Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Armazenamento de Dados Abertos com NoSQL

Jorge Luiz Andrade

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio de Almeida

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientador) — CIC/UnB
Prof. Dr. Professor I — CIC/UnB
Prof. Dr. Professor II — CIC/UnB

CIP — Catalogação Internacional na Publicação

Andrade, Jorge Luiz.

Armazenamento de Dados Abertos com NoSQL / Jorge Luiz Andrade.
Brasília : UnB, 2016.
67 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. bancos de dados, 2. nosql, 3. dados abertos

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Armazenamento de Dados Abertos com NoSQL

Jorge Luiz Andrade

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Maristela Terto de Holanda (Orientador)
CIC/UnB

Prof. Dr. Professor I
CIC/UnB

Prof. Dr. Professor II
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 15 de dezembro de 2016

Dedicatória

Dedico a....

Agradecimentos

Agradeço a....

Abstract

A ciência...

Palavras-chave: bancos de dados, nosql, dados abertos

Abstract

The science...

Keywords: databases, nosql, open data

Contents

List of Figures

List of Tables

Chapter 1

Introdução

Dados abertos tem ganhado importância cada vez maior em nossa sociedade. O volume desses dados, que podem ser definidos como dados livres para acesso, utilização e modificação [5], tem crescido cada vez mais, e vem sendo necessário encontrar novas formas para realizar o seu armazenamento e analise, comumente realizados por meio de bancos de dados.

Bancos de dados podem ser definidos como um conjunto de dados que se relazionam entre si e armazenados de forma que possam ser acessados posteriormente, quando necessario [25]. Os bancos de dados relacionais predominaram por pelo menos nas últimas três décadas, mas seu desempenho em certas aplicações atuais, principalmente naquelas que trabalham com grande volumes de dados, denominado *Big Data*, vem sendo questionado.

Esse questionamento levou à criação do movimento NoSQL, um novo paradigma de armazenamento de dados que ignora certas restrições dos bancos relacionais tradicionais e tentam melhorar seu armazenamento e desempenho por meio de um sistema distribuído em *clusters*, com características de escalabilidade, tolerância à falhas e um melhor desempenho ao se operar grandes volumes de dados.

A utilização de bancos NoSQL para o tratamento de grande volumes de dados provenientes de dados abertos é uma possibilidade a ser analisada, a fim de permitir um acesso mais fácil e rápido por parte da população.

Vários modelos NoSQL são descritos nesse trabalho, e o banco Cassandra, um representante de bancos orientados à colunas, foi utilizado. O Cassandra é atualmente um projeto da fundação *Apache*, tendo sido originalmente proposto e utilizado pelo *Facebook*, com base em trabalhos anteriores da *Amazon* (*Dynamo*) e do *Google* (*BigTable*). É um banco distribuído que armazena seus dados em forma de colunas e linhas com esquema flexível. Essa características é importante para o armazenamento de dados governamentais abertos, devido à sua natureza extremamente variável de um período a outro.

Chapter 2

Dados Abertos

O termo "Dados abertos" no conceito que conhecemos hoje surgiu em 1995 em um documento da agência científica americana no contexto da abertura de dados geofísicos e ambientais. Esse conceito vem sendo ampliado e estimulado nos últimos anos por diversos movimentos [23].

A *Open Definition* define um dado como aberto "se qualquer pessoa está livre para acessa-lo, utiliza-lo, modifica-lo, e compartilha-lo — restrito, no maximo, a medidas que preservam a proveniência e abertura." [5]. A abertura dos dados evita mecanismos de controle e restrições sobre os mesmos, o que permite seu uso de forma livre [23].

A Open Knowledge Foundation, organização mundial que promove a abertura de dados [9], resume esses pontos em:

- **Disponibilidade e Acesso**: os dados devem estar disponíveis como um todo e sob custo não maior que um custo razoável de reprodução, preferencialmente possíveis de serem baixados pela internet. Os dados devem também estar disponíveis de uma forma conveniente e modificável.
- **Reutilização e Redistribuição**: os dados devem ser fornecidos sob termos que permitam a reutilização e a redistribuição, inclusive a combinação com outros conjuntos de dados.
- **Participação Universal**: todos devem ser capazes de usar, reutilizar e redistribuir - não deve haver discriminação contra áreas de atuação ou contra pessoas ou grupos. Por exemplo, restrições de uso 'não-comercial' que impediriam o uso 'comercial', ou restrições de uso para certos fins (ex.: somente educativos) excluem determinados dados do conceito de 'aberto'.

2.1 Dados abertos governamentais

Dados governamentais, em específico, também podem ser fundamentados por três leis e oito princípios.

Em 2009 o especialista em políticas públicas e dados abertos, David Eaves, propôs as seguintes três leis dos dados abertos governamentais, que também podem ser aplicadas a dados abertos em geral [16]:

1. Se o dado não pode ser encontrado e indexado na Web, ele não existe;

2. Se o dado não está disponível em um formato aberto e compreensível por máquinas, ele não pode ser reaproveitado;

3. Se algum dispositivo legal não permite que ele seja replicado, ele é inútil.

Em 2007, um grupo de trinta especialistas em governo aberto, reunidos em Sebastopol, na California, definiu os oito princípios de dados governamentais abertos, sendo eles [1]:

- **Completos**: todos os dados públicos deve estar disponíveis. Dados públicos são dados que não estejam sujeitos a limitações válidas de privacidade, segurança ou privilégio.
- **Primários**: os dados devem ser iguais aos coletados na fonte, no maior nível possível de granularidade, não estando em formas agregadas ou modificadas.
- **Atuais**: os dados devem ser disponibilizados tão rápido quanto necessário para garantir o seu valor.
- **Acessíveis**: os dados devem ser disponibilizados para os mais amplos públicos e propósitos possíveis.
- **Processáveis por máquinas**: os dados devem estar estruturados de forma razoável, de forma que permita um processamento automatizado.
- **Não discriminatórios**: os dados devem estar disponíveis para todos, sem necessidade de registro ou identificação do usuário.
- **Não proprietários**: os dados devem estar disponíveis em um formato que não seja controlado exclusivamente por uma entidade.
- **Livres de licença**: os dados não devem estar sujeitos a qualquer restrições de direitos autorais, marcas, patentes ou segredos industriais. Restrições razoáveis de privacidade, segurança e privilégio podem ser permitidas.

2.2 Contexto brasileiro de dados abertos

A participação brasileira na área de dados abertos tem um marco em 2011 com a criação da *Open Government Partnership*, uma aliança, contando inicialmente com a participação de 65 países, criada para fornecer uma plataforma internacional para reformadores nacionais comprometidos em fazer seus governos mais abertos, responsáveis e sensíveis aos cidadãos. Também foi criado o portal dados.gov.br, que disponibiliza dados governamentais de forma aberta [23].

O poder público brasileiro vem nos últimos anos realizando outras ações que promovem a abertura de dados governamentais. Essas ações visam benefícios como melhoria da gestão pública, transparência, controle a participação social, geração de emprego e renda e estímulo à inovação tecnológica [13]. Para atingir esse fim, no ano de 2012 foi definido, em instrução normativa, a implantação da INDA, Infraestrutura Nacional de Dados Abertos, "um conjunto de padrões, tecnologias, procedimentos e mecanismos de controle necessarios para atender às condições de disseminação e compartilhamento de dados e informações públicas no modelo de Dados Abertos" [7].

O governo tem importância fundamental na questão de dados abertos, devido â grande quantidade de dados que coleta e por serem públicos, conforme a lei, podendo ser tornados abertos e disponíveis para a sociedade [9].

Esses dados governamentais tem relevância tanto no âmbito da transparência, podendo haver rastreamento dos impostos e gastos governamentais; da vida pessoal, como na localização de serviços públicos por parte da população; economicamente, com a reutilização de dados abertos já disponíveis; e também dentro do próprio governo, que pode aumentar sua eficiência ao permitir que a população consulte diretamente dados que antes precisavam de interferência direta e individual por parte de funcionários públicos [9].

