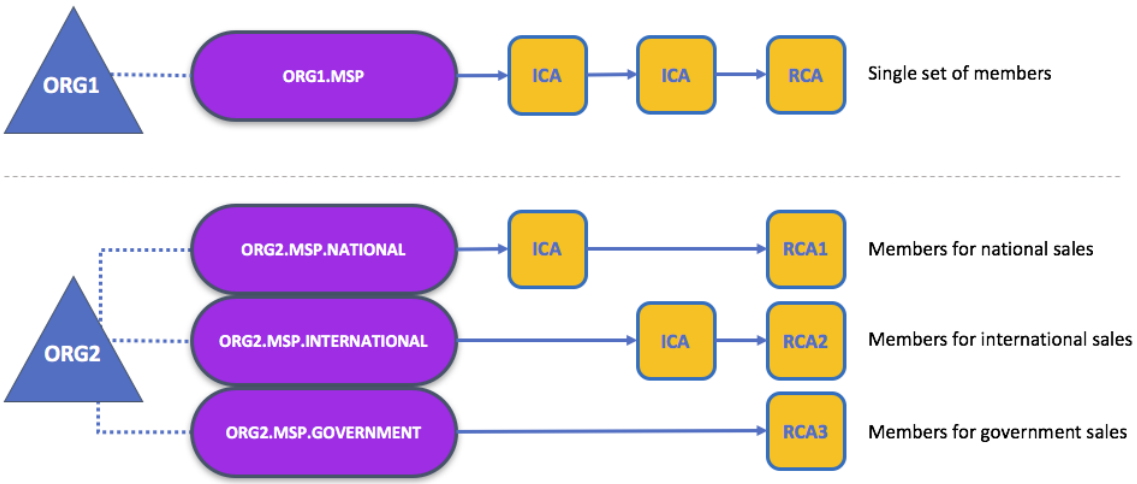

Figura 5.5: Estrutura MSPs[10]- Na figura pode-se ver diversas instâncias de um mesmo MSP provendo serviço a diferentes organizações e grupos de membros

Os MSPs podem atuar armazenando informações referentes a configuração de canais ou podem ser locais, reunindo metadados de nós (*peers* ou ordenadores). Devem existir MSPs locais para cada nó, pois eles guardam informações que permitem sua autenticação. Do mesmo modo, devem existir pelo menos um MSP de canal por organização, pois nele estão salvas dados seus administrativos, como a lista de membros do canal. Contudo, enquanto os *Membership Service Providers* locais são definidos apenas em seus respectivos nós, os MSPs de canal estão guardados em todos os nós e sincronizados através de mecanismos de consenso.

Os Provedores de Serviço de Associação são estruturados na forma de pastas e podem ser entendidos em níveis: de rede, de canal, de *peer* e de ordenador. No caso

[10]Disponível em: https://hyperledger-fabric.readthedocs.io/en/release-1.4/membership/membership.html

de redes esses provedores estão relacionados a metadados administrativos, já no caso de canais e *peers*, eles têm relação com identidades para gerenciar recursos privados.

5.1.6 Protocolo de Disseminação de Dados Gossip

O Protocolo de Disseminação de Dados *gossip* foi construído para melhorar o desempenho e a escalabilidade de redes de cadeias de blocos no Hyperledger Fabric. Ele realiza essa tarefa através de divisão de cargas de trabalho, seja no momento da validação e execução de transações ocorridos nos *peers*, seja no empacotamento e ordenação de dados feito pelo serviço de ordenamento. A ideia é aumentar o paralelismo da cadeia, evitando que a mesma tarefa seja repetida por todos os nós.

Esses objetivos são alcançados através de um protocolo de disseminação de dados seguro e confiável que garante a consistência e a integridade das informações transmitidas. Dessa maneira, mesmo que certo *peer* não tenha validado o último bloco inserido, por exemplo, este ainda pode ter a versão mais recente da cadeia, tendo recebido o bloco referido através do *Protocolo Gossip*.

