ANÁLISE COMPARATIVA DE ESTRATÉGIAS DE REPLICAÇÃO DE DADOS EM SISTEMAS DE ALTA DISPONIBILIDADE

Enrico Lourenço Mainenti[1], Marlon de Matos de Oliveira[2]

Resumo: Este trabalho apresenta uma análise comparativa de três estratégias de replicação de dados em sistemas de alta disponibilidade: replicação síncrona, assíncrona e bidirecional (BDR). O objetivo principal foi avaliar o impacto de cada técnica no desempenho, na consistência e na resiliência dos ambientes configurados, considerando métricas como latência, *throughput*, estabilidade sob carga concorrente e comportamento em cenários de *failover*. A metodologia envolveu a implantação de *clusters* PostgreSQL com ferramentas de gerenciamento de alta disponibilidade, que foram posteriormente submetidos a testes controlados de carga e tolerância a falhas. Os resultados demonstraram que a replicação assíncrona apresentou melhor equilíbrio entre *throughput* e responsividade, ainda que com risco de perda de dados em janelas curtas de falha. A replicação síncrona garantiu consistência forte e preservação das transações durante o *failover*, mas apresentou elevada latência em operações de escrita. Já a replicação bidirecional demonstrou potencial de escalabilidade para múltiplos nós com capacidade de escrita, mas sofreu com conflitos de replicação e alta taxa de falhas em cenários intensos de concorrência. Conclui-se que a escolha da técnica de replicação deve considerar os *trade-offs* entre consistência, desempenho e disponibilidade, de acordo com as exigências do ambiente de aplicação.

Palavras-chave: replicação de dados; alta disponibilidade; PostgreSQL; *failover*; sistemas distribuídos.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), enricomaine@gmail.com

[2]Orientador. Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), marlon.oliveira@gmail.com

ABSTRACT: This work presents a comparative analysis of three data replication strategies in high availability systems: synchronous, asynchronous and bidirectional (BDR). The main objective was to evaluate the impact of each technique on performance, consistency and resilience of configured environments, considering metrics such as latency, throughput, stability under concurrent load and behavior in failover scenarios. The methodology consisted of deploying PostgreSQL clusters using high availability management tools, subjected to controlled load and failure tests. The results showed that asynchronous replication offered the best balance between throughput and responsiveness, although with the inherent risk of data loss in short failure windows. Synchronous replication ensured strong consistency and preservation of transactions during failover, but presented high latency in write operations. Bidirectional replication demonstrated scalability potential for multiple writable nodes, but faced replication conflicts and high failure rates under intense concurrency. It is concluded that the choice of replication technique must consider the trade-offs between consistency, performance and availability, according to the requirements of the application environment.

Keywords: data replication; high availability; PostgreSQL; failover; distributed systems.

1 INTRODUÇÃO

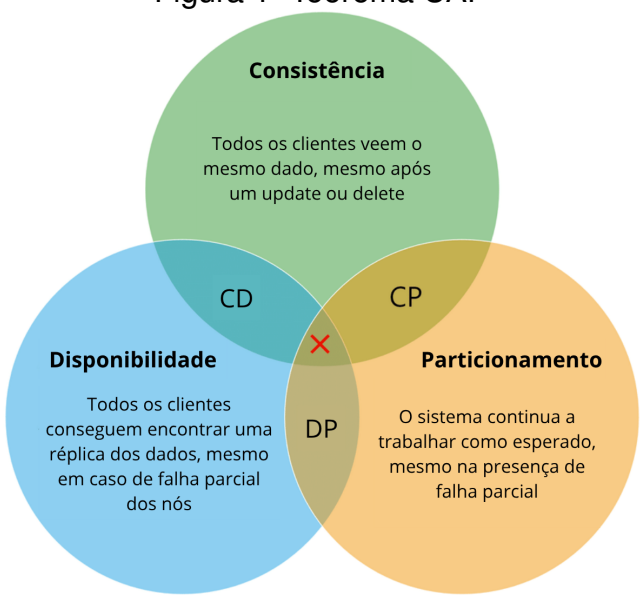
A evolução contínua dos sistemas de computadores levou ao crescimento de aplicações distribuídas, que exigem mecanismos cada vez mais poderosos para garantir desempenho, disponibilidade e confiabilidade. Nesse cenário, a replicação de dados destaca-se como uma das principais estratégias para garantir que as informações permaneçam acessíveis mesmo diante de falhas, além de contribuir para a escalabilidade e experiência do usuário final em ambientes críticos (Pasin, 2003).

Com a expansão dos serviços em nuvem, bancos de dados distribuídos, redes *peer-to-peer* e sistemas críticos, a replicação se tornou uma técnica amplamente utilizada para mitigar os riscos associados à perda de dados e à indisponibilidade de serviços. No entanto, sua implementação enfrenta desafios complexos, particularmente no que diz respeito à manutenção da consistência entre réplicas, à sincronização eficiente de atualizações e à minimização da latência. De acordo com Vogels (2008), a consistência eventual é frequentemente adotada como um compromisso

em sistemas de alta disponibilidade, mas essa abordagem pode não ser adequada para aplicações críticas.

