UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moraes Nunes da Silva

Avaliação da sobrecarga de replicação de bancos de dados relacionais para alta disponibilidade

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moraes Nunes da Silva

Avaliação da sobrecarga de replicação de bancos de dados relacionais para alta disponibilidade

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Humberto Luiz Razente

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2021

Matheus Moraes Nunes da Silva

Avaliação da sobrecarga de replicação de bancos de dados relacionais para alta disponibilidade

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 16 de junho de 2021:

Prof. Humberto Luiz Razente
Orientador

Prof. Marcelo Zanchetta do Nascimento

Prof. André Ricardo Backes

Uberlândia, Brasil
2021

Dedico esta conquista à minha Santa Mãe, Virgem Maria.

Agradecimentos

Agradeço primeiramente a Deus em seu diviníssimo sacramento, a Eucaristia, por ter me dado a graça de terminar a confecção deste trabalho na data de *Corpus Christi*. Também faço um agradecimento mais do que especial à Santíssima Virgem Maria que intercedeu por mim durante todo o tempo de escrita e testes deste trabalho, é claro que sem Ela esta monografia não seria finalizada.

Agradeço aos meus professores da UFU e, em especial, ao meu orientador, Humberto Razente, por ter recebido esta ideia de bom grado, pela paciência e dedicação em me orientar durante esse tempo.

Aos meus pais por me apoiarem em todo o período de estudos da Universidade.

Ao MUR (Ministério Universidades Renovadas) por me acolher logo no primeiro dia de aulas e me sustentar espiritualmente durante todos os anos em que fiquei na Universidade.

Aos meus amigos que me deram o suporte necessário desde o início deste trabalho.

Resumo

O intuito deste trabalho foi avaliar a sobrecarga de replicação de bancos de dados relacionais, utilizando duas arquiteturas transacionais distintas em comparação. Uma arquitetura contendo apenas um banco de dados centralizado e a outra contendo dois bancos de dados em replicação mútua forte. Esta avaliação de sobrecarga foi baseada nas métricas principais de Hardware como a utilização de CPU, memória RAM, Entrada e Saída de Disco e Tráfego de rede, além das próprias métricas do SGBDR implantado que, tendo em vista dois cenários de testes, culminaram em um melhor desempenho para o banco de dados centralizado. O primeiro cenário de testes simulou uma execução de transações de curta duração enquanto o segundo cenário de testes simulou uma execução de longa duração utilizando, em ambos os cenários, um *benchmark* de carga transacional para obter as métricas em avaliação. Para o desenvolvimento de tal atividade, apenas ferramentas de código aberto foram utilizadas a fim de facilitar a reprodução deste trabalho em futuras pesquisas.

Palavras-chave: Bancos de Dados, Bancos de Dados Distribuídos, Replicação de Dados, Alta Disponibilidade, Arquitetura.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Os bancos de dados estão presentes na vida do homem moderno sem que ele o perceba. Interações com sistemas que utilizam bancos de dados tornou-se comum e essencial nas ações diárias e ordinárias de qualquer pessoa. Como exemplo, as tarefas rotineiras em um banco financeiro, interações em redes sociais, compras com cartões de crédito e débito, a busca de livros em catálogos virtuais e a reserva de hotéis via Internet são tarefas presentes na vida cotidiana e que trazem o uso de um banco de dados (ELMASRI; NAVATHE, 2011b).

Segundo (ELMASRI; NAVATHE, 2011b) um banco de dados é uma coleção de dados relacionados. Os dados, por sua vez, são fatos da realidade passíveis de serem armazenados e que trazem um significado implícito. Em um exemplo básico, qualquer caderno de anotações que contenha informações financeiras de compras, como os nomes e valores de produtos, em um supermercado é considerado um banco de dados, pois as informações contidas neste caderno possuem significado, tem relação entre si e estão armazenadas de forma escrita.

Contudo, a utilização mais comum do termo banco de dados é um pouco mais específica que a definição anterior. Seu conceito geral possui uma maior aplicabilidade quando utilizado no âmbito da computação, tratando de armazenamentos de dados lógicos.

Outro conceito dentro da computação intimamente relacionado aos bancos de dados são os chamados Sistemas Gerenciadores de Bancos de Dados (SGBD). Em linguagem puramente técnica, um SGBD é definido como um sistema cujo objetivo global é registrar e manter informação (DATE, 1989) utilizando, para isso, um computador. As utilizações dos termos banco de dados sistema gerenciador de banco de dados podem ser utilizadas intercambiavelmente sem falha no entendimento contextual onde aplicados.

Com o passar dos anos os bancos de dados foram ganhando algumas formas diferentes para tratar e armazenar os dados. A forma mais comumente utilizada e conhecida é do tipo relacional (CODD, 1970), na qual é empregado um Sistema Gerenciador de Banco de Dados Relacional (SGBDR), que trata suas informações em um modelo lógico de tuplas, atributos e suas relações. Outra forma também difundida é do tipo não apenas relacional (NoSQL) que trata suas informações em forma de documentos, grafos, tuplas, ou pares chave/valor.

Estas duas formas podem ser aplicadas em diferentes infraestruturas e para diferentes tipos de cargas de trabalho. Utilizando o exemplo anterior do caderno de compras, quando aplicado a um software que faz uma gestão semelhante, pode-se utilizar a carga de trabalho transacional (ou Processamento de Transações em Tempo Real do inglês *On-Line*

Transactional Processing - OLTP) que funciona de modo a registrar e manter as pequenas operações realizadas no software. Caso seja necessário ter uma visualização agregada de todas as compras já realizadas por indivíduos em determinados períodos de tempo de modo a gerar *insights* e gráficos, pode-se utilizar a carga de trabalho analítica (ou Processamento Analítico de Transações do inglês *On-Line Analytical Processing - OLAP*). Ambas as cargas de trabalho utilizadas em um banco de dados possuem suas aplicações específicas e já estão enraizadas tanto no mercado quanto na academia, e podem ser executadas tanto na forma relacional quanto NoSQL.

Nos dias atuais, os sistemas de computação processam grandes quantidades de dados em um pequeno espaço de tempo, criando uma demanda por sistemas de bancos de dados robustos e capazes de lidar com a quantidade crescente de informação. Dentro deste avanço tecnológico emergiu-se a Computação Distribuída (CD) como meio de distribuir o processamento em computadores interligados por uma rede de computadores.

A Internet trouxe uma facilidade em criar e manter a CD. Como exemplo, as grandes empresas que fornecem infraestrutura possuem seus *Data Centers* geograficamente distantes e interconectados. A CD, bem como a Computação Paralela (CP), tornaram-se o principal foco para o desenvolvimento e utilização de aplicações e estas, por sua vez, demandaram um esforço na criação de bancos de dados que funcionam também de modo distribuído.

Um Banco de Dados Distribuído (BDD) é a coleção de múltiplos bancos de dados interconectados de modo lógico e localizados em diferentes nós de um sistema distribuído. Um Sistema Gerenciador de Banco de Dados Distribuído (SGBDD) é definido como um software que permite a manutenção de um BDD e faz sua distribuição de modo transparente a seus usuários (OZSU; VALDURIEZ, 2020c).

O presente trabalho também tem por objetivo apresentar os conceitos fundamentais dos BDDs, discorrer sobre Bancos de Dados Relacionais (BDR) e replicação multi-mestre (do inglês *multi-master*) e, ao final, apresentar uma avaliação prática de uma replicação multi-mestre e um servidor *stand-alone* utilizando SGBDR. O termo *stand-alone* diz respeito a um banco de dados centralizado em apenas uma instância. Ao longo desta monografia o leitor irá se deparar com os termos *stand-alone*, *single-node* e centralizado. Todos eles representam a mesma arquitetura.

O trabalho desenvolvido utilizou o SGBDR MariaDB *stand-alone* em um primeiro momento e em seguida utilizou uma replicação mútua forte (ou síncrona) com *Galera Cluster* em dois servidores SGBDR MariaDB. Todas as instalações foram realizadas no Sistema Operacional Debian 10 Buster de modo dedicado. Para o monitoramento das transações e carga de trabalho optou-se pela ferramenta *Percona Monitoring and Management (PMM).*

Este estudo está estruturado com os seguintes capítulos:

- O capítulo 2 apresenta uma revisão bibliográfica e trabalhos correlatos a fim de organizar o entendimento deste trabalho;
- O capítulo 3 mostra em detalhes o desenvolvimento prático das arquiteturas em avaliação, mostra também a metodologia, os resultados, as métricas e, por fim, uma breve discussão sobre os resultados obtidos;
- O capítulo 4 reúne as considerações finais, as disciplinas utilizadas no desenvolvimento e o que se espera de trabalhos futuros, culminando, assim, em uma conclusão.

2 Revisão Bibliográfica

Este capítulo aborda os conceitos fundamentais utilizados nesta monografia. Serão apresentados conceitos básicos sobre BDD, Propriedades ACID em Bancos de Dados Relacionais, Replicação de Dados, Consistência Eventual e, ao final, os trabalhos correlatos que influenciaram a escrita desta monografia.

2.1 Bancos de Dados Distribuídos

Para definir um BDD duas caracterśticas são necessárias: (I) seus dados devem estar necessariamente correlacionados entre si e (II) devem residir em um sistema distribuído (OZSU; VALDURIEZ, 2020c). O Sistema Distribuído no qual o banco de dados reside não precisa ser necessariamente um sistema homogêneo, e sim apenas estar interconectado por meio de uma rede de computadores.

Para um BDD possuir a sua forma, o armazenamento físico de seus dados não é tão relevante quanto o armazenamento lógico, visto que a aplicação que fará seu uso deverá abstrair o conceito de distribuição, permitindo ao usuário que veja as informações da mesma forma que se elas estivessem em apenas um SGBD não distribuído.

Segundo (OZSU; VALDURIEZ, 2020c), são quatro os tipos arquiteturais fundamentais para um SGBDD. São "fundamentais" pois não são os únicos, mas exercem o papel de fundamento para a maioria das outras arquiteturas existentes, muitas das quais são utilizadas comercialmente por grandes empresas como a Oracle (Oracle Database), Microsoft (SQL Server) e IBM (DB2).

Os tipos de arquiteturas fundamentais são: cliente/servidor, *peer-to-peer* (P2P), *multidatabase* e nuvem. Arquiteturas são criadas e evoluídas com o passar do tempo. A seguir será apresentada uma breve origem dos BDDs em conjunto com os tipos arquiteturais já expostos.

2.1.1 Origem dos BDDs e suas Arquiteturas Fundamentais

Sua história inicia-se na década de 1980 com a necessidade de se resolver o problema da distribuição de dados. Um dos BDDs dessa época foi o Sistema para Bancos de Dados Distribuídos (do inglês *System for Distributed Databases* ou SDD-1) desenvolvido pela *Computer Corporation of America*. O SDD-1 permitia um banco de dados relacional ser distribuído entre vários nós de uma rede de computadores e acessado como se estivesse em apenas um nó. A interação realizada entre os usuários e o SDD-1 era por meio de uma linguagem procedural de alto nível denomidada *Datalaguage* (BERNSTEIN et al., 1981).

O principal objetivo do SDD-1 era processar consultas com a mínima quantidade de transferência de dados entre os nós componentes do sistema (BERNSTEIN et al., 1981). Esse objetivo ainda é buscado pelas desenvolvedoras dos SGBDDs presentes no mercado, visto que o atraso (*delay*) na transferência de dados em uma rede de computadores ainda é um fator que pode prejudicar aplicações que demandam por velocidade de resposta.

Originalmente, os BDDs emergiram utilizando o conceito arquitetural *peer-to-peer* (P2P), como no próprio sistema SDD-1 e o *Distributed INGRES*. Porém, após o advento dos computadores pessoais e a larga utilização da Internet, passou-se a utilizar um outro conceito arquitetural, o cliente/servidor (OZSU; VALDURIEZ, 2020c).

De forma ampla e comum, o conceito de conexões P2P remete à rede de computadores. Segundo (ORAM, 2001) as redes P2P funcionam com o propósito de seus usuários contribuírem com seus próprios arquivos, tempos de processamento computacional ou outros recursos utilizados em algum sistema compartilhado. O conteúdo da rede e escolha é sempre controlado pelos próprios usuários.

Quando os primeiros BDDs foram desenvolvidos utilizando do conceito arquitetural P2P, não se referiam à arquitetura da rede de computadores utilizada e sim ao modelo de manutenção de dados onde cada nó do BDD possuía funcionalidades similares, mas não completas, em se tratando desta manutenção. Dessa forma, não havia distinção prática entre clientes e servidores (OZSU; VALDURIEZ, 2020c).

O conceito P2P aplicado aos BDDs foi reformulado para se adaptar à modernidade dos sitemas atuais. Segundo (ELMASRI; NAVATHE, 2011a) os nós do BDD devem ser integrados individualmente a um número pequeno de pares e é permitido que um único nó não ofereça todas as funcionalidades de um SGBD completo. Cada nó pode possuir apenas um conjunto de informações modeladas ao seu *bel-prazer* desde que forneça um conjunto semântico aos outros nós. Dessa forma, uma consulta quando aplicada nesta arquitetura é descentralizada, isto é, ela passa de nó em nó até ser satisfeita completamente.

A comunicação cliente/servidor utilizada amplamente nos dias de hoje foi desenvolvida para ligar, por meio de uma rede de computadores, uma grande quantidade de dispositivos diferentes, como impressoras, servidores Web, PCs particulares, switches, servidores de email, entre outros. A ideia central dessa arquitetura é possuir servidores especializados em algum processamento recebendo e respondendo as requisições de algum cliente. Como exemplo pode-se pensar em um servidor de emails SMTP que recebe as requisições de leitura de algum cliente de email (ELMASRI; NAVATHE, 2011c).

Aplicando o conceito cliente/servidor no mundo dos bancos de dados, obtem-se um Sistema Gerenciador de Banco de Dados Relacional centralizado que realiza todo o processamento em cima dos dados e os organiza a fim de devolver uma resposta ao cliente que fez a requisição. O papel do cliente, por sua vez, é realizado pelo software que faz a

conexão com servidor SGBDR e que envia consultas na linguagem SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada) e mostra os resultados em uma tela (ELMASRI; NAVATHE, 2011c).

A terceira arquitetura a ser apresentada é a *multidatabase* que foca na autonomia dos bancos de dados componentes, ou não, de um Sistema Distribuído. Essa autonomia é completa, significando que não existe cooperação direta entre os nós. Porém, para manter o conceito de distribuição é necessário que de alguma forma os bancos de dados autônomos estejam interligados.

Cada *multidatabase* é um SGBD completo que opera sozinho. Dessa forma, para mediar e transformar essa arquitetura em uma arquitetura distribuída o *multidatabase* fornece uma camada de software adicional que faz a interligação com os outros nós do BDD de forma que o usuário acesse cada SGBD separadamente e faça suas correlações semânticas das informações armazenadas neste BDD.

Por fim a quarta arquitetura é chamada de nuvem (tradução vinda do inglês *cloud*, mais comumente utilizada). Essa arquitetura diz respeito à infraestrutura de hardware e software disponibilizada aos bancos de dados ou qualquer outro tipo de aplicação.

Nos dias atuais, os provedores de nuvem (do inglês *cloud providers*) já ganharam um grande espaço no mercado computacional com seu fornecimento de infraestruturas e serviços para qualquer tipo de necessidade. A facilidade em se manusear uma infraestrutura em nuvem de grandes empresas como Amazon (AWS), Microsoft (Azure), Google Cloud Platform (GCP), e Oracle (Oracle Cloud) já é uma realidade que permite a migração de grandes sistemas *on-premises* para nuvens com um mínimo de esforço técnico.

Essas grandes nuvens possuem seus *datacenters* distribuídos geograficamente ao redor do globo e interconectados, seja por meio da Internet ou por redes dedicadas, e de rápida velocidade. Essa característica facilitou o provisionamento de BDDs geograficamente entre as regiões de cada *provider*, podendo até ser em diferentes continentes.

Uma das ideias de se montar um BDD em nuvem é o compartilhamento de recursos, tanto de hardware quanto de software. Na maioria dos sistemas provisionados em nuvem existe o compartilhamento de hardware entre Máquinas Virtuais (VM) para economizar os custos de manteneção da infraestrutura. Este conceito de compartilhamento também é aplicado aos SGBDs, tornando possível a convivência entre vários SGBDs diferentes no mesmo hardware ou até mesmo compartilhando o mesmo SGBD entre diferentes aplicações e empresas.

2.1.2 Dimensões Arquiteturais para um BDD

Segundo (OZSU; VALDURIEZ, 2020c; ELMASRI; NAVATHE, 2011a) existem três dimensões nos modelos arquiteturais de um BDD. Cada uma dessas dimensões contém

níveis (ou graus) que fazem um BDD ser mais ou menos pendente àquela dimensão. A primeira dimensão diz respeito a sua autonomia, a segunda ao seu grau de heterogeneidade e a terceira à sua própria distribuição. A seguir são explicadas cada uma dessas dimensões.

Autonomia: Um BDD possui um maior grau de autonomia quando qualquer um de seus nós consegue operar de forma mais independente em relação aos demais, isto é, quando consegue aplicar transações em si mesmo e não se preocupar tanto com a replicação dessas transações para os outros nós.

Heterogeneidade: O conceito de heterogeneidade é amplo em se tratando de Sistemas Computacionais. Um Sistema Computacional Homogêneo diferencia-se de de um Sistema Computacional Heterogêneo na medida em que este último faz o uso de diferentes arquiteturas de hardware e software visando o melhor desempenho de suas aplicações (STRINGHINI; GONÇALVES; GOLDMAN, 2012).

Em se tratando de um BDD, o grau de heterogeneidade diz respeito ao próprio software SGBDD. Portanto, partindo deste ponto de vista, o mais importante é o próprio modelo de dados, as linguagens de consulta e os protocolos de manutenção de transações.

De forma mais clara, os BDDs homogêneos utilizam o mesmo SGBD em cada nó da rede, mesmo que a arquitetura de hardware ou sistema operacional sejam diferentes dos outros nós. Os heterogêneos utilizam ao menos dois SGBDs diferentes também independente da arquitetura dos nós em que estão instalados (MANICA, 2001).

Distribuição: O grau de distribuição está relacionado aos próprios dados gerenciados pelo SGBD de forma física. Quanto mais estes dados estiverem espalhados pelos nós do sistema, maior é o grau de distribuilção do BDD. Contudo, apesar do grau de distribuição física, o cliente sempre enxergará o BDD de forma lógica como se o banco de dados estivesse centralizado em apenas um servidor.

2.1.3 Vantagens dos Bancos de Dados Distribuídos

Conforme exposto por (OZSU; VALDURIEZ, 2020c) quatro são as vantagens fundamentais dos BDDs: (I) Manutenção tansparente de dados distribuídos e replicados, (II) Confiabilidade por meio de transações distribuídas, (III) Desempenho Melhorado e (IV) Escalabilidade. Todas elas serão apresentadas a seguir.

