Estudo de Eficiência de Arquitetura Distribuída Utilizando Bancos de Dados Relacionais, Não-Relacionais e Cache em Memória

Josué G. Cardoso[1], Prof. Esp. Willen Leolatto Carneiro[1], Prof. Me. Márcio J. Sembay[2]

[1]Centro Universitário Unifacvest - Lages, SC - Brasil

[2]Universidade Federal de Santa Catarina (UFSC) - Florianópolis, SC - Brasil

{josue.cardoso.aluno, prof.marcio.sembay, prof.willenleolatto}
@unifacvest.edu.br

Abstract. *With the popularization of systems with distributed architecture, it is necessary to carry out comparative studies between technologies in order to implement efficient and quality software. This article presents a comparative study of approaches to data persistence using experiments in a controlled environment with an interest in evaluating the response time of each request. To carry out the experiment, three containerized applications were made using Docker so that it is possible to control the consumption of resources in these environments. Among the implemented approaches, the results showed that the use of the non-relational database together with the in-memory cache is significantly more efficient.*

Resumo. *Com a popularização de sistemas com arquitetura distribuída, se faz necessário realizar estudos comparativos entre tecnologias a fim de implementar softwares eficientes e com qualidade. Este artigo apresenta um estudo comparativo de abordagens para persistência de dados utilizando experimentos em ambiente controlado com interesse em avaliar o tempo de resposta de cada requisição. Para realizar o experimento foram utilizadas três aplicações conteinerizadas utilizando o docker de forma que seja possível controlar o consumo de recursos nesses ambientes. Entre as abordagens implementadas, os resultados apontaram que o uso do banco de dados não-relacional em conjunto do cache em memória é significativamente mais eficiente.*

1. Introdução

Os bancos de dados e os sistemas de bancos de dados se tornaram componentes essenciais no cotidiano da sociedade moderna. Diariamente a maioria de nós se depara com atividades que envolvem alguma interação com os bancos de dados [ELMASRI e NAVATHE 2011]. Dessa forma, a definição de qual tecnologia utilizar para persistência dos dados nos softwares é muitas vezes um problema enfrentado por profissionais responsáveis por realizar essa escolha.

Dentre as possibilidades de escolha para persistência dos dados, pode-se destacar três delas que serão objeto de estudo neste artigo: banco de dados relacionais, banco de dados não-relacionais e cache em memória, onde:

- Banco de dados relacionais: No modelo relacional existe a possibilidade de elaboração de um relacionamento lógico entre as informações[...] [KAUFELD 1996]. Todo banco de dados relacional é composto por tabelas que representam relações. Daí o nome relacional. Cada tabela é um conjunto não necessariamente ordenado de linhas ou tuplas [ANICETO e XAVIER 2014].
- Banco de dados não-relacionais: Uma definição de NoSQL é que ele é um conjunto de conceitos que permitem o processamento de dados de forma rápida e eficiente com um foco em performance [KELLY e MCCREARY 2013]. É um modelo alternativo pensado para se modelar os dados sem ter que seguir os padrões rígidos criados para o modelo relacional [ANICETO e XAVIER 2014].
- Cache em memória: O termo cache no contexto da computação refere-se a um repositório destinado à informação de uso frequente. Este repositório geralmente possui uma capacidade reduzida, mas em compensação um tempo de acesso menor se comparado à fonte original dos dados [BENÍTEZ 2014].

Ao mesmo tempo que a persistência dos dados é importante para uma aplicação, a qualidade do software também possui tal importância. Por consequência, foi desenvolvida uma norma ISO (ISO/IEC 25010:2011) para que fosse definido um conjunto de características a fim de atestar a qualidade de um software. As características e subcaracterísticas fornecem terminologia consistente para especificar, medir e avaliar a qualidade do sistema e do produto de software [ISO 2011].

