Avaliação Experimental de Replicação em Banco de Dados para Recuperação de Desastres

Wilson Medeiros[1], **Júlio Mendonça**[2], **Gabriel Alves**[3], **Ermeson Andrade**[1]

[1]DC, Universidade Federal Rural de Pernambuco (UFRPE)
Recife – PE – Brasil

[2]CIn, Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brasil

[3]DEINFO, Universidade Federal Rural de Pernambuco (UFRPE)
Recife – PE – Brasil

{wilson.medeiros, gabriel.alves, ermeson.andrade}@ufrpe.br

jrmn@cin.ufpe.br

Abstract. *IT systems are essential for the operations of any modern business. Such systems must support operations of their corresponding company under any conditions. Disaster Recovery (DR) strategies have been implemented to help organizations mitigate unexpected failures and reduce unnecessary expenses. However, to the best of our knowledge, no other work experimentally analyzes data replication at the database layer with a focus on DR strategies. Therefore, this work evaluates a relational database replication as a mean of implementing a DR solution. We use a real testbed in a public cloud environment to perform extensive experiments aimed at implementing the replication provided by MySQL, considering various scenarios in the context of DR. Our results show how response time, Recovery Point Objective (RPO) and Recovery Time Objective (RTO) vary according to the size of the replicated data, the synchronization type (ex.: asynchronous or semisynchronous) and the configuration of the slave servers. This work can assist DR coordinators or individuals to decide which database replication configuration for disaster recovery is best for their work environment.*

Resumo. *Os sistemas de TI são essenciais para as operações de qualquer negócio moderno. Tais sistemas precisam suportar as operações de suas empresas correspondentes sob quaisquer condições. Estratégias de Recuperação de Desastres (RD) têm sido implementadas para auxiliar as organizações a mitigar falhas inesperadas e reduzir gastos desnecessários. No entanto, no melhor do nosso conhecimento, nenhum trabalho analisa experimentalmente a replicação de dados na camada de banco de dados (BD) com foco em estratégias de RD. Desta forma, este trabalho avalia a replicação em BDs relacionais como uma forma de implementar uma solução de RD. Para isso, nós utilizamos um ambiente de testes real em nuvem pública para executar experimentos extensivos visando a implementação da replicação fornecida pelo MySQL, considerando vários cenários no contexto de RD. Nossos resultados mostram como o tempo de resposta, o Recovery Point Objective (RPO) e o Recovery Time Objective*

(RTO) variam de acordo com o tamanho dos dados replicados, a configuração da replicação (ex.: assíncrona ou semissíncrona) e a configuração das réplicas. Este trabalho pode auxiliar os coordenadores de RD ou indivíduos a decidir qual configuração de replicação de banco de dados para recuperação de desastres é melhor para seu ambiente de trabalho.

1. Introdução

Desastres muitas vezes podem não ser previstos ou evitados. Qualquer organização está propensa à ocorrência de desastres que podem resultar em danos catastróficos [Reese 2009, Andrade et al. 2017]. Em empresas que necessitam de sistemas de Tecnologia da Informação e Comunicação (TICs), além do possível prejuízo físico, a interrupção das operações pode resultar em perdas financeiras consideráveis. Uma pesquisa realizada pela Zetta [Zetta 2016] aponta que 67% das empresas pesquisadas poderiam ter um prejuízo acima de US$ 20.000,00 por dia de indisponibilidade. Um outro estudo realizado pela Unitrends aponta que um grande número de empresas ainda sofrem taxas de perda de dados ou tempo de interrupção acima do aceitável [Unitrends 2019].

Estratégias de Recuperação de Desastres (RD) vêm sendo adotadas para mitigar a perda de dados e garantir a continuidade das operações das empresas [Andrade et al. 2017]. Existem diversas técnicas que podem ser aplicadas como estratégia de RD (ex.: *backup* ou migração de máquinas virtuais). Em especial, a replicação como estratégia de RD tem sido usada por várias empresas, uma vez que a mesma visa garantir redundância de dados em sistemas de TICs [Mendonça et al. 2019]. Essa estratégia possibilita, por exemplo, manter um ou mais servidores secundários (Réplicas ou *Slaves*) atualizados em relação a um servidor primário (*Master*). Assim, se o servidor primário falhar, uma réplica poderá assumir o seu papel.

Apesar dos bancos de dados *NoSQL* terem surgido como alternativa aos relacionais, oferecendo melhor desempenho e escalabilidade, os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) ainda são os mais utilizados mundialmente. Sendo o *MySQL* o SGBDR de código aberto mais utilizado [DB engines 2020, Shay 2018]. Nesse sentido, alguns trabalhos têm sido desenvolvidos para avaliar ou comparar diferentes Sistemas Gerenciadores de Banco de Dados (SGBDs) [Jogi and Sinha 2016, Santana et al. 2016, Wang et al. 2014]. No entanto, no melhor do nosso conhecimento, nenhum dos trabalhos disponíveis na literatura avalia a replicação no contexto de RD.