A fim de manter a consistência da cadeia, assegurando que os membros da rede tenham seus dados atualizados, o protocolo *gossip* continuamente monitora a rede em busca de novos *peers* e também detecta aqueles que estão *o ine*. Os *peers* recebem as atualizações de certa quantidade de outros *peers*, sendo este número configurável, fazendo com que aqueles que estavam com falhas na rede sejam rapidamente sincronizados assim que identificados os blocos ausentes. Além disso, toda mensagem recebida é assinada pelo remetente o que torna fácil a detecção de nós bizantinos maliciosos.

5.2 Mecanismo de Consenso

No Fabric há três possibilidades de escolha para mecanismos de consenso, são eles: o Solo, o Raft e o Kafka. Estes diferem entre si no modo de obtenção do consenso de qual será a ordem correta das transações a serem inseridas no livro-razão. Os elementos da rede responsáveis por desempenharem tal tarefa são *Orderers*, que também realizam controle de acesso aos livro-razão e ao canal.

O serviço de ordenamento de transações do Hyperledger Fabric consiste em três fases, todas envolvendo os OSNs. A fase 1 consiste na proposta, que ocorre no momento

em que certa aplicação de cliente faz uma proposição de transação para um conjunto de *peers* a fim de que estes invoquem contratos inteligentes para produzir uma proposta de atualização do livro-razão. A fase 2 consiste no ordenamento e agrupamento de transações em blocos, que são feitos pelos *Orderers* após receberem a proposta dos *peers*. A fase 3 é a fase de validação e implementação em que o OSN, nó de serviço de ordenameto, distribui o novo bloco a todos os *peers* da rede, de maneira direta ou indireta (através de outros *peers*), estes fazem a validação de cada transação contida no novo livro-razão e o inserem ou não no seu registro da cadeia.

Apesar de cada registro inserido na cadeia de blocos do Hyperledger Fabric seguir esses três passos, há, conforme mencionado, variações sobretudo na forma com que a ordenação de cada transação em blocos é definida na fase 2. Estas são caracterizadas pelos três mecanismos de consenso Solo, Raft e Kafka.

O Solo é uma implementação de serviço de ordenamento não tolerante a falhas, já que possui, como próprio nome indica, apenas um nó de ordenação. O Solo é indicado para testar aplicações e contratos inteligentes e realizar trabalhos de prova de conceito.

O Raft é o principal mecanismo de consenso usado no Hyperledger Fabric, sendo nativo e indicado pelos desenvolvedores da plataforma para uso em produção. Sua implementação é baseada no modelo líder-seguidor (*leader-follower*) que suporta perda de ambos nós (líderes e seguidores), sendo, portanto, considerado um mecanismo tolerante a falhas de travamento (*Crash Fault Tolerant* - CFT).

Os nós Raft podem assumir três estados, candidato, seguidor e líder. O estado inicial padrão do mecanismo é o de seguidor. Caso nenhum nó líder seja notado até determinado período de tempo, os nós seguidores mudam seu estado para o estado candidato. Nesse momento começa uma votação entre os candidatos e, após a obtenção de maioria simples, um nó é eleito e promovido a líder. Se houver falha no nó líder, outro será eleito.

O consenso no Raft é alcançado de modo relativamente simples, podendo ser decomposto em três estágios: a eleição de um líder, conforme já explicitado, a replicação do *log* e a segurança.

A replicação do *log* é feita pelo líder através da rede de forma a forçar outros *logs* a concordarem com ele. Os logs são registros de transações gravados em cada líder em forma de filas e indexados através de um número inteiro. Uma vez recebida a nova transação o líder a insere no *log* e a transmite para seus seguidores e outros líderes através de *Remote*

Procedure Calls. O líder então decide quando é seguro inserir as transações no livro-razão e então começa a enviar o comando de inserção para todos os seus seguidores até que todos possuam o mesmo estado.

Os mecanismos anteriores não garantem que as mesmas transações sejam inseridas em ordem. Um exemplo em que isso ocorre é quando um seguidor inativo, que está momentaneamente indisponível e, portanto, com *log* desatualizado, é eleito líder, o que sobrescreveria os *logs* do líder anterior. O mecanismo de segurança do Raft garante que todo líder eleito tenha a versão mais recente dos *logs*, e faz isso restringindo os seguidores que podem participar de uma possível eleição.

