Análise do impacto no desempenho da replicação de um banco de dados distribuído em uma nuvem computacional

Rennan Felizardo, Marcus Carvalho

[1]Universidade Federal da Paraíba (UFPB)
Departamento de Ciências Exatas
Rio Tinto - Paraíba - Brasil

{rennan.felizardo,marcuswac}@dcx.ufpb.br

Resumo. *Os bancos de dados distribuídos (BDD) de caráter NoSQL alinhados com a computação em nuvem surgiram para suprir os problemas enfrentados com os bancos de dados relacionais no armazenamento e gerenciamento de grande volume de dados dos mais variados tipos. As empresas buscam cada vez mais estes tipos de soluções, onde pagam apenas pelo que usam, porém o planejamento para migrar para este tipo de serviço não é uma tarefa trivial. Neste trabalho foi feita uma análise de desempenho para a implantação de um BDD replicado na nuvem, obtendo resultados de qual modelo de consistência de dados e quantas máquinas e qual tipo de configuração se obtem um melhor desempenho.*

Abstract. *NoSQL-driven distributed databases (DDBs) aligned with cloud computing have emerged to address the problems faced by relational databases in storing and managing large volumes of data of a variety of types. Companies increasingly seek these types of solutions, which pay only for what they use, but planning to move to this type of service is not a trivial task. In this article, a performance analysis was performed for the implementation of a replicated DDB in the cloud, obtaining results of which model of data consistency and how many machines and which type of configuration one obtains a better performance.*

1. Introdução

Com o avanço da tecnologia, a quantidade de informações que tem sido armazenada e processada ao longo dos últimos anos pelas mais diversas aplicações tem crescido de forma exponencial. A imensa popularidade de aplicações intensivas em dados como Facebook, Twitter, Amazon e Google contribuíram para aumentar a exigência de armazenamento e processamento de dados de forma eficiente.

Segundo o estudo realizado pela empresa Cisco [Cisco 2016], estima-se que em 2020 o tráfego IP global já terá ultrapassado os zettabyte (ZB; ou 1024 exabytes) atingindo 2.3ZB. Ainda segundo o mesmo estudo, estima-se que o número de dispositivos conectados à rede IP será três vezes maior do que a população global em 2020, somando

Trabalho de Conclusão de Curso do discente Rennan Felizardo dos Santos, sob a orientação do docente Marcus Williams Aquino de Carvalho submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de Bacharel em Sistemas de Informação.

26,3 bilhões de aparelhos. O resultado dessa proliferação é a geração de grandes quantidades de dados, que precisam ser eficientemente criados, armazenados, compartilhados e analisados. Esse grande volume de dados, dos mais variados tipos (estruturados, semi-estruturados e não estruturados) vem sendo chamado de *Big Data*, que requerem uma poderosa gestão e soluções tecnológicas.

Para lidar com essa grande quantidade de dados, as aplicações podem aumentar o poder de processamento de suas máquinas, adicionando mais processadores, memória ou discos rígidos, por exemplo. Esta abordagem é conhecida como provisionamento vertical (*vertical scaling*, em inglês). No entanto, esta solução que geralmente é adotada para escalar bancos de dados relacionais (BDRs) não é suficiente para lidar com *Big Data*, pois chega a um ponto que não há como aumentar a capacidade de desempenho por causa da limitação de atualizações de *hardware*, além de implicar na compra de equipamentos caros [Lemos and Figueiredo 2014]. Outra alternativa seria a de utilizar várias máquinas trabalhando em conjunto, onde os dados estariam distribuídos e replicados nas máquinas. Esta outra abordagem é conhecida como provisionamento horizontal (*horizontal scaling*, em inglês). Essa solução seria mais viável financeiramente, pois tende a ter custos mais baixos. Porém, é um tipo de provisionamento inadequado para os BDRs tradicionais. Com as limitações dos sistemas de gestão de bases de dados (SGBDs) tradicionais, onde os dados geralmente ficam centralizados, fez-se necessário a busca de novas soluções que atendessem a essas novas necessidades. Neste sentido, surgiram os sistemas de gerenciamento de banco de dados distribuídos (SGBDDs) [Vaish 2013].

Ainda que a abordagem distribuída seja mais barata que a primeira, os custos para implantação e manutenção de toda a infraestrutura são altos e ainda pode ocorrer: das máquinas ficarem ociosas (sobre-provisão); das máquinas não suportarem mais a demanda (sub-provisão); e a elevação dos custos com pessoas, equipamentos e licença dos programas. Com a intenção de transferir custos e responsabilidades, as empresas estão migrando seus serviços para nuvens computacionais, onde é fácil escalar a capacidade computacional ao longo do tempo e os gastos são de acordo com a utilização dos recursos de TI (pay-as-you-go, em inglês) [Taurion 2009].