O objetivo deste trabalho é avaliar a eficiência sob a ótica da qualidade de software (ISO/IEC 25010:2011), utilizando como métrica principal o tempo de resposta (em milissegundos) das requisições disparadas na aplicação e na disponibilidade baixa de recursos de hardware como um parâmetro secundário. A partir do tempo de requisição para realizar determinada ação será analisado e comparado com as demais tecnologias a fim de determinar qual das opções disponíveis apresenta menor tempo de requisição sem sobrecarregar os recursos de hardware disponibilizados.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta informações das tecnologias utilizadas para realizar o experimento, sobre o ambiente que foi utilizado para a execução do mesmo e uma breve explicação das aplicações implementadas. Na Seção 3 são detalhados os resultados obtidos e a discussão dos mesmos. Na Seção 4 são apresentadas algumas possibilidades de melhorias na pesquisa e estudos futuros. E, por fim, na Seção 5 apresenta a conclusão deste trabalho.

2. Experimento

Nesta seção serão descritas as tecnologias utilizadas no experimento, o ambiente onde o experimento foi desenvolvido, assim como os recursos disponibilizados para que cada aplicação seja executada de forma controlada. Também será detalhado como ocorreu a execução das aplicações e a coleta dos dados que serão apresentados na seção de resultados e discussão.

2.1. Arquitetura das aplicações

Para realizar o experimento, foram utilizadas três aplicações:

- Ferramenta de *stress test* ou stresser[1]: Aplicação desenvolvida em .NET Core, responsável por realizar diversas requisições simultâneas para a API a fim de simular a utilização da mesma por usuários e responsável também por coletar o tempo de cada requisição.
- API[2]: Aplicação desenvolvida em Javascript, responsável por gravar, ler e deletar os dados dos bancos de dados, expor endpoints para que a ferramenta de stress test pudesse realizar o experimento e posteriormente é onde o cache em memória será implementado.
- Banco de dados: É a aplicação onde são persistidos os dados. Foi utilizado o SQL Server e o MongoDB nesta camada para realizar o experimento.

Na Figura 1 é possível observar a arquitetura da ferramenta de *stress test*, onde ela consiste em executar 5 *threads* simultâneas, contendo 2.000 requisições cada uma das *threads*, totalizando 10.000 requisições em cada uma das abordagens do experimento. A ferramenta também é responsável por obter o tempo que cada requisição levou para ser executada e posteriormente gravar num arquivo .CSV, onde a partir deles foi possível gerar os gráficos da seção resultados e discussão.

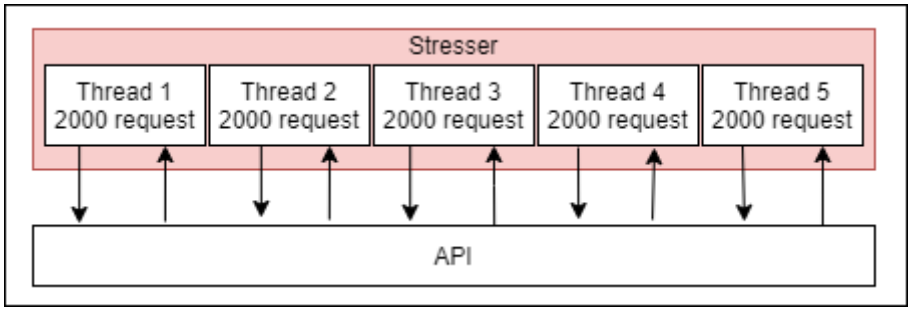

Figura 1. Representação da ferramenta de stress test

Na Figura 2 é apresentada a arquitetura das aplicações e o ciclo de vida para obter os dados da abordagem utilizando banco de dados relacional e não-relacional. O ciclo de vida da requisição consiste em 4 passos:

[1] https://github.com/JosueCardoso/stresser-persistence-methods-nodeJS

[2] https://github.com/JosueCardoso/Persistence-methods-nodeJS

1. Requisição é feita pela ferramenta de *stress test* no *endpoint* CreateAccount;
2. API guarda essa informação no banco de dados de acordo com a sua implementação (no banco de dados relacional é necessário inserir o registro numa tabela e no banco de dados não-relacional é necessário inserir um documento na coleção);
3. Após o registro ou o documento ser salvo no banco de dados é então retornado todos os dados do banco;
4. Todos os dados retornados do banco de dados são enviados como resposta da requisição feita no passo 1 e encerrado o temporizador que estava executando na ferramenta de *stress test*.

