Alta Disponibilidade Utilizando MongoDB

Lucas Barbosa, Felipe Oliveira, Gustavo Bruschi

Curso de Tecnologia em Banco de Dados – Faculdade de Tecnologia de Bauru (FATEC)

Rua Manoel Bento Cruz, nº30, Quadra 3 – Centro – 17.015-171, Bauru, SP - Brasil

{lucas.barbosa31, felipe.oliveira156, gustavo.bruschi}@fatec.sp.gov.br

Abstract. *This article aims to present a cost-effective alternative to a high performance and fault tolerant information storage system using a high availability cluster. With the document-oriented MongoDB database, due to its feature variations, such as its data replication feature known as (replica set), as well as the sharding used to split collection data between multiple servers and yet being open-source. With the focus on small and medium enterprises, it aims to offer a clustered system configured in a way that becomes practically imperceptible to the end user any problem occurred if it occurs. Using software MongoDB and hardware not suitable for such an activity, it is concluded that, with a low investment, it is possible to use such a system efficiently*

Resumo. *Este artigo visa apresentar uma alternativa de custo reduzido de um sistema de armazenamento de informações e ao mesmo tempo, de alto desempenho e tolerante a falhas, utilizando um cluster de alta disponibilidade. Com o banco de dados MongoDB orientado a documento, devido suas variações de recursos, como o seu recurso de replicação de dados conhecido como (replica set), e também o sharding utilizado para dividir os dados de collections entre vários servidores e ainda sendo open-source. Com o foco em pequenas e médias empresas, visa oferecer um sistema em cluster configurado de uma maneira a se tornar praticamente imperceptível ao usuário final algum problema ocorrido, caso o mesmo venha a ocorrer. Utilizando softwares MongoDB e hardwares não próprios para tal atividade, conclui-se que, com um baixo investimento, é possível utilizar tal sistema de uma maneira eficiente.*

1. Introdução

Atualmente, algumas atividades que realizamos como por exemplo (compra online e comentários em redes sociais) podem ser aproveitadas pois, geram informações, que, de alguma maneira, utilizadas pelas empresas para coleta de informações relacionados à pesquisa de mercado, desenvolvimento de novos sistemas, novos aplicativos e preferência de compra dos consumidores. Para conseguir coletar e utilizar essas informações de um modo viável e que gere confiança para tomada de decisões dentro das organizações, é necessário um sistema de armazenamento confiável e de boa performance, afim de ter uma maior competitividade no mercado atual.

A busca por um ambiente seguro, rentável e que consiga gerenciar um grande volume de dados é o que toda empresa almeja. Podemos acrescentar a esse ambiente um

mecanismo que tenha tolerância a falhas mecânicas que, para empresas que não podem parar suas atividades é uma forma de continuidade do serviço e lucratividade em relação as suas concorrentes que não dispõem de tal recurso.

Em meio a todas as opções que surgem no mercado, temos a opção de utilizar a opção de computação em cluster que, nada mais é do que um aglomerado de computadores trabalhando interligados, transmitindo a sensação de ser apenas um único computador. Ela permite uma maior escalabilidade do sistema já existente na empresa, possibilitando unificar dois ou mais computadores, a fim de somar seus poderes de processamento e melhorar a resposta da requisição para o usuário final.

Para armazenar e manipular as informações existentes nesse sistema, é necessário ter um banco de dados, mas essa escolha pode não ser tão simples como aparenta. Para essas atividades, os bancos relacionais vêm dividindo cada vez mais o cenário empresarial e quem vem ganhando parte desse espaço são os bancos de dados não relacionais ou simplesmente bancos NoSQL que na verdade se trata de um banco de dados schemaless ou simplesmente, livre da estrutura existente nos bancos relacionais.

Unificando um banco NoSQL a um sistema de computador em cluster com tolerante a falhas, é possível criar uma excelente alternativa de baixo custo para as empresas que não dispõem de tantos recursos para serem investidos, mas que precisam de uma solução capaz de suprir a sua necessidade para que não percam competitividade no mercado atual.

O objetivo deste trabalho é apresentar uma solução de baixo custo de um cluster de banco de dados tolerante a falhas utilizando o MongoDB, um banco NoSQL de destaque e de grande versatilidade , uma vez que nele é possível ser feito desde os mais simples processamentos de dados até os mais complexos, envolvendo uma imensa massa de dados, como no Big Data, por exemplo. O MongoDB é um banco de dados totalmente orientado a documentação livres (conjunto de campos de diferentes tipagens, chamados de coleção ou collections, não tem como restrições possuir tabelas e colunas podendo existir valores simples, como números, strings, e datas também listas de valores e listas de objetos), criado em 2007 pela 10gen (atual MongoDB inc.) mas, sua primeira versão foi lançada mesmo apenas em 2009. No formato semelhante ao JSON (Java Script Object Notation), utilizado no protocolo HTTP muito conhecido de BSON, também é de código aberto e multiplataforma. Está forma de banco orientada a documento tem por finalidade manter todas as informações em um mesmo (único) documento e ainda ser livre de esquemas, identificadores únicos universais (UUID), consultas através de métodos avançados de agrupamento e filtragem..

Na seção seguinte estão descritos conceitos importantes para o melhor entendimento sobre cluster, alta disponibilidade, MongoDB com seus recursos de alta disponibilidade e árbitro. Também estarão descritos cada etapa do experimento, bem como os métodos utilizados e os equipamentos empregados e adaptações feitas nas seções 2,3,4,5 e 6. As seções 7 e 8 contém os resultados obtidos e a conclusão respectivamente.

2. Banco de Dados Não Relacionais

Para Seguin (2013), nos dias atuais, há uma crescente lista de novas tecnologias e técnicas que estão sendo criadas e rapidamente difundidas nas empresas. Embora essas novidades estejam vindo rapidamente, elas estão sendo substituídas em um ritmo muito mais rápido. A tecnologia inerente à banco de dados não escapou desse avanço e uma das novidades foi o surgimento dos bancos NoSQL.

Quando se ouve o termo NoSQL, a primeira coisa que pensamos, segundo Sadalage e Fowler (2012), é de 'Não' ao SQL. O termo No SQL surgiu no fim dos anos 90 devido a necessidade de um novo modelo de armazenamento de dados, onde surgiria novos modelos de aplicações e, segundo Paniz (2016), o termo que se referia a esse novo modelo era interpretada erroneamente, como se fosse contra ao modelo relacional, sendo chamada de 'Não ao Relacional'.