Este trabalho visa avaliar experimentalmente diferentes cenários utilizando a replicação Primário-Secundário, presente no *MySQL*, para fins de RD. Algumas métricas cruciais para a RD são estimadas, tais como: *Recovery Time Objective* (RTO), *Recovery Point Objective* (RPO), além de tempo de resposta. Nós realizamos testes de carga em um ambiente de testes real para coletar essas métricas, considerando uma gama de diferentes cenários. Desta forma, os resultados focados em RD que apresentam o real comportamento de um sistema que utiliza replicação de dados em BD relacionais são apresentados. Também analisamos o impacto obtido no tempo de resposta ao utilizar diferentes configurações de replicação. Por fim, verificamos que, com as cargas consideradas, não houve sobrecarga relevante ao utilizar uma ou duas réplicas, o que possibilita a adição de redundâncias para o BD primário com pouco ou nenhum impacto no desempenho.

O restante deste trabalho está organizado da seguinte maneira: a Seção 2 apre-

senta os conceitos básicos de RD e descreve os diferentes tipos de replicação no *MySQL*. A Seção 3 apresenta alguns trabalhos relacionados. A Seção 4 detalha como os experimentos e medições foram realizados. A Seção 5 discute os resultados obtidos. A Seção 6 conclui o trabalho e cita possíveis trabalhos futuros.

2. Fundamentos

Esta seção apresenta os principais conceitos para um melhor entendimento do trabalho.

2.1. Recuperação de Desastres

Qualquer infraestrutura computacional ou sistemas de TIC está vulnerável a um conjunto de interrupções. Algumas dessas vulnerabilidades podem ser eliminadas, ou pelo menos minimizadas, através do uso de estratégias de RD [Mendonça et al. 2019]. Essas estratégias proveem soluções apropriadas para evitar perda de dados e/ou diminuir o tempo para a recuperação dos serviços depois de uma interrupção [Bauer et al. 2011]. Assim, podemos definir a RD como a prática de tornar sistemas capazes de sobreviver a falhas inesperadas ou extraordinárias [Reese 2009].

Na análise de soluções de RD, duas métricas são primordiais, o RPO e o RTO. O RPO aponta a quantidade máxima de dados que pode ser perdida desde a realização do último backup até a ocorrência de uma falha, enquanto o RTO representa o tempo máximo necessário para a recuperação do serviço logo após uma interrupção inesperada [Reese 2009]. Assim, conhecer os valores dessas métricas pode auxiliar na escolha das melhores estratégias de RD para indivíduos ou empresas.

2.2. Replicação de Dados no *MySQL*

Essa seção fornece um resumo sobre replicação de dados e detalha os diferentes tipos presentes no *MySQL* 8 [Oracle 2020a]. A replicação de dados é usada para manter a sincronização entre diferentes dispositivos (nós) e é realizada quando ocorrem mudanças nas bases de dados. Os nós podem estar relacionados de diferentes formas:

- **Primário-Secundário**: é a relação mais comumente adotada. Um dos nós é considerado primário, enquanto os demais, nós secundários ou réplicas. O BD primário é o responsável por receber atualizações, aplicá-las e propagá-las a suas réplicas. Essa é a implementação padrão do *MySQL*. Como ela não utiliza protocolos para tratar possíveis conflitos quando mais de um nó recebe atualizações, somente o BD primário pode recebê-las.
- **Group Replication**: Essa relação de replicação visa obter consistência entre os dados dos nós. Nessa replicação, um protocolo deve ser adotado para garantir que todos os nós recebam as mensagens na ordem correta. O *atomic multicast* é um protocolo que pode ser adotado para garantir que todas as mensagens enviadas para um conjunto de nós sejam entregues a todos ou a nenhum deles. *One-phase commit, two-phase commit*, e *three-phase commit* também são apresentados como protocolos de confirmação distribuídos. No *one-phase commit*, utilizado na replicação Primário-Secundário, o BD primário propaga as atualizações mas não há uma fase para confirmar se as réplicas realizaram as mesmas com sucesso. Os protocolos de *two-phase commit* e *three-phase commit* adicionam mais uma etapa, permitindo que os nós decidam sobre uma transação. Assim, os nós tentam alcançar um consenso sobre confirmar ou descartar a transação recebida.

Neste trabalho, abordamos a replicação Primário-Secundário. Ela é realizada em três principais etapas: ***(1)*** ao receber uma atualização, o BD primário salva a transação em seu *binary log* e a envia a suas réplicas; ***(2)*** as réplicas, ao receber a transação, escrevem-na em seu *relay log*[1]; e ***(3)*** uma ou mais *applier threads* aplicam a transação na réplica, salvando-a em seus respectivos *binary logs*, persistindo os dados (realizando *commit* nas transações) em seguida.

A replicação Primário-Secundário pode ser configurada para funcionar de forma **assíncrona** ou **semissíncrona**. Na replicação **assíncrona** o BD primário persiste uma transação sem nenhum tipo de sincronização com as réplicas, enquanto na **semissíncrona** existe uma sincronização na etapa ***(2)*** explicada anteriormente. Nessa sincronização, o BD primário espera por um aviso de recebimento, ou *acknowledgement* (*ack*), antes de dar uma resposta ao usuário e persistir a transação. Uma réplica comunica um *ack* ao salvar todos os metadados da transação em seu *relay log* (dados não persistidos ainda). Por esse motivo essa configuração é denominada semissíncrona. Além disso, o BD primário espera pelos *acks* por um tempo pré-definido, que caso seja alcançado, a replicação é reconfigurada automaticamente para a assíncrona. Em um sistema real, para evitar perda de dados, deve-se desabilitar essa reconfiguração automática da replicação semissíncrona para a assíncrona. Essa desabilitação, porém, degradaria a disponibilidade do sistema, uma vez que não é possível realizar operações de escrita enquanto não houver réplicas suficientes. A quantidade de *acks* necessários para que o BD primário persista uma transação também pode ser definida. Portanto, seria necessário utilizar mais réplicas e um número de *acks* menor que o total delas para mitigar esse problema. A decisão de utilizar ou não uma reconfiguração automática fica a critério de quem implementa a estratégia de RD. É importante perceber que o processo de sincronização tem impacto direto no tempo de resposta, visto que o BD primário aguardará a sincronização para dar resposta ao usuário. Mas o impacto não é tão grande quanto ao utilizar *Group Replication*.

