INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO SUDESTE DE MINAS GERAIS - CAMPUS RIO POMBA

Miguel Magalhães Lopes

Benchmark de desempenho entre bancos de dados em diferentes arquiteturas

Rio Pomba

20XX

Miguel Magalhães Lopes

Benchmark de desempenho entre bancos de dados em diferentes arquiteturas

Trabalho de Conclusão apresentado ao Campus Rio Pomba, do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, como parte das exigências do curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Gustavo Henrique da Rocha Reis

Coorientador: CICLANO

Rio Pomba

20XX

Miguel Magalhães Lopes
Benchmark de desempenho entre bancos de dados em diferentes arquiteturas/ Miguel Magalhães Lopes. – Rio Pomba, 20XX-
31 p. : il. (algumas color.) ; 30 cm.

Orientador: Gustavo Henrique da Rocha Reis

Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas, Campus Rio Pomba, 20XX.

1. 2. I. II. III. IV.

Miguel Magalhães Lopes

Benchmark de desempenho entre bancos de dados em diferentes arquiteturas

Trabalho de Conclusão apresentado ao Campus Rio Pomba, do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, como parte das exigências do curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado. Rio Pomba, 00 de dezembro de 20XX.

Gustavo Henrique da Rocha Reis,
Orientador, IF Sudeste MG - Rio Pomba

CICLANO, Coorientador, IF Sudeste MG - Rio Pomba

Dr. BELTRANO
IF Sudeste MG - Rio Pomba

Me. XXXXXXXXXXXXX
IF Sudeste MG - Rio Pomba

Rio Pomba
20XX

Este trabalho é dedicado a todos

aqueles que me inspiraram, em especial

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

Agradecimentos

Resumo

Palavras-chaves: palavra1. palavra2. palavra3. palavra4.

Abstract

Key-words: word1. word2. word3. word4. word5.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DACC	Departamento Acadêmico de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
arm	ARM, originalmente Acorn RISC Machine, e depois Advanced RISC Machine, é uma família de arquiteturas RISC desenvolvida pela empresa britânica ARM Holdings
IDE	
x64	
x86	
aarch64	
ram	
GPU	
TPU	
CPU	
JVM	JVM (Java Virtual Machine) é uma máquina abstrata. É uma especificação que fornece um ambiente de tempo de execução no qual o bytecode do java pode ser executado. As JVMs estão disponíveis para muitas plataformas de hardware e software (ou seja, a JVM depende da plataforma).
IOT	
SBC	
BIOS	
TTL	

UART

WINE

Sumário

Introdução

Esta pesquisa foi baseada na crescente adoção de processadores arm[3] , que conseguem entregar uma eficiência energética muito superior a comumente utilizada nos computadores e servidores (de arquitetura x86[6]). Esta arquitetura possui uma versão 64 bit, que hoje em dia é praticamente a única variante utilizada a x64[5] . Essa arquitetura é, no geral, apenas uma variante aditiva da x86[6] na qual são adicionadas varias instruções e vários suportes, sendo o principal deles o suporte a comandos 64 bits. O mesmo pode ser dito para a arquitetura aarch64[7] ou arm64 que é uma variante aditiva da arm[3] ,essa não é ,como a x64[5] uma versão única,mas sim uma "denominação"das variantes e evoluções da arquitetura arm[3] com suporte a 64bit.As arquiteturas passaram a ser denominadas dessa forma a partir da armv8,entretanto existem versões do arm[3] ,como o armv7l, que consegue trabalhar com instruções de 64bit,apesar de ser uma arquitetura 32 bits.

O foco da pesquisa foi feito em cima do uso de servidores arm[3] , que é baixo, apesar de totalmente possível e existente no mundo corporativo. Existem alguns servidores comerciais que utilizam a arquitetura arm[3] para seu funcionamento. Desta forma foi feita uma comparação na utilização desses processadores para a simulação de uma aplicação de banco de dados. Essa aplicação simula a utilização de forma realística de uma base de dados de uma locadora.

O cenário foi escolhido a partir de um esquema de banco de dados genérico da internet e foram utilizados os bancos de dados PostgreSQL e MariaDB, visto que são os bancos de dados de propósito geral mais utilizados atualmente. Foi preferido o MariaDB sobre o MySQL visto que não existe uma versão dele para a arquitetura arm[3] e ambos são basicamente o mesmo sistema.

Foi criado um programa para a geração de dados realísticos baseados na biblioteca faker implementada na linguagem Python. Estes dados são gerados para cada país específico em idiomas e caracteres compatíveis com a região escolhida. Desta forma é plausível que estes dados, como nome, telefone, endereço e até mesmo usuário e senha sejam possíveis de existirem. Foi escolhido essa forma de inserção pois um preenchimento de dados totalmente randômico e de tamanho fixo podem apresentar discrepâncias com o desempenho num ambiente real de uso, afetando tanto o tempo, quanto carga dos processadores de

forma negativa. Os dados utilizados em cada teste são exatamente os mesmos, com a diferença apenas da quantidade, simbolizando o uso em um ambiente real. Desta fomra o benchmark se torna mais aplicável a realidade.

1 Fundamentação Teórica

1.1 arquiteturas

Por definição arquitetura de computador é um conjunto de circuitos eletrônicos padronizado associado a um conjunto de instruções de forma a simplificar a programação deles para que funcionem comandos diferentes do binário para a programação de um eletrônico. Os compiladores utilizam esses conjuntos de instrução para que seja convertido o código de uma linguagem de programação para um binário de um programa. A arquitetura também define/limita várias propriedades do hardware, como quantidade máxima de ram[8] , de disco, suporte ou não de saída de video, capacidades de rede e vários outros, mesmo que algumas dessas limitações possam ser contornadas utilizando variações da arquitetura chamadas microarquiteturas.