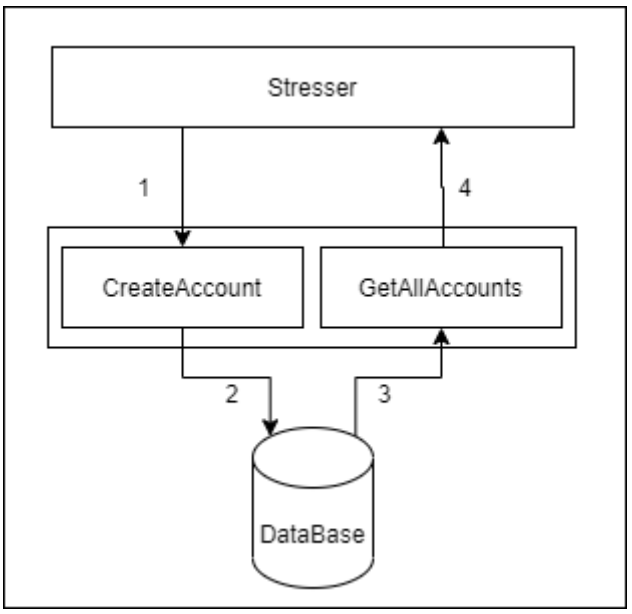

Figura 2. Representação dos endpoints e sua comunicação com o banco de dados

A Figura 3 apresenta a arquitetura das aplicações e o ciclo de vida para obter os dados da abordagem utilizando banco de dados relacional e não-relacional junto do cache em memória. O ciclo de vida da requisição continua tendo 4 passos e seguindo os mesmos preceitos da arquitetura apresentada anteriormente, diferenciando apenas nos passos 2 e 3, onde no passo 2 ao salvar o dado no banco de dados, o dado também é salvo no cache em memória e no passo 3, onde todos os dados são obtidos do cache em memória e não mais do banco de dados.

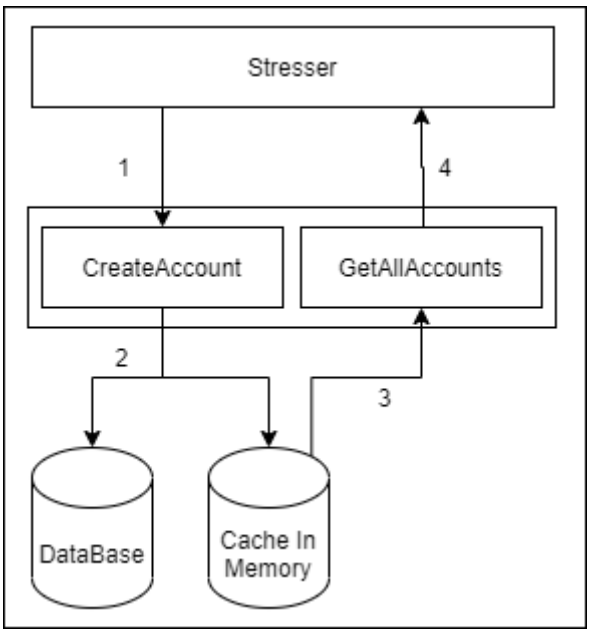

Figura 3. Representação dos endpoints e sua comunicação com o cache em memória e o banco de dados

2.2. Tecnologias

As aplicações executadas no experimento utilizam diversas tecnologias disponíveis no mercado. A escolha de cada uma delas se deu pelos seguintes fatores: conhecimento prévio dos autores, popularidade no mercado de tecnologia e proficiência para o experimento (como exemplo, o docker e o cgroups). No decorrer desta seção serão apresentadas as tecnologias utilizadas e uma breve explicação sobre as motivações das escolhas.