Outra possibilidade de mecanismo de consenso presente no Hyperledger Fabric é o Kafka. Esse, assim como o Raft, é um modelo de sistema tolerante a falhas (*crash tolerant*) com arquitetura de modelo publicador-assinante (*publish-subscribe*), cujas trocas de mensagens são feitas através do protocolo TCP.

Seus componentes são: publicadores, responsáveis pela geração das mensagens; os *brokers*, intermediários que repassam essas mensagens; tópicos, recursos para os quais os dados são enviados; e os consumidores, responsáveis por consumir o conteúdo transmitido.

O Kafka possui um *cluster*, conjunto de *brokers* em diversos servidores a fim de garantir tolerância a falhas. Assim como o Raft os servidores tem uma arquitetura líder-seguidor, sendo que cada líder possui um *log* de mensagens a distribuídas a seus seguidores. Os consumidores são identificados através de etiquetas fazendo com que toda informação publicada naquele tópico chegue até eles.

O Zookeeper é o serviço responsável pela coordenação do sistema, ele funciona como um espaço hierárquico de registro de dados (Znodes), de modo semelhante a um sistema de arquivos. Os Znodes foram idealizados para gravarem metadados e a forma com que eles estão estruturados é o que os difere de um sistema de arquivos convencional. Cada arquivo tem um tamanho limitado e também pode ser um diretório e vice-versa. Uma vantagem do Zookeeper é o desempenho e baixa latência, que são consequências do salvamento em memória.

Os *brokers* Kafka, nós responsáveis por realizarem a ponte entre consumidores e produtores, têm um grande grau dependência do Zookeeper, já que eles não têm visibilidade de muitos metadados importantes da rede. O Zookeeper é capaz de conferir aos *brokers* propriedades importantes, como: detecção de falha em servidores, particiona-

mento de dados, replicação sincronizada de dados. Esta última é a principal razão para a utilização desse mecanismo de consenso no Hyperledger Fabric.

5.3 *Docker Containers*

Todos os módulos da arquitetura do Hyperledger Fabric funcionam com base na tecnologia de contêineres. De modo simplificado, contêineres podem ser definidos como uma unidade padrão de software. A ideia é que eles contenham todo o necessário para que uma aplicação funcione corretamente: de código fonte, à configuração de ambiente e bibliotecas. Desse forma, pode-se obter unidades de software independentes entre si, facilitando a integração da unidade com outros módulos, e também consegue-se eliminar problemas de implantação relativos à configuração de ambiente, já que o contêiner provê um ambiente padrão independente do sistema operacional utilizado.

Os contêineres funcionam de modo similar ao de máquinas virtuais (VMs, ou *Virtual Machines*), no sentido de que ambos implementam o conceito virtualização. Contudo, a virtualização das VMs se dá com auxílio de hipervisores, criando abstração do hardware, seccionando-o e alocando recursos físicos da máquina para cada uma das secções. Assim, podem ser criadas máquinas completamente independentes entre si mas que fazem uso de uma infraestrutura proveniente de um hardware em comum.

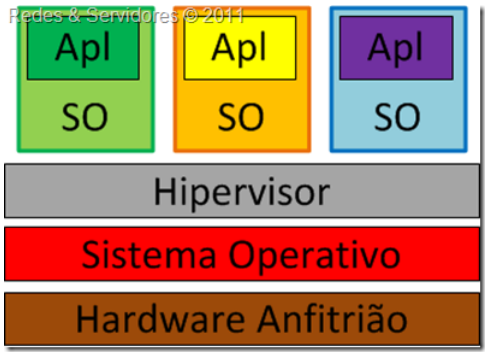

Figura 5.6: Funcionamento Máquina Virtual[12]- Os hipervisores são responsáveis por fazerem a interface dos sistemas operacionais visitantes com o sistema operacional e hardware anfitriões.

[12]Disponível em: https://redes-e-servidores.blogspot.com/2011/11/virtualizacao-total-explicada.html

Apesar de os contêineres também fazerem uso de virtualização, neste caso ela se dá dentro de um mesmo sistema operacional. Os contêineres usam recursos de um só sistema, virtualizando processos de *kernel*. A Figura 5.3 ilustra esse mecanismo.

