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

Armazenamento de Dados Abertos com NoSQL: Um estudo de caso com Dados do Bolsa Família e NoSQL Cassandra

Jorge Luiz Andrade

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

Orientadora
Prof.a Dr.a Maristela Terto de Holanda

Brasília
2017

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

Armazenamento de Dados Abertos com NoSQL: Um estudo de caso com Dados do Bolsa Família e NoSQL Cassandra

Jorge Luiz Andrade

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

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

Prof. Dr. Marcio Victorino
FCI/UnB

Dr. Ruben Huacarpuma
LATITUDE/UnB

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

Brasília, 29 de novembro de 2017

Dedicatória

Aos meus pais, amigos e namorada, que sempre me incentivaram à continuar na busca de meus objetivos.

Agradecimentos

Agradeço aos meus amigos e minha namorada por acreditarem na conclusão do meu curso e na realização deste trabalho.

Resumo

Os bancos não relacionais conhecidos como NoSQL tem se tornando uma alternativa para o armazenamento de grandes volumes de dados em ambientes distribuídos. Este trabalho apresenta as características desses bancos, em especial do Cassandra.

Neste trabalho é proposto o armazenamento de dados abertos do programa Bolsa Família em um banco NoSQL Cassandra, a fim de comparar seu desempenho em um ambiente com duas, quatro e seis máquinas e diferentes volumes de dados.

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

Abstract

Non-relational databases, also known as NoSQL, are becoming a storage alternative for huge volumes of data in distributed systems. This paper presents characteristics of these databases, in special Cassandra.

In this work, it is proposed the storage of open data from *Bolsa Família* program in a Cassandra NoSQL database, in order to compare it's performance on two, four and six machines and different volumes of data.

Keywords: databases, nosql, open data

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os bancos de dados podem ser definidos como um conjunto de dados que se relacionam entre si e armazenados de forma que possam ser acessados posteriormente, quando necessário [21]. Os bancos de dados relacionais vem predominando por pelo menos quatro décadas, mas seu desempenho em certas aplicações atuais, principalmente naquelas que trabalham com grande volumes de dados, denominadas *Big Data*, vem sendo questionado [40].

Esse questionamento levou à criação do movimento NoSQL, um novo paradigma de armazenamento de dados que ignora certas restrições dos bancos relacionais tradicionais, tentando 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 com grandes volumes de dados [40].

O Cassandra é um sistema gerenciador de bancos de dados NoSQL, originalmente proposto e utilizado pelo *Facebook*, que armazena seus dados em forma de colunas e linhas com esquema flexível. Essa característica é importante para o armazenamento de dados governamentais abertos, devido à sua natureza variável.

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 [6], tem crescido cada vez mais, e vem sendo necessário encontrar novas formas para o seu armazenamento e análise, comumente realizados por meio de bancos de dados relacionais.

O governo brasileiro vem, especialmente desde 2011, disponibilizando dados significativos a respeito da administração pública. Esses dados, apesar de estarem disponíveis para livre acesso, em geral não seguem um formato que permita fácil análise. Além disso, devido ao grande volume desses dados, uma abordagem tradicional pode não ser a forma mais adequada para a sua manipulação. Este trabalho visa analisar o desempenho de

um banco de dados NoSQL no contexto dos dados abertos brasileiros, em especial os do programa Bolsa Família fornecidos pelo Portal da Transparência.

1.1 Problema e Hipótese

Vários órgãos da administração pública brasileira disponibilizam seus dados de forma aberta na *web*. Entretanto, o grande espaço de armazenamento necessário para o armazenamento desses dados em sua totalidade pode gerar um desempenho não satisfatório ao se realizar sua inserção e posteriores consultas em um banco de dados relacional.

Desta forma, tem-se nesta monografia como hipótese que a utilização de múltiplas máquinas em um ambiente Cassandra distribuído pode apresentar uma melhora do desempenho que justifique a sua utilização no contexto da análise de dados abertos.

1.2 Justificativa

Apesar de recente, a utilização de bancos de dados não relacionais vem apresentando um bom desempenho em aplicações que exijam a utilização de grandes volumes de dados. Porém, poucos trabalhos realizaram a utilização de bancos NoSQL no contexto dos dados abertos da administração pública brasileira.

Este trabalho possui então como motivação a análise do desempenho de um banco Cassandra ao tratar de dados abertos brasileiros em um ambiente distribuído, comparando seus tempos de execução para inserção e busca de dados com o aumento do número de máquinas em um *cluster*, além de verificar o seu desempenho com diferentes volumes de dados.

1.3 Objetivos

Este trabalho tem como objetivo principal comparar o desempenho de um banco de dados Cassandra em diferentes tamanhos de *cluster*, a fim de se analisar sua melhora em um ambiente distribuído para armazenamento e busca de grandes volumes de dados.

A fim de se alcançar o objetivo geral do trabalho, foram traçados os seguintes objetivos específicos:

- Desenvolver uma aplicação para a inserção e busca dos dados do Bolsa Família em um banco de dados Cassandra, permitindo a realização de testes para as diferentes configuarações.

- Comparar o desempenho dos testes e verificar sua melhora ao se aumentar o número de máquinas no *cluster*.

1.4 Metodologia

Neste trabalho foi utilizado como estudo de caso os dados públicos do programa Bolsa Família. Esses dados foram inseridos e extraídos de um banco de dados não relacional Cassandra, com *clusters* de dois, quatro e seis máquinas e dois volumes distintos de dados. Os tempos dos testes foram analisados e seu desempenho comparado para verificar a viabilidade do uso de um banco de dados Cassandra no tratamento de dados abertos.