Manutenção transparente de dados distribuídos e replicados: Um sistema dito transparente esconde os detalhes de implementação de seus usuários. Isso se faz extremamente importante em um sistema distribuído, visto que a implementação é mais complexa devido às interconexões de rede entre os nós.

Para exemplificar este item tomar-se-á um sistema de gestão de uma rede de supermercados que possui três filiais localizadas nas capitais: São Paulo, Belo Horizonte e

Goiânia. Cada capital possui sua própria organização de produtos e estoques, porém, seus gestores gostariam de manter os dados de precificação centralizados e de rápido acesso para os sistemas locais. Para isso pode-se utilizar um banco de dados distribuído que possui três nós, cada qual em uma das capitais. Os dados de precificação e estoque podem ser replicados entre os três nós e o acesso de cada sistema local será em sua devida capital.

Quando a filial de São Paulo for realizar uma inserção de estoque ou de precificação, ela fará em seu sistema local que possui seu próprio banco de dados. Como este banco de dados faz parte de um BDD, os outros sistemas das outras capitais enxergarão o mesmo dado inserido pela filial de São Paulo devido à replicação.

A manutenção dos dados deste BDD é transparente para as aplicações das três filiais, isto é, nenhuma delas percebe que está utilizando um banco de dados distribuído e ainda assim o está. Os dados estão sendo replicados entre os três SGBDs e o acesso de cada sistema em qualquer um dos nós acontece localmente, aumentando o tempo de resposta às consultas realizadas pelas aplicações.

Confiabilidade por meio de transações distribuídas: BDDs são confiáveis pois utilizam replicação de dados para mantê-los redundantes entre os diversos nós do sistema. Isso significa que caso um nó entre em estado de falha, os outros nós possuem o mesmo dado e podem ser utilizados para manter a aplicação em pleno funcionamento sem que esta perceba que seu banco de dados está problemático.

Desempenho Melhorado: Dois pontos são utilizados para explicar este ítem. O primeiro diz respeito à possível fragmentação de dados em um BDD, isto é, cada nó de um BDD consegue manter apenas os dados que serão utilizados naquela localidade. O segundo ponto diz respeito à quebra de uma consulta em diversas outras consultas que podem ser aplicadas nos outros nós de um BDD, paralelizando sua execução e retornando o resultado mais rapidamente.

Escalabilidade: A escalabilidade de um BDD é de fácil aquisição. Como o banco de dados já está dividido entre os nós, basta apenas adicionar processamento e armazenamento para rede. Diferentemente de um banco de dados centralizado onde todo o processamento está localizado em um único nó, engessando sua escalabilidade.

2.2 Transações e Propriedades ACID

Segundo (ELMASRI; NAVATHE, 2011d) a transação é uma unidade lógica de processamento realizada pelo banco de dados. Seu intuito é levar o banco de dados de um estado consistente para outro estado consistente. Este conceito engloba as operações básicas de um banco de dados como **inserções**, **exclusões**, **atualizações** e **consultas**.

Uma transação pode conter uma ou mais destas operações e trabalha com o con-

ceito de "tudo ou nada". Ela deve ser concluída totalmente ou não ser executada de maneira alguma. Como exemplo pode-se pensar em algumas atividades realizadas por clientes dentro de um e-commerce de roupas, onde estes clientes passam pelo processo comum de compras em um website. Para uma compra online ser consistente ela precisa ser realizada de forma clara e objetiva ou não ser executada, de forma que o produto dê baixa ao estoque de roupas ou nunca saia de lá. De maneira igual, o caixa da loja precisa passar de um estado consistente para outro estado consistente: o pagamento precisa ser realizado ou não ser executado, mesmo que haja devolução da roupa e do dinheiro (essa devolução configuraria uma nova transação).

Algumas propriedades são elencadas por (ELMASRI; NAVATHE, 2011d) a fim de que os SGBDs implementem transações seguras por seus próprios métodos de tratar concorrência e recuperação das informações. Essas propriedades são conhecidas como **ACID**: Atomicidade, Consistência, Isolamento e Durabilidade.

Também é notável que, em uma arquitetura de BDD, garantir essas propriedades não é uma tarefa trivial como exposto no artigo (CAMPÊLO et al., 2020). Nas próximas subseções serão mostradas cada uma dessas propriedades e como podem ser garantidadas de forma prática em uma arquitetura de banco de dados distribuída.

2.2.1 Atomicidade

A atomicidade faz com que uma transação precise ser executada de forma completa ou não ter sido realizada de forma alguma. O próprio banco de dados deve garantir que caso haja algum problema de percurso na execução da transação, ela seja desfeita em todas as suas instâncias.

Para garantir a atomicidade em bancos de dados replicados é possível utilizar o protocolo de confirmação em duas fases (do inglês *Two Phase Commit* ou 2PC) que será visto a seguir.

2.2.1.1 Protocolo de confirmação em duas fases

Segundo (ELMASRI; NAVATHE, 2011e) o 2PC trabalha como um **coordenador** de transações entre os nós componentes do banco de dados replicado. Caso uma atualização seja aplicada a um dado em algum dos nós, o coordenador do ambiente 2PC entra para garantir que essa mesma transação seja consolidada em todos os nós. Caso haja uma falha por qualquer motivo o coordenador trata para que a transação não seja executa em nenhum dos nós e sempre é possível recuperar o banco de dados para um estado em que ou a transação é consolidada ou é revertida.

Este protocolo tem fundamental importância em uma arquitetura replicada, pois, segundo (OZSU; VALDURIEZ, 2020b), quando a sincronização é controlada entre os nós

participantes de um ambiente replicado é possível manter suas cópias idênticas, garantindo a atomicidade. Por exemplo, caso uma aplicação esteja lendo um determinado dado em um dos nós participantes de um ambiente replicado e uma transação que altera o mesmo dado é submetida a um outro nó pouco antes da leitura acontecer, se não houvesse uma garantia atômica entregue pelo 2PC, o dado lido poderia estar diferente entre os nós e não haveria garantias para o correto funcionamento da aplicação. Já com a garantia do 2PC, o dado será o mesmo em todos os nós. A figura 1 apresenta mais detalhadamente o funcionamento deste protocolo.

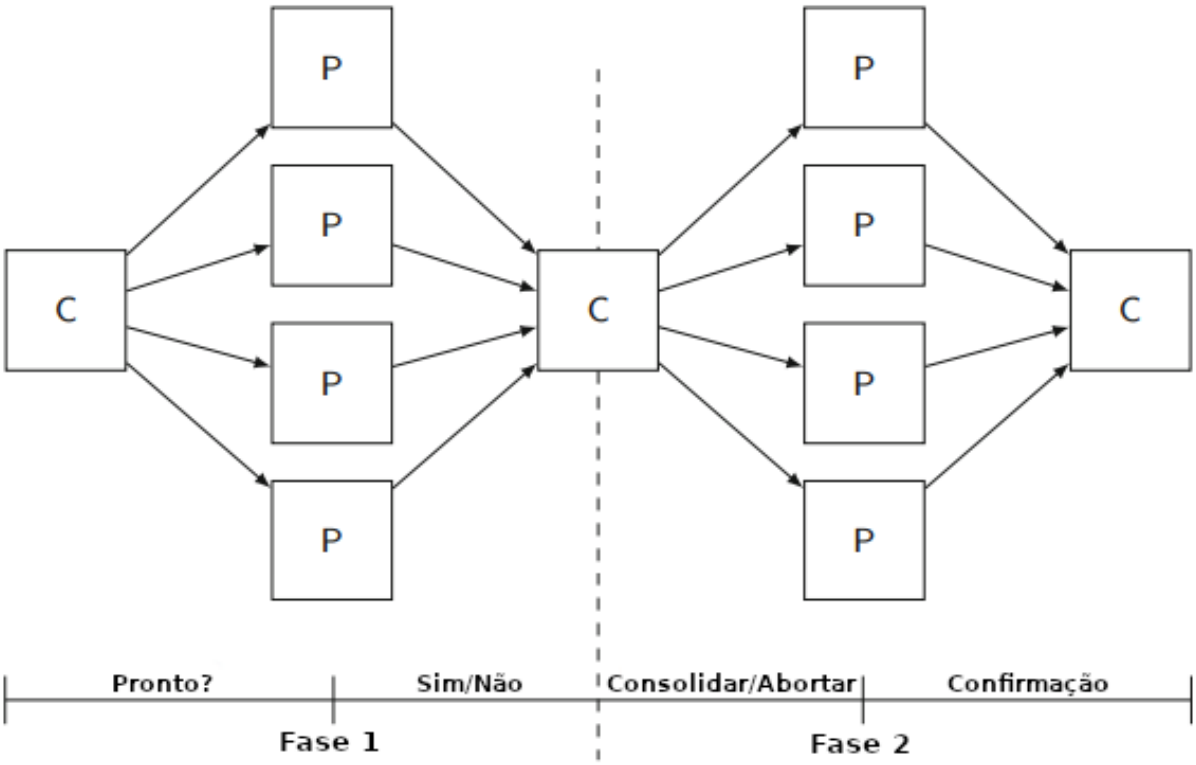

Figura 1 – Estrutura de comunicação centralizada do protocolo 2PC. **C: Coordenador, P: Participante**. Extraído de (OZSU; VALDURIEZ, 2020b)

De acordo com (OZSU; VALDURIEZ, 2020b), a primeira fase do processo organizado pelo 2PC consiste em o coordenador "C" questionar os participantes "P" se estão aptos a consolidarem os dados recebidos na transação e aguardar a resposta de cada um deles. Caso a resposta de cada um dos participantes for positiva sobre a transação, o processo entra na segunda fase onde o coordenador informa aos participantes que consolidem os dados da transação e estes, após a consolidação, respondem ao coordenador que a transação foi executada com sucesso, finalizando assim o processo. Caso a resposta de pelo menos um dos participantes for negativa na primeira fase, o coordenador envia uma mensagem para que os participantes abortem o processo de consolidação. Neste último caso nenhum dado é persistido em nenhum dos nós e o processo é finalizado.

O exemplo anterior faz parte de um modelo de implementação chamado 2PC cetralizado que é o mais simples e fácil de ser estruturado. Existem outras implementações como o 2PC linear que consiste em todos os participantes se comunicando entre si após a primeira mensagem do coordenador, fazendo um intercâmbio entre as fases 1 e 2 até que a transação seja copletamente consolidada ou então seja abortada durante o processo. Outro modelo possível é o chamado 2PC distribuído que faz uso apenas da fase 1, onde cada participante, após receber a primeira mensagem do coordenador, envia um sinal para todos os outros participantes informando sobre a possibilidade de consolidar a transação. Cada participante irá aguardar a mensagem de todos os outros a fim de tomar uma decisão sobre a consolidação da transação, finalizando o processo.

2.2.2 Consistência

A transação precisa levar o banco de dados como um todo de um estado consistente para outro estado consistente, isto é, se for totalmente executada ela precisa ser consolidada na base de dados gerenciada pelo SGBD. Para obter a consistência, algumas práticas são comumente adotadas tanto em bancos de dados centralizados quanto em BDDs: restrições de integridade, operações em cascata, gatilhos e procedimentos armazenados.

De acordo com (MCMINN; WRIGHT; KAPFHAMMER, 2015), as restrições de integridade garantem que os dados manipulados por um SGBD sigam determinados padrões e estejam coerentes com a aplicação. Neste viés, o SGBD tem o dever de rejeitar instruções SQL que não satisfaçam as restrições atribuídas ao banco de dados. Abaixo está uma lista com as restrições de integridade comumente utilizadas pelos SGBDRs e, em seguida um exemplo envolvendo o uso de todas essas restrições.

- Chave primária: utilizada para garantir que os registros de uma determinada tabela no banco de dados sejam únicos. Por definição, a chave primária é o conjunto das restrições *unique* e *not null*.
- Chave estrangeira: informa quais dados em determinadas tabelas são relacionados com outros dados em outras tabelas. Esta restrição garante o relacionamento entre registros espalhados em diversas tabelas no banco de dados.
- *Unique*: garante a unicidade de valores em um determinado conjunto de colunas.
- *Not null*: identifica quais colunas permitem armazenar valores “não conhecidos” ou “não existem”.
- *Check*: restrições de checagem garantem que os dados atribuídos a um registro passem por um predicado de verificação antes de serem armazenados.

Exemplificando, um supermercado precisa manter seus produtos organizados em seu sistema de gerência a fim de ter um maior controle sobre seu estoque. Para isso duas tabelas foram utilizadas: PRODUTO e TIPO_DE_PRODUTO. As Tabelas 1 e 2 apresentam suas colunas, os tipos de dados de cada coluna e suas respectivas restrições de integridade. As tabelas 3 e 4 apresentam alguns dados já adicionados ao banco.

Cada registro nestas tabelas possui um valor único no campo ID que dermina a unicidade dos registros (**chave primária**), essas duas tabelas estão relacionadas pelos campos PRODUTO.TIPO e TIPO_DE_PRODUTO.ID, de forma que cada produto adicionado deve conter uma categoria geral (**chave estrangeira**). A categoria de cada produto deve ser única em todo o banco de dados, ou seja, não pode haver mais de uma categoria na tabela TIPO_DE_PRODUTO (**unique**). Um produto deve conter obrigatoriamente uma descrição a fim de que seja identificado facilmente (**not null**) e seu preço deve satisfazer uma checagem que deve ser maior que 1 real e menor ou igual a 1000 reais (**check**) antes de que o produto seja cadastrado no banco de dados.

De forma análoga, também é possível garantir a integridade dos dados por outros caminhos que não sejam apenas essas restrições. Como exemplo é possível adicionar gatilhos e procedimentos armazenados que realizam algum cálculo sobre os dados logo antes de serem inseridos no banco de dados. No mesmo exemplo do supermercado aqui construído é possível ter um gatilho que é disparado para decrementar o valor do campo ESTOQUE caso um produto seja comprado por um cliente.

Além disso as operações em cascata também são utilizadas para garantir a consistência. Se um tipo de produto for excluído da tabela TIPO_DE_PRODUTO, os registros relacionados com esse tipo também devem ser excluídos da tabela PRODUTO. Neste supermercado, se o tipo “Produtos alimentícios” for excluído, os registros de “Arroz” e “Feijão” também serão apagados.

É importante saber que a terefa de manter um BDD consistente é mais complexa do que manter um banco centralizado consistente, devido a necessidade de se manter as cópias de dados idênticas nos diferentes nós interconectados pela rede.

CAMPO	**TIPO DE DADOS**	**RESTRIÇÃO**
ID	NUMÉRICO	CHAVE PRIMÁRIA
DESCRIÇÃO	CARACTERE	NOT NULL
TIPO	CARACTERE	CHAVE ESTRANGEIRA
ESTOQUE	NUMÉRICO	NOT NULL
PREÇO	DECIMAL	CHECK (> 1 and <= 1000)

Tabela 1 – Descrição da tabela PRODUTO

CAMPO	TIPO DE DADOS	RESTRIÇÃO
ID	NUMÉRICO	CHAVE PRIMÁRIA
CATEGORIA	CARACTERE	UNIQUE
DATA_DE_ADIÇÃO	DATE	NULL

Tabela 2 – Descrição da tabela TIPO_DE_PRODUTO

ID	DESCRIÇÃO	TIPO	ESTOQUE	PREÇO_UNITÁRIO
1	Arroz	1	257	5,49
2	Feijão	1	200	8,00
3	Tesoura	3	56	15,30
4	Vasilha de plático	3	33	4,99
5	Cobertor	2	30	59,90

Tabela 3 – Dados da tabela PRODUTO

ID	CATEGORIA	DATA_DE_ADIÇÃO
1	Produtos alimentícios	01/07/2020
2	Cama, mesa e banho	NULL
3	Produtos gerais	02/08/2020

Tabela 4 – Dados da tabela TIPO_DE_PRODUTO

2.2.3 Isolamento

O isolamento diz respeito a duas ou mais transações sendo executadas no mesmo instante de tempo. Para que a transação tenha essa propriedade, ela necessita de ser executada sem a interferência de outras transações também executadas no mesmo instante de tempo.

Para tal propriedade é possível implementar o conceito *timestamp* global onde cada transação pode obter um valor único controlado pelo banco de dados de acordo com

ordem de chegada de cada transação. Esses valores vão crescendo com o tempo e uma determinada transação só será executada a partir do momento em que todas as outras transaçãos com valores menores já estiverem sido executadas pelo SGBD.

2.2.4 Durabilidade

Se consolidadas, as transações precisam ser duráveis dentro do banco de bados, mesmo que ocorra algum problema ou falha qualquer que seja. A durabilidade é garantida pelo subsistema de recuperação do banco de dados, onde este não pode deixar que uma atualização mais antiga sobreponha uma atualização mais recente.

Essa implementação pode ser obtida utilzando o conceito de filas em estruturas de dados. Por exemplo a estrutura FIFO (*First-In, First-Out*) que tem por objetivo organizar as filas onde o primeiro a entrar é o primeiro a sair. Isso significa que em caso de uma falha de sistema as transações aplicadas ao banco de dados seguirão uma fila onde a primeira transação a ser enviada ao banco deverá ser a primeira a ser executada e persistida em disco não volátil (WEI; PIERRE; CHI, 2009).

2.3 Replicação de Dados

O conceito de Replicação de Dados é bem intuitivo. De maneira geral diz respeito à redundância dos dados contidos em um banco de dados para um ou mais nós do BDD. Segundo (OZSU; VALDURIEZ, 2020a) muitos são os propósitos da replicação: (I) Disponibilidade do Sistema, (II) Desempenho, (III) Escalabilidade e (IV) Requisitos de Aplicação.

Disponibilidade do Sistema: A disponibilidade do sistema passa pelo conceito de replicação total, parcial ou nula. Segundo (ELMASRI; NAVATHE, 2011a) a replicação total de um BDD é um extremo que ocorre quando cada nó do sistema possui uma cópia completa do banco de dados. A replicação nula (também conhecida como alocação não redundante) seria o outro extremo e acontece quando os fragmentos do banco de dados estão distribuídos de forma que cada fragmento esteja em apenas um nó, não havendo replicação entre eles (são disjuntos). Por fim, a replicação parcial está no meio dos dois extremos e consiste em realizar a replicação de apenas alguns fragmentos do banco de dados e não do banco de dados completo.

Dessa forma, quanto mais a replicação tender à replicação total, mais disponível será o banco de dados para suas aplicações. Em um banco de dados totalmente replicado, caso um dos nós entre em estado de falha haverão outros nós que poderão assumir as requisições deixando seus dados disponíveis boa parte do tempo.

Contudo, a replicação total possui suas desvantagens. Ela pode adicionar um atraso

nas atualizações dos dados, pois necessita aplicar suas operações em todos os nós a fim de que todos estejam consistentes.

