Uma Abordagem Experimental para Avaliar os Níveis de Consistência do Banco de Dados NoSQL Cassandra

Saulo Ferreira[1], **Ermeson Andrade**[1], **Júlio Mendonça**[2]

[1] Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
Recife, PE – Brasil

[2]Coordenação de Informática – Instituto Federal de Alagoas (IFAL)
Arapiraca, AL – Brasil

{saulo.gomesferreira,ermeson.andrade}@ufrpe.br, julio.neto@ifal.edu.br

Abstract. *Distributed computing allows the communication between multiple computers, making possible the data distribution between them, for example. In spite of that, this technology raises some architecture problems, like, for instance, the data consistency. The consistency of the data replicated among different servers aims to ensure that the same data is accessed on all running computers. However, ensuring consistency can affect the performance, since each level of consistency has its advantages and disadvantages. Therefore, this work aims at evaluating the impact of the consistency levels on the performance of the NoSQL (Not Only SQL) database Cassandra, where different scenarios and workloads are considered to study the trade-offs that emerge because of such levels. We embrace an experimental approach to evaluate and analyze the system response time when those different consistency levels and workloads are used. The results show that a high load of simultaneous users increases the disparity between the response times that each level presents, as well as the amount of data involved in the requests.*

Resumo. *A computação distribuída permite a comunicação entre vários computadores, possibilitando, por exemplo, a distribuição de dados entre eles. No entanto, essa tecnologia traz alguns problemas, como, por exemplo, a consistência dos dados. A consistência dos dados replicados entre os diferentes computadores visa garantir que o mesmo dado seja acessado em todas os computadores em execução. Entretanto, garantir a consistência pode afetar o desempenho, visto que tem suas vantagens e desvantagens. Assim, este trabalho avalia os impactos dos níveis de consistência no desempenho do banco de dados NoSQL (Não Somente SQL) Cassandra, onde diferentes cenários e cargas de trabalho são considerados para analisar os trade-offs que surgem a partir da utilização desses níveis. Nós adotamos uma abordagem experimental para avaliar e analisar o tempo de resposta do sistema quando esses diferentes níveis de consistência e carga de trabalho são utilizados. Os resultados obtidos mostram que a carga de usuários concorrentes acentua a disparidade entre os tempos de resposta que cada um dos níveis apresenta, bem como a quantidade de dados envolvidos nas requisições.*

1. Introdução

Um sistema distribuído é uma coleção de computadores que se comunicam através de uma rede de conexão [Le Lann 1977]. Tal arquitetura permite que diversos computadores atuem como uma só máquina, de modo a compartilhar tarefas e executar processos paralelos, bem como distribuir ou replicar dados entre si. Adicionalmente, com o avanço da conectividade, que permitiu uma comunicação mais rápida e com menos perdas, a computação distribuída alcançou uma escala global, permitindo que computadores ao redor do mundo pudessem trabalhar paralelamente no cumprimento de tarefas [Nadiminti et al. 2006]. Desta forma, além de aumentar a capacidade de processamento e armazenamento de dados, um sistema composto por máquinas (nós) espalhadas geograficamente pode reduzir os riscos de perda de dados e garantir uma maior disponibilidade do sistema [Bhagwan et al. 2003], uma vez que na ocorrência de falhas em um nó, os outros impediriam que o sistema se tornasse indisponível por completo.

O uso da estratégia de distribuição de nós em diferentes pontos geográficos, que é o principal ponto da computação distribuída, é muito utilizado na arquitetura de sistemas de bancos de dados, onde os dados são espalhados entre os nós e replicados entre eles. O objetivo de tal estratégia é tanto de reduzir a latência — uma vez que desta maneira, os dados podem ficar mais perto de mais usuários ao redor do mundo —, quanto de evitar perdas de dados decorrentes de uma falha em algum dos nós, que poderia ocasionar na perda das informações que estivessem armazenadas em um único servidor [Özsu and Valduriez 1996]. Entretanto, há desvantagens na arquitetura distribuída, uma vez que ao aumentar o número de servidores que compõem o sistema, a validação da equivalência de dados entre os nós pode ser custosa e demandar muito tempo da aplicação [Abadi 2012]. O problema da consistência dos dados distribuídos é encarado de diferentes maneiras, dependendo da necessidade da aplicação. Em alguns casos, o sistema exige que os dados utilizados sejam verificados em todos os nós antes de serem manipulados, como por exemplo, em uma aplicação financeira. Por outro lado, outros sistemas podem assumir que mesmo que o dado modificado não seja o mais atual, em algum momento será, o que é chamado de consistência eventual [Burckhardt 2014]. Nesses casos, os dados são persistidos nos nós de maneira assíncrona, sem que a aplicação necessite esperar a confirmação de sucesso em todos os servidores do sistema. Tal cenário pode ser encontrado em aplicações críticas de tempo real, como por exemplo, em um sistema de tráfego aéreo, cujo o foco primordial é que a aplicação funcione sem interrupções e sem demora na manipulação dos dados.