1.5 Estrutura do Trabalho

Este trabalho está estruturado nos seguintes capítulos:

- **Capítulo 2**: São apresentadas definições e características de dados abertos, sendo feita também sua contextualização no âmbito do governo brasileiro, mostrando ações que vem sendo tomadas nesse sentido.
- **Capítulo 3**: É feita uma exposição de conceitos relacionados a bancos de dados, realizando uma comparação entre os bancos de dados relacionais e os bancos NoSQL. São apresentadas características de um banco relacional, propriedades ACID e normalização de dados. Também são apresentadas aspectos gerais dos bancos NoSQL, bem como os teoremas CAP e BASE. São apresentados também os principais modelos atuais de bancos NoSQL e suas características.
- **Capítulo 4**: Detalha o banco de dados Cassandra, demonstrando suas características e funcionamento. Seu modelo de dados é explicado, bem como sua arquitetura e configurações. A linguagem CQL, responsável pela realização de consultas em um banco Cassandra também é brevemente apresentada.
- **Capítulo 5**: O programa Bolsa Família é apresentado e o modelo Cassandra proposto é exposto. É explicado o desenvolvimento da aplicação desenvolvida nos testes realizados, assim como o ambiente utilizado no trabalho.
- **Capítulo 6**: Os resultados obtidos nos testes de inserção e busca com o *driver* da *Datastax* no banco Cassandra são apresentados. É feita também a comparação dos testes realizados nas diferentes configurações do *cluster*.
- **Capítulo 7**: É feita a conclusão relembrando pontos importantes do trabalho. São também discutidos possíveis trabalhos futuros relacionados ao trabalho apresentado.

Capítulo 2

Dados Abertos

Este capítulo apresenta os conceitos e definições sobre dados abertos, em especial os governamentais e os disponibilizados pelo governo brasileiro. Na seção 2.1 é apresentada a contextualização dos dados abertos, com suas definições e características. A seção 2.2 expõe a classificação de cinco estrelas de dados abertos, descrevendo seus custos e benefícios. A seção 2.3 apresenta definições de dados abertos governamentais e seus oito princípios. Por fim, na seção 2.4 a situação dos dados abertos no contexto brasileiro é descrito.

2.1 Contextualização

O rápido crescimento no volume de dados gerados pela sociedade nos últimos anos tem levado a uma necessidade cada vez maior de ferramentas e pessoas que consigam trabalhar com esses dados. Um estudo da *IDC Digital Universe*, realizado em 2011, estimou que naquele ano o volume de dados criados, replicados e consumidos foi de 1,8 trilhão de *gigabytes* [28]. Esses dados, entretanto, não estão disponíveis de forma aberta ao público, nem estruturados de forma a facilitar a sua compreensão mesmo por aqueles que a eles tem acesso [34].

Nesse contexto, diversas empresas, governos e institutos vem trabalhando para uma maior abertura desses dados, objetivando os chamados dados abertos. O termo "Dados abertos" no conceito que conhecemos hoje surgiu em 1995 em um documento de uma 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 [34].

A *Open Knowledge Foundation*, organização mundial que promove a abertura de dados [13], define um dado como aberto "se qualquer pessoa está livre para acessa-lo, utiliza-lo, modifica-lo, e compartilha-lo — restrito, no máximo, a medidas que preservam a proveniência e abertura." [6]. A abertura dos dados evita mecanismos de controle e restrições

sobre os mesmos, o que permite seu uso de forma livre [34]. Essas características podem ser resumidas nos seguintes pontos:

- **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'.

2.2 Classificação de Dados Abertos

Tim Berners-Lee, o inventor da *Web* e um defensor da abertura de dados, propôs, em 2010, princípios que definem um sistema de classificação de dados abertos por meio de estrelas. Quanto mais aberto é o dado, maior o número de estrelas que ele possui e mais fácil é para enriquece-lo [34].

As cinco estrelas de Tim Berners-Lee são:

- **1 estrela**: O dado está disponível na Internet, em qualquer formato, acompanhado de licença aberta.
- **2 estrelas**: O dado está disponível na Internet de maneira estruturada, em um formato que permita sua leitura por máquinas (por exemplo, XML).
- **3 estrelas**: O dado está disponível na Internet, de maneira estruturada, e em formato não proprietário (por exemplo, *CSV*).
- **4 estrelas**: O dado está disponível na Internet, de maneira estruturada e em formato proprietário. Além disso, deve estar dentro dos padrões estabelecidos pela W3C (RDF): utilização de URI para nomear relações entre coisas e propriedades.
- **5 estrelas**: Além das propriedades anteriores, ter no RDF conexões com outros dados, por meio de *links*, de forma a se obter um contexto de dados relevantes.

Dados publicados seguindo a classificação de estrelas de Berners-Lee conferem uma série de custos e benefícios, tanto para quem os publica quanto para quem os consome, sendo eles [34]:

- **1 Estrela**:

  Quem consome

  - Ver os dados
  - Imprimi-los
  - Guardá-los
  - Modificar os dados como queira
  - Acessar os dados de qualquer sistema
  - Compartilhar os dados com qualquer pessoa

  Quem produz

  - É simples de publicar
  - Não é necessário explicar repetidamente para outros que eles podem usar seus dados

- **2 Estrelas**:

  Quem consome

  - É possível processá-los diretamente com aplicativos proprietários
  - É possível exportar os dados para outro formato estruturado

  Quem produz

  - Ainda é simples de publicar

- **3 Estrelas**:

  Quem consome

  - Pode manipular os dados da forma que quiser, sem a necessidade de nenhum *software* proprietário

  Quem produz

  - Ainda é relativamente simples de publicar
  - Podem ser necessários *plug-ins* ou conversores para exportar os dados do *software* proprietário

- **4 Estrelas**:

  Quem consome

  - Fazer marcações nos dados
  - Reusar partes dos dados
  - Reusar ferramentas e bibliotecas existentes
  - Combinar os dados com outros dados
  - O entendimento do formato RDF pode ser mais difícil do que formatos tabulares (CSV) ou em árvore (XML).