Desempenho: Resultados de consultas aplicadas a banco de dados replicados podem ser obtidas em um nó que esteja mais próximo do requisitante, reduzindo seu tempo de resposta e aumentando o desempenho do ponto de vista do cliente.

Escalabilidade: Bancos de dados replicados aceitam com facilidade o crescimento de uma aplicação pois é possível adicionar nós replicados ao sistema completo levando a uma quantidade maior de respostas possíveis às requisições dos clientes.

Requisitos de Aplicação: Aplicações que utilizam BDDs podem variar seus requisitos quanto à necessidade de uma replicação. Caso a aplicação não seja tão crítica e aceite uma indisponibilidade no BDD ela pode optar por não utilizar bancos de dados replicados e utilizar apenas um SGBD centralizado. Por outro lado, caso a aplicação seja crítica e exija que o sistema esteja disponível 99,99% do tempo, então é necessário criar replicações para tal tarefa. Em outros casos a aplicação apenas necessita que os dados sejam replicados geograficamente para ter uma segurança de não perdê-los em caso de corrompimento total (sem possibilidade de recuperação) de um dos nós.

2.3.1 Consistência em Bancos de Dados Replicados

Como já apresentado anteriormente, a garantia de consistência deve levar o banco de dados de um estado consistente para outro estado consistente após a execução de alguma transação. O estado diz respeito à todos os dados armazenados em um dado momento e a consistência diz respeito à satisfação das restrições de integridade impostas aos dados pelo SGBD.

Quando um banco de dados replicado sofre alterações em seus dados, elas devem ser replicadas para os outros nós interligados. Dessa forma é possível dizer que em um determinado ponto no tempo os dados estão diferentes entre os nós, isto é, antes da replicação acontecer pela rede. Segundo (OZSU; VALDURIEZ, 2020a) existem dois problemas relacionados à consistência em bancos de dados replicados: (I) consistência mútua e (II) consistência de trasação.

Consistência Mútua: Em bancos de dados replicados a consistência mútua dita que todas as réplicas devem conter cópias idênticas de seus dados. A consistência mútua pode ser definida em graus. A consistência mútua forte existe quando, ao se realizar uma atualização no banco de dados, os dados já são imediatamente replicados e atualizados nas réplicas. Geralmente utiliza-se o protocolo 2PC (já citado na seção 2.2.1.1).

Em contrapartida, observa-se uma consistência mútua fraca quando, ao se realizar uma atualização no banco de dados, não é exigido que atualização seja replicada imediatamente para os outros nós. Em outras palavras, não é necessário que todas as réplicas

possuam os dados idênticos quando uma atualização é finalizada. De qualquer forma, é necessário garantir que os dados sejam idênticos em algum ponto no tempo, mesmo que demore um pouco. Este efeito também é chamado de consistência eventual, na qual, após a conclusão de uma transação, a mesma entra em uma fila de replicação, que será processada sequencialmente.

Consistência de Transação: Anteriormente, nas propriedades ACID, foi exposto o conceito de consistência de Bancos de Dados quando um banco, após a execução total de uma transação, passa de um estado consistente para outro estado consistente. Em se tratando de consistência de transação a definição, segundo (OZSU; VALDURIEZ, 2020b), leva em consideração a concorrência entre múltiplas transações sendo executadas ao mesmo tempo. O banco de dados deve permanecer em um estado consistente mesmo que haja múltiplas operações sendo executas no mesmo momento sobre os mesmos dados.

Isto é um problema em um banco de dados replicado quando utilizado em conjunto com uma Consistência Mútua forte. Caso o banco deva manter todas as réplicas idênticas após a execução de cada operação, leva-se em consideração os custos de tempo relacionados ao atraso de tréfego da rede e o processamento do próprio banco de dados. O custo neste caso implica no tempo de resposta na execução das transações que ocorrem simultaneamente nos vários bancos de dados envolvidos na operação por meio da rede.

2.4 Trabalhos correlatos

O trabalho (MOIZ et al., 2010) teve o objetivo de comparar replicação em bancos de dados comerciais e replicação em bancos de dados de código aberto. Para isso primeiramente foram apresentados alguns conceitos fundamentais relacionados à replicação e BDDs, características de tolerância a falhas e, por fim, apresentaram as ferramentas utilizadas.

Código aberto: Postgres-R, Slony-I, ESCADA Replication Server, DB Replicator e Pgpool-II.

Comerciais: IBM Informix Replication, DB2 Data Propagator, Sybase Replication Server, Q- Replication Tools e Oracle's Stream

Já em (DHAMANE et al., 2014) a proposta é avaliar o desempenho de sistemas de replicação de alguns bancos de dados em casos onde os servidores estão em ótima execução e em casos onde há falhas pontuais. Os sistemas utilizados foram Middle-R, C-JDBC e MySQL Cluster. A arquitetura de cada sistema está apresentada na Figura 2.

O *benchmark* utilizado para avaliar estas arquiteturas foi o TPC-W, o qual é transacional e simula uma aplicação e-Commerce para uma loja de livros. Seus resultados foram mais satisfatórios no sistema MySQL Cluster que toma um lado mais comercial

em comparação com os sistemas Middle-R e C-JDBC que são mais utilizados na área acadêmica.

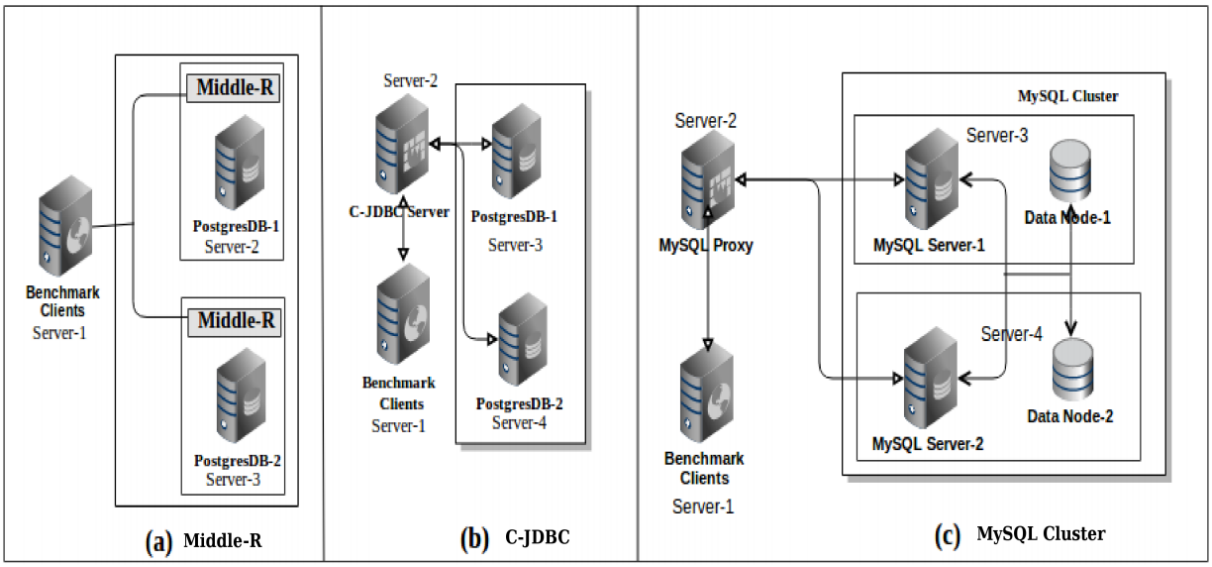

Figura 2 – Implantação de duas réplicas. (a) Middle-R, (b) C-JDBC, (c) MySQL Cluster. Extraído de (DHAMANE et al., 2014)

O principal objetivo do trabalho (CAMPÊLO et al., 2020) foi avaliar a consistência de bancos de dados replicados nos ambientes em nuvem. Esses serviços são denominados DBaaS (*Database as a Service*) onde a administração desses bancos de dados fica ao encargo da própria provedora da nuvem. Ao longo deste trabalho, foram apresentados alguns conceitos fundamentais como as características mais importantes em uma replicação de bancos de dados, modelos de consistência e métodos de consistências nas arquiteturas replicadas.

Para avaliar a consistência desses bancos de dados replicados, o trabalho primeiramente cita os requisitos de armazenamento impescindíveis para esta arquitetura: Autômatos, alta disponibilidade, elasticidade, tolerância a falhas, baixa latência, tolerância ao particionamento, performáticos, confiáveis e escaláveis.

Logo após, os autores relacionam os modelos de consistência com os métodos de consistências nas arquiteturas replicadas, descrevendo suas caracteristias e utilizam alguns sistemas específicos na avaliação prática dos requisitos de armazenamento (como descrito nesta mesma seção). Estes sistemas são: PNUTS, Spanner, Clock-SI, Vela, Indigo, GEO, SSOR, Harmony, VFC, DynamoDB e Pileus.

A monografia aqui apresentada tem correlação com o trabalho (CAMPÊLO et al., 2020) pois este apresenta uma infraestrutura em nuvem, utilizando BDDs e apresenta o desempenho como um dos requisitos de armazenamento para uma arquitetura replicada.

A avaliação realizada nesta monografia utiliza o desempenho como seu principal ponto de avaliação na arquitetura replicada de bancos de dados relacionais.

Contudo, o presente trabalho se assemelha mais ao trabalho de (DHAMANE et al., 2014) que faz comparação de desempenho entre arquiteturas replicadas diferentes. Porém os trabalhos (CAMPÊLO et al., 2020) e (MOIZ et al., 2010) possuem sua parcela de semelhança com esta monografia que, de forma análoga aos trabalhos apresentados nesta seção, têm como enfoque bancos de dados replicados, porém, a avaliação e os estudos mostrados são comparados com uma arquitetura centralizada.

3 Desenvolvimento

Neste capítulo, inicialmente serão apresentadas algumas ferramentas utilizadas para o desenvolvimento deste trabalho como o SGBDR MariaDB em conjunto com o *Galera Cluster*, o *benchmark* TPC-E e a infraestrutura utilizada neste procedimento, a AWS. Logo após será apresentada a integração entre essas tecnologias de forma prática a fim de se criar um cluster de bancos de dados relacionais e um banco de dados *stand-alone*.

3.1 Ferramentas e Tecnologias

Inúmeras são as ferramentas de bancos de dados disponíveis atualmente no mercado. Para o desenvolvimento deste trabalho foram escolhidas, em sua maioria, ferramentas de código aberto por motivos de facilidade no desenvolvimento, auxílio da comunidade e pela gama de documentações disponíveis na Internet.

3.1.1 MariaDB

O Sistema Gerenciador de Banco de Dados MariaDB é um *fork* do código do MySQL realizado em 2009 logo após a aquisição da Sun Microsystems, que até então era proprietária do MySQL, pela Oracle. O MariaDB é um software de código aberto disponível sob a licença GPL v2 (WIDENIUS, 2019; FOUNDATION, 2020).

Como no próprio MySQL, o MariaDB trabalha com o conceito de *Storage Engines* que são acopladas ao SGBD de forma a funcionar como o motor de gerenciamento de transações e armazenamento. Dependendo do tipo de carga de trababalho em uso, algumas *Storage Engines* são mais indicadas que outras.

Para a carga de trabalho transacional (ou OLTP), a *Storage Engine* mais comum e amplamente utilizada é a *InnoDB*. Outras também são indicadas como a *MyRocks* que foi desenvolvida para evitar o desgaste rápido de um SSD e a *Spider* que facilita a fragmentação de dados entre vários SGBDs interconectados por uma rede.

Para a carga de trabalho analítica (ou OLAP), o MariaDB disponibiliza sua *ColumnStore* que trata o armazenamento de forma inversa às cargas OLTP, onde todo o banco de dados está armazenado em formato colunar. Essa peculiaridade o torna rápido e indicado quando se precisa retornar uma grande quantidade de informações de uma só vez.

Existem também algumas *Storage Engines* que não são focadas nas cargas de trabalho OLTP ou OLAP, mas que podem servir para o mesmo propósito ou para qualquer

outra necessidade, desde que o desenvolvedor saiba o que está fazendo. Exemplos de outras *Storage Engines* são a *CSV* que armazena os dados em formato *.csv* (colunas separadas por caracteres específicos), *MyISAM* que oferece uma rápida leitura de dados, *Memory* a qual armazena as tabelas em memória para acesso rápido e não volátil, entre muitas outras.

Neste trabalho o SGBD utilizado foi o MariaDB versão 10.5.8 com a carga de trabalho OLTP e a *Storage Engine* InnoDB.

3.1.2 Galera Cluster

O Galera Cluster é uma ferramenta desenvolvida pela *Codership* para fornecer alternativas de replicação aos SGBDs MySQL, MariaDB e Percona XtraDB. De modo geral o Galera Cluster realiza uma replicação de consistência mútua forte ou, em outras palavras, uma replicação síncrona em que todas as réplicas do banco de dados estão idênticas assim que um dado é atualizado. O cliente enxerga todos os nós como se fossem apenas um (CODERSHIP, 2020).

A versão do Galera Cluster utilizada neste trabalho foi a 4 com o *patch* 26. Essa versão já vem inclusa na instalação do último pacote *mariadb-server* no sistema operacional Linux distribuição Debian 10 Buster.

3.1.3 Percona Monitoring and Management

O PMM é uma ferramenta de monitoramento desenvolvida pela Percona LLC que pode ser utilizada para os SGBDs MySQL e seus semelhantes (MariaDB e Percona), PostgreSQL, MongoDB e a ferramenta de proxy, o ProxySQL (PERCONA, 2020).

Para este trabalho, o PMM foi *conteinerizado* em Docker com a imagem disponibilizada pela própria Percona no Docker Hub. Todos os gráficos apresentados na seção de desenvolvimento foram retirados desta ferramenta. A versão utilizada foi a 2.11.1.

3.1.4 Benchmark TPC-E

Com o objetivo de se obter bons índices e métricas de aprensentação prática para o trabalho proposto, o Benchmark TPC-E foi utilizado como seu alimentador de transações já que é bem documentado, é de fácil uso, e reconhecido pela comunidade acadêmica.

Em 2007 o TPC-E foi lançado pelo *Transaction Processing Performance Council (TPC)* como substituto do TPC-C que estava em vigor desde 1992. Ambos são focados na carga de trabalho OLTP, simulando operações transacionais comuns entre vários negócios e aplicações utilizados até hoje. As diferenças e melhorias entre os dois benchmarks podem

ser vistas no artigo (CHEN et al., 2011) que foca na comparação de desempenho de escrita e leitura de dados (I/O) entre os dois benchmarks.

Este trabalho não tem por objetivo testar e avaliar o benchmark TPC-E. Isso já foi realizado pela comunidade em torno da TPC e pode ser analisado em (TPC, 2010). Sendo assim, para a finalidade deste trabalho, apresentar-se-á apenas o modelo de negócio ao qual o TPC-E foi projetado. O objetivo é empregar a carga de trabalho gerada pelo TPC-E em uma configuração *stand-alone* e em uma configuração em cluster, para avaliar o aumento da carga de processamento.

Este modelo de negócio simula uma corretora especializada em mercado financeiro e está organizado sob o tripé: títulos, contas e clientes. Sua composição prática está dividida em 33 tabelas de modelo transacional, sendo tabelas com dados fixos e tabelas que crescem de acordo com o tempo e volumetria da aplicação. Compondo as 33 tabelas tem-se 188 colunas com variados tipos de dados, entre eles estão tipos de dados contendo caracteres, valores numéricos, valores de data, booleanos, chaves e objetos grandes (LOB). Finalizando a composição prática, algumas restrições são adicionadas: 33 chaves primárias, 50 chaves estrangeiras e 22 restrições de verificação (check constraints) (TPC, 2010).

Para formular um banco de dados realítisco, os dados de clientes são gerados tendo como base o censo realizado no ano 2000 nos países Canadá e Estados Unidos da América. Dados envolvendo bolsas de valores também são baseados em dados reais, como nas bolsas NASDAQ e NYSE. Dessa forma, dir-se-á que os dados do benchmark TPC-E são pseudo-reais, refletindo, assim, o *modus operandi* de um modelo de negócio já existente (TPC, 2010).

Completando este breve resumo sobre o benchmark TPC-E, seu código foi escrito em C++ e está aberto para a comunidade dando a possibilidade de modificações conforme a necessidade de cada um. O código pode ser obtido no próprio site do TPC[1] e possui uma extensa documentação.

O presente trabalho fez uso do código original do TPC-E (TPC, 2010) e também um *fork* deste mesmo código, mas com algumas modificações feitas pela *Percona* que disponibilizou uma ferramenta a mais para testes simples de cargas (PERCONA, 2010). Ambos os códigos aqui utilizados foram compilados com a ferramenta *g++* no sistema Linux, distribuição Debian 10 (Buster).

3.1.4.1 Ferramentas do TPC-E

Após compilar o código, algumas ferramentas são disponibilizadas para realizar os testes de cargas. Neste trabalho foram utilizadas duas ferramentas: *EgenLoader* e *EGen-SimpleTest*. A ferramenta *EgenLoader* tem a finalidade de gerar os dados pseudo-reais em

[1] http://www.tpc.org/tpce/

arquivos com extensão *.txt* enquanto a *EGenSimpleTest* tem a finalidade de executar as cargas, utilizando os dados pseudo-reais, no SGBD. Os códigos 3.1 e 3.2 apresentam um modelo de execução em *shell script* para cada ferramenta.

Exemplo de código 3.1 – Ferramenta *EGenLoader*

```
./bin/EGenLoader -i flat_in -o flat_out -c <numero_de_clientes> \
   -t <clientes_ativos> -f <clientes_por_tpsE> \
   -w <dias_de_comercio>
```

Exemplo de código 3.2 – Ferramenta *EGenSimpleTest*

```
./bin/EGenSimpleTest -c <numero_de_clientes> -a <clientes_ativos> \
   -f <clientes_por_tpsE> -d <dias_de_negocios> \
   -l <numero_de_clientes_em_carga> -e flat_in -D <DSN> -U <usuario> \
   -P <senha> -t <duracao> -r <ramp_up> -u <numero_de_usuarios>
```

3.1.5 Amazon Web Services

A *Amazon Web Services* (AWS) é um provedor de nuvem que oferece uma gama de recursos de infraestrutura e ambientes computacionais disponíveis em todos os continentes do globo. Entre esses recursos, alguns dos principais são o *Elastic Compute Cloud* (EC2) que possibilita a criação e manutenção de máquinas virtuais em nuvem, o Amazon S3 que fornece o serviço de armazenamento de objetos, o Amazon RDS que permite a utilização e configuração rápida de bancos de dados relacionais, entre outros inúmeros serviços.

No presente trabalho apenas o recurso EC2 da AWS foi utilizado. Na Tabela 5 são apresentados os tipos de instâncias alocadas e suas configurações de hardware. Para o leitor é importante saber que os custos dessa infraestrutura foram pagos com a conta para estudantes da AWS que disponibilizou US$ 100 (cem dólares) para utilização. Nenhum valor a mais foi pago.