3. Trabalhos Relacionados

Nesta seção, são apresentados alguns trabalhos que têm sido desenvolvidos para análise de BDs. Os autores [Wang et al. 2014], [Jogi and Sinha 2016] e [Santana et al. 2016] realizaram experimentos comparando diferentes implementações de BDs. [Wang et al. 2014] analisou a replicação utilizando dois BDs *NoSQL*: HBase e Cassandra. A abordagem utilizada focou em analisar os *tradeoffs* entre consistência e desempenho (tempo de resposta e vazão) entre diferentes configurações, além de escalabilidade. Em [Jogi and Sinha 2016], além dos BDs *NoSQL* (*HBase* e *Cassandra*), Jogi e Sinha também utilizaram o *MySQL* (BD relacional) para comparar o desempenho desses BDs em relação as operações de escrita com grandes quantidades de dados. Já [Santana et al. 2016] apresentou um estudo de replicação de BDs para ambientes de computação em nuvem. Através de experimentos, os autores avaliaram diferentes técnicas de replicação, focando principalmente em métricas como tempo de resposta e taxa de falha das transações.

Modelos formais também têm sido empregados para representar e avaliar sistemas com replicações de dados. [Rodrigues et al. 2019] propôs o uso de Redes de Petri Generalizadas (GSPN) [Marsan et al. 1991] para modelar um sistema que utiliza replicação. O trabalho utilizou o BD *NoSQL MongoDB* para avaliar métricas como vazão e disponibili-

[1]O *relay log* funciona como uma fila de transações a serem persistidas na réplica.

dade. No entanto, no modelo adotado os BDs secundários são utilizadas apenas para aumentar a disponibilidade do sistema. No nosso trabalho anterior [Mendonça et al. 2019], avaliamos a replicação de dados em BDs relacionais através de modelos formais. No entanto, experimentos foram conduzidos apenas com o objetivo de parametrizar os modelos desenvolvidos.

Os trabalhos acima citados avaliam o uso de replicação de dados em BDs sem considerar características de RD, como as métricas de RTO e RPO. Diferentemente desses trabalhos, neste artigo avaliamos a replicação de dados em um BD relacional através de testes experimentais considerando diferentes cenários. No nosso estudo, consideramos a análise de RTO e RPO, verificando o comportamento das diferentes configurações da replicação de dados no *MySQL* 8, além de avaliar conjuntamente o desempenho delas através da métrica de tempo de resposta.

4. Arquitetura Experimental

Esta seção detalha a metodologia e o ambiente utilizada na realização dos experimentos.

4.1. Ambiente de testes

Essa seção descreve como o ambiente de teste foi configurado para realização dos experimentos. O ambiente de testes foi configurado em uma nuvem pública, o *Google Cloud* [Google 2020]. Assim, todo o ambiente foi virtualizado. No total, quatro Máquinas Virtuais (VMs) foram utilizadas: uma delas responsável por gerar a carga dos testes, representando requisições de usuários, foi configurada com o *Apache JMeter* [Halili 2008]; Outra, realizando o papel de servidor de BD primário; e as outras representando servidores de BD secundários. Todas as VMs configuradas como servidor de BD utilizaram *MySQL* 8 como SGDB.

Utilizamos dois servidores como réplicas para verificar se existe diferença significativa ao utilizar réplicas em locais geograficamente diferentes (mais ou menos distantes do nó primário). Além disso disso, essa configuração também foi utilizada para verificar se há impacto no processo de replicação ao utilizar uma ou duas réplicas. Dessa forma, as localizações das VMs foram definidas da seguinte maneira: a VM configurada com o *JMeter*, responsável por representar o envio de requisições de usuários, ficou na região *west US-region*; o servidor de BD primário, na região *center US-region*; uma das réplicas, na região *east US-region* enquanto a outra réplica, na região *Europe-west4-a*. Todas as VMs utilizaram como sistema operacional o *Ubuntu 18.04 LTS*, tinham 2 v-CPUs, RAM de 7,5GB e SSD de 80GB.

4.2. Execução dos experimentos e Medições

Essa seção detalha como os experimentos foram conduzidos e quais dados foram coletados. Utilizando o ambiente de testes explicado anteriormente, utilizamos a ferramenta Apache *JMeter* [Halili 2008] para coletar dados no ambiente configurado.