No estado-da-arte, há trabalhos que estudam como esta troca entre consistência e latência se dá em diferentes cenários. Em [Nejati Sharif Aldin et al. 2019], os autores para tentar abranger a necessidade dos sistemas distribuídos, eles criaram diferentes modelos de consistência, de modo que os mesmos são usados para avaliar o desempenho de diferentes sistemas distribuídos sem a necessidade de implementá-los. Em [Bermbach and Tai 2011], é questionado o quão cedo é a consistência eventual do sistema de armazenamento da Amazon, o S3, de modo que é avaliado quanto tempo leva para os nós serem atualizados após uma transação. Já em [Schultz et al. 2019], os autores focam em como os níveis de consistência de cada transação no Sistemas de Gerenciamento de Bancos de Dados (SGBD) NoSQL MongoDB podem ser ajustados, levando em conta os *trade-offs* com a desempenho.

Este trabalho avalia os *trade-offs* entre diferentes níveis de consistência do SGBD NoSQL Cassandra (ex.: *ONE* e *QUORUM*) [Hewitt 2010] e o tempo de resposta decorrente de cada um desses níveis. Consequentemente, analisamos os cenários que apresentam situações mais relevantes, seja por não haver divergências significativas de desempenho ao considerar níveis de consistência distintos, ou por uma configuração apresentar um aumento significativo no tempo de resposta que, dependendo do tipo de aplicação relacionada, pode não valer a pena o *trade-off*. Os cenários considerados neste trabalho exploram situações onde grupos de usuários (de diferentes tamanhos) fazem requisições simultâneas, situações que exploram a quantidade de dados trocada entre cliente e servidor e situações que consideram o número de replicações configuradas no SGBD para o espaço de trabalho. Os resultados mostram que a diferença do tempo de resposta entre os níveis de consistência varia de acordo com a carga de requisições que o sistema recebe e que, em determinados casos, há pouca variação nos tempos de resposta para os níveis de consistência avaliados. Eles também revelam que outros parâmetros como o intervalo médio entre as requisições e o tamanho dos dados podem ter um impacto relevante nos tempos de resposta.

O artigo está dividido em sete seções principais. A Seção 2 apresenta a fundamentação do tema abordado, para explicar as tecnologias que são utilizadas e como elas se encaixam no problema abordado. A Seção 3 apresenta os trabalhos relacionados, mostrando como a literatura inspirou este trabalho e como ele se difere dos demais. Por conseguinte, a Seção 4 apresenta a arquitetura experimental, onde é explicado e exemplificado o modo como foi criado e configurado o ambiente de experimentos. A Seção 5 discute os resultados obtidos, apresentando a relevância dos mesmos. Por fim, a Seção 6 apresenta as limitações do trabalho e a 7, conclusões e como elas podem levar aos trabalhos futuros.

2. Fundamentação

Neste seção são apresentados os principais conceitos para um melhor entendimento deste trabalho.

2.1. Teorema CAP

As escolhas que devem ser tomadas, para focar na característica mais importante de um sistema, podem ser descritas de maneira sucinta pelo Teorema CAP [Simon 2000] (também conhecido como Teorema de Brewer). Este teorema explica que um sistema de armazenamento de dados distribuídos possui três principais garantias: consistência, disponibilidade e tolerância a falhas. Todavia, em seu teorema, Brewer afirma que só é possível priorizar duas das três características ao configurar a arquitetura do sistema.

Entretanto, em um sistema distribuído, as opções se reduzem à consistência ou disponibilidade, uma vez que construí-lo sem tolerância a falha de partição acaba sendo impraticável [Brewer 2012]. Desta maneira, ao optar pela consistência, e tentar manter todos os dados iguais em quaisquer dos nós que eles estejam distribuídos, o sistema deve perder em disponibilidade, uma vez que demandará mais tempo para que os nós sejam verificados e atualizados. Semelhantemente, ao priorizar a disponibilidade, o sistema deve abrir mão da consistência forte e ter ciência de que nem sempre os dados de um determinado nó será o mais atual. Neste trabalho, mais especificamente, o foco é avaliar

os impactos dos níveis de consistência no desempenho dos sistemas, visto que a definição da consistência é fundamental para definir como o SGBD lida com transações entre as réplicas dos dados [Wang et al. 2014].

