Sincronização Eficiente de CRDTs em Escala Utilizando uma Solução Hierárquica de Publish–Subscribe

Leonardo de Freitas Galesky[1] **e Luiz Antonio Rodrigues**[1]

[1]Programa de Pós-Graduação em Computação (PPGComp)
Universidade Estadual do Oeste do Paraná (Unioeste)
Cascavel – PR – Brasil

leonardo.galesky@unioeste.br, luiz.rodrigues@unioeste.br

Abstract. *This study presents VCube-Sync, a system that uses a virtual hypercube topology as the basis for replicating a Conflict-free Replicated Data Type (CRDT)-based data store. CRDTs can ensure consistency in a deterministic and conflict-free manner. At the same time, hypercubes have been previously used for message distribution due to their fault tolerance and logarithmic latency while enabling heuristics based on knowledge of the structured overlay. The protocol presented in this study is based on VCube- PS leveraging synergies between pub-sub and replication systems. VCube-Sync was tested under different loads and network distributions using the Grid5000 testbed, and the results were compared with those of other replication protocols in recent research. The results of this study show that VCube-Sync provides good results in terms of latency, scalability, and bandwidth.*

Resumo. *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). Os CRDTs podem garantir a consistência de forma determinística e livre de conflitos. Ao mesmo tempo, 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 sinergias 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.*

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. Esta escolha é necessária já que o Teorema CAP afirma ser impossível garantir simultaneamente as três propriedades de Consistência (C), Disponibilidade (A) e Tolerância a partições (P) [Vogels 2009].

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 geo-replicados [Saito and Shapiro 2005].

Tais cenários requerem o uso de estratégias especializadas de difusão e replicação que tem sido explorado extensivamente como [Meiklejohn and Van Roy 2015] e [Meiklejohn and Van Roy 2017] em que estes limites de escalabilidade foram avaliados no contexto de sistemas de dados distribuídos sem-líder e escalados horizontalmente, neste caso, foi proposto o uso de protocolos mais eficientes como árvores de difusão ou refinamentos funcionais como difusão por tópicos.

Estes desafios importantes também são expostos por [Fouto et al. 2018], que apresenta resultados positivos na utilização de replicação parcial e da separação das camadas de dados e de controle de causalidade quando necessária. Mais recentemente ainda, [Vieira 2021] explorou o uso de árvores de difusão e estruturas livres de conflito num ambiente com presença dinâmica.

Para a maioria das aplicações, não basta que os dados do sistema sejam replicados eficientemente, mas 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”, ou mais avançadas como utilizando de relógios vetoriais, entretanto implementações não rigorosas podem resultar em perda de dados ou em um estado final inconsistente [DeCandia et al. 2007].

Neste sentido, Tipos de Dados Replicados e Livres de Conflitos - CRDTs (*Conflict-Free Replicated Data Types*) [Shapiro et al. 2011] 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, para isso transmitem todo o conjunto de dados às outras réplicas após alterações na cópia local. Isso implica em crescimento ilimitado da mensagem de replicação e, portanto, apresenta desafios de vazão (*throughput*) e latência [Shapiro et al. 2011].

Por outro lado, os CRDTs baseados em operações são capazes de serializar a operação de alteração local e transmitir apenas ela 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 et al. 2017] e [Younes et al. 2016] e permite que a estrutura de dados seja mais simples e utilize mensagens menores. Recentemente [Bauwens and 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 do padrão era baseado em um barramento que atuaria como um gargalo do sistema, sendo este portanto substituído por uma malha totalmente conectada. Utilizar de uma topologia de rede especializada trouxe melhorias a performance 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 et al. 2014] determina uma topologia estruturada baseada em hipercubo, que apresenta importantes propriedades logarítmicas. A implementação VCube-PS [de Araujo et al. 2019], um sistema *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 um *overlay* 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 ou 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 escalável de sincronização de CRDTs com suporte a replicação parcial via tópicos explorando a sinergia entre sistemas *publisher-subscriber* e protocolos de replicação. Assim como VCube e VCube-PS, a transmissão de operações faz uso de uma rede de sobreposição (*overlay*) baseada em hipercubos, e é capaz de garantir a ordenação causal das mensagens 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.