Uma micro arquitetura é quando adiciona-se tanto um circuito eletrônico novo ao circuito original da arquitetura quanto apenas uma simplificação ou reorganização dos comandos originais de uma arquitetura, entretanto, em uma micro arquitetura essas modificações são muito pequenas de forma a serem mais similares a arquitetura original do que uma nova arquitetura. Dessa forma as micro arquiteturas podem ser consideradas updates de uma arquitetura e quando são acumulados muitos desses updates, pode ser que seja gerada uma nova arquitetura, como foi o caso da arquitetura x86[6] para a arquitetura x64[5] , onde nesta última foi um upgrade grande o bastante da x86[6] a ponto de ser considerado uma nova arquitetura. A principal e mais visível diferença entre esses dois é a mudança de 32bit(na x86[6]) para 64bit(no x64[5]).

As arquiteturas também podem ser definidas sobre características fora da CPU[11] . Elas podem ser definidas em GPU[9] , TPU[10] e vários outros módulos de hardware de um computador, inclusive existindo arquiteturas especiais que são aplicadas em nível de software. Estas não são necessariamente arquiteturas de computador mas sim um tipo diferente de arquitetura, onde existem máquinas abstratas que simplificam a programação de uma linguagem para que ela funcione de forma mais compatível com várias máqui-

nas de arquiteturas de hardware diferentes. Sendo assim, se faz necessário otimizações na parte do código e da máquina virtual, como o caso da JVM[12] do java, em que a máquina virtual de cada arquitetura pode sofrer otimizações e isso faz com que ela funcione de forma melhor dependendo da máquina virtual em uma arquitetura e pior em outra e vice-versa, mas sem alterar o código.

As arquiteturas não são limitadas apenas a esses previamente citados, mas as arquiteturas podem estar presentes em todos os tipos de circuitos integrados. Como exemplo os processadores de roteadores e aparelhos IOT[13] tais como lâmpadas e tomadas inteligentes. Isso quer dizer que uma arquitetura não necessariamente é algo que precise de um hardware potente ou que só funcione ou exista em computadores, mas são a forma como os algoritmos são interpretados no hardware, o que quer dizer que desde que exista um software e um hardware que se comuniquem existe uma arquitetura e provavelmente houve uma conversão da linguagem de programação para a linguagem de máquina dessa arquitetura deste dispositivo

As arquiteturas de computador são definidas para hardwares específicos, mas os softwares não necessariamente precisam de ser funcionais apenas em uma única arquitetura, por mais que ela seja diferente da arquitetura de outro computador. Isso é devido pelo fato dos conjuntos de instruções poderem ser diferentes mas suas funcionalidades gerais podem ser iguais. Mesmo que uma arquitetura seja totalmente diferente da outra, os softwares de uma podem funcionar na outra, por mais que sejam necessárias algumas adaptações. Algumas dessas adaptações podem ser tão grandes que às vezes é muito complexo essa adaptação de código. Para esses casos, ou mesmo para testar o código de uma arquitetura em outra, sem a necessidade dessa adaptação são usados programas chamados de emuladores ou simuladores. Estes programas funcionam como uma camada de compatibilidade entre a arquitetura real da máquina que está rodando e a arquitetura na qual o programa foi projetado para funcionar. Entretanto esse processo pode acarretar em uma perda considerável de desempenho, podendo resultar em casos onde máquinas com 516 gigaflops sejam necessárias para se emular máquinas com 230 gigaflops, como no caso de um emulador do sistema de console Playstation 3 utilizando o emulador rpcs3, e mesmo com essa ineficiência, esse emulador não tem 100% de compatibilidade com os

softwares existentes na plataforma,de forma que nem todos os softwares dessa plataforma funcionam exatamente como deveriam, ou mesmo funcionam. Por mais que ambas as máquinas executem o mesmo kernel de sistema ainda sim haverá perda de desempenho muito grande pois apesar de, em teoria, serem o mesmo sistema operacional a diferença de arquiteturas possui um peso muito maior do que o sistema operacional utilizado.

Esse é um exemplo de como mesmo com tudo para ser um cenário igual de utilização ou mesmo um cenário melhor ao se trocar uma arquitetura de um computador inúmeras adaptações devem ser feitas ou,como no caso do macos,criadas camadas de compatibilidade.Após o lançamento dos macbooks de 2020 com processador M1 que funcionam com a arquitetura aarch64[7] a apple lançou uma camada de compatibilidade dos softwares com arquitetura x64[5] para aarch64[7] chamado de rosetta2 esse software funciona parcialmente como um emulador,exceto que ele faz as adaptações num nível mais próximo do da máquina real e do sistema operacional nativo da máquina,resultando num desempenho muito superior a qualquer emulador existente,o rosetta2 funciona de forma análoga ao projeto WINE[18] do Linux que reinterpreta os programas windows para funcionarem no Linux,você tem uma pequena perda de desempenho por esse processo de reinterpretação em alguns casos,mas em outros essa perda é bem mais visível

As arquiteturas de computador podem variar em diversos fatores de uma para outra de forma que existam varias funcionalidades que não foram pensadas para uma arquitetura que existem em outras.existem também propósitos diferentes para diferentes arquiteturas,como o caso dos processadores arm[3] que foram pensados para entregar uma grande eficiência energética,enquanto os processadores x86[6] foram pensados para apresentarem grande poder de processamento

Existem alguns computadores com processador arm[3] que não são SBC[14] fazendo com que eles possuam várias possibilidades de upgrade, que não são possíveis nos computadores SBC[14] . Sendo assim, existem algumas pequenas variações no funcionamento dos computadores mesmo dentro de uma mesma arquitetura que tenderia a seguir padrões mais uniformes. Entretanto, uma peculiaridade tende a ser comum nos processadores arm[3] , eles costumam apresentar uma GPU[9] integrada e algumas outras unidades de

processamento especializadas nas quais os processadores x86[6] costumam ter que ser adicionadas com chips externos. Uma dessas unidades é o TPU[10] que ficou mais conhecida com o lançamento do Windows 11 que exige em sua instalação por propósitos de segurança. O principal propósito da arquitetura arm[3] entretanto não é se diferenciar tanto da arquitetura x86[6] , mas sim tornar os computadores mais energeticamente eficientes, tanto que um computador doméstico comum utiliza de 200 a 300w por hora enquanto um computador raspberry pi 4, que é o computador arm[3] mais potente atualmente da marca e mais popular, consome em torno de 15w hora. É uma grande diferença, principalmente levando em conta que ambos tem a capacidade de utilizar os mesmos programas de trabalho, se considerarmos sistema operacional Linux e programas open source, tanto editores de texto, navegadores de internet quanto IDE[4] s de programação que estão disponíveis para ambos e para um uso comum funcionam tão bem quanto em um cenário real.

