Leonardo de Freitas Galesky

Sincronização Eficiente de CRDTs em Escala utilizando VCube-PS

Cascavel-PR

2023

Leonardo de Freitas Galesky

Sincronização Eficiente de CRDTs em Escala utilizando VCube-PS

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.

Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel

Centro de Ciências Exatas e Tecnológicas – CCET

Programa de Pós-Graduação em Ciência da Computação – PPGComp

Orientador(a): Prof. Dr. Luiz Antonio Rodrigues

Cascavel-PR

2023

de Freitas Galesky, Leonardo

Sincronização Eficiente de CRDTs em Escala utilizando VCube-PS / Leonardo de Freitas Galesky; orientador Prof. Dr. Luiz Antonio Rodrigues. – Cascavel-PR, 2023.

52p.

Dissertação (Mestrado Acadêmico Campus de Cascavel) – Universidade Estadual do Oeste do Paraná, Centro de Ciências Exatas e Tecnológicas, Programa de Pós-Graduação em Ciência da Computação, 2023.

1. Conflict-Free Replication Data Types. 2. vCube. 3. Replication. I. Rodrigues, Luiz Antonio, orient. II. Título.

Leonardo de Freitas Galesky

Sincronização Eficiente de CRDTs em Escala utilizando VCube-PS

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.

Prof. Dr. Luiz Antonio Rodrigues
Orientador

Prof. Dr. Edson Tavares de Camargo
UTFPR-Toledo/PPGComp-Unioeste

Profa. Dra. Luciana Arantes
Sorbonne Universités/LIP6

Cascavel-PR
2023

Dedico este trabalho aos meus pais, pelo amor incondicional, apoio e sacrifícios que fizeram para me proporcionar uma educação de qualidade. Sem a confiança e incentivo deles, chegar tão longe não seria possível. Dedico também à companheira de todas as jornadas, minha esposa.

Resumo

GALESKY, Leonardo de Freitas. **Sincronização Eficiente de CRDTs em Escala utilizando VCube-PS**. Orientador(a): Prof. Dr. Luiz Antonio Rodrigues. 2023. 52f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2023.

Sistemas distribuídos são utilizados para construir serviços de grande escala na internet, nesse contexto modelos com menor rigor de consistência permitem otimizar a disponibilidade destes sistemas na forma de escalabilidade, latência e tolerância a falhas. Entretanto, o uso de regras mais relaxadas introduz a possibilidade de conflitos que precisam ser arbitrados pelos participantes. Para realizar esse processo de forma descentralizada os protocolos costumam usar de estratégias *ad-hoc* como LWW (*Last Writer Wins*) ou mesmo bloquear o sistema até que o estado inconsistente seja resolvido manualmente. *Conflict Free Replicated Data Types (CRDTS)* definem estruturas de dados que atendem a especificações matemáticas que garantem que operações podem ser realizadas de forma independente e concorrente sem qualquer forma de coordenação, permitindo ainda que regras de resolução de conflitos possam ser definidas com maior granularidade e se adaptando a especificação do domínio onde são aplicados. Este estudo apresenta o VCube-Sync, um sistema que utiliza de uma topologia de hipercubos virtuais como base para replicação de um data-store baseado em Tipos de Dados Replicados e Livres de Conflitos - CRDT (Conflict-free Replicated Data Types). Hipercubos já foram empregados anteriormente como rede de sobreposição estruturada para a distribuição de mensagens devido à tolerância a falhas e latência logarítmica, permitindo ainda o desenvolvimento de heurísticas de otimização baseadas no conhecimento da configuração da sobreposição. O protocolo de replicação apresentado neste estudo foi baseado no VCube-PS explorando a sinergia entre sistemas publicação-subscrição e de replicação. O protocolo foi testado sob várias distribuições de carga e rede usando o testbed Grid5000, e os resultados foram comparados com os de outros protocolos de replicação de pesquisas recentes. Os resultados deste estudo mostram que o VCube-Sync fornece bons resultados em termos de latência, escalabilidade e uso de rede.

Palavras-chave: CRDT; replicação; sistemas distribuídos;

Abstract

GALESKY, Leonardo de Freitas. **Efficient Synchronization of CRDTs at Scale using VCube-PS**. Orientador(a): Prof. Dr. Luiz Antonio Rodrigues. 2023. 52f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2023.

Distributed systems are used to build large-scale services on the Internet. In this context, models with looser consistency guarantees allow for optimizing the availability of these systems in the form of scalability, latency, and fault tolerance. However, using more relaxed rules introduces the possibility of conflicts that need to be arbitrated by the participants. To carry out this process in a decentralized way, protocols often use *ad-hoc* strategies such as LWW (*Last Writer Wins*) or even blocking the system until the inconsistent state is resolved manually. *Conflict Free Replicated Data Types (CRDT)* define data structures that meet mathematical specifications that guarantee that operations can be performed independently and concurrently without any form of coordination, allowing conflict resolution rules to be defined with greater granularity and adapting to the specification of the domain where they are applied. This study presents VCube-Sync, a system that uses a topology of virtual hypercubes for replicating a data store based on CRDT (Conflict-free Replicated Data Types). Hypercubes have been previously employed as a structured overlay network for message distribution due to fault tolerance and logarithmic latency, while also allowing the development of optimization heuristics based on knowledge of the overlay configuration. The replication protocol presented in this study was based on the VCube-PS exploiting synergies between publication-subscription and replication systems. The protocol was tested under various load and network distributions using the Grid5000 testbed, and the results were compared with other replication protocols from recent research. The results of this study show that VCube-Sync provides good results in terms of latency, scalability and network usage.

Keywords: CRDT; replication; distributed systems;

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CRDT	Conflict Free Replicated Data Types
HAT	Highly Available Transactions
LWW	Last Writer Wins
Pub-Sub	Publish Subscribe
P2P	Peer to Peer
Delta-CRDT	Delta State CRDT

Sumário

1 Introdução

Para atender a um grande número de usuários e requisições, sistemas massivos utilizam de escalonamento horizontal e replicação como mecanismos de aumento de disponibilidade. Este processo em um sistema distribuído depende do estabelecimento de uma estratégia de consistência de dados que pode ser mais rigorosa, e menos disponível, como no caso de consistência forte (*strong consistency*), ou mais relaxada como no caso de consistência a termo (*eventual consistency*) em que os nós podem temporariamente divergir e a ordem total das operações não pode ser estabelecida (VOGELS, 2009). Esta escolha é necessária já que, considerando o teorema CAP (*Consistency*, *Availability* e *Partition Tolerance*), para melhorar a disponibilidade de um sistema distribuído e particionado é preciso sacrificar garantias de consistência (GILBERT; LYNCH, 2002).

Mesmo em sistemas com consistência a termo, a etapa de replicação pode ser um gargalo, fazendo com que as réplicas fiquem em um estado inconsistente por longos períodos de tempo, impactando a latência de leitura e a experiência do usuário. Este impacto é maximizado quando se fala de sistemas massivamente distribuídos, ou seja com muitos nós, ou georeplicados (SAITO; SHAPIRO, 2005).

Os cenários mencionados demandam a aplicação de estratégias especializadas de difusão e replicação, as quais têm sido extensivamente exploradas, como documentado em Meiklejohn e Van Roy (2015) e Meiklejohn e Van Roy (2017). Estes estudos avaliaram os limites de escalabilidade em sistemas de dados distribuídos sem líder e escaláveis horizontalmente. Nesses contextos, foram propostos o uso de protocolos mais eficientes, tais como árvores de difusão ou melhorias funcionais através da adoção da difusão por tópicos.

Desafios igualmente significativos são abordados por Fouto, Leitao e Preguica (2018), que apresenta resultados promissores ao empregar replicação parcial e a separação das camadas de dados e controle de causalidade quando necessário. Recentemente, Vieira (2021) investigou a aplicação de árvores de difusão e estruturas livres de conflito em um ambiente caracterizado por presença dinâmica.

Para a maioria das aplicações, não basta que os dados do sistema sejam replicados eficientemente, mas é necessário também que o estado final seja correto. Em sistemas consistentes a termo, operações independentes e concorrentes podem gerar conflitos e, portanto, um estado incorreto. A resolução destes conflitos pode se dar por soluções simples como "a última escrita ganha", prioridades baseadas em metadados, ou até mais avançadas como através o uso de relógios vetoriais e outras estratégias ad-hoc. Entretanto, estas implementações podem resultar em perda de dados ou em um estado final inconsistente

(DECANDIA et al., 2007).

Neste sentido, estruturas de dados convergentes e livres de conflito (CRDT, *Conflict-Free Replicated Data Types*) são estruturas de dados baseadas em conceitos matemáticos simples que garantem a convergência a termo entre réplicas sem a necessidade de um líder. Nestes, a operação de união (*join*) dos dados em diferentes réplicas é determinística e converge em um resultado correto e equivalente. As soluções podem ser divididas em três categorias: baseadas em estado, baseadas em operações e delta-estado.

CRDTs baseados em estado não dependem da ordenação das mensagens, são idempotentes e tolerantes a falhas. Entretanto, transmitem todo o conjunto de dados às outras réplicas após alterações na cópia local. Isto implica em crescimento ilimitado da *payload* e, portanto, apresenta desafios de vazão (*throughput*) e latência (SHAPIRO et al., 2011b). Por outro lado, os CRDTs baseados em operações são capazes de serializar a operação de alteração local e transmiti-la para as demais réplicas. No entanto, geralmente esta implementação depende de entrega única (*only once delivery*) e garantia de ordenação causal das mensagens (YOUNES et al., 2016).

Implementar *middlewares* que apresentem garantia de causalidade é uma das estratégias exploradas para o uso de CRDTs baseados em operações. Esta solução modular é utilizada, por exemplo, por Baquero, Almeida e Shoker (2017) e Younes et al. (2016), e permite que a estrutura de dados seja mais simples e se utilize de mensagens menores. Recentemente, Bauwens e Boix (2021) apresentou otimizações ao mecanismo de ordenação causal e compactação de *log* sob o qual os trabalhos previamente citados se baseiam.

Em Younes et al. (2016) uma solução de dados distribuídos baseada em CRDTs foi implementada utilizando o Redis como camada de armazenamento. Porém, os autores identificaram que o protocolo de replicação padrão era baseado em um barramento, o que atuaria como um gargalo do sistema, este foi portanto substituído por uma malha totalmente conectada. Utilizar-se de uma topologia de rede especializada trouxe melhorias ao desempenho do sistema. Entretanto, o modelo utilizado ainda possui limitações de escalabilidade, especialmente na largura de banda de cada nó.

Além dos exemplos citados, diversas soluções recentes implementam CRDTs, sejam banco de dados como Riak (BROWN et al., 2014) ou *frameworks* de paradigma distribuído como o Phoenix (MCCORD, 2022). Estas soluções usam de algoritmos próprios e acoplados para lidar com desafios de difusão, organizações dos nós do *cluster* e em alguns casos a ordenação causal de mensagens. Outros trabalhos como Vieira (2021) e Akkoorath et al. (2016) apresentam CRDTs associados a protocolos mais modulares como o *Plumtree*, mas consideram topologias não-estruturadas, o que limita as inferências que podem ser realizadas com base no *layout* da sobreposição e no relacionamento entre os seus membros que podem ser utilizadas para desenvolver estratégias mais eficientes de difusão.

O algoritmo VCube, proposto por Duarte, Bona e Ruoso (2014), determina uma topologia estruturada baseada em hipercubo, que apresenta importantes propriedades logarítmicas. A implementação VCube-PS (ARAUJO et al., 2017; de Araujo et al., 2019), um sistema *Publish-Subscribe* (*pub-sub*) baseado em tópicos, explora essas características para obter difusão eficiente e ainda garantir ordenação causal das mensagens. Cada nó arranja os membros de um tópico como uma árvore hierárquica que é uma rede de sobreposição da topologia completa e tem como raiz o emissor da mensagem. Somente os membros de um tópico recebem suas mensagens e *relays*, quando existem, são temporários. Além disso, qualquer nó pode atuar como raiz (fonte) de uma mensagem. Apesar de todas estas propriedades o VCube-PS ainda não havia sido explorado como estratégia de replicação.

Este trabalho apresenta o VCube-Sync, um algoritmo logariticamente escalável de sincronização de CRDTs com suporte a replicação parcial via tópicos explorando a sinergia entre sistemas *pub-sub* e protocolos de replicação. Assim como VCube e VCube-PS, a transmissão de operações faz uso de uma rede de sopreposição baseada em hipercubos e, como o segundo, é capaz de garantir a ordenação causal das *payloads* de replicação, característica essencial para CRDTs baseados em operação. Sua topologia estruturada apresenta grande potencial também para otimizações como agrupamento de mensagens e cálculo de deltas.

1.1 Objetivos

Este trabalho visa otimizar a eficiência de sincronização de bases de dados que implementam CRDTs utilizando do algoritmo de difusão hierárquica VCube como base para a estratégia de replicação.

