IFBA / ADS - Análise e Desenvolvimento de Sistemas manuscript No.
(will be inserted by the editor)

AutoNode - Um SGBD NoSQL com Suporte a Replicação Transparente

Norton José Dantas Pacheco Júnior · Orientadora: Flávia Maristela Santos Nascimento

Resumo Com o rápido crescimento do uso da Internet, as bases de dados que atendem aos diversos serviços disponíveis neste ambiente ganharam destaque. Neste contexto, os sistemas gerenciadores de banco de dados - SGBDs - relacionais passaram a ser amplamente utilizados como principal ferramenta para a gerência dos dados. Apesar do amplo uso e de ser uma tecnologia estabelecida no mercado, os SGBDs relacionais apresentam algumas limitações quando estamos lidando com uma grande massa de dados, principalmente no que diz respeito ao desempenho e à escalabilidade. No sentido de atender a tais requisitos, surge uma solução que trata os dados de forma não estritamente relacional, batizada de NoSQL. O objetivo deste trabalho é desenvolver uma solução de SGBD NoSQL, com foco no requisito de replicação transparente e escalável de bases de dados, visto que a configuração para a replicação nas soluções existentes é explícita, exigindo maior conhecimento do usuário dos SGBDs.

Palavras-chave database - SGBD - relational model - data structure - NoSQL - bigdata - software

1 Introdução

Nos últimos anos o perfil das aplicações computacionais mudou: muitos serviços que antes eram oferecidos numa arquitetura baseada em servidores centralizados e hierárquicos, passaram a lidar com uma infraestrutura baseada em distribuição[30].

O barateamento de equipamentos - processadores, discos rígidos, memória *RAM* - também colaborou para que fosse possível realizar tarefas que exigissem alta disponibilidade e tolerância a falhas, através do processamento distribuído de dados.

Segundo Tanenbaum[36], um Sistema Distribuído "é uma coleção de computadores independentes que se apresentam aos usuários como um sistema único coerente"; e para Coulouris[23] "é um sistema em que os componentes se localizam em uma rede de computadores e coordenam suas ações através de passagem de mensagens".

De fato, o avanço dos sistemas distribuídos ajudou a impulsionar as aplicações, principalmente aquelas baseadas na internet e com isso também foi necessário considerar as vantagens e desafios associados a esta arquitetura. Hoje, boa parte das aplicações manipula um grande volume de dados e apresenta características intrínsecas de sistemas distribuídos.

Como exemplo, podemos citar as aplicações com acesso contínuo a grandes repositórios de dados distribuídos, tais como os sítios de redes sociais, como Orkut, Facebook e Twitter, que armazenam informações de milhões de usuários, inseridas a todo momento e disponibilizadas de maneira quase instantânea. Neste caso, as bases de dados que atendem aos diversos serviços disponíveis ganharam destaque, e os sistemas gerenciadores de banco de dados (SGBDs) representam a principal ferramenta para a gerência de dados.

Existem SGBDs para diferentes modelos de dados, como por exemplo os SGBDs Orientados a Objetos, criados para atender as especificidades da programação orientada a objetos[35]. Entretanto, do ponto de vista da utilização e número de usuários, os SGBDs relacionais se apresentam como uma tecnologia madura e bem estabelecida no mercado. De acordo com o *ranking* dos SGBDs mais populares, apresentado em 01 de Janeiro de 2013 pelo site do DB-Engines[19] - que serve como base de conhecimento sobre SGBDs relacionais e

Instituto Federal da Bahia
Rua Emídio Santos s/n - Barbalho
Tel.: +55-71-21099457
E-mail: norton.ads@gmail.com, flaviamsn@ifba.edu.br

NoSQL - os cinco primeiros sistemas apontados como os mais populares são relacionais.

Apesar de bem estabelecido e amplamente difundido em seus quarenta anos de uso, o modelo relacional apresenta alguns obstáculos a serem superados, principalmente no que diz respeito ao desempenho e escalabilidade[26], que sejam compatíveis com o contexto das aplicações modernas.

Neste sentido, no que se refere a organização dos dados, um novo paradigma vem sendo usado de forma crescente. Sua principal característica é organizar os dados de forma não apenas relacional, sendo por isto batizado de NoSQL. Este novo paradigma vem se destacando pelos ganhos em termos de desempenho, principalmente nas operações de recuperação de dados e inserção[32]. Isto pode ser corroborado ao compararmos o *ranking* dos SGBDs mais populares de 2013 e 2014. Este ano, no ranking apresentado no mês de Fevereiro, os quatro primeiros mais populares são relacionais, mas o quinto SGBD mais popular é um SGBD NoSQL[19].

As principais características dos SGBDs NoSQL são [26,30]:

- alto desempenho nas operações de leitura e escrita;
- suporte ao armazenamento de grandes volumes de dados;
- suporte a escalabilidade;
- baixo custo.

Uma das principais razões para a adoção de soluções NoSQL é a busca por escalabilidade e flexibilidade. Além disso, os SGBDs NoSQL são utilizados para lidar com o desafio de armazenar e tratar enormes quantidades de dados com o máximo de eficiência e rapidez.

Considerando estes aspectos, o presente trabalho propõe uma solução de replicação transparente ao usuário, ou seja, onde não haja necessidade de quem utiliza o sistema indicar onde e quando a réplica é realizada. Diferente das soluções mais difundidas, onde a configuração de replicação é necessariamente explícita, ou seja, o usuário do sistema gerenciador do banco de dados precisa definir o modo de replicação, que vai depender da solução utilizada, considerando tanto SGBDs relacionais quanto SGBDs NoSQL.

2 Estrutura do Documento

Este documento está organizado da seguinte forma. A Seção 3 apresenta a motivação do trabalho. A Seção 4 apresenta uma descrição dos modelos ou paradigmas de base de dados, incluindo uma os modelos de SGBDs relacionais e o paradigma NoSQL, caracterizando-o levantando seus pontos fortes, que o torna viável e a melhor escolha em certas situações. Além disto a Seção compara as arquiteturas relacional e não somente relacional (NoSQL) para SGBDs, ressaltando suas principais limitações no que se refere às aplicações modernas. A Seção 5 trata dos aspectos de replicação para SGBDs, destacando os principais problemas e soluções estabelecidas no contexto dos sistemas distribuídos. A Seção 6 apresenta os trabalhos relacionados à proposta apresentada neste documento. A Seção 7 apresenta o modelo proposto neste trabalho e discute os principais aspectos de implementação. Os detalhes da implementação são apresentados na Seção 8. A Seção 9 apresenta a conclusão. E os trabalhos futuros possíveis de serem desenvolvidos no sistema apresentado estão na Seção 10.

3 Motivação

No contexto dos sistemas modernos o volume de dados a ser armazenado tem sido foco da atenção de muitos trabalhos[26]. Frequentemente, a expressão Big Data é usada para referenciar a grande quantidade de informação e dados de várias fontes gerados diariamente e armazenados no formato digital na internet, em *data centers* e/ou laboratórios acadêmicos. Existem inúmeros exemplos na literatura[18], dos quais vale a pena mencionar alguns:

- **Telescópio do projeto Sloan Digital Sky Survey**: este equipamento busca mapear cerca de 1/4 do espaço visível da terra. Em 2000 gerou mais dados nas suas primeiras semanas de funcionamento do que em toda a história da astronomia[25], e depois de 10 anos acumulou 140TB de dados. Há uma estimativa de que o próximo telescópio a ser posto em atuação em 2016 acumulará essa mesma quantidade de informação a cada 5 dias.
- **Walmart**: este grande varejista lida com dados de mais de 1 milhão de transações de clientes a cada hora, alimentando bases de dados que estima-se alcançar 2,5 PB. Essa quantidade de dados equivale a 167 vezes a quantidade de dados contida nos livros da biblioteca do Congresso Norte-Americano[18].
- **Facebook**: Em julho de 2010, o Facebook armazenava 50 bilhões de fotos de 500 milhões de usuários ativos, acumulando 130 TB de dados de logs a cada dia, 100 milhões de acessos diários e 2 trilhões de objetos em cache para acesso rápido[8].
- **Amazon**: a maior varejista virtual, possui 137 milhões de usuários cadastrados ativos[10]. A Amazon armazena os dados de todos os seus usuários incluindo informações sobre suas compras, produtos pretendidos, históricos das visitas à páginas de produtos, preferências e comportamento mercadoló-

gico. Qualquer um de seus clientes pode visualizar todo seu histórico de compras sem esperar mais do que algumas frações de segundo no carregar de uma página contendo o seu perfil.

Considerando estes exemplos, a primeira pergunta que surge é: "onde essa imensa quantidade de dados pode ser armazenada de modo que esses dados possam ser processados, analisados e pesquisados, de maneira rápida e eficiente?". O paradigma NoSQL surge como possível resposta para este problema, na medida em que propõe novas técnicas para lidar com os aspectos de desempenho, principalmente na operação de recuperação, e escalabilidade das aplicações[30].

Quando se fala em atender aos requisitos de desempenho dos sistemas que lidam com *Big Data*, a escalabilidade surge como um dos principais objetivos dos SGBDs NoSQL[26]. Esta escalabilidade dos sistemas pode ser atingida através de técnicas de replicação, por exemplo, segundo a qual os dados podem ser distribuídos entre diferentes nós separados fisicamente. Desta forma, é possível ter suporte para alta disponibilidade e armazenamento e recuperação eficientes para um grande volume de dados[26].