Ele ainda afirma que não era essa a intenção inicial, então houve uma adaptação na sigla, a fim de desfazer o mal-entendido, pois o acrônimo faz alusão à 'Não apenas SQL' (Not Only SQL em inglês). Para Tiwari (2011), no surgimento do acrônimo, o criador provavelmente queria dizer não relacional, mas com o passar do tempo, foi necessária uma sigla que fizesse referência ao verdadeiro significado do novo modelo, surgindo o acrônimo NonRel, fazendo alusão ao não relacional, mas não obteve muito a aceitação dos usuários desse modelo.

Para Sadalage e Fowler (2012) o NoSQL vem do fato de que o banco de dados desse grupo não irá utilizar o SQL como linguagem padrão, mas sim Shell Scripts. Ainda segundo os autores, o que realmente importa não é o acrônimo correto, mas sim, o real significado e a sua aplicação, se referindo a um banco de dados de código aberto e com dados mal definidos e o 'Não apenas' SQL encaixa-se bem na descrição do parque tecnológico de uma empresa e indo além, faz alusão mais do que apenas à tecnologia, mas sim ao futuro do banco de dados, com uma nova maneira de se pensar na estruturação e armazenamento de dados, mas eles também acreditam que essa interpretação de 'Não apenas SQL' pode vir a ter alguns problemas, pois a maioria escreve NoSQL, enquanto o correto para essa interpretação seria NOSQL.

Segundo Boaglio (2015), hoje a aplicação segue as regras criadas através do banco de dados relacionais, mas nos bancos não relacionais, a ideia é de o banco siga as regras da aplicação, sendo assim, organizando os dados de acordo com a necessidade do negócio ou ainda, segundo Paniz (2016), poderíamos utilizar o banco não relacional juntamente com um banco relacional mas utilizando eles em diferentes momentos e com diferentes funcionalidades e Seguin (2013) complementa dizendo que o NoSQL surgiu como uma solução e não com o intuito de substituir o banco de dados relacional, pois essa tecnologia aborda algumas necessidades específicas, reforçando o que Tiwari (2011) já idealizava quando escreveu que esse modelo alivia os problemas que os bancos relacionais impõem em determinadas situações, além de facilitar o trabalho com uma grande massa de dados. Seguin (2013) ainda sugere que além da utilização do banco de dados relacional paralelamente com o banco de dados não relacional, poderíamos utilizar um software de processamento intensivo de dados, extraindo o máximo de cada ferramenta à serviço da empresa.

Os bancos NoSQL trabalham sem uma estrutura pré-definida, permitindo a adição de campos sem a necessidade de alteração na sua estrutura. Para Fowler e Sadalage (2012), as principais vantagens do NoSQL além deles serem programas de

código aberto, a facilidade de manipular uma grande massa de dados que exigem um grande desempenho e melhoram a produtividade, pois utilizam uma interação de dados com um estilo mais prático.

Paniz (2016) diz que existem diversos tipos de bancos não relacionais citados no quadro 1, entretanto, cada tipo possui peculiaridades, vantagens, desvantagens e diversos modos de implementação, implicando assim em escalabilidade e performance diferentes. Ele ainda cita os principais tipos, apresentados na figura 2 a seguir.

Portanto, independentemente do modo que o autor possa descrever o que significa o acrônimo NoSQL ou até mesmo os tipos de bancos não relacionais mais conhecidos dos tempos atuais, em suma, eles descrevem que o modelo veio para complementar o modelo relacional existente hoje, sendo utilizado, muitas vezes, juntamente com os bancos relacionais, sendo os dois de grande importância para a organização, mas, cada um sendo utilizado para uma determinada funcionalidade dentro do sistema de uma empresa, podendo atuar como apoio um ao outro. Nesse artigo abordaremos sobre o MongoDB, um banco documentar, e algumas de suas principais características.

Segundo Augusto E. (2017), MongoDB é um banco de dados noSQL orientado a objetos chamados de documentos, open-source muito utilizados em uma quantidade muito grande de dados, como exemplo o BIG DATA. O Mongo utiliza de collections (coleções) que é o agrupamento de vários documentos, diferente dos modelos relacionais como podemos ver na figura 1 abaixo, onde os dados seriam as linhas de inserção.

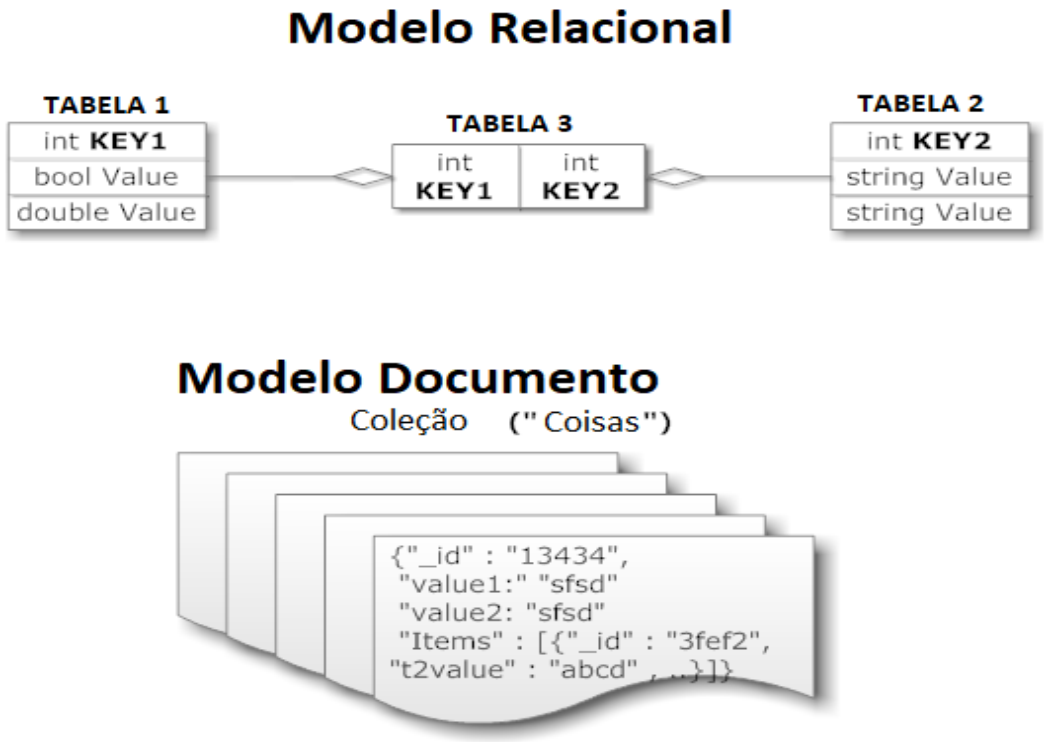

Figura 1. Comparação modelo relacional e modelo documento

Fonte: Erick Augusto (2017)