O restante deste trabalho está organizado nas seguintes seções. A Seção 2 apresenta os trabalhos relacionados. O VCube-Sync é descrito na Seção 3. Os resultados são apresentados na Seção 4. A conclusão e os trabalhos futuros estão na Seção 5.

2. 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 a escalabilidade são priorizados.

C^3 foi construído como uma camada modular aplicável a diferentes *backends*, o C^3 introduz um modelo capaz de fornecer ordenação causal e replicação parcial num ambiente *peer-to-peer* geo-distribuído. Esta camada utiliza identificadores e relógios vetoriais para estabelecer ordenação causal. 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 [Fouto et al. 2018].

Legion é um framework que permite múltiplos clientes a replicarem seu estado de maneira híbrida 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, a fim 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 em limitações nas inferências em relação a estrutura da rede, além disso é presumida a existência de servidores centralizados [Linde et al. 2017].

Cure 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 [Akkoorath et al. 2016].

Selective hearing 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 os 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 [Meiklejohn and Van Roy 2015].

SYNC Tree 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. Para garantir causalidade o modelo explora a entrega *First In, First Out (FIFO)* de operações por um canal de entrega confiável na topologia de árvore conforme [van der Linde et al. 2020]. 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 [Vieira 2021].

Os modelos apresentam características semelhantes ao proposto por este trabalho mas nenhum de forma integral, nossa solução de difusão pretende atender a sistemas massivamente distribuídos, de maneira descentralizada, suportando replicação parcial e utilizando de topologia estruturada que permita inferir propriedades sobre o arranjo dos nós e seus vizinhos. Além disso, nosso sistema utilizará de CRDTs para resolução de conflitos e manutenção de um estado consistente.

O modelo mais similar ao proposto neste trabalho é o *ECO SYNC Tree* e, portanto, será utilizado como referência para fins de comparação.

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, como contenção na rede, através do uso de um canal confiável e reenvio de mensagens. O sistema suporta replicação parcial e, portanto, diferentes nós podem possuir conjuntos de dados diferentes em um mesmo intervalo de tempo.

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 uma determinada partição no caso de CRDTs baseados em estado.

Operações são transmitidas apenas a nós interessados (*subscribers*) numa determinada partição dos dados (tópico). Cada transmissão ocorre através de uma árvore 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 aplicação é construída em módulos independentes inspirados por [Vieira 2021] e [Fouto et al. 2022]: ***membership*** que determina a conexão de rede real entre os nós ***broadcast*** que aplica estratégias de difusão, como sobreposições à rede real e determinação

de arvores de difusão, e garante pré-condições como causalidade; ***replication*** que fornece uma interface ao *data-store* de CRDT; e ***application*** que mantém o estado do *data-store* usando CRDTs. A Figura 1 apresenta o relacionamento entre cada um dos módulos.

Figura 1. Arquitetura da aplicação proposta.

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. Replicação baseada em hipercubos

Uma sobreposição de rede em hipercubo é estabelecida por um nó de índice i dividindo os $N-1$ demais membros do do sistema em $d = \log_2 N$ *clusters* que possuem o mesmo tamanho $(1..d)$. A lista de nós em cada *cluster* s é determinada pela função $c_{i,s}$ apresentada na Equação 1, onde $\oplus$ representa o operador lógico bitwise XOR [Duarte et al. 2014].

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

A Figura 2 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.

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

Figura 2. Organização hierárquica do VCube representado pela tabela $c_{i,s}$ [de Araujo et al. 2019]

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 esta 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 deste sistema a camada de *broadcast* é responsável por manter o mapa do relacionamento entre nós e tópicos.

No VCube-Sync 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. As mensagens podem ser de três tipos: SUB ou *subscribe* que é enviada por um nó para demonstrar interesse em um tópico, UNS ou *unsubscribe* que reverte a operação de SUB, e por fim PUB ou *publication* que no Vcube-Sync corresponde ao envio de uma mensagem de replicação.

A interface do sistema é composta por três funções: $Subscribe(t)$, $Unsubscribe(t)$ e $Publish(t, m)$, em que t representa um tópico e m uma mensagem. Um nó só é capaz de publicar mensagens em tópicos do qual ele faz parte. Após recebidas, as mensagens são entregues usando a função $Co\_Deliver$, esta ação ocorre somente após todas as dependências causais da mensagem serem completas, as mensagens podem ser mantidas em um *buffer* enquanto essa condição não é cumprida.