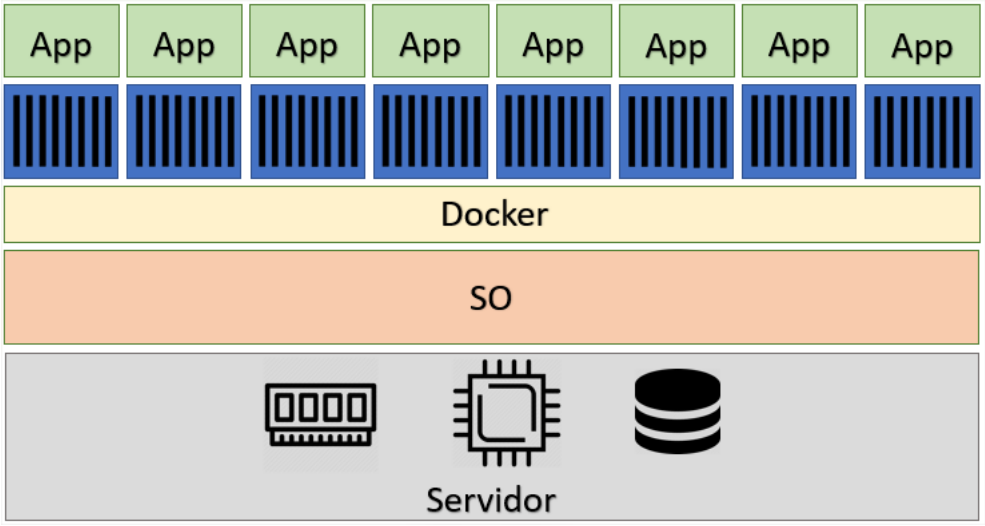

Figura 5.7: Funcionamento contêiner[14]- O Docker fará a interface entre as diversas aplicações virtualizadas, dentro de contêineres, com o Sistema Operacional.

O fato dos contêineres não possuírem um hipervisor, mas uma ferramenta de contêineres, Docker no caso da Figura 5.3, é o que possibilita o compartilhamento de *kernel*. Isso faz com que essa tecnologia seja mais eficiente na alocação de recursos do que de VMs, necessitando de menos recursos para a implementação, já que não é necessária uma instância do sistema operacional por unidade virtualizada.

Portanto, principalmente devido à eficiência, à flexibilidade e à portabilidade foi escolhida uma tecnologia de contêineres para a implementação do Hyperledger Fabric, mais especificamente o software Docker.

5.4 Build Your First Network - BYFN

O *Build Your First Network* é um cenário padrão do Hyperledger Fabric que conta com uma amostra de uma rede já configurada. Essa rede consiste em um canal do qual fazem parte duas organizações que, por sua vez, possuem dois *peers* cada, o mecanismo de consenso utilizado é configurável, contudo o padrão é o Solo. Ainda que esse cenário

[14]Disponível em: https://medium.com/trainingcenter/containers-as-primeiras-pedras-b4b5efa49d88

não tenha relevância em termos de aplicações de uma cadeia de blocos, devido a suas configurações serem simplificadas, ele ilustra bem o funcionamento prático da plataforma.

O sistema operacional mais recomendado para a instalação do Hyperledger Fabric é o Linux e nele devem ser instaladas os pré-requisitos para o funcionamento correto da plataforma e desenvolvimento de *chaincodes* (por meio de SDKs), tais como: o Docker e o Docker-Compose; as linguagens de programação Go, Python e Node JS; e também as amostras, arquivos binários e Imagens Docker disponibilizadas no repositório do Git da plataforma.

As amostras e cenários padrão do Fabric são disponibilizadas na pasta Fabric Samples, cuja estrutura está indicada na Figura 5.8. Essa pasta possui redes já pré-configuradas, como nas partas *basic-network* e *first-network*, amostras de *chaincodes*, arquivos binários que criam instâncias de *peers* e *orderers*, por exemplo, entre outros.