O Teorema CAP, formulado por Gilbert e Lynch (2002) e discutido por Vogels (2008), demonstra que não é possível garantir simultaneamente consistência, disponibilidade e tolerância a falhas em sistemas distribuídos, essa restrição está ilustrada na Figura 1 e exige escolhas estratégicas no uso da replicação, especialmente em ambientes que demandam alta disponibilidade e escalabilidade. Nesse contexto, diferentes modelos de replicação podem ser adotados, como primária-secundária, multimestre e escrita única (Bernstein; Newcomer, 2009), cuja seleção deve equilibrar os *trade-offs* do teorema, além de considerar o tipo de consistência mais adequado ao sistema: forte, eventual ou causal (Dean; Ghemawat, 2008).

Figura 1- Teorema CAP



Fonte: Medium [3].

Ferramentas como PostgreSQL oferecem múltiplas estratégias de replicação, incluindo *streaming replication*, replicação lógica e soluções multimestre como o BDR (PostgreSQL Global Development Group, 2023). Para garantir alta disponibilidade e *failover* automatizado, soluções como Patroni em conjunto com Etcd são amplamente adotadas, pois a saúde das instâncias do banco são continuamente monitoradas e, em caso de falha na instância principal, um novo líder é promovido automaticamente (Zalando, 2023). Já para cenários que exigem replicação bidirecional e baixa latência,

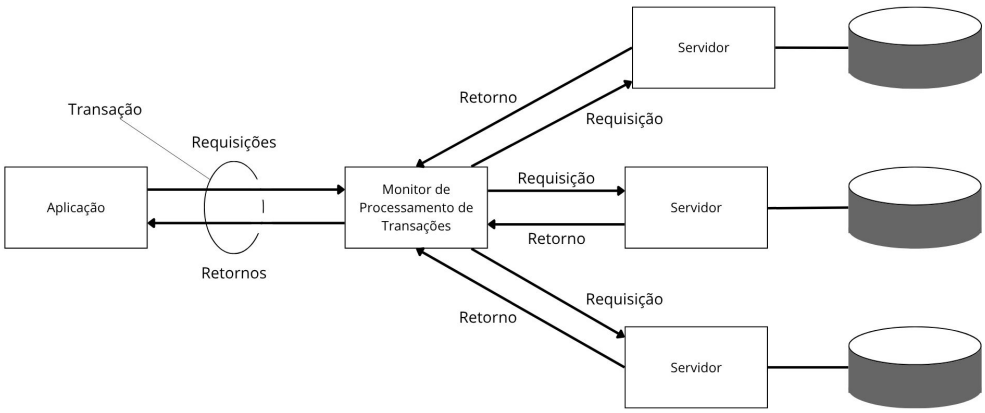
[3]https://medium.com/nerd-for-tech/understand-cap-theorem-751f0672890e

o BDR possibilita a escrita concorrente em diferentes nós, especialmente útil em ambientes geograficamente distribuídos (EDB, 2023).

A eficácia dessas abordagens pode ser validada por meio de testes de carga com ferramentas como Artillery, sendo possível definir cenários customizados contendo múltiplos usuários virtuais executando operações simultâneas, simulando um ambiente de produção (Artillery.io, 2024). Enquanto isso, o monitoramento contínuo com Prometheus e Grafana permite acompanhar métricas críticas como latência de replicação e tempo de *failover* (Labs, 2024).

A replicação de dados constitui elemento essencial no desenvolvimento de sistemas distribuídos de alta disponibilidade, garantindo a continuidade do serviço em ambientes críticos, como bancos de dados, servidores web e sistemas de arquivos (Myo; Bain; Portnov, 2023). Diferentes estratégias de replicação como síncrona, assíncrona ou híbrida, oferecem vantagens específicas em termos de consistência, latência e desempenho, sendo sua escolha determinante para a resistência a falhas e a manutenção da disponibilidade (McMahon, 2023). Desse modo, a Figura 2 ilustra de forma simplificada o funcionamento de um sistema distribuído síncrono.

Figura 2 - Representação de um sistema distribuído



Fonte: Github[4]

Adicionalmente, os métodos de replicação impactam diretamente a segurança e a confiabilidade dos dados, reduzindo riscos de perda ou exposição de informações sensíveis em aplicações críticas, como saúde, finanças e comércio eletrônico (Steen, 2002). Desta forma, compreender os efeitos de cada abordagem e selecionar a mais adequada às necessidades

[4]https://vhnam.github.io/tutorials/distributed-system-cac-kieu-con-da-dieu

específicas é fundamental para o desenvolvimento de sistemas flexíveis, escaláveis e de elevado desempenho.

Nesse contexto, a avaliação das estratégias de replicação foi conduzida a partir de quatro métricas principais: latência, *throughput*, consistência e *failover*. A latência permitiu mensurar o tempo de resposta das operações e o impacto da replicação na experiência do usuário. O *throughput* foi utilizado para avaliar a capacidade do sistema em processar transações simultâneas e sustentar altas cargas de trabalho. Já a consistência foi analisada sob diferentes modelos de replicação, verificando o grau de sincronização e integridade dos dados entre os nós. Por fim, o *failover* foi medido em termos do tempo de inatividade e da eficiência da recuperação após a falha de um nó, aspecto crucial para sistemas de alta disponibilidade. Essas métricas, em conjunto, possibilitaram uma análise abrangente do desempenho e da resiliência das abordagens estudadas.

Estudos na área de computação distribuída têm buscado soluções para esses desafios. Por exemplo, Terry et al. (1995) propuseram a replicação com escrita desconectada para aumentar a disponibilidade sem sacrificar completamente a consistência, porém o modelo não contemplava cenários de alta concorrência e exigia mecanismos adicionais para resolução de conflitos. De forma semelhante, Holman e Anderson (2006) desenvolveram o TAPIR, um protocolo voltado para transações consistentes e de baixa latência em sistemas de larga escala, mas sua aplicação era restrita a determinados ambientes controlados, não abrangendo cargas de trabalho heterogêneas ou situações de falha em múltiplos nós.

