UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JAVAN KAUÊ TAVARES LACERDA

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS PARA ARMAZENAMENTO DE SÉRIES TEMPORAIS

CAMPINA GRANDE - PB

2021

JAVAN KAUÊ TAVARES LACERDA

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS PARA ARMAZENAMENTO DE SÉRIES TEMPORAIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Andrey Brito.

CAMPINA GRANDE - PB

2021

L131a Lacerda, Javan Kauê Tavares.
Avaliação de desempenho de bancos de dados para armazenamento de séries temporais / Javan Kauê Tavares Lacerda. – 2021.

10 f.

Orientador: Prof. Dr. Andrey Elísio Monteiro Brito.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Bancos de dados - desempenho. 2. Armazenamento de series temporais. 3. Eficiência energética. 4. Sistema MariaDB. 5. Sistema InfluxDB. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

JAVAN KAUÊ TAVARES LACERDA

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS PARA ARMAZENAMENTO DE SÉRIES TEMPORAIS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Andrey Elísio Monteiro Brito

Orientador – UASC/CEEI/UFCG

Professor Dr. Fábio Jorge Almeida Morais

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de Outubro de 2021.

CAMPINA GRANDE - PB

ABSTRACT

This work aims to evaluate the performance of some database solutions applied to the storage of time series data in the context of an energy efficiency support system. In this evaluation we experimentally compared two systems: MariaDB and InfluxDB. MariaDB is a general purpose database, and InfluxDB is a specific database for time series data. The results obtained in our experiments indicate that MariaDB, even not being specific for time series, is able to supply the demands, presenting a superior performance than InfluxDB.

Avaliação de desempenho de bancos de dados para armazenamento de séries temporais

Javan Kauê Tavares Lacerda
javan.lacerda@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Andrey Brito
andrey@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Thiago Emannuel Pereira
temmanuel@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Este trabalho busca avaliar o desempenho de algumas soluções de bancos de dados aplicadas para o armazenamento de séries de dados temporais no contexto de um sistema de apoio à eficiência energética. Nesta avaliação comparamos experimentalmente dois sistemas: MariaDB e InfluxDB. Sendo o MariaDB um banco de dados de propósito geral, e o InfluxDB um banco especifico para armazenamento de séries temporais. Os resultados obtidos em nossos experimentos indicam que o MariaDB, mesmo não sendo especifico para séries temporais, consegue suprir bem as demandas, apresentando desempenho superior ao InfluxDB.

PALAVRAS-CHAVE

Eficiência energética, MariaDB, InfluxDB, Séries temporais.

1 INTRODUÇÃO

Um conceito bastante emergente voltado à tecnologia da informação é a internet das coisas [6]. Ela se refere à interconexão digital de objetos cotidianos através da internet. Entre suas aplicações, uma bem popular são as casas inteligentes [7], que é a automatização de rotinas de uma casa, controlando dispositivos domésticos remotamente através de comandos, podendo ser por voz ou algum aplicativo. Mas a casas inteligentes não se resumem apenas em ativar ou desativar dispositivos, pode haver também todo um sistema de monitoramento, que pode ter finalidade voltada para segurança, mas também para análise de rotinas e detecção de possíveis melhorias no uso.

Estas métricas são colhidas regularmente por agentes e enviadas para um concentrador, o qual persiste estas informações em um banco dados, para que iniciativas sejam tomadas posteriormente. Uma das informações presente é o momento em que elas foram capturadas, o que as configura como séries temporais [2].

Este banco dados tem significante importância no monitoramento de um sistema de internet das coisas, pois ele funciona como referencia, onde todas informações são concentradas, exigindo robustez para aguentar grande quantidade de escritas e eventuais leituras.

Existem bancos de dados com propósitos generalistas e existem outros bem específicos. O MariaDB é um dos que se configura como banco de dados de propósito geral, que tende a ter um conjunto de algoritmos preparados para suprir diferentes demandas de forma estável. Diferentemente, o InfluxDB é um banco de dados específico para séries temporais, onde toda sua arquitetura foi traçada para resolver este problema especifico.