  Quem produz

  - Controle granular sobre seus dados, podendo realizar otimizações de acesso
  - Outros publicadores de dados podem fazer ligações com os dados
  - Pode ser necessário investimento de tempo para dividir ou agrupar os dados

- **5 Estrelas**:

  Quem consome

  - Encontrar dados relacionados enquanto consome os dados
  - Aprender sobre o esquema de dados

  Quem produz

  - Os dados podem ser encontrados com maior facilidade
  - Os dados possuem maior valor agregado
  - A organização ganha os mesmos benefícios da vinculação de dados que os consumidores

2.3 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 [25]:

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.

2.4 Contexto Brasileiro de Dados Abertos

A participação brasileira na área de dados abertos teve seu 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 nesse ano o portal dados.gov.br, que disponibiliza dados governamentais de forma aberta [34].

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 da participação social, geração de emprego e renda e estímulo à inovação tecnológica [22]. 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 necessários para atender às condições de disseminação e compartilhamento de dados e informações públicas no modelo de Dados Abertos" [9].

O governo tem importância fundamental na questão de dados abertos, devido à grande quantidade de dados que coleta e por serem públicos, conforme a lei, podendo ser tornados abertos e disponíveis para a sociedade [13].

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 [13].

Apesar desses esforços e da existência de programas avançados de transparência pública, ainda são raros os órgãos que disponibilizam dados de forma aberta. Em geral esses dados estão disponíveis para visualização, mas barreiras técnicas e políticas impedem sua reutilização [12].

Capítulo 3

Bancos de Dados

Neste capítulo será feita a abordagem dos bancos de dados relacionais e não relacionais. A Seção 3.1 apresenta o conceito de bancos de dados relacionais bem como suas principais características. Na Seção 3.2 são definidos os bancos de dados não relacionais, suas características e o que os distingue dos bancos relacionais. Também são apresentados os quatro principais modelos de bancos de dados não relacionais: chave-valor, modelo orientado documentos, modelo orientado a colunas e modelo orientado a grafo.

3.1 Bancos de Dados Relacionais

O armazenamento e 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 [37]. Em 1970, foi proposto por Edgar Codd uma nova forma de armazenamento de dados, que ficou conhecida como modelo de dados relacionais (*relational model of data*) [20].

Um banco de dados relacional é composto por um conjunto de relações, sendo essas relações um conjunto não ordenado de tuplas. Cada tupla consiste em uma série de valores de atributos identificados por seus nomes. O conjunto de atributos em uma tupla da relação é denominado coluna [32]. 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 [40]. Outra questão de importância em bancos de dados 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), programas que lidam com todo o acesso do usuário com o banco de dados [21, 30].

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 [40]. Contudo, nos últimos anos, o modelo conhecido como NoSQL vem surgindo como essa alternativa.

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 [39]. Para garantir que isso ocorra, um SGBD deve prover as seguintes propriedades, conhecidas como propriedades ACID [29]:

- **Atomicidade**, onde uma determinada transação deve ser feita em sua totalidade, ou seja, todas as operações 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 várias aplicações atuais [27].

3.1.2 Normalização

Uma prática 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 [30]. Serão descritas as três primeiras formas normais, como propostas por Edgar F. Codd em 1972 [21].

Primeira Forma Normal

Um banco está na primeira forma normal (**1FN**) se cada tabela estiver organizada por colunas e linhas, com cada linha possuindo uma chave primária 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

Um banco está na segunda forma normal (**2FN**) quando, além de obedecer à primeira forma normal, possui todos atributos não-chave funcionalmente dependentes da chave primária. 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 [30]. Em outras palavras, uma coluna não pode depender apenas de parte da chave primária, ou seja, se uma tabela não possui chave primária composta e esta na primeira forma normal, ela também esta na segunda forma normal.

Terceira Forma Normal

Uma tabela está 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 de Dados NoSQL

O rápido crescimento no volume de dados nos últimos anos, principalmente após a bolha da Internet na década de 90, trouxe a necessidade de certa mudança em relação ao modelo relacional comumente utilizado até então [40]. Modelos relacionais possuem diversas vantagem já citadas, porém restrições como propriedades ACID e normalização levam ao surgimento de problemas quando precisamos aplicá-los nesse domínio recente de expansão dos dados, por apresentarem problemas de escalabilidade, complexidade dos dados e rigidez em seus esquemas [36].

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 já sugeridas em publicações anteriores, como o Google Bigtable em 2006 [19], e Amazon's Dynamo em 2007 [23, 41].