Quadro 1. Tipos de Banco de Dados NoSQL

Chave-Valor	Todos os registros são partes de um único objeto, porém, cada valor possui uma chave diferente, facilitando e ganhando agilidade nas buscas
Colunar	Aonde todos os registros compõem uma tabela e, nessa tabela, podemos conter várias outras tabelas com diferentes propriedades/atributos, facilitando assim a recuperação dos dados
Documento	Conjunto de campos de diferentes tipagens e esse conjunto, chamamos de coleção e, as coleções são schemaless, podendo atualizar a estrutura existente de acordo com a necessidade momentânea
Grafo	Este contém três estruturas básicas, os grafos representando os dados, as arestas representando as ligações e as propriedades representando os atributos dos campos

Fonte: Paniz, 2015

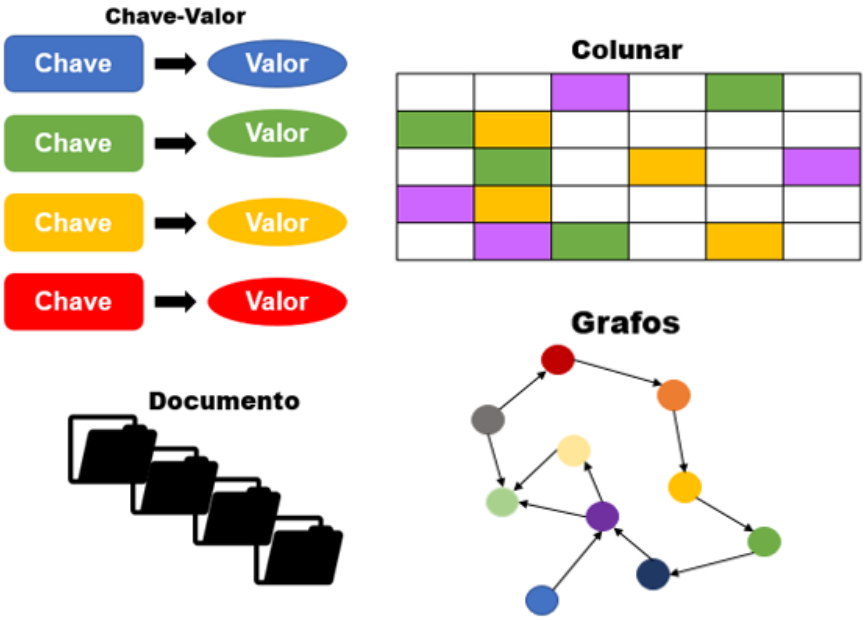

Figura 2. Tipos de banco NoSQL

Fonte: Elaborado pelos autores, 2018

3. Alta disponibilidade

Manter os serviços de um sistema operando, mesmo que o sistema venha ser modificado internamente por algum motivo de falha, ter aplicações em pleno funcionamento com os dados replicados em diferentes localidades (nós) mesmo com a parada de um, outro entra no lugar, isto é uma arquitetura de alta disponibilidade, segundo Boaglio (2015).

Alta disponibilidade pode se dizer em manter um sistema disponível ininterrupto pelo máximo possível de tempo. Para Cruz (2015), o termo Alta Disponibilidade "High Availability" (HA), seria a capacidade de um sistema permanecer disponível o maior tempo possível durante sua execução, garantindo suas funcionalidades e ainda alta tolerância a falha de hardware, software e energia. Normalmente utiliza-se peças(hardware) que geram redundância no sistema, hardware que inicia seu funcionamento automático no momento que é detectado uma falha, temos como exemplo Failover onde, é assumido o controle da operação previamente executada em caso de falha.

Outro método para se obter alta disponibilidade de um sistema e a utilização de cluster, distribuindo as atividades entre eles a fim de ganhar desempenho além de disponibilidade. Funciona como uma máquina (nó), realizando o espelhamento de outra, no caso de falha de uma a outra máquina assume o lugar. Segundo Rezende (2013) cluster é a ligação de dois ou mais computadores independentes também conhecidos de "nós" conectados através de uma rede com o intuito de aumentar o desempenho e disponibilidade durante a execução de tarefas.

Na figura 3 podemos visualizar um modelo de cluster com três nós, onde o processo é executado no primeiro nó e replicado para os nós segundarios, onde são utilizandos apenas para consultas. Em tempo em tempo os nós se conversam analisando se estão ativos, operação conhecida de *Heartbeat,* caso o nó primario cair um dos nós segundarios assume seu lugar.

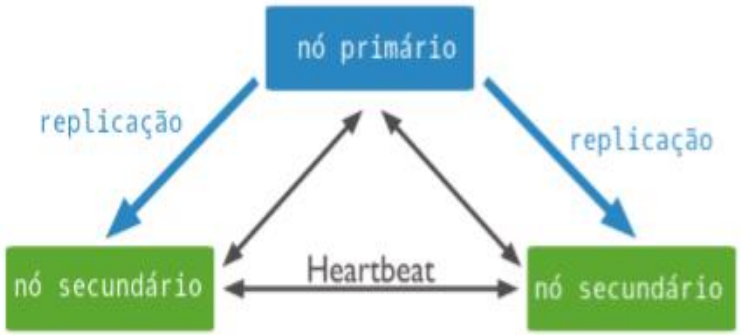

Figura 3. Cluster com três nós

Fonte: Boaglio, 2015

Para Cambiuccci (2011) Alta Disponibilidade deve ser classificada de acordo com o cenário e nível de serviço que estamos considerando. Conforme abaixo na figura 4, podemos verificar uma arquitetura típica de um sistema de alta disponibilidade para dois nós em cluster, com base de dados compartilhada onde, a mais de um caminho e com duplicidade de hardware assim, impedindo a indisponibilidade no caso de um único ponto Single Point of Failure (SPOF).