O fato de poder ser aplicado em diferentes contextos faz que o MariaDB seja bastante popular e de fácil manutenção. Isto faz com que os profissionais de tecnologia da informação busquem aprender sobre, pois é um conhecimento que vai agregar bastante, por sua aplicabilidade no mercado. O InfluxDB não tem a facilidade de manutenção do MariaDB, isto deve-se pela sua especificidade em resolver apenas problemas com dados de séries temporais, entretanto, é esperado que tenha desempenho superior quando aplicado a este tipo de problema.

Este trabalho teve como objetivo avaliar estes bancos de dados na sua capacidade de suprir demandas de dados de séries temporais. Para tal, desenvolvemos um micro-benchmark que conseguiu testa-los forma personalizada, principalmente a capacidade de escrita, pois buscávamos além de testar suas performances, verificar também a elegibilidade para encaixar no fluxo de dados do LiteMe [1]. O LiteMe é um sistema de apoio à eficiência energética que age a partir do monitoramento do uso de energia elétrica.

Este trabalho foi importante para prover embasamento para próximos possíveis arquitetos de software que venham precisar tomar decisões similares no futuro.

O benchmark foi desenvolvido usando Python, de forma que fazia teste de carga nos bancos de dados. Foi usado o mesmo ambiente genérico para ambos, pois o objetivo deste trabalho foi fazer uma comparação direta dos mesmos. O MariaDB conseguiu ter um desempenho muito superior ao do InfluxDB, mesmo usando uma carga que é especialidade do InfluxDB.

Na Seção 2 foram levantadas algumas tecnologias que foram abordadas no trabalho.

Na Seção 3 foi descrito a forma em que os experimentos foram conduzidos, incluindo o planejamento e desenvolvimento do benchmark, o ambiente de execução, validações e a execução em si.

Na Seção 4 foram descritos os resultados apurados durante os experimentos.

Na Seção 5 foram levantadas algumas validações que fizemos no ambiente para garantir a consistência e confiabilidade nos resultados.

Na Seção 6 foram descritas algumas dificuldades encontradas no desenvolvimento no trabalho e a forma em que elas foram contornadas.

Na Seção 7 foram analisados os resultados, respondendo se os objetivos foram alcançados e levantados possíveis melhoras em trabalhos futuros.

2 *BACKGROUND*

Após fazermos um apanhado das principais tecnologias que poderiam ser possíveis soluções, listamos as tecnologias mais relevantes para a nossa avaliação.

2.1 LiteMe

O LiteMe é que um projeto que visa apoio à eficiência energética, e a estratégia para cumprir seu objetivo é a coleta dados e análise do uso de energia, como mostrado na figura 1. Hoje o LiteMe atua validando o uso de energia de casas, fábricas, universidades, entre outros.

2.2 Bancos de dados

2.2.1 MariaDB [4]. É um banco de dados de propósito geral bastante popular [5]. Duas características principais contribuíram para essa popularidade: ter código aberto, e ser baseado no também popular MySQL [5]. Ele é o principal banco de dados de alguns sistemas bem conhecidos como OpenStack, Google, Mozilla e Wikimedia Foundation [5]. Sistemas robustos tendem a adotá-lo principalmente por sua escalabilidade. O MariaDB Foundation disponibliza também alguns gerenciadores de escalabilidade como o Galera [8] e MaxScale [9], que permitem sua escalabilidade horizontal. Também são disponibilizados alguns motores de armazenamento [10] que podem flexibilizar o seu uso, permitindo que siga diferentes estratégias de armazenamento de forma dinâmica. Estes motores podem ser instalados em forma de *plugins*, tornando possível a presença de diversos uma mesma instância do banco, definindo sua aplicação na criação das tabelas. Alguns dos principais motores de armazenamento são descritos a seguir.