Como os processadores arm[3] começaram a ficar mais comuns, visto que algumas fabricantes como a Apple e Gigabyte agora oferecem computadores e servidores baseados nessa arquitetura, faz com que seja cada vez mais fácil de se utilizar tal arquitetura por existirem mais consumidores e consequentemente uma oferta maior de programas feitos para serem executados nessa arquitetura. Visto o quanto um computador com processador arm[3] economiza energia para entregar o mesmo poder de processamento de um outro com processador x86[6] , essa diferença pode ser muito benéfica para os vários tipos de empresas que utilizam servidores, já que isso pode significar um impacto considerável no consumo energético da empresa dependendo do quanto ele é utilizado a nível de processamento.

1.2 bancos de dados

Banco de dados é um método de armazenamento de dados de forma estruturada para que as informações sejam fáceis de serem associadas e filtradas. Podem, também, ser armazenadas de forma a economizar espaço de armazenamento, dependendo da otimização do banco de dados além da possibilidade de realizar redundâncias de segurança dos dados. Através dessas características, justifica-se a escolha dos bancos de dados como alvo do benchmark realizado para essa comparação de arquiteturas.

Sistemas de computadores, dos mais diversos tipos, utilizam banco de dados para armazenamento de suas informações. Dessa forma, as informações ficam estruturadas em um formato padrão permitindo um acesso rápido a elas. os tipos de bancos de dados analizados são os bancos de dados sql relacional que são os mais genéricos,de forma que podem ser utilizados no máximo de aplicações diferentes possiveis,isso faz com que os bancos de dado sql sejam os melhores para serem simulados,um dos que foi analizado para ser testado foi o mongodb e o oracle,mas o mongodb não é relacional e o da oracle não existe uma versão para arm[3] até o momento que o projeto foi pensado.

1.3 docker

docker é um sistema de virtualização de maquinas que busca ser mais simples de ser gerenciado e entregar mais desempenho do hardware real da maquina que está sendo usada.o docker busca executar as chamadas de seus ambientes virtualizados ,chamados de containers,ao nivel do sistema operacional,logo a cima do kernel do sistma,isso faz com que seja possivel dedicar parte dos hardwares da maquina real para um container,sendo possivel dentre outras coisas dividir de forma fixa porções do hardware ,como definir limite de ram usado ou quantidade de nucleos de cpu usados,isso foi aplicado nos testes realizados,para padronizar as expecificações tecnicas dos container usados nos testes,como descrito em seção 3.6,outra das funcionalidades do docker é a implementação de volumes,que é a fora de isolar as pastas do container e manter elas mesmo caso o container seja reinstalado para realizar atualização,essas atualizações são feitas por meio do metodo de instalação dos containers docker,as imagens,as imagens são feitas sequindo o modelo de snapshot,onde cada imagem não tem nenhum programa ou biblioteca atualizada a menos que seja deliberadamente feito pelo proprio programa ou pelo administrador do container durante sua execução,mas esse segundo geralmente não é feito,visto que os containers quando precisam ser atualizados é preferivel usar a atualização por imagem do que executando comandos por dentro dos containers rodando,um grande motivo disso é o docker-compose,como é descrito no seção 1.4. o docker foi desenvolvido a pedido da google usando a linguagem go,também desenvolvida por ela,ele foi desenvolvido visando a simplificação da gerencia dos clusters de processamento da google,que utilizam dezenas de computadores mais fracos para fazer o processamento , de forma que é mais barato para compor o ambiente para o processamento tão potente quanto se comprasse varias

armas muito mais caras,alem disso o docker também facilita a manutenção devido a facil reimplementação de um container caso a maquina tenha algum problema,como visto em situações que ocorreram durante o periodo do desenvolvimento da aplicação,para se reinstalar o stack de aplicações de monitoramento de dados do ELK seção 1.5 foi necessário apenas em torno de 2 horas,incluindo completa formatação do servidor onde o stack estava rodando,instalação do stack e completa configuração do mesmo,isso caso não fosse feito com docker facilmente levaria 1 dia para a completa reinstalação e reconfiguração do stack,isso porque o stack foi instalado facilmente com o docker-compose e todos os volumes contendo os arquivos de configuração possuiam backup,e a restauração de backups de volumes é extremamente simplificada,ainda mais utilizando a ferramenta portainer seção 1.6 com o docker-sock,o docker-sock é um metodo de controle remoto do docker,utilizando o socket todas as configurações e gerenciamentos são feitos da mesma forma como se que se tivesse sendo feito diretamente na maquina que o docker está rodando,isso faz com que o docker seja muito simples de unificar e replicar o mesmo comando em diversas maquinas diferentes.