Chapter 3

Diferenças entre Bancos de Dados Relacionais e NoSQL

O armazenamento a a manipulação de dados tem sido um importante foco da computação desde o seu nascimento, tendo os bancos de dados suas raízes já na década de 60, principalmente em aplicações médicas e científicas [26], e em 1970, Edgar Codd propôs uma nova forma de armazenamento de dados, que ficou conhecida como modelo de dados relacionais (*relational model of data*) [12].

3.1 Bancos de Dados Relacionais

Podemos definir um banco de dados relacional como um conjunto de dados que se relacionam entre si e que são armazenados de forma persistente, podendo ser recuperados quando necessário. Os dados são armazenados em tabelas, que são organizadas por colunas, que definem uma categoria de um dado, e por linhas, que representam a instância de um dado [25]. A popularização desse modelo em virtude de suas características de persistência, concorrência e integração entre múltiplas aplicações, o transformou no modelo padrão de armazenamento computacional, principalmente em ambientes empresarias [28]. Outra questão de importância em bancos de dados computacionais é a não necessidade que o usuário tem de conhecer como esses dados são armazenados, o que foi possível com o uso dos chamados Sistemas Gerenciadores de Bancos de Dados (SGBDs) [20].

Outras opções surgiram ao longo dos anos, como os bancos orientados a objetos ou bancos *XML*. Nenhum deles, entretanto, conseguiu competir com o modelo já tradicional de dados relacionais. [28] Nos últimos anos, entretanto, o modelo conhecido como NoSQL vem surgindo como essa alternativa.

Bancos de Dados relacionais tem como um de seus requisitos a existência de chaves primarias, colunas que que identificam unicamente cada linha, ou registro, de uma tabela. Um banco de dados relacional necessita de três informações para recuperar um dado específico: o nome da tabela, o nome da coluna e a chave primaria do registro que esta sendo buscado [20].

3.1.1 Propriedades ACID

Interações com bancos de dados relacionais tradicionais são feitas por meio de transações, que podem ser definidas como operações de leitura e escrita que devem ocorrer de forma independente umas das outras [27]. Para garantir que isso ocorra, um SGBD deve prover as seguintes propriedades, conhecidas como propriedades ACID [19]:

- **Atomicidade**, onde uma determinada transação deve ser feita em sua totalidade, ou seja, todas as operações de que dela fazem parte devem ser bem sucedidas.
- **Consistência** diz que após cada transação, o estado do banco permanece consistente ao seu modelo.
- **Isolamento** garante que cada transação é executada independentemente de outras que estejam ocorrendo em concorrência.
- **Durabilidade** que define que o resultado de uma transação bem sucedida é persistido no banco, mesmo na eventualidade de falhas no sistema.

Essas propriedades, ao mesmo tempo que garantem a validade do esquema e dos dados em um banco, sacrificam desempenho e disponibilidade, características importantes em varias aplicações atuais [18].

3.1.2 Normalização

Uma pratica comum e recomendada no projeto de bancos de dados relacionais é a normalização. O processo de normalização segue regras conhecidas como formas normais, onde cada forma normal representa um incremento desse conjunto de regras [20]. Existem pelo menos seis formas normais, mas na maioria dos casos um banco é considerado bem projetado quando cumpre as exigências da terceira forma normal (3FN).

Primeira Forma Normal

Uma tabela esta na primeira forma normal (**1FN**) se cada tabela esta organizada por colunas e linhas, com cada linha possuindo uma chave primaria única que a identifica. Além disso cada campo deve possuir apenas valores atômicos. Ou seja, cada coluna deve guardar apenas uma informação, não podendo existir listas ou conjuntos de valores dentro de uma mesma coluna de uma linha.

Segunda Forma Normal

Uma tabela esta na segunda forma normal (**2FN**) quando, além de obedecer à primeira forma normal, possui todos atributos não-chave funcionalmente dependentes da chave primaria. Dependência funcional é definida como uma relação entre dois atributos tal que para cada valor único do atributo A, existe apenas um valor do atributo B associado a ele [20]. Em outras palavras, uma coluna não pode depender apenas de parte da chave primaria, ou seja, se uma tabela não possui chave primaria composta e esta na primeira forma normal, ela também esta na segunda forma normal.

Terceira Forma Normal

Uma tabela esta na terceira forma normal (**3FN**) quando, além de obedecer à segunda forma normal, não apresenta dependências transitivas, ou seja, cada atributo não-chave não pode ser determinado, ou dependente, de outro atributo não-chave.

3.2 Bancos NoSQL

O rápido crescimento no volume de dados nos últimos anos, principalmente após a bolha da Internet na década de 90 [28], trás uma necessidade de certa mudança em relação ao modelo tradicional. Modelos relacionais possuem diversas vantagem já citadas, porém restrições como propriedades ACID e normalização, levam ao surgimento de problemas quando precisamos aplica-los nesse domínio recente de expansão dos dados, por apresentarem problemas de escalabilidade, complexidade dos dados e rigidez em seus esquemas [25].

Isso levou ao surgimento de um movimento em direção ao novo paradigma denominado NoSQL. O termo foi utilizado pela primeira vez em 1998 para denominar um banco de dados que omitia o uso de SQL, o *Strozzi NOSQL*. A definição atual, porém, tem suas bases em uma reunião, conhecida como *NoSQL Meetup* realizada em 2009 em São Franscisco, Estados Unidos. Organizada por Johan Oskarsdon, criador do Last.fm, nela foram discutidas formas mais eficientes e baratas de organização dos dados, como as ja sugeridas em publicações anteriores, como o Google Bigtable em 2006 [11], e Amazon's Dynamo em 2007 [14, 29].

3.2.1 Definição e Características

Apesar do termo não tem uma definição precisa e universalmente aceita, sendo geralmente descrito como *Not Only SQL*, bancos NoSQL em geral são caracterizados, mas não definidos, como sendo não relacionais, sem esquema bem definido, distribuídos e tolerantes a falhas [28]. Buscam um processamento de dados rápido e de forma eficiente, evitando a rigidez dos bancos tradicionais. Entre as razões e vantagens dos bancos NoSQL podemos citar [29]:

- **Evitar complexidade desnecessária**: Bancos relacionais costumam aderir às já citadas propriedades ACID, além de serem restritos em seu esquema de dados. Bancos NoSQL costumam ignorar ou relaxar essas restrições a fim de obter um melhor desempenho.
- **Alto rendimento**: Bancos NoSQL surgiram da necessidade e armazenamento e processamento de um cada vez maior volume de dados, e por isso são construídos objetivando um desempenho melhor, em aplicações específicas, do que de bancos tradicionais.
- **Alta escalabilidade**: Bancos relacionais podem ser escalados verticalmente com a utilização de equipamentos poderosos e caros, e uma operação distribuída costuma ser mais complexa devido à forma de armazenamento de seus dados [25]. Bancos NoSQL foram pensados para execução em um sistema de *clusters*, o que facilita

a sua escalabilidade horizontal e reduz a necessidade de um hardware mais caro e específico, podendo ser utilizado em *hardwares* mais simples.

- **Alta disponibilidade**: Devido à possibilidade de escalabilidade horizontal, bancos NoSQL podem distribuir sua operação em diversos nós de um *cluster*, o que possibilita acesso simultâneo por um grande número de usuários, mesmo que não seja possível acessar algum desses nós.

- ***Open source***: SGBDs tradicionais costumam possuir licenças pagas, gerando um custo financeiro alto, principalmente quando executados em múltiplas maquinas [28]. NoSQLs costumam seguir licenças *open source*, podendo reduzir significativamente os gastos da aplicação.

3.2.2 Teorema CAP

Em 2000 Eric Brewer, pesquisador na *University of California*, propôs o teorema CAP, que define limitações em sistemas distribuidos. O teorema define que podemos garantir somente duas das seguintes três propriedades em um determinado sistema: Consistência (*Consistency*) , Disponibilidade (*Availability*) e Tolerância a partições (*Partition-resilience*) [17]. Essas propriedades podem ser definidas como:

- **Consistência** define que todos os nós possuem os mesmos dados em qualquer dado instante, e um pedido de leitura em qualquer desses nodos garante o dado mais atual possível do sistema.

