Avaliação da Replicação e Tolerância a Falhas nos Sistemas Cassandra e MongoDB

Danilo Bovo[1], **Roger Duarte**[1], **João Eugenio Marynowski**[2]

[1]Pontifícia Universidade Católica do Paraná (PUC-PR)
Caixa Postal 17.315 – 80.242-890 – Curitiba – PR – Brazil

bovodanilo@gmail.com, roger_rrd@yahoo.com.br

[2]Universidade Federal do Parana (UFPR)
Caixa Postal 15.064 – 91.501-970 – Paraná – PR – Brazil

jeugenio@ufpr.br

Abstract. *More and more non-relational (NoSQL) database are used, supplying the need for high availability, storage, and high volume manipulation of unstructured data. There are several papers that evaluate NoSQL databases, but they do not cover fault tolerance characteristics, which are usually used in several computers. This paper aims to perform replication and fault tolerance tests on the main NoSQL systems, Cassandra and MongoDB. The tests follow a proposed methodology and the results show a low error rate in relation to the availability of the information during the execution of the workloads and injection of failures.*

Resumo. *Cada vez mais bancos não relacionais (NoSQL) são utilizados, suprindo a necessidade de alta disponibilidade, armazenamento e manipulação de grande volume de dados não estruturados. Existem vários artigos que fazem avaliação de sistemas NoSQL, porém não abrangem características de tolerâncias a falhas, que normalmente são empregadas em várias computadores. Este trabalho visa executar o teste de replicação e tolerância a falhas nos principais sistemas NoSQL, o Cassandra e MongoDB. Os testes seguem uma metodologia proposta e os resultados mostram baixa taxa de erros em relação à disponibilidade das informações durante a execução das cargas de trabalhos e injeção de falhas.*

1. Introdução

Bancos de dados não relacionais, isto é, NoSQL são utilizados cada vez mais para suportar a quantidade de dados gerada atualmente [Tudorica and Bucur 2011]. Esses bancos possuem escalabilidade horizontal, ou seja, é possível criar um *cluster* com N computadores para suportar a carga de dados e ampliar essa capacidade com a adição de novos computadores. Eles também possuem várias características importantes para o uso distribuído, tais como: ferramentas próprias para administração de banco de dados distribuídos, particionamento de dados, replicação e tolerância a falhas [Moniruzzaman and Hossain 2013] e [Tsuyuzaki and Onizuka 2012].

Normalmente a perda de dados é muito indesejada, objetivando, em caso de incidentes, tempos aceitáveis de perda de dados, ou *Recovery Point Objective* (RPOs[1]), extre-

[1]http://whatis.techtarget.com/definition/recovery-point-objective-RPO

mamente baixos. Então, é importante fazer o teste de replicação e tolerância a falhas em sistemas NoSQL. Existem diversos trabalhos que executam a avaliação de desempenho em bancos NoSQL, porém, poucos fazem o teste de replicação e tolerância a falhas.

Este trabalho propõe realizar testes de replicação e tolerância a falhas em dois dos principais sistemas NoSQL, o Cassandra e o MongoDB. Criamos uma metodologia de testes para isso, onde foi construído um *cluster* e executada as cargas de trabalho padrões do Yahoo! Cloud Serving Benchmark (YCSB)[2][Cooper et al. 2010]. Durante a execução da carga de trabalho, algumas falhas foram provocadas nos computadores do *cluster*, justamente para avaliar a disponibilidade dos dados.

Este documento é organizado da seguinte forma. A Seção 2 apresenta os fundamentos do Cassandra, MongoDB e da ferramenta YCSB, utilizada para realizar a carga de trabalho durante os experimentos. Na Seção 3 relatamos alguns trabalhos relacionados e na Seção 4 descrevemos a metodologia utilizada. A Seção 5 apresenta os experimentos e os resultados obtidos durante a execução dos testes de replicação e tolerância a falhas. Por fim, na Seção 6 são apresentadas as conclusões.

