Estudo Comparativo de Bancos de Dados NoSQL Distribuídos

Kaique Juvencio Costa, Kalvin Vinicius Vasconcellos dos Santos, Roger Thencerá Rojas, Wendel Sergio Duarte Junior, Calebe de Paula Bianchini

Faculdade de Computação e Informática – Universidade Presbiteriana Mackenzie

(UPM) Caixa Postal - 01302-907 – São Paulo – SP – Brazil

kaiquejuvenciocosta@gmail.com, kalvin.vasconcellos@gmail.com,

rothero@gmail.com, wendel_duarte@outlook.com,

calebe.bianchini@mackenzie.br

Abstract. *With the growth of NoSQL databases on the market and the great amount of data in today's digital world, the amount of data often exceeds the computing power of one machine, which makes it necessary to utilize a distributed database system. In this scenario, many IT professionals don't know which database is the best option when starting a new project. This study has the objective of making a performance evaluation and comparison between three popular NoSQL databases (Redis, MongoDB, and Cassandra), distributing data across three distinct nodes that are subject to specific scenarios of data processing (update, read, and write) so that the IT community can have a better understanding of those technologies.*

Resumo. *Com o crescimento dos bancos de dados NoSQL no mercado e a grande quantidade de dados no mundo digital de hoje, essa quantidade de dados muitas vezes excede o poder de processamento de uma máquina sendo necessário utilizar sistemas de banco de dados distribuídos. Nesse cenário, muitos profissionais de tecnologia ao iniciar um novo projeto não sabem qual a melhor opção de banco escolher. Este estudo tem por objetivo comparar o desempenho de três bancos de dados NoSQL (Redis, MongoDB e Cassandra), distribuindo seus dados em três máquinas e em cenários específicos de processamento de dados* (atualização, leitura e escrita)*, visando auxiliar a comunidade a ter um melhor entendimento sobre essas tecnologias.*

1. Introdução

Na era da informação, aplicações e sistemas de *software* estão cada vez mais lidando com grandes volumes de dados, tornando necessário a construção de arquiteturas de *software* robustas e escaláveis. Entre as principais necessidades que as empresas de tecnologia enfrentam, destacam-se duas: suportar um alto tráfego de informações de usuários simultaneamente utilizando um serviço e a capacidade de responder às interações dos usuários rapidamente, através de bases de dados geograficamente próximas ao cliente final [Abadi 2012].

Mediante a essas necessidades, o paradigma do NoSQL ganhou força como uma alternativa escalável ao modelo relacional de dados. O termo NoSQL foi inicialmente introduzido em 1998, por Carlo Strozzi para se referir a um banco de dados de código aberto desenvolvido por ele. Nos últimos anos, o termo se refere a uma categoria de bancos de dados que não garante as propriedades ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) e que são facilmente escaláveis de forma horizontal e distribuída. Esses bancos estruturam dados de diferentes formas, como orientados a documentos, chave-valor e grafos [Berg, Seymour e Goel 2013].

Atualmente, há uma grande variedade de bancos NoSQL distribuídos. Cada tecnologia segue um tipo de implementação, o que favorece diferentes casos de uso e desempenhos. Por exemplo, algumas tecnologias favorecem grandes volumes de operações de escrita simultânea, enquanto outras favorecem grandes volumes de leituras. A escolha adequada de uma determinada tecnologia NoSQL pode determinar o sucesso de um sistema ou produto.

Existem poucos estudos a respeito do tema na língua portuguesa, especialmente se aprofundando no teor comparativo de exemplos modernos de banco de dados NoSQL distribuídos. Com base nessas informações o objetivo do estudo é investigar e comparar o nível de performance de três bancos de dados não relacionais distribuídos em diferentes cenários de inserção, atualização e leitura de dados junto a diferentes cargas de dados, visando auxiliar a comunidade tecnológica a ter um maior entendimento sobre qual a melhor opção para seu projeto de *software*.

2. Desafios de bancos de dados distribuídos

Sistemas de banco de dados distribuídos são definidos como uma coleção de banco de dados rodando em máquinas independentes que estão interligadas em uma rede distribuída. Um nó dessa rede não tem acesso ao estado de outro nó, havendo apenas troca dessas informações entre nós através de mensagens enviadas entre si [Özsu 1991].

A noção de bancos de dados distribuídos ganhou força com o movimento NoSQL. A complexidade de conjuntos de dados desestruturados e o crescente volume de dados são fatores que podem ser considerados como razões para o mercado ter adotado esse novo modelo de dados [Fahd, Venkatraman, e Kaspi 2016]. Bancos de dados NoSQL podem ser classificados de forma geral em quatro categorias: orientados a chave-valor, orientados a coluna, orientados a documento e orientados a grafo [Fahd et al. 2016]. Cada um desses modelos de dados se diferenciam em aspectos de arquitetura, adotando propriedades específicas do teorema CAP (acrônimo de Consistência, Disponibilidade, Partição). O teorema CAP foi proposto, em 2000, na obra "*Towards robust distributed systems*", por Eric Brewer, Professor Doutor da Universidade da Califórnia, que veio se tornar base para auxiliar engenheiros de bancos de dados na concepção de novos projetos.

O teorema CAP afirma que qualquer base de dados distribuída pode ter no máximo duas das três seguintes propriedades: Consistência, Disponibilidade e Partição [Brewer 2000]. Seguindo a definição do autor:

- *Consistency*: diz respeito a um sistema ter a mesma versão dos dados atualizada em todas as máquinas pertencentes à uma rede distribuída [Brewer 2000].

- *Availability*: se refere à capacidade de um sistema responder todas as requisições que recebe, mesmo que algum nós não esteja funcionando normalmente [Brewer 2000].
- *Partition tolerance*: se refere ao sistema ser dividido em partições (pedaços) e ser capaz de tolerar eventuais falhas, ou seja, continuar funcionando normalmente mesmo que uma partição do sistema se torne indisponível e incomunicável devido a algum erro [Brewer 2000].

Além do teorema CAP, outro desafio na arquitetura de bancos de dados distribuídos é a garantia das propriedades ACID (*Atomicity, Consistency, Isolation, Durability*), como citado na seção 1, e BASE (*Basically, Available, Soft state, Eventual consistency*).

Os bancos de dados NoSQL não garantem as propriedades ACID, devido à falta de garantia de consistência em todos os momentos, favorecendo a constante disponibilidade de dados. Em comparação, os bancos de dados SQL garantem as propriedades ACID, pois favorecem uma forte consistência acima da disponibilidade. Nesse cenário, as informações sobre uma atualização de um dado no sistema devem se espalhar de forma imediata, caso contrário pode se criar uma inconsistência de dados [Sharma e Dave 2012]. Em detalhes, as propriedades ACID são definidas da seguinte forma:

- *Atomicity*: ou executa o fluxo por completo ou não executa nenhuma ação. Por exemplo, quando é feita uma transferência bancária, o dinheiro não pode sair de uma conta e não entrar na outra. É preferível que não execute nenhuma ação do que acontecer no cenário citado, ou seja, o fluxo não pode ser executado pela metade.
- *Consistency*: uma única visão dos dados em vários pontos. No exemplo de uma transferência bancária, o valor transferido tem que entrar em débito em uma conta ao mesmo tempo que entra como receita em outra conta.
- *Isolation*: verifica ações que não podem acontecer ao mesmo tempo, que influenciam em outras ou que entram em conflito. Por exemplo, duas pessoas não podem marcar médico no mesmo horário mesmo que realizem a ação de marcar consulta exatamente ao mesmo tempo na aplicação.
- *Durability*: não voltar atrás em operações já realizadas no banco de dados.

