UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ - UNIOESTE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CCET

Programa de Pós-Graduação em Ciência da Computação - PPGComp

Avaliação do vCubeChain em Sistemas Sujeitos a Falha Crash

Dissertação (Mestrado)

Gabriela Stein

Cascavel–PR

2024

UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ - UNIOESTE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CCET

Programa de Pós-Graduação em Ciência da Computação - PPGComp

Gabriela Stein

Avaliação do vCubeChain em Sistemas Sujeitos a Falha Crash

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação, do Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual do Oeste do Paraná - Campus de Cascavel.

Orientador(a): Dr. Luiz Antonio Rodrigues

Cascavel–PR

2024

Ficha de identificação da obra elaborada através do Formulário de Geração Automática do Sistema de Bibliotecas da Unioeste.

Stein, Gabriela
Avaliação do vCubeChain em Sistemas Sujeitos a Falha Crash / Gabriela Stein; orientador Dr. Luiz Antonio Rodrigues. -- Cascavel, 2024.
68 p.

Dissertação (Mestrado Acadêmico Campus de Cascavel) -- Universidade Estadual do Oeste do Paraná, Centro de Ciências Exatas e Tecnológicas, Programa de Pós-Graduação em Ciências da Computação, 2024.

1. vcube. 2. blockchain. 3. escalabilidade. I. Rodrigues, Dr. Luiz Antonio, orient. II. Título.

Gabriela Stein

Avaliação do vCubeChain em Sistemas Sujeitos a Falha *Crash*

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Trabalho aprovado. Cascavel-PR, 26 de setembro de 2024:

gov.br
Documento assinado digitalmente
LUIZ ANTONIO RODRIGUES
Data: 17/12/2024 07:57:29-0300
Verifique em https://validar.iti.gov.br

Dr(a). Luiz Antonio Rodrigues
Dr. Luiz Antonio Rodrigues)

Dr. Edson Tavares de Camargo
Universidade Tecnológica Federal do Paraná

Dr. Allan Edgard Silva Freitas
Instituto Federal da Bahia

Cascavel-PR
2024

Este trabalho é dedicado à minha mãe,
que sempre fez de tudo para que eu chegasse até aqui, quando ela não pôde.

Agradecimentos

Neste momento tão crucial e conclusivo de uma grande jornada, os agradecimentos principais são para os meus pais Osmar e Cleide, que nunca me fizeram titubear na minha decisão de priorizar a educação e minha formação, sem seu apoio incondicional e seus ensinamentos desde criança, jamais teria chegado até aqui. Agradeço também a toda a minha família, meu irmão Guilherme e meus diversos primos, tios e avós que nunca entenderam minha pesquisa mas faziam questão de acreditar na minha capacidade mais do que eu mesma. Não posso deixar de agradecer as minhas amigas, que fizeram questão de sempre estarem presentes com uma palavra de conforto, de incentivo e boas risadas. Minha gratidão especial ao meu companheiro Lucas, por sua eterna paciência e convicção de que todo o meu esforço não seria em vão. Obrigada a todos que sempre desejaram o melhor para mim e pelo cuidado que tiveram comigo para que eu pudesse superar cada obstáculo em meu caminho e chegar aqui. Um obrigado especial ao meu orientador Prof. Dr. Luiz Antonio Rodrigues, pela confiança que depositou em mim ao me aceitar como sua orientanda no programa, espero ter sido digna de seu esforço e dedicação como profissional. Agradeço também aos professores Dr. Edson Tavares de Camargo e Dr. Allan Edgard Silva Freitas, membros da banca de Qualificação e Defesa de Mestrado, pelos conselhos, sugestões e interesse em contribuir para o desenvolvimento deste projeto. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Resumo

STEIN, Gabriela. **Avaliação do vCubeChain em Sistemas Sujeitos a Falha *Crash***. Orientador(a): Dr(a). Luiz Antonio Rodrigues. 2024. 66f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2024.

A blockchain é um sistema de registros de transações, seguro e em constante crescimento, no qual cada usuário dos dados mantém uma cópia dos registros, que só pode ser atualizada se todas as partes envolvidas em uma transação concordarem em atualizar. ou seja, é um livro-razão distribuído (*peer-to-peer*) que é criptograficamente seguro, imutável e atualizável apenas por meio de acordo entre os pares (consenso). Múltiplos mecanismos alternativos de consenso já foram propostos, um trabalho recente na área é a vCubeChain, uma solução de blockchain permissionada escalável. A vCubeChain utiliza do detector de falhas fornecido pelo vCube, um algoritmo de diagnóstico distribuído que conecta virtualmente nós de uma rede e que forma um hipercubo quando todos os processos estão corretos, para manter as propriedades logarítmicas mesmo com processos falhos. O artigo vCubeChain apresentou um conjunto de experimentos de modo a apresentar o desempenho da solução, comparando-o com blockchains bem conhecidas como o Ethereum e Bitcoin. Entretanto, a ferramenta utilizada não permite a simulação de cenários onde os nós da rede falham. Portanto, de modo a analisar o desempenho da vCubeChain com falhas, o presente trabalho propõe complementos ao simulador BlockSim, estendendo-o para simular falhas de processos. Ademais, implementa-se o HyperLedger Fabric, estratégia de blockchain permissionada, para comparação e análise dos cenários entre blockchains públicas e permissionadas. Os resultados demonstraram que o número médio de mensagens trocadas na vCubeChain é inferior aos demais, mesmo em cenários onde o nó líder do consenso falha e uma rodada de eleição deve ocorrer, gerando um grande número de mensagens sendo trocadas entre os nós para propagação dessa informação.

Palavras-chave: blockchain; vCube; escalabilidade; simuladores; BlockSim.

Abstract

STEIN, Gabriela. **vCubeChain Evaluation in Crash-prone Systems** . Orientador(a): Dr(a). Luiz Antonio Rodrigues. 2024. 66f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2024.

Blockchain is a secure and constantly growing system of transaction records in which each data user maintains a copy of the records that can only be updated if all parties involved in a transaction agree to the update. That is, it is a distributed peer-to-peer ledger that is cryptographically secure, immutable, and updatable only by consensus or peer agreement. Several consensus alternatives have already been proposed. The most recent work in this area is vCubeChain, a scalable permissioned blockchain solution. Based on vCube's virtual topology - a distributed diagnostic algorithm that virtually connects the nodes of a network - vCubeChain uses the fault detector provided by vCube, which forms a hypercube when all processes are correct, to maintain logarithmic properties even when processes fail. In the article about vCubeChain, a series of experiments were presented to showcase the performance of the solution and compare it to well-known blockchains such as Ethereum and Bitcoin. However, the tool used does not allow the simulation of scenarios in which the network nodes fail. Therefore, in order to analyze the performance of the vCubeChain in the presence of failures, this paper proposes additions to the BlockSim simulator that extend it to simulate process failures. Furthermore, the HyperLedger Fabric, a permissioned blockchain strategy, is implemented to compare and analyze scenarios between public and permissioned blockchains. The results show that vCubeChain's number of exchanged messages is much smaller than its counterparts, even when the leader node fails and a new leader election happens - leading to a larger number of messages being exchanged between the nodes to propagate this information.

Keywords: blockchain; vCube; scalability; simulator; BlockSim.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

P2P	*Peer-to-Peer*
PoW	*Proof-of-Word*
PoS	*Proof-of-Stake*
ETH	Ethereum (criptomoeda)
BFT	*Byzantine Fault Tolerance*
PBFT	*Pratical Byzantine Fault Tolerance*
CPU	*Central Processing Unit*
GST	*Global Stabilization Time*
FLP	Fischer, Lynch, Patterson
RME	Replicação Máquina de Estados
DESE	*Discrete Event Simulation Engine*
EVM	*Ethereum Virtual Machine*
ID	Identidade
RTT	*Round Trip Time*
RNP	Rede Nacional de Pesquisa
TCP	*Transmission Control Protocol*

Sumário

1 Introdução

Os sistemas distribuídos podem ser descritos como um conjunto de processos que trabalham juntos e se comunicam para realizar uma determinada tarefa. Considerando a troca de mensagens necessária para a cooperação entre processos, qualquer falha em qualquer dos componentes do sistema pode trazer consequências ao resultado da execução. Quando um subconjunto dos processos falha ou se desconecta, o desafio é sincronizar as atividades dos processos que ainda estão em operação de forma consistente (GUERRAOUI; RODRIGUES, 2006). Ou seja, o objetivo é fornecer aos processos que continuam funcionando após uma falha, informações atualizadas e corretas o suficiente de modo que estes possam continuar a operando corretamente e resolvendo suas tarefas.

Um requisito fundamental para a coordenação entre os processos, é que conheçam os estados uns dos outros, para que possam tomar as providências necessárias em caso de possível falha de um ou mais processos. Em alguns tipos de sistemas distribuídos, esta pode ser uma tarefa difícil ou mesmo impossível. Em um sistema assíncrono, no qual os processos ou o sistema de comunicação (envio e recebimento de mensagens) podem apresentar comportamento arbitrariamente prolongado, se torna impossível distinguir um processo muito lento de um processo falho, de forma que o processo lento possa ser confundido com um falho (FISCHER; LYNCH; PATERSON, 1985). Não há garantia de uma cooperação consistente neste cenário.

Chandra e Toueg apresentaram na primeira versão de seu artigo *"Unreliable Failure Detectors for Reliable Distributed Systems"*, detectores de falhas não confiáveis como uma ferramenta para auxiliar o problema do consenso em sistemas assíncronos com falhas. Os autores propuseram a adição de um mecanismo externo de detecção de falhas distribuído, onde cada processo tem acesso a um módulo local de detecção de falhas. Cada módulo local monitora um subgrupo dos processos no sistema e mantém uma lista daqueles que suspeita atualmente de estar falho. O modelo pode cometer erros e suspeitar falsamente de um processo qualquer, mesmo que este esteja funcionando. Se o detector verifica posteriormente do seu erro, pode remover o processo correto de sua lista de suspeitos. Assim, cada módulo pode adicionar e remover processos de sua lista de suspeitos. A qualquer momento, os módulos de detecção de falhas de dois processos diferentes, podem ter uma lista diferente de suspeitos.

Em muitas situações, não apenas um, mas vários processos precisam cooperar e sincronizar suas tarefas. A cooperação entre processos pode algumas vezes ser modelada como um problema de acordo distribuído. Por exemplo, os processos podem precisar concordar se um determinado evento ocorreu (ou não), concordar com uma sequência

comum de ações a serem executadas (a partir de várias alternativas iniciais) ou concordar com a ordem por qual um conjunto de entradas precisa ser processado (CHANDRA; TOUEG, 1996). A existência da necessidade de coordenação a ação e tomada de decisão entre múltiplos processos torna a computação distribuída um pouco mais complexa (CACHIN; GUERRAOUI; RODRIGUES, 2011). Esse tipo de sistema distribuído é chamado de *peer-to-peer*, ou simplesmente P2P. Estes são sistemas dinâmicos caracterizados pela auto-organização de seus componentes, chamados *peers*, com o objetivo do compartilhamento de seus recursos, sejam estes: conteúdo, tempo de processamento ou qualquer outro tipo de serviço. Neste contexto, surge a blockchain, uma nova tecnologia P2P que está se tornando cada vez mais importante no mundo atual, não somente na computação, mas também por sua aplicação em setores como finanças, saúde, arte, entre outros (MAKRIDAKIS; CHRISTODOULOU, 2019).

A blockchain oferece suporte a implementação de uma cadeia de blocos que mantém um registro (*ledger*) permanente, sequenciado, inviolável e em crescimento contínuo, vinculado e protegido por criptografia, no qual todas as transações realizadas naquela rede ficam armazenadas. O consenso é um problema fundamental em computação distribuída confiável, permitindo a decisão em comum dos participantes distribuídos, que também é crucial para a blockchain, pois possibilita a obtenção do acordo sobre o próximo bloco de transação a ser agregado à blockchain (GREVE et al., 2018). A blockchain é resultado da combinação de técnicas robustas provenientes da computação distribuída, como tolerância a falhas bizantinas (ocorrem quando um ou mais componentes falham e não há informações precisas sobre a falha de um componente ou se as informações do sistema estão corretas), sistemas P2P, criptografia e teoria dos jogos.

Pode-se classificar a blockchain de duas formas: permissionada (federada/privada) e não-permissionada (públicas). Quando o conjunto de nós da rede P2P é desconhecido, permitindo entrada e saída de nós de maneira dinâmica e sem autenticação dos participantes, encontra-se uma blockchain pública na qual não existe confiança entre os nós da rede. Os principais representantes dessa categoria são as blockchains baseadas em criptomoedas, tais como a Bitcoin (NAKAMOTO, 2008) e o Ethereum (BUTERIN et al., 2014). Uma das principais estratégias de consenso utilizada em blockchains públicas é *Proof-of-Work (PoW)* que valida novos blocos com base no poder computacional dos usuários. É então gerada uma recompensa como incentivo para o nó vencedor, por exemplo no caso do Bitcoin, o usuário recebe moedas (bitcoins) em troca do trabalho.

Em setembro de 2022, a rede Ethereum passou-se a denominar Ethereum 2.0 e migrou sua estratégia de consenso para *Proof-of-Stake (PoS)*, onde os usuários explicitamente apostam capital na forma de criptomoedas (ETH) em um contrato inteligente na rede Ethereum. Essa estratégia de consenso tem o benefício de consumir menos energia que *Proof-of-Work (PoW)*. O ETH apostado atua como garantia e pode ser destruído se o

validador se comportar de maneira desonesta ou preguiçosa. O validador é então responsável por verificar se os novos blocos propagados pela rede são válidos e, ocasionalmente, criar e propagar novos blocos.

As blockchains permissionadas têm uma rede controlada, onde seus participantes estão autenticados e bem identificados, e podem confiar uns nos outros. Blockchains permissionadas abordam muitos dos problemas que foram estudados no campo da computação distribuída ao longo de décadas, principalmente para o desenvolvimento de sistemas tolerantes a falhas bizantinos (BFT - *Byzantine Fault Tolerance*). Dessa forma, soluções clássicas de consenso como o *Practical Byzantine Fault Tolerance* - PBFT (CASTRO; LISKOV, 2002) podem ser adaptados para blockchains permissionadas.

As técnicas desenvolvidas para chegar a um consenso, replicar o estado, transmitir transações, entre outras, podem trazer muitos benefícios para as blockchain permissonadas. Principalmente, em ambientes onde a conectividade da rede é incerta, os nós podem travar ou serem subvertidos por um adversário e as interações entre os nós são inerentemente assíncronas. O amplo interesse nas tecnologias blockchain desencadeou novas pesquisas sobre protocolos práticos de consenso distribuído. Há também um número crescente de startups, programadores e grupos da indústria desenvolvendo protocolos de blockchain com base em suas próprias ideias, sem depender de conhecimento estabelecido (CACHIN; VUKOLIĆ, 2017).

No trabalho de Freitas, Rodrigues e Duarte Jr. (2023), os autores apresentaram a vCubeChain, uma blockchain permissionada tolerante a falhas crash com o objetivo de ser escalável, baseada na topologia distribuída hierárquica denominada vCube (DUARTE JR.; BONA; RUOSO, 2014). Esta topologia é criada e mantida com base nas informações de diagnóstico, que são obtidas por meio de um algoritmo de detecção de falhas distribuído para nós de rede virtualmente interconectados. Cada processo que executa o vCube é capaz de testar outros processos no sistema para verificar se estão corretos ou falhos. A topologia se reorganiza e mantém suas propriedades logarítmicas (distância máxima virtual entre os nós da rede, número de vizinhos e latência de diagnóstico e quantidade de mensagens) mesmo se um número arbitrário de nós estiver com falhas. Utilizando o vCube se provê um serviço de detecção de falhas e no qual os processos são organizados em *clusters* progressivamente maiores, formando um hipercubo completo quando não há processos falhos e reduzindo significamente o número de mensagens trocadas. O vCube tem sido utilizado tradicionamente em sistemas síncronos, nos quais falsas suspeitas não occorem. Neste trabalho, aproveita-se da versão assíncrona do vCube proposta em Stein, Rodrigues e Jr. (2023), possibilitando novas investigações de soluções em ambientes mais próximas da realidade, incluindo a vCubeChain.

1.1 Objetivos

O objetivo geral deste trabalho é implementar e avaliar o vCubeChain para cenários com falhas considerando a versão do vCube proposta para sistemas não síncronos. A implementação prévia da vCubeChain realizada por Freitas, Rodrigues e Duarte Jr. (2023) utilizou o simulador Blocksim – um simulador de eventos discretos flexível o suficiente para avaliar diferentes implementações de blockchain, portanto o presente trabalho se aproveita das funcionalidades já desenvolvidas anteriomente com o enfoque em adaptar o simulador para execução de testes onde nós da rede falham.

Para realizar esta tarefa, são definidos os seguintes objetivos específicos:

- Adicionar um mecanismo de falhas no simulador Blocksim, utilizado anteriormente para avaliar a vCubeChain;
- Implementar o vCube para sistemas assíncronos no Blocksim;
- Adaptar as implementações do Bitcoin e Ethereum disponíveis no Blocksim para suportar a simulação de falhas;
- Implementar a blockchain permissionada Hyperledger Fabric para realizar a comparação de funcionalidades com o vCubeChain;

1.2 Metodologia

O trabalho se baseia em uma avaliação do desempenho da vCubeChain em cenários com falhas. Para atingir seu objetivo, foi necessária a inclusão de funcionalidades no simulador escolhido, o BlockSim. A escolha pelo BlockSim surgiu devido a sua característica de ser um simulador flexível o suficiente para avaliar diferentes implementações de blockchain. Devido a arquitetura do simulador, é possível fazer as modificações exigidas em alguns dos seus componentes e adicionar o suporte a simulação de falhas de nós.

No artigo de proposta da vCubeChain, a solução é comparada com as blockchains públicas Bitcoin e Ethereum. O Fabric Hyperledger é uma solução permissionada, assim como a vCubeChain, possível de ser implementada no simulador BlockSim. Os testes da solução vCubeChain serão realizados em cenários adicionais nos quais os nós da rede falham, e seus resultados comparados com a blockchain permissionada Fabric Hyperledger e as blockchains Bitcoin e Ethereum, que foram adaptadas para a execução no ambiente de falhas a ser desenvolvido.