- **Disponibilidade** garante é sempre possível ler e gravar dados em um nodo dado que ele esta acessível.

- **Tolerância a partições** garante que o sistema ira continuar funcionando mesmo na hipótese de eventuais falhas de comunicação entre os nodos.

Essas propriedades podem ser agrupadas da seguinte forma e obtendo os seguintes resultados:

- **CA** são sistemas distribuídos cujos nós estejam em uma mesma partição de rede.

- **CP** são sistemas que, em caso de falha em pelo menos um dos nós, ficam indisponíveis até sua total recuperação. Bancos de Dados que seguem as propriedades ACID costumam seguir esse padrão.

- **AP** são sistemas que devem permanecer em funcionamento mesmo durante uma eventual falha em um ou mais de seus nós, mesmo que isso resulte em dados não atualizados durante consultas. Sistemas NoSQL costumam seguir esse padrão.

Sistemas distribuídos, entretanto, por estarem sempre sujeitos a falhas de rede [15], não podem ignorar a Tolerância a Falhas, tendo de fazer uma escolha entre Consistência e Disponibilidade [10].

A figura 3.1 ilustra as diferentes combinações das propriedades CAP e exemplos de bancos de dados que as utilizam.

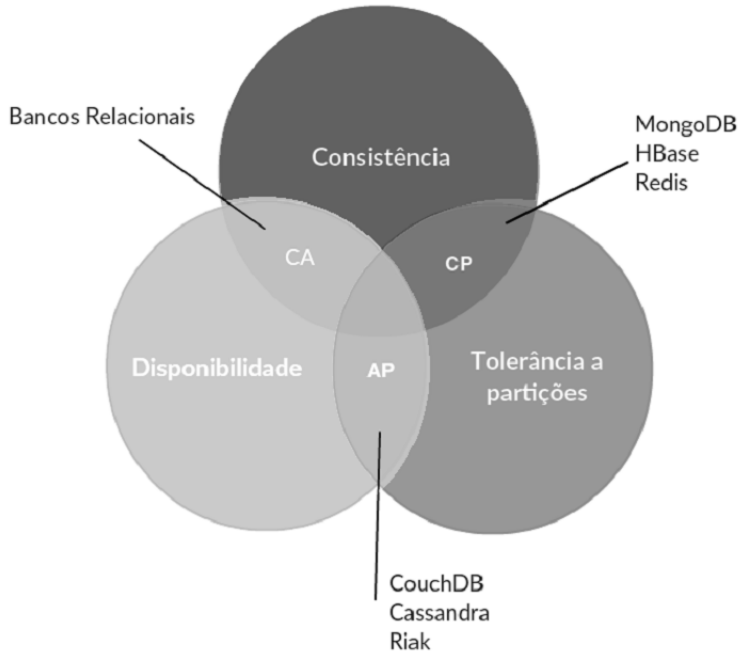

Figure 3.1: Propriedades CAP e exemplos. Adaptado de [8]

3.2.3 BASE

Em um ambiente distribuído, escalabilidade, resiliência e velocidade são mais importantes do que consistência imediata e segurança quanto à veracidade dos dados, não sendo necessária a aderência total às propriedades ACID já citadas [6]. Além disso, de acordo com o **Teorema CAP**, um banco que aceite particionamento não pode possuir alta disponibilidade e consistência simultaneamente. Essa necessidade, tanto de performance quanto de disponibilidade, levou à criação do acrônimo **BASE**, ***B****asically* ***A****vailable* (Basicamente Disponível), ***S****oft State* (Estado Leve) e ***E****ventual Consistency* (Consistência Eventual) [18].

Enquanto um banco **ACID** é pessimista, requerendo que cada operação mantenha a consistência do banco como um todo, **BASE** segue uma visão otimista, entendendo que dados serão eventualmente consistentes.

Sistemas distribuídos costumam manter cópias de dados em varias maquinas em um *cluster* para aumentar a sua disponibilidade, e quando um desses dados é atualizado em uma dessas maquinas é natural que haja um intervalo de tempo até que todas essas cópias sejam atualizadas.

3.2.4 Modelos NoSQL

Bancos de dados NoSQL possuem padrões de modelos de dados, que compartilham certas características em comum e servem a determinadas aplicações específicas, podendo alguns bancos serem classificados em mais de uma categoria. A tabela 3.1 lista os quatro modelos atuais e alguns bancos de dados que se enquadram em cada um deles.

Table 3.1: Modelos de Bancos NoSQL

Modelo de Dados	**Exemplo de bancos de dados**
Chave-Valor	Project Voldemort
	Riak
	Redis
	BerkeleyDB
Documentos	CouchDB
	MongoDB
	OrientDB
Famílias de colunas	Cassandra
	Hypertable
	HBase
Grafos	Neo4j
	OrientDB
	Infinite Graph

Chave-Valor

Bancos com armazenamento em chave-valor existem a muito tempo, como *Berkeley DB*, mas ganharam importância no meio NoSQL a partir do Amazon DynamoDB e do Google BigTable [29].

Consiste basicamente em uma tabela *hash*, sendo o acesso aos dados realizado por meio de uma chave primaria, assim como ocorre em *maps* e dicionários. Esses bancos são completamente livres de esquema e suas operações se resumem a consultar o valor a partir de uma chave, inserir um valor para uma chave ou deletar uma chave e seu valor do banco [21]. O valor armazenado em geral pode representar qualquer tipo de objeto, como uma *string* ou um *BLOB*, não sendo necessária que exista qualquer relação entre diferentes registros, ficando a aplicação responsável pelo seu tratamento.

Esses bancos favorecem escalabilidade sobre consistência, e por isso em geral não possuem ferramentas mais poderosas de consulta e analise de dados [29].

Atualmente temos como exemplos de bancos chave-valor: *Riak*, *Redis*, *Berkeley DB* e *Project Voldemort*.

Documentos

Bancos orientados a documentos armazenam seus dados em forma de documentos, podendo esses terem formato *XML*, *JSON*, *BSON*, etc [28]. Podem ser vistos como a sequência natural do armazenamento por chave-valor, ainda fazendo o armazenamento por meio de um par chave-valor, mas utilizando uma estrutura mais rica para armazenamento dos dados ao armazenar um documento na parte do valor [29]. Cada um desses documentos pode ter certa semelhança uns com os outros, mas não necessitam possuir a mesma estrutura, o que permite uma grande flexibilidade no esquema do banco.

A seguir temos um exemplo de dois documentos. Apesar de parecidos, eles possuem certas diferenças, o que gera essa grande flexibilidade do modelo orientado a documentos.

{

```
  "clienteid" : "f6a6fs86fa",
  "cliente" :
  {
    "primeironome" : "Pedro",
    "sobrenome" : "Silva",
    "gosta" : ["Leitura", "Viagem"]
  }
  "endereco" :
  {
    "estado" : "Sao Paulo",
    "cidade" : "Guarulhos"
  }
}

{
  "clienteid" : "ga9s8g8fe",
  "cliente" :
  {
    "primeironome" : "Maria",
    "sobrenome" : "Costa",
    "gosta" : ["Esportes"]
  }
  "ultimaCompra" : "12/11/2015"
}
```

Esses documentos não são opacos à aplicação, seu conteúdo pode ser consultado diretamente, com consultas diretas em atributos de seus registros. Isso permite a manipulação de estruturas mais complexas, que ainda assim não possuem nenhuma restrição de esquema, sendo fácil a inserção de novos documentos ou a modificação dos documentos já armazenados. Devido à essa flexibilidade, são recomendados para integração de dados e migração de esquemas [21].

Como exemplos de bancos orientados a documentos podemos citar o *CouchDB*, *MongoDB* e *OrientDB*.

Colunas

Bancos de Dados colunares tem sua influência no *Google BigTable* [11], e armazenam seus dados em famílias de colunas que são associadas a uma chave de linha. Cada uma dessas famílias de colunas pode possuir varias colunas, e são consideradas dados relacionados que podem ser acessados ao mesmo tempo [28].