Ao contrário do ACID, o BASE tem como foco principal a disponibilidade permanente [Sharma e Dave 2012], e pode ser definido da seguinte forma:

- *Basically Available*: com o foco na disponibilidade contínua, o sistema continua acessível mesmo no caso de alguns nós sofram de alguma falha.
- *Soft state*: os dados contidos no sistema podem eventualmente expirar caso não ocorra nenhuma atualização, ou seja, os dados não necessariamente persistem indeterminadamente.
- *Eventual consistency*: não é totalmente consistente, ou seja, é possível que uma mesma chave tenha valores distintos em diferentes nós ao mesmo tempo, pois a propagação da atualização é feita de forma assíncrona.

Dessa forma, bancos de dados distribuídos distintos acabam priorizando certas propriedades de acordo com o seu propósito, como será abordado nos experimentos.

3. Experimentos

Para a realização deste estudo foram escolhidos três bancos de dados não relacionais, sendo eles: Redis, MongoDB e Cassandra. A escolha de cada banco para análise foi baseada nas características de cada um. Primeiramente foi analisado o modelo de dados, para que houvesse bancos com diferentes tipos de modelagem - como será descrito a seguir - o Redis possui um modelo de chave-valor, o Cassandra é baseado em colunas e o MongoDB é orientado a documentos. Outro fator que levou a escolha do Redis é que o armazenamento de seus dados é feito em memória principal, sendo assim, é interessante entender seu funcionamento em relação a bancos que armazenam seus dados em disco. Com relação ao Cassandra, sua história se destaca, uma vez que foi criado pelo Facebook para lidar com grandes volumes de dados, além de ser um dos bancos NoSQL mais utilizados hoje em dia [Lourenço, Cabral e Carreiro 2015]. O MongoDB foi escolhido por também ser um dos mais populares quando se diz em bancos não relacionais [Lourenço, Cabral e Carreiro 2015] e também por ser possível uma estratégia de distribuição de dados diferente da estratégia dos outros dois bancos escolhidos. Assim, este artigo reúne bancos com diferentes características para comparação. A seguir, será feita uma breve descrição sobre o funcionamento de cada banco de dados.

O Redis (abreviação para *Remote Dictionary Server*) é um banco de dados NoSQL orientado a chave-valor, no qual se cria e gerencia matrizes de dados, semelhantes à estrutura de um dicionário ou tabela *hash*. Foi inicialmente criado em 2009 por Salvatore Sanfilippo, e atualmente é mantido pela empresa de mesmo nome, Redis. Uma de suas principais características que tornou o banco conhecido é o armazenamento dos dados ser feito utilizando memória principal (RAM), ao invés de memória em disco, como é comum em outros bancos de dados NoSQL. Para evitar possíveis perdas de dados, Redis também suporta salvar *backups* dos dados em disco periodicamente. Por fim, assim como em outros bancos NoSQL, o Redis também suporta um sistema de replicação de dados, permitindo que outras máquinas contenham os dados em redundância e para que possam ser utilizadas para operações de leitura, enquanto o servidor mestre fica responsável pelas operações de escrita [Carlson 2013].

Já o Cassandra é um banco de dados NoSQL distribuído de código aberto, inicialmente desenvolvido pelo Facebook e atualmente mantido pela Apache Software Foundation, criado com o objetivo de atender o grande volume de dados gerado pelo *chat* do Facebook, e mantendo um baixo tempo de resposta em buscas por palavras chaves. Seu modelo de dados é orientado a colunas e teve como inspiração a modelagem de dados usado pelo banco distribuído Bigtable, criado pela Google [Lakshman e Malik 2010]. Dois conceitos fundamentais na arquitetura do Cassandra são particionamento e replicação. A ideia do particionamento é permitir que um *cluster* do Cassandra seja capaz de escalar de forma incremental à medida que novos nós sejam adicionados, permitindo que um grande volume de dados seja dividido entre diferentes máquinas. Além disso, com o objetivo de ser altamente disponível, essa tecnologia utiliza a replicação, armazenando dados de forma redundante em um número de máquinas definido em suas configurações. Nessa arquitetura, cada nó do Cassandra é capaz de receber requisições de escrita e leitura. Uma requisição de escrita é recebida

por um nó e a seguir é roteada para outros nós, de forma que pelo menos um número mínimo de nós confirme o processamento da operação [Lakshman e Malik 2010].

Por fim, o MongoDB é um banco de dados NoSQL orientado a documentos, inicialmente lançado em 2009 pela empresa MongoDB Inc. Sua flexibilidade permite que diferentes tipos de dados não estruturados sejam armazenados na mesma coleção, substituindo o modelo tradicional de banco SQL que trabalha com tabelas. Assim como na arquitetura do Cassandra, o MongoDB realiza uma estratégia de replicação, permitindo que múltiplas cópias de um mesmo dado seja mantido em diversas máquinas para fins de redundância. Em seu sistema de replicação, o MongoDB utiliza o conceito de *replica set*, o qual se consiste em um grupo de servidores, sendo um servidor primário que aceita requisições de clientes, e múltiplos servidores secundários que mantém cópias de todos os dados. Se houver alguma falha no servidor primário, os outros servidores elegem um novo servidor primário [Chodorow 2010]. Por fim, além da replicação, o MongoDB também suporta a estrutura de *sharding*, permitindo que diferentes máquinas contenham parte dos dados ao invés de replicar todos os dados de uma vez. Nessa configuração, os dados são divididos em pedaços, e cada pedaço fica armazenado em uma máquina distinta de um *cluster*. Para um cliente conseguir buscar um dado específico armazenado em alguns dos *shards*, o cliente se conecta em um servidor roteador, que contém a informação de em qual *shard* o dado buscado se encontra [Chodorow 2010].

Cada banco de dados foi distribuído em três máquinas. O Cassandra e o Redis utilizaram um sistema de replicação, onde os dados foram persistidos nas três máquinas, já o MongoDB utilizou um sistema de distribuição de dados de forma que temos uma máquina principal (servidor de configuração), onde existe toda a configuração do servidor distribuído, junto à esse servidor existe um *router*, rodando no mesmo nó do servidor de configuração (este *router* é chamado de mongos), o *router* irá receber todas as requisições feitas ao banco de dados e por sua vez irá direcionar a requisição ao nó 2 ou 3. Não ocorrerá persistência de dados no servidor de configuração do MongoDB, sendo assim teremos as mesmas configurações de hardware para os três bancos de dados, porém o MongoDB utilizará uma máquina inteiramente para configuração do servidor.

Outra escolha importante para este estudo foi a ferramenta de *benchmark*. Hoje já existem diversas ferramentas que podem ser utilizadas com o objetivo de medir o desempenho de um banco de dados. Ao escolher uma ferramenta, era necessário que a mesma já tivesse sido amplamente utilizada e testada para que os resultados fossem mais confiáveis, o que levou o estudo à ferramenta Yahoo! Cloud Serving Benchmark. Existem diversas pesquisas, tanto recentes como mais antigas, o que mostra que a ferramenta já é consolidada no mercado sendo esse um fator crucial para sua escolha [Klein, Gorton, Ernst, e Danohoe 2015].

Como dito acima o Yahoo! Cloud Serving Benchmark (YCSB) é uma ferramenta de *benchmark*. O YCSB possui código aberto e é utilizado para medir o desempenho de bancos de dados, criado em 2010 pelo Yahoo!. O que motivou a criação dessa ferramenta foi facilitar comparações de desempenho entre serviços na nuvem e a crescente quantidade de bancos NoSQL no mercado, o que fazia com que empresas tivessem muitas dificuldades para definir qual era o melhor sistema de banco de dados

para sua aplicação [Cooper, Silberstein, Tam, Ramakrishnan e Sears 2010]. O YCSB é uma aplicação escrita em java, onde é possível se conectar com um banco de dados, carregar os dados para os testes e gerar as operações através de *workloads* (definição de *workload* no parágrafo seguinte). A aplicação pode criar múltiplas *threads* de acordo com a configuração do teste a ser realizado, onde cada *thread* realiza múltiplas operações, se comunicando diretamente com a camada de interface do banco de dados. As *threads* medem as métricas e enviam essas informações ao módulo de estatísticas, ao final do teste o módulo agrega as informações devolvendo os resultados para o usuário [Cooper, Silberstein, Tam, Ramakrishnan e Sears 2010].