No trabalho original de Freitas, Rodrigues e Duarte Jr. (2023), a estratégia de implementação da vCubeChain se baseia na proposta do algoritmo vCube no modelo temporal síncrono, onde os limites temporais de envio e recebimento de mensagens são

bem definidos. Em um sistema não síncrono não existem limites conhecidos para o tempo de execução de processos e do atraso de comunicação. Conforme os resultados presentes em Stein, Rodrigues e Jr. (2023), o presente trabalho utilizou desta versão do vCube parcialmente síncrono como base para a construção da vCubeChain. Considerando que os testes do vCube anterior foram realizados com o simulador Neko (Urban; Defago; Schiper, 2001), uma implementação no Blocksim também foi realizada. Neste trabalho, entenda ambiente assíncrono como parcialmente síncrono com premissa GST.

1.3 Organização do Texto

Esse trabalho está organizado da seguinte forma: o Capítulo 2 serão apresentados os conceitos utilizados no trabalho, além de discutir trabalhos relacionados e alguns dos métodos utilizados na literatura para solucionar o problema da escalabilidade na blockchain. Após, no Capítulo 3 será descrita a solução proposta no trabalho. Os resultados são discutidos no Capítulo 4. O Capítulo 5 conclui o trabalho, apresentando as conclusões retiradas dos experimentos realizados e trabalhos futuros.

2 Referencial Teórico

Nesse capítulo serão apresentados alguns dos fundamentos teóricos básicos de sistemas distribuídos, necessários para compreensão da blockchain, cujas raízes estão inseridas na computação distribuída. Além de apresentar conceitos fundamentais da blockchain e explicar a fundo o modelo de diagnóstico de falhas distribuído VCube, também será apresentada a versão inicial da vCubeChain (FREITAS; RODRIGUES; Duarte Jr., 2023).

2.1 Sistemas Tolerantes a Falhas

Um sistema distribuído consiste de n processos que se comunicam para atingir determinado objetivo. A comunicação entre processos pode ser realizada pela troca de mensagens, onde os processos se comunicam por meio do envio/recebimento de mensagens através de uma rede, ou por meio de memória compartilhada, onde os processos escrevem e leem a memória compartilhada entre eles (KSHEMKALYANI; SINGHAL, 2011).

A coordenação e funcionamento dos processos distribuídos em um sistema está interligada ao *tempo* de execução de suas tarefas e de transmissão das mensagens. O tempo é marcado pelo relógio local do processador (CPU) sobre o qual o processo está sendo executado. O limite (máximo) de tempo para transmitir uma mensagem e o limite de tempo para executar uma tarefa determinam a classificação dos modelos de sistema distribuído.

Um sistema é dito síncrono se existem limites de tempo conhecidos para a transmissão de mensagens e a execução de uma tarefa. Os limite temporais devem ser respeitados *sempre*. Em um sistema distribuído assíncrono não há limites temporais de transmissão e execução, o sistema é definido livre de qualquer conceito de tempo (CACHIN; GUERRAOUI; RODRIGUES, 2011).

Não é possível que sistemas reais se ajustem aos modelos síncronos, nem assíncronos. Estes são modelos teóricos que tratam de extremos. Dessa forma, foram propostos modelos *parcialmente síncronos*, que tentam aproximar-se do funcionamento real. Um destes modelos parcialmente síncronos é o modelo GST (*Global Stabilization Time*). Neste modelo, assume-se que o sistema inicia assíncrono, não respeitando qualquer limite temporal conhecido. A partir de um instante de tempo, denominado GST, o sistema torna-se síncrono até o final da execução. O instante exato em que o sistema faz a transição não é conhecido inicialmente, mas existe. Na prática, o modelo GST assegura que o sistema distribuído vai ter as condições temporais para realizar o que é preciso (DWORK; LYNCH; STOCKMEYER,

1988).

Em sistemas que necessitam da cooperação entre os processos para gerar a saída correta solicitada, é fundamental garantir o funcionamento correto do sistema, independente das falhas que ainda irão ocorrer. Um sistema é dito tolerante a falhas se continua funcionando e produzindo as saídas corretas para as entradas correspondentes, mesmo que alguns dos seus componentes estejam falhos. Com o propósito de impedir que uma falha possa se propagar pelo sistema e possa causar o colapso do sistema como um todo.

Os sistemas computacionais sofrem falhas dos mais diversos tipos. Quando se constrói um sistema tolerante a falhas é essencial definir qual modelo de falha é considerado, que tipo de falha o sistema tolera (BONDAVALLI; SIMONCINI, 1990). Alguns dos modelos de falhas de sistemas distribuídos são apresentados a seguir.

Falha por parada (*crash*): O processo do sistema simplesmente para de funcionar, além de não reagir a qualquer estímulo, não produz qualquer saída para nenhuma entrada e perde seu estado interno completamente. Assim, se o processo se recupera mais tarde, é um processo totalmente novo, sem informação sobre execuções anteriores. Existem duas variações sobre o modelo de falhas *crash*:

- *Crash-recovery:* o processo mantém informações chave em memória não volátil, assim ao se recuperar pós falha, o processo volta a executar "normalmente". Em geral, o estado do processo após o retorno depende de algum mecanismo adicional de *checkpointing*.
- *Fail-stop:* as falhas dos processos são *crash* e todos os processos sem falha têm uma lista de quais estão falhos. Pode-se dizer que é um modelo que inclui o sistema de monitoramento de falhas.

Falha por omissão: um processo que falha por omissão não produz todas as mensagens que deveria, de forma que ao receber determinado estímulo um subconjunto das mensagens de saída é omitido.

Falhas bizantinas: o comportamento do processo falho é arbitrário, pode produzir qualquer saída para qualquer entrada. As falhas bizantinas correspondem a qualquer comportamento fora da especificação e incluem as falhas por omissão, e crash. O modelo de falhas por omissão engloba as falhas crash, no caso em que o conjunto de mensagens otimidas correspondem a todas as mensagens que o processo deveria omitir.

Sistemas tolerantes a falhas bizantinas devem saber lidar com o comportamento de um processo falho no qual ele envia informações conflitantes para diferentes partes do sistema, ou seja, diferente de um processo correto, ele pode exibir qualquer comportamento,

pode parar, omitir envios e entregas de mensagens, ou desviar arbitrariamente de sua especificação.

2.2 Consenso

Os mecanismos de consenso são essenciais para as aplicações distribuídas, sendo os responsáveis que os processos decidam sobre a ordem de suas ações, a veracidade de informações e o funcionamento da aplicação. O consenso é parte de qualquer sistema distribuído que incorpora atividades coordenadas. Além disso, o consenso está intimamente relacionado à tolerância a falhas. Qualquer sistema distribuído assíncrono para o qual o compartilhamento de dados é importante deve ser capaz de realizar o consenso para tolerar certos tipos de falhas (TUREK; SHASHA, 1992).

De maneira informal, considerando um sistema distribuído que consiste de um conjunto de processos, a execução do consenso se inicia a partir da "proposta" de valores iniciais pelos processos (cada processo pode propor um valor a partir de um conjunto pré-definido). Ao final do consenso, todos os processos "decidem" pelo mesmo valor, que é um daqueles que foram propostos inicialmente.

Formalmente, o consenso é definido por três propriedades:

- Acordo: todos os processos corretos decidem o mesmo valor;
- Validação: se um processo correto decide um determinado valor, então este valor foi proposto por um processo do sistema;
- Terminação: todo processo correto eventualmente decide um valor.

O consenso pode ser fácil ou difícil de alcançar, dependendo do modelo de sistema (síncrono ou assíncrono) e das suposições de falha. No famoso artigo de 1985, Fischer, Lynch e Paterson mostraram a impossibilidade de consenso determinístico entre dois ou mais processos em um sistema distribuído assíncrono sujeito a falhas (FISCHER; LYNCH; PATERSON, 1985). Os autores argumentam que em um sistema assíncrono é impossível distinguir um processo lento de um processo falho, isto é, o processo lento pode ser confundido como falho e acabar tomando uma decisão diferente dos demais, o que fere a propriedade de acordo e prova a impossibilidade do consenso nesse contexto. Este resultado decisivo para a área de sistemas distribuídos ficou conhecido como Impossibilidade FLP (sigla proveniente do sobrenome dos autores). Desde então, o problema do consenso foi examinado sob muitas suposições diferentes de sincronia e falha.

2.3 Detectores de Falha Não-Confiáveis

Os detectores de falhas não-confiáveis surgem para contornar a impossibilidade FLP (FISCHER; LYNCH; PATERSON, 1985), teoria que provou a impossibilidade de distringuir um processo falho de um processo lento em um sistema assíncrono.

Em 1996, Chandra e Toueg Chandra e Toueg (1996) consideraram que se os processos executando o consenso tivessem acesso a informação sobre a falha de processos, o consenso se tornaria possível, utilizando um modelo de um mecanismo externo de detecção de falhas que pode cometer erros, ou seja, são não-confiáveis. O detector pode realizar dois tipos de erros: um processo que na realidade está falho é informado como correto pelo detector ou um processo que na realidade está correto é informado como falho pelo detector. É importante observar que os erros cometidos por um detector de falhas não-confiável não devem impedir que qualquer processo correto se comporte de acordo com a especificação, mesmo que esse processo seja (erroneamente) suspeito de ter falhado por todos os outros processos.

Os detectores de falhas são implementados como um oráculo ao qual os processos tem acesso localmente, processos "perguntam" a ele sobre o estado dos outros processos. A saída de um detector de falhas é a lista de processos suspeitos de terem falhado. Portanto o detector informa a "suspeita de falha" de um processo, não a "falha" de um processo.

Em Chandra e Toueg (1996), uma classe de detectores de falhas é especificada por meio das propriedades de completude e precisão. A completude apresenta que o detector de falhas suspeita de processos que efetivamente falharam. Enquanto a propriedade de precisão garante que o detector não suspeita de nenhum processo correto. A propriedade de precisão é muito difícil de ser atingida, senão até impossível. Isso se dá devido ao fato de que é possível que o processo esteja correto mas extremamente lento, causando com que isso seja lido de forma incorreta como uma falha. A propriedade de completude é mais fácil de se conseguir pois o processo é monitorado o tempo todo, e uma falha por parada faz com que o processo deixe de se comunicar.

As classes iniciais de detectores de falhas podem variar entre uma combinação de duas propriedades de completude (fraca e forte) com quatro de precisão (forte, fraca, forte eventual, fraca eventual). Como é possível realizar a transformação de completude fraca para forte, essas classes podem ser reduzidas para quatro principais classes. A completude forte é a que diz que inevitavelmente, todo processo que falha é suspeitado por todo processo correto, enquanto na completude fraca, inevitavelmente todo processo que falha é suspeitado por pelo menos um processo correto. A completude fraca pode ser transformada em forte de uma maneira trivial: o processo que suspeita da falha informa os demais processos corretos, de modo que não é necessário que todos os processos corretos identifiquem a possível falha e sim sejam comunicados dessa suspeita.

As quatro classes de detectores de falha que são definidas por meio da combinação das propriedades, são apresentadas na Tabela 1.

Tabela 1 – Classes de detectores com completude e acurácia fracas (*eventual*).

	Acurácia forte garantida	**Acurácia fraca garantida**
Completude forte	Inevitavelmente perfeito $\Diamond P$	Inevitavelmente forte $\Diamond S$
Completude fraca	Inevitavelmente quase-perfeito $\Diamond Q$	Inevitavelmente fraco $\Diamond W$

Basta que o detector de falhas seja $\Diamond W$ para garantir o consenso. Ele necessita satisfazer as duas propriedades:

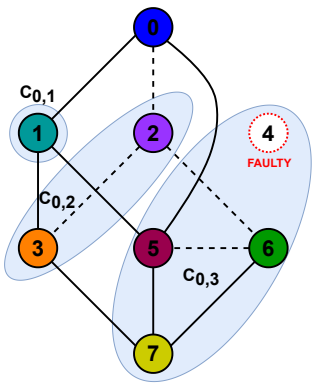
- **Completude:** há um tempo após o qual todo processo que falha é permanentemente suspeito por algum processo correto.
- **Acurácia:** há um tempo após o qual algum processo correto nunca é suspeito por nenhum processo correto.

2.4 VCube

O VCube (DUARTE JR.; BONA; RUOSO, 2014) é um algoritmo de diagnóstico distribuído que conecta virtualmente os nós de uma rede. A topologia virtual lógica criada é hierárquica e corresponde a um hipercubo quando todos os processos estão corretos.

Os processos de um vCube com dimensão $d > 0$ têm identificadores que consistem em d bits. Em um cenário sem falhas, dois processos estão virtualmente conectados se seus endereços binários diferem por um único bit. No exemplo da Figura 1, o processo 1 (001) se conecta com os processos 0 (000), 3 (011) e 5 (101).

Figura 1 – Clusters de um vCube com $2^3 = 8$ processos; p_4 está falho.



Fonte: Da Autora

As arestas virtuais de um vCube correspondem aos testes que os processos corretos executam entre si. O vCube permite que os processos obtenham informações de diagnóstico de cada processo testado corretamente, organizando os processos em clusters $s = 1, .., \log_2 n$ progressivamente maiores, com 2^{s-1} processos. A função $c_{i,s}$ (Equação 2.1) retorna a lista ordenada de processos de cada cluster, onde $\oplus$ é o operador *bitwise* exclusivo (XOR).

$$c_{i,s} = \{i \oplus 2^{s-1},\ c_{i\oplus 2^{s-1},\ 1}, \cdots,\ c_{i\oplus 2^{s-1},s-1}\} \tag{2.1}$$

Clusters são conjuntos de processos e o número de processos em um cluster é uma potência de 2. Supondo inicialmente que o número de processos N no sistema seja uma potência de 2, no primeiro intervalo de testes, o cluster a ser testado é aquele com 1 (2^0) processo. No próximo intervalo, o cluster com 2 (2^1) processos, e assim sucessivamente até que o cluster com $N/2(2^{\log_2 N-1})$ processos seja testado. Em seguida, o primeiro cluster é testado novamente no próximo intervalo, e o processo é repetido. Quando todos os nós estão livres de falhas, a atribuição de testes é equivalente a um hipercubo. No entanto, esta é uma topologia lógica e o sistema subjacente é considerado totalmente conectado, ou seja, qualquer nó é capaz de testar qualquer outro nó se isso for necessário.

A Tabela 2 mostra a função $c_{i,s}$ para 8 processos. Para determinar as arestas da topologia virtual, para cada nó i existe uma aresta (j, i), tal que j é o primeiro nó sem falhas em $c_{i,s}$, $s = 1... \log_2 n$. Quando um processo testador detecta que qualquer outro possa ter falhado, ele continua executando testes até que um encontre um processo sem falhas ou todos os processos do cluster sejam testados com falha. Uma rodada de testes ocorre quando todos os processos livres de falhas executam seus testes atribuídos. Por exemplo, na Figura 1, o processo p_0 originalmente testa o processo p_4 no *cluster* 3, mas depois que p_4 falha, o processo p_0 passa a testar o processo p_5, que é o próximo considerado correto na lista da $c_{0,3}$.

Tabela 2 – Resultado da função $c_{i,s}$ para 8 processos.

s	$c_{0,s}$	$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	$c_{6,s}$	$c_{7,s}$
1	1	0	3	2	5	4	7	6
2	2,3	3,2	0,1	1,0	6,7	7,6	4,5	5,4
3	4,5,6,7	5,4,7,6	6,7,4,5	7,6,5,4	0,1,2,3	1,0,3,2	2,3,0,1	3,2,1,0

Um processo sem falhas pode levar até $\log_2 N$ rodadas de teste para testar um determinado cluster, e a distância mínima de qualquer nó para qualquer outro nó é no máximo $\log_2 N$, portanto, a latência é $\log_2^2 N$ rodadas de teste no pior caso.

A topologia virtual vCube, utilizada neste trabalho, foi originalmente proposta no contexto de diagnóstico hierárquico adaptativo distribuído em nível do sistema (Hi-ADSD) (JR. et al., 2022). Embora todos os processos possam se comunicar diretamente (*topologia fully-connected*), os processos são organizados em uma rede de sobreposição

virtual escalável. No contexto de aplicações, o vCube já foi utilizado em soluções de violação de integridade (ZIWICH; DUARTE; ALBINI, 2005), difusão de mensagens (RODRIGUES; JR; ARANTES, 2014), exclusão mútua (RODRIGUES; JR; ARANTES, 2018) e Publish/Subscribe (ARAUJO et al., 2017), entre outras. Note que nestas soluções o sistema considerado é síncrono.

A solução proposta neste trabalho considera a execução da detecção de falhas do vCube em um ambiente assíncrono. Mais detalhes são apresentados na Seção 3.

2.5 Blockchain

A blockchain é conhecida como uma tecnologia disruptiva, devido ao seu potencial de aplicação em diversas áreas, ainda mais pelas suas propriedades de transparência, resiliência e integridade. Pode ser considerada um *"log"* de dados descentralizado, replicado, imutável e inviolável, ou seja os dados no blockchain não podem ser excluídos e qualquer pessoa pode ler os dados do blockchain e verificar sua precisão. Uma implicação importante dessa arquitetura é a não-existência de uma entidade centralizadora, assim várias partes não confiáveis ou semiconfiáveis podem interagir direta e transparentemente umas com as outras sem a presença de um intermediário confiável.

A criptografia é fundamental para garantir os requisitos de segurança, a confidencialidade e integridade dos dados de uma blockchain. Além de desempenhar um papel importante na autenticação de usuários e recursos.

Um dos recursos fornecidos pela criptografia mais cruciais para o funcionamento da blockchain são as funções de *hash* criptográficas. Uma *função hash* é uma função matemática que aceita uma string de entrada de qualquer comprimento e gera uma string *aleatória* de comprimento fixo. A função deve ser eficientemente computável, ou seja calcular o *hash* de uma string de n bits deve ter um tempo de execução $O(n)$ (NARAYANAN et al., 2016). Além disso, uma *função hash criptográfica* deve ser unidirecional, ou seja a partir de um *input* x podemos facilmente calcular o valor *hash* $H(x)$, mas é muito difícil a partir do valor *hash* $H(x)$ encontrar x.

Uma das propriedade mais importantes de uma função *hash* criptográfica é que ela seja resistente a colisões. Uma colisão ocorre quando duas entradas distintas produzem a mesma saída. De maneira formal: Uma função *hash* H é considerada resistente à colisão se for inviável encontrar dois valores, x e y, tais que $x \neq y$, mas $H(x) = H(y)$. Se a função *hash* H é resistente à colisão, dificilmente iremos encontrar dois valores x e y diferentes com o mesmo *hash*. Portanto, pode-se utilizar tais funções para, por exemplo, representar adequadamente resumos de informações (ou *message digests*). De modo simplificado, é possível identificar se o arquivo que eu recebi é o mesmo que o arquivo disponibilizado em uma central de arquivos, ao comparar o *hash* do meu arquivo e o *hash* do arquivo da

central. Com isso, atesta-se a integridade do arquivo recebido.