O Cassandra possui ainda o conceito de super colunas, que pode ser visto como um agrupamento de colunas que pode ser armazenado dentro de uma família de colunas [28].

Colunas e linhas podem ser adicionadas a qualquer momento, o que gera uma flexibilidade bem maior em relação aos esquemas em geral fixos dos bancos de dados relacionais. Entretanto, famílias de colunas em geral devem ser predefinidas, situação menos flexível que a encontrada nos modelos de chave-valor ou de documentos [21].

Como exemplo de bancos orientados a colunas temos o *HBase* e *Hypertable*, que são implementações *open source* do BigTable, e o *Cassandra*.

Nesse trabalho será utilizado o banco orientado a colunas Cassandra, e suas características serão melhor abordadas no capítulo 4.

Grafos

Diferente dos bancos relacionais e dos já citados modelos NoSQL vistos, um banco de dados em grafos é especializado em dados altamente conectados. São ideias para aplicações que realizam consultas baseadas em relações [21]. Esse modelo realiza o armazenamento por meio de entidades e os relacionamentos entre essas entidades. Entidades podem ser vistas como nós e os relacionamentos como as áreas de um grafo [28]. Esses nós podem possuir propriedades dos objetos que representam, assim como as áreas, que podem possuir atributos do relacionamento e além disso possuem significância em sua direção.

Consultas nesse tipo de modelo são realizadas percorrendo o grafo. Isso possui como vantagem a possibilidade de se modificar a forma que se caminha nesse grafo, não sendo necessárias mudanças em sua estrutura de nós e relações [28].

Uma diferença importante dos bancos orientados a grafos em relação aos modelos anteriores é o seu suporte menor a sistemas distribuídos, não sendo geralmente possível a distribuição dos nós em diferentes servidores [28].

A figura 3.2 ilustra um banco de dados orientado a grafos implementado com a utilização do *Neo4J*.

Como exemplos desse modelo podemos citar o *Neo4J*, o *Infinite Graph* e o *OrientDB*.

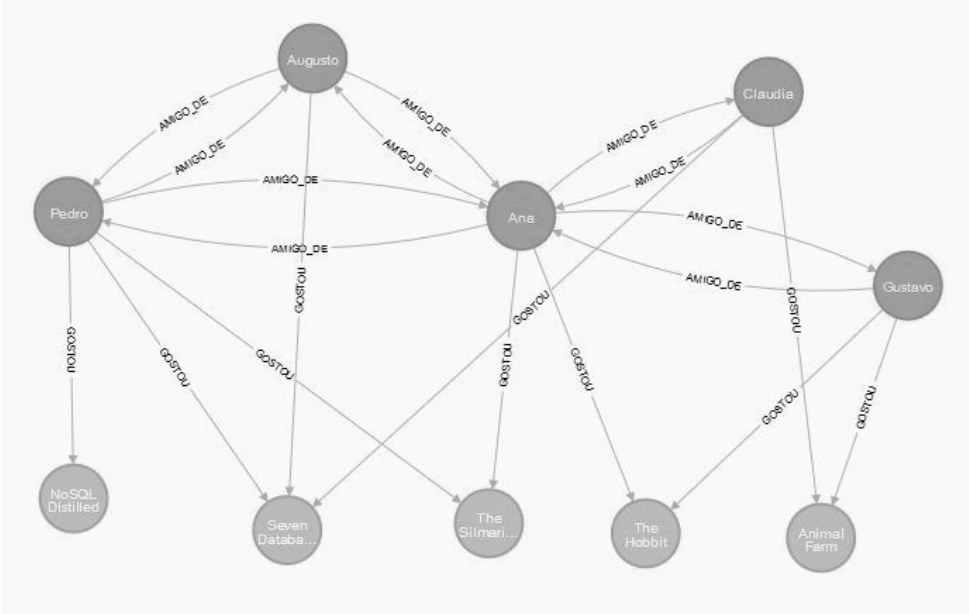

Figure 3.2: Visualização de um banco de dados em grafo

Chapter 4

Cassandra

Esse trabalho ira utilizar o Cassandra como banco de dados para validação de sua hipótese. Como visto no capítulo 3, o Apache Cassandra, distribuição que sera utilizada, é um banco de dados orientado a colunas altamente disponível e distribuído em servidores constituídos de hardware de "prateleira" para gerenciamento de grande volumes de dados [24]. Este capítulo tem como objetivo definir esse banco de dados, suas características, funcionamento, vantagens e desvantagens.

4.1 Definição

O Cassandra se originou em 2007 como um projeto do *Facebook* para resolver um problema na busca da caixa de mensagens. A companhia necessitava de um sistema com alta performance, confiabilidade, eficiência e que suportasse o contínuo crescimento da ferramenta [24, 22].

O projeto foi desenvolvido por Jeff Hammerbacher, Avinash Lakshman, Karthik Ranganathan e Prashant Malik, tendo seu modelo de dados sofrido grande inspiração nos trabalhos anteriores do *Amazon Dynamo* [14] e do *Google Bigtable* [11], e lançado em 2008 como um projeto *open source*. Foi mantido e atualizado apenas pelo Facebook até 2009, quando foi comprado pela Apache [22], sendo utilizado atualmente por companhias como *Netflix*, *Spotify* e até em agências governamentais, como a NASA [2].

O Apache Cassandra pode ser definido como um banco de dados orientado a colunas *open source*, distribuído, descentralizado, elasticamente escalável, altamente disponível, tolerante a falhas e variavelmente consistente [22]. A seguir iremos analisar cada uma dessas características.

4.1.1 Características

Distribuído e Descentralizado

O Cassandra é capaz de ser executado em múltiplas de forma transparente ao usuário, que o enxerga como um sistema unificado. Apesar de ser possível sua execução em um único nó, só é possível obter algum benefício com uma execução distribuída. Além do ganho de performance, a distributividade do sistema garante maior segurança devido à redundância de dados.

Diferente de outros bancos distribuídos que elegem nós como mestres e escravos, o Cassandra opera de forma descentralizada, o que significa que todos os nós são idênticos em sua forma de execução, sendo utilizados protocolos *peer-to-peer* (par-a-par) e *gossip* para manutenção e sincronia entre os nós. Essa descentralização garante que não exista apenas um ponto de falha, o que aumenta sua disponibilidade, e simplifica a operação do e manutenção do *cluster*.

Elasticamente Escalável

Escalabilidade é a propriedade que um sistema tem de atender um crescente número de requisições sem prejuízo de performance. Essa escalabilidade pode ser tanto vertical quanto horizontal. Na escalabilidade vertical o hardware já utilizado no sistema é melhorado, enquanto na escalabilidade horizontal novos máquinas são adicionadas à arquitetura, havendo a divisão da carga do sistema.

O Cassandra possui escalabilidade horizontal elástica, o que significa que sua arquitetura pode escalar tanto para cima quanto para baixo. Na necessidade de uma melhora do desempenho da aplicação, novas máquinas podem ser adicionadas, e o Cassandra se encarrega de fazer a distribuição dos dados de forma transparente, sem necessidade de configurações adicionais ou reiniciamento do sistema. Da mesma forma, em caso de necessidade, máquinas podem ser retiradas do *cluster* sem prejuízo ao todo, devido ao rebalanceamento automático.

Altamente disponível e Tolerante a falhas

A disponibilidade de um sistema é medida de acordo com sua capacidade de responder a requisições. Computadores, e especialmente sistemas distribuídos em rede, estão sujeitos a falhas, que em geral só podem ser contornadas por meio de sistemas redundantes.

Devido a replicação e redundância de dados e a sua capacidade de substituição de nós indisponíveis, o Cassandra pode ser definido como um sistema altamente disponível e tolerante à falhas em suas máquinas.

Variavelmente Consistente

A consistência de uma aplicação diz respeito à sua capacidade de retornar o valor mais atual em uma requisição.

Como visto no Teorema CAP 3.2.2, não é possível a um sistema ser totalmente consistente, disponível e tolerante a falhas.