Para tanto, são definidos os seguintes objetivos específicos:

- Avaliar soluções existentes de tipos de dados distribuídos, em especial utilizando de estruturas de dados convergentes e livres de conflito (CRDTs).
- Implementação de um CRDT baseados em operações.
- Implementação de mecanismo de replicação baseado em hipercubos, considerando os tópicos como critério de *sharding*.
- Instrumentar e emular o comportamento do algoritmo utilizando uma *grid* computacional e obter as métricas de latência, uso de rede e escalabilidade.
- Avaliar o impacto em cada uma das métricas e determinar a viabilidade da solução como mecanismo de replicação para CRDTs e *datastore*.

1.2 Metodologia

Este estudo foi conduzido por meio da implementação do algoritmo proposto, denominado VCube-Sync, utilizando o framework Babel. Essa implementação foi integrada a uma arquitetura de aplicação modular, permitindo a modificação ou substituição do protocolo de replicação sem afetar substancialmente as demais funcionalidades. Detalhes dessa arquitetura estão expostos na Seção 3.1 - Arquitetura da Aplicação.

Para fins de comparação, o mesmo procedimento foi aplicado ao protocolo descrito por Vieira (2021), utilizando o mesmo ambiente e implementação, com a alteração exclusiva do protocolo de replicação da aplicação.

Realizamos simulações em um ambiente de grid modelando cenários de redes com diferentes números de nós e diversos critérios de replicação. Mantivemos o estado da aplicação por meio de CRDTs baseados em operações. Ao final, conduzimos uma avaliação comparativa do algoritmo VCube-Sync como protocolo de replicação. Em todos os cenários, os nós estão completamente conectados, e as topologias são implementadas como uma sobreposição (*overlay*) da rede existente.

1.2.1 Métricas

Foi avaliada a latência de entrega das mensagens e sua visilidade na camada de dados, o volume de dados transmitido para atingir um estado sincronizado e a escalabilidade em relação ao número de nós e distribuição dos tópicos. Estas métricas foram obtidas instrumentando a implementação com ferramentas de *logging*. Estas métricas são similares às obtidas por Vieira (2021).

1.3 Trabalhos relacionados

A seguir são apresentados trabalhos relacionados que implementam protocolos de difusão modulares como ferramenta de replicação. São selecionados em especial aqueles em que a resolução de conflitos e escalabilidade são priorizados.

C³ (FOUTO; LEITAO; PREGUICA, 2018) foi construído como uma camada modular plugável a diferentes *backends*, o C³ introduz um modelo capaz de fornecer ordenação causal e replicação parcial num ambiente P2P geo-distribuído. Esta camada utiliza de identificadores e relógios vetoriais para estabelecer uma barreira causal, garantindo que uma operação só é de fato executada após todas as suas dependências serem satisfeitas. O protocolo foi validado estendendo o banco de dados Cassandra, sendo demonstrado que apresenta boa performance em cenários de replicação parcial assim como os propostos pelo presente trabalho, entretanto as garantias de causalidade não foram aproveitadas nem avaliadas para estruturas de dados com garantias fortes como CRDTs.

Legion (LINDE et al., 2017) é um *framework* que permite múltiplos clientes a replicarem seu estado de maneira hibrida entre si e entre um servidor centralizado. Para garantir que operações possam ser feitas concorrentemente mas sem perder a consistência dos dados o modelo usa de CRDTs baseados em delta-estado, isso também permite que aplicações desconectadas continuem funcionais e possam ser sincronizadas num momento futuro. Além disso, afim de otimizar a latência do sistema, foi determinado também um protocolo de disseminação que estabelece um *overlay* em que nós conhecem apenas um determinado número de vizinhos escolhidos por proximidade e tópicos. Esta implementação é similar ao proposto nesta pesquisa mas possui um critério de topologia não estruturado, implicando que não é possível realizar inferências em relação a estrutura da rede, além disso é presumida a existência de um ou vários servidores centralizados.

Cure (AKKOORATH et al., 2016) é um protocolo de replicação descentralizado para bases de dados chave-valor que fornece consistência causal e atomicidade, ou seja, operações consistentes simultâneas em múltiplas chaves, isto representa uma forma de transações altamente disponíveis (*Highly Available Transactions - HATs*). Para garantir que operações concorrentes irão convergir para o mesmo valor correto, o modelo utiliza de CRDTs baseados em operações e para manter causalidade são utilizados relógios vetoriais, estes também são usados para controlar diferentes versões de um mesmo registro que eventualmente serão *garbage collected*. Este protocolo é a base do banco de dados AntidoteDB. Uma das limitações deste sistema é a não escalabilidade do acompanhamento da causalidade o que impede o crescimento do número de nós participantes, o presente trabalho presume sistemas massivamente distribuídos o que torna a implementação do Cure inviável.

Selective hearing (MEIKLEJOHN; Van Roy, 2015) é um modelo que une o existente *framework* LASP, que tem como estrutura de dados primária CRDTs e possui mecanismos para comunicação e sincronização entre processos, e uma estrutura de difusão epidêmica baseada no protocolo *Plumtree*. O produto final também permite que nós possam ter uma visão parcial do *cluster* através de *overlays* e tem como caso de uso aplicações de internet das coisas e jogos *mobile*. Esta estratégia objetiva reduzir a latência de visibilidade dos dados mantendo ainda *membership* dinâmico. Esta implementação se baseia no modelo de programação *LASP*, não sendo modular ou agnóstica de ambiente e além disso, diferentemente do presente trabalho, considera um *overlay* não estruturado.

SYNC Tree (VIEIRA, 2021) é um protocolo de difusão baseado em uma versão modificado do protocolo Plumtree que apresenta garantias de causalidade e que é capaz de sincronizar centenas de nós. Este modelo é capaz de suportar *membership* dinâmico e introduz uma estratégia de sincronização para entrada e saída de nós. O sistema é utilizado em conjunto de CRDTs que representam uma estrutura de dados distribuída e implementa estratégias como *garbage collection* para atingir melhores resultados de

latência e custo de comunicação, quando todos os mecanismos mencionados atuam em conjunto determina-se o ECO SYNC Tree, que é a versão otimizada do protocolo. Como trabalho futuro é indicado a implementação de replicação parcial, que é um dos tópicos abordados no presente trabalho, além disso, uma diferença importante se trata também do uso de um *overlay* não estruturado.

Os modelos apresentam características semelhantes ao proposto por este trabalho mas nenhum de forma completa, nossa solução de difusão foi desenvolvida visando:

- Atender a sistemas massivamente distribuídos;
- Atuar de maneira descentralizada;
- Suportar replicação parcial explorando a sinergia entre sistemas *publisher-subscriber* e os de replicação;
- Utilizar de topologia estruturada que permita inferir propriedades sobre nós e seus vizinhos;

Além disso, nosso sistema utiliza de CRDTs para resolução de conflitos e manutenção de um estado consistente.

Os modelos mais similares ao proposto são *Selective Hearing* e *SYNC Tree*, este último foi utilizados como principal referência para fins de comparação.

1.4 Organização do Texto

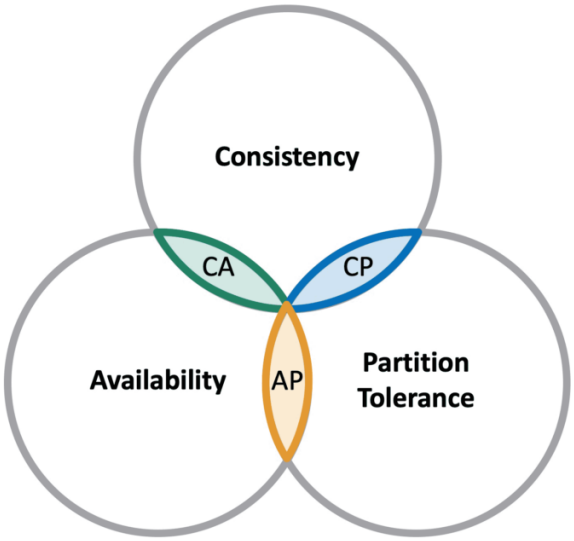
Este trabalho é organizado como segue. O Capítulo 2, Fundamentação Teórica, fornece uma introdução básica à sistemas distribuídos, replicação e a teoria por trás de alguns CRDTs. Ele também destaca os diferentes projetos de CRDT e suas variações. O Capítulo 3 apresenta o protocolo de replicação VCube-Sync e aprofunda os detalhes da implementação. Este capítulo inclui os algoritmos propostos. O Capítulo 4 detalha os experimentos realizados e os resultados obtidos para VCube-Sync em termos de latência e largura de banda. Ao final no Capítulo 5 o conteúdo e resultados da tese são resumidos, incluindo também trabalhos futuros.

2 Fundamentação Teórica

Sistemas distribuídos podem ser descritos por um conjunto de N processos ou nós (p_0, p_1,...,p_{n-1}) independentes que se comunicam através de mensagens a fim de atingir um resultado comum (TANENBAUM, 2017).

O teorema CAP ou conjectura de Brewer (GILBERT; LYNCH, 2002) descreve três propriedades de sistemas distribuídos: consistência (C - *consistency*), que se refere a múltiplos nós possuírem a mesma cópia de um conjunto de dados, disponibilidade (A - *availability*) que se refere ao dados estarem sempre disponíveis e tolerância a particionamento (P - *partition tolerance*) que se refere a capacidade do sistema de continuar funcional após um particionamento de rede. O teorema indica que nesta classe de sistemas só são possíveis combinações absolutas de no máximo duas das três propriedades, como apresentado no diagrama da Figura 1. Ou seja, uma das propriedades precisa ser sacrificada.

Figura 1 – Diagrama de representação do Teorema CAP



Em grandes sistemas distribuídos, especialmente se geodistribuídos, é inevitável que exista particionamento de rede. Portanto, têm-se a opção entre consistência (C) ou disponibilidade (A). Esta escolha não é binária, significando que podem ser feitos *trade-offs* parciais configurando um grande espectro de aplicações com diferentes propriedades (VOGELS, 2009).

Neste capítulo são apresentados os conceitos relevantes ao contexto deste trabalho com foco em modelos de consistência e replicação.

2.1 Modelos de Consistência

Aplicações de grande escala utilizam-se de bases de dados distribuídas e replicadas para atingir latências baixas, alta disponibilidade e tolerância a falhas. O espectro de *trade-offs* de disponibilidade (A) e consistência (C) pode ser resumido em duas grandes categorias, que são consistência forte (*strong consistency*), em que a consistência é priorizada, e consistência a termo (*eventual consistency*), em que a disponibilidade é priorizada. A garantia da consistência se torna especialmente difícil conforme o sistema cresce, já que se esta for mantida com rigor, a ordem das operações precisa ser gerenciada e consentida pelos membros da rede, o que é um tarefa bastante custosa e diretamente relacionado a latência de comunicação entre os nós. Sistemas de consistência a termo superam esta limitação, mas introduzem seus próprios desafios como gerenciamento de conflitos e manutenção de invariantes (BREWER, 2012).

2.1.1 Consistência forte

Bases de dados que apresentam consistência forte num ambiente distribuído utilizam de recursos como consenso e coordenação de forma que, na perspectiva das aplicações que dependem dela, parece haver uma única réplica. Na ótica do desenvolvedor, isto representa uma facilidade ao interagir com os dados, mas para o sistema, significa que qualquer operação que altere o estado deve ter sua ordenação determinada por consenso e ser aplicada a todas as réplicas de forma a resultar em um único estado consistente e correto (MAHMOUD et al., 2013).

Quando se deseja obter este nível estrito de consistência, mas aumentar a disponibilidade e a capacidade, por exemplo permitindo operações de escrita em múltiplas réplicas, é preciso introduzir um procedimento capaz de determinar e garantir a ordem das operações concorrentes. Em geral, para isso são utilizados protocolos de consenso (*quorum consensus*), como *two phase commit* e PAXOS. Entretanto, estas estratégias são de difícil implementação e adicionam latência ao sistema por depender de um grande número de troca de mensagens (GRAY; LAMPORT, 2006). Mesmo outros protocolos como RAFT (ONGARO; OUSTERHOUT, 2014), ainda extendido por múltiplas otimizações como encontrado em Mahmoud et al. (2013) e Lamport (2006), apresentam alta complexidade e custos consideráveis que escalam em proporção ao número de nós do sistema.

2.1.2 Consistência a termo

Bases de dados que apresentam consistência a termo implementam menos garantias e priorizam a disponibilidade do sistema. Esta categoria pode ser ainda sub dividida em Consistência a termo (EC, *eventual consistency*) e Consistência a termo forte (SEC, *strong eventual consistency*) (SHAPIRO et al., 2011b)

Modelos de consistência a termo não dependem de consenso ou sincronização em tempo real, resultando em melhor latência e potencial de tolerância a falhas. Na prática, isso quer dizer que os nós de um sistema posteriormente consistente podem divergir temporariamente, permitindo que operações de escrita possam ocorrer localmente em uma réplica e, somente num momento futuro, sejam sincronizadas com os demais nós. Por isso, têm se tornado a opção para construção de sistemas de grande escala como Cassandra do Facebook (LAKSHMAN; MALIK, 2010) e o Dynamo da Amazon (DECANDIA et al., 2007).