Mais recentemente, Medeiros et al. (2020) avaliaram a replicação de bancos de dados em cenários de recuperação de desastres, utilizando MySQL em ambiente de nuvem pública. Os experimentos compararam replicação assíncrona e semissíncrona sob diferentes cargas (0,1 e 2 req/s) e localizações de réplicas. Os resultados mostraram que, na replicação assíncrona, o tempo de resposta variou apenas conforme o tamanho do blob, enquanto na semissíncrona houve aumento significativo de latência com réplicas distantes e cargas menores. Em termos de RPO (*Recovery Point Objective*), a replicação assíncrona apresentou janelas de perda de dados na ordem de alguns a dezenas de milissegundos, ao passo que a semissíncrona garantiu RPO zero. Já o RTO (*Recovery Time Objective*) permaneceu estável em todos os cenários, com tempos médios de aplicação de transações também na faixa de milissegundos.

Diferentemente desse estudo, que se restringiu a cargas reduzi-

das e médias globais das métricas, o presente trabalho amplia os cenários experimentais, incluindo níveis mais altos de concorrência, cargas mistas de leitura e escrita, e métricas mais refinadas, como latência em percentis p95/p99, *throughput* sob pico e impacto do *failover* automático sob carga ativa. Assim, busca-se oferecer uma visão mais completa e próxima da realidade de sistemas distribuídos em produção.

Bem como Shrestha e Tandel (2023), que contribuíram também com pesquisas na área apresentando uma avaliação detalhada de duas soluções de bancos de dados altamente disponíveis, Percona XtraDB Cluster (PXDB) e MySQL NDB Cluster (MNDB), utilizando *benchmarks* de operações básicas (CRUD) em um ambiente baseado em OpenStack. Nos experimentos de leitura pura (ReadOnly), o PXDB obteve melhor desempenho, alcançando cerca de 11.000 transações por segundo (TPS) com 16 *threads*, superando o MNDB, que apresentou em torno de 9.000 TPS no mesmo cenário. Já nas cargas de leitura e escrita combinadas (Read-Write), bem como nas operações de atualização e exclusão, o MNDB demonstrou vantagem significativa: em alguns testes, chegou a 13.000 TPS, enquanto o PXDB manteve-se em torno de 7.000 TPS. Além do maior *throughput*, o MNDB também apresentou latências menores (abaixo de 20 ms em média) e utilização mais eficiente de recursos, consumindo até 30% menos CPU que o PXDB em situações equivalentes. Ambos os sistemas, no entanto, mostraram robustez frente a falhas, permanecendo disponíveis mesmo após a desconexão de nós, o que reforça a adequação de suas arquiteturas multimestre e de replicação síncrona para cenários críticos.

Embora o estudo ofereça uma contribuição relevante, ele se restringe a um número limitado de *threads* (até 72), identificando que o desempenho ótimo ocorreu com apenas 16 *threads*, o que não reflete situações de concorrência mais intensa que frequentemente caracterizam ambientes reais. Neste trabalho, busca-se justamente avançar nessa direção: apesar de mantermos o mesmo ambiente de execução, ampliaremos o escopo dos experimentos para explorar níveis mais elevados de concorrência, incluindo testes com centenas de *threads*, além de analisar o comportamento em cenários de saturação e picos de carga. Outro diferencial será a inclusão de métricas mais refinadas de observabilidade, como análise de latência em percentis p95 e p99, utilização de rede e I/O de disco, e o impacto do *failover* automático sob carga ativa. Dessa forma, enquanto o estudo de Shrestha e Tandel fornece uma visão comparativa geral entre PXDB e MNDB, nossa proposta pretende oferecer uma análise mais aprofundada

do comportamento de desempenho e disponibilidade sob condições extremas, aproximando os resultados das exigências encontradas em sistemas distribuídos de produção.

Essas propostas destacam a importância de avaliar continuamente o impacto das escolhas de replicação na performance e na confiabilidade de sistemas distribuídos.

Portanto, este trabalho tem como objetivo geral comparar estratégias de replicação de dados em sistemas de alta disponibilidade, avaliando seu desempenho, escalabilidade e confiabilidade, a fim de identificar quais delas se mostram mais eficazes em ambientes críticos. Como objetivos específicos, propõe-se analisar diferentes estratégias de replicação (síncrona, assíncrona e híbrida), avaliar seus impactos na performance, comparar a confiabilidade em cenários de falha, testar a eficiência por meio de métricas como latência e *throughput*, e, por fim, propor melhorias ou ajustes com base nos resultados obtidos.

Por fim, este trabalho está estruturado da seguinte forma: a Seção 2 descreve em detalhes os materiais e métodos utilizados, de modo a garantir a reprodutibilidade e validade do estudo; a Seção 3 reúne a discussão dos resultados obtidos, permitindo a interpretação crítica; e, finalmente, a Seção 4 apresenta as conclusões, destacando as principais contribuições do trabalho, bem como possíveis limitações e direções para pesquisas futuras.