2. Fundamentos

Entre os sistemas gerenciadores de bancos de dados NoSQL mais utilizados, destacam-se o Cassandra e o MongoDB, segundo o DB-Engine Ranking[3]. Estes sistemas possuem modelo de replicação diferente, o Cassandra utiliza o formato *peer-to-peer* e o MongoDB *master-slave* [Abramova and Bernardino 2013]. Ainda apresentam modelos de banco de dados diferentes. O Cassandra é categorizado como orientado à coluna *Column-family* [Hewitt 2010] e o MongoDB é orientado a documento (*Document Store*) [Chodorow 2013]. Assim, apresentam-se como objeto de estudo deste trabalho e são mais detalhados a seguir.

2.1. Cassandra

Cassandra[4] é um sistema de gerenciamento de banco de dados NoSQL *opensource* desenvolvido pela Apache, em Java e é do tipo *Column-family*, onde os dados são armazenados em colunas ao invéz de linhas como no relacional [Hewitt 2010]. O Cassandra tem como principais propriedades: ser descentralizado, tolerante a falhas, fornecer alta disponibilidade, ter consistência configurável, e por ser facilmente escalável [Hewitt 2010].

Uma das principais características do Cassandra é a capacidade de escalonamento horizontal, e isso só possível devido ao particionamento dinâmico dos dados entre os nós do *cluster*. A arquitetura em anel também torna fácil a entrada de novos nós ao *cluster* [Lakshman and Malik 2010]. Todos os nós tem estrutura idêntica e desempenham o mesmo papel, além disso, comunicam-se em pares, alternando o seu par a cada comunicação, arquitetura chamada *peer-to-peer (P2P)* [Hewitt 2010].

O Cassandra utiliza a arquitetura em anel e particiona os dados entre os nós de forma que cada nó seja responsável por uma fatia dos dados. A distribuição é dinâmica percorrendo os nós em sentido horário. Além disso, cada nó tem a informação da

[2]https://github.com/brianfrankcooper/YCSB

[3]https://db-engines.com/en/ranking

[4]http://cassandra.apache.org

distribuição dos dados que são responsabilidade dos outros nós do *cluster* [Hewitt 2010]. O fator de replicação determina em quantos nós um dado será armazenado e existem duas estratégias de replicação [Abramova and Bernardino 2013]:

1. Estratégia simples: recomendado para quando não se tem distinção geográfica dos nós do *cluster*, estes são configurados em uma estrutura de anel, o primeiro nó é determinado pelo sistema e a replicação é feita nos nós subsequentes em sentido horário.
2. Estratégia de topologia de rede: recomendada para quando se tem o *cluster* implementado em mais de um *datacenter*, em lugares diferentes, pois é possível especificar o número de réplicas a ser utilizado em cada *datacenter*.

O protocolo *Gossip* é utilizado para identificar o nó falho e para substituí-lo rapidamente, minimizando a indisponibilidade [Hewitt 2010]. Existem o número de réplicas menos um disponíveis para este procedimento de recuperação, substituindo o nó falho no processamento [Hewitt 2010].

2.2. MongoDB

O MongoDB é um sistema de gerenciamento de banco de dados NoSQL *opensource* desenvolvido em C++ pela MongoDB, Inc[5]. Sua primeira versão foi lançada em 2007 e cita que hoje possui alta disponibilidade através de replicação e tolerância a falhas, escalabilidade horizontal, segurança *end-to-end*, suporte global e ferramentas de gerenciamento para automação, monitoramento e backup.

O MongoDB se destaca no seguimento orientado à documentos (*Document Store*) [Chodorow 2013]. Neste tipo de sistema os dados são gravados em forma de {chave, valor}, mas transformados e indexados como documentos. Cada documento pode então ser acessado através de sua respectiva chave. O agrupamento dos documentos é chamado de coleção e o formato de armazenamento de BSON (*Binary* JSON) [6].

O *cluster* do MongoDB possui os seguintes componentes: *shard*, *mongos* e *config server*. O primeiro, *shard*, é responsável pelo armazenamento dos documentos ou partes deles, uma vez que uma coleção pode ser particionada em vários *shards*. O conteúdo de um *shard* é armazenado em um conjunto de réplicas, *replica set*, que consiste em um conjunto de *hosts* que fazem replicação de todas as informações entre si. O segundo, *mongos*, faz o papel de *query router*, ou seja, a interface entre a aplicação e o *cluster* do MongoDB. Por fim, o terceiro, *config server*, é responsável pelo armazenamento de metadados e configurações específicas do *cluster*. O conteúdo armazenado no *config server* também deve ficar em um *replica set*.