Como a replicação ocorre quando há alteração nos dados, a carga de trabalho que utilizamos contém apenas atualizações, desconsiderando a leitura de dados. Assim, a carga de trabalho foi caracterizada da seguinte forma: Usuários realizam operações SQL de *insert* e *update* durante um período de tempo *T*. Neste período de tempo, várias operações são realizadas pelos mesmos usuários. Na primeira operação de um usuário, ele

realiza um *insert*, nas demais operações um *update* no registro já gerado. Essas operações de *insert* e *update* inserem e modificam um campo do tipo *blob* (dado binário de tamanho variável) [Oracle 2020b] em uma única tabela no BD primário. Essa tabela possui como campos, apenas uma chave primária (do tipo inteiro) e um campo do tipo *blob*.

Dessa forma, coletamos os seguintes dados:

- **Tempo de resposta**: É o tempo decorrido para dar uma resposta ao usuário. Esse valor é disponibilizado pelo *JMeter*. Essa métrica é relacionada ao desempenho do ambiente. O objetivo é verificar como ela é afetada ao ativar as diferentes configurações de replicação.
- ***Master commit* e *Slave commit***: Representam os tempos nos quais uma transação foi salva no *binary log* do BD primário e do BD secundário, respectivamente. Esses valores podem ser coletados através da realização de um *parsing* nos *binary logs* do BD primário e secundário.
- ***Relay start* e *Relay end***: Representam os tempos nos quais a transação começou e finalizou de ser escrita no *relay log* da réplica, respectivamente. Esses valores são disponibilizados no *MySQL* 8 através do *performance_schema*. Porém, como somente os valores da última transação podem ser consultados, coletamos as amostras através de uma *thread* no *JMeter* que realiza a consulta em intervalos de 1 segundo.

4.3. Cenários Analisados

Nesta seção são descritos os diferentes cenários analisados nos experimentos. Foram adotados os seguintes parâmetros para a composição dos cenários:

- **Tipo da sincronização:** assíncrona ou semissíncrona.
- **Configuração das réplicas:** Se o tipo da sincronização for assíncrona:
  - 1 réplica perto do BD primário (*east US-region*);
  - 1 réplica longe do BD primário (*Europe-west4-a*);
  - 2 réplicas;

  Se o tipo da sincronização for semissíncrona:
  - 1 réplica perto do BD primário (*east US-region*);
  - 1 réplica longe do BD primário (*Europe-west4-a*);
  - 2 réplicas e 1 *ack*;
  - 2 réplicas e 2 *acks*.
- **Carga dos usuários:**
  - 0,1 requisições por segundo (0,1 req/s);
  - 2 requisições por segundo (2 req/s).

  Para a carga de 0,1 req/s, 1 requisição a cada 10 segundos é enviada por 1 usuário e tem duração de 1000 segundos. Já para a carga de 2 req/s, as requisições são enviadas por 50 usuários, com intervalos gerados aleatoriamente pelo *JMeter*, de forma a alcançar a vazão desejada de 2 req/s e tem duração de 240 segundos.
- **Tamanho do *blob* enviado:** 500 B, 25 KB, 50 KB, 75 KB, 100 KB.

Cada cenário analisado, é formado pela combinação desses parâmetros. Isto resulta em 30 cenários para a replicação assíncrona e 40 cenários para a replicação semissíncrona, totalizando 70 cenários analisados. Além disso, levando em consideração que a latência pode sofrer variações no ambiente utilizado, cada cenário foi executado duas vezes num intervalo de aproximadamente 13 horas. Deste modo, são geradas duas amostras para cada cenário.