- InnoDB [12] é o motor de armazenamento padrão do MariaDB e também do MySQL. Ele tem caráter generalista, e visa atender de forma solida todo tipo de demanda. Seguindo o seu propósito, ele tende a ser estável em todas usabilidades propostas e boa adaptação com escalabilidade horizontal.
- MyRocks [13] é um motor de armazenamento baseado no banco de dados RocksDB [14], que foi originalmente desenvolvido pelo Facebook. Seu objetivo principal é a performance em escritas, fazendo uso de algumas técnicas para aumentar sua vazão, como o enfileiramento dos dados em memória. Esse seu objetivo casa bem com o que se espera de um banco de dados para séries temporais, pois o padrão é que ocorram muitas escritas.

2.2.2 InfluxDB [11]. É um banco de dados de código aberto, também muito popular, desenvolvido especificamente para séries temporais. O banco foi escrito na linguagem Golang, e tem uma comunidade grande e participativa. O repositório no github tem mais de 21 mil estrelas e mais de 3 mil *forks*.

3 METODOLOGIA

Para o desenvolvimento o experimento, foi necessário executar alguns passos, os quais são descritos a seguir.

3.1 O Benchmark

Inicialmente foi necessário planejar o que iriamos testar, definindo as métricas de interesse, e como escrever um benckmark que conseguisse mudar dinamicamente o banco de dados alvo, pois tínhamos como objetivo testar diferentes configurações.

3.1.1 Séries temporais. Quando tratando-se de séries temporais, é possível observar um comportamento bem comum relacionado a banco de dados: Os dados são escritos de forma granular mas com constância, e sua leitura é feita de forma esporádica mas em grandes quantidades. Trazendo esta ideia para o fluxo do LiteMe, é compreensível esse comportamento quando nos atentamos a forma que o dado é abordado, pois o dado é coletado por múltiplos sensores de energia e de forma regular, e suas leituras ocorrem não muito frequentemente, mas são quantidades consideráveis e de forma sequencial, pois o principal consumidor deste dado é a aplicação de monitoramento, que os ler e mostra em forma de gráficos.

3.1.2 Código. Foi escrito na linguagem Python de forma modularizada, onde cada modulo era responsável por uma tarefa definida, tais como: Comunicação com o MariaDB usando a biblioteca disponível pela linguagem, comunicação com o InfluxDB também usando a biblioteca disponível, e o agente de monitoramento, que era responsável por colher e persistir métricas regularmente. Para simular o paralelismo dos múltiplos sensores enviando informação, foi usada biblioteca *multiprocessing* [15] disponibilizada pela linguagem