Armazenar e processar grandes volumes de dados requer escalabilidade, tolerância a falhas e disponibilidade. A computação em nuvem oferece tudo isso por meio da virtualização de hardware e elasticidade de recursos [Armbrust et al. 2009]. Assim, *Big Data* e computação em nuvem são dois conceitos compatíveis, já que a nuvem permite que grandes dados estejam disponíveis, escaláveis e tolerantes a falhas. Contudo, para implantar um SGBDD na nuvem, que atenda ao nível de qualidade de serviço desejado com o menor custo possível, deve-se haver um planejamento da quantidade e das configurações ideais das máquinas virtuais (VM, do inglês *Virtual Machines*) usadas para hospedar o SGBDD e que atendam a estes requisitos, que não é uma tarefa trivial. Além disso, é difícil saber o impacto de diferentes configurações de consistência no desempenho.

Posto isso, o objetivo deste trabalho é avaliar o desempenho da replicação de um banco de dados NoSQL distribuído em uma nuvem computacional, analisando o impacto no desempenho de diferentes modelos de consistência de dados, como também de diferentes configurações e quantidade de VM's usadas para implantar o SGBDD. Mais especificamente, foi avaliado o banco de dados distribuído MongoDB com diferentes cargas de trabalho, implantado no ambiente de nuvem da *Amazon Lightsail*, para responder pergun-

tas de pesquisa como:

- Qual configuração possui melhor desempenho para um mesmo custo: usando muitas máquinas virtuais de baixa capacidade ou poucas de alta capacidade?
- Qual é a diferença no desempenho ao se usar replicação com consistência forte e consistência eventual nos dados?

O restante do artigo está organizado da seguinte maneira. A seção 2 é aborda a fundamentação teórica para melhor entendimento do assunto abordado neste artigo. A seção 3 especifica a metodologia utilizada. A seção 4 conta com os resultados obtidos e a análise dos mesmos. Por fim, a seção 5 apresenta as considerações finais e os trabalhos futuros.

2. Fundamentação Teórica

Desde a sua criação em 1970, os bancos de dados relacionais tem sido a principal alternativa para o armazenamento de dados, eles foram criados com a intenção de superar problemas que eram enfrentados com o sistema de banco de dados hierárquico e de rede na época [Elmasri and Navathe 2010]. Porém, com o constante crescimento no volume de dados armazenados e analisados e o surgimento de aplicações que manipulam dados complexos como imagens e vídeos, se tornou cada vez mais desafiador gerenciar, escalar e armazenar esses dados nos bancos de dados relacionais, que possuem limitações em sua escalabilidade horizontal e flexibilidade. Para superar essas limitações e atender as novas necessidades em armazenamento e recuperação de dados, um modelo de banco de dados não relacional foi desenvolvido com um conjunto de novos recursos, conhecido como banco de dados NoSQL (*Not only SQL*), isto porque este modelo de banco não utiliza apenas SQL como linguagem de consulta. Os bancos de dados não relacionais surgiram como uma tecnologia inovadora, mas não vieram para substituir os bancos de dados relacionais, surgiram como uma alternativa para o armazenamento e gerenciamento de dados em aplicações que não são adequadas para os BDRs, uma aplicação pode usar um banco de dados não relacional como um complemento de um BDR ou pode usar um banco de dados exclusivamente não relacional.

Em comparação com os BDRs, os banco de dados NoSQL são mais flexíveis e escaláveis horizontalmente, isto é, permitem que novos nós sejam adicionados ou removidos a sua infraestrutura no momento que for necessário e não possuem uma estrutura de dado pré-definida (*schema-less*), os registros podem ter campos diferentes conforme necessário, o que permite a construção de um modelo de dados dinâmico e flexível [Stonebraker 2010]. A arquitetura distribuída não só garante a escalabilidade horizontal, como também melhora a confiabilidade do sistema, mantendo várias cópias de dados.

Em sistemas distribuídos de gerenciamento de dados, são usadas duas técnicas básicas para alcançar o escalonamento horizontal:

1. *Sharding* - É uma abordagem sob a qual cada nó do sistema contém um pedaço de dados chamado *shard* e executa operações nessa peça. Este método é a principal ferramenta para manter a escalabilidade horizontal;
2. Replicação - Onde os mesmos dados são armazenados em diferentes nós do sistema. Essa técnica aumenta a confiabilidade e ajuda a neutralizar falhas.

Para alcançar este tipo de escalonamento, os bancos NoSQL diferente dos BDRs não seguem as propriedades transacionais ACID, onde as transações da base de dados devem ser atômicas, consistentes, isoladas e duráveis [Vaish 2013]. Essas propriedades são muito restritivas para os bancos de dados NoSQL e impossíveis de alcançarem em um banco de dados distribuído (BDD) como sugerido no teorema de Brewer ou teorema CAP, onde diz que um sistema distribuído consegue garantir apenas duas das três propriedades que são capacidade, disponibilidade e tolerância a partição (do inglês, *Capability, Availability* e *Parition tolerance*) [Brewer 2000]. Para resolver este problema os BDDs utilizam as propriedades transacionais BASE (*Basically Available, Soft State* e *Eventually consistent*) em seu modelo de consistência, onde estão basicamente disponíveis e não estão continuamente consistentes, mas o sistema de armazenamento garante que, se nenhuma mudança for feita em um dado, eventualmente todas as cópias distribuídas deste dado retornarão à última informação atualizada, ou seja, se alcança a consistência. A disponibilidade é conquistada por meio de tolerância a falhas, o que evita o comprometimento do sistema como um todo [Pritchett 2008].