Mesmo que as soluções NoSQL atendam às demandas já citadas, como flexibilidade e escalabilidade, para se atingir a replicação de dados é preciso um conhecimento maior da ferramenta, como veremos na Seção 6. Como dito previamente, o presente trabalho tem por objetivo tornar a replicação um processo transparente ao usuário, porque a solução apresentada pelos SGBDs NoSQL existentes não oferece recursos para uma replicação sem a interferência do usuário.

Na próxima Seção, serão apresentados o conceito de SGBD e os modelos de SGBDs relacionais e NoSQL.

4 Sistema Gerenciador de Banco de Dados

Os sistemas gerenciadores de banco de dados, SGBDs, são responsáveis por fornecer ferramentas para o manipulação de dados de maneira segura e eficiente. Dentre as tarefas que fazem parte das obrigações de um SGBD, podemos destacar[33]:

- **Fornecer o compartilhamento de dados de maneira concomitante e segura**: em inúmeros sistemas de informação que fazem uso de bases de dados, é comum a necessidade de utilização compartilhada e simultânea por diferentes usuários. Portanto faz-se necessário atender a esta demanda;
- **Controlar o acesso à base de dados**: bases de dados são utilizadas por diversos tipos de usuários com diversas finalidades, por isso é interessante que haja um meio de controlar o acesso a estes dados, de modo a definir os devidos tipos de permissões dos usuários como leitura e escrita.
- **Efetuar backups ou cópias de segurança dos dados gerenciados**: para evitar que falhas externas à base de dados acabem por corromper ou perder os dados é importante que o SGBD forneça algum tipo de ferramenta ou artifício para realização de cópias de segurança - mais comumente chamadas de *backups*.

Tabela 1 Formato padrão de uma tabela no modelo relacional

	Coluna 1	Coluna 2	Coluna 3
Linha 1	valor 1	valor 2	valor 3
Linha 2	valor 4	valor 5	valor 6

Na Seção 4.1 serão apresentadas as principais características dos SGBDs relacionais e na Seção 4.2 as características dos SGBDs NoSQL. Em seguida, na Seção 4.3 estes dois sistemas serão comparados com o objetivo de salientar suas vantagens e adequações no contexto das aplicações modernas.

4.1 SGBDs Relacionais

Embora existam no mercado SGBDs que lidam com diferentes modelos de dados, como por exemplo o modelo orientado a objetos e o modelo objeto-relacional[35], os SGBDs relacionais são amplamente utilizados[19]. De acordo com o *ranking* apresentado em 2014, os quatro SGBDs mais utilizados são relacionais.

Para a estruturação dos dados, os SGBDs relacionais se limitam ao formato de tabelas para a inserção e recuperação de informações. A estrutura dos dados armazenados é definida de acordo com o *layout* das tabelas, dos nomes e dos tipos de dados de cada uma de suas colunas[35].

A Tabela 1 demonstra que cada linha (também chamada tupla ou registro) de uma tabela representa um relacionamento entre um determinado número de valores (ou atributos).

A Tabela 2 demonstra o uso real de uma tabela no modelo relacional.

As tabelas de um SGBD relacional comumente estão relacionadas entre si, através das chaves estrangeiras. Chave estrangeira trata-se de um atributo ou coluna de uma tabela que também está presente em uma outra tabela, porém como chave primária. Neste caso, as consultas que possuem subconsultas, conhecidas como consultas aninhadas, são mais complexas e podem ser responsáveis por um comprometimento do desempenho do sistema como um todo.

Tabela 2 Exemplo de uso de uma tabela de banco de dados

Cidade	Estado	População	Fundação
Salvador	Bahia	2.676.606	29-03-1549
Recife	Pernambuco	1.546.516	12-03-1537
Feira de Santana	Bahia	556.756	13-11-1832

4.2 SGBD NoSQL

Os SGBDs NoSQL abrangem uma variedade de tecnologias diferentes para banco de dados, que foram desenvolvidas em resposta à crescente necessidade de armazenamento de grandes volumes de dados, objetos e produtos. Além disso, podemos mencionar ainda a elevada frequência de acesso das aplicações, bem como o desempenho esperado e as necessidades de processamento[30].

De fato, os SGBDs NoSQL oferecem uma vasta coleção de tipos de armazenamento. Esses tipos de armazenamento se assemelham aos vários tipos de estruturas de dados bem conhecidos da ciência da computação, dentre os quais podemos mencionar :

- *Key/Value* (Chave/Valor): formato que armazena dados em pares de chave/valor muito parecido com os *hash maps* ou dicionários. Cada chave corresponde a um valor e vice-versa. A velocidade de pesquisa é maior que em bancos relacionais, inclusive pesquisas, inserções e atualizações de dados. O que diferencia as soluções NoSQL entre si é a implementação deste modelo - alguns armazenam na memória RAM (memcached[11], Tarantool[17]), outros armazenam em disco (MemcacheDB[12], Kyoto Cabinet [10], MongoDB[13]), e outros fazem uso do disco e da memória no armazenamento (Redis[15]).
- Documento: formato de armazenamento ideal para textos ou documentos. Como toda a solução é preparada para receber uma grande cadeia de caracteres, a pesquisa de texto (*full text search*) é muito rápida se comparada ao mesmo tipo de pesquisa em bancos relacionais. Temos como exemplo as soluções Apache CouchDB[3] e RavenDB[14], que armazenam os dados em arquivos no formato JSON[1] (Figura 1) e MongoDB, que armazena os dados em arquivos no formato BSON[2] - um JSON binário.
- Coluna ou Família de Coluna: neste formato os dados são armazenados em colunas independente de tabelas, cada coluna de dados é indexada. Com esse formato, somente as colunas envolvidas nas pesquisas são acessadas, resultando numa redução de custo de pesquisas. A depender da implementação, cada coluna pode ser tratada por apenas um processo. Cada coluna pode conter um mesmo conjunto de tipos de dados, o que pode melhorar as taxas de compressão - nos casos que são aplicados. Soluções que seguem esse modelo: Hadoop[4], Hbase[5] e Cassandra[2].
- Grafos: formato que armazena dados que são fortemente ligados entre si. Além dos dados, representados pelos vértices, é possível apresentar os relacionamentos entre eles através das arestas. São utilizados em aplicações que envolvem serviços baseados em localização (GPS), e em aplicações relacionadas a redes sociais - como o Twitter, mantenedora do FlockDB[7], banco de dados tolerante a falhas e orientado a grafos - onde usuários são interligados entre si.

```
{
    "matricula": "20153116019",
    "nome": "Fulano de Tal da Silva",
    "curso": "Análise e Desenvolvimento de Sistemas",
    "cpf": "12345678901",
    "statusMatricula": true,
    "endereco": [
        {
            "logradouro": "Rua Emídio dos Santos",
            "numero": "123",
            "bairro": "Barbalho",
            "complemento": "Próximo ao IFBA",
            "cep": "40301100"
        }
    ],
    "nascimento": [
        {
            "ano": "1997",
            "mes": "04",
            "dia": "01"
        }
    ],
    "disciplinas": [
        {
            "inf003": [
                {
                    "semestre": "2015.1",
                    "prova1": "10",
                    "prova2": null,
                    "prova3": null,
                    "faltas": "2"
                }
            ],
            "inf004": [
                {
                    "semestre": "2015.1",
                    "prova1": "10",
                    "prova2": null,
                    "prova3": null,
                    "faltas": "2"
                }
            ]
        }
    ]
}
```


Figura 1 Exemplo de arquivo JSON, que seria diferente da tabela dos SGBDs relacionais

[1] especificação JSON - json.org
[2] especificação BSON - bsonspec.org

Na seção seguinte, as vantagens e desvantagens dos SGBDs relacional e NoSQL serão apresentadas, de forma a ressaltar a aplicabilidade de cada uma destas soluções.

4.3 Comparativo entre SGBD Relacional e SGBD NoSQL

Quando se pensa em comparar os SGBDs relacional e NoSQL, a escalabilidade é quase sempre a característica mais referenciada[30]. Para os SGBDs relacionais, a escalabilidade pode ser atingida se o SGBD for executado num computador mais robusto e consequentemente, mais caro[30].

Em geral, a escalabilidade é atingida através de técnicas como replicação e particionamento. Cada técnica usada traz consigo um benefício no quesito escalabilidade, mas são direcionadas a problemas específicos.

Por exemplo, os SGBDs que executam muitas operações de leitura podem ter uma melhoria no desempenho através do uso da replicação do repositório de dados principal em repositórios escravos (*slaves*), como forma de balancear o volume de acesso. Já os SGBDs que executam muitas operações de escrita podem ter uma melhoria no desempenho através da replicação do servidor principal (*master*) entre servidores que também terão a possibilidade de escrita (replicação *master-master*), porém essa solução exige um alto nível de dedicação na solução de problemas de sincronização entre os servidores, um vez que dados podem ser inseridos concomitantemente.