Uma vez que o no contexto do VCube-Sync mensagens carregam operações a serem aplicados no estado do CRDT, o Algoritmo 1 apresenta uma modificação aplicada ao método $Co\_Deliver$ do VCube-PS em que o evento é comunicado para outros módulos do software, sendo no mínimo um deles o responsável por gerenciar o estado da aplicação.

Algorithm 1: Extensão do método $Co\_Deliver$ adicionando suporte a comunicação do evento de entrega a outros módulos da aplicação

```
let delivery_subscribers = Ø
function Co_Deliver (m): void
    for r ∈ delivery_subscribers.keys do
        delivery_subscribers[r].ApplyUpdate(m.data)
    end
```

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.

3.2. Ordenação Causal

Dado um tópico t e duas mensagens m e m' e a ordem parcial $\prec$ que representa um relacionamento de acontece-antes (*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 limpa, 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. Tendo como diferença que o VCube-PS implementa também garantia de recepção *First In, First Out* (FIFO) *per-source* como parte do mecanismo para lidar com dinâmicas de entrada e saída de nós. No VCube-Sync consideramos um sistema estável, portanto este mecanismo não é necessário e foi removido o que aumenta a reatividade do sistema por permitir maior paralelismo na transmissão.

3.3. Conflict Free Replicated Data Types (CRDTs)

O VCube-Sync mantém seu estado local utilizando CRDTs. Uma vez que a aplicação é modular, qualquer tipo de CRDT em que ordenação causal seja suficiente pode ser utilizado, este é o caso de CRDTs baseados em estado e a maioria dos CRDTs baseados em operações. A camada de replicação e o protocolo de difusão são capazes de garantir que uma operação só será aplicada ao estado se (1) a operação for iniciada pelo próprio nó, neste caso não há dependências; (2) a operação foi iniciada por outro nó e a pré-condição de entrega de ordenação causal foi satisfeita.

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

4. Resultados Experimentais

Para avaliar o desempenho do VCube-Sync em diferentes cenários foram conduzidos experimentos na plataforma Grid'5000 (`https://www.grid5000.fr`), 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].

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

Cada experimento segue a seguinte sequência de passos: (1) Executar todos os nós via contêiner 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*; e (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.

4.1. 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 3(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 ECO-Sync-Tree 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 ECO-Sync-Tree para replicação total é de 0.854s, 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.

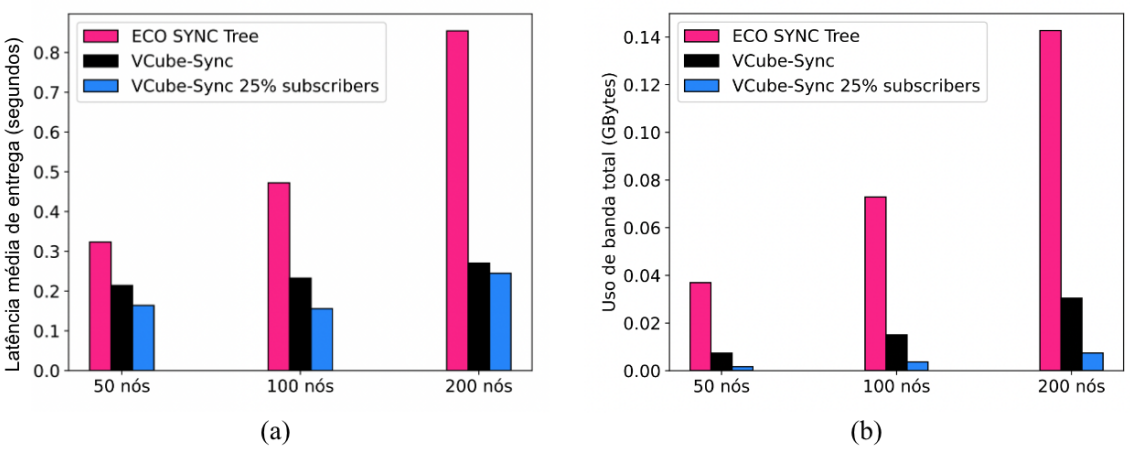

Figura 3. Resultados para um único *Publisher*. (a) Latência média de entrega de mensagens em segundos por protocolo e número de nós; (b) Uso de banda total do experimento em GBytes por protocolo e número de nós.