A escalabilidade e a flexibilidade são uma das principais características da computação em nuvem (*cloud computing*), a facilidade em poder aumentar ou diminuir os recursos computacionais para atenderem de acordo com a demanda pagando apenas o que foi utilizado está atraindo cada vez mais empresas dos mais diferentes portes a utilizarem este modelo de serviço [Abadi 2009]. Ao utilizar um SGBD em nuvem as empresas transferem responsabilidades como manutenção e gerenciamento para os provedores de serviço de nuvem, além dos custos com a infraestrutura, visto que os recursos de TI podem ficar rapidamente obsoletas [Sousa et al. 2010]. Os bancos NoSQL são capazes de tirar vantagem de novos *clusters* e nós de forma transparente, sem exigir um gerenciamento de banco de dados adicional ou distribuição manual de informações. Como administrar um banco de dados com uma grande quantidade de dados pode ser uma tarefa difícil, os bancos de dados NoSQL foram projetados para automaticamente gerenciar e distribuir os dados, recuperar de falhas e reparar todo o sistema [Gajendran 2012].

Na literatura, alguns trabalhos fizeram análises comparativas de SGBDs. Parker [Parker et al. 2013] faz uma comparação entre o banco de dados de caráter NoSQL, o MongoDB, com o banco de dados relacional MS SQL Server. Executaram experimentos usando um banco de dados estruturado de tamanho modesto para determinar o desempenho do banco de dados relacional para o banco de dados NoSQL. Como resultado, o MongoDB se saiu melhor.

Abramova e Bernardino [Abramova and Bernardino 2013] fazem uma comparação entre os banco de dados NoSQL MongoDB e Cassandra em servidores de menor capacidade, observando apenas o tempo de execução para a realização das operações. Como resultado, o Cassandra obteve melhores resultados para quase todos os cenários.

Politowski e Maran [Politowski and Maran 2014] fazem uma comparação do desempenho de consultas entre os bancos de dados MongoDB e PostgreSQL. Concluíram que o MongoDB obteve os melhores resultados, exceto em casos onde a aplicação necessita de uma segurança robusta, onde o PostgreSQL é melhor.

3. Metodologia

Segundo Fonseca [Fonseca 2002], a análise quantitativa parte de dados brutos que podem ser quantificados, que recorre a linguagem matemática para descrever e estabelecer relações entre variáveis e causas de um fenômeno. Com isso, foi realizado uma análise quantitativa, utilizando a ferramenta neutra para geração de cargas de trabalhos (*workloads*) *Yahoo! Cloud Serving Benchmark* (YCSB), um *benchmark* desenvolvido pela *Yahoo!* para sistemas de nuvens computacionais. Realizando as principais operações (leitura, inserção, atualização e varredura de dados) que são executadas em um banco de dados, utilizando um dos BDD mais conhecidos, o MongoDB. A fim de montar a infraestrutura necessária para a realização desta análise, foi utilizado o ambiente de nuvem da *Amazon*, a *Amazon Lightsail* como provedor de nuvem computacional.

A geração dos dados foi dada através da execução de diferentes perfis de *workloads* disponíveis no *benchmark* YCSB, a execução ocorreu em diferentes ambientes de testes que foram divididos de acordo com as configurações das máquinas virtuais (VMs) e quantidade das mesmas. A análise de desempenho se deu através da coleta dos dados gerados pelo *benchmark*, observando as métricas de vazão e latência do sistema.

3.1. Banco de dados distribuído

O BDD escolhido para esta análise foi o MongoDB, por se tratar de um banco de caráter NoSQL, com código-fonte aberto (*open source*, em inglês), escalável, que fornece alta performance, disponibilidade e partição automática [Mongo 2017]. Além de ser o banco NoSQL mais popular do mundo [DB-Engine 2017], possuindo uma comunidade bastante ativa.

O MongoDB é um banco orientado a documentos, que manipula documentos do tipo JSON (*JavaScript Object Notation*) livre de esquema, ou seja, os documentos podem conter diferentes conjuntos de campos, onde um conjunto de documentos são chamados de coleções [Mongo 2017].

Para esta análise o MongoDB foi utilizado no modo de replicação dos dados. A replicação no MongoDB funciona da seguinte forma: cada conjunto de réplicas (*replica set*, em inglês) pode conter um membro primário; vários membros secundários; e um árbitro. O primário é o único membro no conjunto de réplicas que recebe operações de escrita. O MongoDB aplica operações de escrita no primário e, em seguida, grava as operações no *oplog*[1] do primário. Os membros secundários replicam esse *oplog* e aplicam as operações a seus conjuntos de dados. Todos os membros do conjunto de réplicas podem aceitar operações de leitura, mas para esta análise, as operações de leitura foram direcionadas para o membro primário, o que garante uma consistência forte. Se o primário atual não estiver disponível, uma eleição determina o novo primário. Conjuntos de réplicas com um número par de membros pode ter um árbitro para adicionar um voto em eleições de primário [Mongo 2017].