2.2. Níveis de Consistência do SGBD Cassandra

Muito embora exista uma gama enorme de SGBDs, neste trabalho, iremos adotar o Apache Cassandra, que é um SGBD não-relacional focado em desempenho e que pode ser executado em uma estrutura de centenas de nós [Lakshman and Malik 2010]. A adoção se dá por conta da grande relevância do Cassandra, que, atualmente, é o SGBD de "grandes-colunas" mais utilizado no mundo, segundo o Ranking DBEngine [DB-Engines 2021]. Tal SGBD foi desenvolvido pelo Facebook, o qual abriu seu código-fonte em 2008 e hoje é mantido pela fundação Apache. O Cassandra permite a fácil implantação de nós em diferentes *datacenters* que formam um *cluster*, de modo a oferecer estratégias de replicação de dados entre os nós. Como, por exemplo, a *SimpleStrategy* para um *datacenter* e a *NetworkTopologyStrategy* para múltiplos *datacenters*. Apesar do Cassandra ser focado em disponibilidade [Abramova and Bernardino 2013], ele permite a definição do nível de consistência em cenários onde seja realmente necessário abrir mão da disponibilidade por tal aspecto. Entre os níveis de consistência oferecidos pelo Cassandra, os principais são os seguintes:

- *ONE*: Em escrita, o sistema espera que somente um nó do *cluster* finalize a operação para validar a transação com sucesso. Em leitura, ele obtém os dados do primeiro nó que se conectar, ainda que neste não estejam os dados mais atualizados. Vale ressaltar que há uma chance considerável de não se ter o dado mais atualizado na leitura, se esta for feita simultaneamente a uma escrita em outro nó. Este nível de consistência é considerado o mais fraco.
- *QUORUM*: Em escrita, o sistema espera que a maioria dos nós finalize a operação para validar o seu sucesso. Em leitura, ele verifica os dados da maioria dos nós para retornar o mais atualizado entre os avaliados. Vale ressaltar que a definição de "maioria" é relacionado ao primeiro inteiro maior que a metade do fator de replicação configurado no sistema.
- *ALL*: Em escrita, o sistema espera que todos os nós do *cluster* finalize a operação para validar a transação com sucesso. Em leitura, ele obtém os dados de todos os nós e retorna ao cliente os dados daquele nó onde o dado está mais atualizado. Desta forma, se leitura e escrita estiverem configurados para tal consistência, se tem a certeza de que o dado obtido é o mais atualizado do sistema. Este nível de consistência é considerado o mais forte.

3. Trabalhos Relacionados

Entre os trabalhos que exploram mais a fundo a consistência em armazenamento distribuído de dados, [Gomes et al. 2019] propuseram uma abordagem focada em redes de Petri para avaliar tanto a consistência, quanto a disponibilidade de uma instância do SGBD Cassandra configurado com três nós. O trabalho também fez uso de técnicas de *Design of Experiments* (DoE) para planejar os experimentos e determinar os fatores que mais influenciam as métricas coletadas. Segundo tal trabalho, além dos níveis de consistência, o fator de replicação e o tempo de comunicação do nó coordenador são os fatores que mais influenciam no tempo de resposta do sistema. Já em [Dede et al. 2013],

foi analisado o comportamento, em termos de desempenho, do Cassandra quando usado em conjunto com o Hadoop [Borthakur 2007] e o *MapReducing*. Os experimentos realizados no sistema consideraram, entre outros parâmetros, o tamanho dos dados envolvidos na operação e o fator de replicação. O trabalho também considerou o fator de replicação, ao variar até 8 réplicas, chegando a conclusão que tal parâmetro não tem muita influência no desempenho da aplicação do *MapReduce*.

[Liu et al. 2015] propuseram uma análise quantitativa acerca da consistência do Cassandra, utilizando um modelo probabilístico formal que objetiva avaliar quantas vezes o SGBD consegue garantir a consistência dos seus dados sob uma série de condições propostas. O trabalho não focou em uma análise de desempenho, mas sim em estudar as garantias de consistência que o Cassandra possui. Similarmente, adotando o Cassandra e outros SGBDs NoSQL (como o MongoDB [Membrey et al. 2010]), [Diogo et al. 2019] apresentaram uma discussão sobre os níveis de consistência que tais sistemas se propõem a aplicar. Em suas explicações, os autores apresentam alguns resultados de desempenho ao se considerar o número de nós e o fator de replicação como principais parâmetros. Por fim, situaram cada um dos sistemas no seu quadrante do teorema CAP.

Também existe um conjunto de outros trabalhos que focam na avaliação de SGBDs que não necessariamente consideram a consistência ou adotam o Cassanda. Em [Wang et al. 2014], os autores utilizaram o Apache HBase em uma análise de desempenho considerando replicação e consistência. Em [Gorbenko et al. 2019], os autores usaram uma série de sistemas de armazenamento distribuídos (e.x.: DynamoDB [Chodorow 2013] e Hadoop) para avaliar os *trade-offs* entre disponibilidade, consistência e latência. Já em [Bermbach and Tai 2014], os autores buscam fazer um teste de *benchmarking* para monitorar o desempenho da consistência eventual do Amazon S3, com o objetivo de prover uma aplicação de avaliação contínua de *Quality of Service* (QoS).