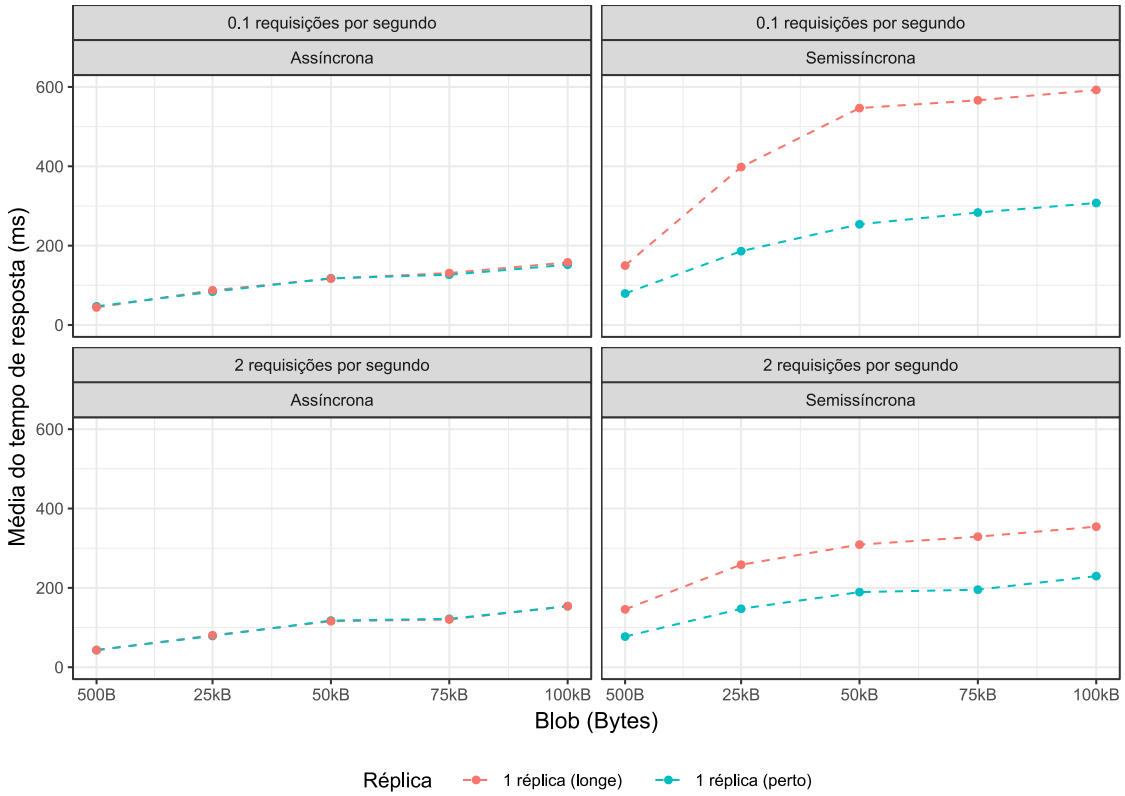

Figura 1. Tempo de resposta médio apresentado para diferentes tamanhos de *blob* e configurações.

5. Resultados Experimentais e Discussão

Esta seção discute os resultados obtidos através dos experimentos realizados no ambiente adotado onde, mais especificamente, são apresentando os resultados referente às métricas de tempo de resposta, RPO e RTO.

5.1. Tempo de Resposta

A Figura 1 apresenta os resultados obtidos para o tempo de resposta ao utilizar 1 réplica perto ou longe em relação ao BD primário. Como pode ser visto na figura, há quatro gráficos que representam combinações de dois parâmetros: ***(1)*** configuração da replicação (assíncrona ou semissíncrona) e ***(2)*** carga dos usuários (0,1 req/seg ou 2 req/s). Note que para cada cenário são apresentados os resultados de tempo de resposta (eixo Y) utilizando tamanhos distintos para o arquivo que é enviado na requisição (*blob* – apresentado no eixo X). Os pontos considerados representam os resultados obtidos na primeira execução dos experimentos.

A partir dos resultados obtidos, podemos fazer algumas observações com relação a replicação assíncrona e semissíncrona. Na replicação assíncrona o tempo de reposta apresentou diferença apenas ao variar o tamanho da requisição que era enviada (*blob*). Isso se deve ao fato do tempo de resposta nesta configuração de replicação depender apenas do quão rápido o servidor primário pode processar uma requisição feita pelo usuário. Por outro lado, na replicação semissíncrona, além do tamanho da requisição enviada (*blob*), houve diferenças significativas com cargas dos usuários e localizações distintas

Tabela 1. Resultados dos testes de hipótese não paramétricos

Comparações entre cenários			
Rep. & workload	**Configurações**	***p-value (latência)***	***p-value (delta relay)***
Assínc. & 0,1 req/s	1 réplica perto x 2 réplicas	0,7786966	0,1240706
	1 réplica longe x 2 réplicas	0,6967137	0,6118413
Assínc. & 2 req/s	1 réplica perto x 2 réplicas	0,2665331	0,1020656
	1 réplica longe x 2 réplicas	0,1237713	0,1413215
Semissínc. & 0,1 req/s	1 réplica perto x 2 réplicas (1 ack)	0,8865125	0,3204590
	1 réplica longe x 2 réplicas (2 acks)	0,8489201	0,9123771
Semissínc. & 2 req/s	1 réplica perto x 2 réplicas (1 acks)	0,6444780	0,1296032
	1 réplica longe x 2 réplicas (2 acks)	0,1002598	0,4275970

das réplicas. Como esperado, esta configuração de replicação apresentou um tempo de resposta maior que a replicação assíncrona, tendo essa diferença atenuada ao aumentar o tamanho da requisição enviada e, da mesma maneira, ao utilizar a réplica mais distante. Além disso, a carga de 0,1 req/s apresentou tempo de resposta maior em relação a carga de 2 req/s. Isso pode ser explicado se considerarmos que as atualizações podem ser enviadas às réplicas em "lotes", diminuindo o tempo de envio de algumas atualizações às réplicas quando se utiliza uma carga requisições maior.

Para comparar os outros cenários nós utilizamos o Teste U de Mann-Whitney [Hollander and Wolfe 1999], que é um teste não paramétrico para comparar duas amostras independentes. Para realizar tal teste, foram juntadas as amostras obtidas nas duas execuções do experimento e também as amostras com tamanhos de *blob* diferentes, a fim de obter uma comparação mais generalizada. O parâmetro comparado foi a **configuração das réplicas** (ver subseção 4.3). A Tabela 1 exibe os resultados do *valor p* para a comparação da latência em cenários distintos. Com um intervalo de confiança de 95%, os resultados do teste indicam que todas essas comparações não apresentam diferenças significativas, pois todos os *valores de p* são maiores que 0,05 (5%). Isso mostra que, com as cargas consideradas, utilizar 2 réplicas ao invés de 1 **não impacta significativamente** no tempo de resposta do sistema.