A replicação segue o mecanismo *Master-Slave* e é identificado como *replica set* no MongoDB. Ao configurar um *replica set* todos os nós que farão parte do *shard* do *cluster* são informados. O *master* recebe todas as solicitações de escrita e faz a replicação aos demais membros do *cluster* de forma assíncrona. Em caso de problemas com o *master*, ocorre o *failover* automático, em que um dos *slaves* assume como *master* no *replica set*. O tempo de deteção de falha ocorre até no máximo em 10 segundos e tempo de eleição para de um novo *master* ocorre entre 10 e 30 segundos.

[5]https://www.mongodb.com/

[6]https://docs.mongodb.com/manual/

2.3. YCSB Benchmarking

O Yahoo! Cloud Serving Benchmark (YCSB)[7][Cooper et al. 2010] é uma ferramenta de avaliação de desempenho (*benchmarking*) empregada para bancos de dados do tipo NoSQL. O YCSB funciona em duas fases: primeiramente é executada a carga de dados inicial sobre um determinado banco de dados, e após é executado um conjunto de consultas (*workloads*) pré-configuradas e executadas através da ferramenta. Os *workloads* disponíveis são:

- ***Workload* A**: as operações são divididas em 50% de leitura e 50% de atualização;
- ***Workload* B**: as operações são divididas em 95% de leitura e 5% de atualização;
- ***Workload* C**: as operações são apenas (100%) de leitura;
- ***Workload* D**: novos registros são inseridos e são os mais populares;
- ***Workload* E**: breves conjuntos de registros são consultados em cada operação;
- ***Workload* F**: um registro é obtido e atualizado.

Os dados inseridos através da carga de dados no banco seguem o formato de chave "userXYZ", sendo XYZ um número aleatório, e 10 campos vinculados a essa chave com caracteres randômicos com 100 bytes cada. Totalizando assim aproximadamente 1,25kb por valor da coleção para cada chave.

3. Trabalhos relacionados

Com o crescimento na utilização de sistemas de gerenciamento de banco de dados NoSQL, muitos artigos são criados com o objetivo de executar o *benchmarking* destes, buscando identificar quais são os melhores. Existem alguns que buscam avaliar apenas o tempo total das operações, como [Abramova and Bernardino 2013], e outros que buscam dar ênfase no *throughput* do conjunto de operações, como [Tudorica and Bucur 2011].

Em [Abramova et al. 2014], os autores buscam comparar diferentes sistemas de gerenciamento de bancos de dados NoSQL com o YCSB. Os autores buscam avaliar a capacidade dos bancos de dados não relacionais considerando os tipos de sistemas e os mecanismos que estes utilizam para persistir os dados em disco.

Em [Tudorica and Bucur 2011], são executados *benchmarking* com o YCSB em sistemas NoSQL e SQL, objetivando avaliar o *throughput* e a diferença durante as operações. O artigo utiliza os *workloads* padrões do YCSB, demonstrando a relação entre latência e o número de operações por segundo. Em 75% dos casos, o Cassandra é o segundo banco com maior número de operações com menor latência, já o MongoDB não foi contemplado neste artigo.

No trabalho [Marynowski et al. 2015] são executadas modelagens de casos de falhas para testar a tolerância falhas em sistemas Hadoop. O método apresentado exige a modelagem do mecanismo de tolerância a falha do sistema a ser avaliado para então gerar os casos de falhas e avaliar o sistema. Este trabalho também não aborda o Cassandra e MongoDB.

O trabalho de Abranova e Bernardino [Abramova and Bernardino 2013] descreve os sistemas de gerenciamento de bancos de dados NoSQL, suas características e

[7]https://github.com/brianfrankcooper/YCSB/

princípios operacionais. Seu foco principal é comparar e avaliar o Cassandra e o MongoDB usando o YCSB. Este artigo é o que mais se aproxima do nosso trabalho, porém seu foco principal está no *benchmarking* dos dois sistemas.

Apesar da existência de diversos artigos avaliando o desempenho de sistemas NoSQL, é interessante avaliar a replicação e tolerância a falhas do Cassandra e MongoDB, sistemas muito requisitados atualmente. Nas seções seguintes são apresentadas a metodologia utilizada e os experimentos, seguidos dos resultados e considerações.