Servidor	**Tipo**	**Arquitetura**	**vCPU**	**RAM**	**SSD**	**Rede**
monitor	t2.large	x86_64	2	8 GB	8 GB	50-300 MB/s
mariadb01	c5.large	x86_64	2	4 GB	50 GB	10 Gbit
mariadb02	c5.large	x86_64	2	4 GB	50 GB	10 Gbit

Tabela 5 – Tipos de instâncias alocadas e suas configurações

O servidor **monitor** foi alocado para a utilização da ferramenta de monitoramento PMM (Percona Monitoring and Management) a fim de monitorar as cargas utilizadas no *benchmark*. Os servidores **mariadb01** e **mariadb02** foram utilizados para as instâncias do SGBDR MariaDB.

3.2 Objetivos e Metodologia

Como mencionado no Capítulo 1, este trabalho tem por objetivo avaliar a sobrecarga de replicação de bancos de dados relacionais. Para isso, duas arquiteturas distintas de bancos de dados foram utilizadas em comparação: (I) Banco de dados *stand-alone* e (II) Banco de dados replicado.

Dois foram os cenários aplicados nessa avaliação e ambos baseados na carga de trabalho do *benchmark* TPC-E: (I) O primeiro cenário utilizou 10 testes simples do TPC-E aplicados em *batch* e (II) o segundo cenário utilizou 100 testes simples do TPC-E com o mesmo objetivo. Cada cenário foi executado nas duas diferentes arquiteturas.

3.2.1 Arquiteturas em comparação

Banco de dados *stand-alone*: Pelo termo *stand-alone* deve-se entender uma instância de banco de dados executando de forma dedicada e que não está em replicação. As cargas de testes executadas nesta instância não trafegam pela rede e não utilizam o protocolo 2PC, dessa forma espera-se que esta arquitetura apresente melhor desempenho em comparação com outras. A figura 3 apresenta em detalhes sua organização.

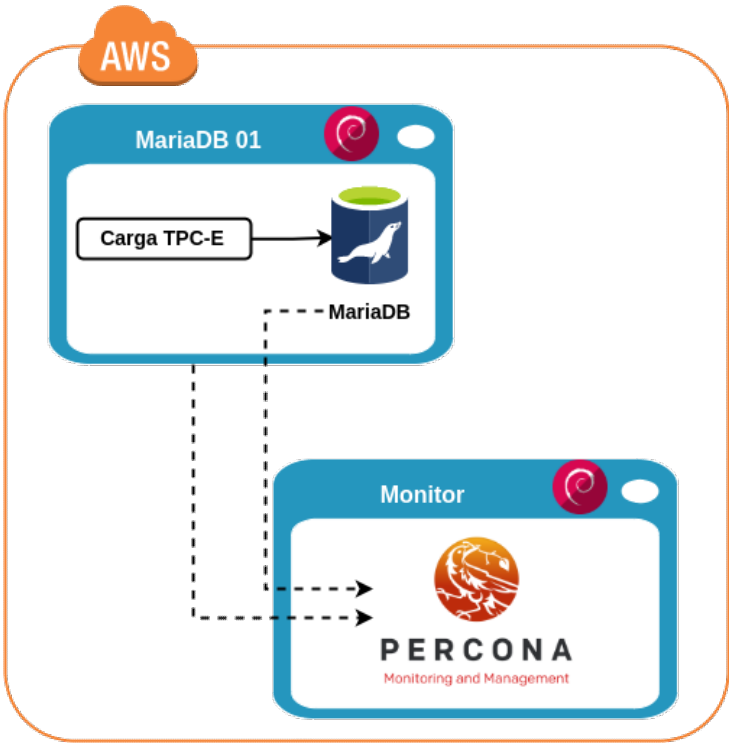

Figura 3 – Arquitetura centralizada.

Banco de dados replicado: A arquitetura de replicação foi realizada em dois servidores dedicados de MariaDB e acontece por meio da ferramenta Galera Cluster. Toda a carga de trabalho aplicada nessa arquitetura foi por meio de apenas um servidor,

o **mariadb01**. O Galera Cluster teve o papel de replicar de forma mútua forte todas as atualizações sobre os dados do servidor **mariadb01** para o servidor **mariadb02**.

A arquitetura de replicação aqui apresentada tem o objetivo de simular uma alta disponibilidade. Para tal, a instância **mariadb01** realiza o papel de servidor primário e a instância **mariadb02** realiza o papel de servidor *stand-by* (ou secundário). Por *stand-by* deve-se entender um servidor pronto para receber toda a carga da aplicação caso o servidor primário entre em estado de falha. A figura 4 apresenta em detalhes sua organização.

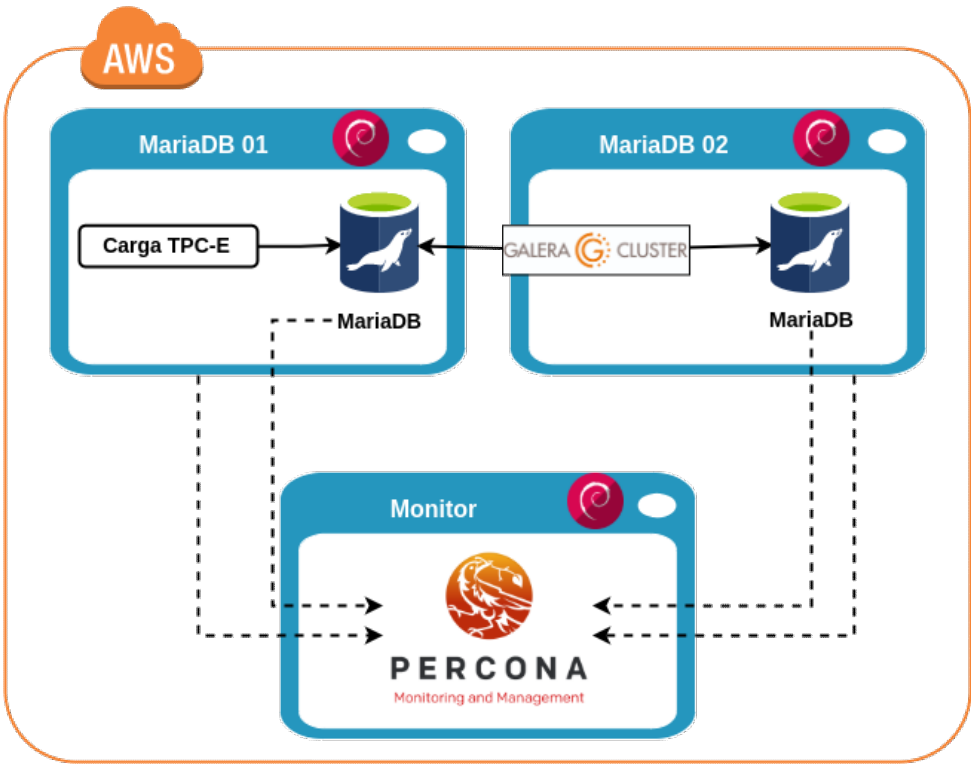

Figura 4 – Arquitetura replicada.

3.2.2 Cenários de testes

O código 3.3 foi executado a fim de gerar os dados pseudo-reais a serem utilizados nas cargas e testes. Uma primeira carga não monitorada foi executada em cada cenário de teste a fim de simular um banco de dados com algumas informações preexistentes. A Tabela 8 apresenta os arquivos gerados por esta ferramenta, seus tamanhos e a quantidade de registros em cada tabela do banco de dados após a carga inicial.

Exemplo de código 3.3 – Execução da ferramenta *EGenLoader*

```
1 ./bin/EGenLoader -i flat_in -o flat_out -c 2000 -t 2000 -f 200 -w 50
```

Primeiro cenário: O primeiro cenário de testes foi executado em cada uma das arquiteturas após a carga inicial. O código 3.4 foi executado dez vezes em seguida, onde o valor de "u" variou entre 2 e 25 em cada uma das dez execuções. Essa variação de "u"

indica que para cada execução havia uma quantidade de conexões simultâneas aplicando as transações paralelamente. A Tabela 6 mostra a quantidade aproximada de instruções relacionadas à Linguagem de Manipulação de Dados (do inglês *Data Manipulation Language* - DML) executadas ao longo do período de testes. Alguns trechos dessas transações podem ser visualizadas no Apêndice A.1.

Exemplo de código 3.4 – Execução da ferramenta *EGenSimpleTest*

```
./bin/EGenSimpleTest -c 2000 -a 2000 -f 200 -d 50 -l 200 -e flat_in \
   -D TPCE -U tpce -P tpce -t 30 -r 10 -u <valor>
```

Tipo de Instrução	**Quantidade**	**Percentagem**
SELECT	703.777	85,53%
INSERT	45.532	5,53%
UPDATE	69.622	8,46%
DELETE	3.902	0,47%
TOTAL	**822.833**	**100%**

Tabela 6 – Cenário 1: Quantidade aproximada de instruções DML

Segundo cenário: Para o segundo cenário de testes, o banco de dados foi recriado e a carga inicial foi refeita e, como no primeiro cenário, também foi executado nas duas arquiteturas. O mesmo script do primeiro cenário foi utilizado, porém a quantidade de execuções em seguida passou de dez para cem. O valor de "u" também variou entre 2 e 25 em cada uma das cem execuções. A Tabela 7 mostra a quantidade aproximada de instruções DML (Data Manipulation Language) executadas ao longo do período de testes. Alguns trechos dessas transações podem ser visualizadas no Apêndice A.2.

Tipo de Instrução	**Quantidade**	**Percentagem**
SELECT	7.076.291	85,32%
INSERT	465.035	5,60%
UPDATE	711.596	8,58%
DELETE	40.705	0,49%
TOTAL	**8.293.627**	**100%**

Tabela 7 – Cenário 2: Quantidade aproximada de instruções DML

Arquivo	**Tamanho**	**Tabela**	**Registros**
AccountPermission.txt	647 KB	account_permission	14.214
Address.txt	136 KB	address	3.004
Broker.txt	1.1 KB	broker	20
CashTransaction.txt	1.4 GB	cash_transaction	13.248.160
Charge.txt	165 B	charge	15
CommissionRate.txt	5.9 KB	commission_rate	240
Company.txt	204 KB	company	1.000
CompanyCompetitor.txt	74 KB	company_competitor	3.000
Customer.txt	322 KB	customer	2.000
CustomerAccount.txt	750 KB	customer_account	10.000
CustomerTaxrate.txt	59 KB	customer_taxrate	4.000
DailyMarket.txt	68 MB	daily_market	1.787.850
Exchange.txt	375 B	exchange	4
Financial.txt	2.7 MB	financial	20.000
Holding.txt	75 MB	holding	1.148.611
HoldingHistory.txt	715 MB	holding_history	19.166.596
HoldingSummary.txt	2.2 MB	holding_summary	99.684
Industry.txt	2.7 KB	industry	102
LastTrade.txt	59 KB	last_trade	1.370
NewsItem.txt	192 MB	news_item	2.000
NewsXRef.txt	31 KB	news_xref	2.000
Sector.txt	193 B	sector	12
Security.txt	204 KB	security	1.370
Settlement.txt	665 MB	settlement	14.400.000
StatusType.txt	69 B	status_type	5
Taxrate.txt	17 KB	taxrate	320
Trade.txt	1.6 GB	trade	14.400.000
TradeHistory.txt	1.5 GB	trade_history	34.559.809
-	-	trade_request	0
TradeType.txt	94 B	trade_type	5
WatchItem.txt	3.2 MB	watch_item	198.048
WatchList.txt	43 KB	watch_list	2.000
ZipCode.txt	278 KB	zip_code	14.741

Tabela 8 – Arquivos, tamanhos, tabelas e registros

3.3 Resultados, execuções e métricas

A seguir serão apresentadas avaliações e comparações das arquiteturas em cada um dos cenários utilizando as métricas obtidas nas execuções dos códigos já apresentados na seção 3.2.2. Primeiramente são apresentados os consumos de hardware como tempo total de execução, utilização de CPU, consumo de memória RAM, entrada e saída de disco, tráfego de rede Inbound e, logo após, os consumos de leituras e escritas do SGBDR MariaDB como a quantidade de registros afetados pela leitura por segundo e registros afetados pela escrita por segundo. As métricas apresentadas em cada uma das próximas subseções foram retiradas do PMM (Grafana). Ao final, a subseção 3.3.3 apresenta uma breve discussão sobre os resultados obtidos e possíveis implementações de cada arquitetura.

Antes de iniciar a avaliação é interessante notar que, conforme visualizado nas tabelas 6 e 7, a carga do TPC-E utiliza uma quantidade maior de consultas do que escritas. Além disso, o fato da arquitetura replicada ser composta por um servidor primário e um servidor *stand-by* a avaliação tem um maior enfoque no servidor primário dessa arquitetura em contraste com o servidor da arquitetura *single-node*, porquanto a carga em replicação é executada na instância **mariadb01**, apenas.

3.3.1 Consumos, gráficos e resultados do primeiro cenário

3.3.1.1 Tempo total de execução

A diferença entre o tempo das execuções é de 1 minuto e 17 segundos a mais na arquitetura replicada. Essa arquitetura precisou cerca de 18,19% a mais de tempo para finalizar suas execuções, um tempo consideravelmente maior para uma carga transacional.

Arquitetura centralizada:

- Início da execução: 22:52:28
- Final da execução: 22:57:54
- Tempo total de execução: 5 minutos e 26 segundos.

Arquitetura replicada:

- **Início da execução:** 00:29:47
- **Final da execução:** 00:36:30
- **Tempo total de execução:** 6 minutos e 43 segundos.

3.3.1.2 Utilização de CPU

Em se tratanto de hardware, o consumo de CPU na arquitetura replicada (Figura 6) foi bem superior na instância **mariadb01** devido aos comandos de consulta serem executados apenas na instância primária (comandos de consulta não são replicados) e também às operações do Galera Cluster. Esse consumo chegou a atingir o máximo possível disponibilizado pelo servidor. Por outro lado, na arquitetura *single-node* o consumo de CPU não passou de 41% (Figura 5), indicando que o processo 2PC necessita de um melhor poder de processamento para garantir a consistência dos dados replicados em operações de curta duração.

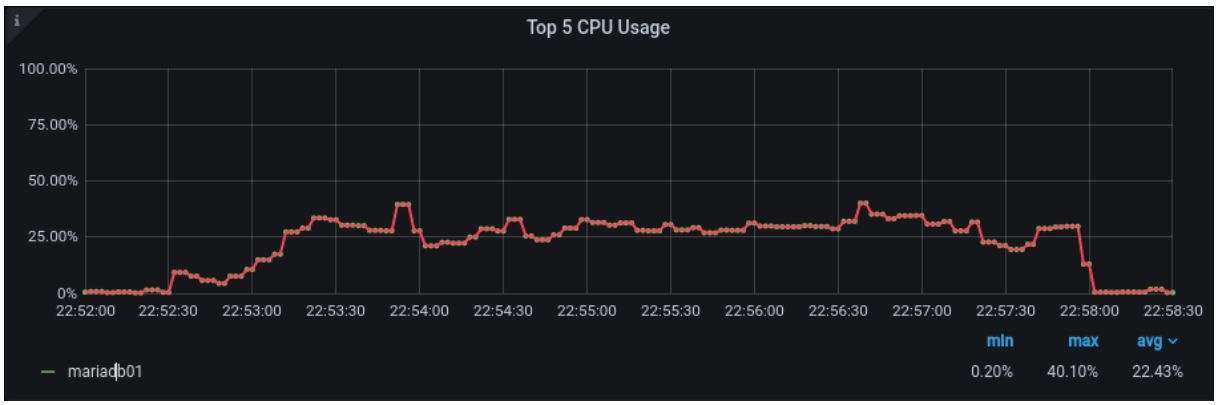

Figura 5 – Primeiro cenário. Arquitetura single-node. Consumo de CPU. Eixo Y em escala linear.

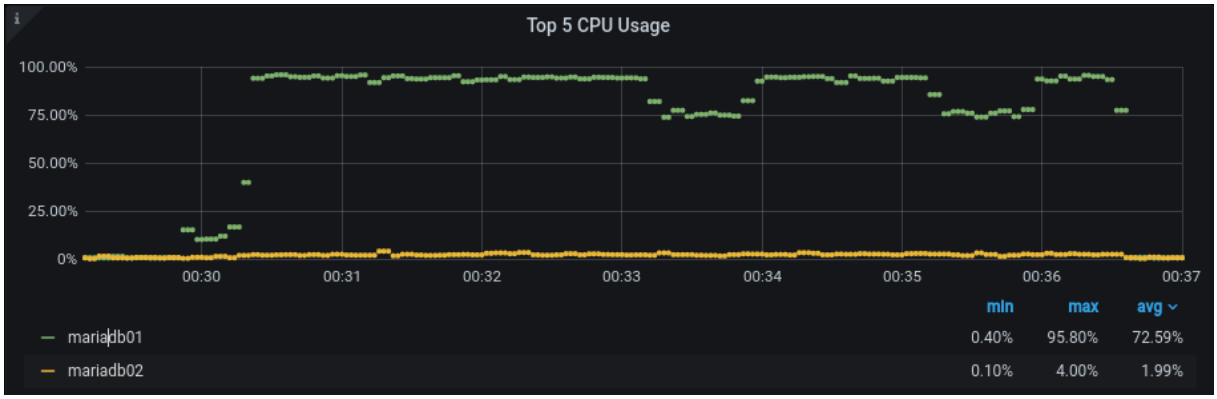

Figura 6 – Primeiro cenário. Arquitetura replicada. Consumo de CPU. Eixo Y em escala linear.

3.3.1.3 Consumo de memória RAM

A utilização de memória RAM em ambas as arquiteturas não teve grandes discrepâncias. Isso se deve pelo fato do *buffer pool* (dados armazenados na memória RAM para acesso rápido) estar utilizando apenas 128 MB do total de memória disponível em cada servidor (4 GB). Isso fez com que as consultas do *benchmark* utilizassem mais a carga de disco (E/S ou I/O) do que a memória RAM para o retorno de informações (ver figuras 7 e 8).