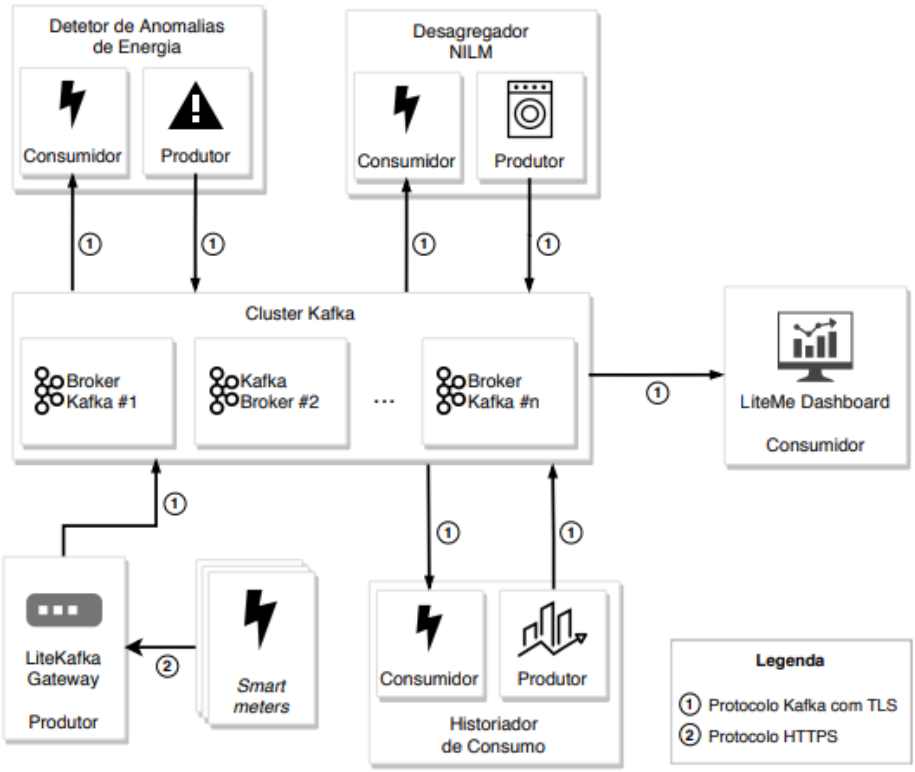

Figura 1: Arquitetura do LiteMe a qual queremos eleger o serviço para historiador de consumo

Python, a qual permite criar diversos processos executando as escritas, aumentando a carga.

3.2 Ambiente

Visando ter um ambiente de execução controlado e imparcial, foram usadas maquinas virtuais disponibilizadas pelo Laboratório de Sistemas Distribuídos, provenientes do seu serviço de nuvem privada. Foram usadas o total de duas máquinas virtuais para rodar o experimento, uma para executar os bancos de dados e outra para o benchmark. O ambiente usado não é representativo em relação ao ambiente de produção do LiteMe, pois o objetivo principal deste trabalho é testar o desempenho dos bancos de dados usando uma carga similar ao de produção mas em um ambiente genérico. Ambas máquinas virtuais tiveram a mesma configuração, que é listada a seguir.

- CPU: 8 núcleos
- Memória ram: 16GB
- Rede: 10GiB

As aplicações dos bancos foram executadas de forma conteinerizada usando Docker [16] e as imagens oficiais do MariaDB e do InfluxDB disponíveis no Dockerhub.

3.3 Execução

Durante a execução, o benchmark fez testes de estresse de escrita, alternando com eventuais leituras.

A carga utilizada foi um texto de exatamente 1KB, visando criar um fluxo similar ao do LiteMe.

As configurações escolhidas para nosso experimento foram as seguintes.

(1) 3 execuções de cada cenário
(2) 1 hora cada execução, ignorando os primeiros 10 minutos (tempo de aquecimento do ambiente)
(3) 50 processos paralelos escrevendo continuamente
(4) 1 processo fazendo leitura das ultimas 10000 linhas da tabela a cada 10 segundos

4 RESULTADOS

4.0.1 MariaDB com InnoBD. Apresentou comportamento estável durante suas execuções, e taxas de 8773 escritas por segundo (e/s) na mínima, 8765e/s no 1º quarter, 8861e/s de mediana, 8893.25e/s no 3º quarter, 8904 na máxima e com a taxa média de 8,846e/s, como mostrado na figura 2.

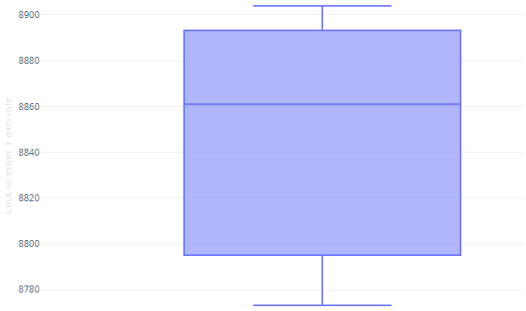

Figura 2: Taxas de escritas por segundo do MariaDB com InnoDB

4.0.2 MariaDB com MyRocks. Com a configuração forçando a escrita no disco a cada mil linhas ele conseguiu alcançar taxas de 18550e/s na mínima, 18810e/s no 1º quarter, 19593e/s de mediana, 19893e/s no 3º quarter, 19994 na máxima e com a taxa média de 19379e/s, como mostrado na figura 3.