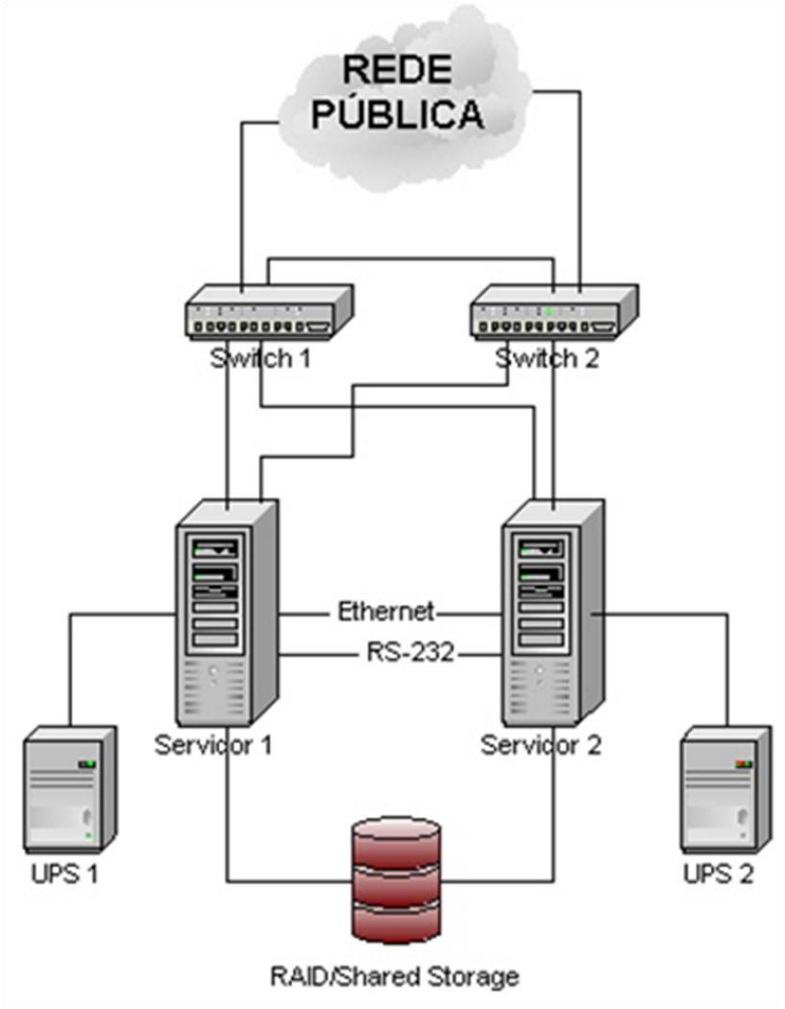

Figura 4. Exemplo de um sistema HA

Fonte: Waldemir, 2011

Segundo Capucho (2016) a disponibilidade do serviço operacional é medida com base na probabilidade do momento em que o sistema esteja em funcionamento, sua porcentagem está ligada diretamente ao termo Uptime (tempo de disponibilidade) do serviço. Na tabela 1 abaixo vemos que para Pereira Filho (2004), a alta disponibilidade pode ser descrita pela incidência de downtime (tempo de indisponibilidade) do sistema.

Tabela 1. Níveis de Disponibilidade

	UPTIME	DOWNTIME	DOWNTIME POR ANO	DOWNTIME POR SEMANA
1	**90%**	**10%**	**36,5 dias**	**16 horas, 51 minutos**
2	**98%**	**2%**	**7,3 dias**	**3 horas, 22 minutos**
3	**99%**	**1%**	**3,65 dias**	**1 hora, 41 minutos**
4	**99,8%**	**0,2%**	**17 horas, 30 minutos**	**20 minutos, 10 segundos**
5	**99,9%**	**0,1%**	**8 horas, 45 minutos**	**10 minutos, 5 segundos**
6	**99,99%**	**0,01%**	**52,5 minutos**	**1 minuto**
7	**99,999%**	**0,001%**	**5,25 minutos**	**6 segundos**
8	**99,9999%**	**0,0001%**	**31,5 segundos**	**0,6 segundos**
9	**99,99999%**	**0,00001%**	**3,15 segundos**	**0,06 segundos**

Fonte: Pereira Filho, 2004

4. MongoDB e Recursos de Alta Disponibilidade e Balanceamento de Carga

Para Boaglio (2015), alta disponibilidade significa ter o mesmo conjunto de dados em diferentes servidores, sendo chamados de nós e estando dentro de uma estrutura denominada replicaset e um replicaset pode ser definido como um conjunto de servidores de replicação de dados, podendo ter entre 2 a 12 servidores, com o mínimo aceitável de 3 servidores. Os nós do replicaset fazem uma constante verificação entre eles para saberem o status de todos os membros do conjunto e essa verificação é denominada heartbeat. Chodorow (2013) nos diz que, heartbeat é um batimento cardíaco com uma curta mensagem e sua principal função é informar ao nó primário se ele consegue se comunicar com todos ou com a maioria dos nós secundários. Ela ainda nos fala sobre alguns dos estados mais comuns que encontramos no MongoDB, além do primário e secundário, estados esses que se encontram no quadro 2.

Seguin (2013) nos fala que, todas as operações feitas no nó primário devem ser replicadas para os nós secundários, a fim de serem cópias exatas do primário. Ele ainda nos diz que o principal objetivo da replicação é fornecer um estado de alta disponibilidade das informações. Chodorow (2013), acrescenta que é gerado um log dessas operações gravadas no nó primário, aonde os nós secundários acessam esses logs com o intuito de atualizar a sua coleção e, caso a sincronização do nó secundário falhe, ele encerrará o processo sem que haja nenhuma alteração. Ela ainda diz que os nós secundários também geram logs, possibilitando serem utilizados para a sincronização de outro nó. O log sempre leva em consideração a operação em execução, sendo gerado uma linha de log para cada operação efetuada, como por exemplo, se excluirmos 1.000.000 documentos em nossa coleção, gerando 1.000.000 de documentos de log, ocasionando uma sobrecarga da coleção.