A escolha de como e quais testes serão realizados é outra definição importante para o estudo. Um *workload* traz essa definição especificando de forma mais detalhada cada teste. A especificação de um *workload* se dá através de um arquivo que será lido pelo YCSB no momento da realização dos testes, esse arquivo contém as instruções de quais testes devem ser feitos no banco de dados. É nesse arquivo que é especificado qual tipo de operação será realizada no banco de dados, como por exemplo: operação de escrita ou leitura.

Para esta pesquisa foram escolhidos três tipos diferentes de *workloads*, o primeiro é chamado de *update heavy*, onde a quantidade de operações é dividida igualmente entre leitura e escrita, o segundo é chamado de *read only*, neste caso 100% das operações são de leitura, já o terceiro e último é o *write only*, que contém 100% das operações sendo de escrita.

Cada *workload* (*update heavy*, *read only* e *write only*) foi testado três vezes, cada uma com uma quantidade diferente de operações, as três quantidades foram: cem mil, um milhão e cinco milhões. O objetivo era de haver vários cenários, para que o comportamento dos bancos fosse observado em diferentes situações de processamento de dados.

3.1 Execução dos testes

Para a realização dos testes foram utilizadas máquinas do ambiente *MackCloud* do Mackenzie, cada máquina continha dois processadores Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 128 GB de memória RAM e 82 TB de *storage*.

Sobre as versões das ferramentas utilizadas para os testes foram:

- CentOS Linux 7 (Core)
- Redis: 6.2.6
- Cassandra: 3.11.9
- MongoDB: 5.0.2
- YCSB: 0.15.0

Como o estudo utiliza uma estratégia de banco de dados distribuídos em diversas máquinas, também é interessante entender e analisar o quanto a velocidade de conexão entre as máquinas interfere para cada banco de dados. Pensando nisso, cada teste deveria rodar em dois cenários diferentes, o cenário 1 utilizaria uma banda de internet igual a 1Gb-*ethernet* e o cenário 2 com 40Gb-*infiniband*.

Cada rodada de testes foi realizada individualmente, nenhum teste foi rodado em paralelo para que os recursos de cada máquina estivessem alocados, exclusivamente,

para a realização dos testes. Também foi definido que cada banco de dados utilizaria exatamente três máquinas para fazer sua replicação ou distribuição de dados. O YCSB permite a execução de testes com diversas *threads*, porém os testes foram executados com apenas uma *thread*. Duas máquinas rodavam somente o banco de dados, enquanto a terceira máquina rodava o banco e a ferramenta de *benchmark*. Ao final de cada teste, os bancos já possuíam muitos dados persistidos em sua base, por este motivo, sempre antes de rodar um novo *workload* todos os dados inseridos no último teste eram apagados para garantir que os próximos não seriam afetados. Sendo assim, para a configuração do ambiente e realização dos testes era necessário seguir as especificações abaixo:

- Os bancos deveriam utilizar as mesmas máquinas para realização dos testes;
- Cada banco utilizaria, exatamente, três máquinas para distribuição ou replicação dos dados;
- A configuração do número de *threads* para enviar requisições aos bancos deve ser 1;
- As máquinas não devem estar executando outros processos não essenciais (processos que não são necessários para o funcionamento do sistema operacional, dos bancos e da ferramenta de *benchmark*) durante a realização dos testes;
- Ao fim de cada teste, os dados gerados pelos mesmos devem ser apagados, dessa forma não afetarão os testes seguintes.

Cada *workload* deveria ser testado três vezes, e cada rodada com uma quantidade diferente de operações, totalizando 18 testes para cada banco e 54 testes no total da pesquisa.

Após a realização de um teste, o YCSB retorna um arquivo contendo diversas métricas e resultados, para o presente artigo as métricas a serem analisadas serão:

***Throughput(ops/sec)*:** Esse valor diz respeito ao número médio de operações realizadas por segundo durante a execução de um *workload*. Essa é a principal métrica de análise do estudo.

Average Latency(us): É a latência média de cada operação em microssegundos realizada pelo teste, representando o tempo médio de resposta de cada operação. No *workload update heavy*, por exemplo, essa métrica tem dois valores: um para operações de leitura e outro para operações de atualização.

4. Resultados

Após a execução dos *workloads* e da coleta dos dados, foram realizadas diversas comparações entre o desempenho alcançado para cada um dos bancos de dados escolhidos. Essa seção contém a apresentação desses dados coletados, assim como análises dos resultados.

As comparações de desempenho incluem o *throughput* alcançado por cada banco, em cada *workload*, e a latência média gasta entre as operações executadas. Além disso, as comparações dos testes foram também feitas para ambientes com diferentes larguras de banda, sendo um de 1Gb-*ethernet* e o segundo com 40Gb-*infiniband*.

4.1 Throughput

A Tabela 1 contém os dados coletados do Redis, para cada *workload*, considerando dois ambientes de testes: 1Gb-*ethernet e* 40Gb-*infiniband.* A métrica sendo comparada é o *throughput* que o Redis realizou com a carga de dados de cinco milhões de linhas inseridas. Cada linha da tabela 1 representa um tipo de *workload* realizado, enquanto cada coluna mostra o ambiente utilizado. A última coluna mostra o resultado da comparação em porcentagem, informando a porcentagem em que o cenário de 40Gb-*infiniband* é mais rápido que o cenário 1Gb-*ethernet*.

Tabela 1. Comparação do *throughput* entre os cenários com 40Gb-*infiniband* versus 1Gb-*infiniband* do Redis por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões de linhas inseridas.

Redis			
	40Gb Infiniband (em ops/sec)	1Gb Ethernet (em ops/sec)	Diferença de Performance (em %)
Workload Update Heavy	25.761,77	20.863,75	23,48%
Workload Read Only	28.904,57	25.000,12	15,62%
Workload Write only	8.779,84	7.767,89	13,03%

Já a Tabela 2 apresenta os dados coletados do Cassandra e, assim como na Tabela 1, contém o desempenho alcançado por esse banco em relação a cada *workload* executado para cada ambiente de teste. A métrica sendo considerada é novamente o número de operações por segundo, considerando a carga de dados de cinco milhões de linhas inseridas.

Tabela 2. Comparação do *throughput* entre os cenários com 40Gb-*infiniband* versus 1Gb-*infiniband* do Cassandra por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões de linhas inseridas.

Cassandra			
	40Gb Infiniband (em ops/sec)	1Gb Ethernet (em ops/sec)	Diferença de Performance (em %)
Workload Update Heavy	1.604,23	1.529,53	4,88%
Workload Read Only	1.372,10	1.192,79	15,03%
Workload Write only	2.793,92	2.090,57	33,64%

Por fim, a Tabela 3 contém os dados coletados do MongoDB. Assim como nas Tabela 1 e 2, os dados se referem a comparação dos dois ambientes de larguras de banda distintas, para cada *workload*, considerando a métrica do *throughput*.

Tabela 3. Comparação do *throughput* entre os cenários com 40Gb-infiniband versus 1Gb-infiniband do MongoDB por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões.