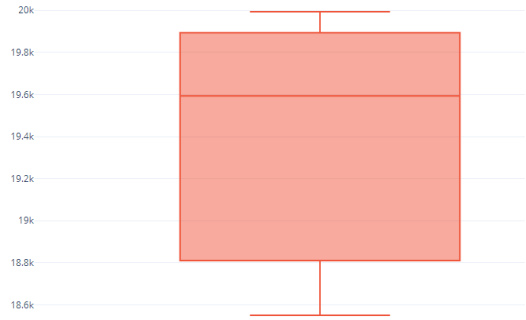

Figura 3: Taxas de escritas por segundo do MariaDB com MyRocks

4.0.3 InfluxDB. O InfluxDB com as configurações padrão, teve o comportamento estável apresentando taxas de 2379e/s na mínima, 2381.5e/s no 1º quarter, 2388e/s de mediana, 2415e/s no 3º quarter, 2424 na máxima e com a taxa média de 2397e/s, como mostrado na figura 4.

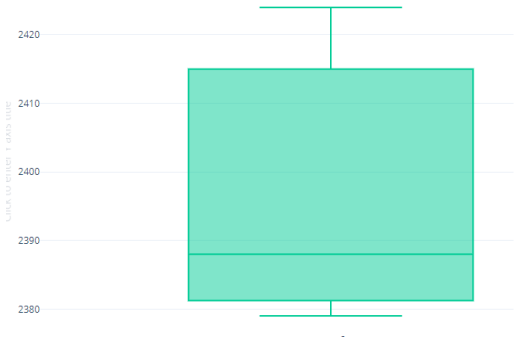

Figura 4: Taxas de escritas por segundo do InfluxDB

5 AMEAÇAS À VALIDADE

Validações na rede foram executadas, para garantir que não haveriam gargalos inesperados na comunicação entre o benchmark e os bancos de dados. Isto foi feito através do monitoramento de banda e latência da conexão entre as duas máquinas virtuais usando o TCPTrack.

Para garantir que o estresse estaria ocorrendo do lado do banco e não do cliente, foi realizado o monitoramento do uso de CPU e da Memória usando o Htop durante as execuções.

Outras possíveis ameaças, mas que não foram testadas, são as possíveis interferências na rede pelo uso máquinas providas por serviços de nuvem. Por praticamente tudo ser virtualizado, alguns detalhes podem ser negligenciados mas que fazem diferença, por exemplo, a comunicação entre a máquina virtual e o disco. Provedores de nuvem tendem a ter um serviço dedicado para o sistema de arquivos das máquinas virtuais, entre estes serviços, temos o Ceph [17]. Isto faz que a comunicação não ocorra diretamente entre a máquina e o disco, mas sim via rede, criando um *overhead* facilmente negligenciado.

6 LIÇÕES APRENDIDAS

Durante o processo de desenvolvimento do trabalho, surgiram alguns desafios que por fim tornaram-se aprendizado.

6.1 Problemas de execução

Um comportamento padrão do Myrocks é o uso máximo de memória RAM para enfileiramento dos dados, buscando performance na escrita. A documentação da Percona [18] alerta sobre a necessidade de ter uma configuração resiliente quando usando-o, pois pode haver eventuais usos completos da memória, fazendo que o processo do banco seja interrompido pelo gerenciador de memória do sistema operacional. A primeira iniciativa para resolver este problema, foi criar uma memória virtual para a maquina. Esta memória virtual funciona como uma memória segundaria, que é alocada no disco, a qual complementa memória RAM principal, tornando-se uma extensão da mesma. Mas isto causou um comportamento inesperado, onde o inicio da execução tinha o desempenho muito melhor que o meio e fim. A explicação para este comportamento é que no seu começo o dado estava sendo escrito somente na memória RAM, mas quando passou a escrever na memória virtual, o desempenho caiu drasticamente, por que a velocidade de escrita do disco é extremamente inferior a da memória. Então em vez de haver um gerenciamento nas escritas usando a memória RAM para enfileirar os dados e eventualmente passar os dados para o disco, estava havendo apenas duas etapas distintas, cem por cento de escritas em memoria no inicio e depois cem por cento de escritas em disco no final. Decidimos então remover

a memória virtual, pois no nosso contexto, ela estaria apenas mascarando o problema e não resolvendo.