Quadro 2. Tipos de estados em um replicaset

Startup	Quando se inicia um membro pela primeira vez. Este estado permanece até que as configurações do replicaset sejam carregadas.
Startup2	Estado que perdura durante todo o processo de sincronização inicial, normalmente leva-se segundos para ocorrer essa sincronização.
Recovering	O membro está em operação, mas ainda não disponível para leitura; Necessário avaliar se o membro está em um estado válido ou será necessário uma sincronização para estar; Quando um membro ficou para trás na sincronização. Em muitos casos como este, pode ser necessário uma ressincronização.
Arbiter	Com este status, não há sincronização de dados no membro, ele apenas participa do heartbeat e, caso necessário, das eleições.
Down	Quando um membro ativo passa a ser inacessível.
Unknown	O membro ganha este estado quando ele nunca conseguiu comunicar-se com nenhum outro.
Removed	O membro foi removido do conjunto. Caso este retorne, fará a transição entre os estados até alcançar seu estado inicial.
Rollback	O nó primário é substituído, mas continha arquivos que não foram replicados para os outros nós. Neste caso, o ex primário começa o processo de rollback desses arquivos.
Fatal	Quando ocorre um erro irrecuperável.

Fonte: Chodorow, 2013

Outro recurso importante é o sharding que, Seguin (2013) nos diz que é uma abordagem de escalabilidade que particiona os dados em vários servidores integrantes de um cluster, sendo o principal método de obter escalabilidade no MongoDB e, nos diz também que, combinar replicação com sharding é o ideal para se obter alta disponibilidade com alto nível de escalabilidade, já para Boaglio (2015), sharding é você dividir sua coleção total em coleções menores e coloca-las em servidores fisicamente distintos. Ele ainda nos diz que, para o bom funcionamento do sharding, é essencial que esses três serviços da quadro 3 estejam rodando nos servidores e para um bom funcionamento de um ambiente de produção, recomenda que tenha no mínimo dois serviços de shards, três serviços de config Servers e dois serviços de query routing instances.

Quadro 3. Serviços essenciais para o sharding

Shards (dados)	Instância do MongoDB que contém os dados particionados
Config Servers (metadados)	Servidores contendo os metadados mapeados da arquitetura
Query Routing Instances (mongos)	Instância que conversará com a aplicação, direcionando leituras e escritas para os shards

Fonte: Boaglio, 2015

Para Chodorow (2013), objetivo do sharding é fazer todos os computadores relacionados transpareçam ser apenas um perante a aplicação e, para isso, é executado o processo mongos, contendo o roteamento dos dados particionados. Normalmente a aplicação do sharding é uma tentativa de reduzir o uso de memória ram, aumentar o espaço em disco disponível, reduzir a carga em um servidor ou ainda para aumentar a taxa de transferência de dados suportada por um único servidor mas, a decisão do momento correto para utilizar o sharding é essencial, pois se for tomada muito cedo, pode vir a acrescentar uma complexidade desnecessária para a operação, com difíceis decisões relacionadas aos padrões de distribuição da coleção, mas caso seja tomada muito tarde, a dificuldade passa a ser fragmentar uma coleção sobrecarregada aonde a mesma não pode parar, acrescenta ainda a autora.

5. Árbitro

É uma instancia (servidor) que não possui dados (não tem cópia do conjunto de dados), ou seja, tem como função votar em um servidor secundário para se tornar o nó primário, quando o nó primário cai. Mas, para isso existe um conceito de votação onde, quando o servidor primário cai é realizado a votação com o restante dos servidores é aí que entra o árbitro. Caso ocorra um empate na votação o árbitro realiza o desempate assim, definindo o nó primário. Ainda se ficar somente um servidor é realizado a votação com o arbitro votando no nó que ficou assim obtendo dois votos, sendo um voto do próprio servidor (os servidores votam neles mesmos) e um voto do arbitro.

Para Cabrezas (2015), árbitro é o servidor que não possui dados, é sempre muito utilizado quando o nó primário é descartado (para de funcionar), pois há partir deste momento a uma votação na escolha de qual no secundário irá assumir o lugar de nó primário onde, o árbitro define qual será o nó secundário que irá assumir este lugar, sendo que ele mesmo não pode ser primário ou secundário.

O árbitro é muito utilizado quando a um número par de nós/membros em um replica set, pois havendo eleição na escolha do melhor nó, ou seja, um novo nó primário não tem o risco de haver empate. A eleição ocorre quando cai o nó primário, identificado pelo Hearbeat que é o responsável por realizar a sincronização entre os nós em uma determinada frequência, onde os nós secundários identificam se o oplog do nó

primário teve alguma alteração de estado, caso tenha alguma alteração é realizado a cópia de forma assíncrona para o oplog dos nós secundários, o hearbeat é quem informa todos os nós que o nó primário ficou inacessível portanto, inicia uma eleição na escolha do melhor nó para ser o nó primário. Para não ter empates na votação e obter resultados inconsistentes, é sempre necessário que se tenha um número ímpar de nós, caso isso não ocorra entra a necessidade de se ter um nó árbitro. Para Kobellarz, J. (2015), árbitro é um nó diferente dos demais nós, é um nó mais enxuto com relação aos demais nós secundários padrão, eles não armazenam dados possui um oplog (fila de operação de escrita em formato de log), serve apenas para votar.

Segundo Monteiro, D. (2017), árbitro nada mais é do que simplesmente um servidor que tem como única função votar em um servidor secundário para ser o servidor primário, já que ele mesmo não possui dados.

O árbitro possibilita que conjunto de réplicas tenham um número ímpar de votantes na eleição do nó primário, sendo que ele mesmo não replica dados, ou seja, o árbitro não tem uma cópia do conjunto de dados, também não pode se tornar um nó primário, segundo MongoDB (2019).

Na figura 5 abaixo é possível verificar um modelo de arquitetura de servidor de um MongoDB, onde, temos um servidor primário, três secundário e o árbitro. Neste modelo de arquitetura imagine que caia o servidor Primário e mais 2(dois) Secundários, isto acarretaria em um problema de disponibilidade pois, o MongoDB não poderia atribuir ao servidor que restou o papel de primário, já que existe um conceito de votação onde é necessário que mais da metade dos servidores escolhem qual deles assumira a função de primário, neste caso restou apenas 1(um) servidor que votaria nele mesmo, mas não é o suficiente. Neste caso é que entra o Árbitro pois, tem como função votar no servidor que deve assumir a função de servidor Primário.