Segundo Bernstein e Das (2013), ao permitir que operações de escrita ocorram em múltiplos nós, introduz-se a possibilidade de conflitos entre operações que ocorreram de forma independente em diferentes réplicas. A resolução destes conflitos pode ser inclusive delegada a aplicação, e realizada por uma série de algoritmos de reconciliação como *Last-writer-wins* (LWW), em que, utilizando de metadados como *timestamps* ou *vector clocks*, se opta pelo valor que tenha sido escrito por último na ordem de operações. Este tipo de resolução de conflito pode implicar em perda de dados, já que um dos valores propostos é descartado. A estratégia LWW é implementada por padrão, por exemplo, pelos bancos de dados Cassandra (LAKSHMAN; MALIK, 2010) e Dynamo (DECANDIA et al., 2007). Heurísticas arbitrárias para resolução de conflitos podem levar a resultados imprevisíveis ou incorretos e a maioria dos sistemas atuais não é capaz de descrever ou lidar com todo o conjunto de cenários de conflito de seu domínio (BREWER, 2012) .

A consistência a termo forte corrige a limitação da consistência a termo, garantindo que todas as replicas corretas que aplicarem as mesmas operações, independente de ordem, devem ter um estado equivalente e correto. Este processo exige maior rigor na estrutura de dados do que sua predecessora, já que deve assegurar um modelo absolutamente livre de conflitos que não possam ser automaticamente reconciliados. Dentre a classe de estruturas de dados que apresentam estas garantias se destaca os *Conflict Free Replicated Data Types (CRDTs)* (SHAPIRO et al., 2011b).

2.2 Estruturas de dados convergentes e livres de conflito

Estruturas de dados convergentes e livres de conflito (CRDT, *Conflict-Free Replicated Data Types*) são estruturas de dados que possuem propriedades matemáticas comprovadas que garantem consistência de maneira determinística, livre de conflitos e de *roll-backs*. CRDTs podem ser baseados em estado, operações ou delta-estado (PREGUIÇA; BAQUERO; SHAPIRO, 2018).

2.2.1 CRDTs baseados em estado

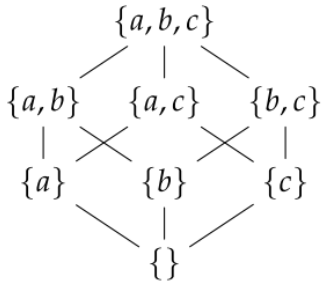
Um CRDT baseado em estado pode ser definido pela tupla $(S, \sqsubseteq, \sqcup)$ em que S é um *join-semilattice*, $\sqsubseteq$ representa ordem parcial e $\sqcup$ é um operador de união (*join*) que deriva o menor limite superior (*least upper bound* - *LUB*) para todos os elementos de S, tal que $\forall s,t,u \in S$:

$$s \sqcup s = s \qquad \text{(idempotência)}$$
$$s \sqcup t = t \sqcup s \qquad \text{(comutatividade)}$$
$$s \sqcup (t \sqcup u) = (s \sqcup t) \sqcup u \qquad \text{(associatividade)}$$

Estas propriedades garantem que a reordenação ou duplicação de mensagens não afetarão a convergência do sistema. Além disso, a perda de mensagens não afeta o resultado final, bastando que seja recebida no futuro uma mensagem contendo aquele estado ou outro de maior ordem do que aquele que foi perdido. O principal ponto negativo desta implementação é o potencial crescimento ilimitado do estado, especialmente dado que a sincronização do sistema acontece através da transmissão do estado atual local completo de cada nó (SHAPIRO et al., 2011b).

Em Shapiro et al. (2011a) são listadas algumas das diversas estruturas de dados diferentes que podem ser considerados CRDTs. Um exemplo é um conjunto de elementos únicos E sem remoção (*grow-only set* - *GSet*): ao tomar o conjunto de todos os subconjuntos de E; $\sqsubseteq$ como a presença do elemento no conjunto e $\sqcup$ como a união de dois conjuntos, definimos este CRDT que pode ser representado pelo diagrama de Hasse da Figura 2 para E = {a, b, c}.

Figura 2 – Diagrama de Hasse do conjunto GSet{a,b,c}



Outro exemplo de um CRDT baseado no estado é o contador incremental apresentado no Algoritmo 1. A mensagem de replicação é um vetor onde cada posição corresponde a um nó no sistema, como num relógio vetorial. Cada nó incrementa a posição que o representa e depois transmite o estado atualizado, isto é o vetor inteiro, para outros membros.

Para calcular o valor total do contador na perspectiva de cada nó, basta somar todos os valores individuais armazenados no vetor.

Algorithm 1 Contador exclusivamente incrementável em CRDT baseado em Estado no processo i (SHAPIRO et al., 2011b)

```
function Init( )
    val = []

function Query( )
    return sum(val)

function Local_Update(n)
    val[i] = val[i] + n

function Merge(X, Y)
    for r ∈ X.val.keys ∪ Y.val.keys do
        val[r] = max(X.val[r], Y.val[r])
```

2.2.2 CRDTs Baseados em Operações

CRDTs baseados em operações são mais espacialmente eficientes que aqueles baseados em estado por transmitir apenas uma representação das operações que ocorrem localmente em cada um dos nós e não seu estado completo. Entretanto, para que a convergência do sistema seja alcançada é necessário um canal de transmissão livre de falhas (*reliable broadcast*). E ainda, caso a estrutura definida não seja comutativa, é preciso que o canal de transmissão ou módulos auxiliares estabeleçam garantia de ordenação causal.

Apesar disto, a entrega causal confiável de mensagens (*reliable causal delivery*) não exige necessariamente consenso e pode ser imune a particionamento, isto é, réplicas conectadas em uma partição da rede podem comunicar suas operações entre si e, a termo, entregar a mensagem a todas as outras réplicas (SHAPIRO et al., 2011a).

Middlewares que implementam *causal broadcast* são uma solução conveniente e bastante utilizada para este tipo de cenário. Ainda no contexto de CRDTs baseados em operações especificamente, uma definição mais completa desta estratégia foi inicialmente proposta e implementada por Baquero, Almeida e Shoker (2014) e revisada em Juan-Marín et al. (2016).

Para cada operação de atualização, este tipo de CRDT deve definir duas funções (PREGUIÇA, 2018):

Generator é executado na réplica a partir da qual a atualização é iniciada. Não tem efeitos colaterais e gera um *Effector* que codifica os efeitos colaterais da operação.

Effector aplica o efeito colateral a todas as réplicas e atualiza o estado.

Um exemplo de contador baseado em operações é apresentado em Algoritmo 2. A mensagem é uma representação serializada do *Effector* contendo o comando e um número inteiro. O valor inicial local do contador é zero. Neste exemplo, o termo *inc* é uma representação da operação a ser aplicada pelo CRDT, ou seja, o incremento do estado local pelo valor n.

Algorithm 2 Contador exclusivamente incrementável em CRDT baseado em Operações (SHAPIRO et al., 2011b)

```
function Init( )
    val = 0

function Generator( )
    return ⟨inc, n⟩

function Effector(n)
    val = val + n
```

2.2.3 CRDTs Baseados em Delta-Estado

CRDTs baseados em delta-estado (δ-CRDT) incorporam as garantias do CRDTs baseados em estado, mas implementam técnicas para redução do tamanho das mensagem como os baseados em operações. Isto é possível implementando *delta-mutators* que retornam o *δ-state*, uma representação do efeito de uma sequência de atualizações no estado baseada na diferença deste em cada par de nós. Este δ pode ser armazenado em um *buffer* que em tempo é transmitido para replicas remotas (ALMEIDA; SHOKER; BAQUERO, 2015).

Um dos desafios dos CRDTs baseados em δ-estado é o critério de propagação que deve evitar, por exemplo, redundância de envio. Sem este controle, o desempenho pode ser inferior ao do modelo totalmente baseado em estado. Algumas das otimizações de sincronização propostas por Enes et al. (2019) utilizam-se de metadados sobre a topologia e *membership* da rede para atingir melhores resultados de banda utilizada, consumo de memória e tempo de processamento até a convergência.

2.2.4 Implementações de CRDTs

CRDTs são encontrados em diversos sistemas. Nesta seção são apresentados alguns de maior notoriedade:

Phoenix presence *Phoenix* (MCCORD, 2022) é um framework de desenvolvimento web desenvolvido em *Elixir*. Uma das funcionalidades chamada de "*Presence*" utiliza CRDTs de deltas estado para permitir que um usuário registre informações num

processo que são replicados a outros membros do *cluster*. Um caso de uso seria, por exemplo, apresentar quais usuários estão online numa sala de *chat*.

Riak O banco de dados Riak (BASHO, 2022) utiliza CRDTs baseados em delta-estado para diversas estruturas de dados distribuídos como contadores e *Maps*. Este banco de dados esteve a frente de diversas das inovações no âmbito de CRDTs (ALMEIDA; SHOKER; BAQUERO, 2018).

AntidoteDB O banco de dados Antidote (AKKOORATH, 2016) tem como objetivo disponibilizar maneiras eficientes para desenvolvimento de aplicações corretas sem precisar abrir mão da performance encontrada em sistemas escaláveis horizontalmente. Seu design é feito de forma a ser provisionado globalmente e geo-replicado sendo ainda assim capaz de realizar operações mais rigorosas como HATs (*highly-available transactions*). No centro destas funcionalidades está o protocolo Cure que se baseia em CRDTs.

Redis Um banco em memória de código aberto. O Redis (REDIS, 2017) é utilizado por milhões de desenvolvedores como uma ferramenta de *cache*. Em 2017 o Redis introduziu uma funcionalidade de geo-replicação em que todos os nós podem receber operações de escrita, convergindo para um único estado correto, com uma solução baseada em CRDTs.

Conclui-se, por fim, que CRDTs são uma solução eficiente para estrutura de dados em ambientes altamente concorrentes. Suas garantias de convergência em um estado final correto tem sido utilizadas por diferentes aplicações para atingir alta escalabilidade e performance. Nota-se, entretanto, que CRDTS baseados em operações e CRDTs baseados em Delta-Estado são dependentes de estratégias otimizadas de replicação e difusão dos mensagens.

2.3 Replicação

A replicação de dados costuma ser usada como um mecanismo de tolerância a falhas e de melhoria de desempenho em sistemas distribuídos. Diferentes réplicas podem manter cópias das informações, garantindo não existir um único ponto de falha e potencialmente distribuindo a carga de acessos. Além disso, é possível otimizar a latência do sistema mantendo réplicas mais próximas aos usuários. Para que exista um estado consistente, ainda que a termo, é necessário que estas réplicas sejam capazes de se comunicar através de um protocolo de replicação (SAITO; SHAPIRO, 2005).

Em sistemas de replicação centralizada, toda a comunicação entre os nós é direcionada por um líder. Já em sistemas de replicação decentralizada, a comunicação é feita entre

nós arbitrários. Esta troca de mensagens ocorre através de primitivas que implementam o envio a todos os participantes (*broadcast*) ou apenas alguns (*multicast*). Após o recebimento de uma mensagem, a sua entrega (*deliver*) pode ser deliberadamente atrasada de forma a satisfazer uma condição pré-definida como a ordenação causal (BIRMAN; SCHIPER; STEPHENSON, 1991).

2.3.1 Modelos de replicação

A replicação total é uma estratégia popular em sistemas distribuídos, na qual os dados são duplicados em vários nós para garantir a disponibilidade e reduzir o impacto de falhas. Nesta estratégia, cada membro do sistema guarda uma cópia completa dos dados, o que permite que o sistema continue funcionando de maneira consistente mesmo que alguns nós estejam indisponíveis (SAITO; SHAPIRO, 2005).

No entanto, o alto custo associado à replicação total é uma de suas principais desvantagens. À medida que os dados crescem em tamanho e complexidade, o volume de armazenamento e largura de banda necessários para manter todos os nós sincronizados também aumenta. Além disso, o custo de propagação de alterações nos dados em todos os nós pode ser significativo, especialmente em sistemas grandes e dispersos geograficamente. Isso pode levar a gargalos de desempenho e deteriorar a operação geral do sistema. Apesar desses desafios, a replicação completa costuma ser considerada uma estratégia de replicação de dados confiável e eficaz em determinados aplicativos, como sistemas financeiros e computação de alto desempenho, nos quais a disponibilidade e a confiabilidade dos dados são essenciais (SAITO; SHAPIRO, 2005).

Já a replicação parcial é uma estratégia em sistemas distribuídos na qual diferentes subconjuntos de nós contém cópias parciais dos dados a partir de alguma regra de particionamento. Neste caso, apenas os nós selecionados mantêm uma cópia dos dados e o restante dos nós acessa os dados por meio desses nós designados. Isso pode reduzir o custo associado à replicação de dados, pois a quantidade de armazenamento e largura de banda necessária é significativamente menor em comparação com a replicação completa. Além disso, o custo de propagação de alterações nos dados também é reduzido, pois só precisa ser feito nos nós selecionados. Ao contrário da replicação completa, a replicação parcial visa equilibrar disponibilidade de dados, desempenho e custo, replicando os dados apenas onde são necessários. Essa estratégia pode ser útil em cenários em que o volume de dados é muito grande para que sejam replicados integralmente para todos os nós ou que outros requisitos, como latência, sejam restritivos (FOUTO; LEITAO; PREGUICA, 2018).