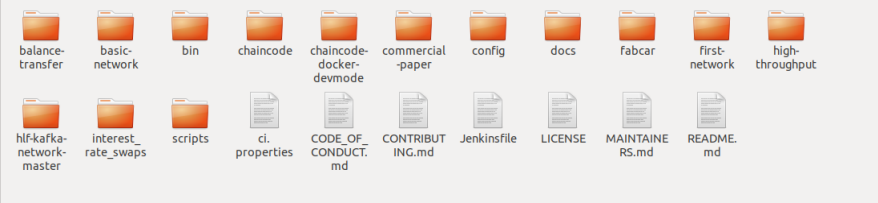

Figura 5.8: Pasta *Fabric Samples*- Na figura podemos ver o diretório raiz do Hyperledger Fabric.

A pasta referente ao Build Your Firt Network é o *first-network* e sua estrutura está ilustrada na Figura 5.9.

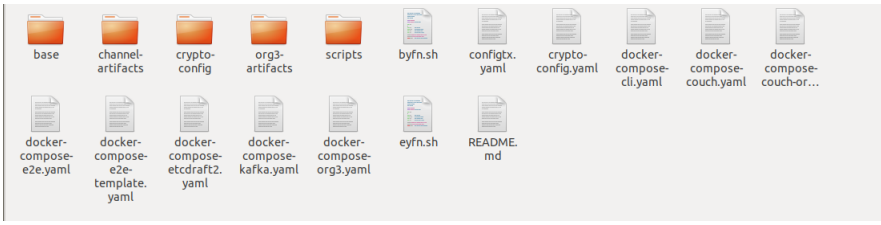

Figura 5.9: Pasta *First Network* - Na figura podemos ver o conteúdo da pasta referente ao cenário *First Network* .

O arquivo responsável por inicializar a rede, possuindo o *shell script* “byfn.sh", que irá acessar os binários contidos na pasta bin e criar os contêineres.

Ao executar o “./byfn.sh generate"no *prompt* de comando, serão gerados: materiais criptográficos (como chaves públicas e privadas) e certificados, usando o arquivo binário

“cryptogen"; o bloco gênesis dos ordenadores e do bloco gênesis do canal; geração dos *peers* âncora de ambas organizações. Esses dados estarão salvos na pasta “crypto-config".

O comando “./byfn.sh up"cria de fato os contêineres Docker e faz com que os *peers* de ambas organizações entrem no canal. Após isso, instala os *chaincodes* em cada um desses *peers* e realiza consultas validando os resultados da execução deles.

Figura 5.10: Contêineres - Na figura podemos ver os contêineres usados no exemplo BYFN.

Na Figura 5.10 ao observar a coluna “names"pode-se notar que os *peers*, o ordenador e o cliente são executados através de um contêiner. Nas primeiras três linhas se encontram os *containers* dos *chaincodes*, e a dissociação entre os *chaincodes* e os *peers*, o fato de serem executados em contêineres separados, facilita o desenvolvimento desses contratos inteligentes, possibilitando até mesmo eles serem escritos em linguagens de programação diferentes.

5.5 Hyperledger Caliper

O Hyperledger Caliper é uma ferramenta de medição de desempenho, *benchmarking*. de cadeias de bloco. O Caliper produz relatórios contendo indicadores de desempenho, como latência máxima e taxa de transferência (*throughput*), de diversas soluções de cadeias de blocos: Hyperledger Besu, Hyperledger Burrow, Ethereum, Hyperledger Fabric, FISCO BCOS, Hyperledger Iroha and Hyperledger Sawtooth.

Os parâmetros avaliados pelo Hyperledger Caliper são:

1. Taxa de envio - Esse parâmetro indica a quantidade de transações submetidas na cadeia de blocos por unidade de tempo.

2. Taxa de Sucesso - Esse parâmetro permite avaliar quantas das transações submetidas foram processadas e registradas com êxito na cadeia de blocos.

3. Taxa de transferência de leitura - Esse parâmetro identifica a taxa de leitura de transações, em transações por segundo. A taxa de transferência de leitura de transações é definida pela razão entre a quantidade de consultas a transações feitas rede de cadeia de blocos e o período de avaliação.

4. Latência de transações - Essa métrica é baseada no tempo decorrido para a inserção de uma transação, medido a partir da submissão até o processamento e o registro no livro-razão. São disponibilizadas as latências mínimas, máximas, média e percentuais.

5. Consumo de recursos - Esse parâmetro mostra os recursos consumidos por cada contêiner Docker, como consumo de CPU, memória, rede.