Ainda que alguns dos trabalhos descritos anteriormente apresentem uma avaliação de desempenho considerando os níveis de consistência e fator de replicação, nenhum deles se aprofunda no impacto da consistência em um número maior de nós, tampouco considera a carga de usuários simultâneos em um cenário de aplicação real. Este trabalho, mais especificamente, apresenta como a latência se comporta de acordo o número de requisições simultâneas aplicadas ao sistema, bem como qual a interferência do tamanho do dado em um cenário onde se faz necessário validação de uma variedade de nós. Desta maneira, este trabalho introduz uma nova perspectiva de avaliação, considerando outros parâmetros relevantes em cenários reais.

4. Arquitetura Experimental

Esta seção detalha a arquitetura experimental usada para a realização dos experimentos.

4.1. Ambiente de Testes

Por lidar com um ambiente distribuído, foram configuradas múltiplas máquinas virtuais executando uma instância do Cassandra em suas dependências. Os endereços de IP de cada uma das máquinas virtuais em execução foram utilizados para configurar o Cassandra, definindo cada um dos nós como *seed*. Tal ajuste possibilita que as replicações e a distribuição dos dados possam ocorrer através de qualquer um dos nós envolvidos, bem como cada servidor possa atuar como coordenador das operações.

Para o ambiente de testes, foi escolhido o Google Cloud Platform (GCP), que é uma suíte de serviços de computação em nuvem fornecida pela Google. Na plataforma, são instanciadas 6 Máquinas Virtuais (VMs) rodando o Ubuntu Minimal, versão 18.04, Bionic. As VMs usadas são do tipo *ec2-small*, modelo que possui configuração com 2 vCPUs e 2 GB de memória RAM. Tais VMs são utilizadas para configurar um *cluster* rodando o Cassandra com dados e réplicas distribuídas através dos 6 nós. O banco de dados, por sua vez, está estruturado por três colunas do tipo *string* e um identificador *integer*. As máquinas são configuradas na zona central dos Estados Unidos, definida no GCP por *us-central1-a*. Por conta da configuração baixa das VMs, o tamanho dos dados utilizados nos testes é limitado proporcionalmente à capacidade de processamento e armazenamento dos nós do sistema.

4.2. Execução dos Experimentos e Medições

Para execução dos testes, foi utilizado o Apache JMeter [Halili 2008], que é uma ferramenta de automação de testes capaz de simular carga e estresse em recursos computacionais. Mais especificamente, o JMeter é utilizado para coletar os tempos de respostas considerando diferentes configurações, como os níveis de consistência e a quantidade de usuários.

Os experimentos foram executados de um mesmo computador, seguindo uma sequência de parâmetros a serem variados de acordo com o cenário analisado. Para simular a carga suportada em cada cenário, utilizamos as *threads* do JMeter, onde foi variada a quantidade de requisições simultâneas. Além disso, os testes foram realizados considerando diferentes níveis de consistência fornecidos pelo Cassandra, e, por se tratar de um sistema feito para o gerenciamento de grandes quantidades de dados, também foi considerado a variação do tamanho dos dados enviados e recebidos pelo SGBD.

4.3. Cenários analisados

Operações	Intervalo entre requisições (ms)	Número de usuários	Fator de replicação	Nível de consistência	Tamanho da requisição
Leitura	50	10	2	*ONE*	800B
Escrita	100	20	3	*QUORUM*	1600B
	500	30	4	*ALL*	2400B
	1000	40	5		3200B
	2000	50	6		4000B
		60			
		70			
		80			
		90			
		100			

Tabela 1. Parâmetros considerados nos experimentos

A Tabela 1 apresenta os parâmetros considerados para definição dos cenários analisados. Os cenários analisados foram executados variando o número de usuários simultâneos que buscavam ou inseriam dados no BD. O objetivo foi de simular uma carga de trabalho real realizada no sistema ao considerar uma quantidade de usuários concorrentes. Para os testes realizados, o número de usuários simultâneos variou de 10 a 100

usuários fazendo, ao todo, 1000 requisições em cada experimento para, por fim, extrair-se a média do tempo de execução de todas as requisições. Já a carga dos usuários é detalhada na Tabela 2, a qual apresenta, na horizontal, a variação do intervalo entre as requisições, e, na vertical, a variação do número de usuários simultâneos. Ao considerar, por exemplo, 10 usuários fazendo requisições ao mesmo tempo com o intervalo de 50 ms entre as requisições, se obtém uma carga de 200 requisições por segundo, como mostra a primeira linha da Tabela 2.