No entanto, a replicação parcial também pode apresentar desafios adicionais, como a necessidade de gerenciar quais nós mantêm uma cópia dos dados e garantir que os nós selecionados estejam sempre disponíveis. Em alguns casos, a replicação parcial também pode levar à redução da disponibilidade de dados, pois o sistema pode não funcionar se os

nós selecionados ficarem indisponíveis. Apesar desses desafios, a replicação parcial costuma ser considerada uma alternativa viável à replicação total em determinados aplicativos, como sistemas distribuídos de grande escala, nos quais o volume e o custo de transmissão dos dados são fatores significativos (SAITO; SHAPIRO, 2005).

2.3.2 Arquitetura de soluções de replicação

Em protocolos de replicação como C³ (FOUTO; LEITAO; PREGUICA, 2018) e *ECO SYNC TREE* (VIEIRA, 2021), este último já projetado para CRDTs, os metadados utilizados para garantir a entrega causal de mensagens são gerenciados por uma camada independente da que efetivamente armazena os dados. Na prática, esta arquitetura implica em vantagens como:

1. Isolamento da lógica de implementação da causalidade a camada apropriada, permitindo o uso de diferentes sistemas de armazenamento;
2. Especialização da camada que garante causalidade reduzindo metadados e otimizando a performance.

Essa estratégia de modularização pode ser implementada para diferentes protocolos, como descrito a seguir.

2.3.3 Protocolos e Estratégias de propagação

Quando uma operação ocorre localmente em um nó, essa mudança ou o novo estado precisam ser comunicados aos outros membros. A politica de propagação define como esta mensagem será comunicada e com quais restrições, por exemplo, se toda mensagem de atualização deve ser obrigatoriamente iniciada por um único nó (*master-backup* ou *multi-master* centralizado) ou múltiplos nós (*multi-master*) (SAITO; SHAPIRO, 2005).

Os nós precisam estar conectados de alguma forma para poder se comunicar. Porém, a topologia da rede real pode ser sobreposta por diferentes topologias virtuais especializadas a nível de aplicação (*overlays*). Tipicamente, os membros do sistema só precisam estar cientes de seus vizinhos conforme o disposto pela camada sobreposta. No caso de *overlays* estruturadas, uma relação entre os nós deve ser estabelecida previamente. Por exemplo, se há disposição dos nós em uma árvore segundo determinada regra, esta informação e as características da rede podem ser exploradas pela aplicação (LEITÃO, 2012).

Para o contexto deste trabalho são apresentados os protocolos de disseminação de mensagens Plumtree e VCube-PS, descritos a seguir.

2.3.3.1 Plumtree

O protocolo *Plumtree* é não estruturado e capaz de otimizar a disseminação de mensagens em ambientes baseados em *gossip*. A sua implementação é totalmente descentralizada e possui mecanismos de tolerância a falhas (LEITAO; PEREIRA; RODRIGUES, 2007). Ele é a base para uma série de implementações de sistemas de replicação que usam CRDTs, como os apresentados por Vieira (2021) e Meiklejohn e Van Roy (2015).

O algoritmo funciona realizando um ciclo de configuração que ocorre apenas quando necessário: após a saída ou entrada de um novo nó no conjunto. Durante a transmissão desta mensagem é identificado o caminho traçado por ela, ou seja, quais foram os nós da rede envolvidos na disseminação, determinando, a partir disto, uma árvore que atua como uma *overlay* da topologia inicial. Nesta fase, são identificados também os caminhos redundantes (LEITAO; PEREIRA; RODRIGUES, 2007).

Uma vez que a *overlay* está definida, o protocolo faz uso de duas estratégias de *gossip*:

Eager push gossip Envia as mensagens pela árvore estabelecida na última configuração.

Lazy push gossip Utiliza das conexões redundantes entre os nós para enviar uma pequena mensagem de controle capaz de validar o recebimento da mensagem principal. Caso o nó alvo não receba a mensagem principal dentro de um *timeout* pré estabelecido, ele pode solicitá-la (*pull request*) por este link alternativo. Este mecanismo de *push-pull* introduz alta disponibilidade e uma maneira de reparar a árvore

O EcoSyncTree (VIEIRA, 2021), um sistema de replicação para CRDTs, utiliza do Plumtree como referência para difusão de mensagens.

2.3.3.2 VCube-PS

O VCube (DUARTE; BONA; RUOSO, 2014) é um algoritmo de diagnóstico distribuído que constrói uma topologia de sobreposição baseado em hipercubo, organizando os nós do sistema em grupos progressivamente maiores. O modelo possui importantes propriedades logarítmicas, como a quantidade de vizinhos, distância entre dois nós e número de testes, mesmo na presença de falhas. Com base na topologia mantida pela VCube, árvores de difusão pode ser dinamicamente criadas e reorganizadas no caso de falha do processo, como explorado nos trabalhos de Rodrigues, Jr e Arantes (2014), Rodrigues, Arantes e Duarte (2014), Rodrigues et al. (2017), Jeanneau et al. (2017), Rodrigues et al. (2018).

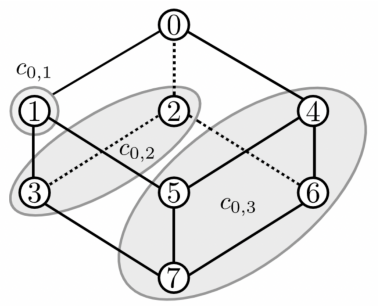
Para estabelecer a rede de sobreposição baseada em hipercubo, um nó de índice i agrupa todos os outros $N-1$ membros do sistema em $d = \log_2 N$ *clusters* de tamanho

similar $1..d$. A lista de nós em cada *cluster* s é definida por uma função $c_{i,s}$ abaixo, onde $\oplus$ é o operador exclusivo lógico *bitwise* XOR (DUARTE; BONA; RUOSO, 2014).

$$c_{i,s} = i \oplus 2^{2-1} || c_{i \oplus 2^{s-1},k} | k = 1, ..., s-1$$

A Figura 3 mostra um exemplo da organização hierárquica do VCube para $N = 8$ e $i = 0$.

Figura 3 – Hipercubo virtual com três dimensões para o nó $i = 0$ (de Araujo et al., 2019)



A Figura 4 apresenta a composição de todos os $c_{i,s}$ para $N = 8$ elementos, isto é, um VCube de três dimensões. Como exemplo, a coluna do processo 0 ($c_{0,s}$) indica que ele se conecta aos processo 1 no *cluster* $s = 1$, ao processo 2 no *cluster* $s = 2$ e ao processo 4 no *cluster* $s = 3$, que são os primeiros elementos na ordem definida pela função.

Figura 4 – Organização hierárquica do VCube representado pela tabela $c_{i,s}$ (de Araujo et al., 2019)

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

O sistema pub-sub baseado em tópicos VCube-PS (ARAUJO et al., 2017; de Araujo et al., 2019) explora essas propriedades para alcançar uma eficiente disseminação de mensagens, tolerância a falhas e entrega causal de mensagens através do uso de barreiras causais. A solução é particularmente eficaz em cenários de *hot topics*, que são comuns em grandes sistemas da Internet.

No VCube-PS, cada tópico arranja seus membros em uma árvore que é um *overlay* da topologia real. Somente os membros de um tópico recebem suas mensagens, e os retransmissores, quando existem, são temporários. Além disso, todo nó pode atuar como a fonte de uma mensagem. Neste sistema, as informações sobre membros e mensagens publicadas são enviadas aos *subscribers* de um tópico por meio de árvores dinamicamente

construídas, cuja raiz está no gerador da mensagem. A ordem causal é observada para um determinado tópico quando as mensagens são entregues.

O VCube-PS usa de barreiras causais para assegurar a entrega causal de operações. Uma das vantagens desse modelo em comparação por exemplo a relógios vetoriais é que não é sempre necessária a transmissão de todos os identificadores do estado do relógio para cada um dos membros do sistema. A dependência de mensagens é estabelecida de forma direta o que comprime o espaço utilizado para metadados.

Destaca-se que o protocolo *Plumtree* se trata de um sistema construído sobre redes não estruturadas mas que estabelece regras de dispersão, através de árvores, em *runtime*. Já o protocolo VCube-PS é estruturado e, portanto, estabelece regras sobre a disposição dos nós já no momento de criação. Ambos apresentam garantias de entrega causal.

3 Solução Proposta

Neste trabalho é apresentado o VCube-Sync, um sistema composto por N réplicas sem restrição de localização. Membros são considerados estáveis, mas o sistema é tolerante a falhas transientes (*transient failures*), como contenção na rede, através do uso de um canal *reliable* e *retries*. O sistema suporta replicação parcial e, portanto, diferentes nós podem possuir conjuntos de dados diferentes em um mesmo intervalo de tempo.

A avaliação do sistema foi realizada através de simulações representando redes com diferentes número de nós e combinações de *publishers* e *subscribers*, os resultados foram comparados com o estado da arte em replicação de CRDTs.

3.1 Arquitetura da Aplicação

O estado da aplicação é armazenado em forma de CRDTs, possibilitando operações concorrentes livres de conflito. A sincronização é feita transmitindo ou uma representação serializada da operação, no caso de CRDTs baseados em operações, ou o estado completo de um determinado *dataset*, no caso de CRDTs baseados em estado.

A aplicação é construída em módulos independentes inspirados por Vieira (2021) e Fouto et al. (2022):

membership Determina a conexão de rede real entre os nós;

broadcast Aplica estratégias de difusão, como *overlays* e determinação de árvores de difusão, e garante pré-condições como causalidade;

replication Que fornece uma interface ao *data store* de CRDT;

application Mantém o estado do data-store usando CRDTs.

A Figura 5 apresenta o relacionamento entre cada um dos módulos. As setas indicam comunicação direta em cada módulo. Isto é, por exemplo, o módulo de *Membership* se comunica apenas com o módulo de *Broadcast*, já o módulo de Broadcast pode se comunicar tanto com o módulo de membership quanto com o de *Replication*. A comunicação implementada se baseia no envio de mensagens, já que permite maior flexibilidade entre execução dos módulos que podem estar em diferentes *threads* (como neste trabalho) ou até mesmo potencialmente em diferentes máquinas.

As operações são transmitidas apenas a nós interessados (*subscribers*) numa determinada partição dos dados (tópico). Cada transmissão ocorre através de uma árvore

Figura 5 – Arquitetura da aplicação proposta.

hierárquica de difusão baseada em hipercubo, composta somente por *subscribers* daquele tópico, que tem como raiz o nó que originou a operação. O protocolo assegura que mensagens são entregues em ordem causal utilizando barreiras causais, mensagens recebidas fora de ordem são mantidas em um *buffer* até que as pré-condições sejam satisfeitas.

A seção a seguir descreve os conceitos empregados no VCube-Sync como a organização dos nós, CRDTs baseados em operação e sua implementação, assim como suas diferenças em relação a protocolos similares como o VCube-PS.

3.1.1 Interface do sistema e Difusão de mensagens

O protocolo de replicação do VCube-Sync é baseado no sistema pub-sub em tópicos VCube-PS, que faz uso das propriedades de hipercubos para distribuir mensagens de maneira eficiente, implementar tolerância a falhas, suportar dinâmicas de entrada e saída de nós, e garantir a ordenação causal da entrega das mensagens por meio do uso de barreiras causais. Este sistema é particularmente efetivo em situações em que existe um grande volume de tráfego em uma pequena parcela dos tópicos, chamados de *hot-topics*, o que é comum em sistemas de grande escala na internet (de Araujo et al., 2019).

Quando um nó i do sistema inicia a transmissão de uma mensagem m, ela será transmitida por uma árvore de transmissão hierárquica que possui i como raiz e é composta por nós que são *subscribers* de um tópico t. Na arquitetura do VCube-Sync, a camada de *broadcast* é responsável por manter o mapa do relacionamento entre nós e tópicos. O tópico corresponde a chave de replicação parcial que mapeia 1:1 com um CRDT armazenado. Um único nó pode fazer parte de múltiplos tópicos, e portanto, armazenar múltiplos itens.

Os nós podem se tornar assinantes de um determinado tópico t ou cancelar sua assinatura, e um nó que é um assinante de um tópico t pode publicar uma mensagem m para todos os outros assinantes de t. Para isso, o protocolo de replicação VCube-Sync expõe uma interface que consiste em três métodos - SUBSCRIBE(t), UNSUBSCRIBE(t) e PUBLISH(t,m) - descritos no Algoritmo 3. Estas operações geram mensagens do tipo SUB, UNS ou PUB, respectivamente, que são disseminadas usando a função CO_BROADCAST, conforme descrito no Algoritmo 4. Isso garante que todos os nós, ou todos os assinantes de um determinado tópico t no caso de mensagens do tipo PUB, sejam mantidos informados sobre o estado do sistema.

A seção a seguir apresenta o mecanismo de garantia de causalidade e destaca as diferenças neste aspecto entre o VCube-Sync e o VCube-PS.