Devido a lógica complexa dos SGBDs relacionais e a maneira como seus dados são estruturados, o acesso para consulta, inclusão ou alteração pode se tornar lento a medida que o número de registros (tuplas) vai aumentando, principalmente quando falamos de dados normalizados e que precisam utilizar junções (*joins*) frequentemente em suas operações.

Denomina-se normalizado, os bancos de dados relacionais que atendem as técnicas pré-determinadas (formas normais), certificando o armazenamento como consistente e buscando diminuir a redundância de dados[33].

Assim, há uma perda de desempenho nas operações de leitura e escrita por conta do aumento da concorrência no acesso aos dados[31]. Neste caso, é necessário que as várias condições sejam estabelecidas para que as propriedades ACID[35] de um banco relacional sejam atendidas.

As propriedades ACID são as características que permitem aos SGBDs executarem transações, evitando que ocorram problemas de concorrência, quando os dados estão disponíveis a mais de um cliente. São elas:

- Atomicidade: Determina que uma transação deve acontecer de uma vez, para que os dados envolvidos não sejam alterados por outra transação, comprometendo assim o resultado;
- Consistência: Determina que uma transação não pode levar o banco de dados a um estado de inconsistência, comprometendo toda a base de dados;
- Isolamento: Determina que uma transação será iniciada somente se não houver outra transação em execução que utilize algum dado em comum;
- Durabilidade: Determina que toda transação deve ter o seu resultado persistente, ou seja, que todas as mudanças realizadas no banco de dados perdurem.

Em um ambiente de alta concorrência é natural o uso do bloqueio (*lock*) de tabelas ou registros para atingir consistência dos dados, porém é um artifício que pode gerar lentidão no acesso, que pode até resultar em *deadlocks*[3] nas tabelas do banco de dados, levando até a falhas no sistema.

Além disto, conceitualmente, os SGBDs relacionais não trabalham adequadamente num ambiente replicado, justamente por conta de prováveis operações de junções nas consultas apresentadas[31]. O fato deste tipo de sistema trabalhar predominantemente com tabelas, frequentemente relacionadas através de chaves estrangeiras, pode representar uma limitação se esta estrutura não for a forma mais adequada para armazenar os dados[30].

Por outro lado, considerando os SGBDs NoSQL, algumas características se destacam e tornam este tipo de solução mais adequada. São elas[26]:

- Alta concorrência em leituras e escritas com baixa latência: o segredo das aplicações que operam em altos níveis de desempenho é a capacidade de lidar com leituras e escritas de dados em seus repositórios (Figuras 2, 3 e 4) . Para atender essa demanda precisamos pensar em outros modos de lidar com dados, diferentes dos tradicionais, considerados pelos SGBDs relacionais[31].
- Armazenamento e acesso eficiente de grande conjunto de dados: sites de pesquisa, pesquisas científicas, aplicações de processamento distribuído dentre vários tipos de aplicações precisam armazenar muitos dados e acessá-los de maneira eficiente.
- Alta disponibilidade: as soluções de armazenamento de dados que seguem o paradigma NoSQL possuem a alta disponibilidade como um requisito de projeto [20]. A grande maioria possue suporte nativo a replicação[20], que é a técnica mais utilizada para alcançar a alta disponibilidade[23].

[3] *Deadlock* é o momento em que um recurso é disputado por duas partes concorrentes, e nenhuma delas consegue acessar, causando uma parada no sistema[36]

– Escalabilidade: cada vez mais os sistemas precisam ter suas arquiteturas planejadas e que levem em consideração seu possível crescimento, portanto um bom sistema precisa estar pronto para atender altas demandas, e por consequência, expandir, ou seja, escalar.
– Baixo custo operacional: ao lidar com grande quantidade de dados através do processamento distribuído, é comum o uso de uma grande quantidade de computadores (servidores), e trabalhar de maneira organizada com um grande número de nós de processamento pode ser uma tarefa difícil, a qual as soluções NoSQL se prestam a resolver, de maneira nativa às suas implementações.

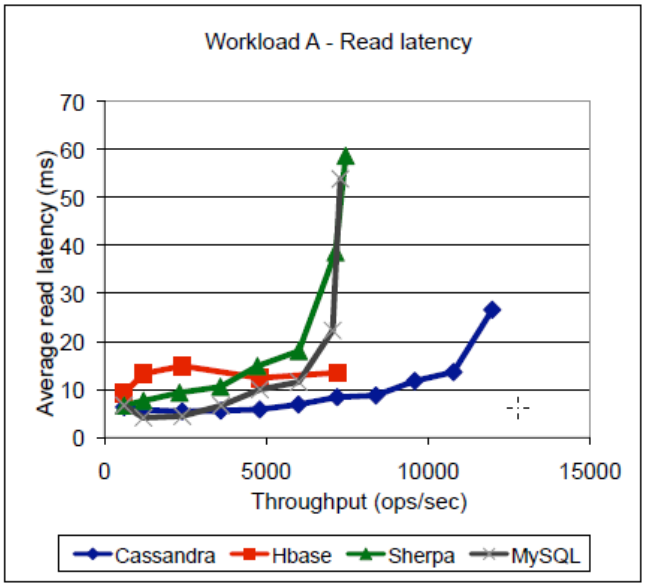

Figura 2 Gráfico de comparação da latência média de leitura (*read*) em um ambiente de escrita (*write*) intensiva[37]

Uma característica predominante entre as soluções NoSQL é o modo como elas são desenvolvidas. A maioria quase absoluta dos SGBD NoSQL são *open source*, ou seja, tem o código fonte aberto[4], e é desenvolvido em comunidade. Para este trabalho, após pesquisas, só foi possível encontrar uma solução NoSQL que não estava disponível com algum tipo de licença livre[5], no caso o BigTable[22]. O BigTable é uma solução NoSQL proprietária da Google que somente pode ser utilizada através do serviço Google App Engine[8]. Porém, existem duas soluções que seguem o mesmo modelo orientado a coluna; HBase e o Hypertable[9].

[4] opensource.org/docs/osd
[5] opensource.org/proliferation-report

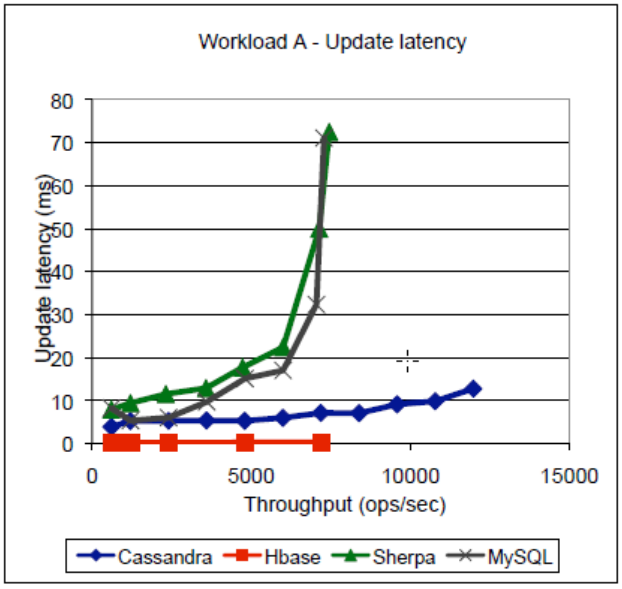

Figura 3 Gráfico de comparação da latência média de escrita (*write*) em um ambiente de escrita (*write*) intensiva[37]

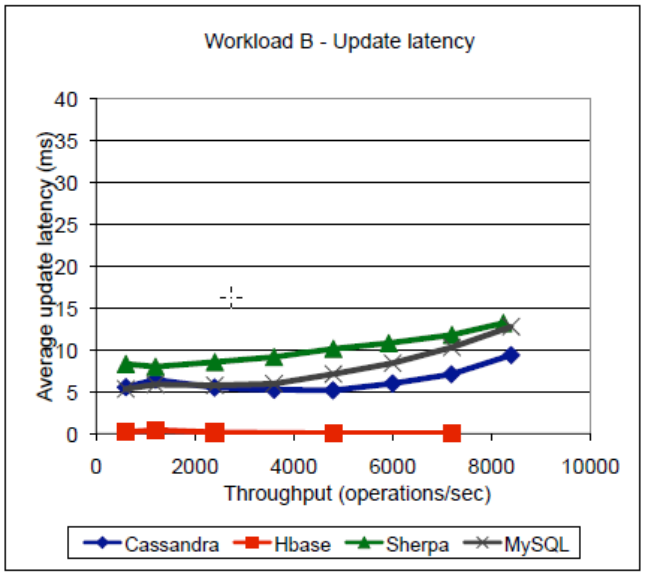

Figura 4 Gráfico de comparação da latência média de escrita (*write*) em um ambiente de leitura (*read*) intensiva[37]

5 Replicação para SGBDs

Sistemas distribuídos são caracterizados por possuírem computadores interligados em rede, que se comunicam e coordenam suas ações através da troca de mensagens. Uma das principais motivações para a adoção de uma arquitetura para sistemas distribuídos é a necessidade de compartilhamento de recursos. No caso das aplicações que lidam com *Big Data* a alta disponibilidade da aplicação é desejável e desempenha um papel

fundamental para o bom funcionamento das aplicações modernas[23].