4. Metodologia

Utilizamos como base os trabalhos de Pankowski [Pankowski 2015] e Akon Dey [Dey et al. 2016] para criar uma metodologia para avaliar a replicação e a tolerância a falhas dos sistemas. O objetivo é simular falhas dos nós em um *cluster* durante a execução de cargas de trabalho (*workloads*).

As falhas aqui são consideradas interrupções dos serviços nos computadores e os *workloads* são os apresentados no YCSB, Seção 2.3, exceto os *workloads* G e H, que foram adicionados para envolver mais operações de atualização:

- ***Workload* G**: as operações são divididas em 5% de leitura e 95% de atualização.
- ***Workload* H**: as operações são apenas (100%) de atualização.

Esses dois *workloads* foram adicionados pois acredita-se que completam as possibilidades de carga de trabalho, melhor avaliando a replicação e tolerância a falhas durante o processo crítico de atualização.

Os passos para a realização dos experimentos seriam:

1. Crie um banco de dados e inserir um conjunto de dados;
2. Execute um *workload*: A, B, C, D, G ou H;
3. Após 1 minuto, falhe o nó primário do cluster;
4. Após 1 minuto, inicie o nó parado em 3 e falhe o novo nó primário;
5. Após 1 minuto, inicie o nó parado em 4 e falhe o novo nó primário;
6. Após 1 minuto, inicie o nó parado em 5 e falhe o novo nó primário;
7. ... e assim por diante.

Essa sequência é repetida até a finalização do *workload*. No Cassandra, que segue o modelo de replicação P2P, conforme Seção 2.1, os nós são eliminados de forma sequencial a cada passo. O tempo de 1 minuto foi estipulado pois no MongoDB, conforme Seção 2.2, o tempo de detectação de queda e eleição do nó primário é de no máximo 40 segundos, e no Cassandra são 10 segundos.

As medidas realizadas durante os experimentos são o número de operações, a latência e o número de erros. Uma operação é uma transação no banco de dados, isto é, uma leitura, atualização, inserção ou deleção de dados. Latência é o tempo levado para a execução completa de uma determinada operação. Erro é uma operação não finalizada com sucesso.

5. Experimentos

Nesta seção, apresentamos o ambiente criado para execução, a carga de dados realizada sobre os bancos e também as cargas de trabalhos e falhas executadas para avaliar a replicação e a tolerância da falhas. Por fim, são apresentados os resultados e uma análise comparativa.

5.1. Ambiente de execução

Utilizamos máquinas virtuais, *Virtual Machines* (VMs), da Amazon EC2[8] para os testes. A configuração das máquinas seguiu o perfil *t2.medium*, cada qual com 2 CPUs, 4GB de RAM e discos do tipo EBS[9], tecnologia criada pela Amazon para armazenamento em discos SSD ou HDD. Os detalhes da instalação podem ser vistos no documento que geramos de instalação do Cassandra e MongoDB[10].

Para o Cassandra foram instanciadas 4 VMs, já que neste sistema todos os nós têm a mesma função e estrutura idêntica. As quatro VMs são iguais, exceto a configuração de cada *host* para pertencerem ao mesmo *cluster*. Já para o MongoDB, utilizamos 5 VMs, instalando em uma o ConfigServer e o MongoS, e nas demais os nós pertencentes ao *replica set*. Todo o processo de teste de tolerância a falhas e replicação foi realizado nas 4 VMs do *replicat set*, resultando no mesmo número de VMs utilizadas para o Cassandra.

Utilizamos o sistema operacional CentOS Linux 7, com sistemas Cassandra 3.11.1 e MongoDB 3.6.2. A versão do YCSB foi a 0.12.0[11]. Realizamos todos os testes configurando o YCSB com 2 *threads*, pois as VMs utilizadas continham também 2 cores.

5.2. Carga de dados

Utilizamos os dados aleatórios do YCSB e os *workloads* A, B, C, D, G e H, conforme descrito em nas seções 2.3 e 4. A quantidade de linhas inseridas em cada banco de dados foi de 1 milhão e o número de operações durante cada *workload* também foi de 1 milhão.