3.1.2 Ordenação Causal

Dado um tópico t e duas mensagens m e m' e a ordem parcial $\prec$ que representa um relacionamento de *happens-before*, se $m \prec m'$ então a ordenação causal da entrega de uma mensagem garante que todas as réplicas somente observarão os efeitos de m' após observarem os efeitos de m.

O VCube-Sync usa de barreiras causais para assegurar a entrega causal de operações. A vantagem desse modelo em comparação por exemplo a relógios vetoriais é que não é sempre necessária a transmissão de todos os identificadores do estado do relógio para cada um dos membros do sistema. A dependência de mensagens é estabelecida de forma direta o que comprime o espaço utilizado para metadados.

Considere duas mensagens geradas pela aplicação, m e m', publicadas para um tópico t. Se a publicação de m precede causalmente a publicação de m', e não existe nenhuma mensagem m'' em que a publicação de m causalmente preceda publicação de m'' enquanto simultaneamente m'' causalmente preceda a publicação de m', então m é um predecessor imediato, ou dependência direta, de m'. O conjunto de mensagens que são dependências diretas de m determina a barreira causal (cb_m).

Uma vez que a barreira causal armazena apenas dependências diretas, sempre que uma nova mensagem m é enviada, a barreira anterior é reiniciada vazia (*garbage collected*), removendo metadados desnecessários e reduzindo a redundância causal já que todas as mensagens predecessoras seriam dependências indiretas de m.

A implementação da barreira causal do VCube-Sync é baseada no algoritmo implementado pelo VCube-PS.

3.1.3 Algoritmos

O VCube-Sync é baseado no sistema *publish-subscribe* VCube-PS, apresentando a mesma interface de aplicação. Apesar disto, conforme citado nas seções pertinentes, algumas alterações foram realizadas ao protocolo de difusão considerando que esta versão do VCube-Sync considera um sistema estável e livre de falhas.

Cada mensagem m é identificada por um identificador único (id) por sua fonte (s) e um contador (c), e também contém informações sobre o tópico (t) ao qual está relacionada. Existem três tipos de mensagens: SUB (assinatura), UNS (cancelamento da assinatura) e PUB (publicação). O conteúdo das mensagens varia de acordo com o tipo: as mensagens SUB e UNS não contêm informações no campo de dados, enquanto que as mensagens PUB contêm a própria mensagem de replicação (dados), bem como a barreira causal (cb).

O estado da camada de difusão do VCube-Sync de cada nó i é mantido pelas seguintes variáveis:

- *counter*: Implementa um relógio lógico que é incrementado sempre que um evento é produzido pelo nó i
- $view[t]$: Armazena a ultima operação de *subscription* ou *unsubscription* por tópico por nó. Usado para determinar se um nó j é atualmente *subscriber* de um tópico.
- $causal\_barrier[t]$: Representa a barreira causal que mantém uma referência a todas as mensagens que são predecessoras da próxima mensagem que será gerada por i para o tópico t. A barreira é composta por um conjunto de identificadores $\langle s, c \rangle$ (fonte, e contador).
- $receivedIds$: Armazena o identificador das mensagens já recebidas. Utilizado para de-duplicar mensagens. Pode utilizar a tupla $\langle s, c \rangle$ ou um identificador externo.
- $delvs[t]$: Armazena a última mensagem recebida para um tópico e nó. Os elementos do conjunto são tuplas $\langle s, c \rangle$ (fonte, e contador).

O Algoritmo 3 apresenta a interface do VCube-Sync. Utilizando os métodos *Subscribe*, *Unsubscribe* e *Publish* o usuário da aplicação é capaz de gerar mensagens dos tipos SUB, UNS, e PUB respectivamente. Esta interface é a mesma do VCube-PS.

Algorithm 3 Métodos da interface da aplicação para o processo i, Adap. de de Araujo et al. (2019)

```
1:  function INIT( )
2:      counter ← 0
3:      view ← ∅

4:  function SUBSCRIBE(topic t)
5:      if ⟨i, SUB, _⟩ ∉ view[t] then
6:          view[t] ← {⟨i, SUB, counter⟩}
7:          CO_BROADCAST(SUB, t, _)
8:          return OK
9:      return NOK

10: function UNSUBSCRIBE(topic t)
11:     if ⟨i, SUB, _⟩ ∈ view[t] then
12:         view[t] ← view[t] {⟨i, SUB, _⟩}        ▷ remove a subscription de t
13:         CO_BROADCAST(UNS, t, _)
14:         return OK
15:     return NOK

16: function PUBLISH(topic t, message data)
17:     if ⟨i, SUB, _⟩ ∈ view[t] then        ▷ somente subscribers podem publicar em t
18:         CO_BROADCAST(PUB, t, data)
19:         return OK
20:     return NOK
```

São apresentadas também duas funções auxiliares $\textsc{Cluster}(i, j)$ e $\textsc{Children}(i, t, h)$ que são utilizadas para construir as árvores de difusão hierárquicas, implementadas conforme de Araujo et al. (2019):

- $\textsc{Cluster}(i, j)$: retorna o índice s do *cluster* de um nó k que contém o nó j, $(1 \leqslant s \leqslant \log_2 N)$. Por exemplo na Figura 3, $\textsc{Cluster}(0, 1) = 1$, $\textsc{Cluster}(0, 2) = \textsc{Cluster}(0, 3) = 2$, e $\textsc{Cluster}(0, 4) = \textsc{Cluster}(0, 5) = \textsc{Cluster}(0, 6) = \textsc{Cluster}(0, 7) = 3$.
- $\textsc{Children}(i, t, h)$: retorna um conjunto que contém todos os nós virtualmente conectados a i. Um filho de i é o primeiro nó do *cluster* $c_{i,s}$ que também é um assinante de t, ou simplesmente o primeiro nó deste *cluster* sem considerar tópicos quando este não for definido. O parâmetro h representa o *cluster* pode variar de 1 a $\log_2 N$. Para $h = \log_2 N$, a função retorna o conjuntos dos primeiros filhos de i; Para qualquer outro valor $h < \log_2 N$, a função retorna apenas uma sub-conjunto de filhos, isto é, apenas aqueles em que o número de *cluster* s é menor ou igual a h $(s \leqslant h)$. Por exemplo, na Figura 3, se $t =$ '$*$', $\textsc{Children}(0, *, 3) = \{1, 2, 4\}$, $\textsc{Children}(0, *, 2) = \{1, 2\}$, e $\textsc{Children}(4, *, 2) = \{5, 6\}$. Por outro lado, se somente os nós 0, 3, e 4 tiverem assinado o tópico t_1, então $\textsc{Children}(0, t_1, 3) = \{3, 4\}$ e $\textsc{Children}(4, t_1, 2) = \varnothing$.

O Algoritmo 4 apresenta o mecanismo de difusão de mensagens para sincronização do VCube-Sync. Este algoritmo é uma adaptação do VCube-PS, tendo como diferenças (i) que o VCube-PS implementa também garantia de recepção *First In, First Out* (FIFO) *per-source* utilizando mensagens do tipo ACK e (ii) que no VCube-PS cada nó i armazena qual o contador da primeira mensagem recebida de cada nó j. Estes mecanismos são implementados para lidar com dinâmicas de entrada e saída de nós. No VCube-Sync consideramos um sistema estável, portanto estes mecanismos não são necessários e foram removidos o que aumenta a reatividade do sistema por permitir maior paralelismo na transmissão.

A emissão de uma mensagem de replicação pela camada de *broadcast* de um nó i é iniciada com a função $\textsc{Co\_Broadcast}(m)$, que cria a mensagem a ser transmitida. A função ReceiveMessage é então chamada e a partir deste ponto o comportamento é o mesmo seja para mensagens geradas localmente, seja por mensagens recebidas pela rede. Esta função inicia a propagação da mensagem para os a lista de vizinhos obtidos pela função $\textsc{Children}(i, t, h)$, e também o processo de entrega da mensagem localmente. A entrega local de mensagens do tipo SUB corresponde a adicionar a mensagem a lista *recs* e invocar a função $\textsc{CheckDelivery}(t)$, já as de outros tipo correspondem a mudanças de assinatura e portanto invocam o método $\textsc{Update}(set_1, set_2)$.

O método $\textsc{CheckDelivery}(t)$ varre o conjunto *recs[t]* que contém todas as mensagens recebidas no tópico t que ainda não foram entregues. Para cada uma, é

validado se a barreira causal é satisfeita, se sim, a mensagem é entregue pelo método Co_Deliver(m), retirada o do conjunto *recs[t]*, e tem seu identificador adicionado ao conjunto *delvs[t]* e a barreira causal deste nó i. Se não, a mensagem continua no conjunto *recs[t]* onde terá a oportunidade de ser checada novamente quando uma nova mensagem de t for recebida.

Finalmente, no contexto do VCube-Sync, os dados da mensagem correspondem a uma operação de replicação de dados. Quando uma mensagem é entregue pelo método Co_Deliver(m), o método NotifyReply(m) transmite a mensagem para a camada *Replication*, onde a operação é aplicada imediatamente com base no tópico correspondente.

3.1.4 Utilização do CRDT

O VCube-Sync utiliza CRDTs para manter seu estado local. Dado que a aplicação é modular, qualquer tipo de CRDT em que ordenação causal seja suficiente pode ser utilizado. Isso inclui CRDTs baseados em estado e a maioria dos CRDTs baseados em operações.

A camada de replicação e o protocolo de difusão do VCube-Sync atuam em conjunto para garantir que uma operação só será comunicada ao *Datastore* que armazena o ao estado se (i) a operação for iniciada pelo próprio nó, neste caso não há dependências externas, ou (ii) a operação foi iniciada por outro nó e a pré-condição de entrega de ordenação causal foi satisfeita.

Considerando a perspectiva de um cliente da aplicação interagindo com o *Datastore*, o fluxo é iniciado pela geração da operação. Ou seja, o procedimento é como a seguir:

1. O cliente utiliza a interface do *Datastore* para modificar (*update*) o estado ou criar um novo CRDT, que equivale a um novo tópico.
2. A função de interface do *Datastore* gera uma versão serializada da operação, que é então transmitida localmente para a camada de replicação.
3. A camada de replicação gera uma mensagem de replicação a partir da operação.
4. A camada de replicação utiliza o mesmo procedimento utilizado para receber mensagens de outros nós, inserindo a mensagem na fila de mensagens a serem entregues e comunicando à camada de *broadcast* para quais outros membros da rede os dados devem ser transmitidos.

Para implementação do *Datastore* e da camada de replicação, foi utilizado como referência o trabalho de Vieira (2021), sendo a principal alteração a adição de suporte para replicação parcial em que o tópico atua como chave de replicação parcial, isto é,

Algorithm 4 Difusão causal do VCube-Sync para o processo i

```
1: function Init( )
2:     ∀t ∈ TOPICS: view[t] ← ∅; recs[t] ← ∅; delv[t] ← ∅;
3:     counter ← 0; receivedIds ← ∅; causal_barrier[t] ← ∅

4: function Co_Deliver(message m)
5:     NotifyReply(m)                          ▷ Entrega a mensagem a camada de replicação

6: function Co_Broadcast(message_type type, topic t, content data)
7:     m = New(message)
8:     m.type ← type; m.s ← i; m.t ← t
9:     m.c ← counter; m.content ← data;
10:    m.cb ← causal_barrier[t]
11:    counter ← counter + 1
12:    causal_barrier[t] ← {⟨i, counter⟩}
13:    ReceiveMessage(m)

14: function ReceiveMessage(message m)
15:    if m.id ∈ receivedIds then
16:        return                               ▷ Ignora mensagens duplicadas
17:    if m.type = SUB then
18:        chd ← Children(i, ∗, Cluster(i, m.s) − 1)
19:    else
20:        chd ← Children(i, m.t, Cluster(i, m.s) − 1)
21:    for all k ∈ chd do
22:        Send(m) to p_k
23:    if ⟨i, SUB, _ ⟩ ∈ view[m.t] then          ▷ i é um subscriber de m.t
24:        if m.type = PUB then
25:            recs[m.t] ← recs[m.t] ∪ {m}
26:            receivedIds ← msgs ∪ {m.id}
27:            CheckDeliverable(m.t)
28:        else
29:            view[m.t] ← Update(view[m.t], {⟨m.s, m.type, m.c⟩})

30: function Update(set_1, set_2)
31:    for all ⟨n_1, _, rc_1⟩ ∈ set_1 do
32:        if ∃ ⟨n_1, _, rc_2⟩ ∈ set_2 then
33:            if rc_2 > rc_1 then
34:                set_1 ← set_1{⟨n_1, _, rc_1⟩}
35:            else
36:                set_2 ← set_2{⟨n_1, _, rc_2⟩}
37:    return set_1 ∪ set_2
```

38: **function** CHECKDELIVERABLE(topic t)
39: **while** ($\exists\ m \in recs[t]:$ CHECKCB$(t, m.cb) = true$) **do**
40: CO_DELIVER(m)
41: $recs[t] \leftarrow recs[t] \backslash \{m\}$
42: $delvs[t] \leftarrow delvs[t] \backslash \{\langle m.s, \_\rangle\} \cup \{\langle m.s, m.c\rangle\}$
43: $causal\_barrier[t] \leftarrow causal\_barrier[t] \setminus \{m.cb\}$
$\cup \{\langle m.s, m.c\rangle\}$

44: **function** CHECKCB(topic t, causal barrier cb)
45: **for all** $\langle s, c\rangle \in cb$ **do**
46: **if** $\exists\ \langle s', c'\rangle \in delvs[t] : s = s'$**and** $c' \geqslant c$ **then**
47: $cb \leftarrow cb \backslash \{\langle s, c\rangle\}$
48: **return** $cb = \varnothing$

existe um CRDT para cada tópico, todos os *subscribers* de um tópico t mantém uma cópia consistente a termo do objeto.