Além disso, o efeito da replicação parcial pode também ser observado no número total de bytes transmitidos para cada simulação apresentado na Figura 3(b). Para 200 nós, o número total de bytes transmitidos pelo ECO-Sync-Tree 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. Isto indica que a fator de redução causado pela replicação parcial está conforme esperado, como somente nós interessados nos tópicos recebem e transmitem mensagens, a redução do número de bytes acontece linearmente na mesma proporção.

Ainda, destaca-se que o VCube não requer o envio de mensagens para manutenção da topologia e, portanto, a taxa de transmissão de bytes tende a zero se mensagens de replicação não estão em trânsito. Por outro lado, o Eco-Sync-Tree utiliza de mensagens

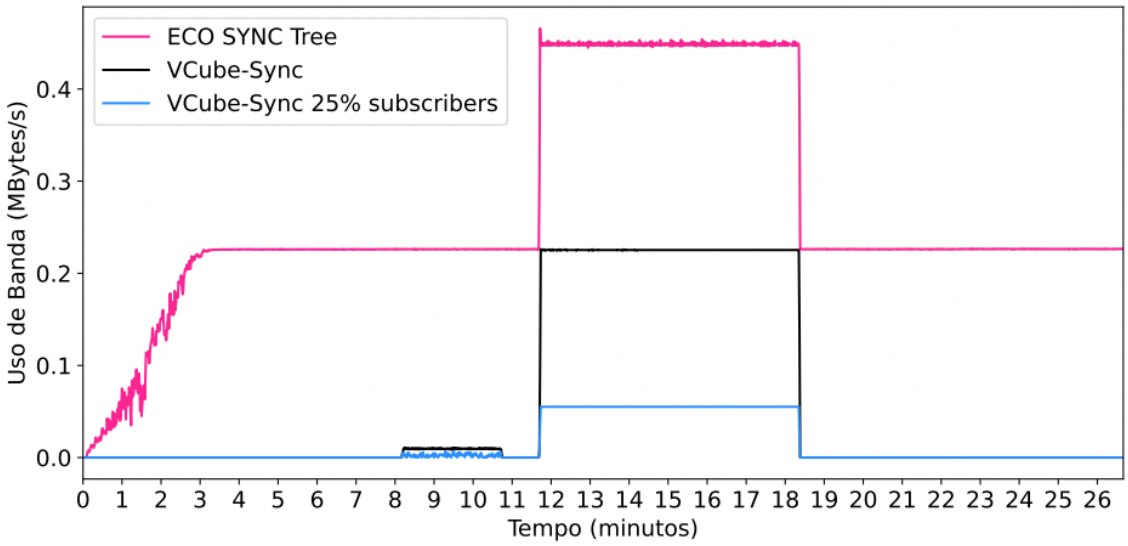

Figura 4. Uso de banda por protocolo em MBytes por segundo para 200 nós.

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 4. Além disso, o número de de mensagens duplicadas observadas pelos nós utilizando o ECO-Sync-Tree com apenas um *publisher* foi de 0 a 12%. Enquanto que para o VCube-Sync nunca há mensagens duplicadas quando não há falhas.

4.2. 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 5(a) observa-se que para 200 nós quando há 100% de *publishers* a latência média de entrega de mensagens do ECO-SYNC-tree é 1,56x a apresentada pelo VCube-Sync. Quando o arranjo contém 25% de *publishers* a diferença é ligeiramente reduzida para 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 Araujo et al. (2019).