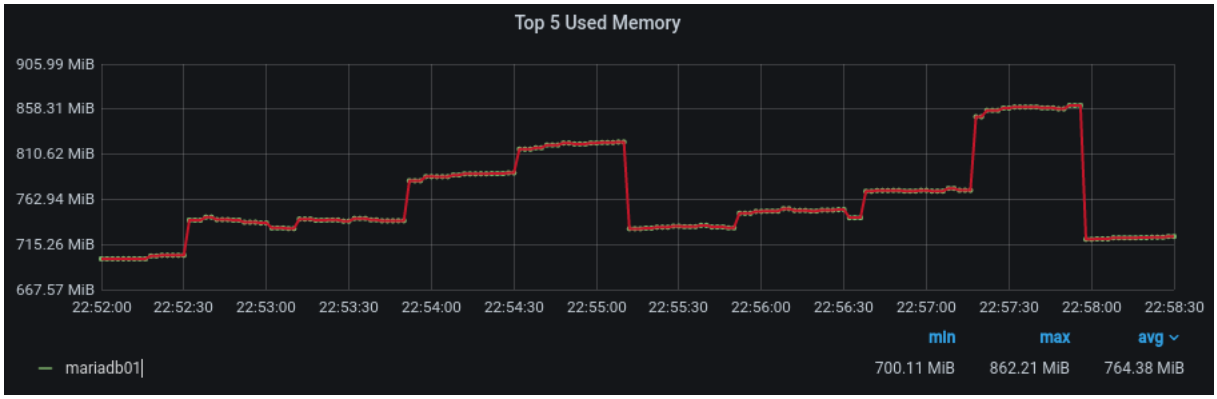

Figura 7 – Primeiro cenário. Arquitetura single-node. Consumo de memória RAM. Eixo Y em escala linear.

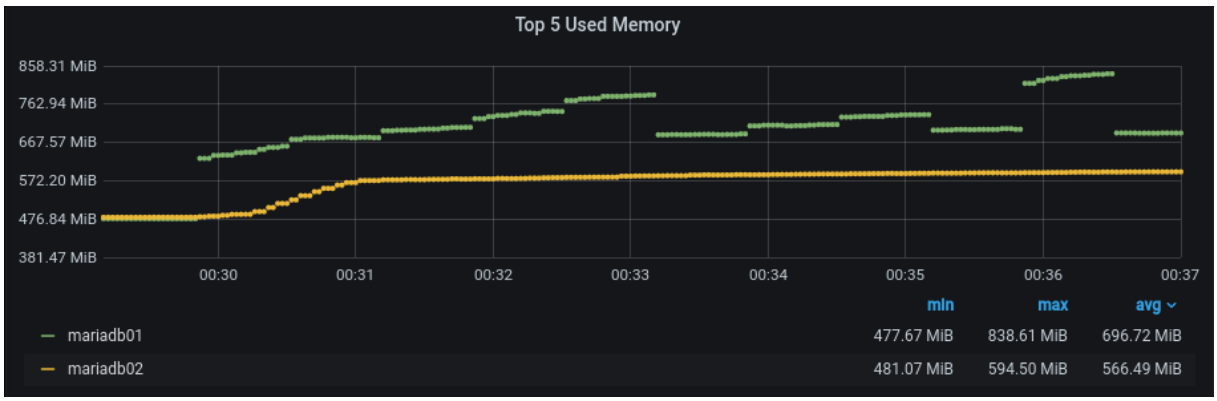

Figura 8 – Primeiro cenário. Arquitetura replicada. Consumo de memória RAM. Eixo Y em escala linear.

3.3.1.4 Entrada e saída de disco

O consumo de disco chegou a quase 76 MB/s na arquitetura *single-node* (Figura 9) e no servidor primário da arquitetura replicada chegou a quase 77 MB/s (Figura 10), dados quase idênticos. O gráfico das operações de E/S envolve tanto leituras quanto escritas ao disco. Ambos os servidores mantiveram este consumo em quase todo o tempo de processamento. No servidor *stand-by* da replicação o consumo foi bem inferior por não receber muitos processos de leituras e a carga do TPC-E estava toda concentrada no nó primário.

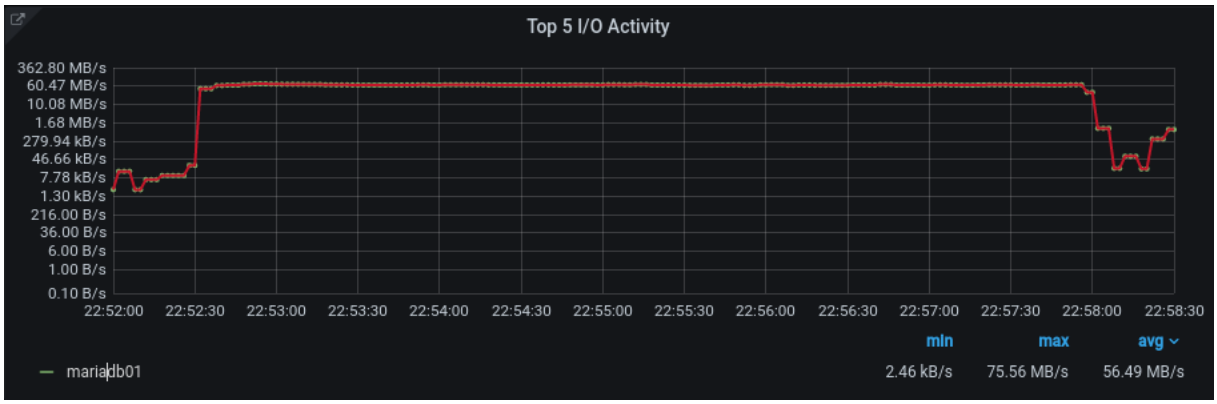

Figura 9 – Primeiro cenário. Arquitetura single-node. Consumo de disco. Eixo Y em escala logarítmica.

Figura 10 – Primeiro cenário. Arquitetura replicada. Consumo de disco. Eixo Y em escala logarítmica.

3.3.1.5 Tráfego de rede Inbound

A arquitetura *single-node* não utiliza rede, portanto seu gráfico não foi incluído nesta subseção. Já o servidor *stand-by* da arquitetura replicada chegou a ter um tráfego *Inbound* de 163,70 KB/s e o servidor primário ficou na média de 13 KB/s (Figura 11). O protocolo 2PC componente da arquitetura replicada faz muitas chamadas de verificação de integridade e tráfego de dados para manter os servidores idênticos.

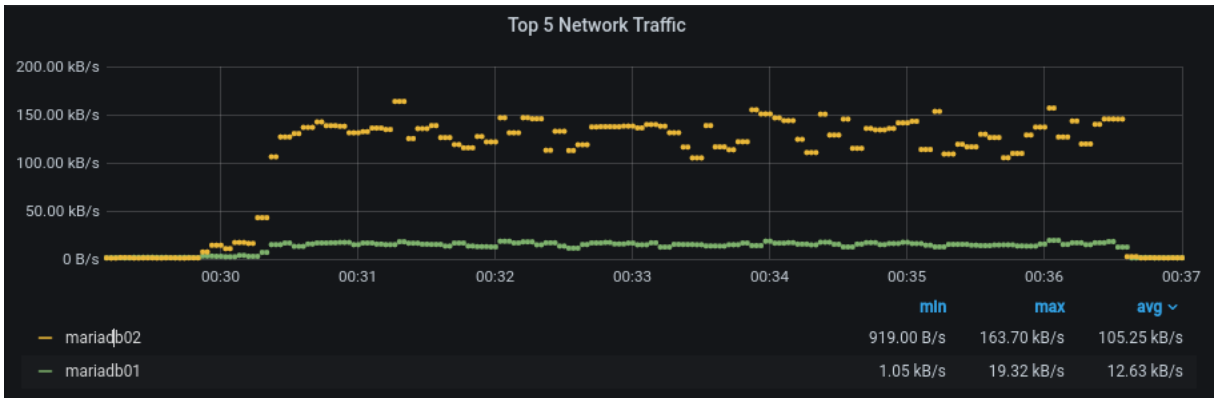

Figura 11 – Primeiro cenário. Arquitetura replicada. Tráfego de rede Inbound. Eixo Y em escala linear.

3.3.1.6 Engine InnoDB - Leituras

Partindo para a análise interna da Engine InnoDB, a arquitetura *single-node* chegou a ler, no máximo, 69.000 registros por segundo (Figura 12). Quase todo seu processo de leitura se manteve acima de 45.000 registros por segundo. A arquitetura replicada chegou a 195.000 registros lidos por segundo na instância **mariadb01** (Figura 13) e a quase 235 na instância **mariadb02** (Figura 14). Esta quantidade maior no servidor *stand-by* da arquitetura replicada aconteceu pois os comandos de consulta não foram replicados para o servidor secundário, sobrecarregando assim apenas o servidor primário.

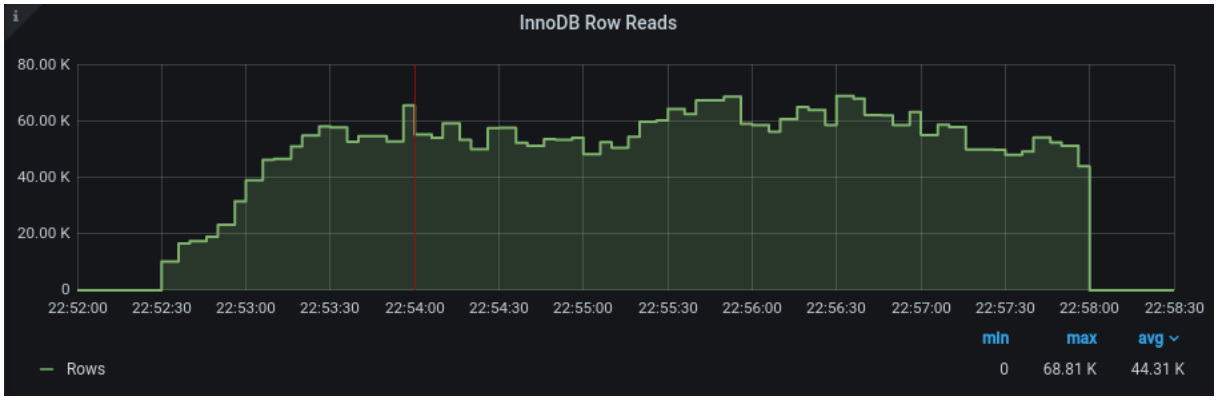

Figura 12 – Primeiro cenário. Arquitetura single-node. Registros lidos. Eixo Y em escala linear.

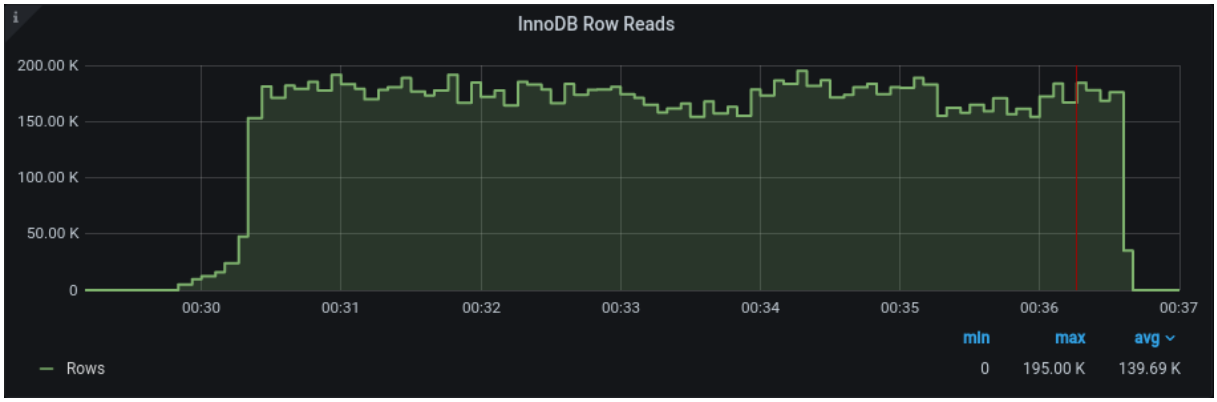

Figura 13 – Primeiro cenário. Arquitetura replicada. Registros lidos no servidor mariadb01. Eixo Y em escala linear.

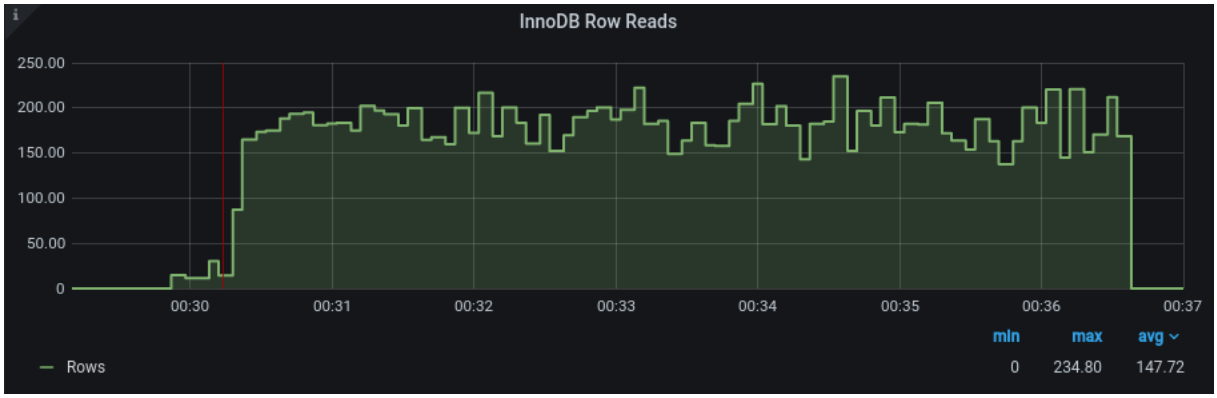

Figura 14 – Primeiro cenário. Arquitetura replicada. Registros lidos no servidor mariadb02. Eixo Y em escala linear.

3.3.1.7 Engine InnoDB - Escritas

O processamento de escritas na Engine InnoDB que envolve inserções, atualizações e deleções não seguiu o padrão das leituras na subseção 3.3.1.6. A arquitetura *single-node* se manteve ocupada com essas operações por todo o tempo de execução, chegando a 184 registros escritos por segundo e mantendo a média de aproximadamente 90 registros escritos por segundo (Figura 15). Em contrapartida a arquitetura replicada chegou a 432 na instância **mariadb01** (Figura 16) e 418 na instância **mariadb02** (Figura 17) em seus maiores picos e ficaram na média de 256 nas duas instâncias.

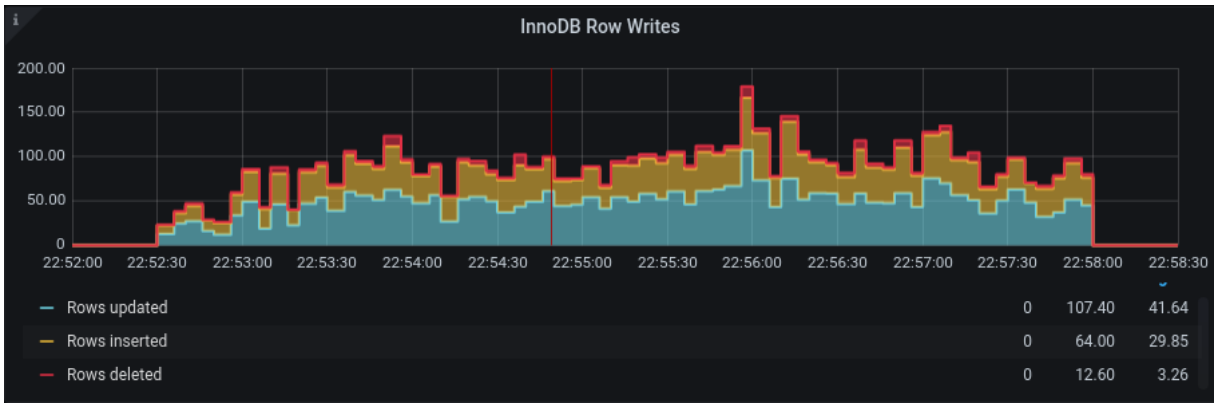

Figura 15 – Primeiro cenário. Arquitetura single-node. Registros escritos. Eixo Y em escala linear.

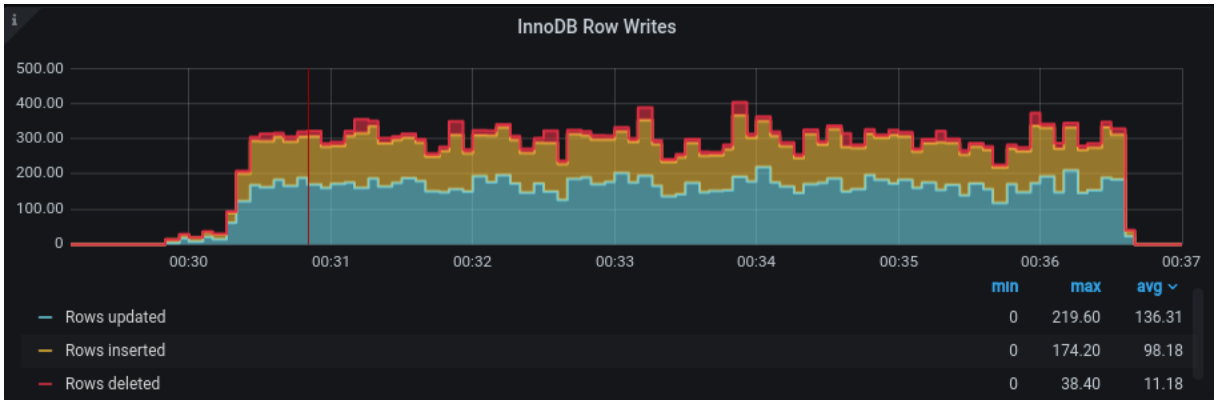

Figura 16 – Primeiro cenário. Arquitetura replicada. Registros escritos no servidor mariadb01. Eixo Y em escala linear.

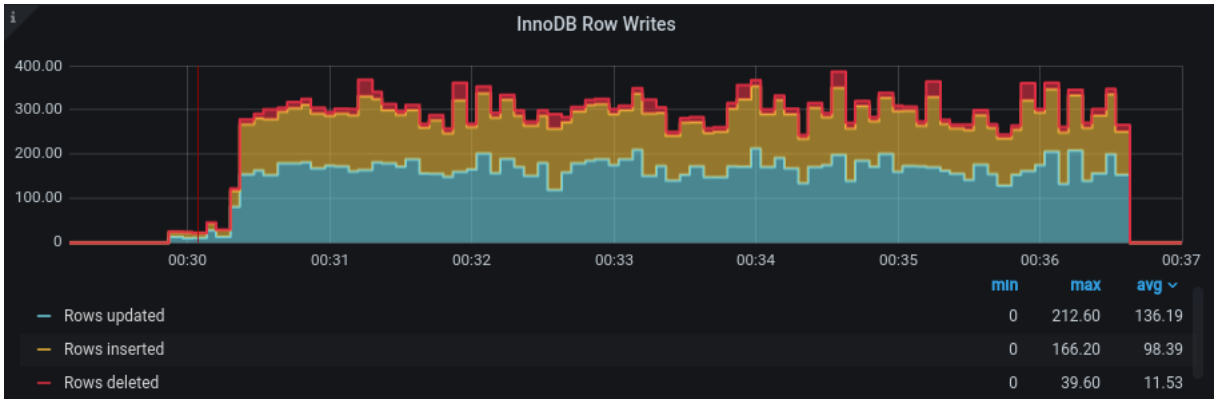

Figura 17 – Primeiro cenário. Arquitetura replicada. Registros escritos no servidor mariadb02. Eixo Y em escala linear.