As operações de leitura, por padrão, são direcionadas ao membro principal. No entanto, o MongoDB permite que as configurações de leitura sejam alteradas [Mongo 2017]. Para este trabalho, dois tipos foram adotados, com o objetivo de analisar o impacto no desempenho ao realizar estas configurações:

[1]Log de operações do MongoDB: https://docs.mongodb.com/manual/core/replica-set-oplog

- Primária - Todas as operações de leitura serão direcionadas para o membro primário, o que garante uma forte consistência;
- Secundária - Todas as operações de leitura serão direcionada(s) para o(s) membro(s) secundário(s), o que garante uma consistência eventual.

3.2. Benchmark

Foi utilizado um dos *frameworks* mais relevantes para avaliação de desempenho de banco de dados NoSQL em serviços de nuvem, o *Yahoo! Cloud Serving Benchmark* (YCSB), *benchmark* desenvolvido pela *Yahoo!* [Cooper et al. 2010].

3.2.1. Carga de trabalho

O *benchmarck* YCSB, realizou as seguintes operações no banco:

- Insert - inserir um novo registro;
- Update - atualizar um registro mudando o valor de um campo
- Read - ler o registro de um campo aleatoriamente escolhido ou de todos os campos;
- Scan - varredura de registros em ordem, começando em uma chave de registro escolhida aleatoriamente. O número de registros a serem percorridos é escolhido aleatoriamente.;

O usuário pode criar diferentes perfis personalizados de cargas de trabalho com as operações desejadas para cada perfil, além disso, o *benchmark* oferece cinco perfis de cargas de trabalho já pré-definidos, os quais foram utilizados nesta avaliação e são exibidos na Tabela 1.

Tabela 1. *Workloads* disponíveis no YCSB

Workload	**Operações**
A - Update Heavy	Read: 50% Update: 50%
B - Read Heavy	Read: 95% Update: 5%
C - Read Only	Read: 100%
D - Read latest	Read: 95% Insert: 5%
E - Short Ranges	Scan: 95% Insert: 5%

Para cada *workload* foram executados 1.000 operações, que é o valor padrão do *benchmark*, mas que também pode ser ajustado.

3.3. Ambiente de Execução

Foi utilizado o ambiente de nuvem computacional privada da *Amazon*, a *Amazon Lightsail*, que fornece servidores virtuais privados pré-configurados de forma prática e rápida [Amazon 2017], características essas que o fez ser escolhido para esta análise.

3.4. Métricas

As métricas de desempenho consideradas importantes para esta avaliação, são as seguintes:

- Vazão;
- Latência;

3.5. Cenários

Para esta análise, foi utilizado como referência o preço e as configurações das máquinas virtuais que o provedor de nuvem computacional *Amazon Lightsail* oferece.

Tabela 2. Configurações das VMs disponibilizados pela *Amazon Lightsail* e seus respectivos preços

Instância	**Núcleos**	**RAM(GB)**	**HD SSD (GB)**	**Preço(R$/hora)**
VM1	1	0,512	20	0,007
VM2	1	1	30	0,013
VM3	1	2	40	0,027
VM4	2	4	60	0,054
VM5	2	8	80	0,108

Para esta avaliação foram escolhidas as instâncias do tipo: VM1; VM3; VM4 e VM5. As instâncias do tipo VM1 foram escolhidas exclusivamente para fazer o papel de árbitro nos conjuntos de réplicas. Os conjuntos de réplicas foram divididos por: tipo de instância; e por número de réplicas, onde o número de réplicas de um conjunto será equivalente em suas configurações de memória RAM a outros conjuntos de réplicas. Em cada conjunto serão realizadas as operações que contém em cada perfil de carga de trabalho do *benchmark* YSCB, cada perfil será executado 10 vezes, para garantir maior confiança nos resultados, onde uma série de 10 será com as configurações de consistência forte e outra eventual. Posto isso, os conjuntos de réplicas foram os seguintes:

Tabela 3. Configurações dos conjuntos de réplicas

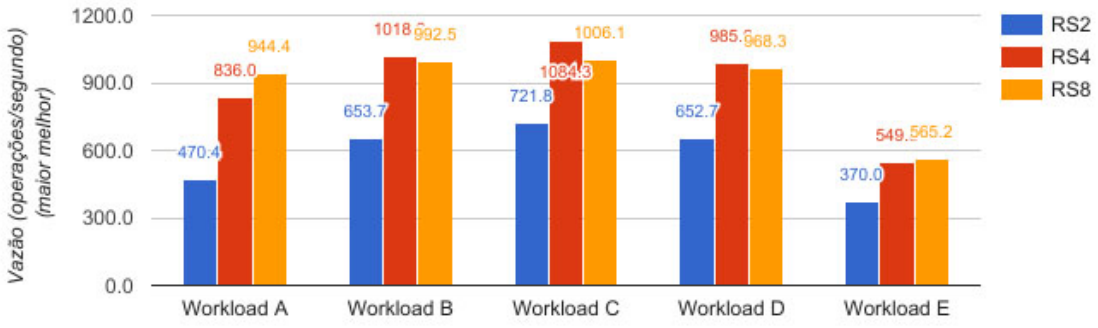
Conjunto de réplicas	**Tipo de instância**	**Número de membros**	**Possui árbitro**
RS2	VM3	4	sim
RS4	VM4	2	sim
RS8	VM5	1	não

Os árbitros não estão sendo contabilizados como membros dos conjuntos de réplicas, devido ao fato de não influenciarem nos resultados obtidos nas aplicações das cargas de trabalho.