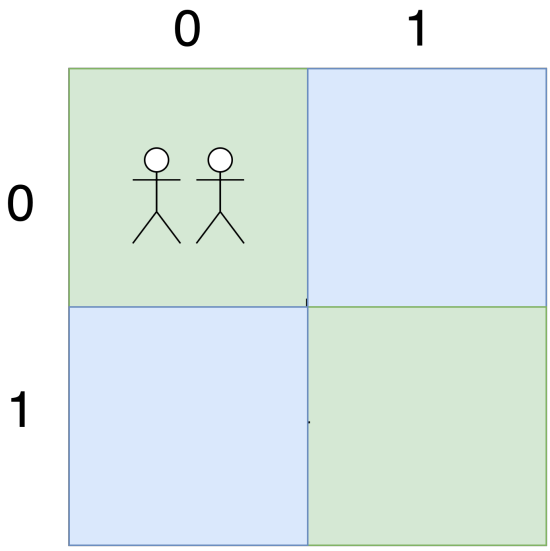
3.1.5 Exemplos de Aplicações

Diversas aplicações já utilizam de CRDTs para manutenção de um estado compartilhado correto como citado na seção 2.2.4. Além destes, alguns exemplos de aplicações práticas são como a seguir:

Redes sociais como Twitter são baseadas em mensagens (*tweets*) publicadas por usuários em suas contas que podem ser respondidos por outros usuários, estas respostas por sua vez também são *tweets*. No contexto do Vcube-Sync a aplicação pode ser modelada como a seguir: cada tópico é atribuído a um *tweet* raiz, ou seja, um *tweet* que não é uma resposta a um *tweet* anterior. O estado CRDT deste *tweet* é modelado como um Conjunto Observado-Removido (*ORSet*) (SHAPIRO et al., 2011a). Cada mensagem PUB representa ou a adição ou a remoção de um comentário sobre o tópico. A leitura do armazenamento de dados deve retornar o tópico inteiro do *tweet* e suas respostas de acordo com a visão do nó lido.

Outro exemplo de cenário seria de jogos online *on-line multiplayer*, como mencionado por de Araujo et al. (2019). Suponha que os jogadores possam se mover através de uma grade 2D em forma de tabuleiro representada por pares de coordenadas como demonstrado na Figura 6. Cada tópico é um azulejo nesta grade. Usando um CRDT ORSet, podemos acompanhar quais usuários estão atualmente ativos em uma determinada área (tópico). Cada mensagem PUB significaria que um usuário está entrando ou saindo da área. A leitura do armazenamento de dados devolveria o conjunto de jogadores que estão ativos em uma determinada área. Este cenário é similar ao conceito de presença ao vivo implementado pelo *framework* Phoenix (MCCORD, 2022).

Figura 6 – Mapa 2D dividido em seções identificadas por pares de coordenadas.



4 Resultados

Para avaliar o desempenho do VCube-Sync em diferentes cenários foram conduzidos experimentos na plataforma Grid'5000 [1], na região 'nancy' no *cluster* 'gros'. Neste ambiente cada *host* está equipado com um processador Intel Xeon Gold 5220 com 18-núcleos, 96 GB de memória RAM, conectados por uma rede com capacidade de 2x25 Gbps. Com base em Vieira (2021) e Fouto et al. (2022) foi utilizado o Docker Swarm para arranjo dos experimentos, executando portanto um nó do protocolo de replicação em cada *container*. Para arranjo da simulação e comunicação entre as camadas da aplicação e entre nós foi utilizado o *framework* Babel desenvolvido pelo laboratório NOVA LINCS da Universidade NOVA de Lisboa (FOUTO et al., 2022).

Afim de determinar o desempenho de diferentes protocolos de replicação para CRDTs baseados em operações foram comparadas as seguintes implementações:

- VCube-Sync. Cada tópico representa uma chave de particionamento, qualquer nó pode atuar como raiz e os nós são arranjados como hipercubo.
- EcoSyncTree. Disposição não-estruturada com otimização baseada nos protocolos *Plumtree* e *HyParView* (VIEIRA, 2021).

Em todos os cenários, os nós se encontram completamente conectados, sendo as topologias implementadas como um *overlay* da rede real.

Assim como em Vieira (2021), foi utilizado o *Linux Traffic Control* (Linux TC) para determinar latências entre cada conjunto de nós. As latências são definidas como uma matriz de dimensão N. Para manter os experimentos comparáveis os valores utilizados emulam a distância euclidiana entre os nós dentro de um intervalo aproximado de [10,100] milissegundos. Por exemplo, no cenário em que há 200 nós, a matriz é de dimensões 10x20 (200), a latência entre os nós 1 e 5 é de 25ms, 1 e 10 é de 50ms, entre 1 e 20 é de 95ms. As mesmas configurações de latência foram utilizadas para todos os experimentos.

Os experimentos foram conduzidos para diferentes números de nós: 50, 100 e 200; e diferentes arranjos de *publishers* e *subscribers*, incluindo cenários de replicação total e parcial. Estes cenários estão agrupados em dois grupos: aqueles em que há somente um *publisher* e outros em que há múltiplos *publishers*. Cada experimento foi executado 3 vezes e, em experimentos em que uma amostra de nós deve ser obtida, a seleção foi feita via amostragem uniforme. Foi utilizado um CRDT baseado em operações do tipo registro no

[1] Os experimentos deste trabalho foram conduzidos utilizando o *testbed* Grid'5000, que é mantido por um *scientific interest group* do Inria e inclui CNRS, RENATER e diversas outras universidades e organizações (https://www.grid5000.fr).

qual cada operação tem de tamanho de 1.024 bytes. Em todos os experimentos o número de nós por máquina do ambiente de testes foi o mesmo.

4.0.1 Cenários avaliados

Foram avaliadas a latência média na entrega das mensagens e o uso de banda sob diferentes configurações de números de nós e arranjos de tópicos. Os eventos da aplicação são registrados pelo *logger* Log4J. Após a geração dos *logs* no ambiente distribuído estes são analisados por outros *scripts* que emitem os resultados finais.

Para cada um dos protocolos avaliados: VCube-Sync e EcoSyncTree as seguintes configurações foram executadas (Tabela 1):

Tabela 1 – Configurações dos testes realizados.

Tipo	N de *Publishers*	N de *Subscribers*
Único *publisher*	1	25%
Único *publisher*	1	100%
Múltiplos *publishers*	25% dos membros	100%
Múltiplos *publishers*	100% dos membros	100%

4.0.2 Rotina de experimentação

Cada experimento segue a seguinte sequência de passos:

1. Preparar e executar todos os nós via container Docker;
2. Iniciar os protocolos de *membership* e aguardar sua estabilização;
3. Iniciar o envio de mensagens do tipo SUB manifestando interesse em determinados tópicos conforme o arranjo de cada experimento;
4. Iniciar o envio de mensagens de replicação por 400 segundos, sendo uma enviada por segundo por *publisher*
5. Após terminar o envio, os nós continuam disponíveis por até 5 minutos para receber mensagens em trânsito ou geradas por nós que iniciaram o passo 3 em ponto mais avançado do tempo.

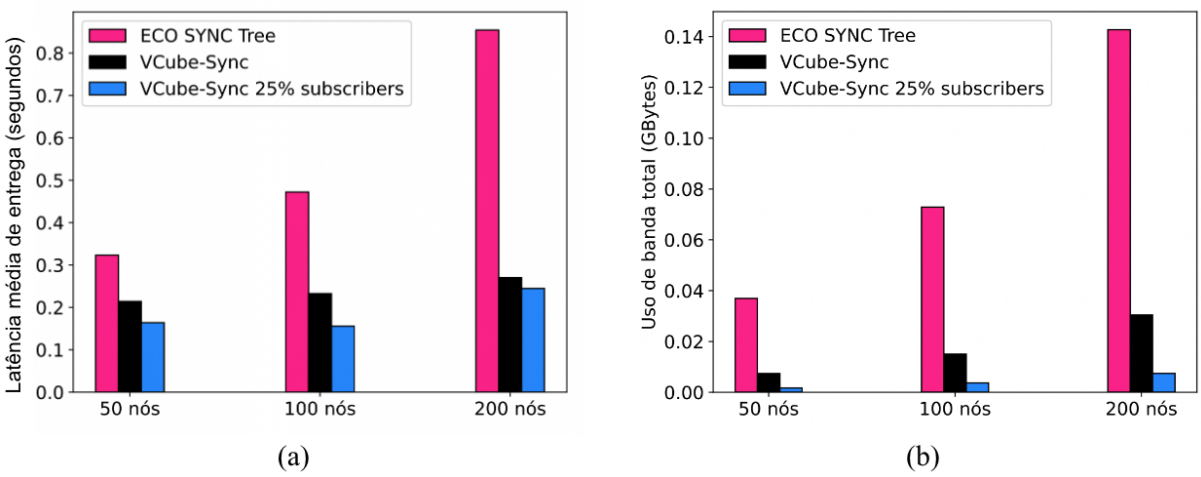
4.0.3 Um único *Publisher*

Este conjunto de experimentos avalia o desempenho de replicação em um cenário em que apenas um nó publica mensagens. Desta forma, é possível também observar o comportamento de quando não há dependência causal.

A Figura 7(a) apresenta a latência média de entrega quando o número de nós que demonstram interesse no tópico é de 100% e de 25%. Este último somente se aplica ao VCube-Sync, já que o protocolo ECOSyncTree não é capaz de realizar replicação parcial e, portanto, todos os nós recebem uma mensagem de replicação mesmo que não estejam interessados em determinado tópico.

A latência média para 200 nós do ECOSyncTree para replicação total é de 0.854 segundos, um valor 3,16x maior do que o observado no VCube-Sync neste mesmo arranjo, e 3,49x maior do que cenário de replicação parcial. Além disso, o efeito da replicação parcial pode também ser observado no número total de bytes transmitidos para cada simulação apresentado na Figura 7(b). Para 200 nós, o número total de bytes transmitidos pelo ECOSyncTree foi de 142,67 MBytes, um valor 4,68x maior do que utilizando o VCube-Sync no arranjo de replicação total e 19.14x maior no cenário de replicação parcial.

Figura 7 – Resultados para um único *Publisher*. (a) Latência média de entrega de mensagens em segundos por protocolo e número de nós; (b) Uso de banda total do experimento em GB por protocolo e número de nós.



Estes resultados indicam que a fator de redução causado pela replicação parcial se comportou conforme o esperado, visto que somente os nós interessados nos tópicos recebem e transmitem mensagens. A redução do número de bytes acontece linearmente na mesma proporção. Destaca-se ainda, que o VCube não requer o envio de mensagens para manutenção da topologia. Portanto, a taxa de transmissão de bytes tende a zero se mensagens de replicação não estão em trânsito.

Por outro lado, o EcoSyncTree utiliza de mensagens para manutenção da topologia o que torna o caso de um único *publisher* pouco otimizado, já que o custo de manutenção corresponde a uma parcela de quase 50% do taxa de uso de rede, como apresentado na Figura 8. Além disso, o número de de mensagens duplicadas observadas pelos nós utilizando o ECOSyncTree com apenas um *publisher* foi de 0 a 12%. Enquanto que para o VCube-Sync nunca há mensagens duplicadas quando não há falhas.

Figura 8 – Uso de banda por protocolo em MB por segundo para 200 nós.

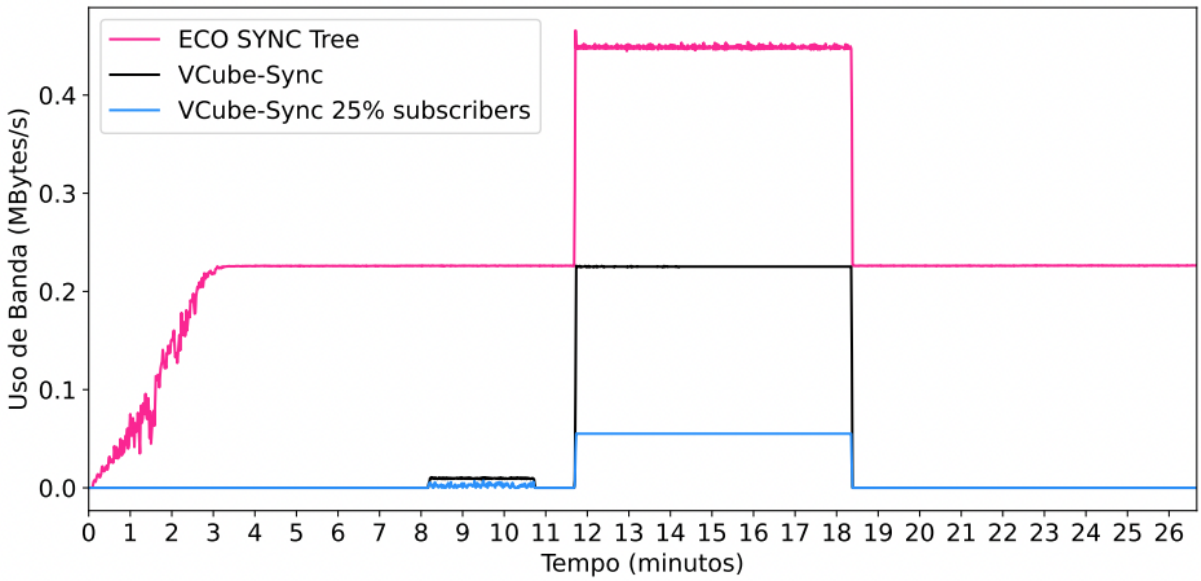


Tabela 2 – Resultados de latência com apenas um *publisher* para diferentes números de nós, algoritmos de difusão e fração de *subscribers*.