A ferramenta de *stress test* foi implementada com .Net Core. Tal escolha se deve ao fato dos autores terem domínio da plataforma, conhecimento sobre controle das múltiplas threads, escrita de arquivos de logs e consumo de APIs. Dessa forma, a ferramenta de *stress test* poderia ter sido desenvolvida em outra tecnologia.

A API, onde ocorrem as principais execuções do experimento, foi desenvolvida com NodeJS, as bibliotecas ExpressJS para criar a aplicação web e NodeCache para implementar o cache em memória na camada de persistência. As tecnologias escolhidas para a implementação da API se devem ao fato da popularidade no mercado de TI mundial [SLASHDATA 2021], fazendo com que fosse possível encontrar materiais que auxiliassem no desenvolvimento.

Na camada de persistência, foram utilizados dois bancos: um relacional (Sql Server) e um não-relacional do tipo documento (MongoDB) para realizar um comparativo entre essas duas tecnologias.

Por fim, para que fosse possível controlar o ambiente onde as aplicações estavam sendo executadas e monitorar o consumo de recursos, foi utilizada a estratégia de conteinerização. Para que pudesse tirar maior proveito dessa estratégia, o docker é a principal tecnologia quando se fala em contêineres [STACK OVERFLOW 2020]. Ao utilizar docker com imagens construídas baseadas no kernel do Linux é possível utilizar o recurso chamado cgroups (grupos de controle), para que seja feito a limitação da disponibilidade de recursos de hardware para cada um dos contêineres.

2.3. Ambiente do experimento

O experimento foi executado em um computador com as seguintes especificações:

- Processador: AMD Ryzen 7 1700, 8 núcleos de 3.0 GHz (3.7 GHz Max Boost);
- Memória RAM: HyperX Fury 2x8GB (16GB) 2666MHz, DDR4;
- Armazenamento: SSD WD Green, 480GB, Leitura 545MB/s, Gravação 430MB/s;

Para obter os resultados foi necessário utilizar o docker para simular um ambiente com poucos recursos disponíveis, de modo que fosse possível acompanhar o uso desses recursos e garantir que eles sejam melhor utilizados. A seguir, na Tabela 1 serão apresentados os recursos disponibilizados utilizando o cgroups para cada contêiner durante a execução dos testes.

Tabela 1. Disponibilidade de recursos nos contêineres

Contêiner	**CPU**	**RAM**
Aplicação sem cache	1 Núcleo	256MB
Aplicação com cache	1 Núcleo	256MB
Sql Server	1 Núcleo	2GB*
MongoDB	1 Núcleo	512MB

Para criar uma instância do contêiner onde o Sql Server está executando é necessário disponibilizar no mínimo 2GB de memória RAM. Não é possível subir um contêiner com menos que isso.

2.4. Execução e coleta de dados

A ferramenta de *stress test*, responsável por gerar as 10.000 requisições em cada uma das abordagens utilizando 5 threads simultâneas, também foi responsável por armazenar o tempo que cada uma das requisições levou para executar todo o ciclo de vida do dado.

Da mesma forma, a ferramenta também foi responsável por salvar o tempo das requisições em arquivos .CSV, para que pudessem ser analisados posteriormente.

Para garantir a validade dos dados obtidos em cada requisição, foram realizadas 3 execuções para cada abordagem, onde em cada uma das execuções os bancos de dados eram limpos e as aplicações reiniciadas. Dessa forma, foram necessárias 12 execuções ao todo (3 execuções para cada uma das 4 abordagens).

A partir das execuções, foram gerados diagramas de caixa a fim de observar a homogeneidade das execuções dentro da mesma abordagem. Dessa forma, foi possível constatar que as execuções não tinham irregularidades. Por conseguinte, foi calculada a média das 3 execuções da mesma abordagem com o propósito de realizar uma comparação entre as 4 abordagens dispostas no experimento.

3. Resultados e discussão

Sabendo-se que foram realizadas 3 execuções para cada uma das 4 abordagens utilizadas no experimento e que foi calculado uma média para essas 3 execuções, a Figura 4 apresenta o comparativo destas médias entre cada uma das abordagens.