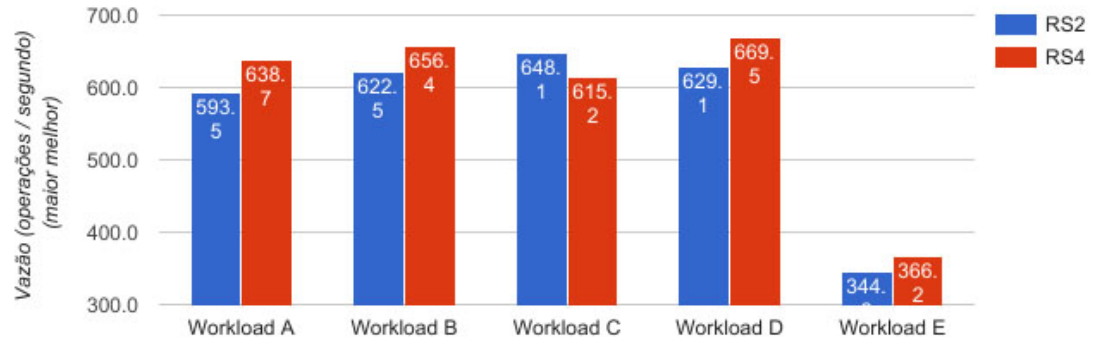
4. Avaliação de Desempenho

Nesta avaliação de desempenho, primeiro será analisado a métrica da vazão e em seguida as latências de leitura, atualização, inserção e varredura.

Na Figura 1, têm a vazão média para os diferentes tipos de carga de trabalho executados nos *replica sets* com o MongoDB. O cenário da Figura 1a é configurado com um modelo de consistência forte, onde é observado que os *replica sets RS8* e *RS4*, por possuírem o mesmo número de núcleos obtiveram um desempenho parecido, diferente do *RS2* que possui apenas um núcleo. Além disso, o *RS2* é o *replica set* que possui o maior número de réplicas e isso faz com que a vazão diminua, porque a réplica primária possui menor capacidade e há um *overhead* maior de atualização das réplicas integrantes do seu *replica set*.



(a) Consistência forte.



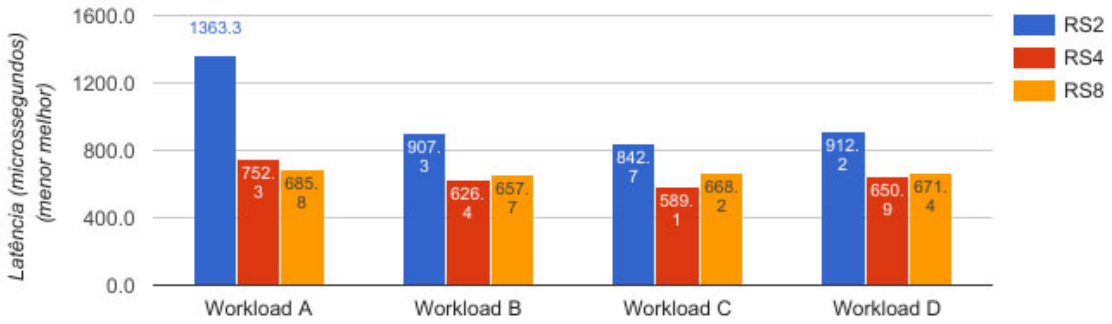
(b) Consistência eventual.

Figura 1. Vazão média para diferentes tipos de *Workloads* e seus respectivos *replica sets* com o MongoDB configurado para ter uma consistência forte e eventual.