MongoDB			
	40Gb Infiniband (em ops/sec)	1Gb Ethernet (em ops/sec)	Diferença de Performance (em %)
Workload Update Heavy	502,05	478,09	5,01%
Workload Read Only	892,56	780,67	14,33%
Workload Write only	467,21	416,95	12,05%

Em relação às três tabelas anteriores, é possível notar que, quando temos uma conexão mais veloz (*Infiniband*), os bancos distribuídos obtiveram um aumento significativo de operações por segundo em relação a conexão mais lenta (*Ethernet*), onde o MongoDB obteve seu maior ganho de desempenho na operação de leitura (*read*

only) com um aumento de 14,33%, enquanto o Redis conseguiu um aumento de 23,48% na operação de atualização de dados (*update heavy*) e, por fim, o Cassandra chegou a 33,64% na operação de escrita (*write only*), sendo o banco que mais ganhou performance com o aumento de velocidade de internet.

Uma vez realizada a análise entre os dois cenários de largura de banda, foi feita uma comparação entre os bancos utilizando o *throughput*. Assim como foi feito acima, aqui serão apresentadas mais três tabelas, cada uma comparando um banco com o outro. Para isso, foi levado em conta os três tipos de *workload* (representados por cada linha da tabela) e a métrica utilizada também foi o número de operações por segundo das cargas de dados de cinco milhões. A última coluna mostra os resultados em porcentagem, informando quantos por cento o banco A (banco representado pela segunda coluna) foi mais rápido que o banco B (representado pela terceira coluna).

Como é mostrado na Tabela 4, o desempenho de operações por segundo do Cassandra foi significativamente maior que o do MongoDB, atingindo uma diferença de quase 500% na operação de escrita (*Write Only).*

Tabela 4. Comparação do *throughput* entre o Cassandra versus MongoDB por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões no cenário 40Gb-*Infiniband*.

Cassandra X MongoDB			
	Cassandra	**MongoDB**	**Diferença de Performance (em %)**
Workload Update Heavy	1.604,23	502,05	219,54%
Workload Read Only	1.372,10	892,56	53,73%
Workload Write only	2.793,92	467,21	498,00%

Analisando os dados do Redis e Cassandra apresentados na Tabela 5, a diferença de desempenho fica mais explícita, onde o Redis consegue chegar a uma diferença de desempenho de mais de 2000% comparada ao Cassandra.

Tabela 5. Comparação do *throughput* entre o Redis versus Cassandra por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões no cenário 40Gb-*Infiniband*.

Redis x Cassandra			
	Redis	**Cassandra**	**Diferença de Performance (em %)**
Workload Update Heavy	25.761,77	1.604,23	1505,87%
Workload Read Only	28.904,57	1.372,10	2006,59%
Workload Write only	8.779,84	2.793,92	214,25%

Analisando os dados do Redis e MongoDB presentes na Tabela 6, o MongoDB foi menos performático em relação ao Redis, sendo que essa diferença de desempenho chega a ser maior de 5000%, um resultado extremamente significativo.

Tabela 6. Comparação do *throughput* entre o Redis versus MongoDB por *workload*. A comparação levou em conta apenas as cargas de dados de cinco milhões no cenário 40Gb-InfiniBand.

Redis x MongoDB			
	Redis	**MongoDB**	**Diferença de Performance (em %)**
Workload Update Heavy	25.761,77	502,05	5031,32%
Workload Read Only	28.904,57	892,56	3138,39%
Workload Write only	8.779,84	467,21	1779,21%

A seguir serão mostrados alguns gráficos e análises referentes aos resultados obtidos na pesquisa. Todos os gráficos seguem um mesmo padrão, o eixo Y é o valor de uma métrica (explicado na seção 3) retornada pelo teste do YCSB. Já o eixo X é referente às três rodadas de teste para cada *workload*. Conforme explicado na seção 3, cada *workload* seria testado três vezes, uma primeira com cem mil operações, na segunda seriam um milhão e uma última com cinco milhões. Sendo assim o eixo Y varia conforme o gráfico e os valores do eixo X são os mesmos para todos os gráficos. Abaixo serão apresentados os gráficos e suas explicações.

O Gráfico 1 apresenta o cenário no qual apenas operações de escrita são executadas, mostrando a relação entre a quantidade de operações por segundo com o número total de linhas a serem inseridas, conforme esse total de linhas a serem inseridas aumenta. Nesse caso, o teste foi feito no ambiente com uma largura de banda de 1Gb-*ethernet* e foram consideradas três quantidades totais de linhas a serem inseridas: 100.000, 1.000.000, e 5.000.000. É possível observar o evidente desempenho superior do Redis para as três quantidades de linhas inseridas, o que é um tema recorrente nos testes. Além disso, nota-se que o Cassandra teve um desempenho melhor do que o MongoDB nas três quantidades de linhas inseridas.

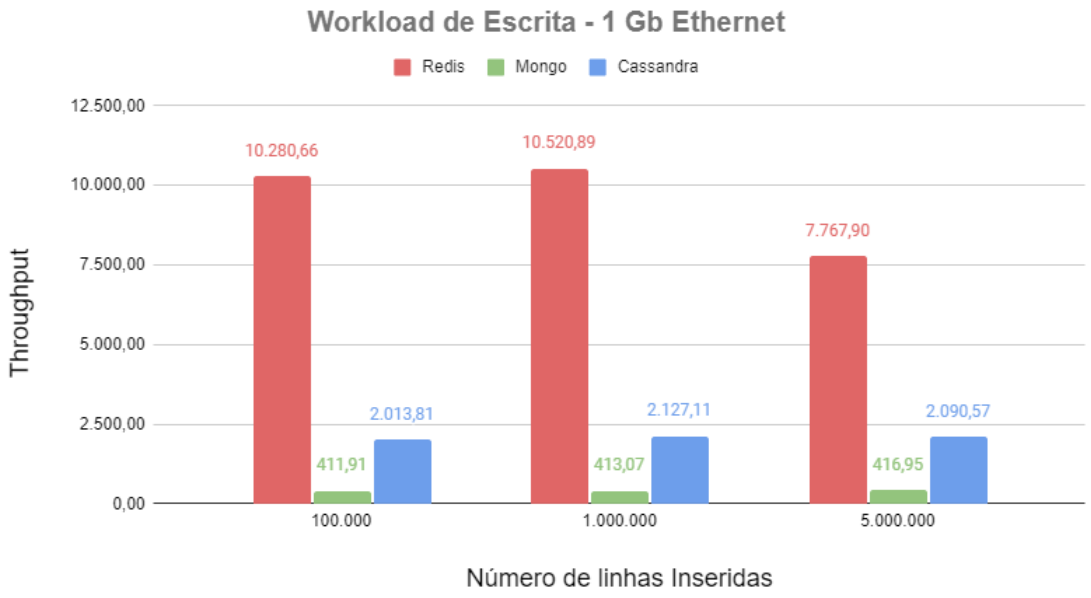

Gráfico 1. Comparação do *throughput* para o *workload* que contém apenas operações de escrita, com 1Gb-*ethernet* de largura de banda.

Como para cada linha inserida o Redis não precisa fazer uma operação de escrita em disco e sim apenas salvar em estruturas de dados em memória principal, isso pode justificar a grande diferença em relação aos bancos que salvam dados em disco. Além disso, apesar do Cassandra e do MongoDB ambos salvarem dados na memória em disco, o desempenho significativamente superior obtido pelo Cassandra em escrita pode estar relacionado a dois fatores: O primeiro é que, no Cassandra, uma operação de escrita envolve escrever os dados em arquivos sequenciais e *append-only* chamados *commit log* [Lakshman e Malik 2010]. Como a operação de escrita envolve apenas adicionar os dados no sistema de arquivo, ela não verifica se o dado já existe, não necessitando primeiramente realizar uma operação de leitura antes da escrita, o que reduz a complexidade e tempo de execução dessa operação. O segundo fator é que o sistema de arquivos do Cassandra é baseado na estrutura de dados árvore LSM (*Log-structured merge-tree)*, a qual tem uma complexidade de tempo de inserção mais baixa do que de busca [O'Neil, Cheng, e Gawlick 1996]. Outro fator que vale ressaltar também é a perceptível queda de desempenho do Redis com a quantidade de 5 milhões, isto pode estar relacionado com o aumento drástico de inserção de dados em memória principal, diminuindo seu desempenho no processamento.