3.2.1 Definição e Características

Apesar de o termo não possuir 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, com esquema flexível, distribuídos e tolerantes a falhas [40]. Buscam um processamento de dados rápido e de forma eficiente, apresentando maior flexibilidade em seu esquema do que os bancos relacionais. Entre as razões e vantagens dos bancos NoSQL pode-se citar [41]:

- **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 se obter um melhor desempenho.
- **Alto rendimento**: Bancos NoSQL surgiram da necessidade de armazenamento e processamento de volumes de dados cada vez maiores, sendo por isso construídos objetivando um melhor desempenho, 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 [36]. 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 *hardware* 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 relacionais costumam possuir licenças pagas, gerando um custo financeiro alto, principalmente quando executados em múltiplas maquinas [40]. 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 pode-se garantir somente duas das seguintes três propriedades em um determinado sistema: Consistência (*Consistency*) , Disponibilidade (*Availability*) e Tolerância a partições (*Partition-resilience*) [26]. Essas propriedades podem ser definidas como:

- **Consistência** define que todos os nós devem possuir os mesmos dados em determinado instante, e um pedido de leitura em qualquer desses nós garante o dado mais atual possível do sistema.
- **Disponibilidade** garante que todas as requisições feitas a um nó disponível resultem em uma resposta do sistema.
- **Tolerância a partições** garante que o sistema irá continuar funcionando mesmo na hipótese de eventuais falhas na comunicação entre os nós.

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

Entretanto, dado que sistemas distribuídos estão sempre sujeitos à falhas de redes [24], em 2012 Brewer publicou uma revisão de seu artigo original, onde diz que já que todos sistema deve suportar Tolerância a Falhas, a escolha deve ser feita entre Consistência e Disponibilidade [18].

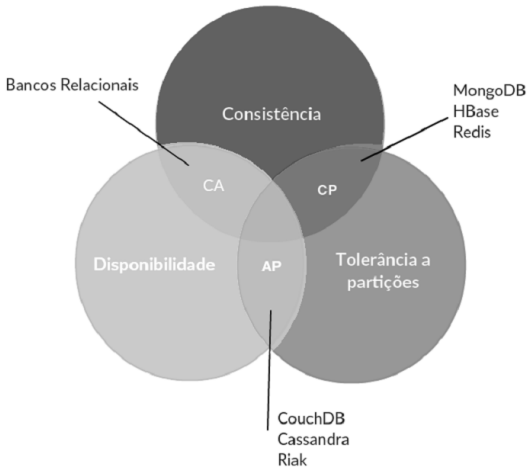

Figura 3.1: Propriedades CAP e exemplos.

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 [8]. 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 desempenho

Tabela 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

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) [27].

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 seus dados serão eventualmente consistentes [38].

Sistemas distribuídos costumam manter cópias de dados em várias máquinas 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.

Chave-Valor

Bancos de dados com armazenamento em chave-valor existem a muito tempo, como o *Berkeley DB*, criado em 1994, ganhando maior importância no meio NoSQL a partir do Amazon DynamoDB [41].

Consistem basicamente em uma tabela *hash*, sendo o acesso aos dados realizado por meio de uma chave primária, 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 [31]. O valor armazenado em geral pode representar qualquer tipo de objeto, como uma *string* ou um *BLOB*, não sendo necessário que exista qualquer relação entre diferentes registros, ficando a aplicação responsável pelo seu tratamento.

Bancos de chave-valor favorecem escalabilidade sobre consistência de dados, e por esse motivo não apresentam formas de consulta analíticas, como *joins* e agregações [41].

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

Documentos

Bancos de dados orientados a documentos armazenam seus dados em forma de documentos, podendo esses terem formato *XML*, *JSON*, *BSON*, etc [40]. 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 [41]. 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 Figura 3.2 apresenta um exemplo de registros armazenados em um banco de dados orientado a objetos. Apesar de semelhantes, os registros não apresentam os mesmos campos, como os campos *endereco* e *ultimaCompra*, o que permite grande flexibilidade no modelo orientado a documentos.

Os documentos armazenados não são opacos à aplicação, ou seja, seus conteúdos podem ser acessados diretamente, com consultas em atributos de seus registros. Isso permite a manipulação de estruturas mais complexas, e mesmo assim não ter restrições 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 [31].

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* [19], e armazenam seus dados em famílias de colunas que são associadas a uma chave de linha. Cada uma des-

```
{
  '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'
}
```

Figura 3.2: Exemplo de banco de dados orientado a documentos (Adaptado) (Fonte: [40]).

sas famílias de colunas pode possuir várias colunas, e são consideradas dados relacionados que podem ser acessados ao mesmo tempo [40].

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 [31].

A Figura 3.3 ilustra um modelo de dados orientado a colunas, em específico com a utilização do banco Cassandra.

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.

Figura 3.3: Modelo de dados orientado a colunas em banco Cassandra (Fonte: [5]).

Grafos

Diferente dos bancos relacionais e dos já citados modelos NoSQL, um banco de dados em grafos é especializado em dados altamente conectados, sendo ideais para aplicações que realizam consultas baseadas em relacionamentos [31]. 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 arestas de um grafo [40]. Esses nós podem possuir propriedades dos objetos que representam, assim como as arestas, que podem possuir atributos do relacionamento e possuindo, além disso, significância em sua direção.

Consultas nesse tipo de modelo são realizadas percorrendo-se o grafo, o que gera como vantagem a possibilidade de se modificar a forma que se caminha nesse grafo, não sendo necessárias mudanças em sua estrutura de nós e arestas [40].

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 [40].

A Figura 3.4 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*.