5.2. *Recovery Point Objective* e *Recovery Time Objective*

Para estimar o RPO, primeiro é preciso identificar em quais cenários algum dado pode ser perdido caso o BD primário falhe. Em um BD relacional, podemos considerar que uma transação foi perdida se o BD primário a persistiu (realizou *commit*) mas falhou antes de enviá-la completamente a ao menos uma réplica. Assim, para estimar tal valor, podemos calcular a diferença entre o tempo *Master commit* (T_{Master_commit}) e o tempo *Relay end* ($T_{Slave\_relay\_end}$) da última transação persistida no BD primário antes da falha. Neste trabalho, denominaremos essa diferença de *Delta Relay* (Δrelay). A Equação 1 detalha esse cálculo.

$$\text{RPO} = \Delta\text{relay} = T_{Slave\_relay\_end} - T_{Master\_commit} \quad (1)$$

Para a replicação assíncrona, consideramos que os valores de Δrelay coletados são os possíveis intervalos do RPO, pois o BD primário poderia vir a falhar em qualquer momento durante a execução. Para a replicação semissíncrona é diferente, pois ela oculta

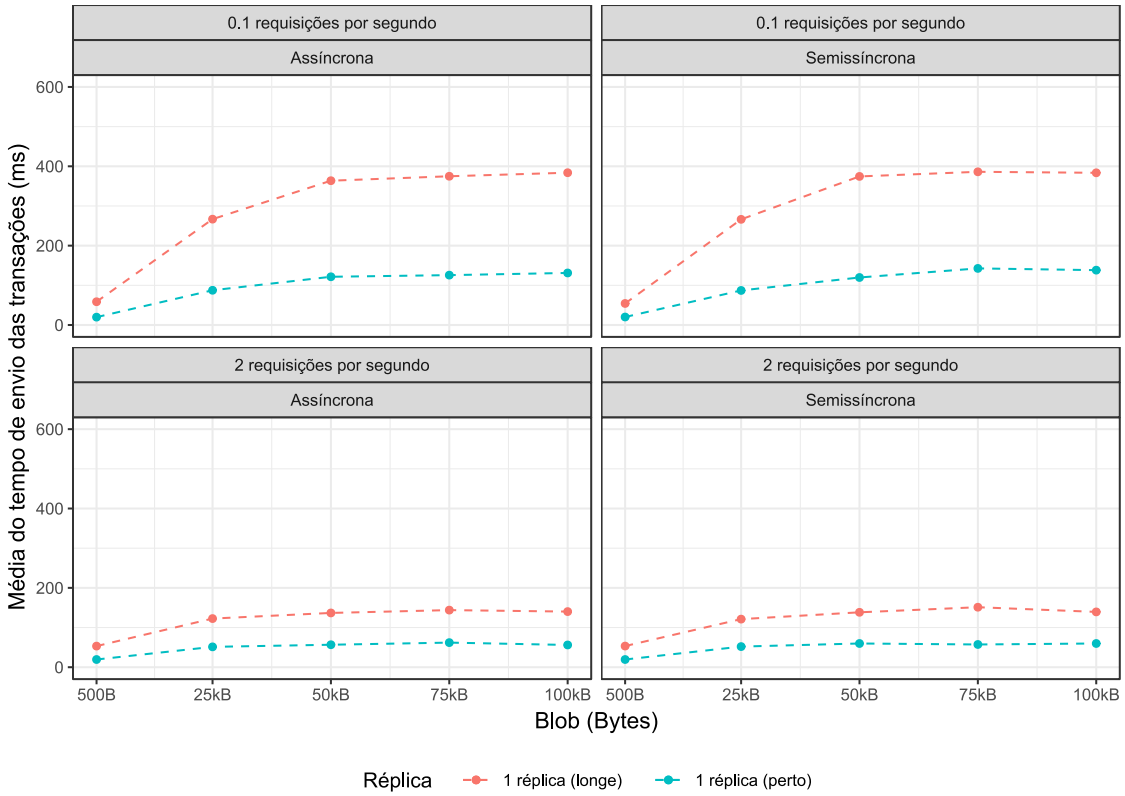

Figura 2. Tempos médios para o Δrelay.

a transação e não dá resposta ao usuário até que um número desejado de réplicas enviem *acks*. Consequentemente, *o RPO é zero enquanto a replicação semissíncrona for usada*.

A Figura 2 apresenta os resultados obtidos para o Δrelay, gerados pelas transações realizadas ao utilizar 1 réplica perto ou longe em relação ao BD primário. O gráfico é estruturado de maneira similar ao do tempo de resposta (ver Figura 1). O envio das transações ocorre em ambas as configurações e, por isso, diferentemente do tempo de resposta, os resultados se comportam de maneira similar tanto na replicação assíncrona quanto na semissíncrona. Apesar disso, o Δrelay de cada configuração influencia diferentemente as métricas de interesse. Na replicação assíncrona, ele indica o intervalo de tempo em que alguma atualização pode ser perdida, ou seja, o RPO. Por outro lado, na replicação semissíncrona, ele indica a sobrecarga esperada no tempo de resposta. Devido à influencia do Δrelay no tempo de resposta da replicação semissíncrona, as ponderações feitas sobre ela na Subseção 5.1 também são válidas aqui. No entanto, a variação do tempo com o aumento do *blob* é mais sutil, pois o Δrelay depende da comunicação feita entre dois nós (*primário-secundário*) ao invés de três nós (*usuário-primário-secundário*), como ocorre para o caso do tempo de resposta com replicação semissíncrona. Por fim, também foram executados testes estatísticos e não houve diferença relevante nesse tempo ao utilizar 1 ou 2 réplicas (ver coluna *p-value (delta relay)*) na Tabela 1.