O Cassandra é por vezes definido como "eventualmente consistente", por trocar parte de sua consistência por alta disponibilidade. Essa definição, porém, não é totalmente correta, e um termo melhor para defini-lo é "variavelmente consistente" (*tuneably consistent*), podendo essa sua consistência ser ajustada de acordo com o tipo de aplicação.

4.2 Modelo de Dados

Um banco de dados Cassandra consiste em um *keyspace* contendo famílias de colunas, que por sua vez definem um conjunto de linhas que englobam várias colunas. Essa dis-

posição de dados é bastante semelhante ao que foi proposto pelo Bigtable [24, 11]. Seu modelo de dados pode ser visto como um mapa multidimensional indexado por uma chave, se assemelhando aos modelos de chave-valor e orientados à colunas. A seguir veremos em detalhes cada um desses conceitos.

A figura 4.1 ilustra de forma resumida o modelo de dados do Cassandra, contendo um *keyspace* e seus componentes.

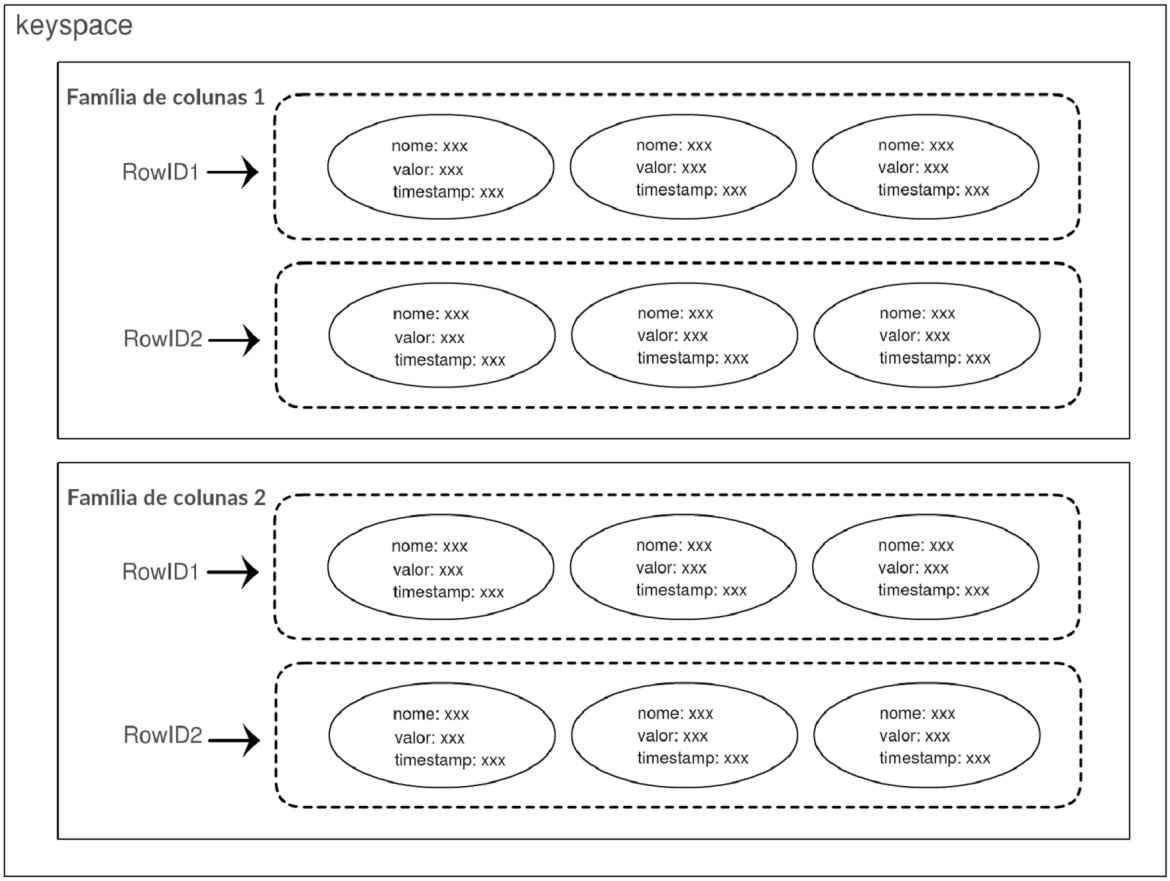

Figure 4.1: Modelo de dados do Cassandra. Adaptado de [3]

Keyspace

Um *keyspace* define o agrupamento de dados mais externo no Cassandra, podendo ser correspondido a um banco de um SGBD relacional. Um *keyspace* define um nome e uma série de atributos que definem o seu comportamento. Atributos do *keyspace* incluem [22]:

- **Fator de replicação** diz respeito ao número de nós que armazenarão uma réplica de cada linha de dados. O fator de replicação tem forte influencia no balanço entre performance e consistência do banco de dados.
- **Estratégia de replicação** se refere a como as réplicas (ou cópias) de um dado serão posicionados no anel do *cluster*. Diversas estratégias de replicação estão disponíveis no Cassandra.

- **Famílias de colunas** pode ser visto como o análogo às tabelas de um modelo relacional, da mesma forma que o *keyspace* é o análogo do banco. Uma família de colunas é um agrupamento para uma coleção de linhas, onde cada linha contém colunas ordenadas.

Colunas e famílias de colunas

Uma família de colunas (ou tabela) no Cassandra é um mapa multidimensional indexado por uma chave. Essa chave é uma *string* sem restrição de tamanho, mas que em geral varia de 16 a 36 *bytes*. O valor desse mapeamento consiste em uma família de colunas, um agrupamento para uma coleção ordenadas de linhas, que por sua vez é uma coleção ordenada de colunas [24, 22].

Uma família de colunas possui dois atributos: um nome e um comparador. O nome identifica a coluna para realização de consultas, enquanto o comparador indica como as colunas serão ordenadas ao serem retornadas em uma consultas, podendo ser *long*, *byte*, UTF8, etc [22].

O modelo de família de colunas se diferencia do modelo relacional por ser o que é chamado comumente de *livre de esquema* (*schema free*) . É possível realizar a inserção, remoção ou alteração de qualquer coluna ou família de colunas a qualquer momento, ficando as aplicações clientes do banco encarregadas de interpretar e manipular o novo modelo de dados.

Ao se inserir um novo dado em uma família de colunas do Cassandra são especificados valores para uma ou mais colunas. O conjunto de valores é chamado de linha, e é identificado unicamente por uma chave primária ou chave de linha. Uma linha não precisa possuir dados para todas colunas presentes na família de colunas à que ela pertence, sendo o espaço alocado apenas para as colunas presentes nessa linha. Isso gera tanto uma economia de espaço quanto uma melhora de performance em relação a um banco relacional, que precisa preencher com valores nulos colunas não utilizadas.

Uma coluna é a unidade básica de armazenamento do Cassandra, e é constituída por um nome, um valor e um *timestamp*. Se difere do conceito de colunas de bancos relacionais pois durante a criação do banco não é necessário a criação de colunas, e sim apenas famílias de colunas. A criação de colunas ocorre apenas durante a inserção de dados no banco e suas colunas correspondentes [22].

4.3 Arquitetura

O Cassandra foi projetado para lidar com grandes massas de dados distribuídas em vários nós sem ponto único de falha, pensado no fato de que tanto um sistema quanto componentes de hardware podem falhar. Ao contrário de outras soluções de bancos de dados distribuídos, sejam elas relacionais ou modelos mais novos como o *Google Bigtable*, em que os nós são definidos como mestres e escravos (*master* e *slave*), a arquitetura Cassandra combate o problema de falhas ao empregar uma distribuição par-a-par (*peer-to-peer*) entre nós estruturalmente idênticos, com dados distribuídos entre todos os nós de um *cluster* [4, 22].