Outra propriedade principal das *funções hash* é a ocultação. Se recebermos a saída da *função hash* $y = H(x)$, não há maneira viável de descobrir qual era a entrada, x. Entretanto, se o conjunto de valores possíveis de x for muito pequeno, é fácil de deduzir x, basta aplicar o *hash* nas possibilidades e encontrar a qual valor de entrada produziu $H(x)$. Para poder alcançar a propriedade de ocultação, x deve ser escolhido de um conjunto que é, de certa forma, muito disperso. Pode-se ocultar até mesmo uma entrada que não está espalhada concatenando-a ($\|$) com outra entrada que está espalhada. Formalmente: Uma *função hash* H está ocultando se: quando um valor secreto r é escolhido de um universo com distribuição aleatória muito espalhada, então, dado $H(r||x)$, é inviável identificar x.

Uma aplicação da propriedade de ocultação é o compromisso (*commitment*). Um compromisso é o análogo digital de selar um valor em um envelope e colocá-lo sobre a mesa onde todos possam vê-lo. Ao fazer isso, você se comprometeu com o que está dentro do envelope. Mas você não o abriu, então mesmo que você tenha se comprometido com um valor, o valor permanece um segredo para todos os outros. Mais tarde, você pode abrir o envelope e revelar o valor com o qual se comprometeu anteriormente. Um esquema de compromisso consiste em dois algoritmos:

- ***com := commit(msg, nonce):*** A função *commit* recebe uma mensagem e um valor aleatório secreto, chamado de *nonce*, como entrada e retorna um compromisso.
- ***verify(com, msg, nonce):*** A função de verificação aceita um compromisso, um *nonce* e uma mensagem como entrada. Retorna *true* se *com == commit(msg, nonce)* e *false* caso contrário.

A blockchain utiliza funções *hashes* em várias partes de seu sistema. No encadeamento dos blocos, cada cabeçalho de bloco contém o *hash* do bloco anterior, o que garante que nada seja adulterado à medida que novos blocos são adicionados. As blockchains de criptomoeda usam *hashes* para proteger as informações e tornar o livro-razão imutável - no caso da criptomoeda, eles são usados para garantir que os dados contidos nos blocos de uma blockchain não sejam alterados.

Uma maneira de associar um *hash* a uma mensagem é com o uso de ponteiros de *hash*. Ponteiros são usados em estruturas de dados para permitir que um elemento de dados se refira a outro. Em sistemas distribuídos, um ponteiro pode ser um nome ou endereço IP de um computador e um identificador de objeto. Um ponteiro de *hash* é uma tupla que contém um ponteiro tradicional junto com o *hash* do elemento de dados que está sendo apontado. Permite validar que a informação apontada não foi modificada.

As funções de *hash* são parte vital das assinaturas digitais que garantem a integridade dos dados e são usadas para autenticação para transações de blockchain. A ideia

básica de uma assinatura digital é uma mensagem que pode ser criada por apenas uma pessoa, mas verificada por qualquer pessoa. Assim, ela pode desempenhar o tipo de função no mundo eletrônico que as assinaturas comuns desempenham no mundo do papel. Além da possibilidade de qualquer pessoa verificar a autenticidade de uma assinatura, também é necessário que essa assinatura não possa ser forjada e reutilizada em outro contexto.

O mecanismo de implementação de assinaturas digitais é por meio da criptografia de chave assimétrica. Utiliza-se um par de chaves, uma chave pública *pk* e uma chave privada *sk*. Uma informação cifrada com uma determinada chave pública só poderá ser decifrada através da chave privada correspondente. Um esquema de assinatura digital (NARAYANAN et al., 2016) consiste nos três algoritmos a seguir:

- *(sk, pk) := generateKeys(keysize)*: O método *generateKeys* recebe um tamanho de chave e gera um par de chaves. A chave secreta ***sk*** é mantida em sigilo e usada para assinar mensagens. ***pk*** é a chave de verificação pública, qualquer pessoa com esta chave pode verificar sua assinatura.

- *sig := sign(sk, message)*: O método *sign* recebe uma mensagem e uma chave secreta, sk, como entrada e emite uma assinatura para mensagem sob sk.

- *isValid := verify(pk, message, sig)*: O método *Verify* recebe uma mensagem, uma assinatura e uma chave pública como entrada. Ele retorna um valor booleano, *isValid*, que será verdadeiro se *sig*, a assinatura para mensagem sob a chave pública ***pk*** for válida e falso caso contrário.

Mensagens costumam ser muito longas, então costuma-se assinar o resumo criptográfico H de uma mensagem *msg* pois *hashes* têm quantidade fixa de bits. Assim, adota-se *sig := sign(sk,H(msg))*. Os endereços na blockchain, importantes para identificar os nós nas transações, são resumos criptográficos das chaves públicas dos envolvidos.

As assinaturas digitais são fundamentais em blockchains, usadas principalmente para autenticar transações. Quando os usuários enviam transações, eles devem provar a todos os nós do sistema que estão autorizados a gastar esses fundos, evitando que outros usuários os gastem. Cada nó na rede verificará a transação enviada e verificará o trabalho de todos os outros nós para concordar com um estado correto.

Em uma blockchain, uma variedade de usuários está transmitindo transações para a rede e os nós (participantes da rede) devem concordar exatamente sobre quais transações foram transmitidas e a ordem em que essas transações aconteceram. Isso resultará em um único registro (*ledger*) global para o sistema. Portanto, em qualquer ponto, todos os nós da rede peer-to-peer têm um registro (livro-razão) que consiste em uma sequência de blocos, cada um contendo uma lista de transações, sobre as quais chegaram a um consenso.

As transações são agrupadas em um bloco. Um bloco é apenas uma lista parcial de transações. Quando um nó está pronto para fazer isso, ele pode adicionar o bloco ao *ledger*, formando uma lista encadeada de blocos que compõem a blockchain. Adicionar um novo bloco é fácil. Aloca-se o bloco, copia o ponteiro de *hash* de cabeçalho para ele (o próximo ponteiro) e atualiza o ponteiro de *hash* de cabeçalho para apontar para o novo bloco e conter um *hash* desse bloco.

Uma técnica de tolerância a falhas que permite manter a consistência de um serviço replicado, é a replicação máquina de estados (RME) (SCHNEIDER, 1990). Uma máquina de estados representa um conjunto de estados do processo e um conjunto de transições entre estados. Uma transição ocorre como consequência de um evento. Na RME cada réplica é uma máquina, todas as máquinas iniciam com o mesmo estado e executam a mesma sequência de operações, de modo que todas as máquinas de estado são idênticas o tempo todo.

A blockchain pode ser vista como um mecanismo de transição de estado pelo qual um estado é modificado de sua forma inicial para a próxima e, eventualmente, para uma forma final por nós na rede blockchain como resultado de um processo de execução, validação e finalização da transação (BASHIR, 2017). Além disso, a blockchain implementa uma RME, de modo a manter consistente o *ledger* distribuído em cada um dos nós da rede. Os blocos de transação são agregados seguindo uma ordem, assegurada por meio de um protocolo de difusão atômica.

Os dois grupos de classificação de redes blockchain são: blockchain pública, e blockchain privada/federada. Em uma blockchain pública, a rede P2P permite uma composição dinâmica de nós, que podem sair e entrar da rede sem a necessidade de identificação.

2.5.1 Consenso para Blockchain Pública

Em blockchains públicas, como a Bitcoin e o Ethereum, os nós pertencentes a rede não têm identidades persistentes e de longo prazo. Não existe uma autoridade central para atribuir identidades aos participantes e verificar se eles não estão criando novos nós à vontade. O termo técnico para isso é um ataque Sybil (DOUCEUR, 2002). Sybils são cópias de nós que um adversário malicioso pode criar para parecer que há muitos participantes diferentes, quando na verdade todos esses pseudoparticipantes são realmente controlados pelo mesmo adversário.

A ideia central por trás dos mecanismos de consenso usuais em uma blockchain pública é a seleção de um nó "aleatório", selecionando nós em proporção a um recurso que espera-se que ninguém possa monopolizar. Se esse recurso for o poder computacional, então é um sistema de Prova de Trabalho (*Proof-of-Work*). Como alternativa, pode ser

proporcional à posse da moeda, e isso é chamado de *Proof-of-Stake*.

A suposição de seleção de nó aleatório torna possível algo chamado consenso implícito (NARAYANAN et al., 2016). Existem várias rodadas no protocolo, cada uma corresponde a um bloco diferente na cadeia de blocos. Em cada rodada, um nó aleatório é selecionado de alguma forma, e esse nó propõe o próximo bloco na cadeia. Não há algoritmo de consenso para selecionar o bloco e nenhum tipo de votação. O nó escolhido propõe unilateralmente qual será o próximo bloco na cadeia de blocos. Outros nós aceitam ou rejeitam implicitamente esse bloco, escolhendo se desejam ou não construir sobre ele. Se eles aceitarem esse bloco, sua aceitação é sinalizada estendendo a cadeia de blocos incluindo o bloco aceito. Por outro lado, se os nós rejeitarem esse bloco, eles estendem a cadeia ignorando esse bloco e construindo em cima do bloco aceito anteriormente.

Os nós pertencentes a uma rede blockchain enviam requisições de transações a serem adicionadas à rede e ordenadas ao longo do tempo. Tais transações são agrupadas em conjuntos (blocos) e o consenso é realizado em rodadas, de modo a possibilitar a ordenação total dos blocos. O consenso da Bitcoin (NAKAMOTO, 2008), utiliza a seleção de um nó "aleatório" para transmitir seu bloco.

Os nós Bitcoin realizam uma Prova de Trabalho, resolvendo um desafio criptográfico de modo a se eleger como líder. Os nós estão simplesmente competindo independentemente para resolver esses desafios criptográficos o tempo todo. De vez em quando, um nó encontra um *nonce* aleatório que satisfaça a propriedade. Esse nó sortudo então pode propor o próximo bloco. Assim o sistema é totalmente descentralizado. Não há ninguém decidindo qual nó propõe o próximo bloco.

Se um bloco anunciado contiver transações inválidas ou alguma outra inconsistência, todos os outros nós destinam-se a rejeitá-lo e continuar trabalhando até encontrar uma solução para um bloco válido. O primeiro bloco válido anunciado contendo uma solução para o desafio é considerado correto. Ao saber disso, outros participantes começam a trabalhar para encontrar o próximo bloco. Assim, a rodada do consenso chega ao seu término, sem que os nós interajam para efetuar confirmação do valor proposto pelo líder. Eles simplesmente validam e agregam o bloco aceito ao livro-razão distribuído, seguindo para uma próxima rodada de consenso.

A qualquer momento, a blockchain de consenso é a versão "mais longa". Normalmente, é a cadeia com mais blocos, mas como a dificuldade de mineração pode variar entre bifurcações longas, a cadeia mais longa deve ser definida como aquela com a maior dificuldade esperada para produzir (BONNEAU et al., 2015). É possível que duas soluções válidas sejam encontradas aproximadamente ao mesmo tempo (dependendo da latência da rede), o que leva a uma bifurcação temporária durante a qual existem duas cadeias de comprimento igual. Os mineradores podem escolher qualquer *fork* neste cenário. Devido à natureza aleatória do desafio criptográfica, uma cadeia será eventualmente estendida além

do outra, ponto em que todos os nós mineradores devem adotá-la. A natureza gradual desse mecanismo de consenso implica que os usuários devem esperar que os blocos sejam encontrados para obter alta confiança de que uma transação está permanentemente incluída no blockchain.

Nós mineradores competem para a criação de novos blocos com as transações processadas. O vencedor, além de coordenar aquela rodada de consenso, recebe bitcoins pelo seu trabalho. A geração da moeda bitcoin integra-se ao processo de eleição randômica. Esse mecanismo provoca os nós maliciosos a seguir a ordem instituída pelos nós honestos, pois os ganhos financeiros em agir honestamente, têm a tendência de compensar as atuações maliciosas.

A Ethereum utiliza um mecanismo de consenso de prova de posse (*proof-of-stake*) que deriva sua segurança de um conjunto de recompensas e penalidades aplicadas ao capital bloqueado pelas partes interessadas. Essa estrutura de incentivos encoraja participantes individuais a operar validadores honestos, pune aqueles que não o fazem e cria um custo extremamente alto para atacar a rede (WOOD et al., 2014).

Esse algoritmo trabalha com a ideia de que um nó ou usuário tem uma participação adequada no sistema, ou seja, o usuário investiu o suficiente no sistema para que qualquer tentativa maliciosa desse usuário superasse os benefícios de realizar algum ataque na rede. Outro conceito importante no PoS é a idade da moeda (*coin age*), que é um critério derivado da quantidade de tempo e número de moedas que não foram gastas. Nesse modelo, as chances de propor e assinar o próximo bloco aumentam com a idade da moeda.

Na *proof-of-stake*, os validadores explicitamente apostam capital na forma de ETH (moeda Ethereum) em um contrato inteligente na Ethereum. Esse ETH apostado atua como garantia que pode ser destruído se o validador se comportar de maneira desonesta ou preguiçosa. O validador é então responsável por verificar se os novos blocos propagados pela rede são válidos e, ocasionalmente, criar e propagar novos blocos.

Para participar como validador, o usuário deve depositar 32 ETH em um contrato de depósito e executar três *softwares* separados: um cliente de execução, um cliente de consenso e um validador (WOOD et al., 2014). Ao depositar seu ETH, o usuário entra em uma fila de ativação que limita a taxa de entrada de novos validadores na rede. Uma vez ativados, os validadores recebem novos blocos de pares na rede Ethereum. As transações entregues no bloco são reexecutadas e a assinatura do bloco é verificada para garantir que o bloco seja válido. O validador então envia um voto (chamado de atestado) a favor desse bloco pela rede.

O tempo no Ethereum é dividido em tempos fixos, *slots* (12 segundos) e *epochs* (32 slots). Um validador é selecionado aleatoriamente para ser um proponente de bloco em cada *slot*. Este validador é responsável por criar um novo bloco e enviá-lo para outros

nós da rede. Também em cada *slot*, um comitê de validadores é escolhido aleatoriamente, cujos votos são usados para determinar a validade do bloco que está sendo proposto.

Nas raras situações em que vários blocos estão na mesma posição perto da cabeça da cadeia, existe um mecanismo de *fork-choice* que seleciona os blocos que compõem a cadeia "mais pesada", medida pelo número de validadores que votaram nos blocos ponderados por seu saldo de ETH apostado.

2.5.2 Consenso para Blockchain Permissionada

Em blockchains permissionadas, o conjunto de participantes é bem definido e bem identificados. Dessa forma, é possível utilizar estratégias de consenso já bem estabelecidas como o *Byzantine Fault Tolerance* e consenso bizantino determinístico para eleger o líder da rodada. Estas estratégias focam em chegar a um consenso mesmo quando há a existência de nós maliciosos (ou com comportamento arbitrário) na rede. Existem diversos métodos de consenso para tolerância a falhas bizantinas, como HiBFT (THAI et al., 2019) - também utilizado em blockchains permissionadas, Tendermint (BUCHMAN, 2016) - baseado em PBFT, entre outros. Entretanto, aqui será abordado o algoritmos mais relevantes para compreensão da blockchain implementada. A principal vantagem de uso do consenso BFT para a blockchain é a latência de decisão e a alta vazão de transações.

O Projeto Hyperledger[1] é um esforço colaborativo para criar uma estrutura de ledger distribuído de código aberto e de nível empresarial e uma base de código (ANDROULAKI et al., 2018). Estabelecido como um projeto da Linux Foundation no início de 2016, o Hyperledger Fabric[2] é uma implementação de uma plataforma de *ledger* distribuída para executar contratos inteligentes, com uma arquitetura modular que permite implementações conectáveis de várias funções (CACHIN et al., 2016). O Hyperledger Fabric implementa uma blockchain permissionada que utiliza o modelo clássico de consenso PBFT (*Pratical Byzantine Fault Tolerance*).

O algoritmo de replicação de máquina de estado (RME), PBFT (CASTRO; LISKOV, 2002), pode ser usado para criar sistemas altamente disponíveis que toleram falhas bizantinas. O mecanismo de recuperação permite que o algoritmo tolere qualquer número de falhas durante a vida útil do sistema, desde que o número máximo de nós maliciosos não seja maior ou igual a um terço de todos os nós do sistema, ou seja, menos de 1/3 das réplicas apresentem falhas dentro de uma pequena janela de vulnerabilidade.

O algoritmo de consenso no PBFT também é realizado em rodadas coordenadas por um líder. Os líderes se sucedem em sequências de visualizações (rodadas de consenso PBFT). Dentro de cada visualização, os nós monitoram o comportamento do líder, que pode ser honesto ou malicioso, caso os nós suspeitem da falha do líder, ele é substituído por

[1] <www.hyperledger.org>
[2] <github.com/hyperledger/fabric>

um protocolo de mudança de visualização. Os nós em um sistema distribuído habilitado para PBFT são ordenados sequencialmente com um nó sendo o primário (ou o nó líder) e outros referidos como secundários (ou os nós de *backup*). Qualquer nó elegível no sistema pode se tornar o líder ao fazer a transição de secundário para primário (normalmente, no caso de uma falha do líder).

O PBFT exige que os nós se comuniquem por meio de uma série de mensagens antes de chegarem a um acordo sobre o próximo bloco a ser adicionado ao registro. Esse processo garante que todos os nós concordem com o mesmo conjunto de transações. No PBFT as mensagens trocadas entre as réplicas em todas as etapas do algoritmo são transmitidas por difusão (*broadcast*). A difusão atômica (CHANDRA; TOUEG, 1996) garante que as mensagens são entregues por todos os processos do sistema exatamente na mesma ordem. O algoritmo de difusão atômica é organizado em rodadas, uma instância do consenso é executada para definir a operação seguinte no RME a cada rodada, assim decide-se qual mensagem vai ser entregue naquela rodada, e as mensagens são entregues em uma ordem determinística, predefinida. Dessa forma, a difusão atômica vai garantir que a sequência de operações executadas por todos os processos seja a mesma.

A difusão atômica ou *atomic broadcast* (HADZILACOS, 1993) garante que as mensagens serão entregues por todos os processos na mesma ordem. Em um modelo de falhas bizantinas, em que até f réplicas podem se comportar maliciosamente, um cliente identifica que sua requisição foi executada com sucesso quando recebe $f + 1$ respostas idênticas dos servidores.