Já para o uso de banda, conforme a Figura 5(b), para o cenário de 200 nós e 100% de *publishers* o VCube-PS apresentou uma vazão total de 6,41 GBytes, um valor 7,39% maior quando comparado ao ECO-SYNC-tree. 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 num cenário com múltiplos *publishers* o *overhead* de manutenção da rede presente no ECO-SYNC-Tree é mínimo, sendo a taxa de envio de mensagens para manutenção da rede no cenário com 200 nós e 100% de *publishers* de

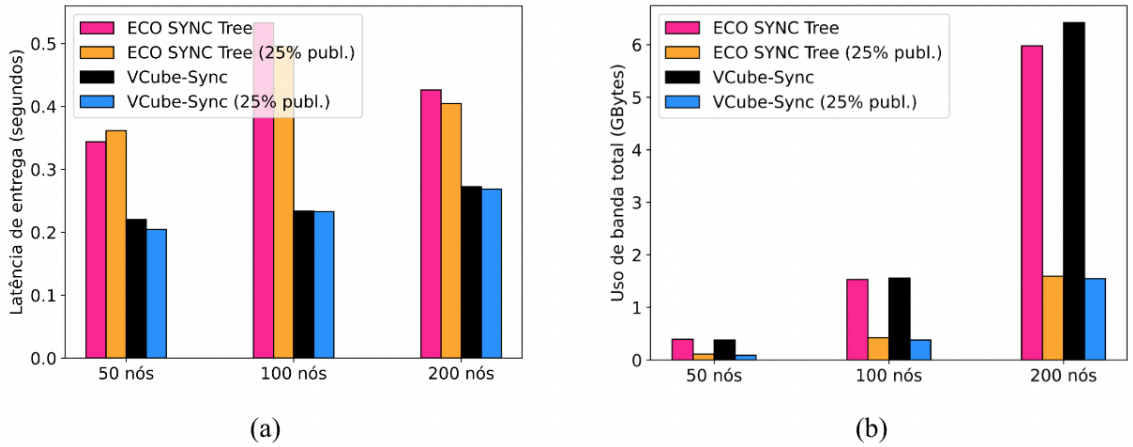

Figura 5. Resultado para Múltiplos *Publishers*. (a) Latência média de entrega de mensagens em segundos por protocolo, número de nós e fração de *publishers*; (b) Uso de banda total do experimento em GBytes por protocolo, número de nós e fração de *publishers*

aproximadamente 10 KBytes/s, enquanto a taxa máxima de vazão do experimento foi de 48,63 MBytes/s.

Estes resultados demonstram que o VCube-Sync apresenta significativa vantagem na latência de entrega de mensagens com um penalidade de um pequeno aumento no número de bytes transmitidos.

5. Conclusão

Neste artigo apresentamos o VCube-Sync, um novo protocolo de replicação parcial baseado em hipercubos que oferece garantias de entrega de operações respeitando ordenação causal utilizando barreiras causais. O estado da aplicação é mantido por CRDTs, o que permite que as operações sejam realizadas concorrentemente assegurando um estado final deterministicamente convergente.

A solução proposta é baseada no protocolo publish-subsbcribe VCube-PS e é capaz de criar árvores de difusão hierárquicas que têm como raiz o nó que realizou uma operação que atualiza o estado. Os resultados demonstram que a solução apresenta excelente desempenho em termos de latência de entrega de mensagens e bons índices em termos de uso de banda, especialmente em cenários com múltiplos tópicos. Estes resultados sugerem que a solução é uma opção promissora para a garantia de consistência em sistemas distribuídos de dados. Além disso, o VCube-PS, que é a base do VCube-Sync, apresentou bom desempenho em cenários simulados com mais de mil nós, o que demonstra a escalabilidade da solução.

Como trabalhos futuros pretende-se incluir ao VCube-Sync a capacidade de lidar com dinâmica de entrada e saída de membros do sistema assim como recuperação de falhas catastróficas. Apesar de o VCube ser utilizado como um detector de falhas, o algoritmo implementado neste trabalho também requer mecanismos extras de sincronização do estado similar ao implementando pelo ECO-Sync-Tree, algumas das propostas envolvem o uso hibrido de outros tipos de CRDTs como baseados em Estado ou Delta-Estado para esta fase. Este processo de sincronização pode potencialmente utilizar metadados do

hipercubo para determinar maneiras ótimas de sincronização.

Além disso, para reduzir a quantidade de mensagens e o custo de comunicação, algoritmos de agregação por causalidade ou tempo poderiam ser usados como por exemplo os apresentados em [de Araujo 2019] e [Rodrigues et al. 2018].