2 MATERIAIS E MÉTODOS

A implementação dos ambientes de teste contou com um conjunto integrado de tecnologias, cada uma com funções complementares no gerenciamento, monitoramento e avaliação dos *clusters*. Os experimentos foram executados em uma máquina com a seguinte configuração de hardware: processador Intel Core i5 13ª geração, 8 GB de memória RAM e disco SSD de 256 GB. O sistema operacional utilizado foi o Ubuntu 24.04.2 LTS (64 bits), garantindo um ambiente estável e compatível com as ferramentas empregadas. O Patroni (v3.0.2) foi utilizado como solução de orquestração de alta disponibilidade para o PostgreSQL (v15.4), automatizando processos como eleição de líderes, configuração de réplicas e recuperação em caso de falha. Para manter consenso distribuído entre os nós, empregou-se o Etcd (v3.5.9), um repositório chave-valor distribuído que registra o estado do *cluster* e garante a consistência das decisões tomadas pelo Patroni. A infraestrutura foi construída em contêineres com o Docker

Engine (v24.0.5), o que proporcionou isolamento e portabilidade dos serviços, enquanto o Docker Compose (v2.20.2) foi utilizado para descrever e gerenciar múltiplos serviços de forma declarativa, facilitando a inicialização e a replicação dos ambientes de teste.

Embora bancos de dados relacionais como o PostgreSQL apresentem desafios de escalabilidade horizontal e de evolução de esquema sem impacto operacional, a adoção de mecanismos de alta disponibilidade mitiga grande parte dessas restrições (McAllister, 2025). Ao combinar orquestração com Patroni e consenso distribuído via Etcd, somados às estratégias de replicação síncrona, assíncrona e lógica (BDR), é possível elevar resiliência, continuidade de serviço e capacidade de atendimento sob carga sem necessidade imediata de migração para soluções NoSQL apenas por motivos de disponibilidade. Essa abordagem preserva o modelo relacional/SQL e suas garantias ACID, ao mesmo tempo em que fornece elasticidade operacional compatível com ambientes críticos.

O monitoramento foi realizado com o Prometheus (v2.48.0), uma ferramenta de coleta de métricas em tempo real que utiliza o modelo de *pull*, consultando periodicamente os serviços monitorados para registrar indicadores como latência de replicação, *throughput* e status de nós. Essas métricas foram visualizadas no Grafana (v10.2.3), que permite a criação de *dashboards* interativos e personalizáveis, possibilitando uma análise detalhada do comportamento dos sistemas em diferentes cenários de carga. Para avaliação de desempenho, empregou-se o Artillery (v2.0.0-29), um *framework* de testes de carga que simula requisições de clientes em padrões diversos (contínuo, rajada e estresse), fornecendo estatísticas como tempo de resposta, taxa de sucesso e número de erros. Dessa forma, a combinação dessas ferramentas permitiu construir um ecossistema controlado, reprodutível e com observabilidade completa, adequado para a análise experimental proposta.

2.1 REPLICAÇÃO SÍNCRONA

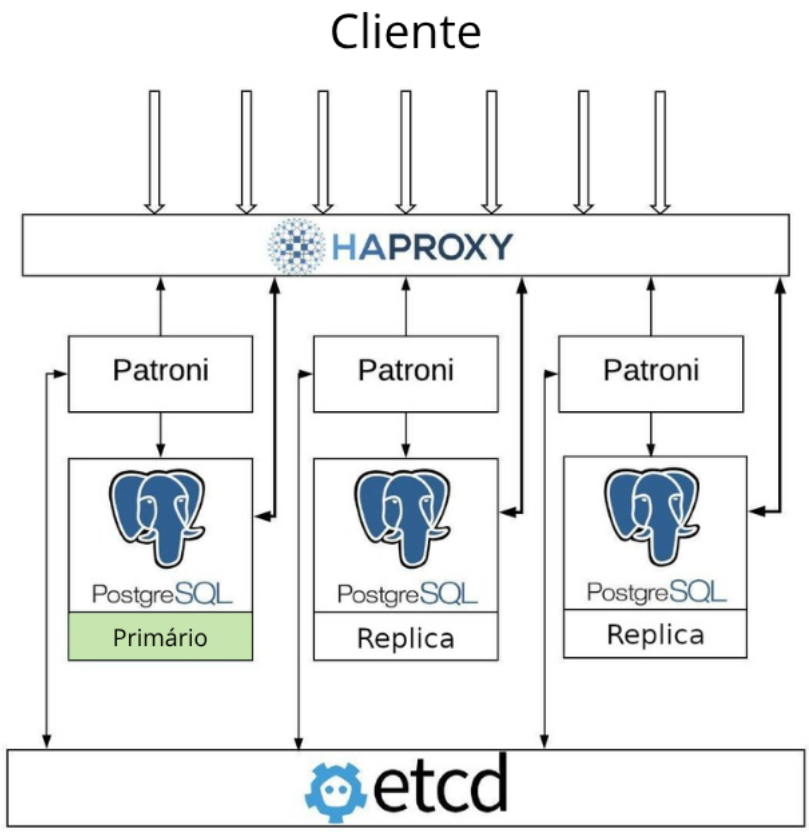
A configuração do *cluster* com replicação síncrona foi projetada para garantir consistência forte entre os nós. Nesse ambiente, as transações só são confirmadas após serem replicadas com sucesso em pelo menos uma réplica. Para isso, o parâmetro *synchronous commit* foi ativado, e o *synchronous standby names* foi configurado para exigir a confirmação de uma réplica. O *cluster* foi composto por três nós PostgreSQL, denominados postgres-patroni-sync-1, postgres-patroni-sync-2 e postgres-patroni-

sync-3. Cada nó foi configurado para operar em portas distintas, garantindo isolamento e comunicação eficiente. O Etcd foi configurado com três nós (etcd1-sync, etcd2-sync e etcd3-sync), responsáveis por gerenciar a coordenação distribuída e garantir a consistência do *cluster*, um fluxograma do funcionamento do patroni pode ser visualizado na Figura 3.