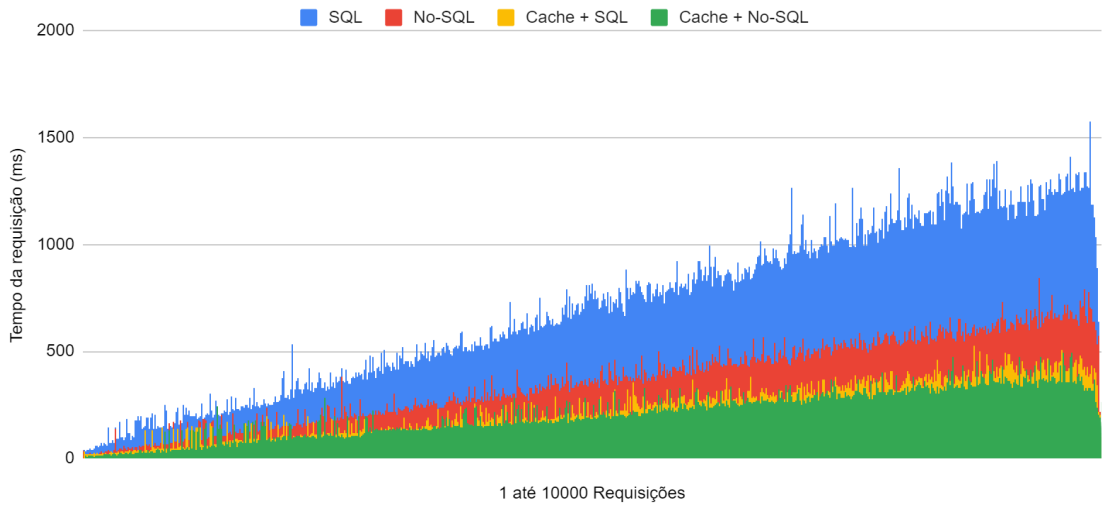

Figura 4. Gráfico de linha para comparativo das abordagens utilizadas na persistência dos dados

Pode-se observar que o tempo de resposta das requisições utilizando como persistência de dados apenas o banco de dados relacional (Sql Server) foi consideravelmente mais elevado que as demais abordagens.

Em contrapartida, o tempo de resposta das requisições utilizando como persistência de dados o banco de dados relacional (Sql Server) e o cache em memória, demonstrou ser uma abordagem mais eficiente, sendo a melhor opção até mesmo em comparação a utilização de apenas o banco de dados não-relacional (MongoDB). Deste modo, caso exista a escolha entre utilizar ou não o cache, a melhor opção é utilizar.

Um ponto interessante, é a notória diferença entre utilizar cache em memória para a leitura dos dados, independentemente de qual banco de dados é utilizado. Este ponto demonstra que uma alteração estratégica no ciclo de vida do dado dentro de uma arquitetura de software é capaz de gerar bons resultados do ponto de vista da eficiência na qualidade de software.

Outro ponto observado, é a sutil diferença entre a utilização do banco de dados relacional (Sql Server) e não-relacional (MongoDB), tendo o cache em memória incluso no ciclo de vida do dado. Desta forma, caso o software tenha sido implementado empregando um banco de dados relacional, dentro das mesmas condições apresentadas no experimento, não é necessário alterar para um banco de dados não-relacional.

4. Melhorias de pesquisa e trabalhos futuros

Este trabalho teve como objetivo abordar apenas o tempo de resposta das requisições em diferentes abordagens de persistência de dados sob condições de baixo consumo de recursos. Porém, existem outros dados que poderiam ser obtidos, adicionar mais condicionais nos ambientes, implementar padrões arquiteturais ou até mesmo discorrer sobre teoremas relacionados a persistência de dados.