3.3.2 Consumos, gráficos e resultados do segundo cenário

3.3.2.1 Tempo total de execução

A diferença entre o tempo das execuções é de apenas 39 segundos a mais na arquitetura replicada. Essa arquitetura precisou cerca de 1,02% a mais de tempo para finalizar suas execuções. Este tempo é ínfimo considerando seu tempo maior de execução em toda a carga e se comparada com o primeiro cenário.

Arquitetura centralizada:

- **Início da execução:** 19:21:03
- **Final da execução:** 20:23:37
- **Tempo total de execução:** 1 hora 2 minutos e 34 segundos.

Arquitetura replicada:

- **Início da execução:** 10:53:39
- **Final da execução:** 11:56:52
- **Tempo total de execução:** 1 hora 3 minutos e 13 segundos.

3.3.2.2 Utilização de CPU

Analisando o hardware, o consumo de CPU nas duas arquiteturas ficou semelhante. A arquitetura *single-node* se manteve na média de 85% de utilização (Figura 18) enquanto na arquitetura replicada o servidor primário se manteve na média de 88% e o *stand-by* em 4% (Figura 19).

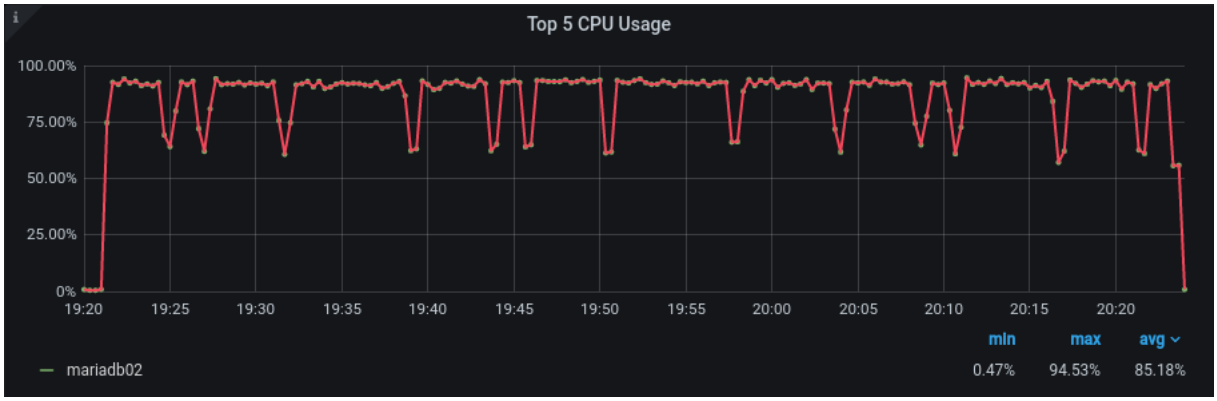

Figura 18 – Segundo cenário. Arquitetura single-node. Consumo de CPU. Eixo Y em escala linear.

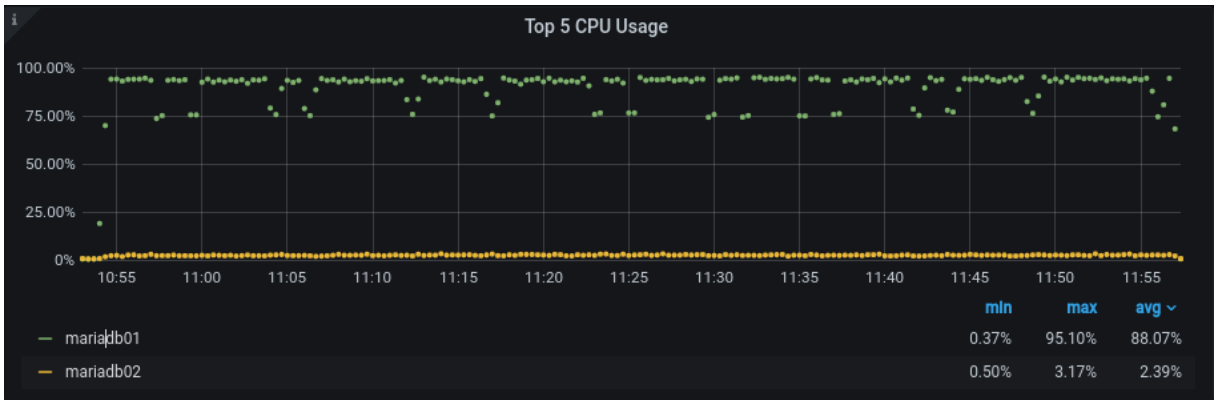

Figura 19 – Segundo cenário. Arquitetura replicada. Consumo de CPU. Eixo Y em escala linear.

3.3.2.3 Consumo de memória RAM

Assim como no primeiro cenário, a utilização de memória RAM em ambas as arquiteturas (Figuras 20 e 21) não teve grandes discrepâncias pelo mesmo motivo explicado anteriormente: baixa quantidade de memória alocada para o *buffer pool*.

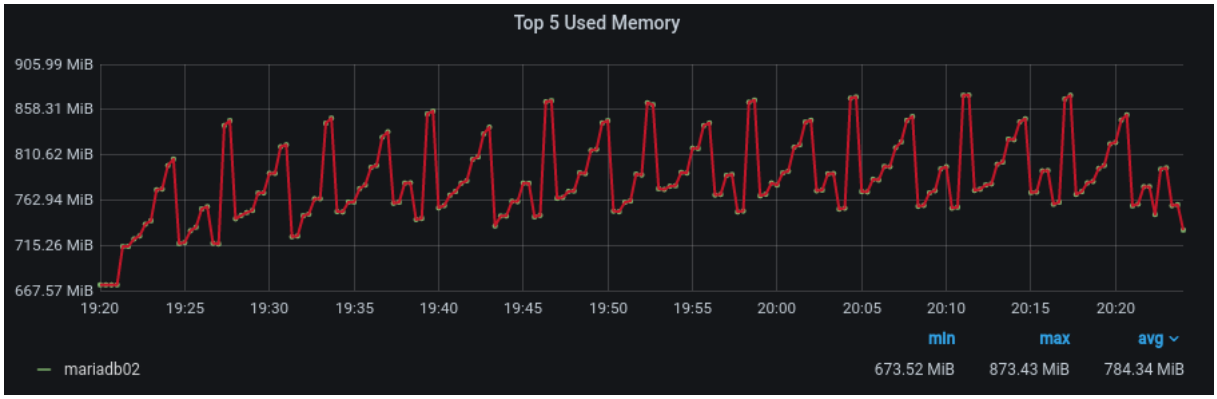

Figura 20 – Segundo cenário. Arquitetura single-node. Consumo de memória RAM. Eixo Y em escala linear.

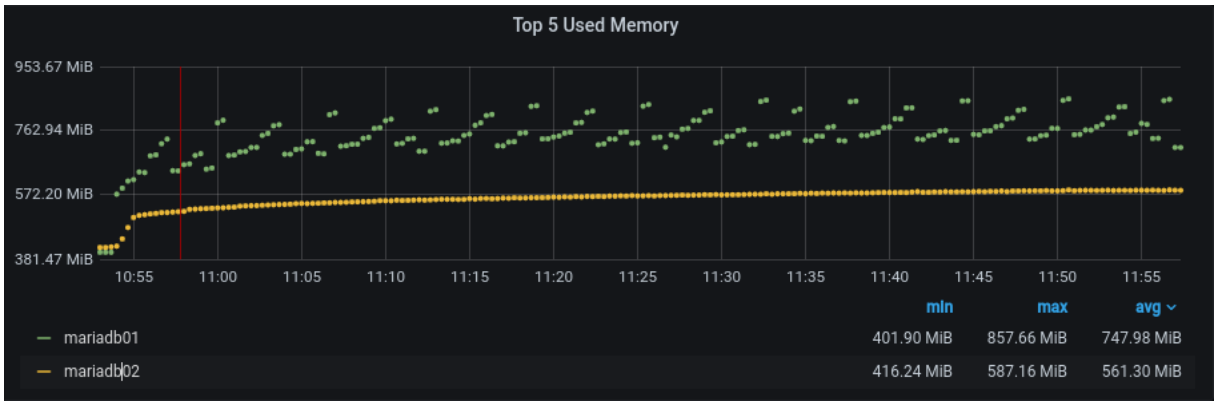

Figura 21 – Segundo cenário. Arquitetura replicada. Consumo de memória RAM. Eixo Y em escala linear.

3.3.2.4 Entrada e saída de disco

O consumo de disco chegou a quase 38 MB/s na arquitetura *single-node* (Figura 22) e no servidor primário da arquitetura replicada chegou a quase 66 MB/s (Figura 23). Ambos os servidores mantiveram este consumo em quase todo o tempo de processamento.

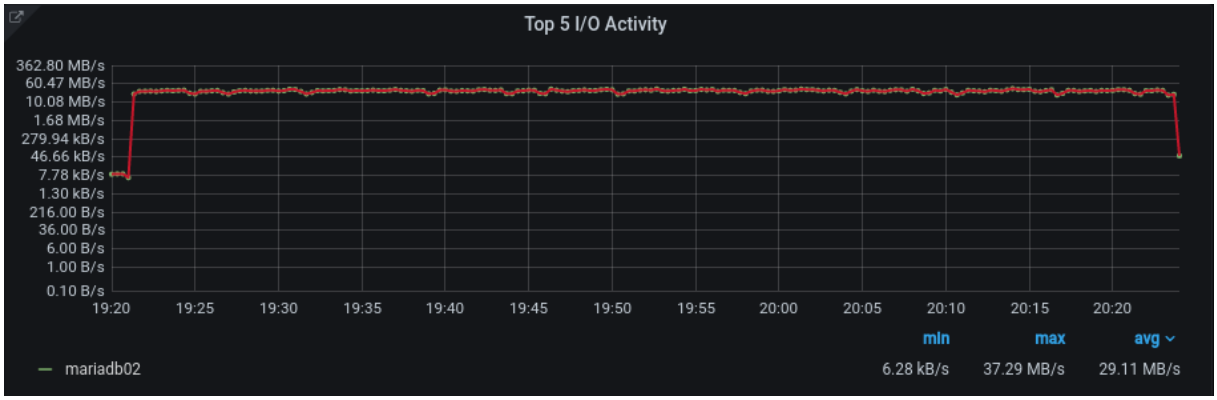

Figura 22 – Segundo cenário. Arquitetura single-node. Consumo de disco. Eixo Y em escala logarítmica.

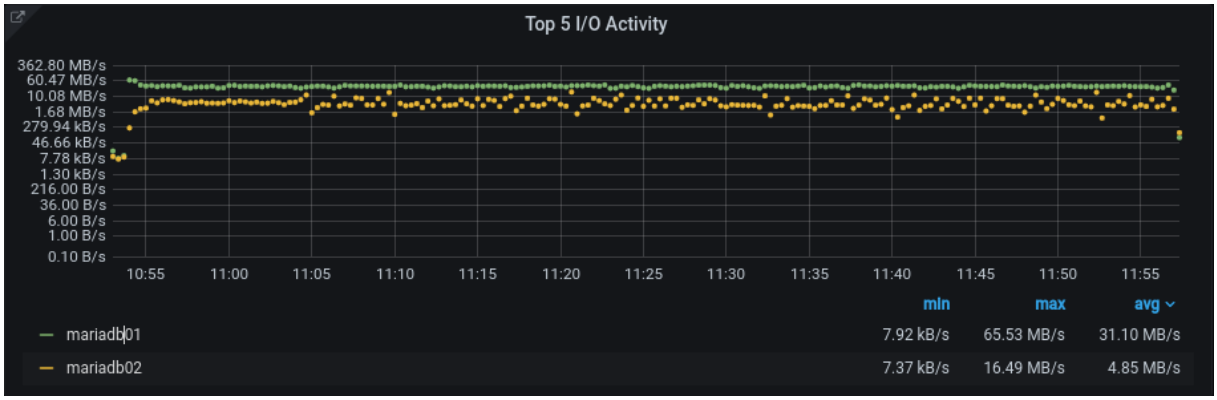

Figura 23 – Segundo cenário. Arquitetura replicada. Consumo de disco. Eixo Y em escala logarítmica.

3.3.2.5 Tráfego de rede Inbound

A arquitetura *single-node* deste cenário também não utiliza rede, portanto seu gráfico não foi incluído nesta subseção. Por outro lado, o servidor *stand-by* da arquitetura replicada chegou a utilizar um tráfego *Inbound* de 157,17 KB/s e o servidor primário ficou na média de 16 KB/s (Figura 24).

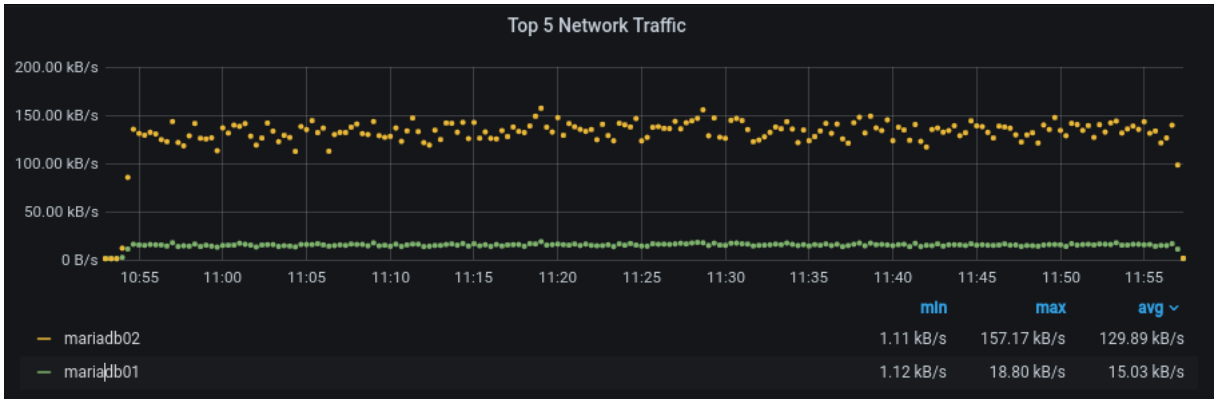

Figura 24 – Segundo cenário. Arquitetura replicada. Tráfego de rede Inbound. Eixo Y em escala linear.

3.3.2.6 Engine InnoDB - Leituras

Partindo para a análise interna da Engine InnoDB, a arquitetura *single-node* chegou a ler, no máximo, 190.000 registros por segundo (Figura 25). Semelhante ao primeiro cenário, a arquitetura replicada no segundo cenário chegou a 192.000 registros lidos por segundo na instância **mariadb01** (Figura 26) e a quase 217 na instância **mariadb02** (Figura 27).

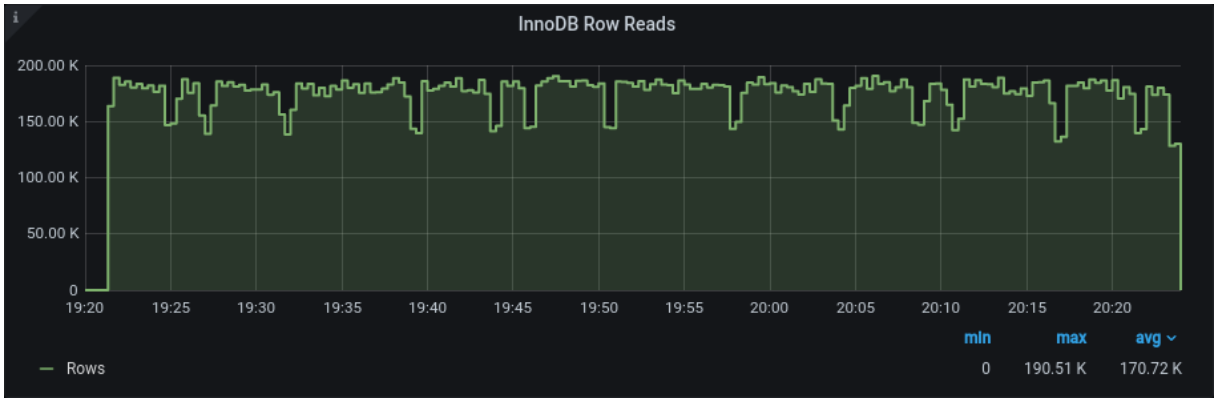

Figura 25 – Segundo cenário. Arquitetura single-node. Registros lidos. Eixo Y em escala linear.

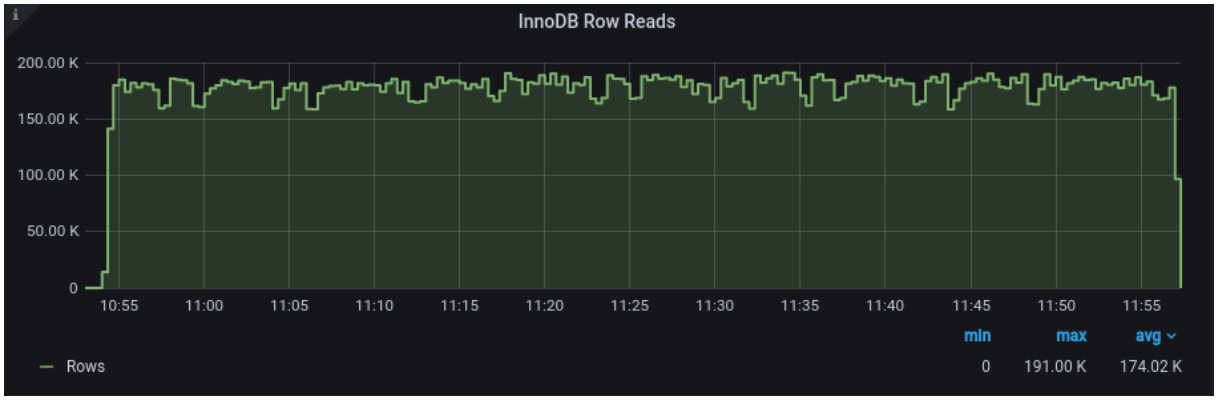

Figura 26 – Segundo cenário. Arquitetura replicada. Registros lidos no servidor mariadb01. Eixo Y em escala linear.