2.2 REPLICAÇÃO ASSÍNCRONA

O *cluster* com replicação assíncrona foi configurado para priorizar o desempenho de escrita, aceitando eventual inconsistência temporária entre os nós. Nesse ambiente, o parâmetro *synchronous commit* foi desativado, permitindo que as transações fossem confirmadas imediatamente no nó líder, sem aguardar a replicação para as réplicas. Assim como no ambiente síncrono, foram utilizados três nós PostgreSQL (postgres-patroni-async-1, postgres-patroni-async-2 e postgres-patroni-async-3) e três nós Etcd (etcd1-async, etcd2-async e etcd3-async).

Figura 3 - Fluxograma Patroni



Fonte: EDB[5].

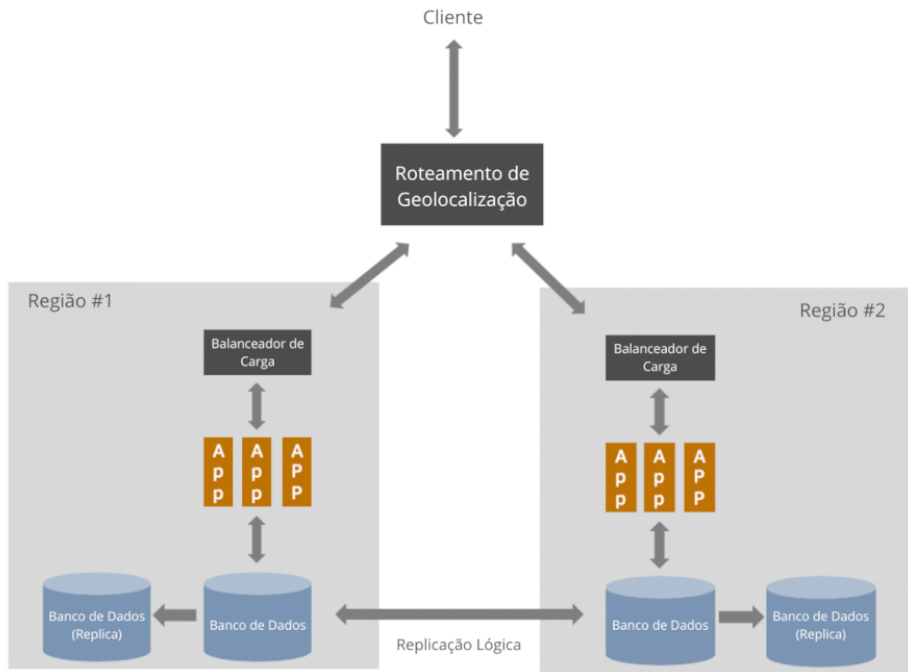
[5]https://www.enterprisedb.com/blog/dos-donts-postgres-high-availability-pt-3-tools-rules

O monitoramento e os testes de carga seguiram a mesma metodologia aplicada ao *cluster* síncrono, alterando-se apenas os nomes dos serviços e o parâmetro *synchronous_commit*, não havendo necessidade de modificar quaisquer outros parâmetros operacionais.

2.3 REPLICAÇÃO BIDIRECIONAL

O ambiente com replicação bidirecional, representado na Figura 4, foi configurado para permitir que todos os nós do *cluster* fossem capazes de realizar operações de leitura e escrita, com replicação automática das alterações entre os nós. Este ambiente foi implementado utilizando a extensão BDR, que oferece suporte nativo à replicação bidirecional no PostgreSQL. O *cluster* foi composto por três nós PostgreSQL, configurados para operar em portas distintas e com comunicação direta entre si. A configuração incluiu a criação de *slots* de replicação e a definição de políticas de resolução de conflitos, uma vez que a replicação bidirecional pode gerar conflitos em cenários de escrita concorrente. O arquivo `docker-compose-multi-master.yml` foi utilizado para definir os serviços e suas dependências, enquanto o script `init-bdr.sh` foi responsável por configurar o ambiente inicial, incluindo a criação de bancos de dados e tabelas de teste.

Figura 4 - Fluxograma BDR



Fonte: AWS[6].

[6]https://aws.amazon.com/pt/blogs/database/postgresql-bi-directional-replication-using-pglogical/

2.4 CARGA DE DADOS E TESTES DE DESEMPENHO

A carga de dados nos ambientes de teste foi realizada de forma automatizada usando o framework Artillery. Antes do início dos testes, foram criadas duas tabelas idênticas em todos os bancos, *test_table* e *metrics_table*. A *test_table* foi pré-carregada com 10.000 registros por nó (mesmo esquema e distribuição em todos os cenários), enquanto a *metrics_table* iniciou vazia e recebeu registros somente para telemetria dos testes. Para cada cenário de replicação, foram desenvolvidos *scripts* específicos que simulam padrões de requisições como rajadas (*burst*), carga contínua e testes simples, permitindo avaliar o comportamento dos *clusters* sob diferentes níveis de estresse. Os *scripts* definem parâmetros como número de usuários virtuais, taxa de chegada e duração dos testes, além de coletar estatísticas detalhadas sobre tempo de resposta, taxa de sucesso e ocorrência de erros. Os resultados obtidos foram posteriormente correlacionados com as métricas de monitoramento coletadas pelo Prometheus e visualizadas no Grafana, proporcionando uma visão abrangente do desempenho e da consistência dos ambientes testados.