Qnt. de usuários	50 ms	100 ms	500 ms	1000 ms	2000 ms
10	200 req/s	100 req/s	20 req/s	10 req/s	5 req/s
30	600 req/s	300 req/s	60 req/s	30 req/s	15 req/s
50	1000 req/s	500 req/s	100 req/s	50 req/s	25 req/s
...	...	...	...	...	...
80	1600 req/s	800 req/s	160 req/s	80 req/s	40 req/s
100	2000 req/s	1000 req/s	200 req/s	100 req/s	50 req/s

Tabela 2. Carga dos usuários

O tamanho dos dados enviados ou recebidos pelo sistema variou de 800 a 4000 bytes. Os três níveis de consistência do Cassandra considerados nos experimentos foram: *ONE*, *QUORUM* e *ALL*. Como as VMs do ambiente criado simulam a execução de nós em um mesmo *datacenter*, só utilizamos níveis de consistência aplicáveis para tal cenário. Isto é, descartamos aqueles utilizados em cenários com múltiplos *datacenters*. Além disso, também consideramos como o sistema se comporta de acordo com a variação do número de replicas configurada (de 2 à 6 replicas), a fim de estimar o impacto dessas configurações no desempenho final dos sistema.

5. Resultados Experimentais e Discussão

Nesta seção, iremos apresentar os resultados obtidos dos experimentos realizados. Vale ser destacado que adotamos o experimento fatorial completo [Cheng 2016] que inclui todas as possíveis combinações entre os níveis dos parâmetros da Tabela 1. Porém, por questões de limitação de espaço, iremos apresentar os resultados mais relevantes. O gráfico presente na Figura 1 apresenta a variação do tempo de resposta obtido em operações de leitura de diferentes tamanhos de dados, os quais variam de 800 a 4000 bytes em relação ao número de usuários simultâneos. O eixo X apresenta a variação de usuários simultâneos, de 10 a 100, enquanto o Y varia de acordo com o tempo de resposta que um determinado tamanho de dado leva para ser buscado, validado e retornado pelo Cassandra. Nesta primeira análise, foi utilizado o nível de consistência *QUORUM*, considerando 6 réplicas dos dados distribuídos entre os nós.

Através dos resultados, é possível observar que quando o cliente buscar poucos dados no banco (800 ou 1000 bytes, como indicado no gráfico), o tempo de resposta não varia, independente de quantos usuários simultâneos estejam realizando esta operação. Entretanto, o propósito do Cassandra é armazenar e prover grandes quantidades de dados, e neste quesito há uma escalada considerável no desempenho do sistema, ao comparar os tempos de leitura de 800 e 4000 bytes no cenário onde há 100 usuários simultâneos. Já se considerarmos poucas requisições simultâneas (menor que 50 usuários concorrentes), o sistema não varia muito no tempo de resposta de acordo com os tamanhos estudados.

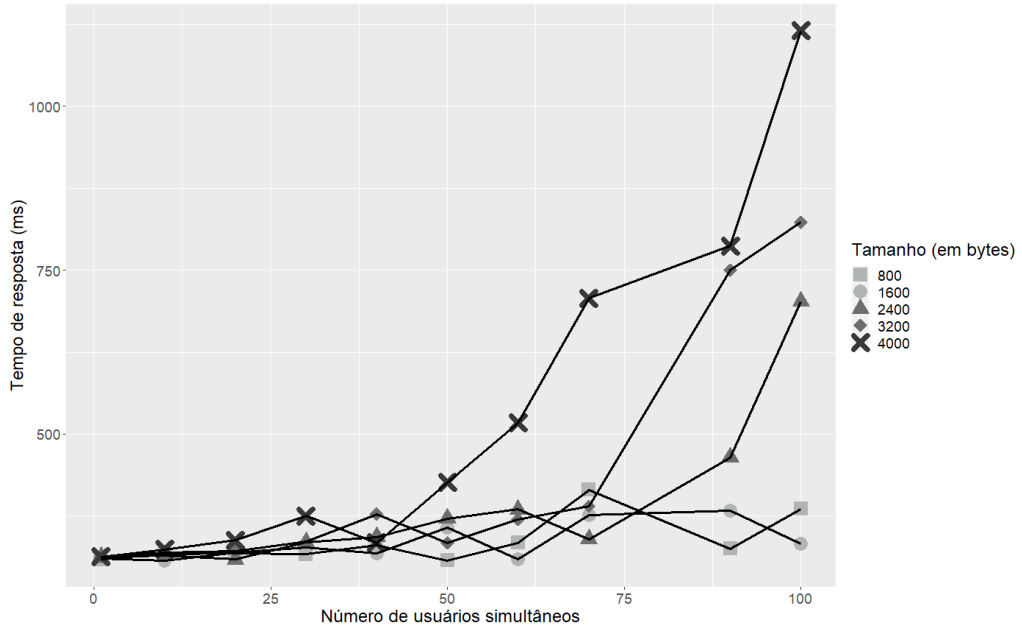

Figura 1. Tempo de resposta de acordo com o tamanho da requisição e do número de usuários simultâneos enviando tais requisições.