Número de Nós	Algoritmo	% de *subscribers*	Latência Média (ms)	Latência 95th percentil (ms)	Latência 99th percentil (ms)
50	VCube-Sync	25%	163,87	182,00	183,00
50	VCube-Sync	100%	214,06	215,00	215,01
50	ECOSyncTree	100%	323,24	348,00	349,00
100	VCube-Sync	25%	155,85	186,00	186,00
100	VCube-Sync	100%	232,46	234,00	235,00
100	ECOSyncTree	100%	472,08	556,00	557,00
200	VCube-Sync	25%	244,77	264,00	265,00
200	VCube-Sync	100%	270,31	272,00	274,00
200	ECOSyncTree	100%	854,34	2.787,40	3.177,60

Para ambos os protocolos, para todos números de nós no cenário de 100% de *subscribers*, o número total de mensagens transmitidas pela rede foi o mesmo. No entanto, o ECOSyncTree apresentou um uso maior de banda em comparação com o VCube-Sync. Isso ocorre porque o ECOSyncTree possui um ciclo de sincronização que é executado continuamente, mesmo quando nenhum nó foi identificado como falho. Em outras palavras, a presença de um mecanismo extra de sincronização tem impacto na utilização da banda da rede pelos dois protocolos.

Ainda com base nos resultados da Tabela 3, fica evidente que o algoritmo VCube-Sync tem um desempenho significativamente melhor do que o algoritmo ECOSyncTree em termos de latência. Em comparação com o ECOSyncTree, o VCube-Sync é capaz de alcançar latências muito mais baixas em todos os arranjos de nós e porcentagens de assinantes testados. Também é interessante notar que à medida que o número de nós

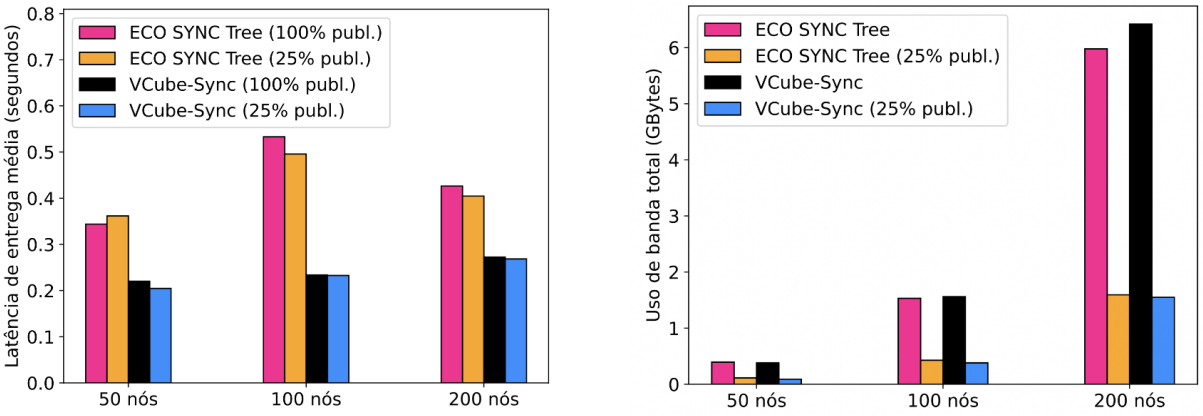
aumenta, a latência de ambos os algoritmos aumenta. Entretanto, o VCube-Sync ainda é capaz de manter uma latência relativamente baixa em comparação com o ECOSyncTree mesmo na presença de maior número de nós, no cenário de 200 nós e 100% de *subscribers* por exemplo a latência média do ECOSyncTree aferida foi 216% maior. Isto sugere que o VCube-Sync pode ser mais adequado para sistemas de maior escala que requerem mais nós. Além disso, a latência do percentil 99 e do percentil 95 para VCube-Sync são consistentemente mais baixas do que as do Eco-Sync. O que indica que o VCube-Sync apresenta uma distribuição mais uniforme e confiável de latência.

4.0.4 Múltiplos *Publishers*

Neste conjunto de experimentos todo os nós são *subscribers* de um único tópico e o número de *publishers* é de 25% e 100% para cada um dos protocolos. Este arranjo permite avaliar também o impacto do balanceamento de carga das mensagens em cada protocolo. No arranjo em que apenas 25% dos membros são *publishers*, a seleção é aleatória e uniforme.

Na Figura 9(a), observa-se que para 200 nós, quando há 100% de *publishers*, a latência média de entrega de mensagens do ECOSyncTree é 1,56x a apresentada pelo VCube-Sync. Quando o arranjo contém 25% de *publishers* a diferença é praticamente a mesma, em 1.51x. Este resultado demonstra a eficiência do VCube-Sync em latência mesmo na presença de grande número de nós, corroborando também com o apresentado por de Araujo et al. (2019).

Figura 9 – Resultado para Múltiplos *Publishers*. (a) Latência média de entrega de mensagens em segundos por protocolo, número de nós e fração de *publishers*; (b) Uso de banda total do experimento em Gigabytes por protocolo, número de nós e fração de *publishers*.



Já para o uso de banda, conforme a Figura 9(b), para o cenário de 200 nós e 100% de *publishers*, o VCube-Synch apresentou uma vazão total de 6,41 GB, um valor 7,39%

maior quando comparado ao ECOSyncTree. Por outro lado, para o cenário de 25% de *publishers*, o VCube-Sync demonstra uma redução de 2,79% no uso de banda total. O aumento no número de bytes no cenário de 100% é causado pelos metadados adicionais que uma mensagem pode conter no VCube-Sync, especialmente a barreira causal.

Outra observação é que, em um cenário com múltiplos *publishers*, a sobrecarga de manutenção da rede presente no ECOSyncTree é mínima, sendo a taxa de envio de mensagens para manutenção da rede no cenário com 200 nós e 100% de *publishers* de aproximadamente 10 KBytes/s, enquanto a taxa máxima de vazão do experimento foi de 48,63 MBytes/s.

Para ambos os protocolos, para todo número de nós no cenário de 100% de *publishers*, o número total de mensagens transmitidas foi 3% maior no VCube-Sync. A hipótese é de que, devido a sincronização intermediária realizada pelo ECOSyncTree, algumas das mensagens de replicação são recebidas após uma sincronização que já possui este evento. Uma vez que mensagens redundantes não são retransmitidas, um pequeno número de operações de replicação acabam não percorrendo toda a árvore, não sendo computadas como mensagens individuais.

Tabela 3 – Resultados de latência com diferentes conjuntos de nós publicando mensagens e algoritmos de difusão em cenários que todos os nós são *subscribers*.

Numero de Nós	Algoritmo	% de Publishers	Latência Média (ms)	Latência 95th percentil (ms)	Latência 99th percentil (ms)
50	VCube-Sync	25%	204,66	249,00	268,00
50	VCube-Sync	100%	220,10	268,00	270,00
50	ECOSyncTree	25%	361,40	468,00	517,00
50	ECOSyncTree	100%	343,82	430,00	495,00
100	VCube-Sync	25%	232,64	283,00	272,00
100	VCube-Sync	100%	233,70	281,99	283.00
100	ECOSyncTree	25%	495,61	609,00	1.611,76
100	ECOSyncTree	100%	532,93	661,00	695,99
200	VCube-Sync	25%	268,39	311,00	329,00
200	VCube-Sync	100%	272,47	329,00	331,00
200	ECOSyncTree	25%	404,45	491,00	509,00
200	ECOSyncTree	100%	426,19	560,99	600,00

Dos resultados na Tabela 3, pode-se verificar que a VCube-Sync tem um bom desempenho em termos de latência quando comparada à ECOSyncTree. A latência da VCube-Sync é consistentemente menor do que a da ECOSyncTree, especialmente em cenários onde todos os nós são *publishers*. Por exemplo, no caso em que existem 50 nós e todos eles publicam mensagens, a latência do VCube-Sync é 220,10 ms, enquanto a do ECOSyncTree é 343,82 ms. Da mesma forma, no caso em que existem 100 nós e

todos eles publicam mensagens, a latência do VCube-Sync é de 233,70 ms, enquanto a do ECOSyncTree é de 532,93 ms.

Além disso, podem observar que a porcentagem de *publishers* tem um efeito significativo sobre a latência de ambos os algoritmos. A medida que a porcentagem de *publishers* aumenta, a latência também aumenta. Por exemplo, no caso em que existem 50 nós e apenas 25% deles publicam mensagens, a latência do VCube-Sync é de 204,66 ms, enquanto a do ECOSyncTree é de 361,40 ms. Entretanto, quando todos os 50 nós publicam mensagens, a latência do VCube-Sync aumenta para 220,10 ms, enquanto que a do ECOSyncTree aumenta para 343,82 ms.

Estes resultados demonstram que o VCube-Sync apresenta uma vantagem significativa na latência de entrega de mensagens, embora com um pequeno aumento no número de bytes transmitidos para replicação em cenários onde apenas um subconjunto de nós publica mensagens.

5 Conclusão

Neste estudo, apresentamos o VCube-Sync, um algoritmo de replicação parcial destinado a estruturas de dados livres de conflitos (CRDTs). Este algoritmo explora a sinergia entre sistemas de *publisher-subscriber* baseados em tópicos e a sincronização de sistemas por meio da difusão de mensagens de replicação. Ao aproveitar as características dos hipercubos como topologia para difusão causal, como também já investigado pelo VCube-PS anteriormente, nosso sistema proposto demonstra a capacidade de manter bases de dados sincronizadas de maneira eficiente, tanto em termos de latência quanto de uso de largura de banda, em comparação com o estado da arte em sistemas similares.

Quando combinado com um *datastore* baseado em CRDTs, o VCube-Sync possibilita o desenvolvimento de bases de dados consistentes e corretas a longo prazo. As garantias de causalidade oferecidas pelo protocolo, por meio do uso de barreiras causais, ainda permitem a utilização de CRDTs baseados em operações, que se destacam em termos de eficiência no tamanho das mensagens.

Realizamos uma avaliação experimental do algoritmo proposto em diversos cenários, utilizando a plataforma Grid'5000. Os resultados foram comparados com o EcoSyncTree, outro algoritmo de replicação para CRDTs. No contexto avaliado, o VCube-Sync apresentou resultados superiores em termos de latência na entrega de mensagens de replicação e, nos cenários de replicação parcial, um uso mais eficiente da largura de banda.

Como parte desta pequisa, foram feitas as seguintes publicações:

- GALESKY, Leonardo de Freitas; RODRIGUES, Luiz Antonio. Efficient CRDT Synchronization at Scale using a Causal Multicast over a Virtual Hypercube Overlay. In Proceedings of the 11th Latin-American Symposium on Dependable Computing (LADC '22) – Student Forum, 2022, Fortaleza, Brazil. Association for Computing Machinery, New York, NY, USA, 2023. 84–88. DOI: <https://doi.org/10.1145/3569902.3569948>

- GALESKY, Leonardo de Freitas; RODRIGUES, Luiz Antonio. Sincronização Eficiente de CRDTs em Escala Utilizando uma Solução Hierárquica de Publish–Subscribe. In: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 41. , 2023, Brasília/DF. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2023 . p. 463-475. ISSN 2177-9384. DOI: <https://doi.org/10.5753/sbrc.2023.484>.

- GALESKY, Leonardo de Freitas; RODRIGUES, Luiz Antonio; DUARTE JR., Elias P.; ARANTES, Luciana. Efficient Synchronization of CRDTs using VCube-PS. In: 12th Latin-American Symposium on Dependable and Secure Computing (LADC),

2023, La Paz, Bolívia. Proceedings [...]. New York, ACM, 2023. p. 1-10. ISBN 979-8-4007-0844-2/23/10. DOI: <https://doi.org/10.1145/3615366.3615421>.

5.1 Trabalhos Futuros

Embora o VCube-Sync apresente limitações em relação à tolerância a falhas, é viável propor um plano claro para superar essa limitação. O primeiro passo nesse sentido envolve a sincronização do estado no momento em que um nó é identificado como reentrante. Uma estratégia possível é a troca de barreiras causais entre vários nós vizinhos, que permite determinar o estado mais atual e modelá-lo como um Delta-State-CRDT.