2.5 CRITÉRIOS DE AVALIAÇÃO DAS TÉCNICAS DE REPLICAÇÃO

Para a análise comparativa, cada ambiente de replicação foi avaliado de acordo com um conjunto de métricas que permitem observar seus pontos fortes e limitações. O *cluster* síncrono foi analisado principalmente quanto à sua capacidade de garantir consistência forte, ou seja, assegurar que todos os nós tenham exatamente os mesmos dados no momento da confirmação de uma transação. Nessa abordagem, uma métrica considerada positiva é a ausência de divergências entre os nós, mesmo em situações de falha. Por outro lado, valores elevados de latência em operações de escrita são interpretados como resultado negativo, já que comprometem a responsividade da aplicação.

O *cluster* assíncrono foi avaliado em termos de *throughout* e tolerância de falhas de rede. Uma taxa alta de operações por segundo, acompanhada de tempos de resposta baixos na maioria das requisições, é entendida como indicador de bom desempenho. No entanto, a ocorrência de janelas de inconsistência durante o *failover* ou a possibilidade de perda de dados recentes é vista como um aspecto crítico, ainda que esperado nesse modelo.

O ambiente configurado com BDR foi analisado pela sua flexibilidade em permitir operações de escrita em múltiplos nós e pela capacidade

de resolver conflitos de replicação. Nesse caso, métricas favoráveis são a manutenção da consistência entre nós após operações concorrentes e a baixa incidência de conflitos não resolvidos. Já um alto número de transações abortadas ou *deadlocks* durante cargas simultâneas é interpretado como sinal de limitação prática da arquitetura.

As métricas de latência foram analisadas por percentis (p50, p95, p99), que indicam valores abaixo dos quais se encontram 50%, 95% e 99% das requisições, respectivamente. A mediana (p50) resume o comportamento típico, enquanto p95 e p99 evidenciam as caudas da distribuição, os eventos menos frequentes porém mais lentos. Em termos práticos, p95 é o limite que apenas 5% das requisições excedem; p99 aproxima o "pior caso" operacional, destacando latências excepcionais sob pico ou falhas. Analisar as caudas permite localizar gargalos intermitentes, avaliar impactos de *failover* e orientar ajustes que reduzam erros e tempos excessivos em condições adversas.

Esse conjunto de critérios estabelece as bases para a discussão dos resultados experimentais, permitindo identificar com clareza os *trade-offs* entre consistência, desempenho e disponibilidade em cada abordagem e fornecendo exemplos concretos do que representa um comportamento desejável ou problemático em termos de replicação de dados.

O código-fonte e os arquivos de configuração utilizados neste trabalho estão disponíveis no repositório Git[7].

3 RESULTADOS E DISCUSSÃO

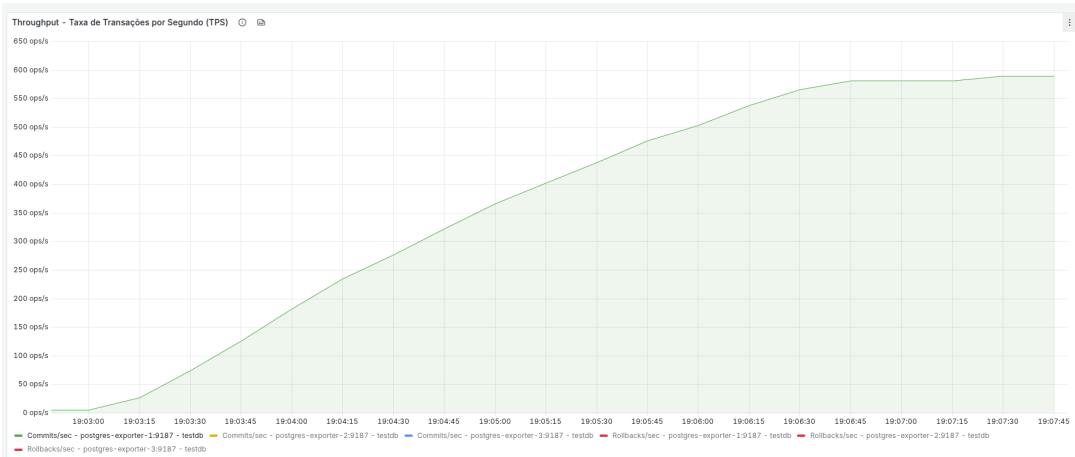
A análise comparativa das três estratégias de replicação, assíncrona, síncrona e bidirecional, evidenciou de forma clara os compromissos entre desempenho, consistência e resiliência em sistemas distribuídos. Nos experimentos realizados, a replicação assíncrona apresentou o melhor equilíbrio entre latência e *throughput*, conforme sintetizado na Tabela 1 e na Tabela 2, com tempos de resposta medianos na ordem de poucos milissegundos mesmo sob elevada concorrência e um volume de aproximadamente 493 operações por segundo. Além disso, não foram observadas falhas de transação ou inconsistências entre os nós, confirmando a robustez dessa abordagem em cenários nos quais pequenas janelas de risco de perda de dados são toleráveis em troca de maior responsividade.