Vale ressaltar que o intervalo entre as requisições utilizado neste experimento (500 ms) tem um impacto significativo no desempenho do sistema.

Em uma segunda análise, considerando a consistência mais forte do sistema (*ALL*), 50 usuários concorrentes, considerando as mesmas 6 réplicas de dados e uma operação de *select* no SGBD, obtivemos os resultados para o tempo de resposta variando os intervalos entre as requisições. A Tabela 3 apresenta os valores obtidos neste segundo experimento. Tais resultados mostram que um intervalo menor entre as requisições tem como efeito um tempo de resposta maior, pois, como foi apresentado na Tabela 2, a carga de trabalho aumenta de acordo com a diminuição do intervalo entre as requisições. Portanto, ao aumentar tal intervalo para 2000 ms, por exemplo, o sistema apresenta uma considerável redução no tempo de resposta, visto que diminui a carga de requisições para o SGBD.

Intervalo entre as requisições (ms)	Tempo de resposta (ms)
50	1863
100	1621
500	717
1000	519
2000	432

Tabela 3. Tempo de resposta obtido variando o intervalo entre as requisições

A Figura 2 apresenta a diferença do tempo de resposta ao variar o fator de replicação, utilizando o nível de consistência *ALL*. Enquanto o eixo X apresenta a variação do fator de replicação (de 2 até 6), o eixo Y indica o tempo de resposta para as operações de leitura (*select*) e escrita (*insert*). Os resultados mostram que as operações de escrita não são muito afetadas pelo número de replicações, onde se realizou 1000 requisições

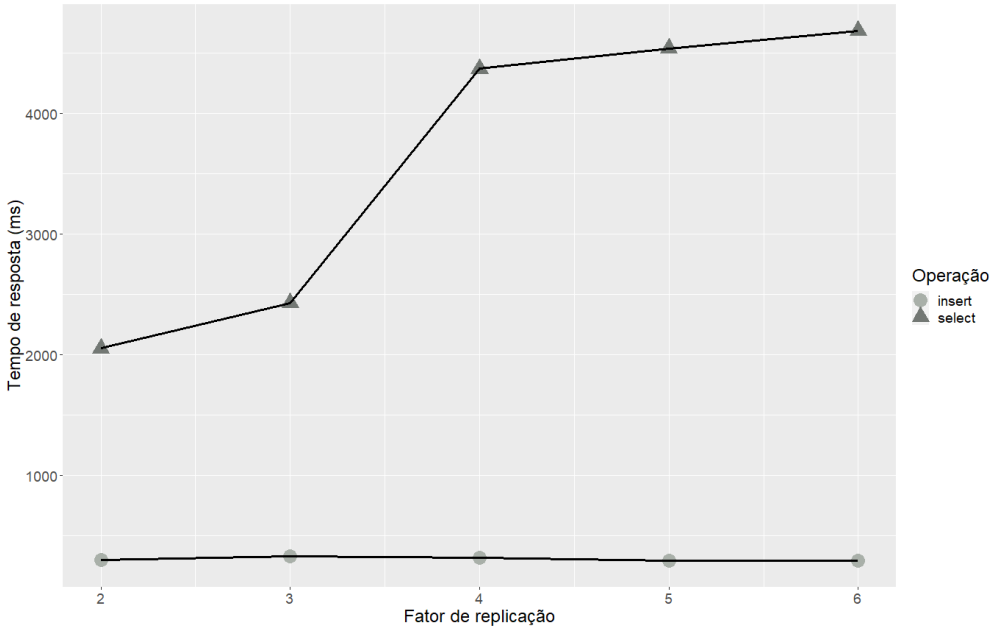

Figura 2. Variação do tempo de resposta considerando o fator de replicação

totais por 100 usuários simultâneos em um intervalo de 500 ms. Esse comportamento não se refletiu nas operações de leitura, onde os mesmos 100 usuários realizaram o mesmo número de requisições no mesmo intervalo anterior. Em tal operação, o tempo de resposta escalou consideravelmente ao aumentar o número de replicas. Ao considerar as 6 réplicas, há um aumento de 192% no tempo de resposta em relação ao cenário com a apenas 2 réplicas. Tais resultados podem ser explicados devido ao fato da operação de escrita (*insert*) no BD considerar apenas um registro de 40 bytes, enquanto a operação de leitura (*select*) considerou um tamanho total de 4000 bytes, por buscar múltiplas linhas no BD. Portanto, o processo de validação das replicações utilizados para os dois cenários é diferente, visto que o processo de leitura dos dados tem que validar uma quantidade maior de dados.