1.4 docker-compose

1.5 elasticsearch monitoring stack

1.6 portainer

portainer é um painel de administração do docker no qual simplifica muito a administração e configuração de varias coisas mais complexas do docker,inclusive facilitando muito a definição de uso de hardware que será usado por cada container e facilitando o backup dos volumes,possibilitando inclusive o download de backups dos volumes diretamente do navegador em formato de arquivo tar.gz,facilitando também a recriação dos containers para atualização deles para versões mais novas,essa ferramenta alem de tudo é muito simples de ser instalada,visto que roda diretamente de dentro de um contaienr docker multi arquitetura,sendo assim o mesmo comando de instalação funciona em qualquer arquitetura disponivel para o container. o portainer também consegue unificar a administração dos diversos servidores utilizados durante o desenvolvimento e testes,sendo necessário apenas instalar o portainer em apenas um dos servidores,todas as outras ma-

quinas só foram necessárias ativar a conexão remota pelo docker socket,isso faz com que o monitoramento de logs,uso de hardware e outros possa ser feito por alto diretamente pelo navegador,mas esse monitoramento não pode ser logado por isso foi preferido seguir o metodo descrito no seção 3.6.

1.7 biblioteca faker

A biblioteca faker é uma biblioteca conhecida para a geração de mock data, que são dados gerados randomicamente para testar a capacidade de um algoritmo em lidar com dados. Essa biblioteca é comumente utilizada na etapa de testes de um algoritmo, seja para testes de segurança, para proteger de bots de criação de contas ou algo semelhante. É também utilizado para testes de estresse, quanto para qualquer tipo de teste que dependa de dados "realisticos".
Essa biblioteca foi selecionada pois é de fácil utilização, possui documentação abrangente e de fácil compreenção e suporte para múltiplos idiomas. Apesar da biblioteca apenas suportar todos os seus tipos de geração mais completos na localização dos EUA, dentre esses está um tipo que corresponde a todo um perfil de usuário, contendo desde endereço até a senha fraca ou forte. Para propósitos de testes, foram gerados apenas dados na localização do Brasil.

1.8 biblioteca psutil

a biblioteca psutil é compilada em c,uma biblioteca para monitoramento de hardware,ela consegue monitorar diversos tipos de informações do hardware,como informações detalhadas de uso de cpu,ram,processos,disco e varios outros dados completos.a biblioteca consegue listar varios dados das formas mais completas possiveis.a biblioteca é a mais utilizada e mais importante das bibliotecas de monitoramento de hardware da linguagem python.a psutil é de muito simples utilização e documentação muito bem feita

2 Trabalhos Relacionados

3 desenvolvimento

O desenvolvimento do software foi feito utilizando vários métodos de análise de log com o objetivo de agilizar a depuração, possibilitando que os dados gerados pudessem ser facilmente conferidos durante o desenvolvimento.
Durante esse capítulo, serão descritos os procedimentos mais importantes das etapas de desenvolvimento e funcionamento do software, com o intuito de proporcionar um entendimento simples.

3.1 bibliotecas criadas

algumas bibliotecas foram criadas para facilitar o desenvolvimento,delas algumas valem a pena serem mencionadas mais detalhadamente,mas outras apenas vão ser citadas aqui. foi criada uma biblioteca para gerenciar a interação com arquivos sqlite,essa biblioteca trabalha de forma simplificada para o acesso ser mais rapido de se implementar e lidar apenas com comunicação de parametros em formato json,dessa forma quando se é feita uma query o input deve ser um dictionary com as keys sendo os nomes das colunas e os conteudos os valores buscados,isso só é válido pois ele só busca valores iguais,não se preocupa com valores aproximados ou limiares,isso poderia ter sido implementado,mas não foi visto como necessário para o propósito dos testes propostos.a outra vantagem é que os retornos das consultas é dado em formato dictionary e quando valores são inseridos apenas são necessários os dictionaries compativeis com o sqlite. uma outra dessas bibliotecas foi uma para o tratamento de erros,essa classe é uma derivação da classe padrão do python de exceção,essa biblioteca é composta de 3 classes cada uma para um tipo de mensagem e tratamento de erro,apenas para simplificar e facilitar o debug do codigo durante a manipulação dele,ela não faz real diferença em relação ao funcionamento em si,exceto no fator de poderem ser chamadas correções expecíficas dos erros quando um try-catch é usado numa função. ainda foi criada uma biblioteca de timer,usada nos testes de tempo do projeto,essa biblioteca apenas é uma contração de forma simples e com tratamento de erro do uso da biblioteca time do python,essa tecnica de medida de tempo é

amplamente utilizada,mas apenas para fins de simplificação no momento do uso foi criada essa biblioteca.

3.1.1 loggingSystem

essa biblioteca é a responsável por toda a gerencia e logs e por todo o monitoramento do subseção 3.1.3,essa biblioteca basicamente é um complemento da biblioteca padrão de logs do python,que tem uma implementação de comunicação com os servidores logstash,a biblioteca de forma inteligente verifica se os parametros de comunicação com o logstash estão funcionais,senão automaticamente salva todos os logs em um arquivo .log cujo nome foi informado durante a instancia do objeto da classe,a classe apenas consegue fazer essa verificação durante sua instancia,caso após isso a conexão seja perdida os logs não são enviados a lugar algum. as mensagens de erro são gerenciadas de forma simplificada,onde existe um parametro que define o formato da mensagem de log e o nome do gerenciador de log,alem de outros pequenos parametros para identificação de onde saiu o log que foi registrado. existem alguns metodos nessa biblioteca que auxiliam no tratamento do stack overflow,onde esses metodos monitoram de onde saiu a execução do elemento que executou essa função,como no caso do tratamento de erro do stack overflow na criação de um dado de inserção da classe de criação de banco de dados,onde caso durante o tratamento de erro que faz com que a classe seja chamada novamente em loop até que consiga retornar um valor válido essa chamada recursiva não gere um esgotamento de memória,já que caso chegue a um valor máximo de execuções outra exceção é chamada e simplesmente em vez de retornar um valor de inserção é retornado um valor nulo. alem de saber qual a função que a chamou esse metodo existe uma outra implementação onde retorna toda a pilha de chamadas do algoritimo,sendo assim permitindo que um tratamento de erro mais complexo seja possivel sem grandes alterações no código.