Além disso, em sistemas distribuídos as falhas ganham um papel de destaque, uma vez que uma falha em um computador ou uma finalização inesperada em algum ponto da rede pode não ser imediatamente percebida por outros componentes com os quais ele se comunica. Prever e tratar este tipo de situação deve ser o foco da atenção dos projetistas de tais sistemas.

Neste contexto, a replicação ganha destaque, uma vez que é a principal solução para prover alta disponibilidade e tolerância a falhas em sistemas distribuídos[23]. A primeira, tem sua importância ressaltada por conta da atual tendência no sentido do desenvolvimento de aplicações móveis, com características de pervasividade e ubiquidade. Desta forma, a replicação é uma alternativa para maximizar o tempo de disponibilidade de dados, e pode ser trabalhada de modo a torná-lo virtualmente infinito, dependendo de boas escolhas de projeto e de questões externas, como a compra de grande quantidade de computadores e construção de *datacenters*. Já a segunda, é preocupação constante em qualquer tipo de aplicação computacional, uma vez que a única certeza que temos é que as aplicações falham[23].

O principal objetivo da replicação de dados é manter cópia dos mesmos em vários computadores. De fato, a replicação busca a melhoria de serviços e neste sentido, as principais motivações são[23]:

- Melhoria do desempenho: A melhoria do desempenho é alcançada através da divisão da carga de trabalho entre nós ou servidores de um sistema distribuído. Como exemplo temos os sistemas *web* onde os servidores compartilham o mesmo nome DNS (*Domain Name Server*) que podem ser associados aos vários endereços IP (*Internet Protocol*) dos servidores;
- Maior disponibilidade: Quando um conjunto de dados é compartilhada em nós, os dados ficam disponíveis em mais de um lugar, tornando o acesso aos dados menos concentrado, sem sobrecarga nas bases de dados;
- Tolerância a falhas: Dados mais disponíveis não são necessariamente dados rigorosamente corretos. Para que a replicação de dados se realize de maneira que os repositórios se mostrem consistentes, é preciso garantir tal consistência no caso da ocorrência de possíveis falhas (descritas na Seção 5) .

Ao escolher um modelo ou protocolo de replicação de dados, temos que pesar os ganhos e a dificuldades de implementação das possíveis soluções - esta situação é comumente conhecida como *trade-off*. Um bom projeto reúne uma coleção de boas escolhas que busquem cumprir o requisito principal proposto: a replicação.

5.1 Tipos de Replicação

No trabalho de implementar a replicação em um sistema distribuído, Coulouris[23] divide os tipos de replicação em Replicação Passiva e Ativa. Nas próximas seções estes tipos de replicação serão tratados.

5.1.1 Replicação passiva

No modelo de replicação passiva (*backup* primário)[23] (ilustrada na Figura 5), o cliente ou *front-end* que deseja salvar ou atualizar um dado envia a requisição para o gerenciador primário de dados, que por sua vez fica responsável pela replicação. Após realizar a requisição do cliente, ele envia os dados ou atualização para os outros gerenciadores de dados. Desta forma, os gerenciadores que receberam ou atualizaram os dados no segundo momento são vistos como servidores *backups* (reservas).

No momento em que o gerenciador de dados principal falhar, um dos gerenciadores de dados *backups* ou secundários pode assumir a função do gerenciador principal, que estava sendo utilizado pelo cliente ou *front-end*.

Na replicação passiva é preciso que a figura do gerenciador de dados principal seja determinada assim como as figuras dos gerenciadores secundários. No momento da falha do gerenciador principal, é executado um processo de eleição do próximo gerenciador principal, que será eleito entre os gerenciadores secundário.

Neste caso, temos a replicação como artifício para alcançar tolerância a falhas. Porém, para que o sistema consiga sobreviver a um número f de falhas de gerenciadores de dados, é preciso que haja um número de $f+1$ gerenciadores de dados.

A replicação passiva não é um modo de alcançar alto desempenho ou escalabilidade, pois todos os clientes acessam somente um gerenciador de réplica. Aumentando o número de gerenciadores de réplica, não ocorrerá o aumento da capacidade de resposta, e portante não haverá aumento no desempenho.

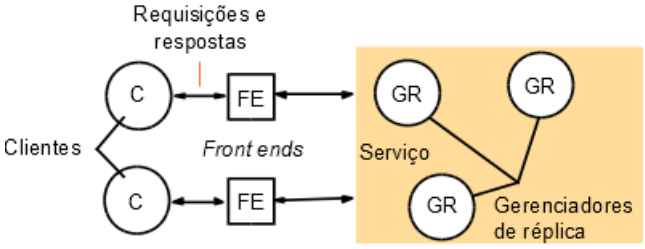

Figura 5 Replicação passiva

5.1.2 Replicação ativa

No modelo de replicação ativa (ilustrado na Figura 6), os gerenciadores de réplicas desempenham papéis equivalentes, não há a determinação de um gerenciador principal. Os responsáveis pela replicação dos dados são os *front-ends*. Eles que se encarregam de se comunicar com os gerenciadores de réplica e enviar as requisições por *multicast*[23][6].

No momento de falha de um dos gerenciadores de réplica, os *front-ends* que utilizam aquele gerenciador com problema deixam de enviar mensagens para ele.

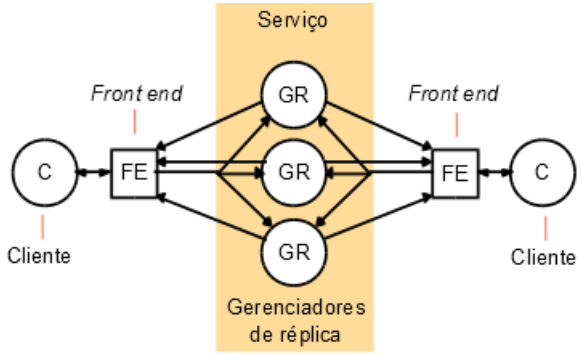

Figura 6 Replicação ativa

6 Trabalhos Relacionados

Dentro do universo dos bancos de dados relacionais, verificamos que a replicação não é algo trivial ou automática. Cada solução de SGBD possui a sua maneira de resolver a questão da replicação. Nesta seção, será apresentada brevemente como as soluções mais populares[19] alcançam a replicação de dados.

6.1 Replicação com Oracle

O SGBD Oracle utiliza duas ferramentas para implementar a replicação de uma base de dados[6]: o Oracle GoldenGate e o Oracle Streams. Portanto, quem utiliza o Oracle como SGBD e quer implementar a replicação de dados precisa dominar uma ferramenta que não faz parte do SGBD.

[6] *Multicast* é a comunicação ou envio de mensagem para mais de um destinatário de rede

6.2 Replicação com Microsoft SQL Server

No Microsoft Sql Server 2012, é utilizado o paradigma que divide o trabalho de replicação em *agents*[7] em cada servidor: *Agent Administration*, *Snapshot Agent*, *Log Reader Agent*, *Distribution Agent*, *Merge Agent* e *Queue Agent*. Cada servidor pode desempenhar os seguintes papéis na replicação: *distributor*, *subscriber* e *publisher*.

A Figura 7 ilustra a arquitetura de replicação Microsoft SQL Server, com os seus papéis.

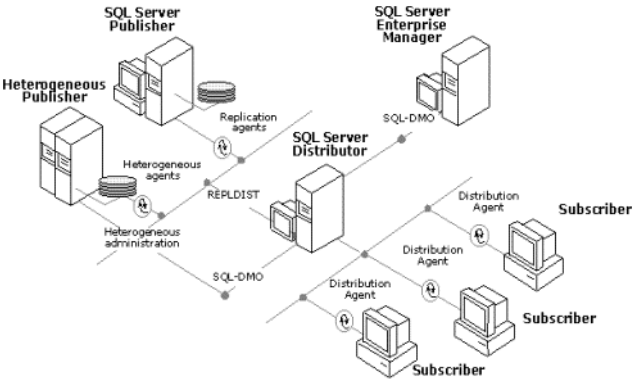

Figura 7 Arquitetura da replicação do Microsoft SQL Server

Todo esse número de *agents* e papéis, exige que o usuário conheça bem da ferramenta para implementar a replicação.

6.3 Replicação com MySQL

A solução de código-fonte aberto mais popular[19], o SGBD relacional MySQL apresenta como meio de replicação os papéis de *master* e *slave*[8]. O modo de replicação pode ser do tipo *master-slave* ou do tipo *master-master*.

No modo de replicação *master-slave* (Figura 8), um servidor trabalha como repositório principal (*master*), e quando um dado é acrescentado (*insert*) ou atualizado (*update*), a operação é registrada no arquivo de *log* (registro) e enviada a um ou mais servidores secundários, chamados *slaves*. No momento que ocorre uma falha no servidor *master*, é possível tornar um servidor *slave* em um repositório principal. Porém, é preciso pensar em alguns potenciais problemas[21] para executar isso:

- No momento de apontar os clientes para o novo servidor *master*, é preciso atualizar o novo servidor com

[7] http://technet.microsoft.com/en-us/library/bb677158.aspx

[8] dev.mysql.com/doc/refman/5.5/en/replication-howto.html

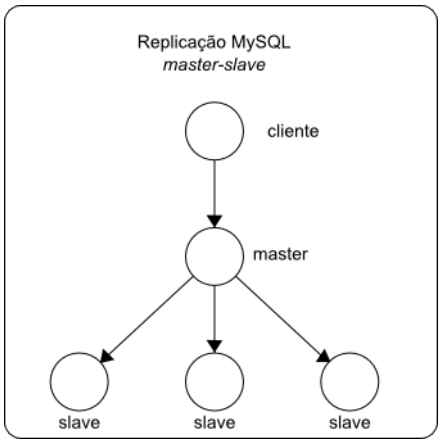

Figura 8 Replicação MySQL *master-slave*

as numerações das operações registradas no servidor *master*, se o arquivo de log não for comprometido;
- Ao tornar um servidor *slave* em um servidor *master*, não significa que os outros servidores *slaves* têm o repositório idênticos.
- Ao recuperar o servidor *master* defeituoso, algumas informações contidas nele podem nunca ter saído ou replicadas para outros servidores.