Essa decisão arquitetural de nós atuando de maneira par-a-par tem como objetivo melhorar a disponibilidade e facilidade de escalabilidade do sistema. Na ocasião de um nó

ficar indisponível, existe um potencial impacto na vazão de dados do sistema, entretanto isso não causa uma interrupção no serviço. Ao mesmo tempo, ao se inserir um novo nó no sistema, é necessário que ele recebe informações sobre a topologia do anel em que está inserido e dados que ele será responsável. Após isso, entretanto, ele pode integrar o anel e receber requisições assim como os outros nós, sem necessidade de configurações complexas [22].

O Cassandra é um banco de dados particionado em linhas, com cada linha organizada em tabelas com uma chave primária obrigatória. Sua arquitetura permite que qualquer usuário autorizado se conecte a qualquer nó em qualquer *data center* e acesse os dados por meio da linguagem *CQL*, que possui sintaxe próxima a do *SQL*, com abstração de uma tabela com linhas e colunas [4].

Requisições de leitura e escrita podem ser enviadas a qualquer nó do *cluster*, devido à característica de homogeneidade entre eles. Ao realizar uma conexão com um cliente, o nó atua como coordenador dessa operação, servindo de ponte entre a aplicação e os nós que possuem o dado requisitado. O coordenador também determina quais nós devem receber a requisição de acordo com a configuração do *cluster* [4].

4.3.1 Protocolo Gossip

Cada nó frequentemente troca informações de estado sobre ele mesmo e outros nós do cluster utilizando um protocolo *gossip*. *Gossip* é um protocolo de comunicação par-a-par em que os nós realizam troca de informações periódicas sobre o estados eles mesmos e sobre outros nós do *cluster* de que eles tem conhecimento. Seu nome vem do conceito humano de "fofoca" (*gossip*), uma forma de comunicação em que cada par pode escolher com quem ele deseja trocar informações [4, 22].

Protocolos *gossip* em geral assumem uma rede falha, e são em geral utilizados em sistemas em rede grandes e descentralizados, em geral em mecanismos de replicação em bancos de dados [22].

O processo *gossip* executa a cada segundo, e troca mensagens de estado com até três outros nós no *cluster*. Uma mensagem *gossip* tem uma versão associada a ela, de forma que durante sua troca, informações obsoletas são reescritas com o estado mais atual do nó em questão.

Para evitar problemas na comunicação *gossip*, deve-se utilizar uma mesma lista de nós *seed* para todos os nós do *cluster*. Um nó *seed* é aquele que mantem informações sobre todos os nós da rede. Por padrão, um nó lembra outros nós com quem ele tenha trocada informações entre reinícios do sistema, o nó *seed* tem como única função inicializar o processo *gossip* para novos nós que estejam sendo adicionados ao *cluster* [4].

4.3.2 Operações de Leitura e Escrita

Ao se realizar uma operação de escrita em um nó, a informação é armazenada imediatamente em um *commit log*. O *commit log* é um mecanismo de recuperação de falhas que garante a durabilidade de um banco Cassandra. Uma operação de escrita só terá sucesso se for escrita no *commit log*, garantindo que mesmo que essa operação não seja armazenada em memória, seja possível recuperar seus dados [22].

Os dados do *commit log* são então indexados e escritos em uma estrutura em memória RAM chamada *memtable*. Sempre que os objetos armazenados na *memtable* alcançam um limiar definido, seu conteúdo é carregado em disco em um arquivo *SSTable* e uma nova *memtable* é criada [22, 4].

O conceito da *SSTable* surgiu com o *Google Bittable*. Assim que os dados oriundos da *memtable* são armazenados na *SSTable* eles se tornam imutáveis, não podendo ser modificados pela aplicação. Essa operação de escrita é realizada no final do arquivo (*append*), não sendo necessário leituras ou buscas, razão pela qual o Cassandra é recomendável para sistemas com maior demanda para escritas [22, 4].

Os arquivos *SSTable* são então particionados e replicados no cluster, o que garante a redundância dos dados. Periodicamente, as *SSTables* são compactadas e dados obsoletos são descartados, além disso a consistencia do *cluster* é mantida por meio de mecanismos de reparo [22, 4]. As operações descritas estão esquematizadas na figura 4.2.

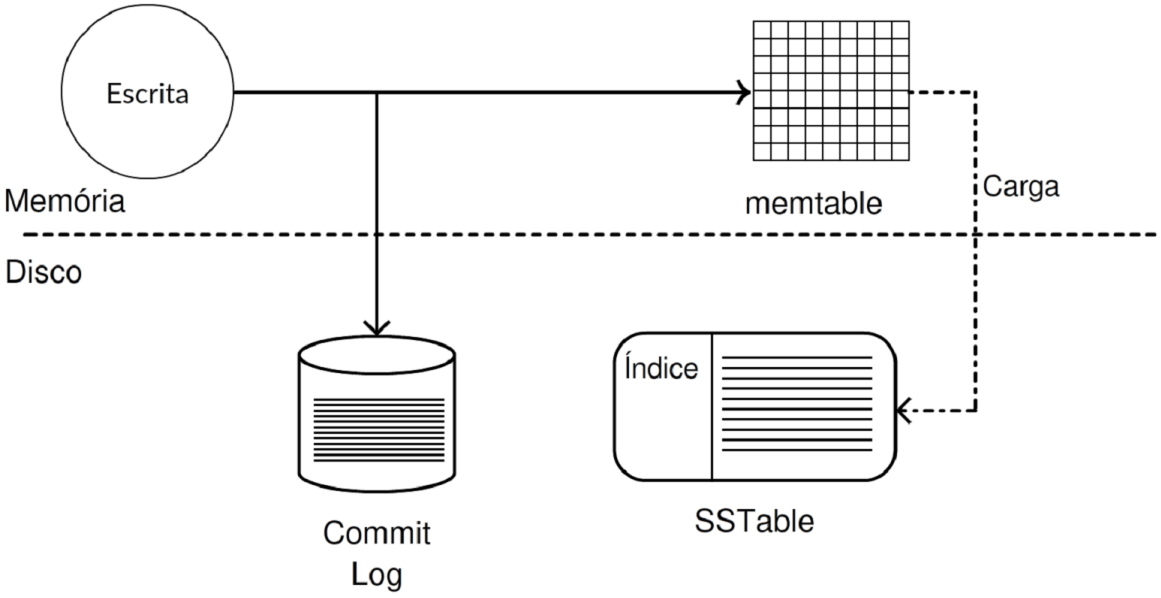

Figure 4.2: Escrita de dados em um banco Cassandra. Adaptado de [4]

Ao se realizar uma operação de leitura o Cassandra inicialmente busca a informação solicitada na *memtable*. Caso não seja encontrada, é realizada uma busca na *cache* de linha, uma estrutura em memória que armazena em memória um subconjunto da partição de dados armazenado nas *SSTable*. Se a *cache* de linha não possuir o dado em específico, é realizada uma consulta no *Bloom Filter* associado à *SSTable* em questão [4].

Um *Bloom Filter* é um algoritmo de busca não determinístico que testa de forma rápida se um elemento faz parte de um conjunto. É dito não determinístico pois é possível obter falso positivos em uma consulta, mas não falso negativos. Em outras palavras, um *Bloom Filter* garante que um dado não está presente no conjunto, mas se ele indicar que o dado está presente uma consulta deve ser realizada no conjunto para se obter uma confirmação [22].

Caso o *Bloom Filter* indique que o dado pode estar presente em uma *SSTable* são realizadas algumas operações de busca de índice para encontrar sua localização, e o dado é então retornado na consulta [4].

4.3.3 Distribuição de Dados

Distribuição e replicação de dados no Cassandra são conceitos que se conectam. Devido ao fato de um *cluster* Cassandra ser uma rede sem hierarquia entre nós, o sistema faz cópias (ou réplicas) dos dados e as distribui entre os nós. Esses dados são organizados por tabelas identificadas por chaves primárias, que determinam em que nó desse *cluster* esse dado será armazenado [4].

O Cassandra utiliza um sistema de *hashing* consistente, que permite a distribuição de dados no *cluster* de forma a minimizar a reorganização do mesmo quando nós são inseridos ou removidos. O *hashing* consistente particiona os dados baseados em uma chave de partição (*partition key*), que é obtida do primeiro componente da chave primária de cada tabela. Dessa forma, cada nó em um *cluster* é responsável por um intervalo de dados baseado nesse mapeamento [4].