Infelizmente abordar todos esses assuntos neste trabalho o tornaria muito extenso. Dessa forma, será realizado um breve esclarecimento de como esses pontos podem acrescentar neste estudo:

- Utilizar objetos mais complexos: Diferente do objeto utilizado para estudo na persistência dos dados (objeto conta, onde possui 4 propriedades e nenhum relacionamento com outros objetos), seria interessante utilizar outros objetos a serem persistidos, como por exemplo: usuário, endereço e telefone. Dessa forma, seria possível observar mais a fundo a distinção entre os bancos relacionais e não relacionais.
- Adicionar tarefas complexas: No momento, a API expõe dois endpoints para cada abordagem: um para a criação da conta e outro para deletar todos os dados persistidos. Deste modo, não existe complexidade para realizar essas tarefas. Seria interessante implementar segurança para armazenar senhas (*salt*+*hash*), validar os valores que são passados como parâmetros nos endpoints e criar os relacionamentos via código dos objetos como foi sugerido no parágrafo anterior.
- Obter o consumo de recursos dos contêineres por requisição: Com igualdade a observação elaborada no tempo de resposta da requisição, com a utilização do docker e o cgroups é possível auferir o quanto as aplicações estão utilizando dos recursos disponibilizados. Consequentemente esses dados seriam empregados numa projeção com o propósito de determinar quanto de recurso será necessário disponibilizar caso queira conservar-se a eficiência do software.
- Discorrer sobre teorema CAP: Cada uma das abordagens de persistência de dados objetos de estudo neste trabalho possuem características que são explicadas pelo teorema CAP, que por sua vez, indica quão consistente, disponível ou tolerante a particionamento é um sistema de arquitetura

distribuída. Em vista disso, seria benéfico discorrer sobre cada uma das abordagens pelo ponto de vista deste teorema.

5. Considerações finais

Com a popularização de aplicações utilizando arquitetura distribuída, tem sido necessário pensar no tempo em que o usuário fica aguardando até que sua requisição seja processada e os resultados esperados sejam apresentados a ele. Igualmente é importante colocar no mesmo cálculo o mínimo consumo de recursos para que o menor tempo de requisição seja alcançado.

Para o deleite de todos os envolvidos no processo (desde usuários, desenvolvedores e etc), a engenharia de software tem trabalhado para que características da qualidade de software como a eficiência tornem-se cada vez mais acessíveis.

Este trabalho teve como objetivo demonstrar que sutis mudanças na arquitetura de um software traz resultados consideráveis. Contudo, as implementações e tecnologias utilizadas foram apenas para experimentação, sendo necessário considerar os pontos levantados na seção de melhorias de pesquisa e trabalhos futuros e também considerar que cada software possui suas próprias características e requisitos, de forma que a melhor abordagem para um software pode não ser a melhor abordagem para outro.

Referências

Aniceto, Rodrigo e Xavier, Renê. (2014). Um estudo sobre a utilização do banco de dados NoSQL Cassandra em Dados Biológicos.

Benítez, Nelson Rodrigo Pérez. (2014). Sistema de memória cache de vídeo segmentado para protocolo http dinâmico.

Elmasri, Ramez e Navathe, Shamkant B. (2011). Sistemas de Banco de Dados.

ISO/IEC 25010:2011. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

Kaufeld, J. (1996). Access 95 para Windows para leigos: Um manual para novos usuários. Ludemir, J. São Paulo: Berkeley Brasil.

Kelly, Ann e McCreary, Dan. (2013). Making Sense of NoSQL : A Guide for Managers and the Rest of Us by Ann Kelly and Dan McCreary. Manning Publications Company.

Silva, Gilmar José da e Ferreira, Júlio Cesar Oliveira. (2017). Análise comparativa de desempenho de consultas entre um banco de dados relacional e um banco de dados não relacional.

SlashData. (2021). State of the developer nation. 21th Edition. Disponível em <https://slashdata-website-cms.s3.amazonaws.com/sample_reports/_TPqMJKJpsfPe7ph.pdf>. Acesso em: 24 de nov. de 2021.

Stack OverFlow. (2020). 2020 Developer Survey. Disponível em <https://insights.stackoverflow.com/survey/2020#technology-platforms-all-respondents5>. Acesso em: 24 de nov. de 2021.