Ao variar o nível de consistência, é possível ver a disparidade do desempenho do sistema em decorrência da escolha de uma consistência mais fraca ou mais forte. Nos experimentos realizados com os três principais níveis de consistência que o Cassandra disponibiliza, foram obtidos os resultados contidos na Figura 3. Tais experimento consideraram diferentes usuários executando operações de leitura de 4000 bytes de dados a cada 500 ms no BD, em um cenário com fator de replicação 6. Como é mostrado no gráfico, os valores não se alteram muito quando há um número reduzido de usuários simultâneos no sistema (ex.: até 30). A variação nesses casos acaba sendo mais relacionada à alguma instabilidade de rede do que propriamente do servidor que hospeda o SGBD. Entretanto, após 30 usuários, o SGBD já demonstra uma queda relevante no desempenho, devido ao fato de ter que realizar checagens constante em todos os nós e réplicas configurados. Os resultados mostram que uma carga de 100 usuários simultâneos é suficiente para apresentar um tempo de resposta médio de 10000 ms, se utilizado o nível de consistência *ALL*. Esse comportamento se apresenta através do estresse aplicado tanto da carga, quanto do intervalo ajustado entre as requisições, além de considerarmos o maior tamanho de dado do nosso ambiente experimental.

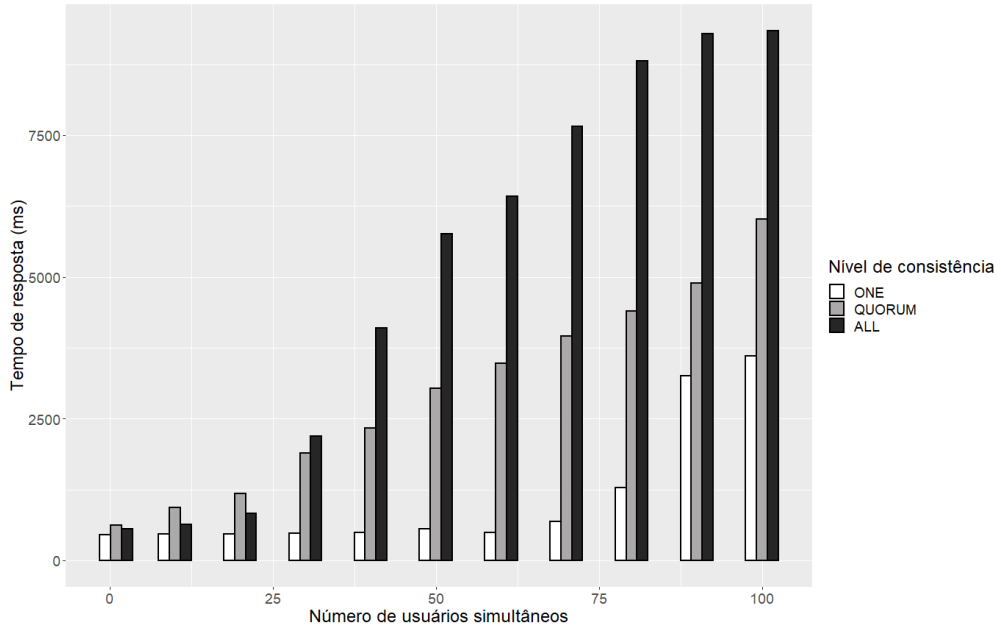

Figura 3. Variação do tempo de resposta considerando o nível de consistência

Os resultados mostram o quanto a consistência forte (*ALL*) é custosa, uma vez que exige que o SGBD verifique todas as replicações dos dados distribuídas entre os nós no *cluster* para finalizar a transação. O fluxo de validação deste nível de consistência aumenta consideravelmente o tempo de resposta em relação aos níveis *QUORUM* e *ONE*, que exigem a validação em uma quantidade menor de réplicas. A diferença do tempo de resposta chega a ser quase 60% menor do nível de consistência máximo (*ALL*) e o mínimo (*ONE*), ao apresentar uma queda de 9654 ms para 3966 ms no cenário onde há 100 usuários fazendo requisições simultâneas.

6. Limitações do Trabalho

Este trabalho apresenta limitações no que concerne arquitetura a experimental adotada, visto que a mesma considera VMs de baixa configuração, com número limitado de vCPUs e memória RAM. Por conta disso, há uma limitação no tamanho dos dados considerados nos testes, sendo reduzido proporcionalmente à capacidade de processamento e armazenamento das máquinas. É importante pontuar também a questão da plataforma de estruturação da arquitetura experimental, sendo todos os nós do sistema implantados sobre a Google Cloud Platform, podendo não ser completamente generalizado para outros serviços de computação em nuvem.

O Cassandra oferece uma arquitetura descentralizada, onde cada nó pode atuar como coordenador da operação corrente. Entretanto, nos experimentos realizados, consideramos apenas requisições em um mesmo nó, que, por sua vez, atuava como coordenador e validador da distribuição, replicação e consistência dos dados. Desta maneira, não podemos afirmar que os resultados seguem o mesmo padrão em uma execução simultânea em diferentes nós do mesmo *cluster*.