Seguindo a metodologia de testes e o procedimento descrito na documentação do YCSB[12], primeiramente executamos a inserção de dados sobre os dois bancos de dados. Os resultados dessas inserções podem ser visualizados no gráfico da Figura 1. O tempo de execução no Cassandra foi muito superior, totalizando aproximadamente 18 minutos contra 6 minutos do MongoDB. Nenhum dos bancos de dados apresentou erros durante a execução do procedimento.

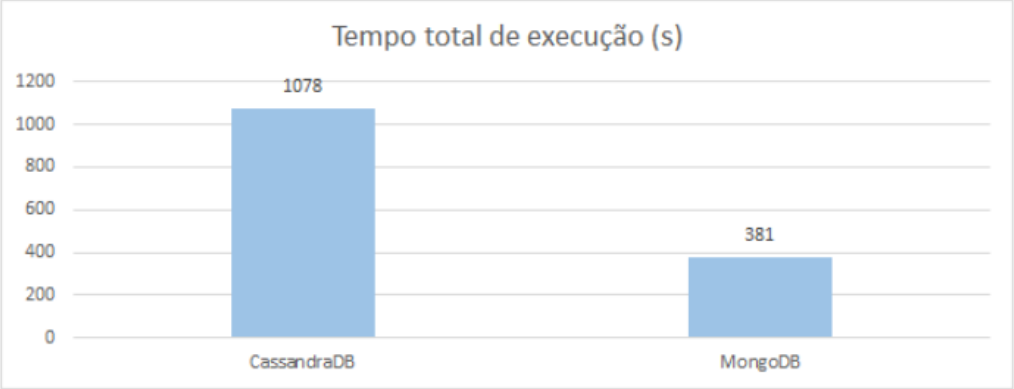

Figura 1. Tempo de execução durante o procedimento de carregamento inicial de dados.

[8]https://aws.amazon.com/pt/ec2
[9]https://aws.amazon.com/pt/ebs/
[10]http://bit.ly/2Fxbb07
[11]https://github.com/brianfrankcooper/YCSB
[12]https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload

5.3. Teste com o *Workload* A

Nas cargas de trabalhos, apresentamos os resultados obtidos utilizando tabelas. Nestas, demonstramos a quantidade total de operações, ou seja, o número transações (*SELECTS* e *INSERTS*) executadas sobre o de banco de dados, e também a quantidade de erros, que são transações não finalizadas com sucesso. A causa de uma transação não finalizar com sucesso pode ser diversa, desde por uma procura de um item que foi inserido mas não foi replicado até por um tempo de resposta excedido *timeout* gerado pelo sistema.

A Tabela 1 contém os resultados obtidos da execução do *workload* A, podemos observar que neste *workload* o número de operações de leitura e atualização ficou fixado em 50% do total para cada tipo. A latência média por operação do MongoDB foi melhor que o Cassandra, principalmente para a operação de leitura, com 766,65μs contra 2468,98μs. Do conjunto total de operações, podemos notar que o Cassandra não apresentou nenhum erro, seja para leitura ou atualização. Já o MongoDB, do conjunto de 500007 operações realizadas de atualização, ocorreram dois erros.

Tabela 1. Execução do *Workload* A

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de Erros
Cassandra	READ	500626	2468,98	0
	UPDATE	499374	1700,53	0
MongoDB	READ	499991	766,65	0
	UPDATE	500007	651,72	2

5.4. Teste com o *Workload* B

A Tabela 2 contém os resultados obtidos da execução do *workload* B. Nota-se que para os dois sistemas o total de erros foi igual a zero para as duas operações. Para a operação de leitura, o Cassandra apresentou latência média por operação de 1044,81μs e o MongoDB de 612,37μs. Já para a operação de atualização, o Cassandra foi poior, apresentando latência média por operação no valor de 893,89μs enquanto o MongoDB foi de 636,38μs. Neste caso, tanto o Cassandra como MongoDB, demonstram excelentes resultados em relação a tolerância a falhas.