O Gráfico 2 apresenta o mesmo cenário de execução de apenas operações de escrita, mas agora em um ambiente com uma banda larga de 40Gb-*infiniband*. Nota-se que a ordem de desempenho dos bancos continua a mesma, com o Cassandra e o MongoDB apresentando um ganho de desempenho em relação ao teste anterior.

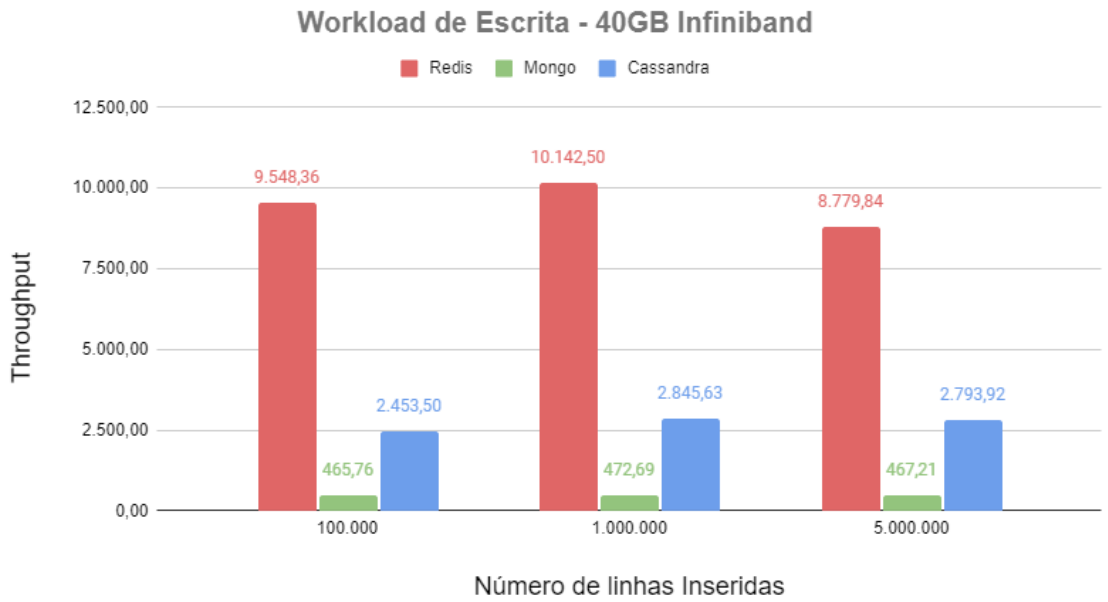

Gráfico 2. Comparação do *throughput* para o *workload* que contém apenas operações de escrita, com 40Gb-*infiniband* de largura de banda.

Já o Gráfico 3 mostra os resultados referentes aos *workloads* de apenas leitura para o cenário 1 (1Gb-*Ethernet*). Ele mostra uma relação de quantidade de operações por segundo (eixo Y) pelo número total de operações realizadas (eixo X). Novamente é possível observar o desempenho superior do Redis para as três quantidades de carga de dados, atingindo valores ainda maiores do que no *workload* de escrita.

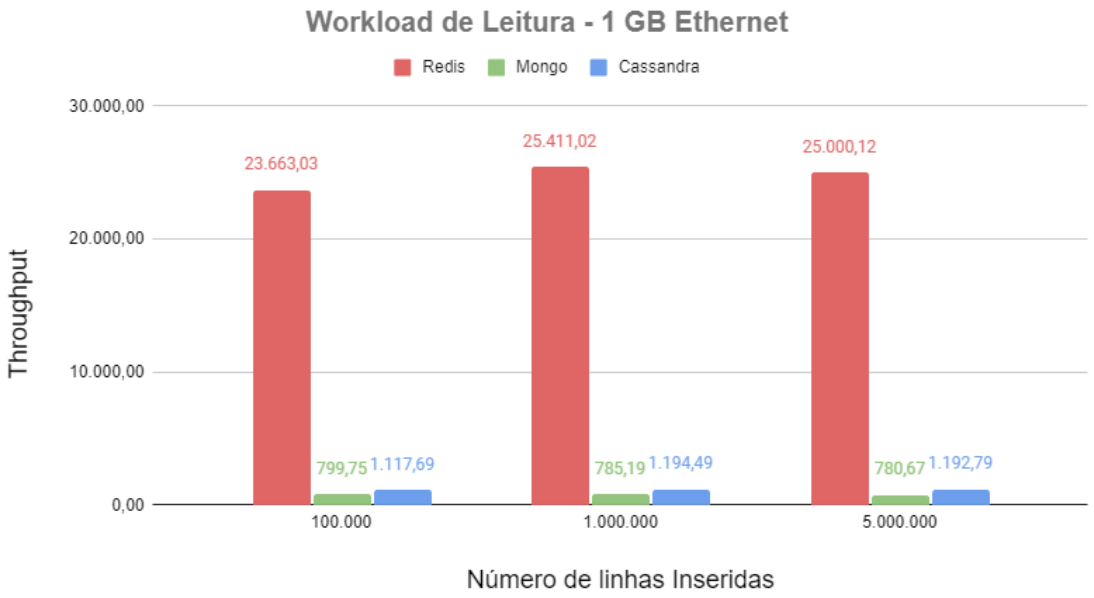

Gráfico 3. Comparação do *throughput* para o *workload* que contém apenas operações de leitura, com 1Gb-*ethernet* de largura de banda.

No cenário deste teste, representado pelo Gráfico 3, a diferença entre o desempenho do Cassandra em comparação ao MongoDB foi menor, uma vez que a quantidade de operações por segundo do Cassandra diminuiu e a do MongoDB aumentou se comparado aos gráficos anteriores. Uma possível explicação para essa diferença pode ser relacionada pelo modelo de dados de cada banco, uma vez que o modelo orientado a documento, utilizado pelo MongoDB, possui um desempenho melhor para operações de leitura em relação ao orientado à coluna, utilizado pelo Cassandra. Isso ocorre pois os dados no modelo de documentos estão mais próximos um do outro, já que foram salvos em formato JSON, sendo assim todos os dados estão dentro de um objeto maior e ao buscar por esse objeto, todos os dados contidos nele já são retornados [Newbedev 2021].

Já o Gráfico 4, assim como o Gráfico 3, mostra resultados de apenas leitura, porém para o cenário 2 (40Gb-*infiniband*). Nele vemos a mesma representação de valores que o Gráfico 3, ou seja, mostra uma relação de quantidade de operações por segundo (eixo Y) pelo número de total de operações realizadas (eixo X). Neste gráfico, é possível ver que os resultados do Gráfico 3 se repetem, no sentido comparativo de um banco para o outro, porém com os três bancos com os resultados superiores devido ao aumento da largura de banda.

O Gráfico 5 é referente ao *workload* que contém operações de atualização e leitura, sendo 50% da quantidade de operações executadas de atualização e os outros 50% sendo de operações de leitura. Nesse teste, o qual foi realizado no ambiente de 1Gb-*Ethernet*, também é possível observar o alto desempenho do Redis em relação aos outros bancos, semelhante ao resultado obtido no *workload* de apenas leitura. O Cassandra novamente obteve um desempenho superior ao do MongoDB, com uma diferença mais significativa do que o cenário de apenas operações de leitura.