Depois de pesquisarmos sobre as configurações do MyRocks, conseguimos resolver o problema configurando o motor, forcando-o a passar as informações da memória para o disco a cada N linhas enfileiradas. Isto foi possível configurando a variável rocksdb_bulk_load=1 que vem nativamente desativada (rocksdb_bulk_load=0). Isto fez que a cada 1000 linhas escritas, onde 1000 é o valor padrão, o banco escrevesse o dado que estava enfileirado no disco e liberando o espaço na memória.

6.2 Escrevendo o benchmark

Durante o processo de desenvolvimento do benchmark, nos deparamos com dois problemas principais, que quando solucionados, trouxeram desempenho e aprendizado.

6.2.1 Threads vs Processos. Para criar o paralelismo na execução, inicialmente, foi usada a biblioteca threading de Python, a qual permite criar threads, mas quando foi percebido que independente do numero de threads que fosse criada, o uso de CPU tanto da maquina do banco de dados quanto da maquina em que o benchmark executava mantinha-se o mesmo. Foi quando descobrimos que a forma que thread é implementada pela biblioteca threading não cria paralelismo de verdade, mas sim uma alternância dentro do mesmo processo, tendo afinal uma única thread. A solução para o problema foi usar a biblioteca multiprocessing em vez da biblioteca threading. Esta biblioteca permite criar processos diretamente com o sistema operacional em vez de threads. Após esta mudança o desempenho cresceu significativamente, tornando possível estressar ambas as máquinas.

6.2.2 Remoção de logging do caminho crítico. Foi implementado no benchmark a funcionalidade de logar informações das consultas, tais como latência e número de consultas executadas no ultimo segundo. Inicialmente, cada processo guardava essas informações em memória e a cada dez segundos enviava essa informação para um InfluxDB que fornecia esses dados para um Grafana [19], onde mostrava o gráfico da execução em tempo real. O problema foi na questão de cada processo ser responsável por enviar sua informação, pois a cada dez segundos, o processo tinha que parar de estressar o banco para logar, o que causou inconsistência no benchmark. Após observar essa falha, partimos para a implementação da versão otimizada, a qual foi desenvolver um módulo responsável por *logging*, o qual rodava em um processo independente. Foi criado um Array compartilhado onde cada processo de execução ia salvando suas informações, e o processo responsável por logar sabia ler essa informação e enviá-la para o InfluxDB. Isto tirou a necessidade dos processos de execução pararem para logar, tornando o benchmark consistente.

7 CONCLUSÃO

Este trabalho examinou a possibilidade de um banco de dados de propósito geral suprir demandas de dados de séries temporais e sua elegibilidade em tornar-se o banco de dados principal do LiteMe. A forma de medir a capacidade dos bancos de dados de propósito geral em suprir demandas de dados de séries temporais foi comparando a performance fazendo escritas com o InfluxDB, que é o banco de dados mais popular para esse propósito. E a forma de decidir qual banco seria o melhor candidato a se tornar o banco de dados do LiteMe foi escolhendo o com melhor resultado no benchmark.

7.1 Análise

Os resultados apresentados não foram os esperados mas foram extremamente satisfatórios, pois esperava-se que o InfluxDB por ser um banco de dados desenvolvido para um ecossistema onde há bastante escritas, que são as séries temporais, tivesse um desempenho superior aos outros de propósito geral, neste cenário. Mas foi satisfatório pelo fato de encontrarmos uma solução genérica que consegue ter melhor desempenho que a solução especifica.