A difusão atômica e o consenso apresentam igualdade em qualquer modelo de falhas, ou de forma mais precisa, em todos os modelos onde a difusão confiável pode ser solucionada (CHANDRA; HADZILACOS; TOUEG, 1996). Isso implica que qualquer algoritmo destinado ao consenso pode ser empregado para resolver a difusão atômica e, inversamente, também é válido.

Um dos métodos de difusão confiável utilizados no contexto de blockchains, é o *gossip* (difusão epidêmica) (EUGSTER et al., 2004). Neste algoritmo, todos os processos do sistema podem ser envolvidos na disseminação de informações, cada processo recebe mensagens e encaminha-as para um número de nós selecionado aleatoriamente limitado ("vizinhos"), assim assume-se que depois de um tempo, há uma alta probabilidade que todos os processos do sistema tenham recebido as mesmas mensagens. O trabalho de (AYSAL et al., 2009) comprova a utilização do *gossip* como mecanismo de consenso.

Além do PBFT, outras técnicas de Replicação de Máquina de Estados estão sendo usadas como algoritmo de consenso em blockchains permissionadas, como por exemplo o Paxos (LAMPORT, 1998) e o Raft (ONGARO; OUSTERHOUT, 2014), ambas estratégias tradicionais baseadas em rodadas de votos para eleição do líder.

O Paxos (LAMPORT, 2001) é um algoritmo para obtenção de consenso, principalmente dentro da replicação de estado de máquina, no qual diversos processos podem propor valores. O algoritmo Paxos segue as seguintes diretivas: (1) Somente um valor que foi proposto será escolhido; (2) Somente um valor será escolhido; (3) Um processo não aprende um valor escolhido, a não ser que este valor tenha sido escolhido. Os processos podem assumir três papéis dentro do algoritmo, aquele de *proposers* – processos que propõem um próximo valor a ser escolhido, *learners* – processos que aprendem a decisão, *acceptors* – processos que decidem qual o próximo valor a ser escolhido.

No Paxos, existem 2 fases. Na primeira fase o processor *proposer* envia uma mensagem para uma maioria dos processos *acceptors*. O *acceptor* concorda em aceitar e responde com sua aceitação. Na segunda fase, se o *proposer* recebeu o aceite da maioria dos processos, enia uma mensagem com o valor.

Raft e Paxos são algoritmos de tolerância a falhas *crash*. Embora seus protocolos sejam semelhantes aos algoritmos BFT, eles só podem tolerar falhas em até 50% dos nós, enquanto os algoritmos bizantinos podem tolerar nós corrompidos com comportamento arbitrário (nós agindo maliciosamente). Ambos os algoritmos são compostos por dois estágios: eleição do líder e replicação de log. O líder é responsável por ordenar as transações. O estágio de seleção do líder é executado usando um tempo limite aleatório para cada servidor quando um líder existente falha. Quando um líder é escolhido, o estágio de replicação de log é acionado. Neste estágio, o líder aceita entradas de log de clientes e transmite transações para fazer sua versão do log de transações.

O Raft implementa o consenso elegendo primeiro um líder e, em seguida, dando ao líder a responsabilidade completa pelo gerenciamento do log replicado. O líder aceita entradas de log de clientes, replica elas em outros servidores e informa aos servidores quando é seguro aplicar entradas de log às suas máquinas de estado. Ter um líder simplifica o gerenciamento do log replicado. Por exemplo, o líder pode decidir onde colocar novas entradas no log sem consultar outros servidores, e os dados fluem de forma simples do líder para outros servidores. Um líder pode falhar ou se desconectar dos outros servidores, caso em que um novo líder é eleito (ONGARO; OUSTERHOUT, 2014). Corda (BROWN et al., 2016) e Quorum (CHASE, 2018) são duas implementações de blockchain que usam Raft como seu método de consenso.

2.6 vCubeChain

A vCubeChain (FREITAS; RODRIGUES; Duarte Jr., 2023) implementa uma blockchain permissionada escalável, na qual os processos formam um VCube e utilizam do serviço disponibilizado de detecção de falhas para eleger um líder. Assume-se que os processos presentes na rede são autenticados, assim, não existem ataques de personificação

nos quais um oponente simula ser o líder. No entanto, é importante observar que, devido a falsas suspeitas, à demora na propagação das informações do estado dos processos pelo sistema, ou em cenário de falha do líder, é possível que, por algum período de tempo, mais de um dos processos se considere como líder. Mesmo assim, a vCubeChain converge para um único líder, após um período finito de tempo, de modo a manter a consistência global. O líder na vCubeChain é o processo correto de maior identificador na rede, quando surgem falsas suspeitas de falha do líder, os próximos processos com identificadores superiores podem se considerar o líder, entretanto o sistema sempre converge para um único líder.

A vCubeChain utiliza mesma estrutura de dados que a maioria das blockchains, cada bloco, desde o primeiro - denominado gênese e proposto pelo primeiro líder – consiste em um conjunto de transações bem como do resumo criptográfico (*hash*) do bloco anterior e do *hash* do bloco atual.

A proposição de blocos na rede inicia-se com os usuários enviando novas transações a qualquer processo, se este processo não é líder, envia a transação ao líder (processo correto de maior identificador). O líder então valida a transação e a inclui em um bloco candidato, construido com um conjunto de transações válidas. O bloco completo é disseminado na vCubeChain por meio do RBCAST, um algoritmo de difusão confiável autonômica (JEANNEAU et al., 2017) que utiliza a topologia hierárquica do vCube para assegurar que a propagação dos blocos pelo sistema irá precisar de um número logarítmico de passos de comunicação para se completar.

Antes do processo enviar a transação ao líder, a transação é salva em um *buffer*, sendo removida somente quando o processo receber um bloco que a contém ou uma notificação do líder de que a transação não é válida (de acordo com o estado do livro-razão). Se o líder atual se torna suspeito, o processo envia todas as transações deste *buffer* para o mais novo líder eleito. Se ambos líderes, suspeito e novo, persistirem transações em novos blocos, então um *fork* (bifurcação) ocorre.

Através de uma função, os processos aprendem quem é o líder atual. No caso de falha do líder, é eleito o processo de maior identificador entre os processos corretos. Os processos devem utilizar a primitiva adequadamente, para saberem se houve alguma mudança de líder. Quando um processo correto é incorretamente suspeito de ter falhado, ele pode voltar à liderança assim que o vCube contornar a suspeita incorreta. Como se assume o modelo GST, a partir de um certo instante de tempo o líder não será mais suspeito incorretamente.

O algoritmo de consenso da vCubeChain é baseado no resultado de (DOLEV; DWORK; STOCKMEYER, 1987), onde os autores provam que o consenso pode ser reduzido à difusão confiável com ordenação de mensagens. Em um cenário ideal, o líder recebe transações dos demais processos do sistema. O líder estabelece uma ordem local sobre as transações, após verificar sua validade. Com a validade garantida, o líder propõe

um bloco contendo as transações. Este bloco é disseminado utilizando o mecanismo de difusão confiável do vCube, que garante sua entrega por todos os processos corretos após um intervalo de tempo finito usando uma árvore de difusão (*spanning tree*).

Diante de instabilidades ou possível assimetria de informações, é possível que dois processos p e q se tornem líderes, ambos propondo blocos na cadeia conhecida. A vCubeChain permite o *fork* da blockchain, ou seja, duas subcadeias se bifurcam ao agregar blocos discordantes. Devido à difusão confiável, todos os processos corretos terão a mesma visão da blockchain, incluindo o *fork* e as subcadeias derivadas. Neste contexto, as transações são sugeridas com base nas informações da cadeia secundária mais extensa. Se o líder recebe transações provenientes do último bloco que ele não conhece, ele não as insere em uma proposta de bloco até que esteja a par delas.

A formação de subcadeias a partir de um *fork* requer um mecanismo de ajuste de cadeia. Na vCubeChain, a difusão confiável envolve confirmações de recebimento do bloco pelos processos corretos. O líder pode determinar quais blocos propostos foram recebidos pelo conjunto de processos percebidos como corretos. Assume-se uma janela de estabilidade de B blocos, ou seja, se um processo p permanecer líder e tiver pelo menos B blocos confirmados a frente de outra cadeia, sua cadeia pode ser considerada estável. Assim, as transações da cadeia que não persiste deverão ser reenviadas.

Essa estratégia de consenso é uma solução de compromisso baseada na premissa de que, após eventuais disputas entre líderes, a percepção dos processos corretos irá convergir. Assim, nessa convergência, o líder poderá resolver esses *forks* após um número B de blocos confirmados, dissolvendo quaisquer divergências com outro líder.

2.6.1 Resultados prévios da vCubeChain

A seguir são apresentados os resultados parciais disponibilizados e os cenários utilizados por Freitas, Rodrigues e Duarte Jr. (2023), ainda considerando a execução do vCube em um sistema síncrono.

O simulador BlockSim (FARIA; CORREIA, 2019) não suporta, originalmente, a simulação de falhas. Dessa forma, no artigo original, a vCubeChain foi comparada as implementações do Ethereum e Bitcoin já presentes no BlockSim apenas em cenários sem falhas.

Utilizou-se os parâmetros de configuração definidos em Faria e Correia (2019). Com o Ethereum utilizando o valor padrão de *gas limit* - limite máximo que um nó na rede Ethereum se dispõe a pagar por uma transação - de 21.000 e o *gas limit* para o bloco - limite de transações que podem ser inseridas em um bloco - de 2.1 milhões. A Tabela 3 demonstra a distribuição de nós em três *sites*. Por exemplo, para 8 processos, há dois nós em Cascavel, dois nós em Salvador e quatro nós em Curitiba. Um dos nós em Curitiba é

configurado como minerador na execução de Bitcoin ou Ethereum e como líder ao executar vCubeChain.

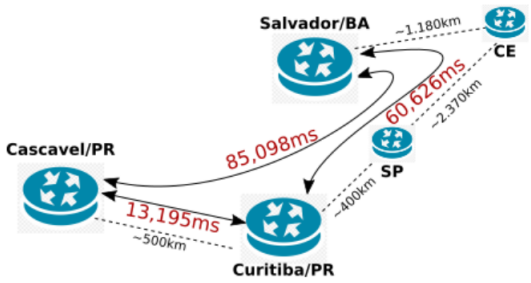
Tabela 3 – Locais onde os nós são alocados.

Sites	Não-mineradores	Minerador/Líder
Cascavel	N/4	0
Salvador	N/4	0
Curitiba	(N/2) - 1	1

Fonte: Freitas, Rodrigues, Jr (2023)

A latência em segundos entre os nós em cada site foi configurada usando os parâmetros da Figura 2. Os valores foram calculados com base nos dados de RTT reais (ping) obtidos na rede da RNP (Rede Nacional de Pesquisa). Nós em locais diferentes utilizaram a distribuição normal (média e desvio padrão) e a distribuição Gama inversa foi usada para nós no mesmo local. Os parâmetros completos estão disponíveis no Github[3].

Figura 2 – Latência em milissegundos entre os nós de cada *site*



Fonte: Freitas, Rodrigues, Jr (2023)

Para cada execução, foram simuladas mil transações, divididas em mil rodadas de uma transação a cada 15 segundos. As transações foram distribuídas aleatoriamente entre os processos usando a função *transaction factory* do simulador.

As simulações foram executadas usando um computador Dell Inc. XPS 8950 comCPU 12th Gen Intel® Core™ i7-12700 × 20, 16 GB de RAM e 512 GB SSD, sistema operacional Ubuntu 22.04.1 LTS de 64 bits.

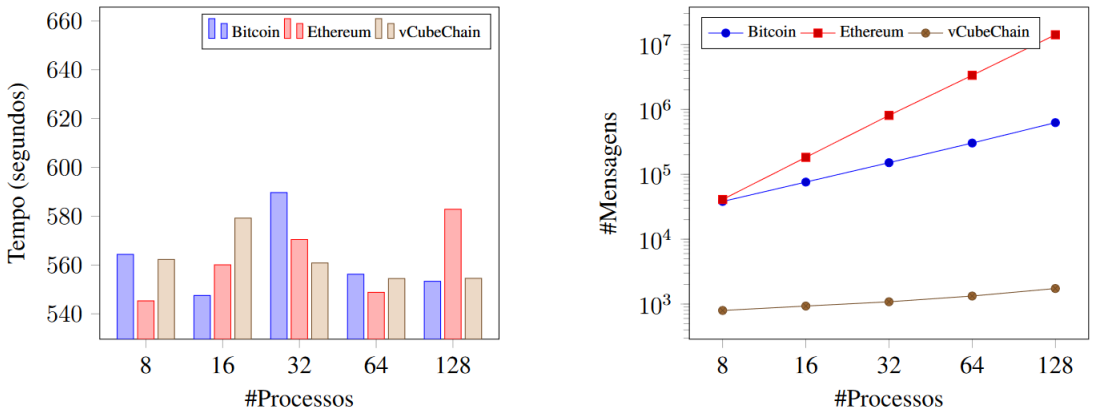
A Figura 3 apresenta o tempo médio necessário para validar todas as transações (figura na esquerda) e o número total de mensagens enviadas (figura na direita). A comparação entre as três soluções foca em escalabilidade de tempo e de número de mensagens, devido a falta de suporte à simulação de cenários com falhas.

O tempo médio necessário para validar todas as transações é calculado a partir de 30 execuções em cada cenário.Os três algoritmos simulados têm um tempo de validação

[3] <https://github.com/arluiz/FD-blocksim>

de transações similar. Apesar disso, o tempo de execução da simulação do Ethereum foi significativamente maior quando comparado ao Bitcoin e vCubeChain.

Figura 3 – Tempo médio para validar todas as transações (esq.) e número médio de mensagens em escala logarítmica (dir.)



(a) Tempo de execução (b) Número de Mensagens

Fonte: Freitas, Rodrigues, Jr (2023)

O número total de mensagens enviadas, apresentadas em escala logarítima para facilitar a visualização, pode ser observada na direita da Figura 3. O Bitcoin, Ethereum e a vCubeChain utilizam uma estratégia de disseminação para o cabeçalho e o corpo da mensagem, onde para cada mensagem contendo o cabeçalho de uma transação ou bloco, envia-se uma solicitação para o corpo (texto) da mensagem, gerando mais uma nova mensagem com o conteúdo. Apesar desta estratégia de disseminação ser a mesma, a vCubeChain utiliza uma árvore de difusão para a transmissão da mensagem, reduzindo satisfatoriamente o número de mensagens trocadas. Além disso, o Bitcoin e o Ethereum utilizam-se da transmissão de um-para-todos e os processos no Bitcoin enviam mensagens de início da comunicação para disseminar o conhecimento acerca das novas transações e blocos.

O número de blocos criados na execução do Ethereum é limitado pelo número de transições permitidas pelo *limit gas* do bloco. O Bitcoin e a vCubeChain utilizaram o limite de tamanho do bloco físico (2MB), assim, em todos os cenários, todas as transações foram agrupadas em um único bloco.

Ao observar as conexões TCP abertas em uma execução sem falhas, vCubeChain abre menos conexões porque só se comunica com o líder e até $\log_2(N)$ vizinhos, enquanto Bitcoin e Ethereum abrem $(N^2 - N)/2$ conexões, em uma estratégia um-para-todos.

2.7 Simuladores de Blockchain

Esta seção discute os simuladores utilizados para o estudo de blockchains encontrados na literatura, em especial o Blocksim (FARIA; CORREIA, 2019), utilizado neste trabalho.

Embora haja interesse generalizado em desenvolver sistemas blockchain para casos de uso específicos, faltam ferramentas para realizar sua avaliação. As avaliações atuais são frequentemente baseadas na emulação, que imita o comportamento de um sistema sobre um grande número de arquivos. Essa abordagem, no entanto, gera uma grande sobrecarga e carece de escalabilidade para implantações no mundo real. Além disso, o consumo de energia de um sistema de grande escala precisa ser considerado.

Uma alternativa é a simulação. Simuladores de redes e sistemas distribuídos são ferramentas importantes para avaliar o desempenho de protocolos e sistemas sob uma ampla gama de condições. Os simuladores fornecem um ambiente que simplifica a implementação e implantação de protocolos. Com a simulação é possível estudar um sistema de grande escala com milhares de nós em uma máquina e obter resultados em um tempo razoável.

A linguagem na qual o simulador foi implementado, os requisitos de instalação, a facilidade da configuração dos arquivos de simulação e o formato da saída foram as principais propriedades selecionadas. Devido a necessidade de um simulador que forneça as informações necessárias tanto para realizar a implementação da topologia, como também para testar seu funcionamento e adquirir o maior número possível de estatísticas dos cenários. A seguir estão dispostas as características agregadas:

- Linguagem de Implementação: *BlockSim* do Faria e *BlockSim* do Alharby utilizam *Python*, *SimBlock* é implementado em *Java*.
- Requisitos de Instalação:
  - BlockSim (ALHARBY; MOORSEL, 2020): *python 3* ou superior e as bibliotecas `pandas, numpy, sklearn, xlsxwriter`.
  - BlockSim (FARIA; CORREIA, 2019): *python 3* ou superior e as bibliotecas `simpy 3.0.11, schema, scipy 1.3.3, pysha3`.
  - SimBlock (AOKI et al., 2019): `JDK 1.8.0` ou superior, `Gradle 5.1.1` ou superior.
- Configuração:
  - BlockSim (ALHARBY; MOORSEL, 2020): o arquivo *InputsConfig.py* tem os parâmetros de configuração, no qual é possível selecionar qual o modelo base para ser utilizado, assim como os parâmetros do bloco, transação, nó e simulação.

Entretanto, a alteração do parâmetro *Ttechnique*, que especifica a maneira da modelagem das transações conforme está descrito no artigo, causa erros que não permitem a execução do simulador, assim, se torna necessário alterar os arquivos dos modelos que já estariam supostamente prontos, para realizar a simulação com o parâmetro especificado pelos autores.

  - BlockSim (FARIA; CORREIA, 2019): o arquivo *config.json* permite configurar os parâmetros para execução, sendo eles: blockchain que aponta para um grupo específico de modelos disponíveis e a lista de todos os locais disponíveis correspondentes às medições em *throughput-received.json*, *throughput-sent.json* e *latency.json*, que podem ser alterados conforme a necessidade da simulação.
  - SimBlock (AOKI et al., 2019): seu arquivo *SimBlock.settings* tem duas classes com os parâmetros de configuração: *NetworkConfiguration.java* e *Simulation-Configuration.java*, nas quais é possível configurar os parâmetros relacionados a rede, e os da blockchain, conforme esses parâmetros são alterados é possível simular diferentes cenários.

- Formato do Saída: *BlockSim* do Faria e *SimBlock* disponibilizam a saída da simulação no formato JSON, enquanto o *Blocksim* do Alharby disponibiliza uma tabela no formato XML. As informações em comum entre os três simuladores são: quantidade de blocos e cadeia de eventos (*timestamp* de transmissão, bloco remetente e bloco destinatário).