A replicação síncrona, por sua vez, demonstrou elevada estabilidade e ausência de falhas, mas com impacto significativo na latência.

[7] https://github.com/Enricomaine/replication-tests

Embora o *throughput* tenha se mantido alto, próximo de 581 operações por segundo conforme mostrado na Figura 5, as latências nos percentis superiores atingiram valores altos, com p95 em torno de sete segundos e p99 próximo de oito segundos em operações de escrita. Esse comportamento confirma que a confirmação entre líder e réplicas impõe penalidades importantes na responsividade, ainda que garanta consistência forte no instante do *commit*. Dessa forma, trata-se de uma solução adequada para sistemas críticos em que a durabilidade dos dados é requisito central, mesmo que isso comprometa a experiência do usuário em operações mais lentas.

Figura 5 - Resultados Patroni síncrono



Fonte: Do autor.

O ambiente configurado com BDR apresentou resultados heterogêneos. Apesar de uma latência mediana, em torno de quatro segundos, a variabilidade foi elevada, com latência superior a quatro segundos em p99, e a taxa de falhas ultrapassou 80% dos usuários virtuais. Foram observados *deadlocks*, transações abortadas e divergência temporária de registros entre os nós, evidenciando que a arquitetura, embora ofereça potencial de escalabilidade para escrita concorrente, exige ajustes avançados de configuração para alcançar estabilidade em cenários de produção.

Ao relacionar os resultados obtidos com os estudos prévios, pode-se observar uma convergência parcial, mas também divergências relevantes quanto ao comportamento sob carga e à estabilidade dos sistemas. Assim como relatado por Medeiros et al. (2020), que identificaram um aumento expressivo da latência em replicações semissíncronas sob distâncias elevadas entre réplicas, os resultados deste trabalho também confirmam o impacto da sincronização obrigatória sobre a responsividade: a replicação síncrona apresentou p95 superior a 7 segundos, valor significa-

Tabela 1 - Comparação de Latência (ms) - p50/p95/p99

Estratégia	Operação	p50 (ms)	p95 (ms)	p99 (ms)
Patroni Assíncrono	insert	2.0	10.9	16.0
	update	8.9	23.8	34.8
	delete	10.9	29.1	40.0
Patroni Síncrono	insert	186.8	6838.0	7709.8
	update	278.7	7117.0	8024.5
	delete	279.6	7096.5	8692.8
BDR	insert	7.3	41.8	4115.2
	update	9.6	48.7	4486.3
	delete	8.2	45.1	4243.5

Fonte: Do autor.

Tabela 2 - Comparação de Throughput, Consistência e Tempo de Failover

Estratégia	*Throughput (op/s)*	Consistência	*Failover (s)*
Patroni Assíncrono	493	Não foram encontradas divergências	3.8
Patroni Síncrono	581	Não foram encontradas divergências	8.6
BDR	29.6	Apresentou divergência entre nós	25.4

Fonte: Do autor.

tivamente mais alto do que os milissegundos observados no modelo assíncrono. Contudo, enquanto Medeiros et al. (2020) registraram ganhos marginais de desempenho em cenários de baixa carga (até 2 req/s), neste estudo, o ambiente síncrono manteve *throughput* elevado (581 op/s) mesmo com centenas de requisições concorrentes, indicando maior estabilidade do PostgreSQL sob o gerenciamento do Patroni em comparação ao MySQL.

No caso das arquiteturas multimestre, Shrestha e Tandel (2023) relataram que o MySQL NDB Cluster atingiu picos de até 13.000 TPS em cargas mistas de leitura e escrita, superando o PXDB, mas apresentando degradação de desempenho em cenários mais intensos. De forma semelhante, o ambiente BDR deste estudo demonstrou uma boa latência mediana (8 ms), porém queda acentuada de desempenho e alta taxa de falhas (mais de 80% das transações) sob saturação, corroborando parcialmente as limitações apontadas por Shrestha e Tandel (2023). A principal diferença reside na escala e na profundidade das métricas: enquanto os autores anteriores se restringiram a médias globais e número limitado de *threads* (até 72), este trabalho ampliou o escopo experimental para centenas de *threads*

e avaliou latências nos percentis p95 e p99, além do impacto do *failover* automático sob carga ativa, o que permitiu observar de forma mais realista os efeitos de saturação e recuperação em sistemas distribuídos. Assim, os resultados aqui obtidos aproximam-se das condições típicas de produção e complementam as evidências empíricas apresentadas nos estudos correlatos.

Diante dessas observações comparativas, pode-se afirmar que a replicação assíncrona mantém desempenho consistente, mesmo sob alta concorrência, diferindo positivamente dos resultados limitados a baixas taxas de requisição reportados por Medeiros et al. (2020). Já a replicação síncrona confirma os achados desses autores quanto à penalidade de latência, mas demonstra maior resiliência operacional. Por outro lado, o modelo bidirecional reproduz, em parte, as dificuldades de estabilidade registradas por Shrestha e Tandel (2023), reforçando que, apesar do potencial de escalabilidade, a complexidade de resolução de conflitos ainda representa um entrave à adoção em larga escala.

4 CONCLUSÃO

Este trabalho teve como objetivo geral comparar estratégias de replicação de dados em sistemas de alta disponibilidade, avaliando seu desempenho, escalabilidade e confiabilidade. Para isso, foram analisadas três abordagens distintas: replicação síncrona, assíncrona e bidirecional, sob cenários de carga controlada e utilizando métricas de latência, *throughput*, consistência e eficiência do *failover*.