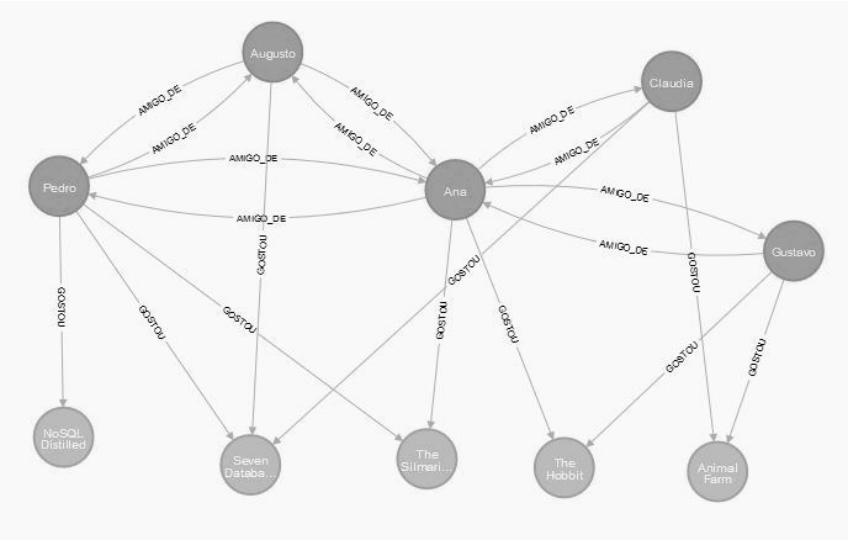

Figura 3.4: Visualização de um banco de dados em grafo.

Capítulo 4

Cassandra

Este trabalho utiliza o Cassandra como sistema gerenciador de banco de dados para análise dos dados obtidos do INEP. Será realizada uma comparação do seu desempenho em relação à abordagem relacional, além da verificação dos benefícios de uma aplicação distribuída.

Como visto no Capítulo 3, o Apache Cassandra, é um sistema gerenciador de banco de dados orientado a colunas altamente disponível e distribuído em servidores constituídos de *hardware* de "prateleira" para gerenciamento de grande volumes de dados [35]. Este capítulo tem como objetivo descrever o NoSQL Cassandra, suas características, funcionamento, vantagens e desvantagens.

4.1 Definição

O Cassandra teve origem em 2007 como um projeto do *Facebook* para resolver problemas 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 [35].

O projeto foi desenvolvido por Jeff Hammerbacher, Avinash Lakshman, Karthik Ranganathan e Prashant Malik, sendo seu modelo de dados fortemente inspirado nos trabalhos anteriores do *Amazon Dynamo* [23] e do *Google Bigtable* [19]. Seu código foi aberto pelo *Facebook* em 2008, e adotado em 2009 como um projeto incubado da Apache [33]. Atualmente é utilizado por companhias como *Netflix*, *Spotify* e em agências governamentais, como a NASA [3].

O Apache Cassandra pode ser definido como um sistema gerenciador de banco de dados orientado a colunas, *open source*, distribuído, descentralizado, elasticamente escalável, altamente disponível, tolerante a falhas e variavelmente consistente [33].

4.2 Características

Distribuído e Descentralizado

O Cassandra é capaz de ser executado em múltiplas máquinas de forma transparente ao usuário, que o enxerga como um sistema unificado. Apesar de ser possível sua execução em um único nó, tem-se um melhor desempenho com a execução distribuída, além de uma melhora na confiabilidade 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, aumentando sua disponibilidade, e simplificando a operação e manutenção do *cluster* [33].

Elasticamente Escalável

Escalabilidade é a propriedade que um sistema tem de atender um crescente número de requisições sem prejuízo de desempenho. 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, ou seja, na necessidade de uma melhora do desempenho da aplicação novas máquinas podem ser adicionadas, ficando o Cassandra encarregado de realizar a distribuição dos dados de forma transparente, sem necessidade de configurações adicionais ou reiniciamento do sistema. Da mesma forma, caso necessário, máquinas podem ser retiradas do *cluster* sem prejuízo ao sistema como um todo, devido ao rebalanceamento automático [33].

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 [33].

Variavelmente Consistente

Como visto no Teorema CAP, em um sistema distribuído não é possível garantir total disponibilidade e consistência, definida como a capacidade do sistema de retornar o valor mais atual em uma requisição.

O Cassandra é por vezes descrito como "eventualmente consistente", por trocar parte de sua consistência por alta disponibilidade. Essa definição, porém, não é totalmente correta, podendo ser melhor definido como "variavelmente consistente" (*tuneably consistent*), podendo essa sua consistência ser ajustada de acordo com o tipo de aplicação [33].

4.3 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 disposição de dados é bastante semelhante ao que foi proposto pelo *Bigtable* [19, 35]. 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 tem-se 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.

Keyspace

Um *keyspace* define o agrupamento de dados mais externo no Cassandra, podendo ser correspondido a um banco de um SGBD relacional, sendo determinado por um nome e uma série de atributos que descrevem o seu comportamento. Atributos do *keyspace* incluem [33]:

- **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 desempenho 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** podem ser vistas 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.