Figura 5. Integrantes de um cluster

Fonte: https://docs.mongodb.com/manual/core/replica-set-arbiter

6. Materiais e Métodos

Para a elaboração dos experimentos, foi utilizado um computador hospedeiro com o sistema operacional Windows 8.1, processador Intel I3, 8 GB de memória ram e HD de

aproximadamente 500 GB. Foram criadas 3 máquinas virtuais no software VirtualBox (6.0.2 R128162). No quadro 4 abaixo mostra a composição e algumas características das máquinas virtuais.

Quadro 4. Composição das máquinas virtuais

Denominação dos nós	Função	IP	Porta	Versão do Windows	Memória Ram	HD	Núcleo do Processador
Nó1/Teste7-1	Controlador do cluster	192.168.10.173	27020	7 Ultimate 64bits	1 GB	30 GB	1 núcleo
Nó2/Teste7-2	Réplica do controlador	192.168.10.130	27021	7 Ultimate 64 bits	1 GB	30 GB	1 núcleo
Nó3/Teste7-3	Réplica do controlador	192.169.10.207	27022	7 Ultimate 64 bits	1 GB	30 GB	1 núcleo

Fonte: Autores, 2019

Foi instalado nas máquinas o MongoDB (3.6.10 2008R2Plus SSL), e criada as pastas: (1)data, (2)db, (3)Rs, sendo as duas primeiras com o objetivo de iniciar a instância do MongoDB uma vez que elas não são criadas na instalação e a última criada para o armazenamento dos dados do cluster . Dentro da última pasta, foi criada mais três pastas, (server1, server2 e server3) contendo nelas as pastas conf, db e logs. Dentro da pasta db, foi criado os arquivos host e conf, contendo os ip's das 3 máquinas e o bind 0.0.0.0 respectivamente. Este procedimento serve para criar o armazenamento dos 3 nós, indicar quais os ip's que terão os acessos permitidos e a faixa de ip.

Para ser iniciada a instância dos nós, foi utilizado no prompt de comando o comando *mongod --bind_ip_all --dbpath C:\Rs\data\server1\db --replSet tg1 --port 27020* em cada nó, modificando o número da porta, proporcionando que o servidor em cada nó esteja ativo e pronto para "ouvir" comandos. Com as instâncias funcionando foi aberta uma nova janela do prompt de comando e executado o comando *mongo --port 27020* em cada nó, modificando apenas a porta para a porta em que a instância foi iniciada. Com as instâncias prontas e com o utilitário conectado, foi necessário escolher um dos nós como primário, sendo escolhido o nó1, facilitando o entendimento pois foi associado o nó1 com primeiro e nele foi executado o comando para indicar ao conjunto que aquele nó será o primário, o *rs.initiate()* e ainda nele, foi executado o comando *rs.status()* a fim de verificar se realmente foi criado o nó primário.

Para indicar ao conjunto quais computadores são integrantes do conjunto, foi utilizado o comando *rs.add('192.168.10.130:27021')* e *rs.add('192.168.10.207:27022)* no nó1 para adicionar os nós 2 e 3 respectivamente, indicando entre parênteses primeiro o ip fixo e a porta de comunicação da instância já iniciada anteriormente. Para verificar se as adições ao conjunto foram feitas com sucesso, foi usado o *rs.status()*, aonde foi verificado que as adições haviam ocorrido com sucesso.

Com os integrantes adicionados, ainda foi preciso configurar que o conjunto já estava pronto e apto para a inserção e replicação das informações, então foi executado nos nós 2 e 3 o comando *rs.slaveOk()*, terminando assim a configuração de replicação mas ainda é necessário configurar o conjunto para que, caso o primário caia, algum dos outros dois integrantes consiga assumir seu lugar e deixar disponível novamente a informação no menor tempo possível e, para isso, foi necessário acrescentar um árbitro em todos os nós, visto que ele apenas conta para a escolha do novo primário.

Quadro 5. Comandos utilizados no ambiente

Comando	Objetivo
rs.initiate()	Utilizado para indicar qual será o primeiro nó e o controlador do cluster.
rs.status()	Mostra os integrantes do cluster, suas configurações e status de cada membro.
rs.add()	Utilizado com a porta da instância e o ip do nó dentro dos parênteses e adicioná-lo como membro integrante do cluster.
rs.slaveOk()	Utilizado para indicar ao cluster de que o membro está pronto para receber os dados do principal.
rs.addArb()	Utilizado com a porta da instância e o ip do nó árbitro dentro dos parênteses e adicioná-lo como membro integrante do cluster.

Fonte: Autores, 2019

Em todos os nós foi necessário iniciar a instância do árbitro com o comando mongod *--bind_ip_all --dbpath C:\data\arb --replSet tg1 --port 27024* sendo utilizado a porta 27024 para o nó1, 27025 para o nó2 e 27026 para o nó3 e, além da instância, foi preciso acrescentá-los ao conjunto já existente com o comando *rs.addArb('192.168.10.207:27024')* para o nó1 e para acrescentar os demais árbitros, foi alterado o ip e a porta para o correspondente ao nó e instância do árbitro iniciada. Ao final, para conferir todas essas configurações, foi utilizado o *rs.status()* e o resultado visto foi de 1 nó primário, 2 nós secundários e 3 nós árbitros.

Para a realização dos testes, foi criado um script para inserir 10.000 documentos contendo informações aleatórias conforme o quadro 6 abaixo e foi executado no nó1, o primário. Após o término da sua execução, foi verificado se os nós 2 e 3 (secundários) haviam recebido essas informações e por causa da quantidade de documentos, houve uma espera de 1 minuto até a conclusão da replicação entre todos os integrantes do cluster.