A partir de sua versão 1.2 o Cassandra permite que a cada nó físico seja atribuído mais de um *token*, que determina a posição desse nó no anel e a porção de dados de que ele é responsável. Esse novo paradigma de distribuição foi chamado de nós virtuais (*virtual nodes* ou *vnodes*). Nós virtuais permitem que cada nó possua um grande número de pequenos intervalos de partições distribuídos pelo *cluster* [4].

A figura 4.3 mostra a distribuição de nós em um anel, comparando uma arquitetura sem nós virtuais com uma utilizando nós virtuais.

Na primeira parte da figura a cada nó é atribuído um único *token* que representa sua posição no anel. Cada nó armazena dados determinados pelo mapeamento da chave de partição para um valor no intervalo entre o nó anterior e o valor do *token*. Cada nó também possui réplicas de cada linha de outros nós do *cluster*.

Na segunda parte da figura, nós virtuais são selecionados aleatoriamente e de forma não contígua. A posição de uma linha é determinada pelo mapeamento da chave de partição dentro de vários pequenos intervalos de partição pertencentes a cada nó [4].

Replicação de Dados

O Cassandra armazena réplicas de dados em múltiplos nós para garantir confiabilidade e tolerância a falhas. A estratégia de replicação determina em que nós as cópias serão armazenadas. O número de cópias em um *cluster* é chamado fator de replicação. Se um fator de replicação um é utilizado, existe apenas uma copia de cada linha no *cluster*, ou seja, se o nó contendo essa linha cai, a linha não pode ser acessada. Todas as réplicas são igualmente importantes, não existe uma réplica mestre ou primária. Como regra geral, o fator de replicação não deve exceder o número de nós no *cluster*. É possível, porém, aumentar o fator de replicação e inserir o número desejado de nós posteriormente [4].

Existem duas estratégias de replicação:

- ***SimpleStrategy*** é utilizada para um único *data center* e um único *rack*. A primeira réplica de um nó é determinada pelo particionador, e réplicas adicionais são posicionadas nos próximos nós no anel em sentido horário, sem se considerar a topologia do sistema;
- ***NetworkTopologyStrategy*** deve ser utilizada quando se tem, ou se planeja ter, um *cluster* distribuído em vários *data centers*. Essa estratégia especifica quantas réplicas se deseja em cada *data center*. As réplicas são inseridas em um mesmo *data*

center caminhando-se no anel em sentido horário até se alcançar o primeiro nó em outro *rack*. A *NetworkTopologyStrategy* tenta colocar réplicas em *racks* distintos, por entender que nós em um mesmo *rack* frequentemente falham ao mesmo tempo por problemas de energia, resfriamento ou falhas na rede.

Estratégias de replicação são definidas por *keyspace* e definidas durante sua criação [4].

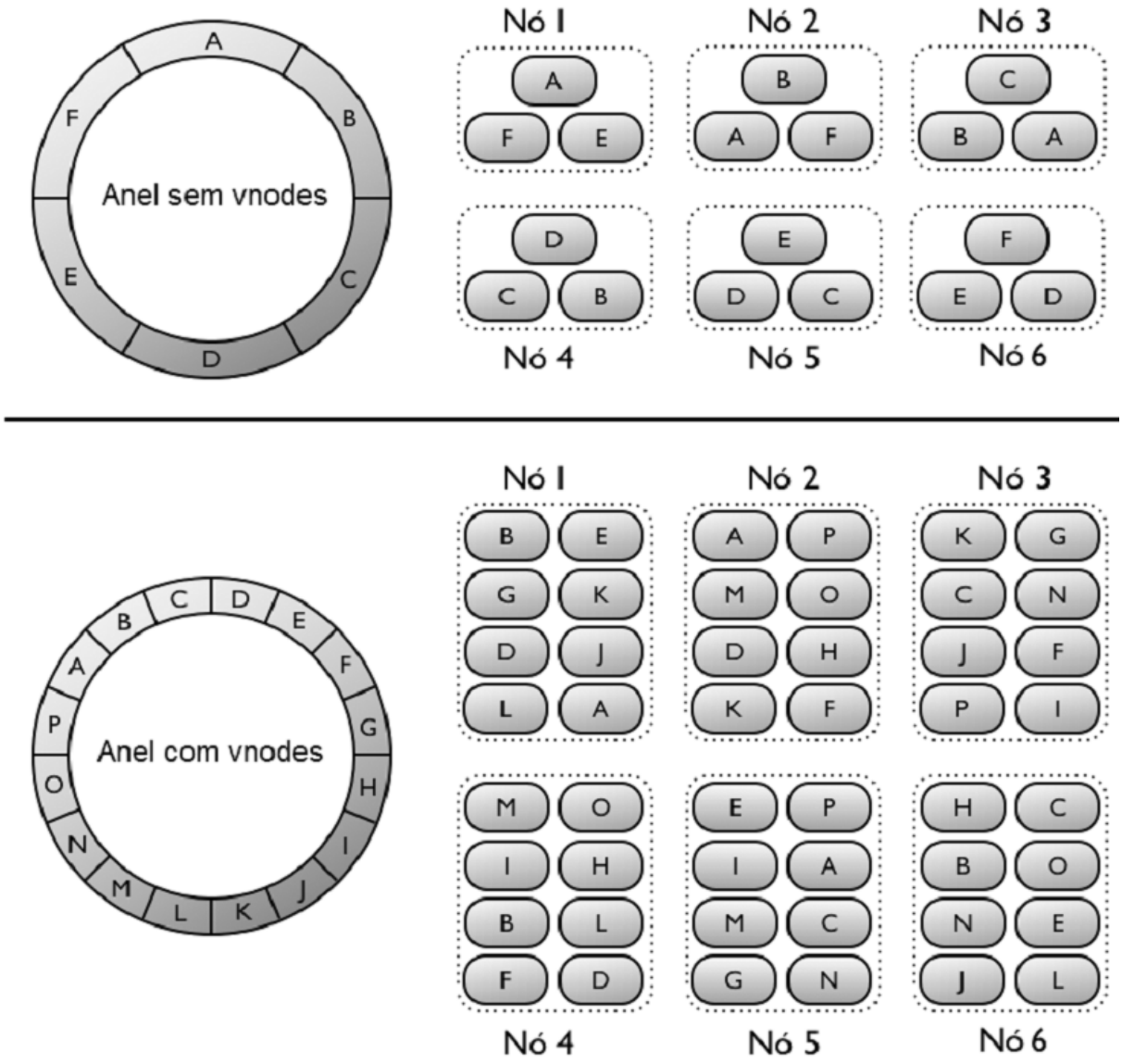

Figure 4.3: Distribuição de nós em um anel. Adaptado de [4]

Particionadores

Um particionador determina qual nó irá receber a primeira réplica de um dado e como distribuir outras réplicas entre nós de um *cluster*. Cada linha de dados é unicamente identificada por uma chave primária, que pode ser a mesma que sua chave de partição, mas que também pode incluir outras colunas. Basicamente um particionador é uma

função que deriva um *token* a partir de uma chave de primária de uma linha, geralmente por meio de um mapeamento (*hash*). Cada linha de dados é então distribuída no *cluster* de acordo com o valor desse *token*.

O Cassandra fornece três particionadores, que podem ser definidos no arquivo de configuração principal do Cassandra, o *cassandra.yaml*. São ele:

- ***Murmur3Partitioner*** distribui os dados de forma uniforme no *cluster*, de acordo com valores da função *hash MurmurHash*, que cria valores de 64 bits com a chave de partição. É o particionador padrão e o recomendável para a maioria dos novos *clusters*, e fornece um desempenho melhor que o do *RandomPartitioner*;
- ***RandomPartitioner*** também distribui os dados uniformemente no *cluster*, porém com a utilização de uma função *hash* MD5. Essa é uma função criptográfica com um tempo de execução mais longo que a *MurmurHash*. Como o Cassandra não exige um *hash* criptográfico, o particionador *RandomPartioner* reduz o desempenho do Cassandra de 3 a 5 vezes em comparação ao *MurmurPartitioner*;
- ***ByteOrderedPartitioner*** distribui os nós no *cluster* em ordem alfabética. É utilizado quando se deseja um particionamento ordenado, porém causa problemas de dificuldade de balanceamento de carga, pontos quentes em escritas sequenciais e balanceamento desigual de carga para múltiplas tabelas. É incluído no Cassandra por razões de retrocompatibilidade.