  - Os resultados do *BlockSim* de Alharby incluem o *ledger* blockchain, número de blocos minerados, número de blocos obsoletos (*uncles*) e as recompensas obtidas por cada minerador, como os resultados ficam dispostos em uma planilha *excel*, a exploração e visualização é muito fácil, entretanto, não se tem informações muito avançadas.
  - Uma funcionalidade interessante do *output* fornecido pelo *SimBlock* é o visualizador online disponibilizado pelos autores, no qual é possível carregar o arquivo da sua simulação e visualizar os nós, as transações e a propagação do bloco. Entretanto, seus arquivos de saída não contêm muitas informações, e aquelas que estão presentes, acabam ficando desorganizadas dentro do arquivo JSON, de modo que se faz necessário um *script* para extrair os dados.
  - Assim como no *SimBlock*, os dados resultantes da execução do *BlockSim* do Faria estão em um enorme arquivo JSON, entretanto, é possível visualizar que mais dados relevantes estão presentes.

Para avaliação do modelo foi importante saber no mínimo: o número de transações de cada nó, as transações adicionadas na fila e tempo de propagação de blocos. Dados

adicionais ajudariam na análise necessária após a implementação das extensões necessárias para a topologia *VCube*.

Após a comparação de desempenho dos três simuladores e as estatísticas resultantes, por meio da configuração de diferentes cenários, foi possível estabelecer qual simulador demonstra maior possibilidade de extensão, a partir do qual será implementada o mecanismo de simulação de falhas. Além disso, deve ser levado em consideração a escolha dos autores do vCubeChain, de modo que os resultados anteriores e os cenários de testes possam ser replicados e comparados. Desse modo, escolheu-se o simulador BlockSim (FARIA; CORREIA, 2019), ademais de suas características demonstrando a facilidade de extensão, a blockchain vCubeChain já está implementada e disponibilizada.

Assim, é possível realizar testes e determinar a escalabilidade e benefícios de uma blockchain sobre o *VCube* em cenários com falhas de nós, comparando-a com simulações dos protocolos mais conhecidos de blockchain.

2.7.1 BlockSim

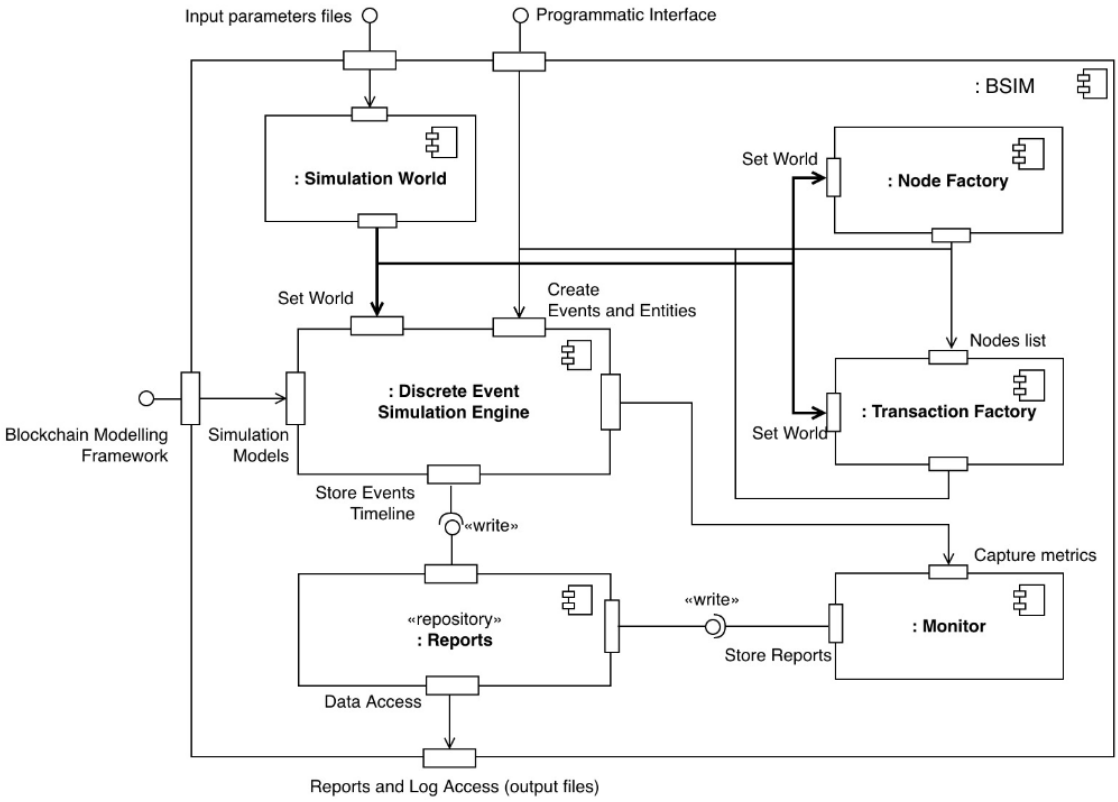
O Blocksim (FARIA; CORREIA, 2019) é um simulador de eventos discretos, desenvolvido em Python. No qual blockchains podem ser implementados de forma simples e ter seu desempenho avaliado em diferentes condições. O BlockSim fornece uma estrutura e um conjunto de modelos básicos de simulação comuns a vários blockchains (blocos, transações, *ledger*, rede). Esses modelos podem ser estendidos para avaliar suas próprias decisões de *design* e implementação. O *framework* recebe os modelos criados e os executa no simulador, de acordo com um conjunto de eventos definidos pelos usuários. Essa abordagem fornece uma solução versátil sem o necessidade de implementar um simulador do zero e pode ser estendida para simular outros blockchains.

O artigo *vCubeChain: Uma Blockchain Permissionada Escalável*, utilizou o BlockSim para implementar a vCubeChain e comparar com o Bitcoin e o Ethereum. O simulador não suportava simulação de falhas, assim a avaliação das soluções se concentrou na comparação de escalabilidade, principalmente no tempo de execução e no número de mensagens. Um dos objetivos principais do presente trabalho, é extender o simulador de modo a fornecer o suporte a avaliação com simulação de falhas.

A arquitetura do BlockSim está presente na Figura 4. A figura mostra os principais componentes, conectores e interfaces da implementação do simulador, que serão descritos a seguir.

Discrete Event Simulation Engine: O núcleo de um simulador de eventos discretos é o mecanismo de simulação de eventos discretos, ou seja, *Discrete Event Simulation Engine* (DESE). Os autores aproveitaram uma estrutura existente chamada SimPy (SimPy. . . ,). SimPy é uma estrutura de simulação de eventos discretos baseada em processo e em Python

Figura 4 – Arquitetura do BlockSim



Fonte: Faria, Correia (2019)

padrão. Os processos no SimPy são baseados em funções do gerador Python e podem ser usados para simular redes assíncronas ou para implementar sistemas multiagentes. Os geradores permitem que o programador especifique uma determinada função para ser encerrada e, posteriormente, reentrada no ponto da última saída, permitindo que as funções alternem a execução entre si. A saída e a reentrada são executadas usando a palavra-chave *yield* do Python.

O componente DESE suporta várias funcionalidades principais, tais como: agendamento de eventos; filas e processamento de eventos; comunicação entre componentes; gerenciamento do relógio de simulação; e controlar o acesso de recursos pelas entidades. O usuário do BlockSim também pode usar todas as funcionalidades do SimPy ao criar novos modelos. No entanto, o SimPy é uma estrutura para construir modelos ou simuladores arbitrários, enquanto o BlockSim fornece uma estrutura mais personalizada para simular blockchains, fornecendo componentes adicionais.

Simulation World: É responsável por tratar os parâmetros de configuração da simulação, organizados como um conjunto de arquivos:

- Arquivo de Configuração: informa o nome da blockchain que está sendo simulada,

localização dos nós, configurações que dependem da blockchain que está sendo simulada (por exemplo, probabilidade de blocos órfãos, tamanho da mensagem, tamanho do bloco e limite de gás);

- Arquivo de atraso: inclui distribuições de probabilidade para tempo de validação das transações, tempo entre blocos, etc.;
- Arquivo de latência: contém as distribuições de probabilidade correspondentes à latência entre as possíveis localizações dos nós;
- Vazão de arquivos recebidos e enviados: configura as distribuições de probabilidade de vazão de recebimento e vazão de envio entre as possíveis localizações dos nós.

Esses parâmetros são necessários para os modelos de simulação, que são definidos usando o *Blockchain Modeling Framework*. O usuário precisa atribuir os arquivos ao *simulation world* e também especificar a hora de início e a duração da simulação. Este componente então retorna um *world* variável que será passado para os diferentes componentes, disponibilizando todos os atributos que caracterizam o *simulation world*.

Transaction and Node Factory: A fábrica de transações (*transaction factory*) é responsável por criar lotes de transações, que são modelados como fenômenos aleatórios. Essas transações são transmitidas por um nó aleatório em uma lista, quando a simulação está sendo executada. A fábrica de nós (*node factory*) cria nós que são usados durante a simulação. O usuário pode especificar a localização, número de nós e identificador.

Programmatic Interface: A interface de programação (programmatic interface), é a principal forma de interação disponível ao usuário. Por meio da linguagem Python e SimPy, o usuário pode escrever seus próprios modelos, usar os existentes para definir seus próprio sistema blockchain, ou modificar aspectos de modelos já implementados. Essa interface também é responsável por iniciar a simulação. Quando iniciado, o DESE consumirá os eventos e entidades antes de inicializar a simulação, para saber quais modelos serão usados.

Monitor and Reports: O objetivo do monitor é capturar, durante a simulação, as métricas: número de transações que cada nó transmite ou recebe; transações adicionadas à fila; blocos processados; tempo para propagar transações e blocos.

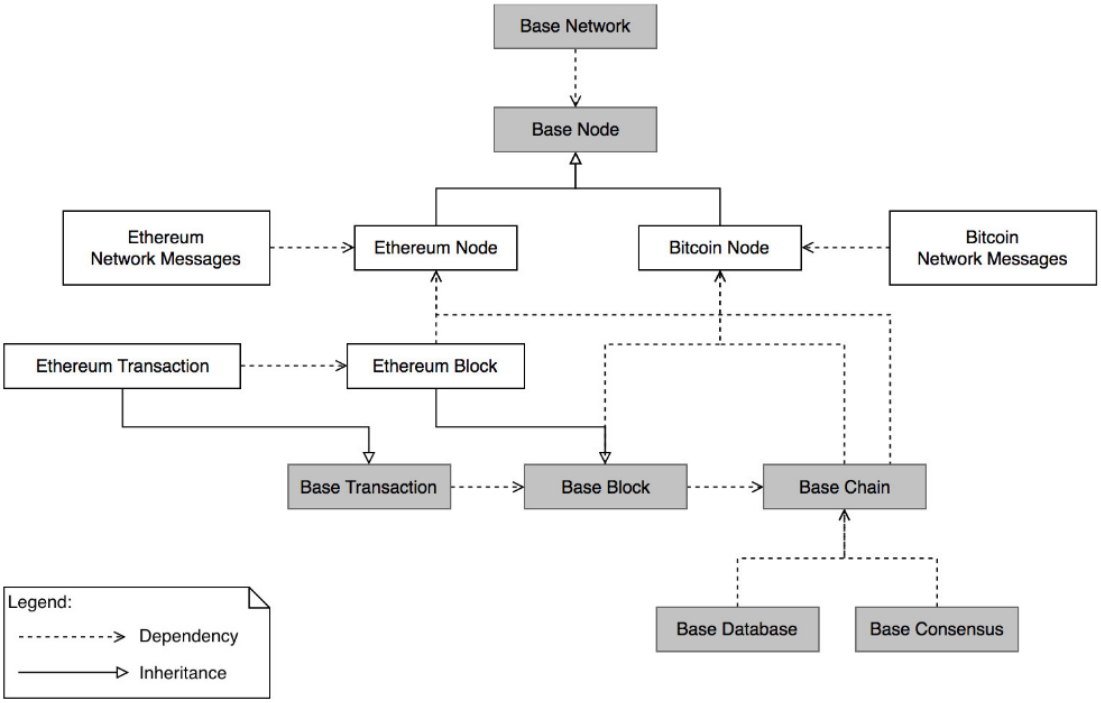
Blockchain Modelling Framework: Para poder simular blockchains arbitrários, deve-se considerar diferentes camadas presentes em um sistema blockchain:

- Nós: especifica as responsabilidades e como um nó opera ao fazer parte de uma determinada rede;
- Consenso: especifica os algoritmos e regras para um determinado protocolo de consenso;

- *Ledger:* define como o *ledger* é estruturado e armazenado;
- Transação e bloco: especifica como as informações são representadas e transmitidas;
- Rede: estabelece como os nós se comunicam uns com os outros;
- Criptografia: define quais funções criptográficas serão usadas e como.

As camadas de Nó, Transação, Bloco, Consenso e Rede estão disponíveis como classes que podem ser estendidas pelo usuário. Estes são então usados pelo DESE para criar entidades do sistema blockchain, que interagem dentro de eventos definidos nos modelos. Na Figura 5, é possível observar as classes disponíveis no *framework*, aquelas destacadas em cinza, são os modelos básicos possíveis de extensão e serão aprofundadas a seguir.

Figura 5 – Diagrama de classes do *framework* de modelagem



Fonte: Faria, Correia (2019)

Chain Model: Modelo responsável pela reprodução do comportamento de uma cadeia, implementado originalmente como uma funcionalidade abstrata que funciona para diferentes blockchains. A funcionalidade mais importante é adicionar um bloco à cadeia. O modelo primeiro verifica se o bloco está sendo adicionado à *head* (cabeça) da cadeia (o *hash* anterior do bloco aponta para a *head* da cadeia), se sim, simplesmente adiciona o bloco à cadeia. Caso contrário, o bloco é adicionado a uma *parent queue* (fila "pai") que será consultada toda vez que um novo bloco for adicionado, verificando

se o novo bloco aponta para um bloco na *parent queue*. De modo a resolver o problema de quando um nó recebe o um bloco "filho" antes do bloco "pai" devido a atrasos na rede. Quando um bloco não está sendo adicionado à *head*, mas o *hash* anterior aponta para um bloco antigo, o modelo cria uma bifurcação na cadeia instanciando uma cadeia secundária. Em seguida, verifica se o bloco deve ser a nova *head* calculando a dificuldade da cadeia, de acordo com as implementações da blockchain. Se esse for caso, a cadeia secundária é aceita como cadeia principal.

Consensus Model: Provê as regras aplicadas para validar blocos e transações. No modelo abstrato, optou-se por não realizar validações, por outro lado, o modelo adiciona um atraso que simula o processo de validação e assumimos que todos os blocos e transações são válidos. O modelo de consenso também define as regras para calcular a dificuldade de um novo bloco (para PoW). O modelo de consenso pode ser estendido e a dificuldade da equação alterada de acordo.

Network Model: O modelo de rede é responsável por conhecer o estado de cada nó durante a simulação, estabelecer os canais de conexão entre os nós e aplicar a latência de rede nas mensagens que estão sendo trocadas. O atraso de latência de rede aplicado depende da localização geográfica dos nós de destino e origem. A estrutura de simulação dá ao usuário a liberdade de escolher quais nós conectar, ele não implementa um protocolo de descoberta peer-to-peer (P2P) específico. É possível definir um modelo adicional para simular um determinado protocolo de descoberta P2P. O processo de mineração de um novo bloco, ou seja, a solução de um desafio criptográfico para obtenção de um novo bloco em PoW (em blockchains que utilizam essa abordagem de consenso), é em parte simulado pelo modelo de rede, pois conhece e pode interagir com qualquer nó. Assim, durante toda a simulação, a entidade da rede seleciona um nó para transmitir seu bloco candidato. O intervalo entre cada seleção, que chamamos de *heartbeat* da rede, corresponde ao tempo entre os blocos, dependendo do sistema blockchain que está sendo simulado. Cada nó tem uma taxa de *hash* correspondente. Quanto maior a taxa de *hash*, maior a probabilidade do nó ser escolhido. O modelo de rede também simula a ocorrência de blocos órfãos, ou seja, de blocos que precisam ser descartados devido a uma bifurcação na cadeia de blocos devido à natureza probabilística do PoW. O modelo de rede simula esse comportamento selecionando dois nós para transmitir seus blocos candidatos. Este evento ocorre apenas com uma probabilidade, definida na configuração.

Node Model: O modelo de nó é responsável por simular a funcionalidade de um nó operando em uma rede P2P. Quando uma simulação começa, um nó se conecta a uma lista de nós. Quando ocorre uma conexão, o nó de origem começa a escutar por comunicações de entrada para um nó de destino durante a simulação. Por outro lado, um nó pode enviar uma mensagem para um vizinho específico ou transmitir

uma mensagem para todos os vizinhos. No contexto do simulador, um evento está sendo agendado para ser processado por outra entidade, o nó de destino. O modelo de nó também é responsável por aplicar um atraso ao receber e enviar mensagens. Este atraso depende do tamanho da mensagem. O tamanho de cada mensagem é especificado dependendo do sistema blockchain sendo simulado e a taxa de transferência correspondente para onde o nó pretende enviar ou receber a mensagem. Para as primeiras conexões entre nós, é aplicado um atraso de latência três vezes correspondente ao *handshake* TCP. Depois disso, as seguintes comunicações se aplicam apenas a um atraso de latência, que é referenciado no modelo de rede.

Esses modelos básicos podem ser estendidos para suportar diferentes sistemas blockchain criando modelos de nível superior, como o Bitcoin e Ethereum. A seguir são descritas as alterações e extensões para os modelos Bitcoin e Ethereum (FARIA; CORREIA, 2019). Além disso, o modelo vCubeChain empregado em (FREITAS; RODRIGUES; Duarte Jr., 2023) também é apresentado.

Bitcoin: Utilizando o *Blockchain Modelling Framework*, modela-se a blockchain Bitcoin reutilizando os modelos base representados na Figura 5. Configura-se o *simulation world* com o tamanho máximo do bloco e a distribuição de probabilidade do número de transações por bloco, obtidos a partir dos dados dos últimos dois anos da rede pública do Bitcoin. O tamanho máximo de bloco na Bitcoin é de 1 MB.