Quadro 6. Script de criação dos documentos

```
for
(var i = 0; i < 10000; i++)
```

```
{db.teste.insert({"foo": "bar","baz" :i,"z":10 - i})}
```

Fonte: Autores, 2019

7. Resultados

Para simular o acesso ao banco de dados de uma aplicação, foi instalado na máquina hospedeira um dos clientes do MongoDB, o mongochef (um utilitário utilizado no computador cliente para acessar o banco de dados e realizar o procedimento necessário utilizando a linguagem nativa do MongoDB ou o próprio SQL através de um conversor incluso nele). Nele, foi necessário realizar uma configuração com os integrantes do conjunto de replicaset criado nas máquinas virtuais. Na figura 6 está a configuração utilizada para a conexão, possibilitando o mongochef localizar todos os nós do conjunto e o campo *Primary preferred* faz referência a qual nó o mongochef irá procurar acessar primeiramente.

Após a conexão com o nó primário, foi executado no mongochef um script para a inserção de 20.000 documentos, com a finalidade de criar uma quantidade de dados relevante para a realização dos testes. O script utilizado para realizar a inserção no conjunto de replicaset é demonstrado no quadro 7 e sua execução foi feita diretamente no mongochef conectado ao nó primário. Após a execução, foi executado no mongochef o comando *db.tg1.count()*, que retornou à quantidade de documentos inseridos na base, retornando 20.000. Para confirmar se a quantidade inserida no nó primário foi inserida nos nós 2 e 3, foi acessado o utilitário do MongoDB nas máquinas e foi utilizado o mesmo comando e a quantidade retornada foram os mesmos 20.000 documentos inseridos no nó primário, ficando comprovado assim que a replicação dos documentos do primário para os secundários foi concluída com sucesso.

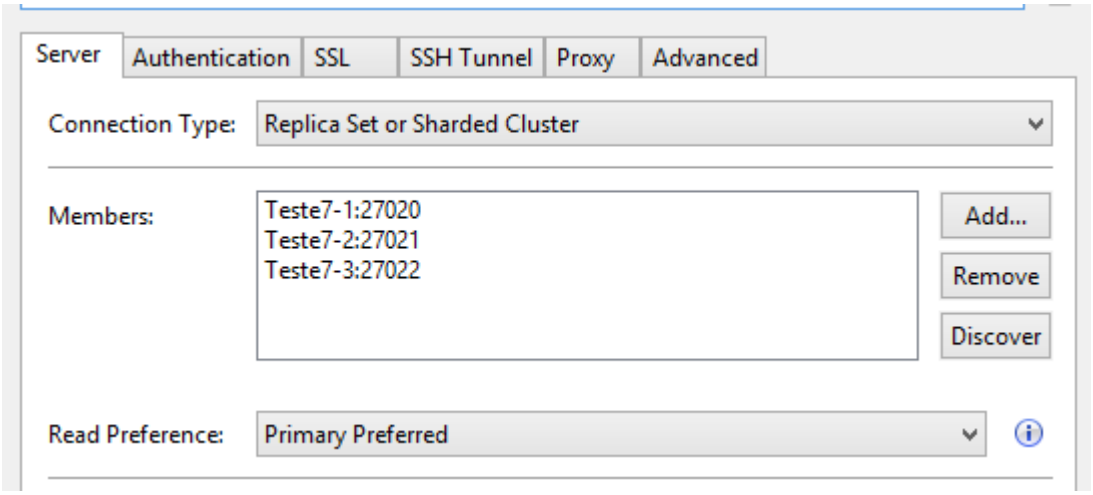

Figura 6. Tela de configuração mongochef

Fonte: Autores, 2019

Quadro 7. Script de criação dos documentos

```
For
(var i = 0; i < 20000; i++)
{db.tg1.insert({"foo": "bar","baz" :i,"z":10 - i})}
```

Fonte: Autores, 2019

Para a realização do experimento da alta disponibilidade, foi simulado um problema com a rede do nó principal, o que ocasionaria a perda de comunicação dele com os demais nós integrantes. Para essa simulação acontecer no presente experimento, foi desligada a máquina virtual contendo o nó1 e um dos árbitros, conseguindo assim torná-los incomunicáveis com o restante dos integrantes.

A partir desse momento, entra em ação os nós árbitros que foram adicionados ao conjunto de nós, realizando uma votação interna entre eles e os nós restantes, ressaltando mais uma vez que nesse tipo de situação os nós ativos votam em si mesmo, por isso a importância do árbitro e essa situação ocorre em poucos segundos, e o nó com o maior total (o nó inativo neste caso também vota nele mesmo) de votos passa a ser o novo primário.

Para verificar qual dos nós assumiria o posto de principal, foi realizado um teste nos utilitários de ambos os nós ativos. Neles, foram utilizados o comando *rs.status()*, o qual tem como principal objetivo trazer as informações dos integrantes do conjunto e, neste caso, esse comando também irá trazer atualizado a informação se este nó ainda continua sendo o secundário ou se ele passou a ser o nó primário entre os integrantes. Após a execução desse comando em ambos os nós, foi verificado que o novo nó primário passou a ser o nó3.

Em meio ao processo de votação, na máquina hospedeira, na conexão do mongochef com o nó primário, ainda é possível consultar informações contidas no banco e, só foi acontecer algum problema quando foi realizado um insert, o qual resultou uma mensagem de erro dizendo não ser possível localizar o nó1. Para que seja possível realizar novas inserções de informações, foi fechada a conexão existente e foi aberta uma outra, na qual foi possível observar na aba da conexão que o mongochef estava conectado ao nó3. Com isso, foi realizada uma operação de insert com informações aleatórias apenas com o objetivo de validar a autonomia do nó3 sobre o conjunto do replicaset e, ao ser executado o script, verificamos que a informação foi inserida com sucesso.

Após a operação de insert ser executada, foi necessário consultar no nó2, o único secundário no momento, se esta informação foi replicada a ele e para isso, no utilitário deste nó foi executado o comando *db.tg1.find().sort({"_id":-1}).limit(1)* para nos trazer a informação ali armazenada e ser possível realizar a comparação dela com a que inserimos pelo mongochef, já que neste caso ele trará apenas o último documento inserido na base de dados.