O MySQL apresenta uma forte limitação forte na replicação *master-master* (Figura 9). Um servidor *master* só pode replicar para somente um outro servidor *master*. Portanto, no caso de falha de um servidor *master* contendo o repositório principal, só existe a opção de mais um servidor alternativo.

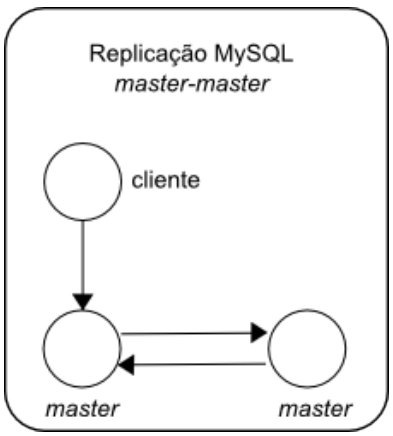

Figura 9 Replicação MySQL *master-master*

Para superar estas limitações, as empresas que lidam com grande quantidade de dados e usam o MySQL, como o Google, Facebook e Twitter, criam suas aplicações em torno deste software, de modo a contornar suas limitações ou acrescentando outra camada de software em suas aplicações, que funcionam como *middleware*.

6.4 Replicação com MyStore

MyStore[28] é uma solução NoSQL fruto de um trabalho acadêmico, que se propõe a aprimorar a disponibilidade de dados não-estruturados, apresentando um novo meio de armazenamento distribuído, através do uso de outra duas solução NoSQL, o MongoDB.

A replicação como forma de assegurar a eficiência e confiabilidade no armazenamento de dados é comum em várias implementações e arquiteturas de sistemas distribuídos, como no MyStore.

O modelo de replicação de dados implementada pressupõe um número N de cópias físicas distribuídas entre os nós, tendo assim o N como o fator de replicação que deve ser pré-determinado. Cada dado inserido através de um *put*, é enviado a um nó que age como coordenador que armazena e retransmite (replica) o mesmo dado para outros nós que fazem parte de uma faixa própria do coordenador.

Também é implementado o fator W limite mínimo de escritas. O número de replicações é contado, e se for menor que W, a operação de escrita é dada como falha.

Para leitura de dados (*get*) é considerado o fator R como limite mínimo de leitura. Se o número de leituras de um determinado dado é maior que R, é considerada uma operação bem sucedida. Usando o fator R, durante as operações de leitura, se o número de leituras é menor que R, é considerada falha de replicação - possivelmente ausência de alguns nós - e então são executadas mais operações de escritas para alcançar o fator R.

A latência de escritas e leituras (*gets/puts*) é definida pelo tempo da replicação mais lenta. Durante a escrita, se um nó falha, o dado é escrito no nó seguinte, até que a escrita seja dada como bem sucedida.

6.5 Replicação com Cassandra

A replicação de dados para alcançar alto desempenho e confiabilidade é um dos paradigmas considerados na implementação do Cassandra[29]. Assim como no MyStore, os dados são replicados em N nós, fator que deve ser pré-definido previamente. Também prevê um coordenador que se encarrega de assegurar a replicação entre nós da sua faixa. Porém, esse coordenador é autônomo e funciona de modo a controlar várias outras funções de um sistema distribuído, o Zookeeper[27].

O Zookeeper se encarrega de tarefas importantes na coordenação de um sistema distribuído, incluindo controle de replicação e atualização de dados distribuídos em nós, controle de filas de processamento, resolução de nomes (como o DNS) e outras funcionalidades. Seu principal foco é a detecção e resolução de falhas em nós,

portanto não abordaremos o Zookeeper, mas paradigmas e modelos de replicação.

6.6 Replicação com MongoDB

A replicação implementada no Mongodb[16] exige que seja seguido um modelo de conjunto de réplicas limitadas a 12 nós ou servidores. Cada servidor é visto com um membro de um conjunto de servidores réplicas (*replica set members*) e deve ser configurado individualmente. Após configurados e inicializados, os nós elegerão um nó primário, que funcionará como um gerenciador de réplica primário (como apresentado na Seção 5.1.1).

Diferente do MongoDB, o sistema proposto não apresenta limitação na quantidade de servidores réplicas. De fato, o que vai limitar a quantidade de servidores réplicas é a infraestrutura de rede, pois quanto maior o número de nós, maior será o número de mensagens de replicação transmitidas pela rede. Outra diferença é a inexistência de um nó primário ou principal.

Desta forma, podemos verificar que a configuração para replicação em SGBDs NoSQL é explícita e está fortemente atrelada à arquitetura do SGBD.

A proposta deste trabalho é implementar uma solução que tira do usuário a necessidade de configurar a replicação. Cada nó replicará dados sem que o usuário aponte onde e como. Na Seção seguinte o modelo proposto será apresentado.

7 Modelo Proposto

7.1 Considerações

Ao se projetar um sistema distribuído é preciso considerar o ambiente no qual ele será executado ou hospedado. Hoje em dia, o padrão de interligação das partes distribuídas mais utilizado é a rede Ethernet (ambiente físico) junto com a família de protocolo TCP/IP (ambiente lógico). Considerando-se que além do próprio sistema, o ambiente também é suscetível a problemas, temos que levantar premissas ou hipóteses para evitar ou contornar as possíveis ocorrências de falha.

Desta forma, levando-se em consideração as possíveis falhas no meio de comunicação entre nós de um sistema distribuído, os projetistas e estudiosos de sistemas distribuídos definiram falsas suposições, mais conhecidas como "as 8 falácias da computação distribuída"[34], que devem ser questionadas e contornadas em nível de projeto. São elas:

1. a rede é confiável;
2. a latência é zero;
3. a largura da banda é infinita;
4. a rede é segura;
5. a topologia da rede não muda;
6. a rede possui um administrador;
7. o custo de transporte é zero;
8. a rede é homogênea.

Essas falácias foram definidas por Peter Deutsch, antigo desenvolvedor da Sun Microsystems, e James Gosling, outro antigo desenvolvedor da mesma empresa, mas mais conhecido como criador da linguagem de programação Java.

A palavra "rede" não se refere somente à rede física de computadores utilizada para interligar os nós de um sistema distribuídos. Ela se refere também ao conjunto de nós coexistentes de um sistema distribuídos. Então, a primeira falácia de que "a rede é confiável" refere-se à possibilidade de falha de um equipamento, ou de um ou mais nós.

A falha em um nó é algo perfeitamente possível e por isso deve ser considerada. Um nó pode se visto como um computador, uma máquina virtual sendo executada em um computador ou até mesmo um processo em execução. Para que qualquer um dos três apresente uma falha só é preciso que ocorra uma falha em um dos vários componentes físicos de um dos computadores.

É comum a comercialização de equipamentos que possuam a especificação de MTBF (*Mean Time Between Failure*), traduzindo, período médio entre falhas. Este índice é estabelecido por fabricantes de equipamentos após testes realizados em laboratório - não sendo possível a comparação entre equipamentos de fabricantes diferentes, pois cada um realiza testes de acordo com a sua própria metodologia e não há padronização.

Outro fator relevante, a latência pode ser considerada como tempo de resposta de uma determinada requisição entre nós distintos, ou ainda quanto tempo leva para dados serem trafegados de um nó para outro.

7.2 Filosofia

O objetivo do modelo proposto foi de construir um sistema distribuído composto por nós autônomos, onde qualquer um pode receber tarefas de escrita, de leitura e de replicação de dados. As tarefas são compartilhadas entre os nós através da troca de mensagens. Cada mensagem possui informações necessárias para armazenamento ou leitura de dados.

Um dos objetivos do projeto é construir um sistema distribuído capaz de ser executado em computadores com vários tipos de configuração - inclusive as mais modestas - e em vários tipos de sistemas operacionais. Porém, tendo em mente o desejo de alcançar um bom nível

de desempenho, precisamos lembrar que todo sistema operacional tem suas demandas de processamento, memória e armazenamento. Para que essas demandas não comprometam o desempenho do nó, o ideal é que o sistema operacional seja enxuto e que faça uso de poucos recursos computacionais do computador no qual será instalado.

Com o intuito de executar o sistema em computadores com várias configurações, é necessário implementar o sistema de modo a torná-lo portável[18], evitando que a sua execução seja impedida devido a diferença de hardware entre os computadores. Por isso a escolha do uso da plataforma Java.