O protocolo de rede Bitcoin define as mensagens trocadas entre os nós. Para cada mensagem, o nome, carga útil e tamanho são definidos. As seguintes mensagens estão presentes no modelo: *inv* para anunciar o conhecimento de novas transações ou blocos, *getdata* para recuperar o conteúdo de um bloco ou transação específica, *tx* para enviar uma única transação, *block* para enviar um corpo específico de um bloco; *headers* para enviar cabeçalhos de bloco, *getheaders* para solicitar cabeçalhos de bloco. O usuário pode modificar facilmente os tamanhos das mensagens no arquivo de configuração. Os tamanhos são retirados da documentação do Bitcoin.

O nó Bitcoin herda do modelo base, uma funcionalidade pré-definida para operar um nó em uma rede P2P. Dois tipos de nós são previstos: nó minerador e não-minerador. Um nó não-minerador tem apenas a função de esperar e validar novos blocos da rede ou validar e publicar novas transações. Já o nó minerador valida e coleta cada nova transição em uma fila de transições. A criação de um bloco candidato é o processo de coletar as transações pendentes e encaixá-las em um bloco. O nó somente publica seu bloco candidato na rede, quando selecionado pelo modelo base de rede. Esse processo simula a mineração de um novo bloco. O simulador não executa operações criptográficas ou validações; apenas aplica um atraso correspondente ao processo de validação em um sistema real, que é medido anteriormente.

Ethereum: A implementação do Ethereum é similar à Bitcoin. Os nós Ethereum também podem ser divididos em duas classes - nó minerador e nó não minerador -, fornecendo a mesma funcionalidade do modelo de node Bitcoin.

Nesse caso, a configuração do *simulation world* também tem presente *gas limit*, a quantidade máxima de *gas* que o proponente daquela transação está disposto a pagar, e o limite de *gas* por bloco. Por exemplo, se o ambiente é configurado para ter um limite de *gas* por bloco de 10.000, e um limite de *gas* de transação de 1.000, será possível encaixar 10 transações por bloco. O gás é consumido pela execução das instruções da máquina virtual Ethereum (EVM). O valor do gás consumido por instrução varia de 1 a 32.000.

As mensagens do protocolo Ethereum Network são definidas no modelo da seguinte forma: *Status* informa um nó de seu estado atual, enviado após o *handshake* inicial e antes de qualquer outra mensagem, *NewBlockHashes* anuncia um ou mais novos blocos que apareceram na rede, *Transactions* envia uma ou mais transações, *BlockBodies* envia corpos de bloco, *GetBlockBodies* são usados para recuperar corpos de blocos específicos usando *hashes* de bloco, *BlockHeaders* envia cabeçalhos de bloco para um nó, *GetBlockHeaders* solicita cabeçalhos de bloco. Os tamanhos das mensagens foram retirados da documentação Ethereum.

O modelo de transação Ethereum estende o modelo de transação base apenas adicionando novos atributos, como o preço do *gas* e o *gas limit*. O produto desses dois atributos é usado para calcular a taxa de transação. O modelo de bloco Ethereum estende o modelo de bloco base apenas adicionando estes mesmos atributos.

vCubeChain: A vCubeChain (FREITAS; RODRIGUES; Duarte Jr., 2023) utiliza as mesmas configurações do Bitcoin. Porém as transações são enviadas apenas para o líder (processo com o ID mais alto), o líder é responsável pela propagação dos blocos de transação para o restante da rede, seguindo a hierarquia fornecida pelo vCube.

A implementação da vCubeChain utiliza a topologia vCube, na qual os nós são interligados seguindo as regras próprias do algoritmo. Ao contrário das implementações do Bitcoin e Ethereum, onde a topologia de comunicação totalmente conectada possibilita uma comunicação direta entre todos os nós, sem intermediários. É calculado um tempo de processamento para cada mensagem enviada e recebida na rede, que depende do tamanho da mensagem. O processo de conexão, que ocorre apenas no primeiro contato entre dois nós, simula o *handshake* do TCP com uma tripla latência.

No modelo implementado pelo simulador, tanto o Bitcoin quanto o Ethereum utilizam uma estratégia de disseminação para o cabeçalho e o texto da mensagem. Ou seja, para cada mensagem contendo o cabeçalho da transação, é enviada uma solicitação

do corpo da transação, que gera uma nova mensagem com o conteúdo da transação. O mesmo vale para os blocos. Para fins de comparação, a VCubeChain manteve esse comportamento.

2.8 Trabalhos Relacionados

O algoritmo de diagnóstico distribuído vCube já foi utilizado para construir um sistema de difusão confiável hierárquica (RODRIGUES et al., 2017). Nele, as mensagens são propagadas por árvores geradoras que se adaptam dinamicamente à medida que falhas são detectadas pelo mecanismo de monitoramento. Em comparação com uma solução de força-bruta e com uma solução de árvore não autonômica, a solução do vCube foi eficiente e escalável.

O Paxos é um algoritmo de consenso tolerante a falhas para sistemas assíncronos sujeitos a falhas por parada e recuperação (crash-recovery) e é um dos algoritmos de consenso mais importantes, proposto originalmente para a replicação máquina de estado. Variações do algoritmo, com destaque para o *Ring* Paxos, foram propostas com o intuito de melhorar o desempenho do sistema. A topologia virtual para sistemas distribuídos conhecida como vCube foi utilizada como um algoritmo de consenso escalável que implementa uma instância do Paxos. O algoritmo foi implementado através de simulação e foram obtidos resultados positivos para a quantidade de mensagens trocadas durante a execução do consenso, quando comparado a uma versão do Paxos em anel inspirada no *Ring* Paxos (TERRA, 2020).

Considerando que o consenso é essencial para o funcionamento da blockchain e para a criação de um sistema consistente, no qual todos os nós concordem com a ordem dos blocos e sobre os seus conteúdos. Resolvê-lo no contexto da blockchain, onde o conjunto de participantes não é bem conhecido e estão sujeitos a falhas bizantinas, é muito complexo (MIERS et al., 2019).

De modo a solucioná-lo a blockchain pode implementar uma máquina de estados replicada (RME) para a manutenção consistente do livro razão distribuído em cada um dos servidores que compõem a rede P2P. Nesse caso, os blocos de transações são incorporados ao livro razão seguindo uma ordem total, por meio de um protocolo de difusão atômica (GREVE et al., 2018). O protocolo de difusão atômica e um dos protocolos de consenso que garantem que todas as mensagens do sistema sempre sejam entregues na ordem determinada. Apesar disso, o consenso na blockchain é complexo e diversas abordagens são utilizadas e estudadas, como *Proof-of-Work* e *Proof-of-Stake*, também temos protocolos de blockchain específicos para ambientes *permissioned*, no qual o livro-razão distribuído não é acessível publicamente e só pode ser acessado por usuários com permissões (CACHIN; VUKOLIĆ, 2017).

Os problemas de escalabilidade da blockchain podem ser classificados em três categorias: taxa de transferência, armazenamento e rede (XIE et al., 2019). A taxa de transferência dos sistemas blockchain está relacionada ao número de transações em cada bloco e ao tempo de intervalo do bloco. Tomando o blockchain do Bitcoin como exemplo, o tempo de intervalo do bloco é de aproximadamente 10 minutos, e o número de transações em cada bloco é restrito pelo tamanho do bloco, que é 1MB. O Bitcoin só pode lidar com 7 transações por segundo em média. O que não pode atender aos requisitos dos atuais cenários de pagamento digital, considerando por exemplo, o sistema VISA que pode lidar em média com 2.000 transações por segundo. Portanto, não pode ser utilizado em outras aplicações, como armazenamento distribuído e serviço de crédito. Além disso, diferentes sistemas de blockchain carregam diferentes regras de negócios e requisitos, então a escalabilidade é a questão central do desenvolvimento atual do blockchain.

O problema do armazenamento surge ao aplicar o blockchain em ambientes de negócios reais, uma enorme quantidade de dados gerados por vários dispositivos é processada. Nos sistemas tradicionais de blockchain, cada nó deve processar e armazenar a transação completa de volta ao bloco de origem. Assim, não é possível aplicar diretamente a ambientes reais onde os nós possuem recursos limitados de armazenamento e computação. Portanto, é necessário estudar como armazenar dados em nós de cadeia de blocos com recursos limitados de forma eficaz.

A rede é o terceiro fator que afeta a escalabilidade dos sistemas blockchain. Por um lado, a rede blockchain tradicional é um meio de transmissão, no qual cada nó retransmite todas as transações. Este modo de transmissão de dados não pode ser dimensionado para lidar com um grande número de transações devido à necessidade de recursos de largura de banda da rede. Por outro lado, em sistemas blockchain tradicionais, cada transação é transmitida duas vezes para todos os nós: quando uma transação é gerada, ela é transmitida primeiro para todos os nós e quando um bloco contendo a transação é minerado, é transmitido para todos os nós pela segunda vez. Essa abordagem não apenas consome muitos recursos de rede, mas também aumenta o atraso de propagação do bloco.

Em (HERRERA-JOANCOMARTÍ; PÉREZ-SOLÀ, 2016), os autores nos apresentam uma descrição abrangente das soluções de escalabilidade mais relevantes propostas para a rede Bitcoin e descrevem seu impacto na privacidade dos usuários com base nas propostas publicadas até o momento de escrita. É possível identificar que algumas das ideias de solução propõem desacoplar transações padrão do núcleo blockchain e gerenciá-las por meio de uma rede de pagamentos paralela, relegando o uso do blockchain Bitcoin apenas para transações que consolidam vários desses movimentos fora da cadeia. Tais mecanismos geram novos atores no cenário de pagamento Bitcoin, os provedores de serviços de pagamento, e surgem novas questões de privacidade em relação aos usuários Bitcoin.

Os fatores que influenciam a escalabilidade de uma Blockchain estão descritos no

artigo em (EKLUND; BECK, 2019). No qual é possível concluir que o desempenho de um sistema, quando medido em termos de tempo de resposta de consenso (latência da rede blockchain ou tempo para convergência/acordo), número de transações por segundo ou taxa de transferência, e recursos de computação (e energia) consumidos, podem ser entendidos considerando as dimensões do projeto de um sistema blockchain. Desde que se saiba o tipo de sistema blockchain necessário que determina; a complexidade do protocolo de consenso utilizado; a topografia do fluxo de tráfego na rede; o desempenho e complexidade da linguagem específica de domínio que implementa contratos inteligentes; e pelo crescimento previsto em tamanho e complexidade do próprio *ledger*.

Kim, Kwon e Cho (2018) fazem uma análise dos métodos que se propõem a solucionar o problema da escalabilidade, dividindo-os em cinco categorias: 1) *On-chain*, que se refere ao aumento da escalabilidade modificando apenas elementos dentro de um blockchain; 2) *Off-chain* cuja solução é processar as transações fora do blockchain. Isso também é chamado de solução de canal de estado, porque mantém o estado da cadeia principal e aplica o último estado que foi processado no outro canal; 3) *Side-chain*, que a abordagem é trocar recursos de diferentes blockchains, seu objetivo é trazer a função de outro blockchain para o blockchain atual; 4) *Child-chain* tem uma estrutura pai-filho, processa as transações na cadeia filho e registra os resultados na cadeia pai; e 5) *Inter-chain* que é uma maneira de permitir a comunicação entre os vários blockchains, agrupando várias. O *Inter-chain* é a tecnologia de infraestrutura para implementação da *side-chain*.

Outros trabalhos foram realizados anteriormente que utilizam o Blocksim como base para construção de simuladores para usos específicos. Um deles é SIMBA (FATTAHI; MAKANJU; FARD, 2020), que estende um simulador existente adicionando o recurso de uma árvore *Merkle* aos nós de blockchain para melhorar a eficiência e permitir que avaliações mais realistas sejam realizadas. Além dele, temos Talaria (XING et al., 2021), um novo simulador de blockchain estendido a partir do Blocksim com o objetivo de incluir blockchains permissionadas para medir, avaliar e adaptar uma variedade de soluções de blockchain para operações comerciais, além de suportar vários protocolos e casos de uso, principalmente para gerenciamento de cadeia de suprimentos. O Talaria é capaz de simular diferentes tipos de autoridades maliciosas e uma carga de transação diária variável em cada nó. Um trabalho recente e relevante é descrito em (POLGE et al., 2021), que constrói um novo simulador Blockperf a partir da extensão do Blocksim, com o propósito de superar as limitações anteriores do simulador, como a falta de extensibilidade e falha em cobrir todos os aspectos/métricas que sustentam um sistema blockchain. Ambos os simuladores (Blocksim e Blockperf) são comparados utilizando uma ferramenta de *benchmarking* para uma simulação com Bitcoin (único modelo avaliado pelos autores) e os resultados demonstram que o Blockperf provê resultados mais realísticos que aqueles no Blocksim.

Assim, identifica-se que utilizando o sistema de testagem e obtenção de informações de diagnóstico do vCube, é possível replicar o processamento de transações descentralizado da blockchain, sendo necessário adequar a topologia para que contenha as principais propriedades: Descentralização, Disponibilidade e Integridade, Transparência e Auditabilidade, Imutabilidade e Irrefutabilidade, Privacidade e Anonimidade, Desintermediação e Cooperação e Incentivos. E principalmente, implementando a estrutura de dados do livro razão distribuído, que utiliza de resumos criptográficos (funções *hash*), assinaturas digitais e infraestrutura de chave pública. Com isso, utilizou-se da propriedade intrínseca de escalabilidade do vCube para criar uma blockchain totalmente escalável a partir da topologia de hipercubo.

3 Solução Proposta

Este capítulo apresenta aversão do detector de falhas vCube em ambientes assíncronos e a avaliação da vCubeChain considerando o suporte a falhas que incorporado ao Blocksim. Após o mecanismo de falhas estar implementado e validado, foram executados cenários de teste para cada uma das soluções: Bitcoin, Ethereum, vCubeChain - já apresentadas na literatura em experimentos sem falha. Além dessas, foi implementada a HyperLegder Fabric, blockchain permissionada, de modo a comparar e avaliar seus resultados com as soluções anteriores.

3.1 vCube em Sistemas Assíncronos

O vCube (DUARTE JR.; BONA; RUOSO, 2014) foi inicialmente apresentado como uma ferramenta no contexto de diagnóstico hierárquico adaptativo distribuído em nível do sistema (Hi-ADSD). O diagnóstico distribuído assume implicitamente um sistema síncrono, e é assim que o vCube foi originalmente especificado.

O diagnóstico em nível de sistema foi proposto como uma abordagem baseada em teste para monitorar e identificar componentes com falha. Os detectores de falhas não confiáveis surgiram como uma abstração para, a partir do monitoramento de falhas de processos, permitir a execução de consenso em sistemas assíncronos sujeitos a falhas *crash*. Em Jr. et al. (2023) apresenta-se um novo modelo de diagnóstico que permite a especificação de detectores de falhas escaláveis, com o propósito de unificar o diagnóstico e a detecção de falhas.

Detectores de falha foram propostos para sistemas assíncronos, de modo a classificar cada processo em um de dois estados: correto ou suspeito, supondo falhas *crash*. Como não há garantias de tempo, não apenas os processos falhos são classificados como suspeitos, mas também os processos suspeitos incorretamente de ter falhado. Embora tenha havido abstrações construídas sobre o vCube que assumem o modelo assíncrono (por exemplo, (JEANNEAU et al., 2017)), estes lidam com incertezas de tempo em um nível superior.

Neste trabalho, foi proposto e publicado em Stein, Rodrigues e Jr. (2023) uma solução que considera a execução da detecção de falhas do vCube em um ambiente assíncrono. Para lidar com as falsas suspeitas, que podem deixar a visão que os processos corretos têm dos outros processos em um estado temporariamente inconsistente, quando um processo detecta que foi suspeito, ele deixa o sistema, em definitivo.

O Algoritmo 1 apresenta o pseudocódigo do vCube adaptado de DUARTE JR., Bona e Ruoso (2014) para uso em um sistema assíncrono. O processo i mantém registros

de *timestamps* para o estado de todos os outros processos no vetor $STATE_i[]$. O algoritmo prevê que no caso de uma falsa suspeita, ou seja, se um processo i foi suspeitado erroneamente por um processo j, o processo i, ao receber esta informação, para de executar e deixa o sistema (linha 8). O resultado é um detector de falha $\Diamond P$ que é inevitavelmente perfeito (CHANDRA; TOUEG, 1996) se os processos corretos permanecerem em um único componente conectado, ou seja, se não houver partições no grafo de teste.

Algoritmo 1 Detector de falhas vCube assíncrono no processo i

1: $STATE_i[j] \leftarrow correct$, $\forall j = 0, .., n-1$
2: **repeat**
3: **for** $s \leftarrow 1$ **to** $\log_2(n)$ **do**
4: **for all** $j \in c_{i,s} \mid i$ é o primeiro processo correto em $\in c_{j,s}$ **do**
5: TEST(j)
6: **if** j is tested *correct* **then** i obtains $STATE_j[]$ and j obtains $STATE_i[]$
7: **if** $STATE_j[i] = suspect$ **then** ▷ i foi suspeito por j (falsa suspeita)
8: Terminar o processo e deixar o sistema
9: **else**
10: Atualiza $STATE_i[]$ com as novas informações em $STATE_j[]$
11: **else**
12: **if** $STATE_i[j] = correct$ **then** $STATE_i[j] \leftarrow suspect$
13: **if** $\forall j = 0, .., n-1, j \neq i$: $STATE_i[j] = suspect$ **then**
14: Terminar o processo e deixar o sistema
15: Aguarda até a próxima rodada de testes
16: **until** forever

O Neko[1] (Urban; Defago; Schiper, 2001) *framework* Java que permite a simulação de algoritmos distribuídos. Inicialmente, ele foi utilizado para implementação desta versão assíncrona do vCube e execução de testes.

O vCube foi comparado com uma solução clássica todos-para-todos (ALL), na qual cada processo correto envia um teste individual para todos os demais processos considerados corretos. Para uma comparação justa, em caso de falsa suspeita, o processo suspeito também deixa o sistema.

Todos os testes foram realizados em $\log_2^2(n)$ rodadas, que é latência máxima de diagnóstico do vCube no pior caso, de modo a garantir que todos os processos serão testados por todos os outros.

Por utilizar uma estratégia de testes hierárquica, o vCube possui uma maior latência de propagação de eventos no sistema (diagnóstico). Porém, em termos de tempo de execução e número de mensagens, os resultados de simulação mostram especialmente a redução significativa no número de mensagens do vCube, o que também reduz o tempo de execução das rodadas de teste.