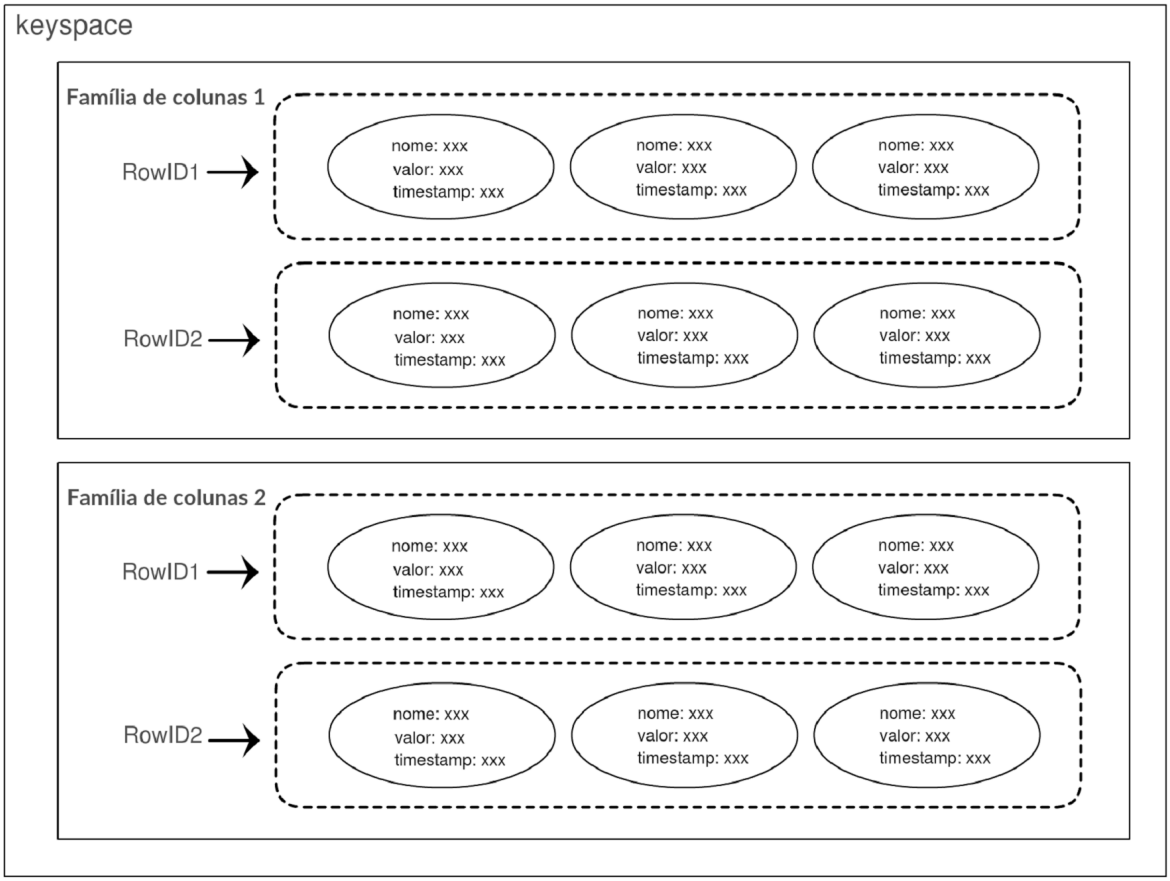

Figura 4.1: Modelo de dados do Cassandra (Fonte: [4]).

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 [33, 35].

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 [33].

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 desempenho 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 [33].

4.4 Arquitetura

O Cassandra foi projetado para lidar com grandes massas de dados distribuídas em vários nós sem ponto único de falha, dado o 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* [5, 33].

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, sem causar, entretanto, uma interrupção no serviço. Ao mesmo tempo, ao se inserir um novo nó no sistema, é necessário que ele receba 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 [33].

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 [5].

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* [5].

4.4.1 Protocolo Gossip

Os nós do Cassandra se comunicam entre si por meio de *Gossip*, 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.

Protocolos *gossip* em geral assumem uma rede falha, e são utilizados em sistemas em rede grandes e descentralizados, em geral em mecanismos de replicação em bancos de dados [33].

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, tendo o nó *seed* a única função de inicializar o processo *gossip* para novos nós que estejam sendo adicionados ao *cluster* [5].

4.4.2 Operações de Leitura e Escrita

Ao se realizar uma operação de escrita em um nó no Cassandra, o dado é armazenado 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 [33].

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 [33].

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 rea-

lizada 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 [5, 33].

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 [5, 33]. As operações descritas estão esquematizadas na Figura 4.2.