7.3 Arquitetura

O estilo arquitetural do sistema proposto está próximo ao *Peer-to-peer* (P2P). O modelo descrito no presente artigo baseia-se em nós automatizados capazes de se conectar com outros nós, visando conseguir a replicação de dados de maneira confiável e eficiente.

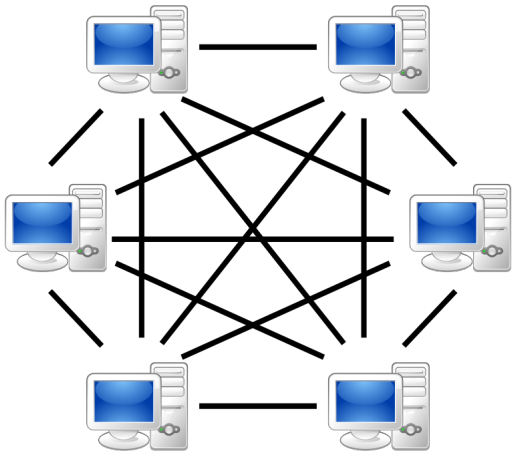

Figura 10 Arquitetura *Peer-to-peer* (P2P)[23]

Cada nó é responsável por se conectar a outro nó. É preciso inicialmente indicar o endereço IP de algum outro nó em execução. O método de descoberta de outro nós poderia utilizar o artifício de *broadcast* - emissão de pacotes em massa para todos os pontos da rede - porém esse método pode utilizar muito recurso da rede e de processamento na medida que a quantidade de nós for aumentando.

No momento da primeira conexão, o nó em execução passa a lista de nós conhecidos e em execução e seus respectivos endereços IP para o nó recentemente iniciado.

No momento 1, o primeiro nó é iniciado. No momento 2, um novo nó (nó 2) é executado, se conecta a um nó existente (nó 1), que por sua vez, no momento 3, transmite informação sobre outros nós existentes na rede (nó 3) como é mostrado na Figura 11.

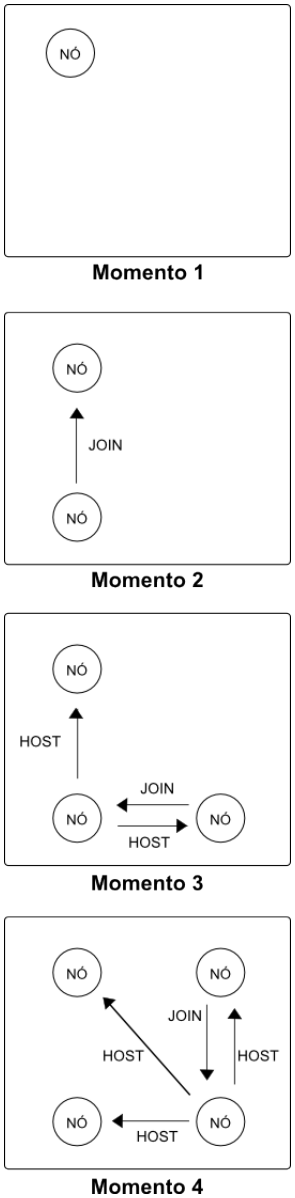

Figura 11 Inicialização do conjunto de nós

8 Implementação

Tendo como objetivo tornar a replicação um processo transparente ao usuário, a implementação é que se encarrega de fazer os nós se comunicarem entre si para alcançar a replicação de dados.

A implementação apresentada usa como referência o Cassandra[29], um SGBD NoSQL orientado a colunas, desenvolvido originalmente pela equipe de desenvolvimento do Facebook e hoje é mantido pela fundação Apache[1]. O Cassandra também é implementado utilizando a linguagem Java.

A construção do sistema gerenciador de banco de dados é dividida em módulos a fim de facilitar e organizar de maneira eficiente para manutenção futura e melhor entendimento do código.

Por se tratar de um projeto utilizando a tecnologia Java, portanto, tendo ela uma linguagem de programação orientada a objetos, os módulos da aplicação serão implementados como classes. Alguns destes mesmos módulos são executados de modo concorrente, por isso as classes principais da aplicação implementam a classe *Runnable*.

8.1 Segurança

Assumimos que a segurança pode ser implementada fora do sistema, ou seja, o sistema operacional e os outros componentes externos do sistema devem prover segurança. Isso se deve ao fato de que a segurança, apesar de requisito importante, não faz parte do grupo de requisitos funcionais.

Porém, por se tratar de um projeto de sistema distribuído, composto por vários nós, é preciso estabelecer algum método que identifique ou certifique que um determinado nó, de fato, faz parte do sistema.

Para tanto, foi introduzido o conceito de duas chaves - que podem ser vistas como senhas. Uma identifica o sistema (GROUPID), como conjunto de nós que trabalham juntos, e outra serve com identificação do próprio nó (HOSTID), funcionamento como artifício para impedir que um nó externo (possivelmente malicioso) faça parte do conjunto.

Cada nó se comunica com o resto do conjunto - através do envio de mensagens - usando essas duas chaves. E cada nó destinatário identifica e processa a mensagem utilizando as chaves enviadas pelo remetente.

8.2 Módulos

Core - responsável por iniciar o nó, ler as informações iniciais dos arquivos de configuração locais para a execução, e inicia os outros módulos. O Core é implementado na classe *main.java*, sendo a classe principal do projeto. Para incialização, recebe como argumento o endereço IP de um nó pré-existente/em execução:

```
#java main.class
```

Server - módulo que escuta/espera por conexões e repassa as novas conexões às novas *threads* para recepção de dados. É implementado na classe *Server.java* e tem como argumento a porta a ser escutada:

```
Server server = new Server(9000);
/* 9000 refere-se à porta */
Thread thServer = new Thread(server);
/* associa uma Thread ao objeto da classe Server */
thServer.start();
/* inicia a execução da Thread */
```

WorkerServer - iniciado pelo *Server* quando este recebe uma conexão, se encarrega de receber os dados a serem armazenados e replicados. Também associado a uma *Thread* criada pelo *Server*, recebe como argumento o *socket* criado pelo *Server* para uma nova conexão:

```
WorkerServer worker = new
WorkerServer(clientSocket);
Thread thWorker = new Thread(worker); /* associa
uma Thread ao objeto da classe WorkerServer */
thWorker.start(); /* inicia a execução da Thread */
```

Replicator - responsável por reenviar dados recebidos para replicação em outros nós. Quando uma mensagem de replicação é adicionada no *pool* de mensagens, ele é instanciado no *core*, e encarregado de enviar os dados através de conexões com outros nós, e por sua vez, agindo como um cliente comum, tendo a única diferença no tipo de mensagem (REPL).

```
replicator = new Replicator(m, listaHost);
/* o m refere-se ao dado a ser replicado, e listaHost se
refere à lista de hosts aos quais o dado será
replicado*/ Thread thRep = new Thread(replicator);
/* associa uma Thread ao objeto da classe Replicator
*/
thRep.start(); /* inicia a execução da Thread, pois a
replicação é executada concorrentemente à classe main
para evitar que o sistema fique bloqueado */
```

Storage - responsável por armazenar os dados vindos dos clientes e dos outros nós participantes do conjunto de nós. Ele é uma *interface* que é implementada pela

classe ColunaSet, que é criada e manipulada na classe *main.java*.

```
Storage storage = new ColunaSet(dirStor);
/* criação do conjunto de colunas para
armazenamento */
```

Os métodos implementados são os seguintes:

```
public void add(String coluna, String chave, String
valor);
/* adiciona um valor relacionado à chave fornecida */
public String getValor(String coluna, String chave);
/* retorna o valor relacionado à chave fornecida */
public String getChave(String coluna, String valor);
/* retorna uma possível chave relacionada ao valor
fornecido */ public int getPopChave(String coluna,
String chave);
/* retorna a popularidade da chave, ou seja quantos
nós replicaram essa chave, quanto maior o valor, mas
nós possuem essa chave e valor */ public void
create(String nomeColuna);
/* cria uma nova coluna */
```

8.3 Comunicação

Os nós conversam entre si através de mensagens que definem as solicitações enviadas por cada nó. As solicitações são dividas em solicitações de armazenamento, replicação, recuperação e entrada no grupo.

Padrão de mensagem:

```
>>>OPERACAO>>>COLUNA>>>CHAVE>>>
VALOR>>>HOSTID>>>GROUPID
```

As mensagens seguem um padrão, usando como separador a *string* ">>>", onde:

- o campo "OPERACAO" indica o tipo de operação;
- o campo "COLUNA" indica qual coluna está envolvida no campo;
- o campo "CHAVE" indica a chave cujo valor será consultado, alterado ou excluído;
- o campo "VALOR" indica o valor a ser utitlizado;
- o campo "HOSTID" indica qual *host* enviou a mensagem;
- o campo "GROUPID" indica a identificação do grupo. Este último campo serve para autenticar a mensagem, se ela não tem o GROUPID correto, será ignorada e descartada.