O InfluxDB foi testado na versão padrão, sem nenhuma configuração adicional. Talvez seja possível otimizá-lo, mas não era o objetivo deste trabalho. Ele também disponibiliza a funcionalidade de fazer escritas em lotes, o que pode ter um desempenho melhor, mas que não se aplica ao nosso tipo de carga. Os resultados apresentados pelo MariaDB com ambos motores de armazenamento confirma que é sim possível suprir bem demandas de dados de séries temporais.

A configuração de banco de dados definida como a melhor opção para o LiteMe foi o MariaDB com o MyRocks. O benchmark foi desenvolvido com este propósito, usando carga similar à presente no seu fluxo, e os resultados, comparados na figura 5, foram bem conclusivos sobre qual configuração resolve melhor o problema. O número de clientes do LiteMe está crescendo, e capacidade de suportar grande quantidade de escritas, fez que esta configuração seja eleita a melhor para a situação.

7.2 Trabalhos futuros

Este trabalho abriu um leque de possíveis melhorias e ampliações. Entre elas, a contribuição do código do benchmark pra que ele fique mais completo e consiga testar diferentes bancos de dados. Outra possível iniciativa é testar o MariaDB com o MyRocks em um ambiente de *Staging* do LiteMe, usando cargas reais e num ambiente de fato representativo.

8 AGRADECIMENTOS

Agradeço primeiramente a Deus, que por graça me salvou e que por amor, permitiu que eu terminasse essa graduação,

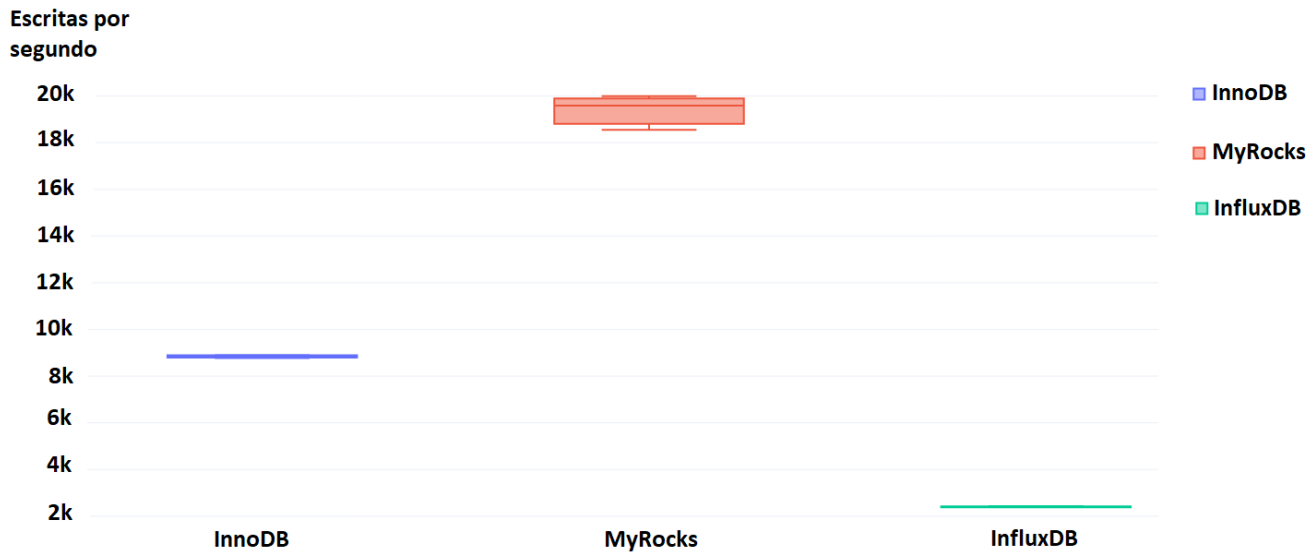

Figura 5: Comparativo de desempenho entre os bancos dados o qual o MyRocks sobressaiu

que seja tudo para a sua glória. À minha amada esposa Milena, que acreditou em mim, confiou e deu suporte durante toda a caminhada. À minha família por toda a base que proveram para que eu pudesse chegar até aqui. Em particular, à minha mãe Jacira Tavares, por ter servido de inspiração, transparecendo a importância da dedicação e perseverança com os estudos. Ao meu pai Marcos Arruda por ter sido espelho, mostrando a importância de manter-se integro, honesto e ético independente da situação. E a minha irmã Samira Lacerda por ter me ensinado a importância de respeitar as diferenças, e crescer como ser humano a partir delas.