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. 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).

Referências

Akkoorath, D. D., Tomsic, A. Z., Bravo, M., Li, Z., Crain, T., Bieniusa, A., Preguiça, N., and Shapiro, M. (2016). Cure: Strong semantics meets high availability and low latency. In *36th IEEE Int'l Conf. Distributed Comp. Systems (ICDCS)*, pages 405–414.

Baquero, C., Almeida, P. S., and Shoker, A. (2017). Pure operation-based replicated data types. *CoRR*, abs/1710.04469. http://arxiv.org/abs/1710.04469.

Bauwens, J. and Boix, E. G. (2021). Improving the Reactivity of Pure Operation-Based CRDTs. In *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*, pages 1–6, New York, NY, USA. ACM. https://dl.acm.org/doi/10.1145/3447865.3457968.

Brown, R., Cribbs, S., Meiklejohn, C., and Elliott, S. (2014). Riak dt map: A composable, convergent replicated dictionary. In *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*, PaPEC '14, New York, NY, USA. ACM. https://doi.org/10.1145/2596631.2596633.

de Araujo, J. P. (2019). *A communication-efficient causal broadcast publish/subscribe system*. Theses, Sorbonne Université.

de Araujo, J. P., Arantes, L., Duarte, E. P., Rodrigues, L. A., and Sens, P. (2019). Vcube-ps: A causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, 125:18–30. https://www.sciencedirect.com/science/article/pii/S0743731518307822.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *Operating Systems Review (ACM)*, pages 205–220.

Duarte, E. P., Bona, L. C. E., and Ruoso, V. K. (2014). Vcube: A provably scalable distributed diagnosis algorithm. pages 17–22. IEEE. http://ieeexplore.ieee.org/document/7016729/.

Fouto, P., Costa, P. A., Preguiça, N., and Leitão, J. (2022). Babel: A framework for developing performant and dependable distributed protocols. https://arxiv.org/abs/2205.02106.

Fouto, P., Leitao, J., and Preguica, N. (2018). Practical and fast causal consistent partial geo-replication. pages 1–10. IEEE. https://ieeexplore.ieee.org/document/8548067/.

Linde, A., Fouto, P., Leitão, J., Preguiça, N., Castiñeira, S., and Bieniusa, A. (2017). Legion: Enriching internet services with peer-to-peer interactions. pages 283–292.

McCord, C. (2022). Phoenix presence.

Meiklejohn, C. and Van Roy, P. (2015). Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation. In *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, volume 2016-Janua, pages 62–67. IEEE. https://ieeexplore.ieee.org/document/7371444/.

Meiklejohn, C. S. and Van Roy, P. (2017). Loquat: A framework for large-scale actor communication on edge networks. *2017 IEEE Int'l Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2017*, pages 563–568.

Rodrigues, L. A., Duarte Júnior, E. P., de Araujo, J. P., Arantes, L., and Sens, P. (2018). Bundling Messages to Reduce the Cost of Tree-Based Broadcast Algorithms. In *LADC 2018 - 8th Latin-American Symposium on Dependable Computing*, Foz do Iguaçu, Brazil.

Saito, Y. and Shapiro, M. (2005). Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81. https://doi.org/10.1145/1057977.1057980.

Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. (2011). Conflict-free replicated data types. In Défago, X., Petit, F., and Villain, V., editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg. Springer Berlin Heidelberg.

van der Linde, A., Fouto, P., Leitão, J. a., and Preguiça, N. (2020). The intrinsic cost of causal consistency. In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, PaPoC '20, New York, NY, USA. Association for Computing Machinery.

Vieira, E. (2021). Eco sync tree: A causal and dynamic broadcast tree for edge-based replication. Master's thesis, NOVA University Lisbon.

Vogels, W. (2009). Eventually consistent. *Commun. ACM*, 52(1):40–44. https://doi.org/10.1145/1435417.1435432.

Younes, G., Shoker, A., Almeida, P. S., and Baquero, C. (2016). Integration challenges of pure operation-based crdts in redis. In *First Workshop on Programming Models and Languages for Distributed Computing*, PMLDC '16, New York, NY, USA. ACM. https://doi.org/10.1145/2957319.2957375.