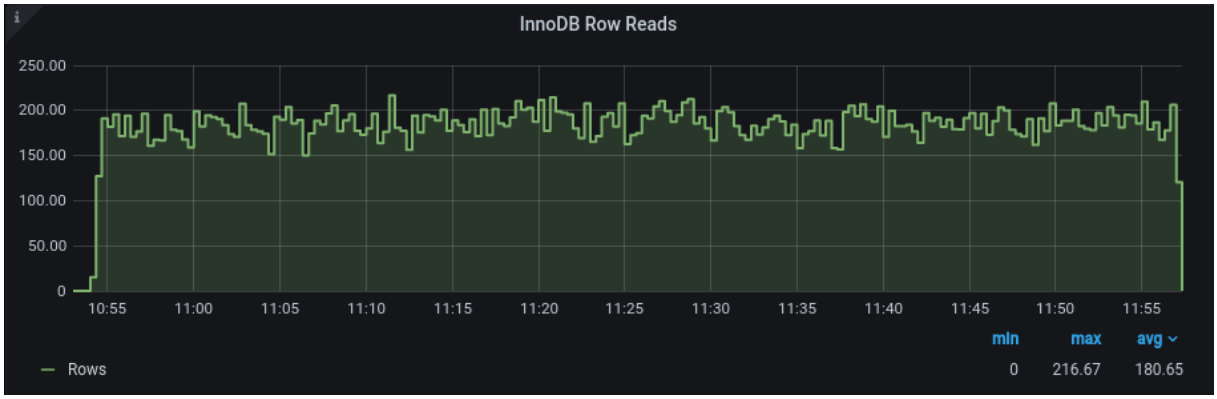

Figura 27 – Segundo cenário. Arquitetura replicada. Registros lidos no servidor mariadb02. Eixo Y em escala linear.

3.3.2.7 Engine InnoDB - Escritas

O processamento de escritas na Engine InnoDB para a arquitetura *single-node* chegou a 378 registros escritos por segundo e mantendo a média de aproximadamente 299 registros escritos por segundo (Figura 28). De forma semelhante a arquitetura replicada chegou a 370 nas duas instâncias e com médias de 300 também nas duas instâncias (Figuras 29 e 30).

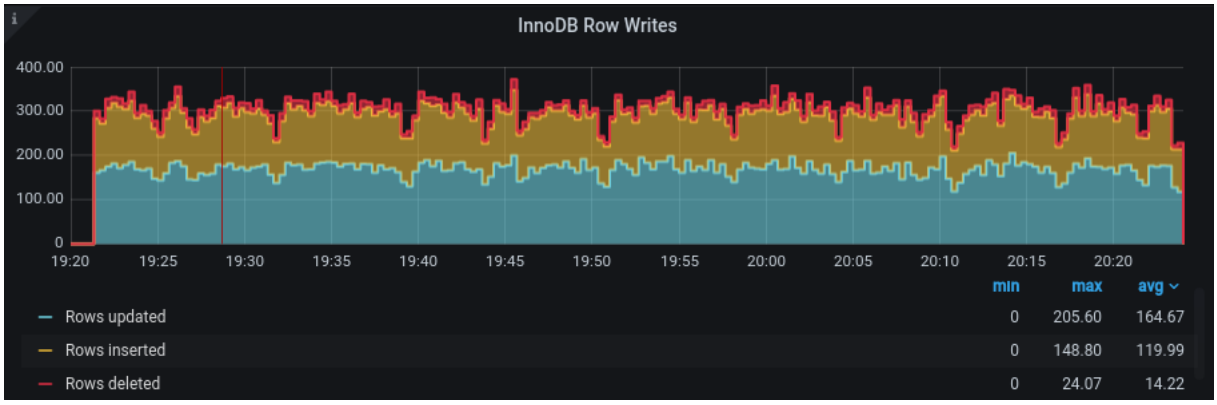

Figura 28 – Segundo cenário. Arquitetura single-node. Registros escritos. Eixo Y em escala linear.

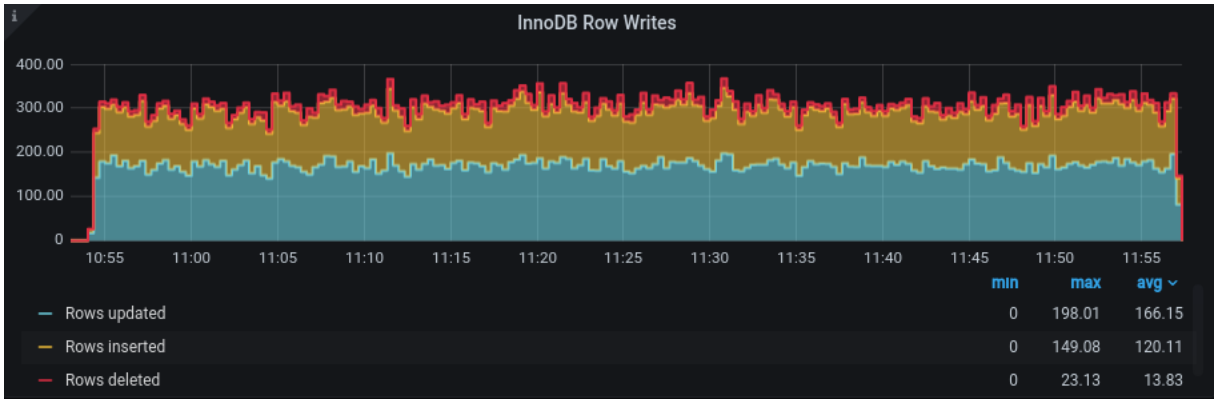

Figura 29 – Segundo cenário. Arquitetura replicada. Registros escritos no servidor mariadb01. Eixo Y em escala linear.

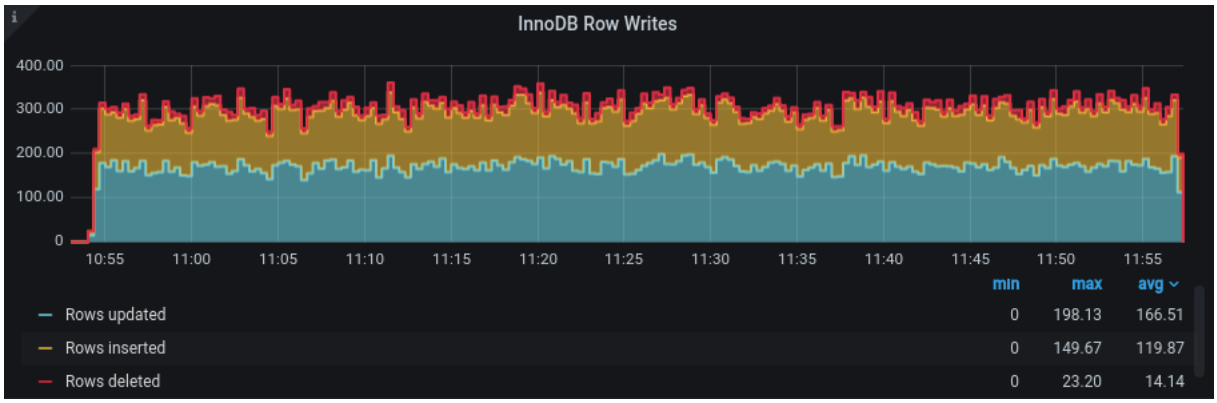

Figura 30 – Segundo cenário. Arquitetura replicada. Registros escritos no servidor mariadb02. Eixo Y em escala linear.

3.3.3 Discussão

Baseado nas avaliações anteriores é possível inferir que para execuções de pequenas transações por longos períodos de tempo e mantendo-as em quantidade estável, o desempenho da arquitetura replicada se mantém semelhante ao desempenho da arquitetura *single-node*. Por outro lado, transações em quantidades estáveis, mas executadas em períodos curtos, tendem a ser consideravelmente mais rápidas na arquitetura *single-node*. A arquitetura *single-node* demanda menos recursos de hardware para executar as mesmas operações.

É importante notar que na arquitetura replicada apenas dois servidores foram utilizados com a finalidade de apresentar o conceito de alta disponibilidade em bancos de dados. Para garantir essa alta disponibilidade, o DBA (Database Administrator) pode configurar um balanceador de cargas (ou proxy) para organizar as conexões nestas duas máquinas, de forma que caso o servidor primário esteja indisponível todas as conexões passem a utilizar o servidor *stand-by*. Além disso, com as ferramentas aqui apresentadas é possível adicionar mais servidores ao cluster com a implicação de que haverá aumento no tempo de replicação das transações.

Além disso também é importante notar que o protocolo 2PC não impôs um grande impacto na arquitetura replicada visto que as máquinas virtuais não estavam geograficamente distribuídas. Elas estavam na mesma rede local de acordo com as implementações internas da AWS.

Ambas as arquiteturas podem ser utilizadas de acordo com a necessidade de uma aplicação. Caso a aplicação demande um tempo alto de disponibilidade do banco de dados, isto é, o banco de dados deve estar apto a responder as requisições em grande parte do tempo então é desejável que ele seja replicado. Em contrapartida, caso a aplicação seja simples e não demande uma alta disponibilidade, ou precise de um maior desempenho, a arquitetura *single-node* satisfaz essa demanda.

4 Conclusão

Hoje em dia, o mundo está se digitalizando cada vez mais e os bancos de dados tem um papel muito importante para que isso aconteça. A disponibilidade e velocidade de resposta de algumas aplicações afetam diretamente seus usuários que, por sua vez, estão ficando mais exigentes quanto a essas características. Por isso os bancos de dados replicados são essenciais nas arquiteturas de aplicações com esse viés, pois eles garantem um grau de disponibilidade ótimo para o bem de suas aplicações e, consequentemente, de seus usuários.

O desenvolvimento prático desta monografia apresentou uma avaliação de sobrecarga na replicação de bancos de dados relacionais tendo como ponto central a comparação de desempenho entre uma arquitetura replicada de banco de dados e uma arquitetura centralizada. Utilizou-se, principalmente, de métricas de hardware como CPU, memória RAM, disco e rede, além de métricas internas de software como a taxa de escritas e leituras na Engine InnoDB.

Um objetivo secundário deste trabalho foi mostrar como é possível montar ótimas arquiteturas (e completas) de bancos de dados em nuvem com ferramentas de código aberto e livres como o MariaDB, PMM, Galera Cluster e avaliá-las com o TPC-E. Essas ferramentas diminuem drasticamente o custo para aplicações que demandam alta disponibilidade e velocidade de respostas em comparação com bancos de dados comerciais. A comunidade em volta de ferramentas de código aberto tende a crescer e torná-las mais competitivas em frente ao mercado comercial.

Os dados apresentados foram satisfatórios para a arquitetura replicada, visto que ela não gerou grande sobrecarga de desempenho em relação a arquitetura centralizada. Portanto aplicações variadas podem usufruir tanto de uma ótima alta disponibilidade, quanto dos custos baixos das ferramentas de código aberto.

4.1 Trabalhos Futuros

Para que o presente trabalho pudesse acontecer algumas disciplinas foram utilizadas: Bancos de Dados 1 e 2, Sistemas Operacionais, Redes de Computadores, Sistemas Distribuídos e Engenharia de Software.

Em trabalhos futuros pretende-se utilizar a arquitetura de replicação apresentada em aplicações reais fazendo uso de ferramentas de proxy que automatizam o *failover* das réplicas, fazendo o aumento do número de nós replicados, e fazendo a comparação desses resultados com outras ferramentas de código aberto e ferramentas comerciais. Outro

ponto a ser extendido é a utilização de novas funcionalidades de replicação fornecidades pela comunidade e otimizá-las de acordo com cada arquitetura, além de extender os tipos de arquiteturas que podem ser utilizadas em uma alta disponibilidade de SGBDRs.

Referências

BERNSTEIN, P. A. et al. Query processing in a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, Association for Computing Machinery, New York, NY, USA, v. 6, n. 4, p. 602–625, dez. 1981. ISSN 0362-5915. Disponível em: <https://doi.org/10.1145/319628.319650>. Citado 2 vezes nas páginas 14 e 15.

CAMPÊLO, R. A. et al. A brief survey on replica consistency in cloud environments. *J. Internet Serv. Appl.*, v. 11, n. 1, p. 1, 2020. Disponível em: <https://doi.org/10.1186/s13174-020-0122-y>. Citado 3 vezes nas páginas 19, 27 e 28.

CHEN, S. et al. Tpc-e vs. tpc-c: Characterizing the new tpc-e benchmark via an i/o comparison study. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 39, n. 3, p. 5–10, fev. 2011. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1942776.1942778>. Citado na página 31.

CODD, E. F. A relational model of data for large shared data banks. *Commun. ACM*, ACM, New York, NY, v. 13, n. 6, p. 377–387, jun. 1970. ISSN 0001-0782. Citado na página 11.

CODERSHIP. *Galera Cluster Documentation*. 4. ed. Helsinki, Finland, 2020. Disponível em: <https://galeracluster.com/library/documentation/>. Citado na página 30.

DATE, C. J. Conceitos básicos. In: _____. *Introdução a Sistemas de Banco de Dados*. [S.l.]: Editora Campus, 1989. p. 589–623. ISBN 85-7001-392-2. Citado na página 11.

DHAMANE, R. et al. Performance evaluation of database replication systems. In: *Proceedings of the 18th International Database Engineering; Applications Symposium*. New York, NY, USA: Association for Computing Machinery, 2014. (IDEAS '14), p. 288–293. ISBN 9781450326278. Disponível em: <https://doi.org/10.1145/2628194.2628214>. Citado 4 vezes nas páginas 6, 26, 27 e 28.

ELMASRI, R.; NAVATHE, S. B. Bancos de dados distribuídos (capítulo 25). In: _____. *Sistemas de banco de dados*. [S.l.]: Pearson, 2011. p. 589–623. ISBN 978-85-4301-381-7. Citado 3 vezes nas páginas 15, 16 e 24.

ELMASRI, R.; NAVATHE, S. B. Bancos de dados e usuários de banco de dados (capítulo 1). In: _____. *Sistemas de banco de dados*. [S.l.]: Pearson, 2011. p. 2–18. ISBN 978-85-4301-381-7. Citado na página 11.

ELMASRI, R.; NAVATHE, S. B. Conceitos e arquitetura do sistema de banco de dados (capítulo 2). In: _____. *Sistemas de banco de dados*. [S.l.]: Pearson, 2011. p. 19–36. ISBN 978-85-4301-381-7. Citado 2 vezes nas páginas 15 e 16.

ELMASRI, R.; NAVATHE, S. B. Introdução aos conceitos e teoria de processamento de transações (capítulo 21). In: _____. *Sistemas de banco de dados*. [S.l.]: Pearson, 2011. p. 500–522. ISBN 978-85-4301-381-7. Citado 2 vezes nas páginas 18 e 19.

ELMASRI, R.; NAVATHE, S. B. Técnicas de recuperação de banco de dados (capítulo 23). In: _____. *Sistemas de banco de dados*. [S.l.]: Pearson, 2011. p. 543–560. ISBN 978-85-4301-381-7. Citado na página 19.

FOUNDATION, M. *About MariaDB Server*. 2020. Website. Disponível em: <https://mariadb.org/about/>. Citado na página 29.

MANICA, H. *Bancos de dados distribuídos heterogêneos: arquiteturas, tecnologias e tendências*. Dissertação de Mestrado, Programa de Pós-graduação em Ciência da Computação, Universidade Federal de Santa Catarina, 2001. Disponível em: <http://repositorio.ufsc.br/xmlui/handle/123456789/80286>. Citado na página 17.

MCMINN, P.; WRIGHT, C. J.; KAPFHAMMER, G. M. The effectiveness of test coverage criteria for relational database schema integrity constraints. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 25, n. 1, dez. 2015. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/2818639>. Citado na página 21.

MOIZ, S. A. et al. Database replication: A survey of open source and commercial tools. *International Journal of Computer Applications*, v. 13, 01 2010. Citado 2 vezes nas páginas 26 e 28.

ORAM, A. Description. In: _____. *Peer to Peer: Harnessing the Power of Disruptive Technologies*. [S.l.]: O'Reilly, 2001. p. 4. ISBN 0-596-00110-X. Citado na página 15.

OZSU, M. T.; VALDURIEZ, P. Data replication. In: _____. *Principles of Distributed Database Systems*. [S.l.]: Springer, 2020. p. 247–280. ISBN 978-3-030-26253-2. Citado 2 vezes nas páginas 24 e 25.

OZSU, M. T.; VALDURIEZ, P. Distributed transaction processing. In: _____. *Principles of Distributed Database Systems*. [S.l.]: Springer, 2020. p. 183–246. ISBN 978-3-030-26253-2. Citado 4 vezes nas páginas 6, 19, 20 e 26.

OZSU, M. T.; VALDURIEZ, P. Introduction. In: _____. *Principles of Distributed Database Systems*. [S.l.]: Springer, 2020. p. 1–31. ISBN 978-3-030-26253-2. Citado 5 vezes nas páginas 12, 14, 15, 16 e 17.

PERCONA. *Introducing tpce-like workload for MySQL*. 2010. Website. Disponível em: <https://github.com/Percona-Lab/tpce-mysql>. Citado na página 31.

PERCONA. *Percona Monitoring and Management Documentation*. Durham, USA, 2020. Disponível em: <https://www.percona.com/doc/percona-monitoring-and-management/index.html>. Citado na página 30.

STRINGHINI, D.; GONÇALVES, A. R.; GOLDMAN, A. Introdução à computação heterogênea. In: *Congresso da Sociedade Brasileira de Computação*. [S.l.]: SBC, 2012. Citado na página 17.

TPC. *TPC BENCHMARK ™ E*. 1. ed. San Francisco, USA, 2010. Disponível em: <http://www.tpc.org/tpce/>. Citado na página 31.

WEI, Z.; PIERRE, G.; CHI, C.-H. Scalable transactions for web applications in the cloud. In: . [S.l.: s.n.], 2009. p. 442–453. Citado na página 24.

WIDENIUS, M. *MySQL-MariaDB History talk*. 2019. Website - PDF. Disponível em: <https://mariadb.org/wp-content/uploads/2019/11/MySQL-MariaDB-story.pdf>. Citado na página 29.

Apêndices

APÊNDICE A – Trechos de logs das transações aplicadas nos cenários de testes

Este apêndice apresenta um trecho dos logs de transações retirados dos servidores MariaDB. Os logs subsequentes foram extraídos utilizando uma ferramenta do SGBD chamada *General Query Log* que armazena todas as transações aplicadas ao servidor, independente de seu tipo (DML ou DDL).