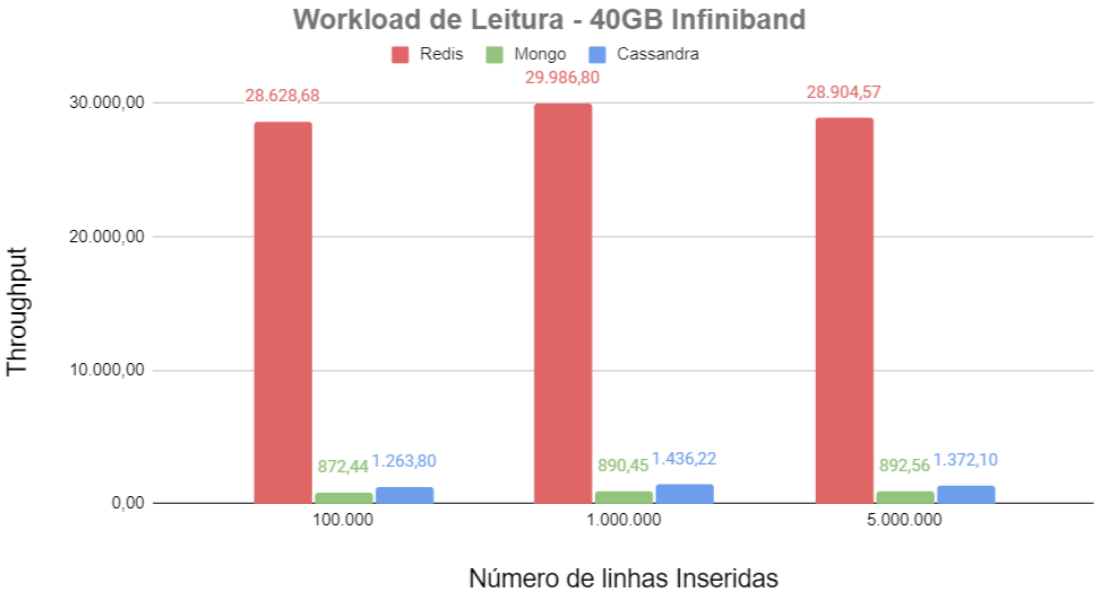

Gráfico 4. Comparação do *throughput* para o *workload* que contém apenas operações de leitura, com 40Gb-*infiniband* de largura de banda.

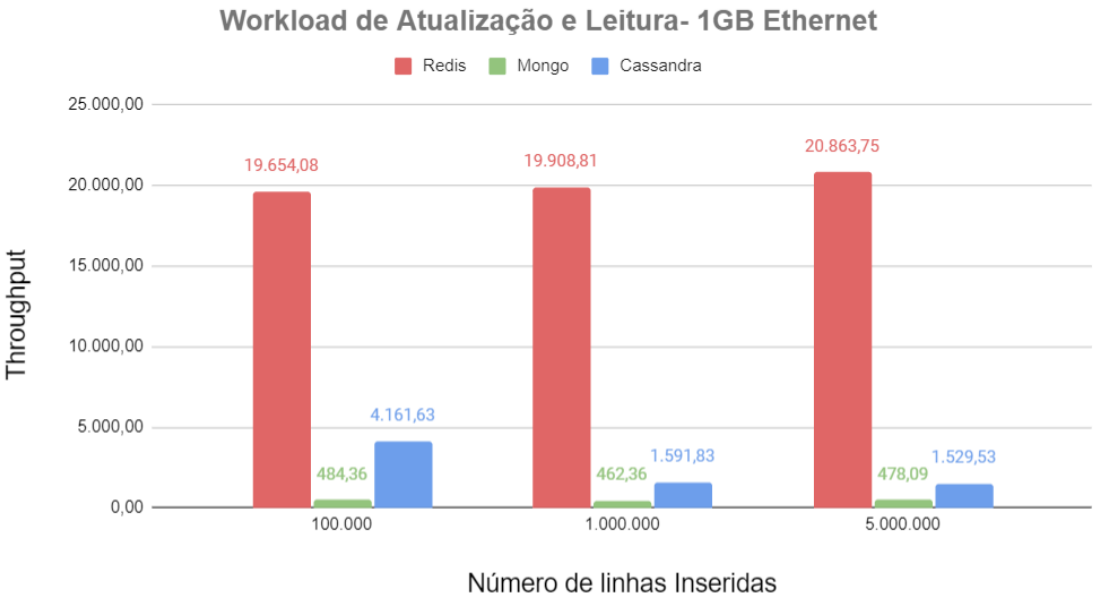

Gráfico 5. Comparação do *throughput* para o *workload* que contém operações de atualização e leitura, com 1Gb-*ethernet* de largura de banda.

Na arquitetura do Cassandra, uma operação de atualização não requer que ocorra uma validação se a chave sendo atualizada realmente existe, reduzindo a necessidade de haver uma operação de leitura nesse processo [Lakshman e Malik 2010]. Para que não existam problemas de concorrência, o MongoDB utiliza o processo de *locking* em

operações de atualização, neste estudo foi utilizada apenas uma *thread* para os testes, porém o processo de *locking* em uma *collection* pode afetar o desempenho em suas operações [Mongodb 2021]. Essas características podem justificar o desempenho superior do Cassandra em relação ao MongoDB.

Por fim, o Gráfico 6 apresenta os dados do mesmo *workload* que contém operações de atualização e leitura, como visto anteriormente, agora sendo executada no ambiente com a largura de banda de 40Gb-*infiniband*. A ordem de resultados se manteve igual novamente, com o Redis tendo um ganho significativo de desempenho, assim como no *workload* de leitura.

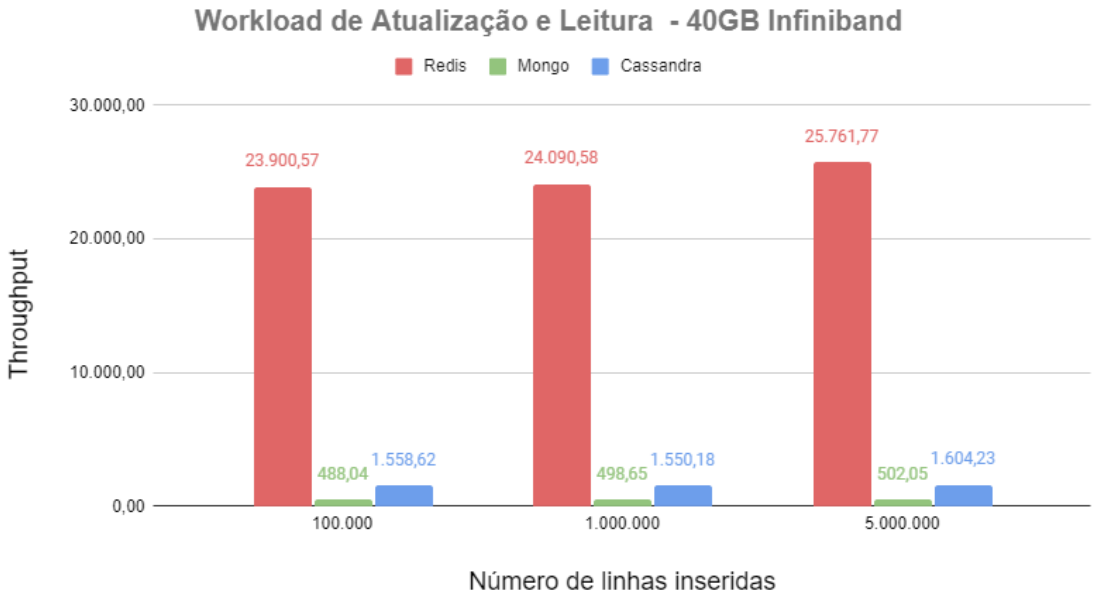

Gráfico 6. Comparação do *throughput* para o *workload* que contém operações de atualização e leitura, com 40Gb-*infiniband* de largura de banda.