5.5.1 Arquitetura

A arquitetura do Hyperledger Caliper é dividida em três camadas, a Camada de *Benchmarking*, a Camada Central e de Interface e a Camada de Adaptação. Na Figura 5.11 pode ser visto como está estruturada esta arquitetura.

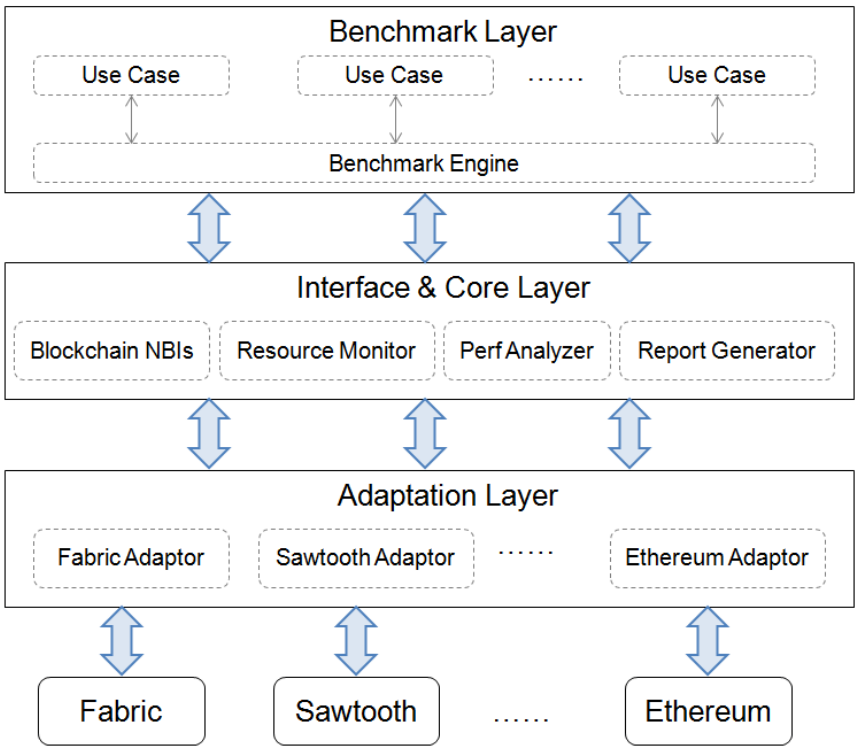

Figura 5.11: Arquitetura Hyperledger Caliper[16]. Na figura pode ser notadas as camadas plataforma, as partes constituintes de cada uma e como é dada a interface entre elas.

Na camada de *Benchmark* podem ser selecionados os casos de uso da plataforma, como cenários de teste. Cada um desses casos estará ligado com a *Benchmark Engine*, que faz a interface com a camada Central. A camada central é responsável por inicializar a cadeia de blocos, instanciar contratos inteligentes e também tem a função de realizar as medições e gerar relatórios. Por sua vez, a camada de adaptação é usada para integrar os sistemas de cadeias de blocos suportados no *framework* do Caliper.

5.6 Aplicação

Através de um cenário de teste, usou-se o Hyperledger Caliper para testar o desempenho de uma cadeia de blocos executada na plataforma Hyperledger Fabric. O cenário consistiu de uma cadeia com três organizações, cada uma constituída por um *peer*, e um nó ordenador, mecanismo de consenso Solo. A Figura 5.12 apresenta o relatório obtido pelo Caliper.

No relatório pode-se observar que a cadeia obteve uma latência média de, aproximadamente, 45 segundos e um *throughput* de 6 transações por segundo além de uma taxa de sucesso de 523 transações, não obtendo nenhuma falha. Nota-se também que os contêineres onde estão os *peers* são os maiores consumidores de memória da rede, cerca de 150MB. Por outro lado, nesse teste o ordenador realizou a maior parte da leitura de discos, 2.4 MB.

É importante ressaltar que o cenário apresentado deve ser usado apenas para testes. A cadeia de blocos testada difere das apresentadas de aplicações em produção, tanto pelo tamanho, contendo apenas três *peers*, quando pelo fato de não ser tolerante à falhas, possui apenas um nó ordenador.