Tabela 2. Execução do *Workload* B

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de erros
Cassandra	READ	950064	1044,81	0
	UPDATE	49936	893,89	0
MongoDB	READ	950241	612,37	0
	UPDATE	49759	636,38	0

5.5. Teste com o *Workload* C

A Tabela 3 apresenta os resultados obtidos no *workload* C, que possui apenas operações de leitura. Novamente os dois sistemas apresentaram excelentes resultados em relação a redundância, com nenhum erro durante a execução dos procedimentos. Em relação a

latência média por operação, o MongoDB apresentou latência média de 586,82μs contra 1216,39μs do Cassandra, sendo praticamente duas vezes melhor.

Tabela 3. Execução do *Workload* C

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de erros
Cassandra	READ	1000000	1216,39	0
MongoDB	READ	1000000	586,82	0

5.6. Teste com o *Workload* D

A Tabela 4 apresenta os resultados obtidos no *workload* D. Neste *workload* são executadas algumas inserções (5% do total de operações) e a maioria das operações de leitura são realizadas nestes dados inseridos, ou seja, os novos dados são os mais populares. Podemos observar que as latências sofreram pouca variação, entretanto, o número de erros durante as leituras do Cassandra foram muito superiores ao MongoDB, com 5285 erros. Apesar de ser um valor relativamente alto, comparando com o total de operações (944667), isso representa menos de 1%. Para as operações de inserção, nenhum erro foi gerada nos dois sistemas.

Tabela 4. Execução do *Workload* D

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de erros
Cassandra	READ	944867	1350,41	5285
	INSERT	49848	1163,98	0
MongoDB	READ	950033	824,17	12
	INSERT	49955	674,06	0

5.7. Teste com o *Workload* G

A Tabela 5 apresenta os resultados obtidos no *workload* G, em que 95% do total de operações são de atualização. Neste *workload*, o Cassandra não apresentou erros durante as operações, demonstrando assim ótima tolerância a falhas. Já o MongoDB apresentou 10 erros apenas para as operações de atualização. Em relação à latência, o MongoDB finalizou com uma latência inferior ao Cassandra, com média de 564,82μs para operações de leitura e 776,52μs para operações de atualização.

Tabela 5. Execução do *Workload* G

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de erros
Cassandra	READ	49709	1644,27	0
	UPDATE	950291	949,17	0
MongoDB	READ	49796	564,82	0
	UPDATE	950194	776,52	10

5.8. Teste com o *Workload* H

A Tabela 6 apresenta os resultados do *workload* H, que possui 100% das operações voltadas para atualização. Novamente neste caso, o Cassandra apresentou melhores resultados com nenhum erro gerado durante a execução do procedimento. Já o MongoDB apresentou 10 erros em relação ao total de operações.

Tabela 6. Execução do *Workload* H

Banco	Operação	Total de operações	Latência média por operação (μs)	Total de erros
Cassandra	UPDATE	1000000	892,68	0
MongoDB	UPDATE	1000000	745,18	10

5.9. Análise comparativa

De uma maneira geral, foi possível observar que os dois sistemas apresentaram baixa quantidade de erros durante os testes de replicação e tolerância a falhas, totalizando 5319 erros dentro de um conjunto de 12 milhões de operações, isto é, em 0,044% das operações. O Cassandra apresentou o maior número de erros, 5285, contra 34 do MongoDB. Mas o Cassandra apresentou erros em apenas um *workload*, já o MongoDB apresentou erros em quatro *workloads*. Porém, em todos os casos de erro do MongoDB, o número de erros em relação ao total de operações não ultrapassou a marca de 0,05%. Esses erros ocorreram no momento de eleição do novo nó primário dentro do *replica set*.

O *workload* D foi uma das melhores cargas de trabalho utilizadas, pois exige a sincronização imediata dos dados a medida que estes são adicionados aos bancos. Essa sincronização é necessária, pois as operações de leitura são executadas em grande parte sobre estes novos dados. Neste *workload*, o MongoDB apresentou um número de falhas muito inferior, com apenas 12 falhas contra 5285 do Cassandra. Neste caso, o MongoDB mostrou ser mais ágil em relação à replicação dos dados recentemente inseridos. Caso seja necessário, é possível reduzir os erros utilizando o nível de consistência configurável do Cassandra, aumentando o número de nós que precisam retornar sucesso em uma determinada operação. Desta forma, um número de nós maiores deve conter os dados recentemente inseridos, diminuindo as probabilidades de erros. Como mantivemos as configurações padrões do Cassandra, os testes com nível de consistência alterado não foram realizados.