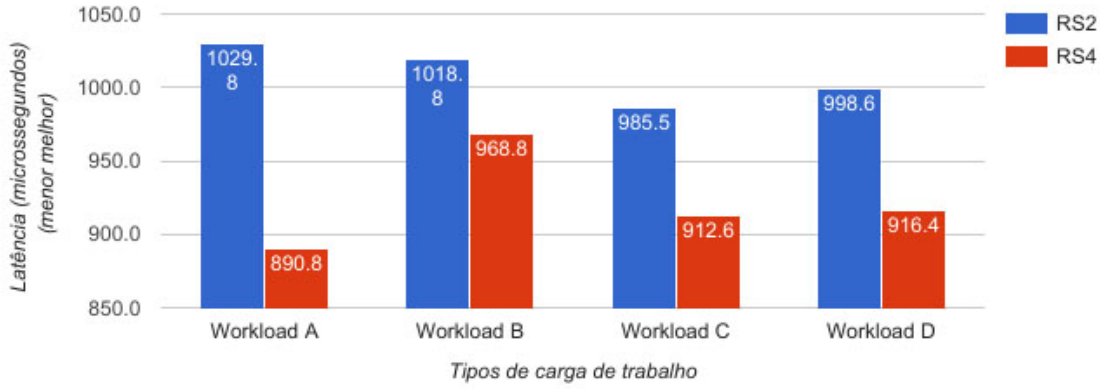
Em comparação com a Figura 1b, na qual o MongoDB foi configurado com uma consistência eventual onde operações de leitura também são executadas nas réplicas secundárias, o *replica set RS2* melhorou o seu rendimento em 56% em comparação ao seu desempenho no *Workload A* e conseguiu superar o *RS4* no *Workload C* isto porque possui um maior número de réplicas secundárias que conseguem atender uma maior demanda neste tipo de configuração.

A Figura 2 apresenta a latência média de leitura para os diferentes tipos de carga

de trabalho executados nos *replica sets* com o MongoDB. Na Figura 2a, configurado com consistência forte, observa-se que o *replica set RS2* foi o que apresentou a maior média de latência em todos os tipos de carga de trabalho, principalmente no *workload A* onde obteve a média de latência muito superior em comparação aos outros conjuntos de réplicas. Já os *replicas sets RS4* e *RS8* mantiveram uma média de latência próximos um do outro. Nas cargas de trabalho *B*, *C* e *D* onde a porcentagem de operações de leitura é mais alta, o *replica set RS4* teve um melhor desempenho, mesmo com a configuração de memória (4GB) menor que a do *RS8* (8GB).



(a) Consistência forte.

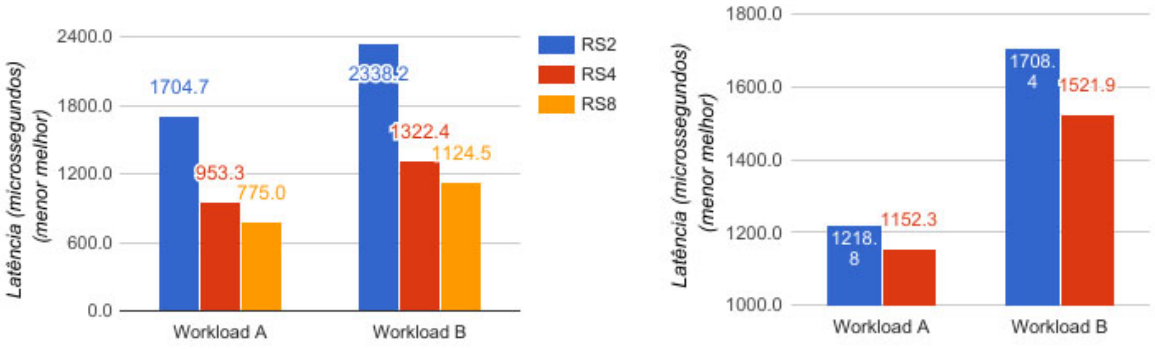


(b) Consistência eventual.

Figura 2. Latência média para as operações de leitura com consistência forte e eventual.

Na Figura 2b, com consistência eventual, o *replica set RS4* se saiu melhor em todos os tipos de cargas de trabalho em comparação ao *RS2*, obtendo o melhor resultado no *workoad A* e maior diferença em relação ao *RS2*, mostrando que as configurações das máquinas influenciam fortemente, como já era esperado. O *replica set RS2* obteve seu melhor resultado no *workload C*, onde são realizados apenas operações de leitura, mas em comparação com o seu desempenho quando a configuração de consistência é forte, obteve uma melhora apenas no *workload A*, diferente do *RS4* que teve um aumento na latência em todas as cargas de trabalho, mostrando que a configuração de consistência influenciou no seu desempenho.

A Figura 3 apresenta a latência média de atualização para diferentes tipos de carga. Na Figura 3a, com consistência forte, observa-se que a medida que o tamanho de memória aumenta, a latência diminui; dessa forma, o replica set *RS8* obteve o melhor desempenho em todas as cargas de trabalho. A carga de trabalho *B* foi a que exigiu maior esforço para as máquinas.



(a) Consistência forte.

(b) Consistência eventual.

Figura 3. Latência média para as operações de *update* com consistência forte e eventual.