A versão do Caliper usada nesse teste pode ser encontrada no Github da plataforma através do *Hash* “4156c4da7105fd1c2b848573a9943bfc9900". Além disso, a versão do Fabric usada foi v1.1, disponível em “https://github.com/hyperledger/fabric/tree/release-1.1".

[16]Disponível em: www.bcloudpartners.com/home/blockchainhyperledgercaliperintro

Caliper Report

Basic information

DLT: fabric
Benchmark: simple
Description: This is an example benchmark for caliper, to test the backend DLT's performance with simple account opening & querying transactions
Test Rounds: 1
Details

Benchmark results

Summary
open

System Under Test

Version: 1.1.0
Size: 3 Orgs with 1 Peer
Orderer: Solo
Distribution: Single Host
Details

Summary

Test	Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
1	open	523	0	33.8 tps	72.17 s	10.86 s	44.98 s	6.0 tps

open

round 0

performance metrics

Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
open	523	0	33.8 tps	72.17 s	10.86 s	44.98 s	6.0 tps

resource consumption

TYPE	NAME	Memory(max)	Memory(avg)	CPU(max)	CPU(avg)	Traffic In	Traffic Out	Disc Read	Disc Write
Process	node local-client.js(avg)	105.7MB	85.3MB	49.91%	7.13%	-	-	-	-
Docker	dev-peer0.org1.example.co...le-v0	50.4MB	45.9MB	18.88%	1.55%	902.7KB	375.4KB	32.0KB	0B
Docker	dev-peer0.org2.example.co...le-v0	51.4MB	46.1MB	16.26%	1.32%	888.5KB	361.2KB	108.0KB	0B
Docker	dev-peer0.org3.example.co...le-v0	50.0MB	46.0MB	15.39%	1.48%	914.8KB	385.7KB	20.0KB	0B
Docker	dev-peer0.org1.example.co...es-v1	33.2MB	28.2MB	0.02%	0.00%	344B	233B	0B	0B
Docker	dev-peer0.org2.example.co...es-v1	37.6MB	32.6MB	0.02%	0.00%	298B	191B	0B	0B
Docker	dev-peer0.org3.example.co...es-v1	37.1MB	32.2MB	0.03%	0.00%	344B	233B	0B	0B
Docker	peer0.org2.example.com	160.9MB	157.8MB	35.30%	11.13%	6.3MB	5.4MB	1.4MB	3.6MB
Docker	peer0.org3.example.com	162.6MB	159.7MB	39.38%	11.41%	6.3MB	5.4MB	1.1MB	3.6MB
Docker	peer0.org1.example.com	163.0MB	160.1MB	33.24%	11.66%	6.2MB	10.3MB	1.1MB	3.6MB
Docker	couchdb.peer0.org1.example.com	56.3MB	46.8MB	53.50%	25.60%	1.2MB	2.2MB	2.3MB	4.3MB
Docker	ca.org3.example.com	5.9MB	5.4MB	0.00%	0.00%	0B	0B	0B	0B
Docker	orderer.example.com	43.2MB	34.7MB	21.72%	2.43%	2.8MB	7.7MB	2.4MB	2.8MB
Docker	couchdb.peer0.org3.example.com	50.7MB	43.2MB	49.44%	26.03%	1.2MB	2.2MB	1.1MB	4.1MB
Docker	ca.org1.example.com	5.5MB	4.9MB	0.00%	0.00%	0B	0B	0B	0B
Docker	couchdb.peer0.org2.example.com	52.8MB	45.6MB	47.31%	23.51%	1.2MB	2.3MB	1.4MB	4.2MB
Docker	ca.org2.example.com	5.4MB	4.8MB	0.00%	0.00%	0B	0B	0B	0B

Test Environment

benchmark config


```
{
  "name": "simple",
  "description": "This is an example benchmark for caliper, to test the backend DLT's performance with sim
  "clients": {
    "type": "local",
    "number": 5
  },
  "rounds": [
    {
      "label": "open",
      "txDuration": [
        15
      ],
      "rateControl": [
        {
          "type": "linear-rate",
          "opts": {
            "startingTps": 25,
            "finishingTps": 75
          }
```


SUT

```
not provided
```

Figura 5.12: Resultados - Hyperledger Caliper

Capítulo 6

Conclusão e Trabalhos Futuros

A busca por melhoras no desempenho de cadeias de blocos é constante, sendo o principal fator que estimula pesquisas na área. Apesar de significativas evoluções, desde o aparecimento do *Proof of Work*, notório por uso ineficiente de energia, ainda há desafios a serem superados por desenvolvedores de algoritmos especializados para cadeias privadas e também públicas.