Em relação a latência foi possível observar que em todos os 6 *workloads* o MongoDB obteve melhor resultado, desde a carga de dados. Esse comportamento difere do artigo descrito em [Abramova and Bernardino 2013], onde o Cassandra obteve melhores resultados, principalmente para os *workloads* A, B, C, G e H. Em [Abramova and Bernardino 2013], os testes foram realizados em sistemas não clusterizados e também com as configurações padrões dos dois bancos de dados. A única diferença ocorreu em relação ao número de operações, que em nosso caso foi de 1 milhão para cada *workload*, contra 700 mil de [Abramova and Bernardino 2013].

6. Conclusão

Foi realizada a instalação de dois principais sistemas de gerenciamento de bancos de dados NoSQL em ambiente distribuído, provendo a replicação e a tolerância a falhas e com

o objetivo de testar essas funcionalidades e avaliar o quanto são efetivas. Foi apresentada uma metodologia utilizando a Yahoo! Cloud System Benchmark (YCSB) para as cargas de trabalho enquanto falhas foram provocadas nos componentes dos sistemas para avaliar a replicação e a tolerância a falhas. Como resultado pudemos medir a quantidade de erros e a latência média das operações geradas durante os experimentos.

Os sistemas apresentaram erros durante os testes de replicação e tolerância a falhas, porém, esse valor foi menor que 0,05% das operações. O pior considerando o número de erros foi o Cassandra, mas o MongoDB apresentou erros em maior número de testes. Com esse resultado obtido, podemos concluir que os dois bancos implementam as duas funcionalidades alvos de nosso estudo, provendo alta disponibilidade e redundância dos dados armazenados, mas que embora um valor baixo de ocorrências de erros, este pode envolver dados valiosos e que precisa ser levado em conta para que seja evitado.

Como trabalho futuro, seria interessante aplicar uma carga com diferentes e variadas falhas, e caracterizar melhor quais foram os erros, classificando-os e avaliando-os para identificar o comportamento ocorrido no sistema além de apenas não finalizar a operação. Outro ponto importante para ser trabalhado é sobre a consistência dos dados que pode ser avaliada para identificar possíveis falsos positivos, onde não apresentou erro na operação mas pode haver inconsistência no resultado obtido, característica não avaliada neste trabalho.

Referências

Abramova, V. and Bernardino, J. (2013). NoSQL databases: MongoDB vs Cassandra. In *Proc. of the C3S2E - Int Conf on Computer Science and Software Engineering*, pages 14–22, New York, New York, USA. ACM Press.

Abramova, V., Bernardino, J., and Furtado, P. (2014). Experimental Evaluation of NoSQL Databases. *International Journal of Datase Management Systems (IJDMS)*, 6(3):1–16.

Chodorow, K. (2013). *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2 edition.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proc. of the SoCC - ACM Symposium on Cloud Computing*, pages 143–154. ACM.

Dey, A., Fekete, A., Nambiar, R., and Rohm, U. (2016). YCSB+T : Benchmarking Web-Scale Transactional Databases. In *Proc. of the ICDEW - Intl. Conf. on Data Engineering Workshops*, pages 485–489.

Hewitt, E. (2010). *Cassandra: The Definitive Guide*. O'Reilly Media, Inc., 1 edition.

Lakshman, A. and Malik, P. (2010). Cassandra - a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44.

Marynowski, J. E., Santin, A. O., and Pimentel, A. R. (2015). Method for testing the fault tolerance of MapReduce frameworks. *Computer Networks*, 86:1–13.

Moniruzzaman, A. B. M. and Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison.

Pankowski, T. (2015). Consistency and availability of Data in replicated NoSQL databases. In *Proc. of the ENASE - IEEE Int. Conf. on Evaluation of Novel Approaches to Software Engineering*, pages 102–109.

Tsuyuzaki, K. and Onizuka, M. (2012). NoSQL database characteristics and benchmark system. *NTT Technical Review*, 10(12).

Tudorica, B. G. and Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. In *Proc. of the RoEduNet - Int Conf on Networking in Education and Research*, pages 1–5. IEEE.