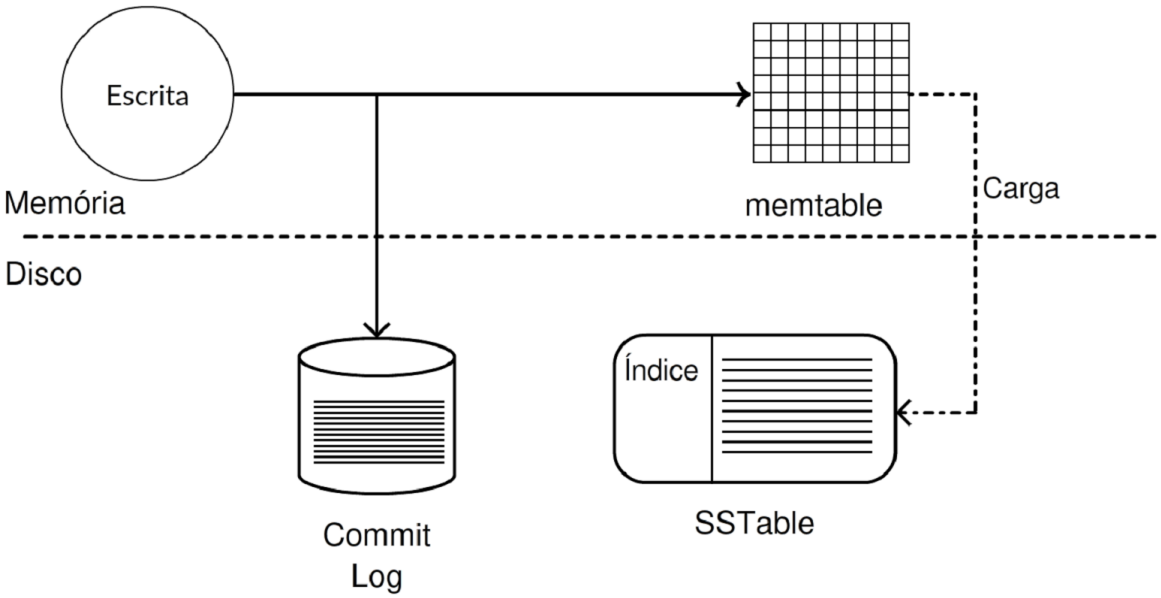

Figura 4.2: Escrita de dados em um banco Cassandra (Fonte: [5]).

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 [5].

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 [33].

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 [5].

4.4.3 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 qual nó desse *cluster* o dado será armazenado [5].

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 [5].

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* [17].

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*, possuindo também 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.

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 quais nós as cópias serão armazenadas. O número de cópias em um *cluster* é denominado fator de replicação. Se um fator de replicação um é utilizado, existe apenas uma copia de cada linha no *cluster*, e caso o nó em que essa linha esteja armazenada fique indisponível, não é mais possível acessa-la. Todas as réplicas são igualmente importantes, não existindo uma réplica mestre ou primária. Como regra geral, o fator de replicação não deve exceder o número de nós

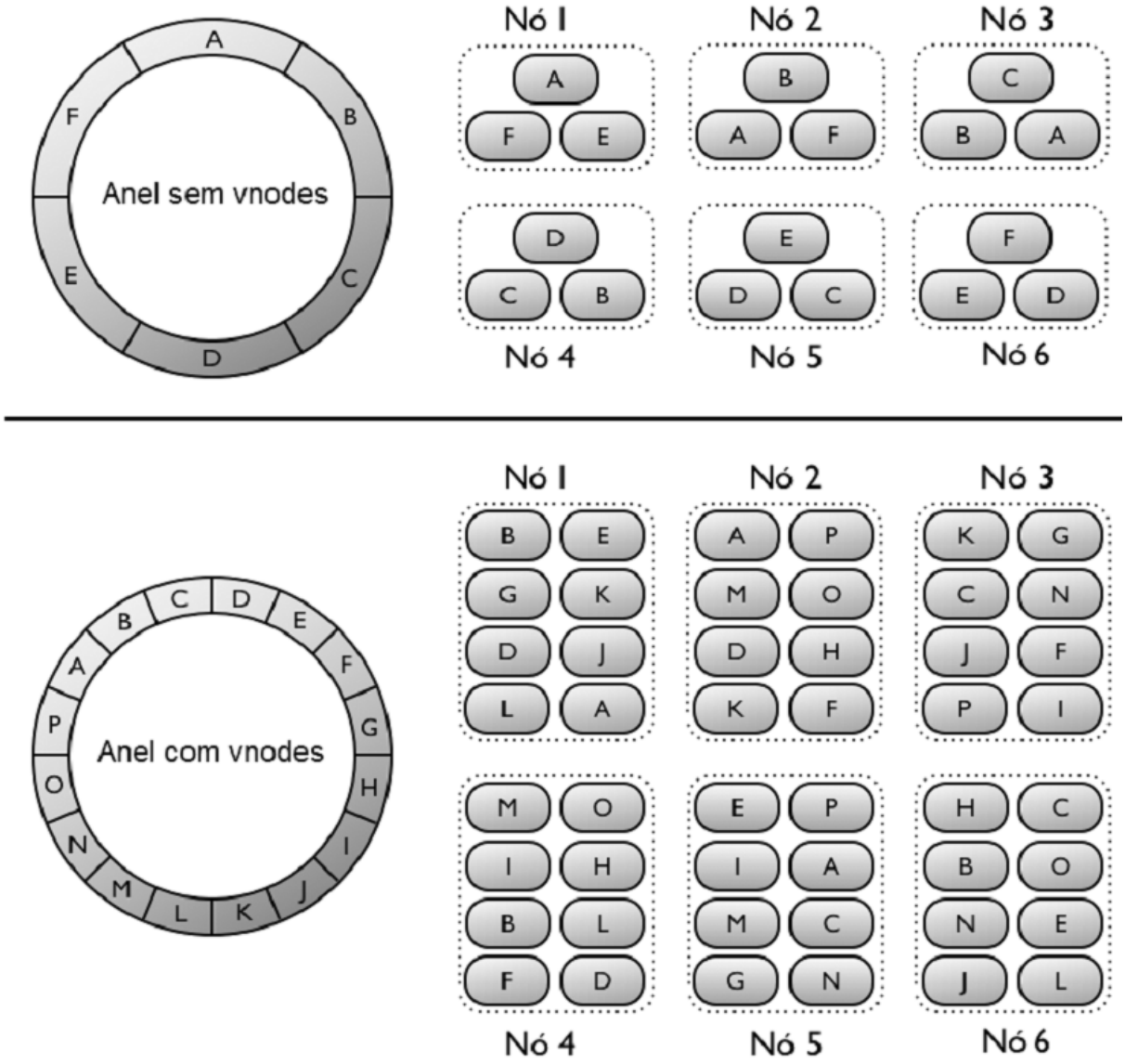

Figura 4.3: Distribuição de nós em um anel (Fonte: [17]).

no *cluster*. É possível, porém, aumentar o fator de replicação e inserir o número desejado de nós posteriormente [5].