[1] Código-fonte disponível em: <https://github.com/arluiz/neko>

3.2 vCubeChain em Sistemas Assíncronos

O trabalho de Freitas, Rodrigues e Duarte Jr. (2023) apresenta a blockchain permissionada vCubeChain, escalável por definição, baseada na topologia distribuída dinâmica vCube. A vCubeChain elege um líder, que utiliza uma estratégia autonômica de difusão confiável para disseminar os blocos na rede. Cada bloco consiste de múltiplas transações. Em caso de falsas suspeitas pelo detector de falhas, vários líderes concorrentes podem ser eleitos. Entretanto, prova-se que a vCubeChain sempre retorna a um estado consistente. Um conjunto de testes foi exposto, que demonstra o desempenho da vCubeChain em comparação as alternativas Bitcoin e Ethereum, principalmente se concentrando na escalabilidade da solução em termos do tempo de execução e no número de mensagens. Além disso, é apresentada a especificação e provas de corretude da blockchain.

O presente trabalho estende os cenários de testes abordados e evidenciar o comportamento da vCubeChain quando os nós da rede estão sujeitos a falhas. Inicialmente, as funções e estruturas necessárias para a simulação de falhas foram adicionadas ao simulador dentro de cada um dos modelos. Após a implementação inicial, as soluções presentes foram executadas e avaliadas. Nessa fase, foi observado o comportamento do mecanismo, sua interação com os algoritmos propostos originais e as mudanças necessárias nos procedimentos já estabelecidos para lidar com as novas funções.

Toda a construção iniciou-se a partir dos modelos existentes e fornecidos pelos autores em Freitas, Rodrigues e Duarte Jr. (2023), foram adicionadas funcionalidades específicas de acordo com o comportamento das blockchains. Tanto para Bitcoin, Ethereum, vCubeChain e Fabric - os nós selecionados para falha são removidos da rede e os outros cessam sua comunicação com o nó falho. Como as blockchains tem características próprias, o encerramento do funcionamento e remoção do nó foi feito de maneiras distintas.

O mecanismo de falhas escolhido para implementação foi a injeção de falhas - o usuário consegue configurar um tempo específico e qual nó ele quer que falhe em se comunicar na rede. Dessa forma, é possível garantir cenários específicos, com um número assegurado de nós falhos e com as características objetivadas. Em todas as blockchains, o nó só é "descoberto" como falho quando é sua vez de ser o líder da rodada do consenso. Além disso, no momento em que o nó é caracterizado como falho, já é retirado da rede.

Ao implementar o mecanismo de simulação de falhas ao simulador BlockSim, além das informações de escalabilidade como tempo de execução e número total de mensagens enviadas, será possível adquirir dados como o atraso causado por uma falha de líder na rede, latência na eleição de um novo líder, o número de transações de cada nó, as transações adicionadas na fila e tempo de propagação de blocos, entre outras informações pertinentes a compreensão dos efeitos de falhas na solução proposta.

Utilizando o componente principal do simulador Blocksim - o *Discrete Event*

Simulation Engine, descrito na Seção 2.7.1, implementou-se uma falha na rede como um evento. Dessa forma, o evento de falha já está agendado antes mesmo de iniciar a simulação. As modificações mais significativas foram feitas no modelo de Rede, considerando que a maior influência de uma falha de nó na rede, se daria quando este fosse o líder do consenso - ou seja, o nó escolhido para transmitir seu bloco.

Para implementar a função que agenda uma falha - *fault*, foram utilizado funcionalidades já presentes nos modelos dos nós das blockchains, como o *disconnect* - responsável por retirar o nó da lista de seções ativas da rede. No Bitcoin e no Ethereum - no momento em que o nó falha, ativa-se a função *disconnect* e o nó é retirado das rede, a função de comunicação *broadcast* só envia mensagens para nós que estão com seções ativas, assim em casos onde o nó que falha é apenas algum outro membro da rede, as mensagens já deixam de ser trocadas com o nó falho - sem necessidade de criar um novo mecanismo. Na vCubeChain, além de retirar das seções ativas, o nó falho também não faz mais parte da estrutura de clusters, de forma análoga, no Fabric, o nó é retirado da lista de vizinhos fornecida pelo algoritmo de comunicação Gossip – isso é realizado por meio da funcionalidade já existente em ambos os modelos de nós *notify_crash*. No fragmento 3.1, é possível identificar a função de agendamento de falhas, com seus devidos parâmetros.

Listing 3.1 – Fragmento do código da função de agendamento de falhas

```
def fault(delay, msg, address, n):
                yield world._env.timeout(delay)
                print(f"time {delay}: {msg}")
                # kill node by id
                # address is the id of the node in the list
                if world.blockchain not in ['vcubechain', 'fabric']:
                # Crash selected node, remove from the list of
                # current active nodes
                    nodes_list[address].crashed = True
                    nodes_list[address].disconnect(nodes_list)
                elif world.blockchain == 'vcubechain':
                    # Crash selected node, remove from
                    #the list of miners
                    nodes_list[address].crashed = True
                    nodes_list[address].is_mining = False
                    nodes_list[address].hashrate = 0
                    #remove its connections
                    nodes_list[address].disconnect(nodes_list)
                    for node in nodes_list:
                        #suspect fault and remove the address
```

```
                # from correct node list
                node.vcube.suspect(n)
                #find new leader
                node.check_leader()
            nodes_list.pop(address)
        else:
            #crash node
            nodes_list[address].crashed = True
            nodes_list[address].is_mining = False
            nodes_list[address].hashrate = 0
            nodes_list[address].disconnect(nodes_list)
            for node in nodes_list:
                #suspect fault and remove the address
                #from correct node list
                node.gossip.suspect(n)
                #find new leader
                node.check_leader()
            nodes_list.pop(address)
```

A partir disso, é possível identificar que a principal implicação que uma falha teria no sistema, seria quando o nó selecionado para transmitir seu bloco na rede falhasse. No Bitcoin, o nó escolhido é um dos mineradores com maior *hashrate* - o equivalente no modelo real Bitcoin, de poder computacional suficiente para liderar aquela rodada do consenso (*proof-of-work*). No Ethereum, a funcionalidade é similar, utilizando o *gas* do nó minerador como avaliação da probabilidade daquele nó ser o escolhido como líder da transmissão do bloco.

Na vCubeChain, a transmissão do bloco é sempre feita pelo líder – no algoritmo de consenso fornecido, é o processo de maior identificador na rede. Portanto, quando é o líder da rodada que é o nó escolhido para falhar, é necessário iniciar uma nova rodada de eleição de líder. Nesse caso, o nó com o segundo maior identificador se torna o novo líder e assim por diante, essa informação é propagada para todos os outros nós e somente com sua confirmação, retorna-se para a transmissão do bloco.

O Hyperledger Fabric foi implementado no simulador com as mesmas configurações do Bitcoin, apresentadas na seção 2.7.1. Porém as transações são enviadas apenas para o líder (processo com o ID mais alto), o líder é responsável pela propagação dos blocos de transação para o restante da rede – similar a vCubeChain. Os nós são totalmente conectados, ou seja, um nó pode se comunicar com todos os outros. Entretanto, o método para consenso utilizado é o PBFT que utiliza do algoritmo de difusão confiável Gossip.

Neste algoritmo, um nó transmite a informação para um subconjunto dos nós, que

por sua vez, irão transmitir para outro subconjunto e assim, depois de um tempo, todos os nós estarão com a informação. Neste caso, o líder transmite as transações para o conjunto de "vizinhos" que irão fazer o mesmo para seus "vizinhos". Pode-se identificar que nesse processo o número de mensagens trocadas será menor do que nas implementações do Bitcoin e Ethereum. O processo de conexão, que ocorre apenas no primeiro contato entre dois nós, simula o *handshake* do TCP com uma tripla latência, calculado a partir de um tempo de processamento para cada mensagem enviada e recebida na rede, dependente do tamanho da mensagem.

O Hyperledger Fabric manteve a mesma estratégia de disseminação para o cabeçalho e o texto da mensagem utilizada para o Bitcoin, Ethereum e vCubeChain na qual para cada mensagem recebida que contém o cabeçalho da transação, é enviada uma solicitação do corpo da transação, gerando uma nova mensagem com o conteúdo da transação. O mesmo vale para as mensagens dos blocos.

Este trabalho teve como objetivo examinar o comportamento da vCubeChain com nós falhos, estabelecendo por meio da execução dos testes, se as consequências das falhas podem ser contornadas pela solução apresentada. Para tal, foi construído este procedimento para simulação de falhas por meio da extensão do simulador BlockSim adicionando o suporte a estes cenários. A versão atualizada com as extensões está disponível no Github[2].

[2] <https://github.com/gabrisingr/FD-blocksim-fault>

4 Resultados e Discussões

De modo a avaliar tanto a alteração no Blocksim para o suporte a simulação de falhas, assim como realizar a avaliação da vCubeChain nesses cenários, foram conduzidos experimentos em um notebook Dell Inc. Inspiron 15 3530 - usando o sistema operacional Ubuntu 22.04.4 LTS com o processador Intel 13th Gen Intel® Core™ i7-1355U × 12, com 10 núcleos, 16GiB de memória RAM, equipado com HD de 1.0 TB.

As configurações do Blocksim foram baseadas naquelas mencionadas e utilizadas em (FREITAS; RODRIGUES; Duarte Jr., 2023), sendo as seguintes:

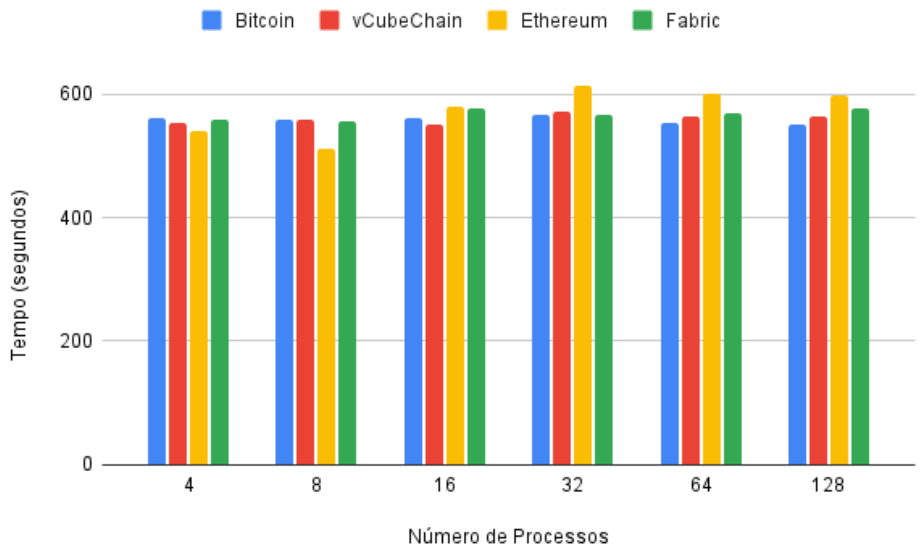
- Para Ethereum, foi usado o valor padrão de *gas limit* de 21.000 e *block gas limit* de 2,1 milhões;
- Para Bitcoin, vCubeChain e Fabric, o limite do tamanho de bloco foi de 2MB e para o número de transações por bloco, foi calculado pelos autores por meio de distribuição de probabilidade utilizando dados reais da rede Bitcoin de número médio de transações por bloco.
- Os nós foram distribuídos entre três localizações, cada uma podendo conter mais do que um nó, da maneira que a Tabela 3 demonstra.
- A latência de comunicação entre os nós está descrita na Figura 2. A distribuição normal (média e desvio padrão) foi usada para nós em locais diferentes e a distribuição Gama inversa foi usada para nós no mesmo local. Os valores foram calculados pelos autores com base nos dados de RTT reais (ping) obtidos na rede da RNP (Rede Nacional de Pesquisa).

4.1 Cenários Simulados

Foram realizados experimentos com 4, 8, 16, 32, 64 e 128 processos. Cada experimento foi simulado 30 vezes, com duração de 10000 segundos. Essa duração foi escolhida para garantir que pelo menos 1 bloco fosse construído por todas as soluções. Em cada rodada de teste, foram simuladas mil transações, divididas em mil rodadas de uma transação a cada 15 segundos. As transações foram distribuídas usando a função *transaction factory* do simulador.

O limite máximo de processos é devido ao Ethereum – seu tempo de execução e número de mensagens trocadas é muito superior aos demais, o que ocasiona que a simulação não consiga ser executada no equipamento utilizado – também possui uma implementação

Figura 6 – Tempo para validação de todas as transações na rede.



muito verbosa, seus arquivos de log gerados pelo programa chegam a mais de 1GB de tamanho. Além disso, o tempo de execução total do Ethereum dependendo do número de processos pode chegar a mais de 10 horas, enquanto as outras soluções são executadas em minutos.

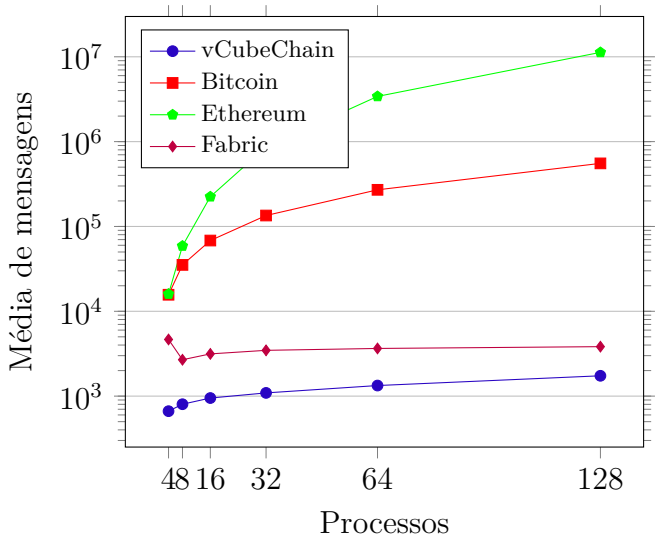
O primeiro cenário de experimentação foi o cenário normal, sem nenhuma falha, das blockchains Bitcoin, Ethereum, vCubeChain e Hyperledger Fabric. O segundo cenário de experimentação – injetou-se uma falha no nó líder (no caso do Ethereum e Bitcoin, o nó com maior *hashrate*) no tempo após 500 segundos da simulação. Esse tempo foi escolhido para que a maioria das soluções já tivessem realizado todas as transmissões de transações e fosse iniciar uma rodada do consenso para transmissão do bloco. De modo a testar o cenário limite das soluções, decidiu-se falhar o máximo de nós. Neste cenário, foi necessário injetar falhas em (n/2 - 1) nós, mantendo pelo menos um nó minerador que possa liderar a rodada do consenso.

4.2 Resultados da Simulação

O primeiro cenário simulado foi aquele sem nenhuma falha. Uma execução normal, sem nenhuma alteração dos parâmetros iniciais. Na Figura 6 verifica-se o tempo médio necessário para validar todas as transações, ou seja, o tempo para construção de um bloco - em de cada uma das blockchains, em cenários onde nenhum nó falhou. É possível observar que o tempo de validação para todos os algoritmos é muito similar.

A Figura 7 apresenta a média do número total de mensagens enviadas, entre as 30 execuções realizadas quando nenhum dos processos falha durante a execução. Os resultados estão representados em escala logarítimica para facilitar a visualização.

Figura 7 – Número médio de mensagens.



Neste cenário é possível observar o comportamento esperado da vCubeChain em comparação às demais blockchains. Identifica-se que mesmo com o crescimento do número de processos, o total de mensagens trocadas entre os processos permanece muito inferior aos outros. A Tabela 4 apresenta estes mesmos resultados arredondados com 2 casas decimais, para visualizar as diferenças entre os valores.

Tabela 4 – Número médio de mensagens no cenário sem falhas.

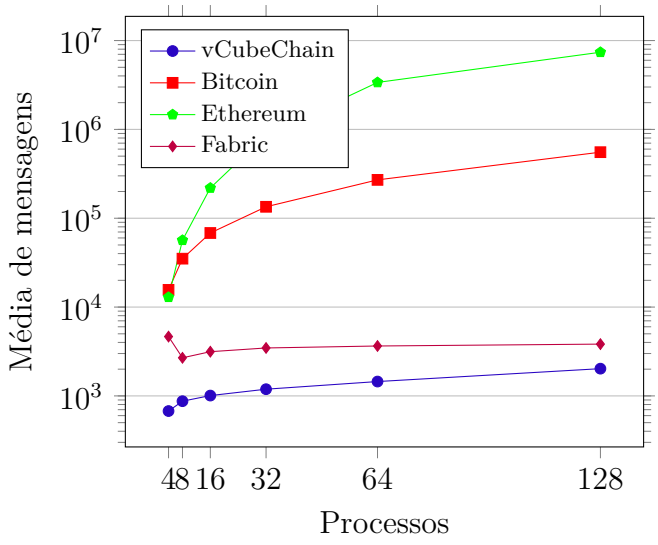
Processos	**Bitcoin**	**Ethereum**	**vCubeChain**	**Fabric**
4	15.670,65	15.988,04	664,45	4.652,93
8	35.235,07	58.904,52	802,52	2.684,22
16	68.239,90	224.305,17	949,31	3.146,45
32	134.581,04	864.490,59	1.091,10	3.465,50
64	270.335,65	3.412.752,00	1.332,59	3.647,00
128	554.133,52	11.276.896,50	1.736,38	3.829,12

Identifica-se que o número de mensagens trocadas no Ethereum, Bitcoin são muito maiores que nas blockchains permissionadas vCubeChain e Hyperledger Fabric. Isso ocorre devido aos protocolos de comunicação utilizados por cada uma das soluções. Principalmente a vCubeChain, que tem a menor média entre as soluções testadas. A vCubeChain utiliza da topologia de clusters fornecida pelo vCube, na qual cada processo troca mensagens com um cluster de tamanho progressivamente maior. Esse algoritmo gera árvores de difusão de mensagens, se comunicando com $\log_2 N$ vizinhos, enquanto na estratégia de difusão *one-for-all* (um-para-todos) utilizada pelo Ethereum e Bitcoin, são trocadas $(N^2 - N)/2$ mensagens. O algoritmo de difusão confiável utilizado pelo Fabric, é o Gossip, as mensagens são transmitidas pelo líder para seus k vizinhos escolhidos aleatoriamente que consequentemente encaminham para seus k vizinhos aleatórios, até que todos estejam cientes da informação. Em todas as execuções - de ambos os cenários - o número de

vizinhos foi $k = 3$. No melhor caso desse algoritmo, o número de mensagens trocadas é N.