Para finalizar os experimentos, ainda é necessário saber se o nó inativo, ao voltar a sua atividade e conseguirá receber a informação inserida nos demais integrantes enquanto ele estava inativo. Foi executado a inserção de mais 20.000 documentos no nó primário e a máquina virtual contendo o nó1 foi iniciada, foi preciso iniciar a instância do MongoDB, o árbitro e seu utilitário com os comandos já mencionados no capítulo anterior Materiais e Métodos. Após todo o processo de inicialização, foi deixado ele

sem nenhuma atividade por aproximadamente 1 minuto e, passado esse tempo, foi executado alguns comandos para verificar se o nó agora ativo é o nó primário/secundário e para saber se a informação inserida enquanto ele estava desligado também foi replicada. Foram executados os comandos *rs.status()* e o *db.tg1.find* respectivamente e seus retornos foram de status secundário entre os integrantes do cluster e a pesquisa, após o tempo de 00:38:36, retornou que as informações inseridas enquanto ele estava desligado, isso porque quando ele foi iniciado novamente, no tempo em que ficou parado ele fez uma busca nos seus log's com os log's do nó primário afim de encontrar alguma diferença e, como neste caso ele encontrou, se utilizou do próprio log do primário para poder atualizar seus documentos e ficar com as mesmas informações dos demais integrantes.

8. Conclusão

Após a finalização do presente artigo, foi possível identificar a possibilidade ter uma estrutura de alta disponibilidade, tolerante a falhas em um ambiente de cluster para empresas de pequeno a grande porte utilizando recursos Open Source, onde foi utilizado hardwares impróprios para ambientes de cluster, foi possível adapta-los e criar um ambiente clusterizado e propício para a alta disponibilidade ser praticada.

Foi verificada que o ambiente tornou-se muito funcional no quesito replicação de dados, uma vez que devidamente configurado, foi possível duplicar as informações inseridas facilmente e mostrou-se muito eficiente quando foi necessário substituir o nó primário em caso de falhas, garantindo assim um ambiente confiável para que as empresas utilizadoras destes consigam uma grande vantagem computacional, uma vez que não precisa haver gasto com computadores específicos para esse tipo de ambiente e ainda é possível reaproveitar os computadores em boas condições para a montagem do ambiente como foi possível ver nas pesquisas deste trabalho.

Este ambiente relativamente simples potencializa o desempenho da empresa, dando a ela poder de competição com empresas que podem investir em equipamentos e softwares específicos para esse tipo de ambiente. Lembrando sempre que este ambiente ou qualquer outro ambiente nestas condições descritas neste trabalho, são dependentes da estrutura da infraestrutura da empresa, onde pode entrar questões como: cabeamento, solução de energia, poder computacional das máquinas agregada no cluster.

Portanto, foi provado que é possível construir um ambiente de alta disponibilidade e replicação das informações utilizando apenas ferramentas open source existentes há um bom tempo no mercado. A utilização deste ambiente fica recomendada principalmente para empresas de pequeno porte que estão iniciando no mercado e buscam competitividade para crescer e aumentar a sua fatia do mercado.

Para trabalhos futuros, propõe-se a utilização do sharding em uma grande base de dados, aonde essa base pode ser dividida para alguns computadores, aumentando assim horizontalmente o poder computacional utilizado ao invés de aumentar verticalmente, ou seja, realizar upgrades de disco, memória ou até mesmo a compra de um novo computador para ser utilizado.

Referências

Boaglio, F (2015) "MongoDB Construa novas aplicações com novas tecnologias", São Paulo, São Paulo, Casa do código.

Cabrezas, S. "Guia para criar um replicaSet com MongoDB" https://www.beeva.com/beeva-view/desarrollo/guia-para-crear-un-replicaset-con-mongodb/, abril.

Cambiuccci, W. "Construindo arquitetura de alta disponibilidade" http://blogs.msdn.microsoft.com/wcamb/2011/08/09/construindo-arquiteturas-de-alta-disponibilidade/, outubro.

Capucho T. (2016) "MySQL Cluster: Alta Disponibilidade com Interface NoSQL" https://www.devmedia.com.br/mysql-cluster-alta-disponibilidade-com-interface-nosql/33982/, março.

Carlos (2007) "Servidores em Cluster e Balanceamento de Cargas" https://www.hardware.com.br/artigos/cluster-carga/, fevereiro.

Chodorow, K (2013) "MongoDB The Definitive Guide", Sebastopol, Califórnia O'Reilly Media.

Cruz T. (2015) "O que é alta disponibilidade de banco de dados" http://blog.dbaacademy.com.br/alta-disponibilidade/, outubro.

MongoDB, (2008) "Adicionar um árbitro ao conjunto de réplica" https://docs.mongodb.com/manual/tutorial/add-replica-set-arbiter/, abril.

MongoDB, (2008) "Réplica Definir Árbitro" http://docs.mongodb.com/manual/core/replica-set-arbiter/, abril.

Monteiro D. (2017) "Você conhece a replicação no MongoDB?". http://db4beginners.com/blog/replicacao-no-mongodb/, abril.

Kobellarz, Jordan (2015) "#6 MongoDB – Replicação" http://jordankobellarz.github.io/mongodb/2015/08/09/mongodb-replicacao.html, abril.

Paniz, D (2016) "NoSQL Como armazenar os dados de uma aplicação moderna", São Paulo, São Paulo, Casa do código.

Pereira Filho, N. A "Serviços de Pertinência para Clusters de Alta Disponibilidade." http://www.teses.usp.br/teses/disponiveis/45/45134/tde-04102004-104700/, maio.

Redação PTI (2011) "Negócios X Infraestrutura de TI: em busca da alta disponibilidade" https://www.profissionaisti.com.br/2011/07/negocios-x-infraestrutura-de-ti-em-busca-da-alta-disponibilidade/, fevereiro.

Rezende R. (2013) "Alta disponibilidade no banco de dados Oracle" https://www.devmedia.com.br/alta-disponibilidade-no-banco-de-dados-oracle/28971, outubro.

Sadalage, P.J and Fowler, M (2012) "NoSQL Distilled A brief guide to the emerging world of polyglot persistence", Crawfordsville, Indiana Pearson Education, Inc.

Seguin, K (2013) "The Little MongoDB Book" https://github.com/karlseguin/the-little-mongodb-book, outubro.

Tiwari, S (2011) “Professional NoSQL”, Indianápolis, Indiana, Wrox.

Augusto, E. (2017) “Fundamentos de MongoDB I”.https://medium.com/gdgcampinas/fundamentos-de-mongodb-i-ea673780d53b, agosto.

Dakar, R. “MongoDB – O que é e para que serve”. http://desenvolvedor.ninja/mongodb-o-que-e-e-para-que-serve/, agosto.