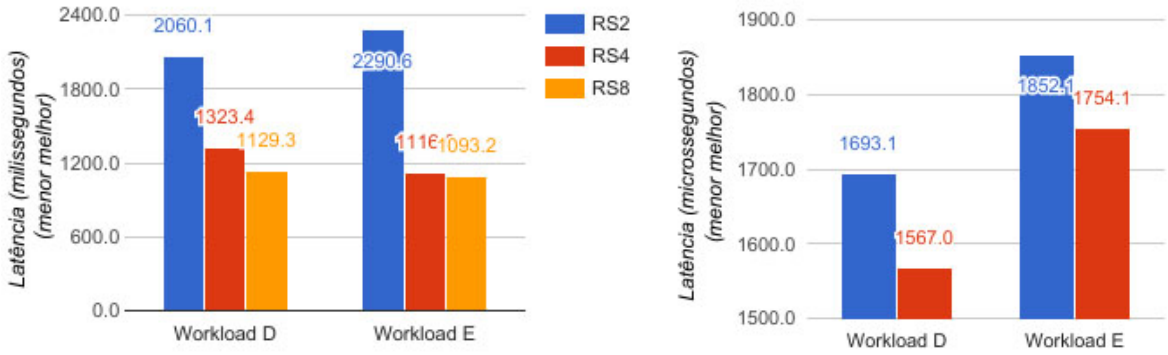
Na Figura 3b, com consistência eventual, nota-se que o *replica set RS4* foi o que obteve os melhores resultados para a latência de atualização, visto que possui maior número de núcleos e de tamanho de memória. Porém, ao comparar o desempenho das diferentes configurações de consistência, o conjunto de réplicas *RS2* foi que obteve o melhor resultado, o *replica set RS4* obteve um aumento considerável ne média de latência ao mudar a configuração de consistência em todas as cargas de trabalho e o *RS2* obteve essa diferença apenas no *workload B*.

A Figura 4 apresenta as médias de latências das operações de inserção de dados para diferentes tipos de carga de trabalho. Nota-se que os comportamentos foram os mesmos observados nas operações de atualização, visto que ambas são operações de escrita, tanto com consistência forte (Figura 4a) quanto com consistência eventual (Figura 4b).

A Figura 5 apresenta a latência média de varredura para diferentes tipos de carga de trabalho. Na Figura 5a, com consistência forte, é observado que a medida que o tamanho de memória aumenta, a latência média diminui. O *replica set RS2* foi o que teve maior esforço para realizar a operação de varredura, com uma diferença notável em relação aos outros conjuntos de réplicas.

Na Figura 5b, com consistência eventual, o *replica set RS4* continua a apresentar a melhor média de latência, porém mais uma vez, ao comparar as médias das diferentes configurações de consistência o *RS4* apresentou um aumento considerável em sua média, o mesmo aumento de média aconteceu para o *replica set RS2* porém a diferença não foi alta. Dessa forma, as operações de varredura custam mais para as máquinas ao se mudar a configuração de consistência de forte para eventual.

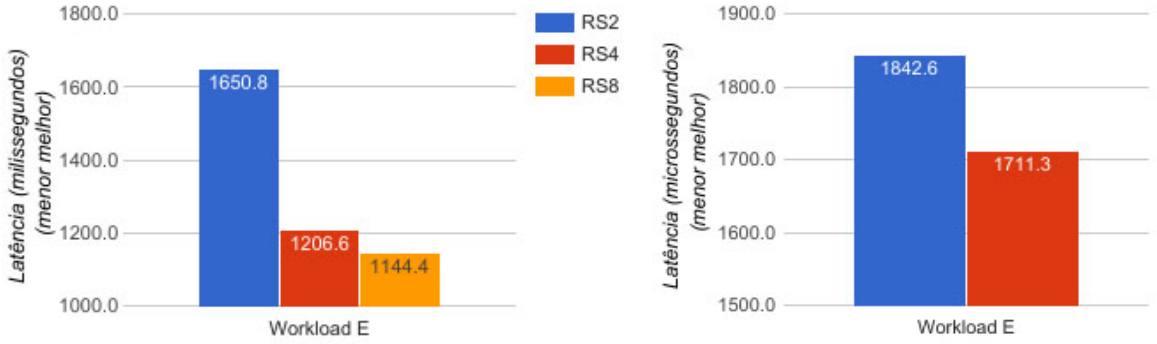
De todos os tipos de carga de trabalho, a carga que mais exige esforço das



(a) Consistência forte.

(b) Consistência eventual.

Figura 4. Latência média para as operações de *insert* com consistência forte e eventual.



(a) Consistência forte.

(b) Consistência eventual.

Figura 5. Latência média para as operações de *scan* com consistência forte

máquinas, é o *workload E*, visto que apresenta as menores médias de vazão tanto com configuração de consistência forte quanto eventual.

5. Conclusão e Trabalhos Futuros

Neste artigo, foram apresentadas as principais características de um banco de dados NoSQL distribuído e os benefícios ao se adotar este tipo de banco de dados nos serviços de nuvem computacional. Foram realizados diversos testes com o objetivo de avaliar o desempenho de um banco de dados distribuído com muitas máquinas de baixa capacidade e com poucas máquinas de capacidade maior, como também foi verificado o impacto no desempenho ao garantir forte ou fraca consistência nos dados. Para realizar esta análise, foi utilizado o banco de dados *MongoDB* na nuvem computacional da *Amazon Lightsail*, usando a ferramenta de *benchmark YCSB* desenvolvido pela *Yahoo!* em diferentes configurações de *VM*.