O RTO refere-se ao tempo que o sistema volta ao estado operacional após a ocorrência de um desastre. Geralmente, um monitor de desastres deve detectar a

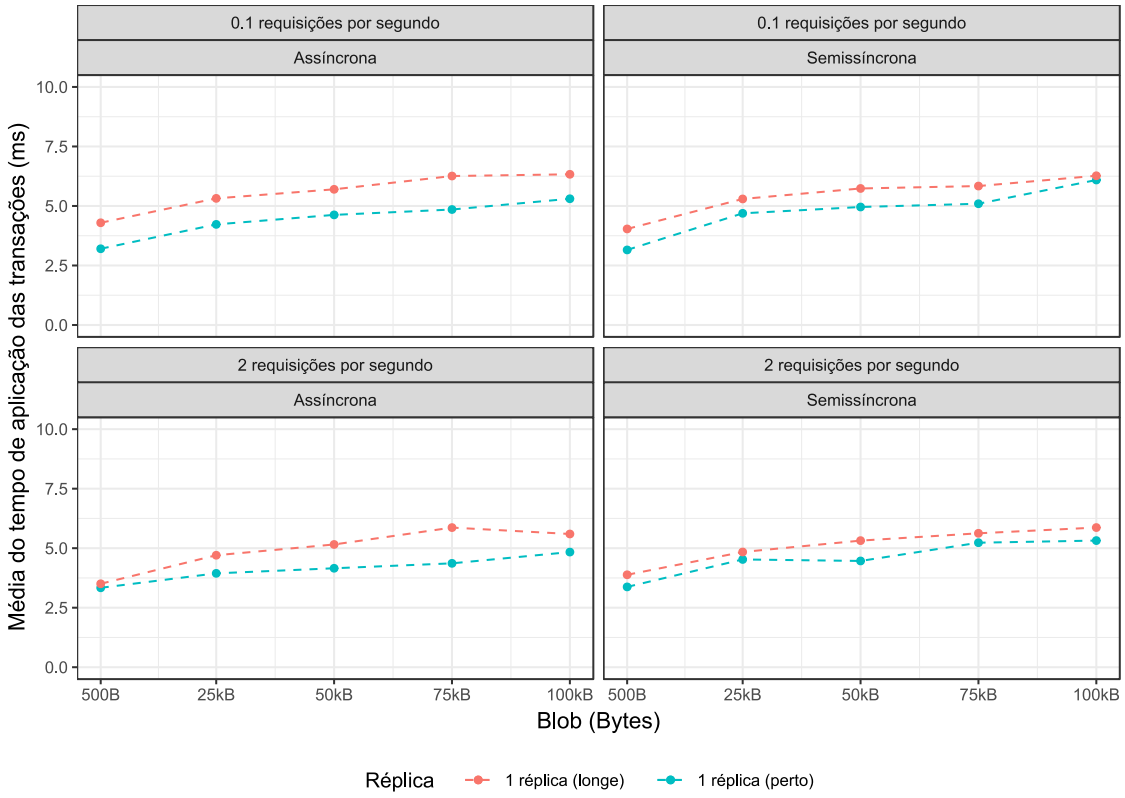

Figura 3. Tempo médio para aplicar uma atualização na fila.

ocorrência de um desastre ou falha do BD primário para que um processo de *failover*[2] se inicie. Ou seja, uma das réplicas deve assumir o papel de BD primário após um desastre. Para que isso ocorra, é necessário esperar que a réplica termine de aplicar todas as transações pendentes (enfileiradas em seu *relay log*). Pois caso novas transações sejam realizadas nele antes disso, seus dados assumirão um estado diferente daqueles no BD primário, além de que as aplicações das transações pendentes podem vir a falhar (pela inconsistência gerada). Isso se deve à falta de consenso entre os nós antes do *commit*. Assim, se após o processo de *failover* ainda houver transações na fila para a réplica aplicar, assumimos que o RTO pode ser calculado pelo tempo que ela levará para aplicar todas as transações já enfileiradas. Isso se dá subtraindo o tempo *Slave commit* (T_{Slave_commit}) do tempo *Relay end* ($T_{Slave\_relay\_end}$) da última transação enfileirada antes do BD primário falhar, como mostrado na Equação 2. Então o RTO seria o máximo entre o tempo de *failover* ($T_{failover}$) e o tempo de aplicação da última transação ($T_{Slave\_apply\_relay}$), como mostrado na (Equação 3).

$$T_{Slave\_apply\_relay} = T_{Slave\_commit} - T_{Slave\_relay\_end} \tag{2}$$

$$RTO = max(T_{Slave\_apply\_relay}, T_{failover}) \tag{3}$$

[2]Processo no qual um servidor secundário assume as operações no lugar de um servidor primário que entrou em estado de falha.