4.2 Latência média

Os Gráficos abaixo mostram os valores de latência média obtidos por cada banco durante a execução dos workloads de escrita e leitura. O eixo Y representa o valor dessa latência média em microsegundos. Já o eixo X é referente às três rodadas de teste de cada *workload*, conforme explicado na seção 3, cada *workload* seria testado três vezes, uma primeira com cem mil operações, na segunda seriam um milhão e uma última com cinco milhões.

O Gráfico 7 mostra os resultados de latência média para o *workload* de escrita, no ambiente com largura de banda de 1Gb-*ethernet*. É evidente a alta latência do MongoDB em comparação aos outros bancos, sendo que, nesse mesmo *workload,* o MongoDB realizou o menor número de operações por segundo. Observa-se também a baixíssima latência do Redis em comparação aos outros bancos, que conforme explicado acima na seção 4.1, pode estar relacionado ao acesso à memória principal.

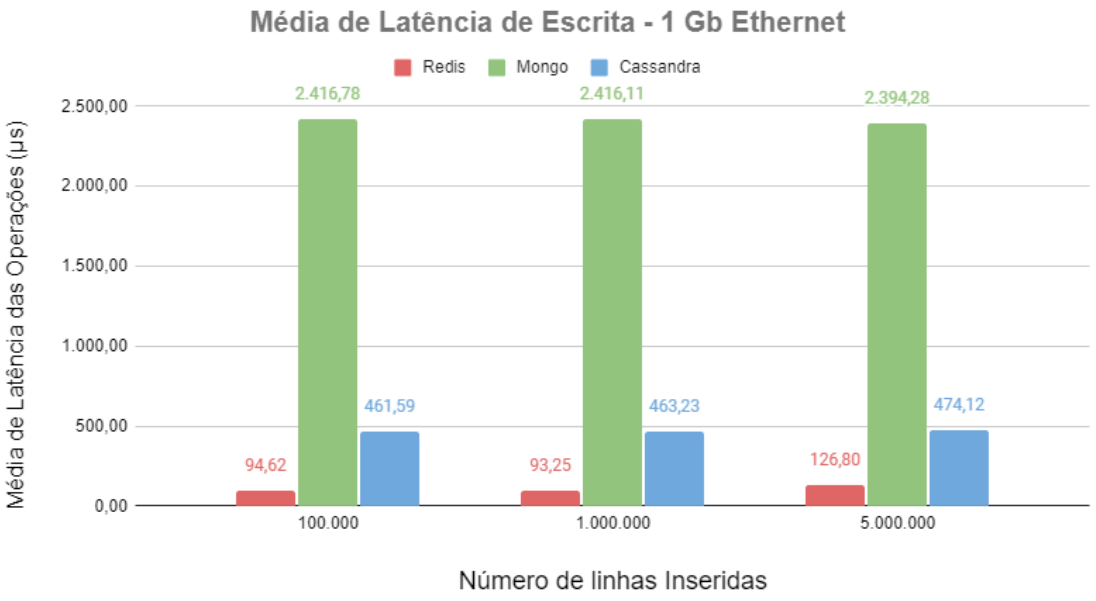

Gráfico 7. Comparação da latência média das operações de escrita, com 1Gb-*ethernet* de largura de banda.

Já o Gráfico 8 apresenta as latências médias para o mesmo *workload* de escrita, só que agora no ambiente com largura de banda de 40Gb-*infiniband*. Percebe-se um pequeno ganho de desempenho por parte do MongoDB e do Cassandra quando exposto a uma banda larga maior.

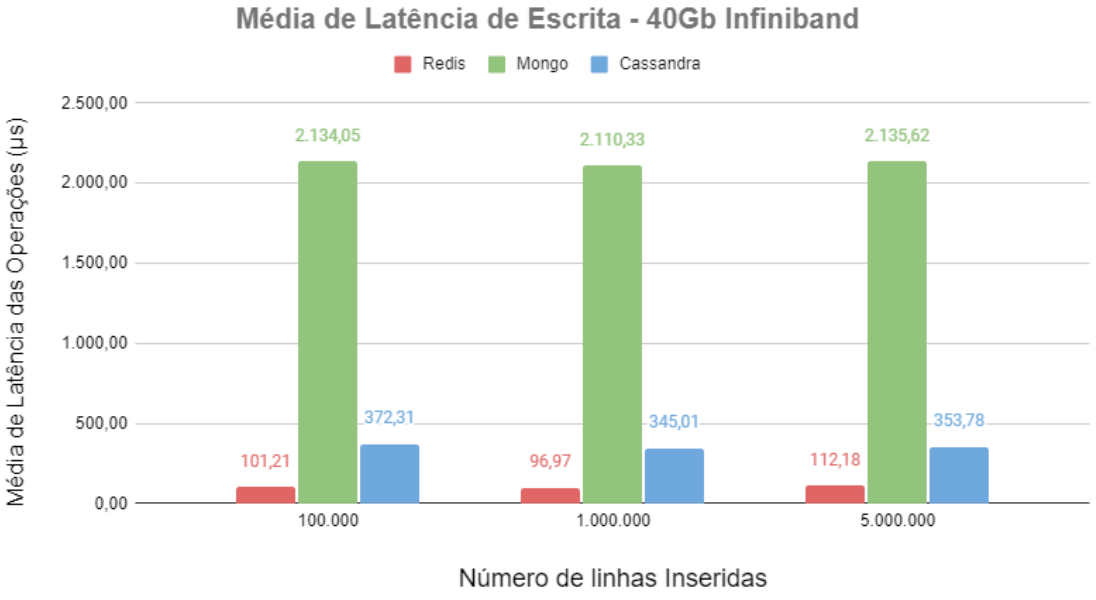

Gráfico 8. Comparação da latência média das operações de escrita, com 40Gb-*infiniband* de largura de banda.

No Gráfico 9 estão os valores de latência média para o *workload* de leitura para o ambiente com largura de banda de 1Gb-*ethernet* . Percebe-se que todos os bancos tiveram valores de latência menores do que no *workload* de escrita. Novamente o Redis

obteve o melhor resultado, com valores de latência de leitura aproximadamente 3 vezes menores do que os de escrita. Como foi explicado na seção 4.1, esse valor pode estar relacionado ao tempo de acesso à memória principal que é bem inferior ao tempo de acesso da memória em disco. Em comparação, o Cassandra teve uma latência média de aproximadamente o dobro em leitura comparada à escrita, o que corrobora o resultado anterior de que o Cassandra desempenha mais em operações de escrita.

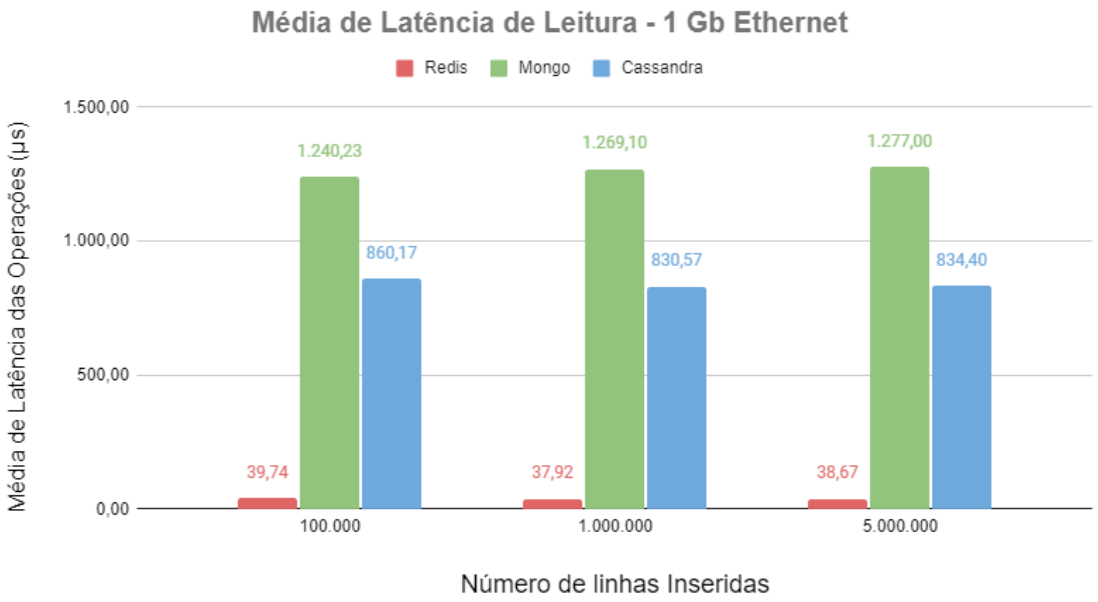

Gráfico 9. Comparação da latência média das operações de leitura, com 1Gb-*ethernet* de largura de banda.