8.4 Mensagens

Na comunicação entre os nós são utilizadas oito tipos de mensagens:

- Mensagem de armazenamento (STOR);
- Mensagem de replicação (REPL);
- Mensagem de recuperação (RECV);
- Mensagem de entrada de grupo (JOIN);
- Mensagem que apaga dado da coluna (DELE);
- Mensagem que apaga coluna (DELC);
- Mensagem que envia informações de hosts para outro host (HOST);
- Mensagem que remove host da lista de hosts conhecidos (LEAV);

A mensagem de armazenamento (STOR) - é enviada por um cliente para um nó integrante do sistema, que por sua vez reenvia (replica) a outro nó. O cliente envia a coluna - onde o par deve ser salvo -, uma chave e um valor, junto com a informação do cliente - neste caso CLIENT - e a informação do grupo de nós (GROUPID), seguindo o seguinte padrão:

```
>>>STOR>>>COLUNA>>>CHAVE>>>
VALOR>>>CLIENT>>>GROUPID
```

A mensagem de replicação (REPL) - é enviada por um nó para outros nós. O nó que recebe esta mensagem, identifica que é uma mensagem replicada e somente serializa;

```
>>>REPL>>>COLUNA>>>CHAVE>>>
VALOR>>>HOSTID>>>GROUPID
```

A mensagem de recuperação (RECV) - é enviada por um cliente para um dos nós, e funciona como uma solicitação do valor armazenado referente a uma determinada chave:

```
>>>RECV>>>COLUNA>>>CHAVE>>>
null>>>CLIENT>>>GROUPID
```

A mensagem de entrada no grupo (JOIN) carrega a identificação do nó (*host*), a identificação do grupo (GROUPID) que funciona como uma senha e a identificação do nó. Só os nós que possuem a chave podem compartilhar - enviar e receber - mensagens dos nós que participam do grupo:

Junto com a chave do grupo, é enviado o ID do nó, que tem que ser único para cada nó do grupo. No modelo, é utilizado o endereço MAC (da camada física da

>>>JOIN>>>HOSTID>>>NODEIP>>>
NODEPORT>>>HOSTID>>>GROUPID

rede Ethernet) como identificador do nó. Isso não significa que o endereço é utilizado pelo nó - pelo menos, não explicitamente, uma vez que o grupo se comunica através da rede utilizando a família de protocolo TCP/IP, e em um nível ou camada mais baixa, está sendo utilizado o padrão Ethernet.

A mensagem que apaga dado da coluna (DELE) carrega a identificação da coluna e a chave que aponta para o valor que será apagado:

>>>DELE>>>COLUNA>>>CHAVE>>>
null>>>HOSTID>>>GROUPID

A mensagem que apaga coluna (DELC) carrega a identificação da coluna que será apagada do armazenamento:

>>>DELC>>>COLUNA>>>null>>>
null>>>HOSTID>>>GROUPID

A mensagem que envia lista de *hosts* para outro *host* (HOST) carrega a identificação de um *host* da lista de *hosts* conhecidos. Essa lista será enviada de um nó presente para um nó que está entrando no grupo. A lista não será enviada de uma só vez. A informação sobre os integrantes da lista será enviada individualmente, ou seja, cada mensagem vai conter a informação de somente um nó da lista:

>>>HOST>>>HOSTID>>>NODEIP>>>
NODEPORT>>>HOSTID>>>GROUPID

A mensagem que remove *host* da lista de *hosts* conhecidos (LEAV) carrega a informação de um *host* ou nó que deixou de responder ou aceitar novas mensagens de replicação. O nó que receber esta mensagem exclui o nó pertecente na mensagem da lista de nós conhecidos:

>>>LEAV>>>HOSTID>>>NODEIP>>>
NODEPORT>>>HOSTID>>>GROUPID

8.5 Consistência dos dados

A consistência dos dados é garantida através do índice de popularidade. No momento da recuperação de dados, é preciso garantir que o usuário vai conseguir o dado armazenado correto, ou seja, é preciso alcançar a consistência dos dados, e para isso é adotado um índice de consistência de dado.

Quando o dado enviado pelo cliente é armazenado em um dos nós. Junto com o dado é armazenado uma espécie de meta-dado, que funciona como contador e pode ser visto como um índice de popularidade do dado, entre os nós do grupo. Quanto maior é o índice, mais popular é dado, portanto, mais consistente. Como exemplificação do modo de armazenamento temos:

COLUNA(CHAVE(VALOR, INDICE))

O índice agindo como contador funciona de modo a contar quantos servidores possuem a réplica de determinado dado, portanto um dado que possua índice 3, está replicado em pelo menos três nós do grupo.

O nó que executa a solicitação de recuperação enviada por um cliente, verifica a existência e o índice de popularidade do dado, antes de retornar ao cliente. Se o índice é menor do que o estipulado no arquivo de configuração - por exemplo 3 -, ele repassa a solicitação a outro nó. Se, no nó seguinte, o dado não existir ou não tiver um índice aceitável, é retornado a mensagem de dado inexistente.

8.6 Inicialização do servidor

O servidor é inicializado através da classe *main.java*, que é responsável por instanciar as outras classes ou módulos do nó e inicializar as threads concorrentes:

A Figura 11 mostra como se dá o processo de inicialização do conjunto de nós. A Figura 12 mostra o fluxograma do processo de inicialização de um nó.

8.7 Armazenamento

Quando uma mensagem contendo um dado para armazenamento é enviada por um cliente para um servidor do grupo, por exemplo:

>>>STOR>>>NOME>>>2009216019>>>
Norton>>>CLIENT>>>001122334455

O sistema seleciona a coluna “NOME”, armazena a chave “2009216019” e o valor “Norton”.

O código que executa o armazenamento é o seguinte:

As outras funcionalidades de armazenamento foram descritas na Seção 8.2.

```
storage.add(COLUNA,CHAVE,VALOR);
/* armazena um valor relacionado à uma chave na
coluna passada como parâmetro */
```

8.8 Replicação

Como no modo de replicação ativa apresentada por Coulouris[23], descrita na seção 5.1.2, o sistema apresentado não possui capacidade de linearização, ou seja, se um dos nós parar de responder por um momento, por um período mais breve o possível, e houver comunicação de dados entre os outros nós sem a sua participação, ele será considerado um nó com problema e será retirado da lista de *hosts* conhecidos.

O modo de replicação adotado se assemelha ao Gossip[9][23] cuja arquitetura é apresentada na Figura 13. A diferença entre a arquitetura Gossip e o modo de replicação adotado é a ausência de *front-ends*. O cliente faz acesso diretamente aos gerenciadores de réplica, utilizando o driver de conexão, este pode ser visto como um componente de comunicação com o conjunto de nós, o próprio sistema distribuído.

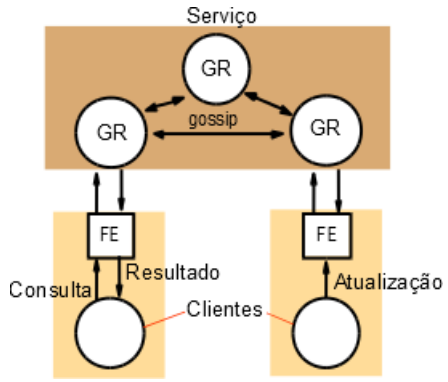

Figura 13 A arquitetura *Gossip* para replicação[23]

8.9 A tolerância a falhas

A falha arbitrária ou bizantina acontece quando um dos nós, ao replicar um dado, envia o valor do dado alterado, capaz de criar inconsistência na base de dados, como mostra a Figura 14.

A maneira de contornar uma quantidade de falhas arbitrárias f é garantir que o conjunto tenha $3f+1$ nós, fazendo com que o nó ou os nós que falham arbitrariamente se tornem a minoria entre $3f + 1$ nós.

As falhas por omissão[23] podem ser: por omissão de processo; ou por omissão de comunicação.

[9] expressão do Inglês, melhor traduzida como fofoca

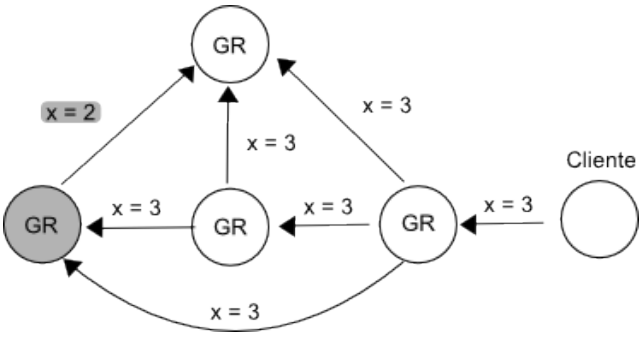

Figura 14 Exemplo de falha bizantina

A falha por omissão de processo ocorre quando um dos gerenciadores de réplica não responde, ou seja, para de receber mensagens e também não é capaz de enviar mensagens para réplica.e

A falha por omissão de comunicação ocorre quando uma mensagem enviada não é recebida em seu destinatário, ou seja, a comunicação não é contretizada de fato. Neste caso, o nó que apresenta erro na conexão, e por consequência, na transmissão da mensagem é retirado da lista de *hosts* conhecidos pelo nó remetente, que por sua vez envia a indentificação do nó com problema para todos os outros *hosts* conhecidos.

No sistema aqui descrito, as duas falhas são vistas do mesmo modo, e são tratadas utilizando o mesmo processo, a retirada do nó com problema da lista de *hosts* conhecidos.