3.1.2 paralelLib

essa biblioteca possui 2 implementações diferentes de paralelização,utilizando a classe threading e a classe multiprocessing,a classe threading utiliza threads para a paralelização de execuções,esse método foi inicialmente utilizado por ser de mais simples implementação e mais simples monitoramentoda execução,entretantodevido as limitações de segurança dessa classe foi escolhida uma implementação utilizando a classe multi-

processing,onde essa classe cria um grupo de subprocessos para o codigo que irão ser responsáveis pela execução do algoritimo selecionado. ambas as classes implementadas,a derivada de threading e a derivada de multiprocessing foram feitas de formas análogas e de simples substituição em codigo de uma para a outra de acordo com a necessidade,sendo assim funcionam muito parecidas.ambas funcionam com uma classe de gerencia chamada paralel,no caso de threading sendo paralel_thread e no caso de multiprocessing sendo paralel_subprocess,assim como classes worker com nomes semelhantes,worker_thread e worker_subprocess. as classes de gerencia funcionam gerando um array de objetos da classe worker,sendo cada um equivalente a um subprocesso/thread da execução paralela,também existe um array ,que pode ser apenas um elemento,com as funções que serão executadas no processamento paralelo,alem disso gera um objeto que contem os elementos que serão processados durante o processamento paralelo,esses objetos são passados como kwargs das funções que serão executadas,podem ser aceitas varias funções des de que sigam apenas uma regra,essas funções devem ser exatamente iguais,mas de objetos instanciados diferentes,como no caso utilizado no algoritimo,onde cada função é de um objeto com uma credencial diferente para a comunicação com o banco de dados,de forma que não seja executada mais de uma comunicação com uma credencial por vez. os workers vão ciclando as funções a medida que vão iterando pelo array dos parametros.durante a execução dessas funções vão sendo removidas do objeto contendo os kwargs um por um o kwarg logo antes de ser usado,dessa forma garantido que uma operação não será feita 2 vezes ,isso é importante visto o propósito desse algoritimo,que é simular a mesma quantidade e operações em bancos de dados diferentes,se um elemento fosse utilizado mais de uma vez esses valores se alterariam.caso um dos elementos ja tenha sido usado e acabe sendo pelo pelo worker,um erro vai ocorrer,o worker vai ignorar esse elemento e tentará o próximo disponivel,caso o numero de elementos seja menor que 1 ele irá então terminar o loop e parar a execução após o fim da execução de todos os processos paralelos o objeto de gerencia ,se for requisitado,irá retornar os resultados das operações onde o processamento paralelo foi pedido,para os propósitos deste algorítimo isso não foi necessário nenhuma vez,sendo assim essa funcionalidade está incompleta.

3.1.3 monitorContainer

essa biblioteca se baseia na biblioteca seção 1.8 essa biblioteca utiliza um metodo não muito seguro chamado eval,uma função nativa do python que consegue executar uma função declarada como um string,sendo assim podendo muito facilmente serem alterados os parametros que seriam consultados do hardware onde o container está sendo rodado. essa biblioteca faz a consulta a partir de um arquivo json que lista as funções da biblioteca psutil que devem ser executadas,acompanhadas dos parametros de entrada e o filtro de retorno que será aplicado,isso faz com que muito facilmente possam ser pesquisadas apenas as informações realmente necessárias,simplificando assim o trabalho da analize de logs. a biblioteca ainda implementa o uso da biblioteca de logs para poder enviar todos os logs de forma automática para o logstash ou para um arquivo de log local,como descrito em subseção 3.1.1

3.1.4 gerenciadosDeBD

essa biblioteca é feita para que a gerencia das conexões do banco de dados seja mais facil,visto que tem as funções adaptadas para serem iguais para o mariadb e para o postgres,alem disso existem tratamentos de erro customizados para todos os erros que foram observados durante a etapa de desenvolvimento,dessa forma todos os erros q acontecem são tratados de forma similar,mas como as bibliotecas são diferentes e desenvolvidas por entidades diferentes,varias das correçoes de erros e funcionamentos são diferentes,um exemplo simples disso é q para a biblioteca do postgres para executar um arquivo sql só precisa do arquivo ser aberto e executada a função read,que retorna o string contido nele,mas na biblioteca do mariadb tem que ser lido linha a linha e executada linha a linha,na biblioteca criada foram feitas as adaptações para que independente do banco de dados usado a função de executar um arquivo sql apenas precisa do caminho do arquivo no sistema,ela ja abre o arquivo e o executa. algumas outras coisas são feitas,no postgres por exemplo existe uma função de rollback,que é usada para desfazer alguma operação quando ela é identificada como acontecendo com erro numa comunicação com o banco de dados,não existe algo equivalente para o mariadb,sendo assim a classe criada trata esse erro tentando executar mais 5 vezes a operação,para garantir que não foi algum erro de conexão aberta,após essas 5 vezes o algoritimo considera q a operação inserida está com problema e a ignora. essa biblioteca serve principalemnte para essa generalização,alem de

controlar a inserção,execução de sql,ainda gerencia consulta,criação de usuario,ler diretamente o sqlite para realizar as operações escritas lá,dessa forma essa biblioteca simplifica muito a interação com o banco de dados,essa simplificação chega num ponto que tornou possivel a paralelização ,visto que uma adaptação para isso é bastante complexa caso o código não tenha sido pensado nisso des de o começo,sendo assim o algoritimo sendo modularizado acabou fazendo possivel a implementação de forma paralela. a biblioteca ainda faz com que seja possivel consultar o sqlite dos dados gerados diretamente na comunicação com o banco de dados final,isso faz também com que seja mais economico de memória ram o algoritimo,mas faz também com que ele consuma por mais tempo disco e cpu,mas não necessáriamente em maior quantidade,isso faz com que tenha sido necessário um controle mais preciso da quantidade de threads na execução do benchmark,pois como o cpu é usado por mais tempo isso poderia causar um travamento da maquina que está gerando os testes e uma subsequente inconsistencia dos dados de benchmark gerados.