A.1 Primeiro cenário

```
1  SET TRANSACTION ISOLATION LEVEL READ COMMITTED
2  SELECT t_id, DATE_FORMAT(t_dts,'%Y-%m-%d␣%H:%i:%s.%f'), st_name, tt_name
       ↪ , t_s_symb, t_qty, t_exec_name, t_chrg, s_name, ex_name FROM trade
       ↪ , status_type FORCE INDEX(PRIMARY), trade_type FORCE INDEX(PRIMARY
       ↪ ), security, exchange WHERE t_ca_id = 43000003672 AND st_id =
       ↪ t_st_id AND tt_id = t_tt_id AND s_symb = t_s_symb AND ex_id =
       ↪ s_ex_id ORDER BY t_dts DESC LIMIT 50
3  SELECT c_l_name, c_f_name, b_name FROM customer_account, customer,
       ↪ broker WHERE ca_id = 43000009624 AND c_id = ca_c_id AND b_id =
       ↪ ca_b_id
4  COMMIT
5  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
6  SELECT ca_name, ca_b_id, ca_c_id, ca_tax_st FROM customer_account WHERE
       ↪ ca_id = 43000004082
7  SELECT c_f_name, c_l_name, c_tier, c_tax_id FROM customer WHERE c_id =
       ↪ 4300000409
8  SELECT b_name FROM broker WHERE b_id = 4300000002
9  SELECT s_co_id, s_ex_id, s_name FROM security WHERE s_symb = 'CFNL'
10 SELECT co_name FROM company WHERE co_id = 4300000640
11 SELECT lt_price FROM last_trade WHERE lt_s_symb = 'CFNL'
12 SELECT tt_is_mrkt, tt_is_sell FROM trade_type WHERE tt_id = 'TMS'
13 SELECT hs_qty FROM holding_summary WHERE hs_ca_id = 43000004082 AND
       ↪ hs_s_symb = 'CFNL'
14 SELECT h_qty, h_price FROM holding WHERE h_ca_id = 43000004082 AND
       ↪ h_s_symb = 'CFNL' ORDER BY h_dts ASC
```

```
15 SELECT sum(tx_rate) FROM taxrate, customer_taxrate WHERE tx_id =
       ↪ cx_tx_id AND cx_c_id = 4300000409
16 SELECT cr_rate FROM commission_rate WHERE cr_c_tier = 3 AND cr_tt_id = '
       ↪ TMS' AND cr_ex_id = 'NASDAQ' AND cr_from_qty <= 100 AND cr_to_qty
       ↪ >= 100
17 SELECT ch_chrg FROM charge WHERE ch_c_tier = 3 AND ch_tt_id = 'TMS'
18 UPDATE seq_trade_id SET id=LAST_INSERT_ID(id+1)
19 SELECT LAST_INSERT_ID()
20 INSERT INTO trade(t_id, t_dts, t_st_id, t_tt_id, t_is_cash, t_s_symb,
       ↪ t_qty, t_bid_price, t_ca_id, t_exec_name, t_trade_price, t_chrg,
       ↪ t_comm, t_tax, t_lifo) VALUES (200000014544002, '2021-05-17␣
       ↪ 09:49:03', 'SBMT', 'TMS', 1, 'CFNL', 100, 26.28, 43000004082, '
       ↪ Michelle␣Ciesco', NULL, 2.50, 8.41, 0, 0)
21 INSERT INTO trade_history(th_t_id, th_dts, th_st_id) VALUES
       ↪ (200000014544002, '2021-05-17␣09:49:03', 'SBMT')
22 COMMIT
23 set @@sql_select_limit=DEFAULT
24 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
25 SET TRANSACTION ISOLATION LEVEL READ COMMITTED
26 SELECT t_ca_id, t_tt_id, t_s_symb, t_qty, t_chrg, t_lifo, t_is_cash FROM
       ↪  trade WHERE t_id = 200000014544001
27 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
       ↪ FROM trade, trade_type WHERE t_id = 200000006028162 AND t_tt_id =
       ↪ tt_id
28 SELECT tt_name, tt_is_sell, tt_is_mrkt FROM trade_type WHERE tt_id = '
       ↪ TMB'
29 SELECT hs_qty FROM holding_summary WHERE hs_ca_id = 43000011161 AND
       ↪ hs_s_symb = 'ELP'
30 SELECT ca_b_id, ca_c_id, ca_tax_st FROM customer_account WHERE ca_id =
       ↪ 43000011161
31 UPDATE holding_summary SET hs_qty = -5400 WHERE hs_ca_id = 43000011161
       ↪ AND hs_s_symb = 'ELP'
32 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
       ↪ FROM settlement WHERE se_t_id = 200000006028162
33 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
       ↪  cash_transaction WHERE ct_t_id = 200000006028162
34 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
       ↪ trade_history WHERE th_t_id = 200000006028162 ORDER BY th_dts
       ↪ LIMIT 3
```

```
35 SELECT h_t_id, h_qty, h_price FROM holding WHERE h_ca_id = 43000011161
   ↪ AND h_s_symb = 'ELP' ORDER BY h_dts ASC
36 INSERT INTO holding_history(hh_h_t_id, hh_t_id, hh_before_qty,
   ↪ hh_after_qty) VALUES(200000012952143, 200000014544001, -500, 0)
37 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
   ↪ FROM trade, trade_type WHERE t_id = 200000008353972 AND t_tt_id =
   ↪ tt_id
38 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
   ↪ FROM settlement WHERE se_t_id = 200000008353972
39 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
   ↪  cash_transaction WHERE ct_t_id = 200000008353972
40 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
   ↪ trade_history WHERE th_t_id = 200000008353972 ORDER BY th_dts
   ↪ LIMIT 3
41 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
   ↪ FROM trade, trade_type WHERE t_id = 200000007892403 AND t_tt_id =
   ↪ tt_id
42 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
   ↪ FROM settlement WHERE se_t_id = 200000007892403
43 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
   ↪  cash_transaction WHERE ct_t_id = 200000007892403
44 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
   ↪ trade_history WHERE th_t_id = 200000007892403 ORDER BY th_dts
   ↪ LIMIT 3
45 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
   ↪ FROM trade, trade_type WHERE t_id = 200000008317354 AND t_tt_id =
   ↪ tt_id
46 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
   ↪ FROM settlement WHERE se_t_id = 200000008317354
47 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
   ↪  cash_transaction WHERE ct_t_id = 200000008317354
48 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
   ↪ trade_history WHERE th_t_id = 200000008317354 ORDER BY th_dts
   ↪ LIMIT 3
49 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
   ↪ FROM trade, trade_type WHERE t_id = 200000014509726 AND t_tt_id =
   ↪ tt_id
50 DELETE FROM holding WHERE h_t_id = 200000012952143
```

```
51 INSERT INTO holding_history(hh_h_t_id, hh_t_id, hh_before_qty,
    ↪ hh_after_qty) VALUES(200000012997201, 200000014544001, -100, 0)
52 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
    ↪ FROM settlement WHERE se_t_id = 200000014509726
53 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
    ↪  cash_transaction WHERE ct_t_id = 200000014509726
54 DELETE FROM holding WHERE h_t_id = 200000012997201
55 INSERT INTO holding_history(hh_h_t_id, hh_t_id, hh_before_qty,
    ↪ hh_after_qty) VALUES(200000013054351, 200000014544001, -200, 0)
56 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
    ↪ trade_history WHERE th_t_id = 200000014509726 ORDER BY th_dts
    ↪ LIMIT 3
57 SELECT t_bid_price, t_exec_name, t_is_cash, tt_is_mrkt, t_trade_price
    ↪ FROM trade, trade_type WHERE t_id = 200000013860721 AND t_tt_id =
    ↪ tt_id
58 DELETE FROM holding WHERE h_t_id = 200000013054351
59 SELECT s_ex_id, s_name FROM security WHERE s_symb = 'ELP'
60 SELECT c_tier FROM customer WHERE c_id = 4300001117
61 SELECT cr_rate FROM commission_rate WHERE cr_c_tier = 2 AND cr_tt_id = '
    ↪ TMB' AND cr_ex_id = 'NASDAQ' AND cr_from_qty <= 800 AND cr_to_qty
    ↪ >= 800
62 UPDATE trade SET t_comm = 65.16, t_dts = '2021-05-17␣09:49:03.000000',
    ↪ t_st_id = 'CMPT', t_trade_price = 29.09 WHERE t_id =
    ↪ 200000014544001
63 INSERT INTO trade_history(th_t_id, th_dts, th_st_id) VALUES
    ↪ (200000014544001, '2021-05-17␣09:49:03.000000', 'CMPT')
64 UPDATE broker SET b_comm_total = b_comm_total + 65.16, b_num_trades =
    ↪ b_num_trades + 1 WHERE b_id = 4300000013
65 INSERT INTO settlement(se_t_id, se_cash_type, se_cash_due_date, se_amt)
    ↪ VALUES(200000014544001, 'Margin', '2021-05-19', -23341.7)
66 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
    ↪ FROM settlement WHERE se_t_id = 200000013860721
67 SELECT ca_bal FROM customer_account WHERE ca_id = 43000011161
68 COMMIT
```

A.2 Segundo cenário

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
2 SELECT ca_name, ca_b_id, ca_c_id, ca_tax_st FROM customer_account WHERE
    ↪ ca_id = 43000000531
```

```
3  SELECT c_f_name, c_l_name, c_tier, c_tax_id FROM customer WHERE c_id =
       ↪ 4300000054
4  SELECT b_name FROM broker WHERE b_id = 4300000009
5  SELECT s_co_id, s_ex_id, s_name FROM security WHERE s_symb = 'ABER'
6  SELECT co_name FROM company WHERE co_id = 4300000457
7  SELECT lt_price FROM last_trade WHERE lt_s_symb = 'ABER'
8  SELECT tt_is_mrkt, tt_is_sell FROM trade_type WHERE tt_id = 'TMB'
9  SELECT hs_qty FROM holding_summary WHERE hs_ca_id = 43000000531 AND
       ↪ hs_s_symb = 'ABER'
10 SELECT cr_rate FROM commission_rate WHERE cr_c_tier = 2 AND cr_tt_id = '
       ↪ TMB' AND cr_ex_id = 'PCX' AND cr_from_qty <= 100 AND cr_to_qty >=
       ↪ 100
11 SELECT ch_chrg FROM charge WHERE ch_c_tier = 2 AND ch_tt_id = 'TMB'
12 UPDATE seq_trade_id SET id=LAST_INSERT_ID(id+1)
13 SELECT LAST_INSERT_ID()
14 INSERT INTO trade(t_id, t_dts, t_st_id, t_tt_id, t_is_cash, t_s_symb,
       ↪ t_qty, t_bid_price, t_ca_id, t_exec_name, t_trade_price, t_chrg,
       ↪ t_comm, t_tax, t_lifo) VALUES (200000014598741, '2021-05-17␣
       ↪ 21:55:29', 'SBMT', 'TMB', 1, 'ABER', 100, 24.20, 43000000531, '
       ↪ Jacquie␣Gionest', NULL, 4.50, 10.65, 0, 1)
15 INSERT INTO trade_history(th_t_id, th_dts, th_st_id) VALUES
       ↪ (200000014598741, '2021-05-17␣21:55:29', 'SBMT')
16 COMMIT
17 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
       ↪ FROM settlement WHERE se_t_id = 200000000781245
18 SELECT ct_name FROM cash_transaction WHERE ct_t_id = 200000000781245
19 UPDATE cash_transaction SET ct_name = 'Market-Sell␣400␣Shares␣of␣COMMON␣
       ↪ of␣Bay␣View␣Capital␣Corporation' WHERE ct_t_id = 200000000781245
20 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
       ↪  cash_transaction WHERE ct_t_id = 200000000781245
21 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
       ↪ trade_history WHERE th_t_id = 200000000781245 ORDER BY th_dts ASC
       ↪ LIMIT 3
22 SELECT se_amt, DATE_FORMAT(se_cash_due_date, '%Y-%m-%d'), se_cash_type
       ↪ FROM settlement WHERE se_t_id = 200000008053877
23 SELECT ct_name FROM cash_transaction WHERE ct_t_id = 200000008053877
24 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
25 SELECT co_name, in_name FROM company_competitor, company, industry WHERE
       ↪  cp_co_id = 4300000037 AND co_id = cp_comp_co_id AND in_id =
```

```
    ↪ cp_in_id LIMIT 3
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SELECT ca_name, ca_b_id, ca_c_id, ca_tax_st FROM customer_account WHERE
    ↪ ca_id = 43000018541
SELECT t_ca_id, t_tt_id, t_s_symb, t_qty, t_chrg, t_lifo, t_is_cash FROM
    ↪  trade WHERE t_id = 200000014598732
SELECT c_f_name, c_l_name, c_tier, c_tax_id FROM customer WHERE c_id =
    ↪ 4300001855
SELECT tt_name, tt_is_sell, tt_is_mrkt FROM trade_type WHERE tt_id = '
    ↪ TMB'
SELECT b_name FROM broker WHERE b_id = 4300000019
SELECT hs_qty FROM holding_summary WHERE hs_ca_id = 43000005652 AND
    ↪ hs_s_symb = 'ALOT'
SELECT s_co_id, s_ex_id, s_name FROM security WHERE s_symb = 'BLDP'
SELECT ca_b_id, ca_c_id, ca_tax_st FROM customer_account WHERE ca_id =
    ↪ 43000005652
SELECT co_name FROM company WHERE co_id = 4300000499
UPDATE holding_summary SET hs_qty = 5600 WHERE hs_ca_id = 43000005652
    ↪ AND hs_s_symb = 'ALOT'
SELECT lt_price FROM last_trade WHERE lt_s_symb = 'BLDP'
SELECT tt_is_mrkt, tt_is_sell FROM trade_type WHERE tt_id = 'TLB'
INSERT INTO holding_history(hh_h_t_id, hh_t_id, hh_before_qty,
    ↪ hh_after_qty) VALUES(200000014598732, 200000014598732, 0, 200)
SELECT hs_qty FROM holding_summary WHERE hs_ca_id = 43000018541 AND
    ↪ hs_s_symb = 'BLDP'
INSERT INTO holding(h_t_id, h_ca_id, h_s_symb, h_dts, h_price, h_qty)
    ↪ VALUES(200000014598732, 43000005652, 'ALOT', '2021-05-17␣21:55:29'
    ↪ , 25.73, 200)
SELECT cr_rate FROM commission_rate WHERE cr_c_tier = 1 AND cr_tt_id = '
    ↪ TLB' AND cr_ex_id = 'AMEX' AND cr_from_qty <= 800 AND cr_to_qty >=
    ↪  800
SELECT s_ex_id, s_name FROM security WHERE s_symb = 'ALOT'
SELECT c_tier FROM customer WHERE c_id = 4300000566
SELECT ch_chrg FROM charge WHERE ch_c_tier = 1 AND ch_tt_id = 'TLB'
SELECT cr_rate FROM commission_rate WHERE cr_c_tier = 1 AND cr_tt_id = '
    ↪ TMB' AND cr_ex_id = 'AMEX' AND cr_from_qty <= 200 AND cr_to_qty >=
    ↪  200
UPDATE seq_trade_id SET id=LAST_INSERT_ID(id+1)
SELECT LAST_INSERT_ID()
```

```
49 UPDATE trade SET t_comm = 20.58, t_dts = '2021-05-17␣21:55:29.000000',
   ↪ t_st_id = 'CMPT', t_trade_price = 25.73 WHERE t_id =
   ↪ 200000014598732
50 INSERT INTO trade(t_id, t_dts, t_st_id, t_tt_id, t_is_cash, t_s_symb,
   ↪ t_qty, t_bid_price, t_ca_id, t_exec_name, t_trade_price, t_chrg,
   ↪ t_comm, t_tax, t_lifo) VALUES (200000014598742, '2021-05-17␣
   ↪ 21:55:29', 'PNDG', 'TLB', 1, 'BLDP', 800, 26.14, 43000018541, '
   ↪ Ricardo␣Scavone', NULL, 6.00, 66.92, 0, 0)
51 INSERT INTO trade_history(th_t_id, th_dts, th_st_id) VALUES
   ↪ (200000014598732, '2021-05-17␣21:55:29.000000', 'CMPT')
52 INSERT INTO trade_request(tr_t_id, tr_tt_id, tr_s_symb, tr_qty,
   ↪ tr_bid_price, tr_b_id) VALUES (200000014598742, 'TLB', 'BLDP',
   ↪ 800, 26.14, 4300000019)
53 UPDATE broker SET b_comm_total = b_comm_total + 20.58, b_num_trades =
   ↪ b_num_trades + 1 WHERE b_id = 4300000003
54 INSERT INTO trade_history(th_t_id, th_dts, th_st_id) VALUES
   ↪ (200000014598742, '2021-05-17␣21:55:29', 'PNDG')
55 COMMIT
56 INSERT INTO settlement(se_t_id, se_cash_type, se_cash_due_date, se_amt)
   ↪ VALUES(200000014598732, 'Cash␣Account', '2021-05-19', -5171.58)
57 UPDATE customer_account SET ca_bal = ca_bal + -5171.58 WHERE ca_id =
   ↪ 43000005652
58 INSERT INTO cash_transaction(ct_dts, ct_t_id, ct_amt, ct_name) VALUES('
   ↪ 2021-05-17␣21:55:29.000000', 200000014598732, -5171.58, 'Market-
   ↪ Buy␣200␣shared␣of␣COMMON␣of␣Astro-Med,␣Inc.')
59 SELECT ca_bal FROM customer_account WHERE ca_id = 43000005652
60 COMMIT
61 SET TRANSACTION ISOLATION LEVEL READ COMMITTED
62 SELECT t_id, DATE_FORMAT(t_dts,'%Y-%m-%d␣%H:%i:%s.%f'), st_name, tt_name
   ↪ , t_s_symb, t_qty, t_exec_name, t_chrg, s_name, ex_name FROM trade
   ↪ , status_type FORCE INDEX(PRIMARY), trade_type FORCE INDEX(PRIMARY
   ↪ ), security, exchange WHERE t_ca_id = 43000016723 AND st_id =
   ↪ t_st_id AND tt_id = t_tt_id AND s_symb = t_s_symb AND ex_id =
   ↪ s_ex_id ORDER BY t_dts DESC LIMIT 50
63 SELECT fi_year, fi_qtr, DATE_FORMAT(fi_qtr_start_date,'%Y-%m-%d'),
   ↪ fi_revenue, fi_net_earn, fi_basic_eps, fi_dilut_eps, fi_margin,
   ↪ fi_inventory, fi_assets, fi_liability, fi_out_basic, fi_out_dilut
   ↪ FROM financial WHERE fi_co_id = 4300000037 ORDER BY fi_year ASC,
   ↪ fi_qtr LIMIT 20
```

```
64 UPDATE cash_transaction SET ct_name = 'Market-Buy␣800␣Shares␣of␣COMMON␣
       ↪ of␣Bay␣View␣Capital␣Corporation' WHERE ct_t_id = 200000008053877
65 SELECT ct_amt, DATE_FORMAT(ct_dts, '%Y-%m-%d␣%H:%i:%s.%f'), ct_name FROM
       ↪  cash_transaction WHERE ct_t_id = 200000008053877
66 SELECT DATE_FORMAT(th_dts, '%Y-%m-%d␣%H:%i:%s.%f'), th_st_id FROM
       ↪ trade_history WHERE th_t_id = 200000008053877 ORDER BY th_dts ASC
       ↪ LIMIT 3
67 COMMIT
```