Os resultados mostraram que a replicação assíncrona apresentou o melhor equilíbrio entre latência e capacidade de processamento, alcançando elevado *throughput* e baixa variação nos tempos de resposta, ainda que com o risco inerente de perda de dados em janelas curtas de falha. A replicação síncrona, por sua vez, confirmou sua robustez ao garantir consistência forte e ausência de perdas de dados durante o *failover*, mas com impacto expressivo na responsividade, especialmente em operações de escrita sob concorrência elevada. Já o ambiente bidirecional com BDR demonstrou potencial para escalabilidade horizontal, mas apresentou elevada taxa de falhas e instabilidade sob cargas intensas, evidenciando a necessidade de configurações avançadas e mecanismos mais eficazes de resolução de conflitos.

No que se refere especificamente ao *failover*, observou-se que tanto os ambientes síncronos quanto assíncronos gerenciaram a recupe-

ração de falhas de forma eficiente, com interrupções mínimas na disponibilidade do serviço. Esse resultado reforça a maturidade das soluções baseadas em Patroni e Etcd para automação da alta disponibilidade em PostgreSQL. Contudo, no BDR, os conflitos de escrita prejudicaram a estabilidade após a recuperação, comprometendo a confiabilidade da arquitetura em cenários críticos.

Dessa forma, pode-se concluir que a replicação assíncrona é indicada para sistemas que priorizam desempenho e toleram consistência eventual, enquanto a replicação síncrona se mostra mais adequada para domínios em que a durabilidade das transações é requisito central. O modelo com BDR, embora conceitualmente atraente, ainda demanda investigações adicionais para consolidar-se como alternativa viável em ambientes de missão crítica.

Como limitações deste estudo, destacam-se a não exploração de estratégias avançadas de particionamento lógico no BDR e a ausência de cenários distribuídos geograficamente, que poderiam impactar de forma significativa a latência e o comportamento do *failover*. Como trabalhos futuros, sugere-se avaliar abordagens híbridas que combinem replicação síncrona entre nós locais e assíncrona entre nós remotos, além de explorar o uso de protocolos alternativos de consenso, como Raft e TAPIR, em comparação direta com as métricas aqui aplicadas.

REFERÊNCIAS

ARTILLERY.IO. **Artillery - Modern Load Testing for Developers and SREs**. 2024. Disponível em: <https://www.artillery.io>. Acesso em: 15 jun. 2025.

BERNSTEIN, P. A.; NEWCOMER, E. **Principles of transaction processing**. [S.l.]: Morgan Kaufmann, 2009.

DEAN, J.; GHEMAWAT, S. **MapReduce: simplified data processing on large clusters**. [S.l.]: ACM New York, NY, USA, 2008. 107–113 p.

EDB. **BDR - Bi-Directional Replication for PostgreSQL**. 2023. Disponível em: <https://www.2ndquadrant.com/en/resources/pgbdr-bidirectional-replication-for-postgresql/>. Acesso em: 15 jun. 2025.

GILBERT, S.; LYNCH, N. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services**. [S.l.]: ACM New York, NY, USA, 2002. 51–59 p.

HOLMAN, P.; ANDERSON, J. H. **Locking under pfair scheduling**. [S.l.]: ACM New York, NY, USA, 2006. 140–174 p.

LABS, G. **Grafana Documentation**. 2024. Disponível em: <https://grafana.com/docs/>. Acesso em: 15 jun. 2025.

MCALLISTER, C. **The limitations of PostgreSQL in financial services**. 2025. Acesso em: 02 dez. 2025. Disponível em: <https://www.cockroachlabs.com/blog/limitations-of-postgres/>.

MCMAHON, P. L. The physics of optical computing. **Nature Reviews Physics**, Nature Publishing Group UK London, v. 5, n. 12, p. 717–734.

MEDEIROS, W. et al. **Avaliaçao experimental de replicaçao em banco de dados para recuperaçao de desastres**. 2020. 121–132 p.

MYO, A. K.; BAIN, A. M.; PORTNOV, E. M. Development of a load distribution model for information processing centers in automated power supply control systems. In: IEEE. **2023 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)**. [S.l.], 2023. p. 1045–1050.

PASIN, M. Réplicas para alta disponibilidade em arquiteturas orientadas a componentes com suporte de comunicação de grupo.

PostgreSQL Global Development Group. **PostgreSQL Documentation**. 2023. Disponível em: <https://www.postgresql.org/docs/>. Acesso em: 15 jun. 2025.

SHRESTHA, R.; TANDEL, T. **An Evaluation Method and Comparison of Modern Cluster-Based Highly Available Database Solutions.** 2023. 131–138 p.

STEEN, M. V. **Distributed systems principles and paradigms**. 2002. 1 p.

TERRY, D. B. et al. **Managing update conflicts in Bayou, a weakly connected replicated storage system**. [S.l.]: ACM New York, NY, USA, 1995. 172–182 p.

VOGELS, W. **Eventually Consistent: Building reliable distributed systems at a worldwide scale demands trade-offs? between consistency and availability.** [S.l.]: ACM New York, NY, USA, 2008. 14–19 p.

ZALANDO. **Patroni - A template for PostgreSQL High Availability with ZooKeeper, etcd or Consul**. 2023. Disponível em: <https://github.com/zalando/patroni>. Acesso em: 15 jun. 2025.