O segundo cenário no qual foram realizados os experimentos foi a adição de uma falha no processo de maior id após 500 segundos do ínicio da simulação – a partir do qual já teria sido feita a propagação de um bloco na rede. Com esse cenário, visou-se observar o comportamento referente a troca de mensagens e recuperação de um nó falho dentro das blockchains estudadas. Na vCubeChain e no Fabric, o processo de maior identificador é o líder que comanda o consenso entre as transações a serem aceitas. Já no Bitcoin e no Ethereum, como trabalham com modelos probabilísticos, onde o nó que irá enviar a transação depende de seu poder computacional, escolheu-se forçar a falha de um dos mineradores que apresentavam – de acordo com as informações de configuração da rede – ter maior "chance" de serem os escolhidos. A Figura 8 apresenta o total de mensagens trocadas pelas soluções em cenários onde o processo responsável pelo consenso naquela rodada foi o processo falho.

Figura 8 – Número médio de mensagens trocadas por cada blockchain simulada em um cenário com um processo falho.



Na vCubeChain, sempre que o processo falho é o líder, deve-se eleger um novo processo (escolhe o próximo processo com maior identificador) e propaga-se essa informação na rede. Com esse passo extra, gera-se um maior número de mensagens do que em um cenário onde não ocorreu nenhuma falha, ou seja, nenhuma mudança de líder. O mesmo é verdade para o Fabric, ao falhar o líder, elege-se um novo e propaga essa informação na rede. No caso do Bitcoin e do Ethereum, não existe a propagação de nenhuma informação na rede, o nó é identificado que está falho quando será o responsável pela propagação do bloco. Nesse caso, remove-se o nó da rede e outro é eleito. Na Tabela 4.2 é apresentado o valor médio das mensagens no cenários com falhas para cada uma das soluções testadas, com o número de processos variando de 4 até 128. Os valores foram arredondados para duas casas decimais, para melhor visualização. A partir dessa tabela, é possível identificar que

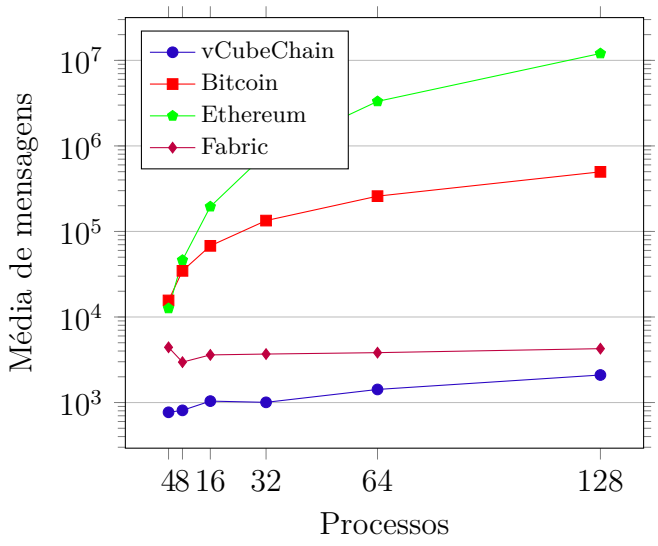
a vCubeChain sempre mantém o número de mensagens muito inferior as outras soluções testadas.

Tabela 5 – Número médio de mensagens no cenário com falhas.

Processos	Bitcoin	Ethereum	vCubeChain	Fabric
4	15.611,93	12.910,65	675,35	4.231,33
8	35.068,72	56.541,00	871,86	3.225,00
16	68.202,00	219.142,62	1.008,76	3.469,50
32	134.548,45	864.042,90	1.189,86	3.865,00
64	270.297,07	3.378.493,50	1.450,76	3.992,22
128	554.043,10	7.389.275,79	2.026,24	4.028,00

A Figura 4.2 apresenta a média do número total de mensagens enviadas no terceiro cenário apresentado, aquele cujo propósito é apresentar o limite de falhas na execução das soluções. O comportamento das soluções segue similar ao apresentado anteriormente. Neste cenário, falhou-se nós aleatórios, em momentos aleatórios durante a execução dos testes.

Figura 9 – Número médio de mensagens trocadas por cada blockchain simulada no cenário de limite de falhas.



É possível observar na tabela 6 os valores médios de mensagens de cada uma das blockchains. A média do número de mensagens apresentou-se menor na maioria dos casos, entretanto, em algumas situações, aumentou-se o número de mensagens. Pode-se entender que isso acontece devido a escolha aleatória dos nós a falharem, podendo-ser ou não, nós mineradores. Assim, quando o nó escolhido é o nó líder da rodada, são necessárias mais mensagens trocadas entre os processos para a nova rodada de escolha do líder.

Tabela 6 – Número médio de mensagens no cenário com falhas.

Processos	**Bitcoin**	**Ethereum**	**vCubeChain**	**Fabric**
4	15.429,56	12.595,34	769,07	4.421,30
8	34.632,77	46.244,57	810,48	2.967,96
16	67.903,12	195.100,96	1.039,45	3.604,00
32	133.980,05	815.360,99	1.005,22	3.688,34
64	258.642,21	3.315.406,69	1.422,81	3.829,00
128	497.612,42	12.040.319,00	2.098,50	4.264,05

5 Conclusão

Este trabalho apresentou a vCubeChain, uma blockchain permissionada, hierárquica e escalável, já implementada no simulador BlockSim (FARIA; CORREIA, 2019) e detalhou-se o funcionamento do simulador com o mecanismo de falhas. Além disso, foi implementada uma versão da blockchain permissionada Hyperledger Fabric.

Realizou-se o desenvolvimento do mecanismo de simulação de falhas no simulador BlockSim, de modo a executar os testes - baseados no artigo original de Freitas, Rodrigues e Duarte Jr. (2023) - em um cenário com falhas. A versão do vCube para sistemas assíncronos foi incluída no simulador e utilizada nos testes. Os resultados foram comparados com os modelos já presentes: Bitcoin, Ethereum e com a nova implementação Hyperledger Fabric.

Os experimentos foram executados baseados em dois cenários principais: sem nenhum nó falho na rede; com o líder da rodada do consenso falho. A partir desses experimentos, comparou-se as soluções. Se utilizou também os resultados prévios apresentados em (FREITAS; RODRIGUES; Duarte Jr., 2023) como parâmetro para verificação de que a adição do mecanismo de falhas não influenciasse nos resultados obtidos no simulador já existente.

Identificou-se nos resultados, que a solução vCubeChain demostra ter um número de mensagens muito inferior aquelas as quais ela foi comparada, mesmo que o número de processos na rede cresça. Com isso, podemos inferir que uma blockchain utilizando a topologia hierárquica do vCube, permanece com suas propriedades escaláveis. Além disso, de uma maneira geral, o desempenho da vCubeChain demonstrou ser superior as demais alternativas.

Como parte dessa pesquisa foram feitas as seguintes publicações:

- Stein, G., Rodrigues, L. A., & Duarte Jr, E. P. (2023, May). Utilizando o vCube para Detecção de Falhas em Sistemas Assíncronos. In Anais do XXIV Workshop de Testes e Tolerância a Falhas (pp. 29-42). SBC. DOI: <https://doi.org/10.5753/wtf.2023.800>
- Stein, G., Rodrigues, L. A., Duarte Jr, E. P., & Arantes, L. (2023, October). Diamond-P-vCube: An Eventually Perfect Hierarchical Failure Detector for Asynchronous Distributed Systems. In Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing (pp. 40-49). DOI: <https://doi.org/10.1145/3615366.3615420>

O desempenho de uma blockchain vai muito além do número de mensagens trocadas, do tempo de latência para validação de todas as transações e de sua resiliência a falhas.

Como trabalho futuro, pode-se pensar em realizar uma avaliação de outras características de blockchains permissionadas, além da construção da vCubeChain como um software independente de simulação, de modo a avaliar seu desempenho em cenários reais.

Referências

ALHARBY, M.; MOORSEL, A. van. Blocksim: An extensible simulation tool for blockchain systems. *Frontiers in Blockchain*, Frontiers, v. 3, p. 28, 2020. Citado na página 35.

ANDROULAKI, E. et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: *Proceedings of the thirteenth EuroSys conference*. [S.l.: s.n.], 2018. p. 1–15. Citado na página 28.

AOKI, Y. et al. Simblock: A blockchain network simulator. In: IEEE. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.], 2019. p. 325–329. Citado 2 vezes nas páginas 35 e 36.

ARAUJO, J. P. de et al. Vcube-pb: Uma solução publish/subscribe autonômica e escalável com difusão confiável causal. In: SBC. *Anais do XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2017. Citado na página 22.

AYSAL, T. C. et al. Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal processing*, IEEE, v. 57, n. 7, p. 2748–2761, 2009. Citado na página 29.

BASHIR, I. *Mastering blockchain*. [S.l.]: Packt Publishing Ltd, 2017. Citado na página 25.

BONDAVALLI, A.; SIMONCINI, L. Failure classification with respect to detection. In: IEEE. *[1990] Proceedings. Second IEEE Workshop on Future Trends of Distributed Computing Systems*. [S.l.], 1990. p. 47–53. Citado na página 17.

BONNEAU, J. et al. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: IEEE. *2015 IEEE symposium on security and privacy*. [S.l.], 2015. p. 104–121. Citado na página 26.

BROWN, R. G. et al. Corda: an introduction. *R3 CEV, August*, R3 CEV New York, NY, USA, v. 1, n. 15, p. 14, 2016. Citado na página 30.

BUCHMAN, E. *Tendermint: Byzantine fault tolerance in the age of blockchains*. Tese (Doutorado) — University of Guelph, 2016. Citado na página 28.

BUTERIN, V. et al. A next-generation smart contract and decentralized application platform. *white paper*, v. 3, n. 37, p. 2–1, 2014. Citado na página 12.

CACHIN, C.; GUERRAOUI, R.; RODRIGUES, L. *Introduction to reliable and secure distributed programming*. [S.l.]: Springer Science & Business Media, 2011. Citado 2 vezes nas páginas 12 e 16.

CACHIN, C. et al. Architecture of the hyperledger blockchain fabric. In: CHICAGO, IL. *Workshop on distributed cryptocurrencies and consensus ledgers*. [S.l.], 2016. v. 310, n. 4, p. 1–4. Citado na página 28.

CACHIN, C.; VUKOLIĆ, M. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017. Citado 2 vezes nas páginas 13 e 44.

CASTRO, M.; LISKOV, B. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, ACM New York, NY, USA, v. 20, n. 4, p. 398–461, 2002. Citado 2 vezes nas páginas 13 e 28.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 43, n. 4, p. 685–722, 1996. Citado na página 29.

CHANDRA, T. D.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 43, n. 2, p. 225–267, 1996. Citado 4 vezes nas páginas 12, 19, 29 e 49.

CHASE, J. M. *A Permissioned Implementation of Ethereum*. [S.l.], 2018. Citado na página 30.

DOLEV, D.; DWORK, C.; STOCKMEYER, L. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 34, n. 1, p. 77–97, 1987. Citado na página 31.

DOUCEUR, J. R. The sybil attack. peer-to-peer systems. *Lecture Notes in Computer Science*, v. 2429, p. 251–260, 2002. Citado na página 25.

DUARTE JR., E. P.; BONA, L. C.; RUOSO, V. K. Vcube: A provably scalable distributed diagnosis algorithm. In: IEEE. *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. [S.l.], 2014. p. 17–22. Citado 3 vezes nas páginas 13, 20 e 48.

DWORK, C.; LYNCH, N.; STOCKMEYER, L. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 35, n. 2, p. 288–323, 1988. Citado na página 17.

EKLUND, P. W.; BECK, R. Factors that impact blockchain scalability. In: *Proceedings of the 11th international conference on management of digital ecosystems*. [S.l.: s.n.], 2019. p. 126–133. Citado na página 46.

EUGSTER, P. T. et al. Epidemic information dissemination in distributed systems. *Computer*, IEEE, v. 37, n. 5, p. 60–67, 2004. Citado na página 29.

FARIA, C.; CORREIA, M. Blocksim: blockchain simulator. In: IEEE. *2019 IEEE International Conference on Blockchain (Blockchain)*. [S.l.], 2019. p. 439–446. Citado 6 vezes nas páginas 32, 35, 36, 37, 42 e 60.

FATTAHI, S. M.; MAKANJU, A.; FARD, A. M. Simba: An efficient simulator for blockchain applications. In: IEEE. *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. [S.l.], 2020. p. 51–52. Citado na página 46.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 32, n. 2, p. 374–382, 1985. Citado 3 vezes nas páginas 11, 18 e 19.

FREITAS, A. E. S.; RODRIGUES, L. A.; Duarte Jr., E. P. vcubechain: Uma blockchain permissionada escalável. In: SBC. *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2023. p. 127–140. Citado 10 vezes nas páginas 13, 14, 16, 30, 32, 42, 43, 50, 54 e 60.

GREVE, F. G. et al. Blockchain e a revolução do consenso sob demanda. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)-Minicursos*, 2018. Citado 2 vezes nas páginas 12 e 44.

GUERRAOUI, R.; RODRIGUES, L. *Introduction to reliable distributed programming*. [S.l.]: Springer Science & Business Media, 2006. Citado na página 11.

HADZILACOS, V. Fault-tolerant broadcasts and related problems. *Distributed systems*, ACM Press, p. 97–145, 1993. Citado na página 29.

HERRERA-JOANCOMARTÍ, J.; PÉREZ-SOLÀ, C. Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. In: SPRINGER. *International Conference on Modeling Decisions for Artificial Intelligence*. [S.l.], 2016. p. 26–44. Citado na página 45.

JEANNEAU, D. et al. An autonomic hierarchical reliable broadcast protocol for asynchronous distributed systems with failure detection. *Journal of the Brazilian Computer Society*, Springer, v. 23, p. 1–14, 2017. Citado 2 vezes nas páginas 31 e 48.

JR., E. D. et al. O elo perdido: Um modelo de diagnóstico distribuído para a implementação de detectores de falhas não confiáveis. In: *Anais do XXIII WTF*. Porto Alegre, RS, Brasil: SBC, 2022. p. 29–42. ISSN 2595-2684. Disponível em: <https://sol.sbc.org.br/index.php/wtf/article/view/21503>. Citado na página 21.

JR., E. P. D. et al. The missing piece: a distributed system-level diagnosis model for the implementation of unreliable failure detectors. *Computing*, v. 105, 2023. ISSN 0010-485X. Disponível em: <https://doi.org/10.1007/s00607-023-01211-8>. Citado na página 48.

KIM, S.; KWON, Y.; CHO, S. A survey of scalability solutions on blockchain. In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. [S.l.: s.n.], 2018. p. 1204–1207. Citado na página 46.

KSHEMKALYANI, A. D.; SINGHAL, M. *Distributed computing: principles, algorithms, and systems*. [S.l.]: Cambridge University Press, 2011. Citado na página 16.

LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems*, ACM New York, NY, USA, v. 16, n. 2, p. 133–169, 1998. Citado na página 29.

LAMPORT, L. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, p. 51–58, 2001. Citado na página 30.

MAKRIDAKIS, S.; CHRISTODOULOU, K. Blockchain: Current challenges and future prospects/applications. *Future Internet*, v. 11, n. 12, 2019. ISSN 1999-5903. Disponível em: <https://www.mdpi.com/1999-5903/11/12/258>. Citado na página 12.

MIERS, C. et al. Análise de mecanismos para consenso distribuído aplicados a blockchain. *SBC*, p. 22, 2019. Citado na página 44.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, p. 21260, 2008. Citado 2 vezes nas páginas 12 e 26.

NARAYANAN, A. et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. [S.l.]: Princeton University Press, 2016. Citado 3 vezes nas páginas 22, 24 e 26.

ONGARO, D.; OUSTERHOUT, J. In search of an understandable consensus algorithm. In: *2014 USENIX annual technical conference (USENIX ATC 14)*. [S.l.: s.n.], 2014. p. 305–319. Citado 2 vezes nas páginas 29 e 30.

POLGE, J. et al. Blockperf: A hybrid blockchain emulator/simulator framework. *IEEE Access*, IEEE, v. 9, p. 107858–107872, 2021. Citado na página 46.

RODRIGUES, L. A. et al. Uma solução de difusão confiável hierárquica em sistemas distribuídos assíncronos. In: SBC. *Anais do XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2017. Citado na página 44.

RODRIGUES, L. A.; JR, E. P. D.; ARANTES, L. Arvores geradoras mınimas distribuıdas e autonômicas. *Anais do 32o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos–SBRC*, v. 2014, p. 1–14, 2014. Citado na página 22.

RODRIGUES, L. A.; JR, E. P. D.; ARANTES, L. A distributed k-mutual exclusion algorithm based on autonomic spanning trees. *Journal of Parallel and Distributed Computing*, Elsevier, v. 115, p. 41–55, 2018. Citado na página 22.

SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 22, n. 4, p. 299–319, 1990. Citado na página 25.

SimPy Read the Docs. <https://simpy.readthedocs.io/en/latest/index.html>. Acessado: 16-08-2023. Citado na página 37.

STEIN, G.; RODRIGUES, L.; JR., E. P. D. Utilizando o vcube para detecção de falhas em sistemas assíncronos. In: *Anais do XXIV Workshop de Testes e Tolerância a Falhas*. Porto Alegre, RS, Brasil: SBC, 2023. p. 29–42. ISSN 2595-2684. Disponível em: <https://sol.sbc.org.br/index.php/wtf/article/view/24680>. Citado 3 vezes nas páginas 13, 15 e 48.

TERRA, A. d. C. *Investigando a implementação do consenso escalável sobre o VCUBE*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná. Curitiba, 2020. Citado na página 44.

THAI, Q. T. et al. Hierarchical byzantine fault-tolerance protocol for permissioned blockchain systems. *The Journal of Supercomputing*, Springer, v. 75, n. 11, p. 7337–7365, 2019. Citado na página 28.

TUREK, J.; SHASHA, D. The many faces of consensus in distributed systems. *Computer*, IEEE, v. 25, n. 6, p. 8–17, 1992. Citado na página 18.

Urban, P.; Defago, X.; Schiper, A. Neko: a single environment to simulate and prototype distributed algorithms. In: *Proceedings 15th International Conference on Information Networking*. [S.l.: s.n.], 2001. p. 503–511. Citado 2 vezes nas páginas 15 e 49.

WOOD, G. et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, v. 151, n. 2014, p. 1–32, 2014. Citado na página 27.

XIE, J. et al. A survey on the scalability of blockchain systems. *IEEE Network*, v. 33, n. 5, p. 166–173, 2019. Citado na página 45.

XING, J. et al. Talaria: A framework for simulation of permissioned blockchains for logistics and beyond. *arXiv preprint arXiv:2103.02260*, 2021. Citado na página 46.

ZIWICH, R. P.; DUARTE, E.; ALBINI, L. C. P. Distributed integrity checking for systems with replicated data. In: IEEE. *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*. [S.l.], 2005. v. 1, p. 363–369. Citado na página 22.