8.10 Considerações

O sistema apresentado não apresenta meios de resolver falhas por particionamento. O particionamento ocorre quando um subgrupo do sistema distríbuído para de se comunicar com o grupo todo, se isolando logicamente, ou seja, particionando o sistema distribuído[23]. Portanto, quando um ou mais nós se separa do conjunto de nós, não há meio de linearização, então não há como atualizar o nó reintegrante com os dados já inseridos ou alterados do nó.

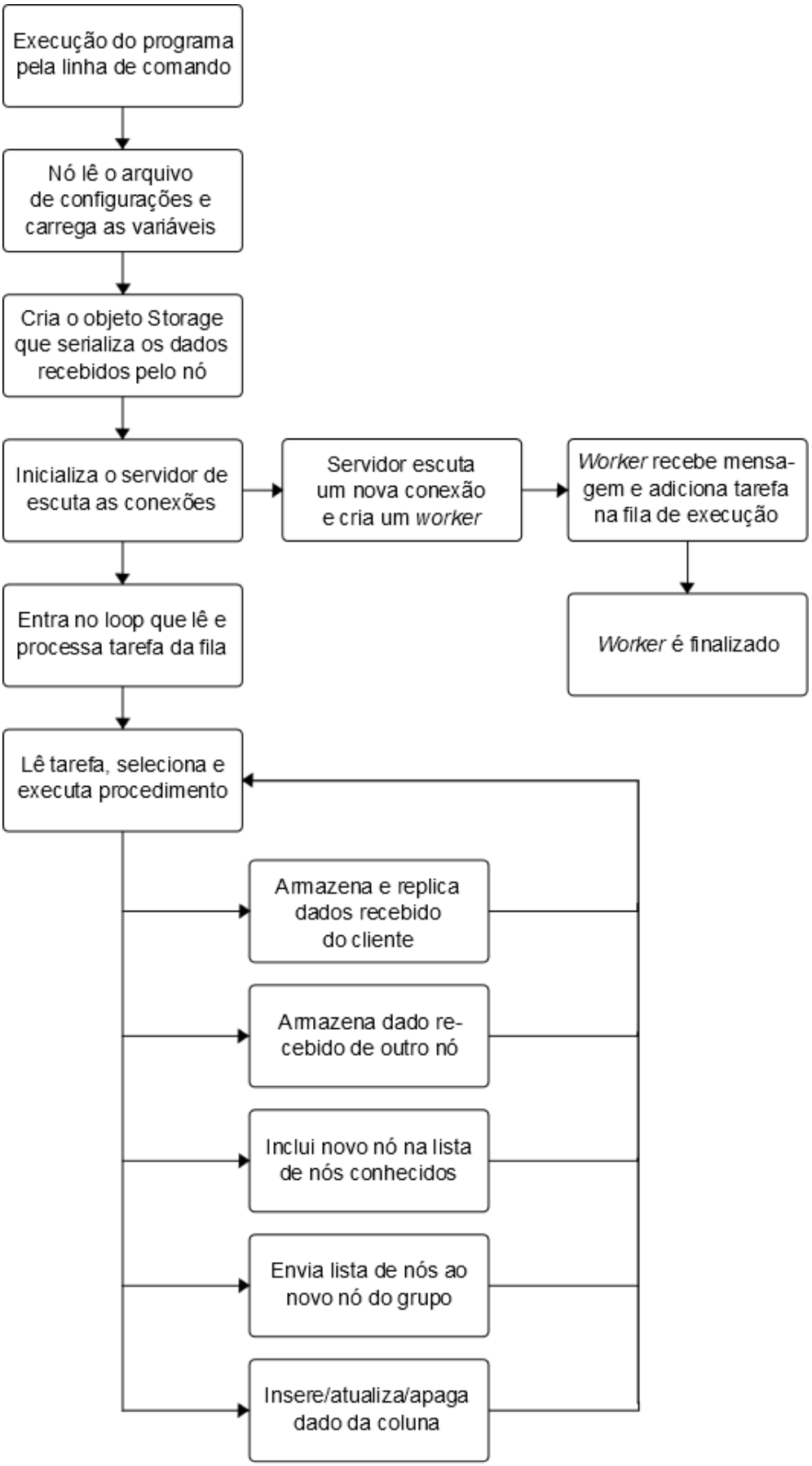

Figura 12 Fluxograma de execução de um nó

9 Conclusão

O software apresentado tem como função principal o armazenamento de dados orientado a colunas, com suporte a replicação transparente ao usuário do SGBD.

Foram indicados e levantados, na Seção 6 os procedimentos necessários para se alcançar a replicação nas soluções de SGBD mais utilizadas[19], e o modo explícito da tarefa que deve ser realizada pelo usuários destes SGBDs.

10 Trabalhos Futuros

Como trabalho futuro, é possível implementar um modo de execução de tarefas distribuídas entre os nós. Para tanto, os tipos de tarefas poderão ser elencadas em um arquivo XML (eXtensible Markup Language - Linguagem de Marcação Extensível) que relacionará a tarefa com a sua classe, que por sua vez será carregada dinamicamente pelo sistema. Esse objetivo se assemelha ao algoritmo *Map/Reduce*[24] criado pela Google para distribuir processamento de grande volume de dados entre vários computadores.

Outra característica relevante e possível de ser implementada é a atualização automatizada. Uma vez que os nós compõem um sistema distribuído, é possível que o conjunto de computadores utilizados possa crescer ao ponto, que a administração de todo o conjunto alcance um custo muito alto. Ou seja, um grande número de computadores exige uma padronização na manutenção do sistema, tanto de *software* quanto de *hardware*.

A linearização também é uma tarefa possível de implementada. Com a linearização, vai ser possível que novos nós sejam incluídos no conjunto e passem a possuir o repositório idêntico aos atuais nós integrantes.

Referências

1. Apache. URL http://www.apache.org
2. Apache cassandra project. URL http://cassandra.apache.org
3. Apache couchdb. URL http://couchdb.apache.org
4. Apache hadoop. URL http://hadoop.apache.org
5. Apache hbase. URL http://hbase.apache.org
6. Data replication and integration. URL http://www.oracle.com/technetwork/database/features/data-integration/
7. Flockdb. URL http://github.com/twitter/flockdb
8. Google app engine: Platform as a service. URL http://developers.google.com/appengine
9. Hypertable. URL http://hypertable.org
10. Kyoto cabinet: a straightforward implementation of dbm. URL http://fallabs.com/kyotocabinet
11. memcached - a distributed memory object caching system. URL http://memcached.org
12. Memcachedb: A distributed key-value storage system design for persistent. URL http://memcachedb.org
13. Mongodb. URL http://www.mongodb.org
14. Ravendb - 2nd generation document database. URL http://ravendb.net
15. Redis. URL http://redis.io
16. Replica set tutorials - mongodb manual. URL docs.mongodb.org/manual/administration/replica-sets
17. Tarantool - a nosql database in a lua script. URL http://tarantool.org
18. Data, data everywhere (2010). URL http://www.economist.com/node/15557443
19. Knowledge base of relational and nosql database management systems (2013). URL http://db-engines.com/en/raking
20. Nosql databases explained (2014). URL http://www.mongodb.com/learn/nosql
21. Bell, C., Kindahl, M., Thalmann, L.: MySQL High Availability. O'Reilly (2010)
22. Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A distributed storage system for structured data. In: Operating Systems Design and Implementation, 2006 7th USENIX Symposium on, pp. 205–218 (2006)
23. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Sistemas Distribuídos, Conceitos e Projetos. Bookman (2013)
24. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Operating Systems Design and Implementation, 2004 5th USENIX Symposium on (2004)
25. Gu, Y., Grossman, R., Szalay, A., Thakar, A.: Distributing the sloan digital sky survey using udt and sector. In: e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on, pp. 56–56 (2006)
26. Han, J., Haihong, E., Le, G., Du, J.: Survey on nosql database. In: Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, pp. 363–366 (2011)
27. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: Wait-free coordination for internet-scale systems. In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'10, pp. 11–11. USENIX Association, Berkeley, CA, USA (2010)
28. Jiang, W., Zhang, L., Qiang, W., Jin, H., Peng, Y.: Mystore: A high available distributed storage system for unstructured data. In: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, pp. 233–240 (2012)
29. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. **44**(2), 35–40 (2010)
30. Leavitt, N.: Will nosql databases live up to their promise? Computer **43**(2), 12–14 (2010)
31. Li, Y., Manoharan, S.: A performance comparison of sql and nosql databases. In: IEEE Pacific Rim Conf. Communications, Computers and Signal Processing, pp. 15–19 (2013)
32. Liu, Y., Wang, Y., Jin, Y.: Research on the improvement of mongodb auto-sharding in cloud environment. In: Computer Science Education (ICCSE), 2012 7th International Conference on, pp. 851–854 (2012)
33. Oliveira Celso H. Poderoso de; Oliveira, C.H.P.d.: SQL - Curso prático. NOVATEC (2008)

34. Rotem-Gal-Oz, A.: Fallacies of distributed computing explained (1994)
35. Silberschatz, A., Korth, H., Sudarshan, S.: Sistema de Banco de Dados. Makron Books (1999)
36. Tanenbaum, A.S., Steen, M.V.: Sistemas Distribuídos, princípios e paradigmas. Pearson (2008)
37. Tudorica, B.G., Bucur, C.: A comparison between several nosql databases with comments and notes (2011)