Contudo, nota-se que não existe um único mecanismo de consenso que atue de modo otimizado em todos os cenários nos quais uma cadeia de blocos pode estar inserida. Além de limitações práticas, há restrições teóricas, como a imposta pelo Teorema CAP, que obrigam os desenvolvedores de cadeias de blocos a optarem por certas propriedades em detrimentos de outras e, ao fazerem isso, estarão limitando, em termos de desempenho por exemplo, o escopo de atuação da cadeia. Percebe-se, assim, uma tendência de blockchains se tornarem cada vez mais especializadas.

Portanto, plataformas de desenvolvimento que, como o Hyperledger Fabric, oferece desempenho razoável e dê liberdade ao desenvolvedor para moldar uma cadeia de blocos conforme sua necessidade, tendem a ter maior adesão. Uma sugestão para trabalhos futuros é criar um *framework* para realizar *benchmarking* em cadeias de blocos e assim conseguir avaliar, com base em dados quantitativos, o desempenho de plataformas de blockchains existentes.

Bibliografia

[1] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[2] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[4] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[5] Fabíola Greve Greve, Leobino Sampaio Sampaio, Jauberth Abijaude Abijaude, Antonio Coutinho Coutinho, Ítalo Valcy Valcy, and Sílvio Queiroz Queiroz. Blockchain e a revolução do consenso sob demanda. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)-Minicursos*, 2018.

[6] Barbara Liskov. From viewstamped replication to byzantine fault tolerance. In *Replication*, pages 121–149. Springer, 2010.

[7] M. T. Oliveira, G. R. Carrara, N. C. Fernandes, C. V. N. Albuquerque, R. C. Carrano, D. S. V. Medeiros, and D. M. F. Mattos. Towards a performance evaluation of private blockchain frameworks using a realistic workload. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 180–187, Feb 2019.

[8] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, (2):125–143, 1977.

[9] Wojciech Galuba and Sarunas Girdzijauskas. Peer-to-peer system. *Encyclopedia of Database Systems*, pages 2081–2082, 2009.

[10] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[11] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[12] Dmitry Efanov and Pavel Roschin. The all-pervasiveness of the blockchain technology. *Procedia Computer Science*, 123:116–121, 2018.

[13] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3:37, 2014.

[14] Nick Szabo. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials*, 6, 1997.

[15] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[16] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Verissimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2):141–161, 2011.

[17] Diogo Menezes Ferrazani Mattos, Otto Carlos Muniz Bandeira Duarte, and Guy Pujolle. A lightweight protocol for consistent policy update on software-defined networking with multiple controllers. *Journal of Network and Computer Applications*, 122:77 – 87, 2018.

[18] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[19] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[20] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[21] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[22] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for Internet-scale systems. In *USENIX annual technical conference*, volume 8. Boston, MA, USA, 2010.

[23] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1:11, 2014.

[24] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[25] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[26] Brad Chase and Ethan MacBrough. Analysis of the xrp ledger consensus protocol, 2018.

[27] Arati Baliga. Understanding blockchain consensus models. In *Persistent*. 2017.

[28] Alysson Bessani, João Sousa, and Eduardo E. P. Alchieri. State machine replication for the masses with bft-smart. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, pages 355–362, Washington, DC, USA, 2014. IEEE Computer Society.

[29] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In Jan Camenisch and Doğan Kesdoğan, editors, *Open Problems in Network Security*, pages 112–125, Cham, 2016. Springer International Publishing.

[30] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[31] M. M. Jalalzai and C. Busch. Window based bft blockchain consensus. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social*

Computing (CPSCom) and IEEE Smart Data (SmartData), pages 971–979, July 2018.

[32] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[33] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–580, June 2019.

[34] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.