O segundo passo consiste em explorar a topologia estruturada do hipercubo para melhorar a eficiência do sistema. Em vez de utilizar mensagens de ACK e FIFO para garantir a entrega de mensagens, é possível explorar o mencionado mecanismo de sincronização para possibilitar entregas mais flexíveis, incluindo um mecanismo de sincronização periódica por meio de deltas.

Ainda como trabalhos futuros, considerando que o envio de múltiplas pequenas mensagens pode apresentar maior custo de comunicação do que o envio de um menor número de grandes mensagens, algoritmos de agregação por causalidade poderiam ser explorados, como apresentado por Rodrigues et al. (2018). Dessa forma, o VCube-Sync poderia agrupar as mensagens em ordem causal para minimizar o custo de comunicação. Cabe ressaltar que, no caso do VCube-Sync, as mensagens já são visíveis no *datastore* somente em ordem causal, o que significa que o não recebimento não impactaria na deterioração da experiência do usuário.

Referências

AKKOORATH, A. B. D. D. *Antidote: the highly-available geo-replicated database with strongest guarantees*. SyncFree, 2016. Disponível em: <https://pages.lip6.fr/syncfree/attachments/article/59/antidote-white-paper.pdf>. Citado na página 24.

AKKOORATH, D. D. et al. Cure: Strong semantics meets high availability and low latency. In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. Nara, Japan: IEEE, 2016. p. 405–414. Citado 2 vezes nas páginas 13 e 16.

ALMEIDA, P. S.; SHOKER, A.; BAQUERO, C. Efficient state-based crdts by delta-mutation. In: BOUAJJANI, A.; FAUCONNIER, H. (Ed.). *Networked Systems (NETYS 2015)*. Cham: Springer International Publishing, 2015. p. 62–76. ISBN 978-3-319-26850-7. Citado na página 23.

ALMEIDA, P. S.; SHOKER, A.; BAQUERO, C. Delta state replicated data types. *Journal of Parallel and Distributed Computing*, v. 111, p. 162–173, jan 2018. ISSN 07437315. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0743731517302332>. Citado na página 24.

ARAUJO, J. P. D. et al. A publish/subscribe system using causal broadcast over dynamically built spanning trees. In: IEEE. *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. [S.l.], 2017. p. 161–168. Citado 2 vezes nas páginas 14 e 28.

BAQUERO, C.; ALMEIDA, P.; SHOKER, A. Making operation-based crdts operation-based. *Proceedings of the 1st Workshop on the Principles and Practice of Eventual Consistency, PaPEC 2014*, 04 2014. Citado na página 22.

BAQUERO, C.; ALMEIDA, P. S.; SHOKER, A. Pure operation-based replicated data types. *CoRR*, abs/1710.04469, 2017. Disponível em: <http://arxiv.org/abs/1710.04469>. Citado na página 13.

BASHO. *Riak Distributed Data Types*. Basho, 2022. Disponível em: <https://riak.com/products/riak-kv/riak-distributed-data-types/index.html?p=10947.html>. Acesso em: 15 mai. 2022. Citado na página 24.

BAUWENS, J.; BOIX, E. G. Improving the Reactivity of Pure Operation-Based CRDTs. In: *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*. New York, NY, USA: ACM, 2021. p. 1–6. ISBN 9781450383387. Disponível em: <https://dl.acm.org/doi/10.1145/3447865.3457968>. Citado na página 13.

BERNSTEIN, P. A.; DAS, S. Rethinking eventual consistency. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2013. (SIGMOD '13), p. 923–928. ISBN 9781450320375. Disponível em: <https://doi.org/10.1145/2463676.2465339>. Citado na página 20.

BIRMAN, K.; SCHIPER, A.; STEPHENSON, P. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 9, n. 3, p. 272–314, aug 1991. ISSN 0734-2071. Disponível em: <https://doi.org/10.1145/128738.128742>. Citado na página 25.

BREWER, E. CAP twelve years later: How the "rules" have changed. *Computer*, v. 45, n. 2, p. 23–29, feb 2012. ISSN 0018-9162. Citado 2 vezes nas páginas 19 e 20.

BROWN, R. et al. Riak dt map: A composable, convergent replicated dictionary. In: *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*. New York, NY, USA: Association for Computing Machinery, 2014. (PaPEC '14). ISBN 9781450327169. Disponível em: <https://doi.org/10.1145/2596631.2596633>. Citado na página 13.

de Araujo, J. P. et al. Vcube-ps: A causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, v. 125, p. 18–30, 2019. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731518307822>. Citado 8 vezes nas páginas 7, 14, 28, 31, 33, 34, 37 e 43.

DECANDIA, G. et al. Dynamo: Amazon's highly available key-value store. In: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. New York, NY, USA: Association for Computing Machinery, 2007. (SOSP '07), p. 205–220. ISBN 9781595935915. Disponível em: <https://doi.org/10.1145/1294261.1294281>. Citado 2 vezes nas páginas 13 e 20.

DUARTE, E. P.; BONA, L. C. E.; RUOSO, V. K. Vcube: A provably scalable distributed diagnosis algorithm. In: . IEEE, 2014. p. 17–22. ISBN 978-1-4799-7562-4. Disponível em: <http://ieeexplore.ieee.org/document/7016729/>. Citado 3 vezes nas páginas 14, 27 e 28.

ENES, V. et al. Efficient synchronization of state-based crdts. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. Los Alamitos, CA, USA: IEEE Computer Society, 2019. p. 148–159. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/ICDE.2019.00022>. Citado na página 23.

FOUTO, P. et al. Babel: A framework for developing performant and dependable distributed protocols. In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. Los Alamitos, CA, USA: IEEE Computer Society, 2022. p. 146–155. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/SRDS55811.2022.00022>. Citado 2 vezes nas páginas 30 e 39.

FOUTO, P.; LEITAO, J.; PREGUICA, N. Practical and fast causal consistent partial geo-replication. In: . IEEE, 2018. p. 1–10. ISBN 978-1-5386-7659-2. Disponível em: <https://ieeexplore.ieee.org/document/8548067/>. Citado 4 vezes nas páginas 12, 15, 25 e 26.

GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, Association for Computing Machinery, New York, NY, USA, v. 33, n. 2, p. 51–59, jun 2002. ISSN 0163-5700. Disponível em: <https://doi.org/10.1145/564585.564601>. Citado 2 vezes nas páginas 12 e 18.

GRAY, J.; LAMPORT, L. Consensus on transaction commit. *ACM Trans. Database Syst.*, Association for Computing Machinery, New York, NY, USA, v. 31, n. 1, p. 133–160, mar 2006. ISSN 0362-5915. Disponível em: <https://doi.org/10.1145/1132863.1132867>. Citado na página 19.

JEANNEAU, D. et al. An autonomic hierarchical reliable broadcast protocol for asynchronous distributed systems with failure detection. *Journal of the Brazilian Computer Society*, Springer London, v. 23, p. 1–14, 2017. Citado na página 27.

JUAN-MARÍN, R. et al. Scalability approaches for causal multicast: A survey. *Computing*, Springer-Verlag, Berlin, Heidelberg, v. 98, n. 9, p. 923–947, sep 2016. ISSN 0010-485X. Disponível em: <https://doi.org/10.1007/s00607-015-0479-0>. Citado na página 22.

LAKSHMAN, A.; MALIK, P. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 35–40, apr 2010. ISSN 0163-5980. Disponível em: <https://doi.org/10.1145/1773912.1773922>. Citado na página 20.

LAMPORT, L. Fast paxos. *Distributed Computing*, v. 19, p. 79–103, October 2006. Disponível em: <https://www.microsoft.com/en-us/research/publication/fast-paxos/>. Citado na página 19.

LEITÃO, J. *Topology Management for Unstructured Overlay Networks*. Tese (Doutorado) — Technical University of Lisbon, 09 2012. Citado na página 26.

LEITAO, J.; PEREIRA, J.; RODRIGUES, L. Epidemic broadcast trees. In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. [S.l.: s.n.], 2007. p. 301–310. Citado na página 27.

LINDE, A. et al. Legion: Enriching internet services with peer-to-peer interactions. In: . [S.l.: s.n.], 2017. p. 283–292. Citado na página 16.

MAHMOUD, H. et al. Low-latency multi-datacenter databases using replicated commit. *Proc. VLDB Endow.*, VLDB Endowment, v. 6, n. 9, p. 661–672, jul 2013. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/2536360.2536366>. Citado na página 19.

MCCORD, C. *Phoenix Presence*. Github, 2022. Disponível em: <https://github.com/phoenixframework/phoenix_pubsub/blob/master/lib/phoenix/tracker.ex>. Acesso em: 15 mai. 2022. Citado 3 vezes nas páginas 13, 23 e 37.

MEIKLEJOHN, C.; Van Roy, P. Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation. In: *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*. IEEE, 2015. v. 2016-Janua, p. 62–67. ISBN 978-1-5090-0092-0. ISSN 10609857. Disponível em: <https://ieeexplore.ieee.org/document/7371444/>. Citado 3 vezes nas páginas 12, 16 e 27.

MEIKLEJOHN, C. S.; Van Roy, P. Loquat: A framework for large-scale actor communication on edge networks. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2017*, p. 563–568, 2017. Citado na página 12.

ONGARO, D.; OUSTERHOUT, J. In search of an understandable consensus algorithm. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, 2014. p. 305–319. ISBN 978-1-931971-10-2. Disponível em: <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>. Citado na página 19.

PREGUIÇA, N.; BAQUERO, C.; SHAPIRO, M. Conflict-free replicated data types crdts. In: SAKR, S.; ZOMAYA, A. (Ed.). *Encyclopedia of Big Data Technologies*. Cham: Springer International Publishing, 2018. p. 1–10. ISBN 978-3-319-63962-8. Disponível em: <https://doi.org/10.1007/978-3-319-63962-8_185-1>. Citado na página 20.

PREGUIÇA, N. M. Conflict-free replicated data types: An overview. *CoRR*, abs/1806.10254, p. 1–41, 2018. Disponível em: <http://arxiv.org/abs/1806.10254>. Citado na página 22.

REDIS. *Active-Active geo-distributed Redis*. Redis, 2017. Disponível em: <https://docs.redis.com/latest/rs/databases/active-active/>. Acesso em: 15 mai. 2022. Citado na página 24.

RODRIGUES, L. et al. Bundling messages to reduce the cost of tree-based broadcast algorithms. In: *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. [S.l.]: IEEE, 2018. p. 115–124. Citado na página 47.

RODRIGUES, L. A.; ARANTES, L.; DUARTE, E. P. An autonomic implementation of reliable broadcast based on dynamic spanning trees. In: IEEE. *2014 Tenth European Dependable Computing Conference*. [S.l.], 2014. p. 1–12. Citado na página 27.

RODRIGUES, L. A. et al. Bundling messages to reduce the cost of tree-based broadcast algorithms. In: IEEE. *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. [S.l.], 2018. p. 115–124. Citado na página 27.

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 27.

RODRIGUES, L. A.; JR, E. P. D.; ARANTES, L. Arvores geradoras mínimas distribuídas e autonômicas. *Anais do 32 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos–SBRC*, v. 2014, p. 1–14, 2014. Citado na página 27.

SAITO, Y.; SHAPIRO, M. Optimistic replication. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 37, n. 1, p. 42–81, mar 2005. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/1057977.1057980>. Citado 4 vezes nas páginas 12, 24, 25 e 26.

SHAPIRO, M. et al. *A comprehensive study of Convergent and Commutative Replicated Data Types*. [S.l.], 2011. 50 p. Disponível em: <https://hal.inria.fr/inria-00555588>. Citado 3 vezes nas páginas 21, 22 e 37.

SHAPIRO, M. et al. Conflict-free replicated data types. In: DÉFAGO, X.; PETIT, F.; VILLAIN, V. (Ed.). *Stabilization, Safety, and Security of Distributed Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 386–400. ISBN 978-3-642-24550-3. Citado 6 vezes nas páginas 13, 19, 20, 21, 22 e 23.

TANENBAUM, M. v. S. A. *Distributed Systems*. 3. ed. [S.l.: s.n.], 2017. Citado na página 18.

VIEIRA, E. *ECO SYNC TREE: A Causal and Dynamic Broadcast Tree for Edge-based Replication*. Dissertação (Mestrado) — NOVA University Lisbon, 11 2021. Citado 9 vezes nas páginas 12, 13, 15, 16, 26, 27, 30, 35 e 39.

VOGELS, W. Eventually consistent. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 1, p. 40–44, jan 2009. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1435417.1435432>. Citado 2 vezes nas páginas 12 e 18.

YOUNES, G. et al. Integration challenges of pure operation-based crdts in redis. In: *First Workshop on Programming Models and Languages for Distributed Computing*. New York, NY, USA: Association for Computing Machinery, 2016. (PMLDC '16). ISBN 9781450347754. Disponível em: <https://doi.org/10.1145/2957319.2957375>. Citado na página 13.