3.2 geração de dados

os dados foram todos gerados para um sqlite projetado para ser simples de aceitar qualquer formato de dados que pudesse ser gerado para qualquer tabela Essas etapas funcionam da seguinte forma:
No início, os dados foram gerados em sqlite pelo fato que, caso haja algum problema e o computador, que estava gerando os dados, seja abruptamente desligado, não se perdem as informações que já foram criadas e salvas no banco de dados. Desta forma, economiza tempo uma vez que esta etapa é a mais demorada.
o codigo consegue gerar os dados de 3 formas,apenas a ultima é realmente utilizada. As outras duas foram feitas para propósito de testes

- gerar uma quantidade x de dados de uma tabela específica
- gerar uma quantidade x de cada tipo de dado para cada tabela do banco de dados final
- gerar uma quantidade aleatória de cada tipo de dado para cada tabela do banco de dados final até atingir o total de operações informadas

Todos esses tipos de geração são feitos pela mesma função que são alterados pelos parâmetros passados a ela. Assim a função prioriza os parâmetros referentes aos três tipos de geração.

Além dos parâmetros relacionados aos tipos de geração informados, existem os parâmetros relacionados a seção 1.7 e uma lista que define quais tipos de operação, descritos em seção 3.4, que define, caso não esteja vazia, quais os tipos de dados que serão gerados. Isso foi útil pois utilizou-se o tipo de criação de dados de inserção na primeira etapa, antes de gerar os dados dos outros tipos.

Devido a limitações intencionais, o valor total de operações inseridos em cada operação deve considerar a quantidade da operação anterior. Isso se faz necessário pois o algorítimo verifica apenas a quantidade total de elementos cadastrados no sqlite, ou seja, no caso dos testes feitos,foram gerados 50.000 operações de inserção,e posteriormente foram pedidos 5.000.000 operações randômicas,o que resultou em 50.000 operações de inserção seguidos de 4.950.000 operações randômicas gerados. a geração de dados inicialmente foi pensada para rodar em um servidor de forma sequencial,isso simplesmente por que não teria processamento paralelo inicialmente no siftware,entretanto quando isso se tornou necessário também foi possivel utilizar o codigo previamente existente da geração de dados para a geração de forma paralela,isso se mostrou uma grande vantagem para a etapa de desenvolvimento onde varios bancos de dados de testes foram gerados para que fosse garantido que todas as partes desenvolvidas do software estivessem funcionando corretamente,devido a forma como isso foi pensado,o grande limitador da velocidade de geração de dados é o fator randomico,devido a toda a recurção que ocorre devido aos tratamentos de erros,alem disso o outro maior limitador é a velocidade de disco,visto que a aplicação do sqlite aceita apenas uma inserção por vez no banco de dados de forma paralela,isso faz com que quanto mais rapido fosse possivel a escrita em disco mais rapidamente esse fator limitador era deixado de lado,devido ao fato de durante os testes praticamente todos ter sido usado ssd para o salvamento desses dados isso não impactou em quase nada na velocidade de geração dos dados

3.3 software de benchmark

o benchmark foi feito utilizando um software feito também desenvolvido por mim para esse propósito,o software de benchmark está mais para um software para gerar es-

tresse na maquina na qual está rodando o banco de dados por meio de varias execuções de comandos direto no banco de dados. o software de benchmark funciona carregando e construindo os dados a partir das inserções no arquivo sqlite,os testes de estresse foram tentados de algumas formas: primeiramente o algoritimo inicia o container que contém o banco de dados que está sendo analizado,após isso é inserindo uma quantidade definida de operações que é lida de forma sequencial do banco de dados inicial,após isso são executadas essas operações até que terminem,depois disso o mesmo procedimento é feito para a outra maquina,após isso o container com o banco de dados é desligado e o próximo tipo de banco de dados é iniciado nas duas maquinas,e o processo se repete a outra forma é um pouco mais rapida e eficiente em relação a execução das multiplas maquinas.Isso se dá pelo fato que o processo de inserção é feito de forma paralela,de forma que as inserções de cada banco de dados em todas as maquinas é feito simultaneamente,mas também de forma sequencial,o funcionamento é muito parecido com o da anterior,primeiro é criada uma thread para cada maquina,em cada thread é lido de forma sequencial as operações que serão inseridas,assim que as operações acabam em qualquer uma das threads o container do banco de dados é parado e aguara a outra thread acabar para poder começar o mesmo procedimento para o outro tipo de banco de dados a terceira forma é parecida com a segunda,diferente apenas na forma como as operações são executadas,a partir desse metodo as operações são inseridas de forma paralela,sendo assim cada thread de cada maquina possui uma quantidade definida de threads filhas,essas threads filhas executam as operações a partir do que existe numa queue de elementos que quer dizer que uma operação não necessáriamente será executada ,visto que pode depender de uma operação que ainda não foi executada,isso é possivel em alguns casos raros,assim como num ambiente real,onde um funcionario de uma empresa pode editar um elemento ao mesmo tempo que outro edita o mesmo elemento,em ambientes reais existem tratamentos para que isso não ocorra,mas no ambiente desses testes,nenhum tratamento para impedir isso foi feito,exatamente para simbolizar o pior cenário possivel para uma aplicação com comunicação com bancos de dados a ultima forma é uma variante da segunda e terceira formas de operação,onde são feitos os testes para cada maquina de forma sequencial, mas a execução das operações é paralela,essa ultima foi pensada para que possam ser testadas alguns ambientes a procura de erros,não foi pensada necessáriamente para o uso no benchmark final o software de benchmark por si só não é o bastante para analizar o desempenho das arquiteturas,ele depende do container docker descrito em **??**,esse container modificado possui um metodo

de monitorar de dentro para fora tudo que acontece dentro dele por meio do seção 3.7

3.4 tipos de dados gerados

Foram selecionadas para os teste apenas as operações mais utilizadas por um banco de dados:

- inserção de um novo dado
- leitura completa de todos os dados de uma tabela
- busca de elementos filtrados em determinada tabela
- busca de apenas alguns dados de elementos filtrados em determinada tabela
- edição de elementos
- deleção de elementos filtrados

Antes de serem geradas, essas operações passam por vários processos de tratamento de erro para se certificar que não houve depêndencia alguma que não foi gerada como, por exemplo, gerar uma cidade sem existir um país cadastrado. Esse foi um dos motivos de ter sido utilizado um arquivo sqlite ao invés de outro método de armazenamento. Dessa forma pode-se executar essa consulta de dependência de forma rápida e apenas retornar índices válidos para associações de tabela. os varios dados gerados pelo programa de geração de dados são usados parcialmente para a geração dos dados novos,mas todos os dados gerados são gerados da forma mais simplificada e sendo assim,poucas coisas que ja existiam dentro do banco de dados dos dados ja cadastrados,apenas os ids cadastrados são usados,apenas para que as associações de dados sejam possiveis,os outros dados são ignorados,sendo assim não são consultados os dados para as queries e nem nenhum outro dado como update,sendo assim,a maioria dos dados gerados não conseguem retornar algum valor,como updates ou deleções,isso foi feito dessa forma apenas para simplificar o algoritimo,e seria totalmente possivel a modificação para que esses dados sejam levados em conta no futuro.

3.5 alimentação do algoritmo

O algoritmo funciona de forma que qualquer banco de dados possa ser utilizado para ter seus dados gerados. Basta utilizar um arquivo json e seguir um determinado padrão que é composto por uma tag para cada banco de dados, o nome do banco e dentro dele uma estrutura json com uma tag com o nome da coluna com seu conteúdo em um array de string contendo o primeiro tipo de dado e os outros valores adicionais.
Dentro do algoritimo existem vários tipos de dados aceitáveis, tais como id, nome, associação e timestamp, sendo que cada um deles possui um tipo bem definido pelo seu nome, mas o único que vale a pena citar seu funcionamento é o associação. Como descrito em seção 3.9 existe uma tabela do sqlite contendo as quantidades de elementos associados a cada tabela do banco de dados final. O tipo de associação vai pegar esse valor e usar uma função de seleção randômica para que seja escolhido um elemento de id existente no intervalo descrito, caso não exista algum elemento dessa tabela, por um tratamento de erro é gerado um elemento para ela, permitindo o funcionamento da associação no novo elemento que foi criado.
Os únicos dados que não são passados para o algoritmo funcionar são o país que deve ser gerado os dados, de acordo com seção 1.7, e a quantidade de dados que devem ser geradas, ambos sendo necessários de serem inseridos na chamada da função. Os outros dados relevantes referentes ao funcionamento da chamada da função estão em seção 3.2

3.6 containers docker

os containers docker foram criados a partir do sistema alpine linux,no qual foram feitos 2 containers diferentes,um deles para o mariadb e outro para o postgres,os containers se certificam de criar o banco de dados de forma correta durante a inicialização dele,após isso o container durante a sua inicialização se certifica que o python está instalado e tudo necessário para que o daemon ,descrito em seção 3.7,funcione,após isso o banco de dados do container é finalmente iniciado. essa forma de instalação do python se certifica que tanto o interpretador quanto as bibliotecas usadas estão sempre atualizadas todas as vezes que o container é instanciado,apesar de esse metodo causar uma grande demora na primeira inicialização do container,mas devido a forma como o script de incialização funciona somenta a primeira inicialização é impactada na sua velocidade de inicialização.

os containers também possuem um healthcheck para verificar se o banco de dados está acessivel por fora do container,entretanto o contaienr não consegue exibir corretamente a saude do container devido a algum bug que não foi possivel de corrigir o container baseado no alpine possui uma especificidade,para a arquitetura armhf,presente no orangepipc +,ele só funciona até a versão 3.12 sem ter grandes modificações,isso devido a uma mudança no metodo como o sistema operacional manipula o relogio do sistema,o que quer dizer que não é possivel utilizar o gerenciador de pacotes dele que ,para se conectar ao servidor online,utiliza o ssl que,dentre outras coisas,utiliza o horario do sistema para certificar que a conexão é segura.sendo assim a versão utilizada foi a versão 3.12. outra expecificidade do alpine é que ele é um dos poucos sistemas linux que não possui o gcc incluso como pacote padrão do sistema,isso faz com que o python tenha alguns problemas ao instalar algumas bibliotecas,dentre elas o psutil,que foi utilizado pelo seção 3.7,para solucionar isso foi necessário instalar um programa chamado linux-headers que é um conjunto de programas e bibliotecas padrões das distribuições linux,alem é claro do próprio gcc,após a adição desses programas o container pode funcionar corretamente

3.7 daemon de monitoramento

o daemon de monitoramento monitora os status da maquina na qual está rodando,seja uma maquina fisica ou um container docker,para esse fim é tilizada a seção 1.8 que é a principal biblioteca python quando se trata de monitoramento de hardware. o daemon utiliza também uma biblioteca feita para simplificar o tratamento de log,uma das funcionalidades contidas nessa biblioteca é a comunicação com o aglutinador de logs logstash,isso feito em cima da biblioteca python-logstash. o arquivo de configuração do daemon é um json constituido de 2 partes,os parametros para se conectar ao logstash e os dados que serão coletados da maquina local,esses dados coletados serão enviados para o servidor logstash onde lá serão trabalhados o daemon envia esses dados com um intervalo de 0.1 segundos,que é apenas passado para que não sejam enviados dados errados de cpu para o servidor de log,mas poderia ser aumentado para diminuir o estresse na maquina que está rodando ele e diminuir um pouco o estresse do servidor de coleta de logs,que com as maquinas usadas não é necessário