Agradeço também aos meus colegas de turma, apresentados nas pessoas de Caio Lira, Felipe Mota, Ícaro Dantas, Igor Farias, Joeberth Souza, Lucas Victor, Marcos Barros e Paulo Felipe por toda amizade, paciência, cafés no quiosque regados de boas conversas, e cooperação nas disciplinas durante toda a graduação. A todos meus colegas de trabalho do Laboratório de Sistemas Distribuídos, por terem de alguma forma me ensinado algo. Em particular a Kaio Oliveira e Icaro Vasconcelos, por terem sido além de colegas de trabalho, amigos, ajudando-me em tomadas de decisões importantes na minha vida. Ao Prof. Dr. Thiago Emanuel pelos conselhos e por ter me guiado e orientado durante o desenvolvimento deste trabalho. E ao Prof. Dr. Andrey Brito não só por ter me orientado neste trabalho, mas por ter me permitido trabalhar com ele, acreditando no meu potencial e guiando o meu desenvolvimento profissional.

REFERENCIAS

[1] Página oficinal do LiteMe, acessado dia 27 de Setembro de 2021, https://www.liteme.com.br/about.
[2] Philippe Esling, Carlos Agon. 2012, Time-series data mining.
[3] George F. Coulouris, Jean Dollimore, Tim Kindberg, 2005, Distributed Systems, Concepts and Design.
[4] Documentação oficial do MariaDB, acessado dia 27 de Setembro de 2021, https://mariadb.org/documentation.
[5] Sadissa Babeni, 24 de janeiro de 2020, Most Popular Databases in 2020: Here's How They Stack Up, https://ormuco.com/blog/most-popular-databases
[6] Kevin Asthon, 22 de Junho de 2009, That "Internet of Things"thing, RFID JOURNAL.
[7] Koen Kok, Stamatis Karnouskos et al. 22 de Setembro de 2009, Smart houses for a smart grid, IEEE Xplore.
[8] Documentação oficial do MariaDB, acessado dia 2 de Outubro de 2021, https://mariadb.com/kb/en/what-is-mariadb-galera-cluster/.
[9] Documentação oficial do MariaDB, MariaDB MaxScale, acessado dia 2 de Outubro de 2021, https://mariadb.com/resources/datasheets/mariadb-maxscale/.
[10] Documentação oficial do MariaDB, Storage Engines, acessado dia 2 de Outubro de 2021 , https://mariadb.com/kb/en/storage-engines/.
[11] Documentação oficial do InfluxDB, acessado dia 2 de Outubro de 2021, https://www.influxdata.com/products/influxdb/.
[12] Documentação oficial do MariaDB, acessado dia 2 de Outubro de 2021, https://mariadb.com/kb/en/innodb/.
[13] Documentação oficial do MariaDB, acessado dia 2 de Outubro de 2021, https://mariadb.com/kb/en/myrocks/.
[14] Site oficial do RocksDB, acessado dia 2 de Outubro de 2021, http://rocksdb.org/.
[15] Documentação oficial de Python, acessado dia 02 de Outubro de 2021, https://docs.python.org/3/library/multiprocessing.html/.
[16] Site oficial do Docker, acessado dia 02 de Outubro de 2021, https://www.docker.com/.
[17] Documentação oficial do Ceph, acessado dia 02 de Outubro de 2021, https://docs.ceph.com/en/pacific/start/intro/.
[18] Documentação oficial do Percona, acessado dia 02 de Outubro de 2021, https://www.percona.com/doc/percona-server/8.0/myrocks/limitations.html.
[19] Site oficial do Grafana, acessado dia 02 de Outubro de 2021, https://grafana.com/grafana/.