Tanto o *MurmurPartitioner* quanto o *RandomPartitioner* utilizam to*tokens* para auxiliar na designação de porções iguais de dados para cada nó e na distribuição de forma uniforme desses dados a partir de todos as tabelas do anel ou de outros agrupamentos, como um *keyspace*.

Ao se selecionar a configuração de quantas réplicas serão armazenadas em cada *data center*, as duas considerações primárias são:

1. ser capaz de satisfazer leituras locais sem incorrer em latência entre *data centers*;
2. cenários de falha.

Os dois modos mais comuns para configurar *clusters* em múltiplos *data centers* são [4]:

- Duas réplicas por *data center*: essa configuração tolera falhas em nós únicos por grupo de replicação, permitindo ainda leituras locais em um nível de consistência *ONE*.
- Três réplicas por *data center*: essa configuração tolera tanto a falha de um nó por grupo de replicação em um nível de consistência $LOCAL\_QUORUM$ quanto falhas em múltiplos nós com nível de consistência *ONE*

Grupos de replicação assimétricos também são possíveis, com cada *data center* adotando a configuração preferível. Já estratégias de replicação são definidas por *keyspace* e definidas durante sua criação [4].

Níveis de consistência

Consistência se a quão atuais e sincronizadas todas as réplicas de uma linha de dados do Cassandra está em um dado momento. Operações de reparo automático do Cassandra garantem que todas as réplicas serão eventualmente serão consistentes, porém o tráfego constante de dados entre um sistema largamente distribuído pode levar à inconsistências em algum momento [4].

Níveis de consistência podem ser configurados para gerenciar o balanço entre disponibilidade de dados e acurácia dos dados. Essa configuração pode se dar dentro de um *cluster*, de um *data center* ou mesmo em operações individuais de leitura e escrita [4].

Um conceito importante ao se considerar níveis de consistência é o **quorum**. O nível de $QUORUM$ determina o numero de nós que forma um quorum e pode ser calculado pela equação 4.1 [4].

$$quorum = (\frac{SomaDosFatoresDeReplicacao}{2}) + 1 \tag{4.1}$$

Consistência de escrita

A seguir estão listados alguns dos diferentes níveis de consistência de escrita, do mais forte ao mais fraco, sendo o nível *ONE* o nível padrão de consistência [4]:

- ***ALL***: Uma escrita deve ser feita no *commit log* e na *memtable* em todos os nós réplicas em um *cluster* para aquela partição. Fornece a mais alta consistência e mais baixa disponibilidade entre os todos os níveis;
- ***QUORUM***: Uma escrita deve ser feita no *commit log* e na *memtable* em um quorum de réplicas de nós entre todos os *data center*. Mantém forte consistência porém apresenta certo nível de falha;
- ***ONE***: Uma escrita deve ser feita no *commit log* e na *memtable* de pelo menos um nó. Satisfaz as necessidades da maioria dos usuários, pois não possui exigências rígidas.
- ***ANY***: Garante baixa latência e permite que uma escrita nunca falhe. Fornece a mais alta disponibilidade e a mais baixa consistência

Consistência de leitura

A seguir estão listados alguns dos diferentes níveis de consistência de leitura, do mais forte ao mais fraco, sendo o nível *ONE* o nível padrão de consistência [4]:

- ***ALL***: Retorna um registro depois que todas as réplicas tenham respondido. A operação irá falhar se uma réplica não responder. Fornece a mais alta consistência e mais baixa disponibilidade entre os todos os níveis;
- ***QUORUM***: Retorna um registro assim que um quorum de réplicas de todos os *data centers* tenham respondido. É utilizado tanto em únicos quanto múltiplos *data centers*, para manter uma forte consistência dentro do *cluster*, desde que seja tolerável certo nível de falha.

- ***ONE***: Retorna uma reposta da réplica mais próxima. Garante a mais alta disponibilidade, desde que seja tolerável uma relativamente alta probabilidade de que um dado antigo seja lido, já que a réplica contatada nem sempre tera a versão mais atual da informação.

Referências

[1] The annotated 8 principles of open government data. https://opengovdata.org/. Acessado em 04 de junho de 2016. 3

[2] Companies using nosql apache cassandra. http://www.planetcassandra.org/companies/. Acessado em 30 de maio de 2016. 13

[3] Considerações sobre o banco de dados apache cassandra. http://www.ibm.com/developerworks/br/library/os-apache-cassandra/. Acessado em 02 de junho de 2016. vi, 15

[4] Datastax distribution of apache cassandra™ 3.x. http://docs.datastax.com/en/cassandra/3.x/cassandra/cassandraAbout.html. Acessado em 02 de junho de 2016. vi, 16, 17, 18, 19, 20, 21, 22

[5] Definição de conhecimento aberto. http://opendefinition.org/od/2.0/pt-br/. Acessado em 08 de abril de 2016. 1, 2

[6] Graph databases for beginners: Acid vs. base explained. http://neo4j.com/blog/acid-vs-base-consistency-models-explained/. Acessado em 05 de maio de 2016. 9

[7] Inda - infraestrutura nacional de dados abertos. http://www.governoeletronico.gov.br/acoes-e-projetos/Dados-Abertos/inda-infraestrutura-nacional-de-dados-abertos. Acessado em 10 de abril de 2016. 3

[8] Nosql systems. http://blog.imrashid.com/nosql-systems/. Acessado em 08 de maio de 2016. vi, 9

[9] O que são dados abertos? http://opendatahandbook.org/guide/pt_BR/what-is-open-data/. Acessado em 07 de maio de 2016. 2, 4

[10] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012. 8

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. pages 205–218, 2006. 7, 11, 13, 15

[12] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 5

[13] Tribunal de Contas da União. 5 motivos para abertura de dados na administração pública. http://portal3.tcu.gov.br/portal/pls/portal/docs/2689107.pdf, 2015. Acessado em 08 de maio de 2016. 3

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. 41(6):205–220, 2007. 7, 13

[15] Peter Deutsch. The eight fallacies of distributed computing. http://today.java.net/jag/Fallacies.html. Acessado em 28 de maio de 2016. 8

[16] David Eaves. The three laws of open government data. https://eaves.ca/2009/09/30/three-law-of-open-government-data/. Acessado em 04 de junho de 2016. 2

[17] Armando Fox and Eric Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 174–178. IEEE, 1999. 8

[18] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based scalable network services. 31(5), 1997. 6, 9

[19] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983. 6

[20] Jan L. Harrington. *Relational database design and implementation : clearly explained.* Morgan Kaufmann, 2009. 5, 6

[21] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. pages 336–341, 2011. 10, 11, 12

[22] Eben Hewitt. *Cassandra: The Definitive Guide.* O'Reilly Media, 2010. 13, 15, 16, 17, 18

[23] Seiji Isotani and Ig I. Bittencourt. *Dados Abertos Conectados.* Novatec, 2015. 2, 3

[24] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010. 13, 15, 16

[25] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010. 1, 5, 7

[26] M Lynne Neufeld and Martha Cornog. Database history: From dinosaurs to compact discs. *Journal of the American society for information science*, 37(4):183, 1986. 5

[27] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems.* Osborne/McGraw-Hill, 3 edition, 2003. 6

[28] Pramod J. Sadalage and Martin Fowler. *NoSQL Essencial.* Novatec, 2013. 5, 7, 8, 10, 11, 12

[29] Christof Strauch. Nosql databases. *Stuttgart Media University*, 2011. 7, 10