Após coleta e análise dos resultados nos testes de carga realizados neste trabalho, foi observado que a capacidade de CPU tem mais impacto no desempenho do que a capacidade de memória. Como o volume de dados testado na avaliação não era muito grande, uma VM de tamanho médio já tinha tamanho suficiente para armazenar os dados em memória. Observou-se também que usando o modelo de consistência forte de dados, onde todas as operações são feitas na réplica primária, é melhor ter uma réplica primária em uma *VM* com grande capacidade e ter poucas réplicas secundárias, visto que todas operações são concentradas no membro primário e que ele deve atualizar todas réplicas secundárias a cada nova escrita. Além disso, ao realizar as operações de leitura de dados apenas na réplica primária, é possível realizar *caching* dos dados de forma mais eficiente; o mesmo não acontece quando se adota um modelo de consistência eventual, onde as operações de leitura são balanceadas também para as réplicas secundárias, reduzindo a capacidade de *caching* nas operações de leitura.

No entanto, nem sempre o modelo de consistência forte tem melhor desempenho. Para o *Workload A*, que possui metade das operações de leitura e outra metade de atualização, a réplica primária fica ocupada fazendo operações de escrita, que exige operações em disco, e atualizando as réplicas secundárias. Desta forma, as réplicas secundárias serão mais úteis pois poderão processar as operações de leitura enquanto a réplica primária está sobrecarregada; isto dá indícios que a consistência eventual pode trazer ganho de desempenho quando a carga possui uma porcentagem considerável de escrita, mesmo reduzindo a capacidade de *caching* na réplica primária. Vale salientar que o volume de dados nos testes realizados foi muito baixo. Provavelmente, a replicação com consistência eventual também pode trazer um ganho maior de desempenho quando o volume de dados for grande, pois o *caching* na réplica primária, no modelo de consistência forte, se tornará menos eficiente.

Apesar de ter um desempenho menor na maior parte dos testes, utilizar um nível maior de replicação com várias máquinas traz ganhos na disponibilidade e integridade dos dados. Porém, essa garantias trazem impacto no desempenho, como visto nos resultados, que mostrou que ter uma única *VM* com alta capacidade computacional é melhor do que ter várias *VMs* de baixa capacidade, para um mesmo custo na nuvem computacional e para os testes realizados, que possuem um volume de dados pequeno.

Possíveis trabalhos futuros podem ser elaborados a partir deste tema, como: avaliar o impacto no desempenho nas diferentes preferências de leitura do *MongoDB*; avaliar o desempenho do *MongoDB* com um maior volume de dados; avaliar o desempenho com a técnica de *sharding*; fazer uma análise semelhante de desempenho com outros SGBDs e comparando os resultados.

Referência

Abadi, D. J. (2009). Data management in the cloud: Limitations and oportunities. *IEEE Data Eng. Bull*, pages 32:3–12.

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, New York, NY, USA. ACM.

Amazon (2017). Amazon lightsail. `https://amazonlightsail.com/`. Acessado em 26/05/2017.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Brewer, E. A. (2000). Towards robust distributed systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7–, New York, NY, USA. ACM.

Cisco (2016). Cisco VNI forecast and methodology, 2015-2020. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`. Acessado em 03/10/2016.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA. ACM.

DB-Engine (2017). Db–engines ranking. `http://db-engines.com/en/ranking`. Acessado em 24/03/2017.

Elmasri, R. and Navathe, S. (2010). *Fundamentals of database systems*. AddisonWesley, 6th edition.

Fonseca, J. J. S. (2002). Metodologia da pesquisa científica. *Fortaleza: UEC*.

Gajendran, S. K. (2012). A survey on NosSQL databases. *University of Illinois*.

Lemos, P. H. d. S. and Figueiredo, P. S. (2014). Uma Análise dos Novos Sistemas de Bancos de Dados Relacionais Escaláveis. `http://monografias.poli.ufrj.br/monografias/monopoli10010084.pdf`. Monografia (Bacharel em Engenharia de Computação e Informação), UFRJ (Universidade Federal do Rio de Janeiro), Rio de Janeiro, Brasil. Acessado em 30/03/2017.

Mongo (2017). Mongo 3.4 manual. `https://docs.mongodb.com/manual/`. Acessado em 24/03/2017.

Parker, Z., Poe, S., and Vrbsky, S. V. (2013). Comparing nosql mongodb to an sql db. In *Procedimentos da 51ª Conferência ACM Sudeste*, page 5. ACM.

Politowski, C. and Maran, V. (2014). Comparac¸ao de performance entre postgresql e mongodb. *Universidade Regional do Noroeste do Estado do Rio Grande do Sul*.

Pritchett, D. (2008). Base: An acid alternative. *Queue*, 6(3):48–55.

Sousa, F. R. C., Moreia, L. O., Macêdo, J. A. d., and Machado, J. C. (2010). Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. *SWIB*, pages 101–130.

Stonebraker, M. (2010). Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11.

Taurion, C. (2009). *Cloud Computing-Computação em Nuvem*. Brasport.

Vaish, G. (2013). *Getting started with NoSQL*. Packt Publishing Ltd.