A Figura 3 mostra os resultados obtidos para o T_{apply_relay}, gerados pelas transações realizadas ao utilizar 1 réplica perto ou longe em relação ao BD primário. É importante ser destacado que esse tempo de processamento depende apenas de quão rápido a réplica consegue processar e persistir uma atualização no seu *binary log*, sem necessidade de comunicação com outros nós. Por este motivo, a quantidade de réplicas não influencia no resultado. Além disso, as diferenças entre os cenários exibidos no gráfico foram mais sutis. Dessa forma, o RTO com os parâmetros considerados apresenta diferenças mínimas entre todas as configurações.

5.3. Limitações

Os cenários considerados buscam representar situações reais de interação entre os usuários e o sistema implementado. Porém, nem sempre a carga de trabalho esperada de um sistema será similar às que aqui foram utilizadas. Ademais, há cenários que poderiam apresentar diferenças relevantes que não foram considerados neste trabalho, como utilizar mais de duas réplicas e realizar testes de estresse (sob cargas extremas) a fim de verificar as principais limitações da implementação. Por fim, neste trabalho não realizamos as operações de recuperação de falhas no ambiente adotado, pois seu escopo foca na análise dos mecanismos de replicação.

6. Conclusão e Trabalhos Futuros

Neste trabalho avaliamos a implementação da replicação presente no *MySQL* 8. Através de um sistema implantado em uma nuvem pública e nós geograficamente distantes, testes experimentais foram realizados para este fim. Apresentamos o impacto gerado no desempenho ao utilizar replicação semissíncrona no lugar de assíncrona, considerando nós em localidades diferentes. Também estimamos o RPO esperado ao utilizar replicação assíncrona e verificamos que a utilização de duas réplicas resultou em sobrecarga irrelevante em relação a apenas um com as cargas consideradas. Finalmente, verificamos que o RTO não apresenta diferenças significativas entre os cenários considerados. Espera-se que os resultados apresentados e os pontos destacados auxiliem coordenadores de DR e indivíduos que considerem adotar essa técnica como solução de RD.

Como trabalho futuro, almejamos avaliar e comparar outras implementações presentes em BDs relacionais. Também pretendemos verificar o impacto no desempenho ao utilizar diferentes níveis de consistência entre os nós e propor possíveis alternativas a fim de minimizá-lo. Por fim, objetivamos realizar testes de estresse para melhor apontar as limitações de cada implementação estudada.

Agradecimentos

Esta pesquisa foi parcialmente financiada pelo CNPq - Brasil, processo 406263/2018-3.

Referências

Andrade, E., Nogueira, B., Matos, R., Callou, G., and Maciel, P. (2017). Availability modeling and analysis of a disaster-recovery-as-a-service solution. *Computing*, 99(10):929–954.

Bauer, E., Adams, R., and Eustace, D. (2011). *Beyond Redundancy: How Geographic Redundancy Can Improve Service Availability and Reliability of Computer-Based Systems*. Wiley.

DB engines (2020). Db-engines ranking. [Online]. https://bit.ly/2s90XvI.

Google (2020). Google Cloud. [Online]. https://cloud.google.com.

Halili, E. H. (2008). *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing Ltd.

Hollander, M. and Wolfe, D. (1999). *Nonparametric Statistical Methods*. Wiley Series in Probability and Statistics. Wiley.

Jogi, V. D. and Sinha, A. (2016). Performance evaluation of MySQL, Cassandra and HBase for heavy write operation. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pages 586–590. IEEE.

Marsan, M., Balbo, G., Chiola, G., Conte, G., Donatelli, S., and Franceschinis, G. (1991). An introduction to generalized stochastic petri nets. *Microelectronics Reliability*, 31(4):699 – 725.

Mendonça, J., Medeiros, W., Andrade, E., Maciel, R., Maciel, P., and Lima, R. (2019). Evaluating database replication mechanisms for disaster recovery in cloud environments. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2358–2363. IEEE.

Mendonça, J., Andrade, E., Endo, P. T., and Lima, R. (2019). Disaster recovery solutions for it systems: A systematic mapping study. *Journal of Systems and Software*, 149:511 – 530.

Oracle (2020a). MySQL 8.0 Reference Manual. [Online]. https://dev.mysql.com/doc/refman/8.0/en/.

Oracle (2020b). The BLOB and TEXT Types. [Online]. https://dev.mysql.com/doc/refman/8.0/en/blob.html.

Reese, G. (2009). *Cloud application architectures: building applications and infrastructure in the cloud*. "O'Reilly Media, Inc.".

Rodrigues, M., Vasconcelos, B., Gomes, C., and Tavares, E. (2019). Evaluation of nosql dbms in private cloud environment: An approach based on stochastic modeling. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE.

Santana, M., Armendáriz-Iñigo, J. E., and Muñoz-Escoí, F. D. (2016). Evaluation of Database Replication Techniques for Cloud Systems. *Computing and Informatics*, 34(5):973–995.

Shay, T. (2018). Most popular databases in 2018 according to stackoverflow survey. [Online]. https://bit.ly/2DCwqhj.

Unitrends (2019). Data protection, cloud, and proof draas delivers – unitrends 2019 survey results. Tech Report. [Online]. https://bit.ly/2XubXDo.

Wang, H., Li, J., Zhang, H., and Zhou, Y. (2014). Benchmarking replication and consistency strategies in cloud serving databases: Hbase and cassandra. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 71–82. Springer.

Zetta, I. (2016). State of Disaster Recovery 2016. [Online]. https://bit.ly/2H6TwhN.