7. Conclusões e Trabalhos Futuros

O impacto dos níveis de consistência no desempenho varia muito de acordo com o número de usuários simultâneos. Tal diferença, por sua vez, se torna ainda mais acentuada de acordo com o aumento da carga de requisições simultâneas. Também foi mostrado que o aumento da carga através da redução do intervalo entre as requisições, faz com que o tempo de resposta seja aumentado consideravelmente. Tais fatores, portanto, devem ser considerados na hora da implantação de um sistema de armazenamento de dados com o SGBD Cassandra.

Os resultados obtidos neste trabalho podem ajudar arquitetos de software ou administradores de sistemas a planejar a implantação dos sistemas deles, considerando diversos aspectos aqui abordados. Através dos resultados obtidos aqui, um(a) arquiteto(a) ou administrador(a) de BD, pode, por exemplo, optar por uma consistência mais forte caso seu sistema venha a receber poucos usuários simultâneos, pois a variação é mínima. Como trabalho futuro, almejamos comparar o Cassandra com outros SGBDs NoSQL mais utilizados no mercado, a fim de estimar a variação de desempenho em tais sistemas. Também almejamos considerar outros métodos de replicação do Cassandra como o *NetworkTopologyStrategy*.

Referências

Abadi, D. (2012). Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, 45(2):37–42.

Abramova, V. and Bernardino, J. (2013). Nosql databases: Mongodb vs cassandra. In *Proceedings of the international C* conference on computer science and software engineering*, pages 14–22.

Bermbach, D. and Tai, S. (2011). Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, pages 1–6.

Bermbach, D. and Tai, S. (2014). Benchmarking eventual consistency: Lessons learned from long-term experimental studies. In *2014 IEEE International Conference on Cloud Engineering*, pages 47–56. IEEE.

Bhagwan, R., Savage, S., and Voelker, G. M. (2003). Understanding availability. In *International Workshop on Peer-to-Peer Systems*, pages 256–267. Springer.

Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21.

Brewer, E. (2012). Cap twelve years later: How the"rules"have changed. *Computer*, 45(2):23–29.

Burckhardt, S. (2014). Principles of eventual consistency.

Cheng, C.-S. (2016). *Theory of Factorial Design*. Chapman and Hall/CRC.

Chodorow, K. (2013). *MongoDB: the definitive guide: powerful and scalable data storage*. "O'Reilly Media, Inc.".

DB-Engines (2021). DB-Engines Ranking. `https://db-engines.com/en/ranking`. [Online; accessed 17-jan-2021].

Dede, E., Sendir, B., Kuzlu, P., Hartog, J., and Govindaraju, M. (2013). An evaluation of cassandra for hadoop. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 494–501. IEEE.

Diogo, M., Cabral, B., and Bernardino, J. (2019). Consistency models of nosql databases. *Future Internet*, 11(2):43.

Gomes, C., Borba, E., Tavares, E., and Junior, M. N. d. O. (2019). Performability model for assessing nosql dbms consistency. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–6. IEEE.

Gorbenko, A., Romanovsky, A., and Tarasyuk, O. (2019). Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency. *Journal of Network and Computer Applications*, 146:102412.

Halili, E. H. (2008). *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing Ltd.

Hewitt, E. (2010). *Cassandra: the definitive guide*. "O'Reilly Media, Inc.".

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

Le Lann, G. (1977). Distributed systems-towards a formal approach. In *IFIP congress*, volume 7, pages 155–160. Toronto.

Liu, S., Nguyen, S., Ganhotra, J., Rahman, M. R., Gupta, I., and Meseguer, J. (2015). Quantitative analysis of consistency in nosql key-value stores. In *International Conference on Quantitative Evaluation of Systems*, pages 228–243. Springer.

Membrey, P., Plugge, E., Hawkins, T., and Hawkins, D. (2010). *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Springer.

Nadiminti, K., De Assunçao, M. D., and Buyya, R. (2006). Distributed systems and recent innovations: Challenges and benefits. *InfoNet Magazine*, 16(3):1–5.

Nejati Sharif Aldin, H., Deldari, H., Moattar, M. H., and Razavi Ghods, M. (2019). Consistency models in distributed systems: A survey on definitions, disciplines, challenges and applications. *arXiv e-prints*, pages arXiv–1902.

Özsu, M. T. and Valduriez, P. (1996). Distributed and parallel database systems. *ACM Computing Surveys (CSUR)*, 28(1):125–128.

Schultz, W., Avitabile, T., and Cabral, A. (2019). Tunable consistency in mongodb. *Proceedings of the VLDB Endowment*, 12(12):2071–2081.

Simon, S. (2000). Brewer's cap theorem. *CS341 Distributed Information Systems, University of Basel (HS2012)*.

Wang, H., Li, J., Zhang, H., and Zhou, Y. (2014). Benchmarking replication and consistency strategies in cloud serving databases: Hbase and cassandra. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 71–82. Springer.