Por fim, o Gráfico 10 apresenta as latências médias para o mesmo *workload* de leitura, agora considerando o ambiente com largura de banda de 40Gb-*infiniband.* Novamente a ordem de desempenho de cada banco se manteve igual ao do ambiente com menor largura de banda, com o Cassandra e o MongoDB apresentando uma redução de latência quando exposto a maior largura de banda.

5. Conclusão

Os objetivos propostos no início da pesquisa, de capturar *benchmarks* de cada banco de dados distribuído *NoSQL* e realizar uma comparação entre os desempenhos em cada cenário foram alcançados. Comparando a média da quantidade de operações por segundo por cada banco, nota-se que diferenças de desempenho significativas foram exemplificadas de acordo com qual tecnologia foi utilizada em certo tipo de *workload* que um sistema ou produto possa esperar receber.

Para uma persistência de dados de grandes *workloads* de escrita, o Cassandra pode ser considerado uma boa solução, maximizando a quantidade de operações desse tipo executadas por segundo. Enquanto para uma persistência de *workloads* de leitura, o MongoDB se mostrou uma boa escolha. Além disso, em cenários com o requisito de uma latência muito baixa com grandes volumes de leitura e atualização. O Redis é certamente uma tecnologia que oferece um alto desempenho, em troca da utilização de um tipo de memória mais cara, como a memória RAM ao invés de memória em disco.

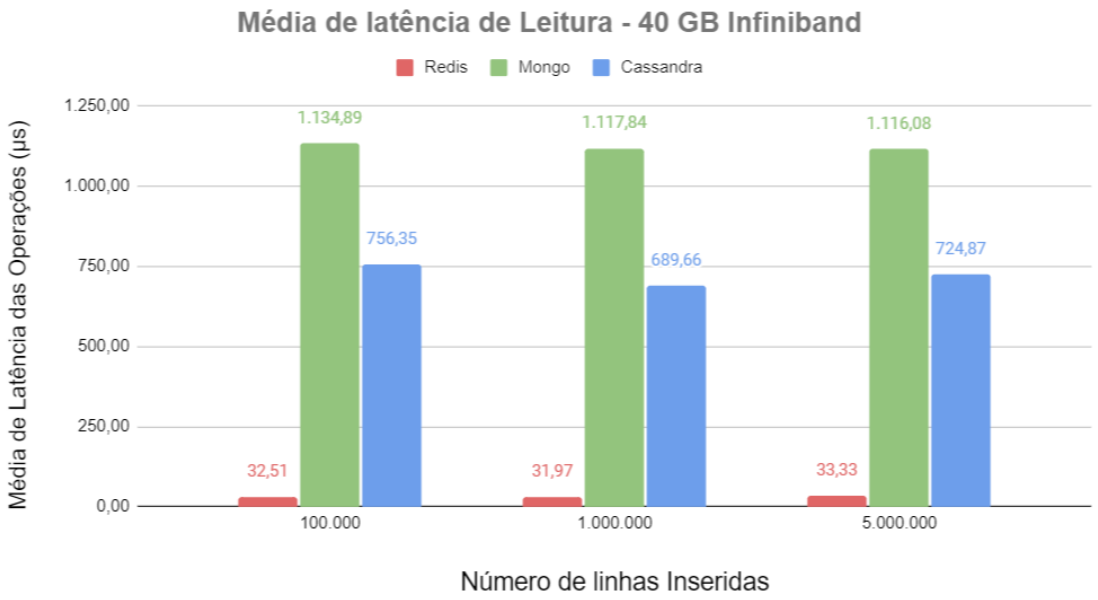

Gráfico 10. Comparação da latência média das operações de leitura, com 40Gb-*infiniband* de largura de banda.

Agradecimentos

Os autores agradecem ao MackCloud (https://mackcloud.mackenzie.br), Centro Multidisciplinar de Computação Científica e Nuvem da Universidade Presbiteriana Mackenzie, pelo apoio na realização desta pesquisa.

Referências

Abadi Daniel. (2012) "*Consistency Tradeoffs in Modern Distributed Database System Design*", Artigo Acadêmico - Yale University.

Berg Kristi, Seymour Tom, Goel Richa. (2013) "*History of Databases*", International Journal of Management & Information Systems.

Brewer, Eric. (2012) "*Cap Twelve Years Later: How The Rules Have Changed*", Artigo Acadêmico - Berkeley, University of California.

Brewer, Eric. (2000) "*Towards Robust Distributed Systems*".Berkeley,University of California.Disponível em: <https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>. Acesso em: 15 nov. 2021.

Carlson, Josiah. "*Redis in Action*". (2013)1. ed. Manning, p. 1-320.

Cooper Brian; Silberstein Adam; Tam Erwin; Ramakrishnan Raghu; Sears Russell. (2010) "*Benchmarking Cloud Serving Systems with YCSB, Yahoo! Research*", Santa Clara, CA, USA.

Chodorow, K. (2010) "*MongoDB: The Definitive Guide*", Segunda Edição, O'Reilly Media, Sebastopol, CA, USA.

Da Silva, M e Tavares, H. (2015) "*Redis Essentials: Harness the power of Redis to integrate and manage your projects efficiently"*, Birmingham, UK, Packt Publishing Ltd.

"*FAQ: Concurrency"*. MONGODB, 2021. Disponível em: <https://docs.mongodb.com/manual/faq/concurrency/>. Acesso em 09 dez 2021

"*How does column-oriented NoSQL differ from document-oriented?*". NEWBEDEV, 2021. Disponível em: <https://newbedev.com/how-does-column-oriented-nosql-differ-from-document-oriented>. Acesso em: 23 nov 2021.

Klein John; Gorton Ian; Ernst Neil; Donohoe Patrick. *Performance Evaluation of NoSQL Databases: A Case Study*, USA. Camegie Mellon University.Software Engineering Institute, 2015. p.7

Lakshman A. Malik P. (2009) "*Cassandra - A Decentralized Structured Storage System"*. Artigo acadêmico - Cornell Bower University

Lourenço, J.R., Cabral, B., Carreiro, P. (2015) "Choosing the right NoSQL database for the job: a quality attribute evaluation". Journal of Big Data.

O'Neil, P., Cheng, E., Gawlick, D. et al. (1996). "*The Log-Structured Merge-Tree (LSM-Tree)"*. Acta Informatica 33, 351–385.

Sharma V, Dave M. (2012) *"SQL and NoSQL Database"* International Journal of Advanced Research in Computer Science and Software Engineering.

Sitalakshmi Venkatraman, Kiran Fahd, Samuel Kaspi, Ramanathan Venkatraman. (2016) "*SQL Versus NoSQL Movement with Big Data Analytics"*, International Journal of Information Technology and Computer Science.

Tamer Ozsu. M; Valduriez, Patrick. (2011) "*Principles of Distributed Database Systems"*, Springer, Canada, p.497-502.