**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. Flávio de Barros Vidal

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.
47 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. Flávio de Barros Vidal
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 [2], 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 [21]. 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*.

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.

# Chapter 2

# Dados Abertos

A *Open Definition* define um dado como aberto "se qualquer pessoa esta livre para acessa-lo, utiliza-lo, modifica-lo, e compartilha-lo — restrito, no maximo, a medidas que preservam a proveniência e abertura." [2]. A abertura dos dados evita mecanismos de controle e restrições sobre esses dados, o que permite seu uso de forma livre [19].

A Open Knowledge Foundation, organização mundial que promove a abertura de dados [6], resume esses pontos como:

- **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'.

O termo "Dados abertos" surgiu em 1995 em um documento de uma agência científica americana, e esse conceito vem sendo ampliado e estimulado nos últimos anos por diversos movimentos [19].

A participação brasileira tem um marco em 2011 com a criação da *Open Government Partnership*, contando inicialmente com a participação de 65 países. Também foi criado o portal dados.gov.br, que disponibiliza dados governamentais de forma aberta [19].

O poder público brasileiro vem nos últimos anos realizando outras ações que promovem a abertura de dados governamenais. Essas ações visam benefícios como melhoria da gestão pública, transparência, controle a participação social, geração de emprego e renda e estímulo à inovação tecnológica [10]. 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" [4].

O governo tem importancia 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 [6].

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 ja 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 funcionarios públicos [6].

# 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 ja na década de 60, principalmente em aplicações médicas e científicas [22], e em 1970, Edgar Codd propos uma nova forma de armazenamento de dados, que ficou conhecida como modelo de dados relacionais (*relational model of data*) [9].

## 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 necessario. 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 [21]. 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 [24]. Outra questão de importância em bancos de dados computacionais é a não necessidade que o usuario tem de conhecer como esses dados são armazenados, o que foi possível com o uso dos chamados Sistemas Gerenciadores de Bancos de Dados (SGBDs) [16].

Outras opções surgiram ao longo dos anos, como os bancos orientados a objetos ou bancos XML. Nenhum deles, entretanto, conseguiu competir com o modelo ja tradicional de dados relacionais. [24] 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 [16].

### 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 [23]. Para garantir que isso ocorra, um SGBD deve prover as seguintes propriedades, conhecidas como propriedades ACID [15]:

- **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 [14].

### 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 [16]. 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 [16]. 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 rapido crescimento no volume de dados nos últimos anos, principalmente após a bolha da Internet na década de 90 [24], tras uma necessidade de certa mudança em relação ao modelo tradicional. Modelos relacionais possuem diversas vantagem ja 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 [21].

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 [8], e Amazon's Dynamo em 2007 [11, 25].

### 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, distribuidos e tolerantes a falhas [24]. 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 [25]:

- **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 verticamente com a utilização de equipamentos poderosos e caros, e uma operação distribuida costuma ser mais complexa devido à forma de armazenamento de seus dados [21]. 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 [24]. 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*) [13]. 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 [12], não podem ignorar a Tolerância a Falhas, tendo de fazer uma escolha entre Consistência e Disponibilidade [7].

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

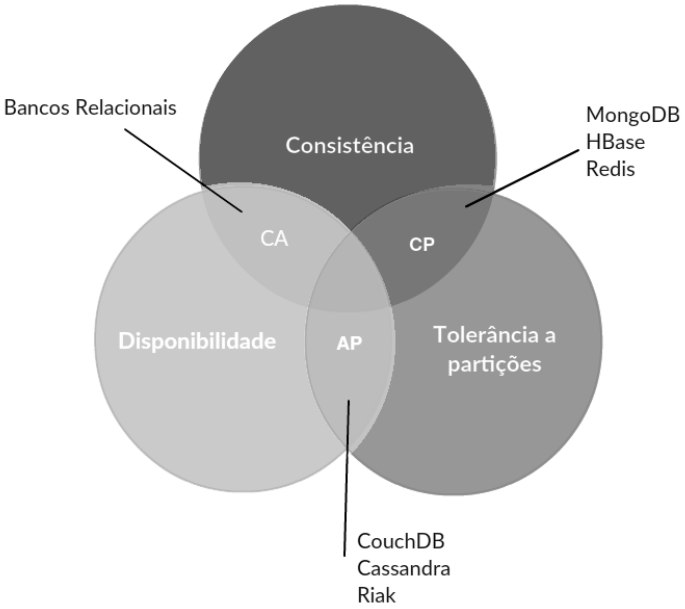


Figure 3.1: Propriedades CAP e exemplos. Adaptado de [5]