3.8 ambiente de desenvolvimento

O ambiente utilizado foi dividido em duas partes, programação local e remota. Na programação remota as execuções foram feitas em um servidor baseado em Raspberry PI e na programação local foram feitas no próprio computador local. A programação remota se provou bem útil quando houve a necessidade de troca de computador ou sistema operacional, facilitando ainda a implementação de um servidor unificado de análise de log
Foram utilizados Visual Studio Code e DBeaver para o desenvolvimento da aplicação principal e ainda foi utilizada uma implementação da suite ELK(elasticsearch ,logstash e kibana) de análise de log. Os dois primeiros foram escolhidos dentre outros motivos por serem opensource e estarem disponíveis tanto para Linux quanto Windows, visto que ambos sistemas operacionais foram utilizados para o desenvolvimento de acordo com a necessidade no momento.
Para o controle de versão foi utilizado o Git, deixando registrado todo o histórico de alterações do programa, o que se mostrou bem util para o rastreio de erros ocorridos durante o longo tempo de desenvolvimento do código.
Para os testes iniciais do código foram utilizados containers docker não limitados durante a etapa de implementação dos scripts do DBbench, que demandam a existência dos servidores dos bancos de dados sendo executador, ao contrário do restante do desenvolvimento da aplicação, que não interagiu diretamente com os bancos de dados.
A geração do banco de dados inicial foi feito completamente em um Raspberry Pi 4 com um pequeno overclock, essa geração foi feita durante alguns dias, visto que o mini pc, de acordo com testes de velocidade feitos seção 4.1, esse bd constitui-se de todas as operações feitas durante o teste de estresse.

3.9 sqlite

O banco de dados do sqlite foi projetado para ser totalmente maleável e modular, de forma que não teriam que ser geradas várias tabelas para os vários tipos de dados do benchmark. Foi pensado no seguinte método para se facilitar o desenvolvimento sendo uma tabela de índices de total de elementos de cada tabela, sendo cada elemento composto apenas pelo nome da tabela e pelo total de elementos cadastrados, a outra tabela é um pouco mais complexa.

- a tabela de operações a ser executada é constituida de uma coluna inteira para o tipo de operação que será realizada, de acordo com seção 3.4
- uma coluna é uma string contendo o nome do banco de dados que será executada a operação
- uma coluna inteira para, se for necessário, conter o id no banco de dados do elemento trabalhado na operação. No caso de uma inserção é o id do novo elemento por exemplo
- uma coluna text, nessa coluna serão inseridos valores adicionais necessários para a execução da operação, como os parametros de quais colunas devem ser atualizadas em um update. Aqui os dados inseridos são salvos em formato json para facilitar o trabalho com a linguagem python, visto que existe uma conversão direta de string json para o tipo dictionary do python
- seguindo a mesma idéia da coluna anterior, essa coluna também é uma coluna text onde são inseridos dados em formato json. Esses dados são os dados obrigatórios de qualquer operação

Dessa forma, independente se é apenas uma operação de listagem completa, que só necessita de ter preenchida a coluna com o nome do banco de dados e a coluna com o tipo de operação ou se for uma operação de update onde todos os campos podem estar preenchidos, o banco de dados sqlite consegue lidar de forma rápida e segura com qualquer uma das operações e dados gerados pelo algoritmo.

4 testes

Os testes foram realizados utilizando um Orangepi PC+ e um computador de mesa . O Orange Pi é um SBC ARM baseado no processador allwinner h3 com 3 usb 2.0 , 1GB de memória RAM DDR3, uma porta de rede 10/100 e wifi bgn. Essa configuração é relativamente antiga e seu processador é um 4-core de 1.3ghz no clock máximo,é um processador relativamente bom mas o conjunto de especificações não é bom o bastante para substituir um computador atual, devido a sua limitação de memória ram e de capacidade gráfica, mas consegue funcionar de forma satisfatória como servidor doméstico ,visto que seu processador é bom o bastante para operações simplificadas e poucos acessos,mas quando se tratam de muitos acessos ele pode não ser potente o bastante para aguentar.

O computador de mesa é um dual core intel com 4 gb de memória ram ddr3, 4 usb 2.0 porta de rede gigabit e algumas portas sata2,entretanto essas portas sta não serão usadas já que o propósito é manter as 2 maquinas o mais próximas em relação a hardwaree possivel Outros métodos que serão usados para manter as maquinas mais similares serão limitar o clock de ambos para 1.2ghz,que é o clock mais estável para o orange pi ,a memória usada será limitada a 750 mb para o stack do docker e tanto o sistema operacional quanto os dados do docker serão salvos em cartões de memória microsd classe 10 com limitação de acesso de 10 MB/s de escrita e leitura , no orangepi será usado o leitor da própria placa para conectar o microsd e no pc será usado um adaptador usb. alem disso serão usadas versões do sistema debian,no orangepi o armbian e no pc o prórpio debian padrão,ambos na versão buster

4.1 testes de tempo

Foi feito um teste aditivo que testará 30 ciclos com adição de 100 em 100 elementos e cada ciclo. Foram feitos um teste de 4 sub ciclos internos para cada um dos 30 ciclos, tendo como objetivo o quanto de tempo é gasto em média para casos com muitos ou poucos dados gerados, podendo assim gerar um valor de base de valor mínimo gasto obrigatóriamente para cada interação.

Os testes dessa vez foram modificados com o objetivo de salvar os dados em um arquivo

para consulta futura. Os testes dessa vez foram realizados em um pc arm64 e um amd64. Os dados resultaram em valores de tempo consistente em relação a diferença de frequências dos computadores. Sendo assim, se existe um valor de perda entre as arquiteturas para esse teste é um valor irrisório, sendo que o pc amd64 apresentou testes cerca de 2 vezes mais rápidos e seu clock é aproximadamente o dobro do arm64. Esses valores foram dados em relação ao tempo gasto por elemento em cada interação.

5 Metodologia

6 Resultados

6.1 Resultados do Método

7 Conclusão

Referências

Anexos

ANEXO A – logs teste de tempo