### 3.2.3 BASE

Em um ambiente distribuido, escabalibilidade, resiliência e velocidade são mais importantes do que consistência imediata e segurança quanto à veracidade dos dados, não sendo necessaria a aderência total às propriedades ACID ja citadas [3]. Além disso, de acordo com o **Teorema CAP**, um banco que aceite particionamento não pode possuir alta disponibilidade e consistência simultâneamente. 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) [14].

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 [25].

Consiste basicamente em uma tabela *hash*, sendo o acesso aos dados realizado por meio de uma chave primaria, assim como ocorre em *maps* e dicionarios. 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 [17]. 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 [25].

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 [24]. 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 [25]. 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 facil a inserção de novos documentos ou a modificação dos documentos ja armazenados. Devido à essa flexibilidade, são recomendados para integração de dados e migração de esquemas [17].

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* [8], 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 [24].

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 [24].

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 [17].

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

### Grafos

Diferente dos bancos relacionais e dos ja 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 [17]. 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 areas de um grafo [24]. 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 camanha nesse grafo, sem ser necessarias mudanças em sua estrutura de nós e relações [24].

Uma diferença importante dos bancos orientados a grafos em relação aos modelos anteriores é o seu suporte menor a sistemas distribuidos, não sendo geralmente possível a distribuição dos nós em diferentes servidores [24].

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

# 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 [20]. 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 compania necessitava de um sistema com alta performance, confiabilidade, eficiência e que suportasse o contínuo crescimento da ferramenta [20, 18].

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* [11] e do *Google Bigtable* [8], e lançado em 2008 como um projeto *open source*. Foi mantido e atualizado apenas pelo Facebook até 2009, quando foi comprado pela Apache [18], sendo utilizado atualmente por companias como *Netflix*, *Spotify* e até em agências governamentais, como a NASA [1].

Pode ser definido como um banco de dados orientado a colunas *open source*, distribuído, descentralizado, elasticamente escalavel, altamente disponível, tolerante a falhas e variavelmente consistente [18]. 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 distribuidos 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 prejuizo 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 CAP3.2.2, 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*, formado por colunas agrupadas em conjuntos chamados famílias de colunas. Esse modelo é bastante semelhante ao

que foi proposto pelo Bigtable [20, 8]. 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.

### *Keyspace*

Um *keyspace* define o maior agrupamento de dados 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 [18].

Atributos do *keyspace* incluem:

- **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*.
- **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 [20, 18].

O modelo de família de colunas se diferencia do modelo relacional por ser o que é chamado comumente de *livre de esquema* (*schama 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 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.

# Referências


[1] Companies using nosql apache cassandra. http://www.planetcassandra.org/companies/. Acessado em 30 de maio de 2016. 12

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

[3] 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. 8

[4] 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

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

[6] O que são dados abertos? http://opendatahandbook.org/guide/pt_BR/what-is-open-data/. Acessado em 07 de maio de 2016. 2, 3

[7] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012. 7

[8] 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. 6, 10, 12, 14

[9] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 4

[10] 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. 2

[11] 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. 6, 12

[12] Peter Deutsch. The eight fallacies of distributed computing. http://today.java.net/jag/Fallacies.html. Acessado em 28 de maio de 2016. 7

[13] 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. 7

[14] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based scalable network services. 31(5), 1997. 5, 8

[15] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983. 5

[16] Jan L. Harrington. *Relational database design and implementation : clearly explained*. Morgan Kaufmann, 2009. 4, 5

[17] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. pages 336–341, 2011. 9, 10, 11

[18] Eben Hewitt. *Cassandra: The Definitive Guide*. O'Reilly Media, 2010. 12, 14

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

[20] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010. 12, 14

[21] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010. 1, 4, 6

[22] 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. 4

[23] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. Osborne/McGraw-Hill, 3 edition, 2003. 5

[24] Pramod J. Sadalage and Martin Fowler. *NoSQL Essencial*. Novatec, 2013. 4, 6, 7, 9, 10, 11

[25] Christof Strauch. Nosql databases. *Stuttgart Media University*, 2011. 6, 9