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* [5].

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*, fornecendo 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.

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 [5].

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 [5].

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 [33]:

- ***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* na maioria dos nós do *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 [33]:

- ***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 dos nós tenha 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 terá a versão mais atual da informação.

4.5 Linguagem CQL

A linguagem *CQL*(*Cassandra Query Language*), introduzida na versão 0.8 do Cassandra, é uma linguagem de *script*, utilizada como interface padrão para um banco de dados Cassandra, sendo então a maneira recomendada de interação com o mesmo [33].

Sua utilização é similar à do SQL convencional, com a abstração de tabelas sendo constituídas por colunas e linhas. Devido à natureza desnormatizada do um modelo de dados em um banco Cassandra, o CQL possui limitações no uso de queries consideradas mais complexas, como joins e subqueries [33].

Para interação direta com o banco também é disponibilizado o *cqlsh*(*CQL shell*), um utilitário desenvolvido em Python, que por meio de linha de comando permite a execução de chamadas CQL, como criação de *keyspaces*, tabelas, inserções e consultas [10].

Capítulo 5

Estudo de Caso

Neste capítulo será apresentado o estudo de caso utilizado neste trabalho, sendo ele o programa Bolsa Família. A Seção 5.1 apresenta informações sobre o programa bem como as características dos dados disponibilizados pelo governo brasileiro e os dados efetivamente utilizados no trabalho. Na Seção 5.2 o modelo de dados utilizado no Cassandra é definido, e as decisões de escolha das características do *keyspace* e da tabela criada são explicadas. A Seção 5.3 apresenta a arquitetura do ambiente utilizado nos testes e suas configurações. Por fim, a Seção 5.4 expõe a aplicação desenvolvida responsável pelo tratamento, inserção e consulta dos dados no banco Cassandra.

5.1 O Programa Bolsa Família

O Bolsa Família é um programa federal em operação desde 2003, quando foi instituído pelo então presidente Luiz Inácio Lula da Silva, com objetivo de "combater a fome e promover a segurança alimentar e nutricional, combater a pobreza e outras formas de privação das famílias e promover o acesso à rede de serviços públicos, em especial, saúde, educação, segurança alimentar e assistência social" [2]. O programa se caracteriza pela transferência de renda para famílias brasileiras em situação de pobreza e extrema pobreza, ou seja, que recebam até R$170,00 e R$85,00 respectivamente. Atualmente cerca de 13,9 milhões de famílias são atendidas pelo programa, recebendo em média R$182,00 cada, totalizando R$27,4 bilhões em 2016, que recebem recursos do Ministério do Desenvolvimento Social, por meio da Caixa Econômica Federal [7, 11].

Os dados do Bolsa Família utilizados para estudo de caso nesse trabalho foram obtidos no *site* do Portal da Transparência, um projeto da Controladoria Geral da União, que disponibiliza informações sobre os gastos do dinheiro público, com objetivo de melhorar a transparência da gestão pública [15].

Tabela 5.1: Campos utilizados

Campo	**Tipo**	**Utilizado**
UF	Text	Sim
Código SIAFI Município	Int	Sim
Nome Município	Text	Sim
Código Função	-	Não
Código Subfunção	-	Não
Código Programa	-	Não
Código Ação	-	Não
NIS Favorecido	Bigint	Sim
Nome Favorecido	Text	Sim
Fonte-Finalidade	Text	Sim
Valor Parcela	Double	Sim
Mês Competência	Timestamp	Sim

Esses dados são disponibilizados em formato aberto *.csv*, e agrupados mensalmente, trazendo informações sobre a transferência de recursos federais aos participantes do programa.

5.1.1 Dados Utilizados

Para os testes de desempenho foram utilizados um total de 30 arquivos, correspondentes a trinta meses do programa, compreendidos entre Julho de 2014 e Dezembro de 2016. O total de arquivos foi determinado considerando-se a capacidade de armazenamento das máquinas do ambiente utilizado, especialmente nos testes de *cluster* com apenas duas máquinas. Cada arquivo utilizado possui em média 16GiB de tamanho e 13.968.749 registros.

Cada arquivo apresenta como campos: UF, Código SIAFI Município, Nome Município, Código Função, Código Subfunção, Código Programa, Código Ação, NIS Favorecido, Nome Favorecido, Fonte-Finalidade, Valor Parcela e Mês Competência. Como os testes realizados não buscam uma análise interpretativa dos dados armazenados, e sim a verificação de uma possível melhora de desempenho, um subconjunto dessas colunas foi escolhido para a realização desse trabalho. A Tabela 5.1 apresenta os campos presentes nos arquivos, bem como seus tipos e quais campos foram utilizados na modelagem dos dados. Os campos Código Função, Código Subfunção, Código Programa, Código Ação se repetem em todos os registros e por isso não foram utilizados.

5.2 Modelo de dados Cassandra

O modelo de dados do Cassandra foi criado utilizando-se um *keyspace* com fator de replicação de apenas um. O fator de replicação foi definido considerando que a tolerância a falhas não é um fator relevante nos testes a serem realizados, devido ao ambiente controlado em que eles são feitos, considerando também um possível impacto nos tempos de inserção e consulta. O Código 5.1 apresenta o comando CQL utilizado para sua criação.

O Cassandra apresenta duas estratégias de replicações de dados, a *SimpleStrategy* e a *NetworkTopologyStrategy*, sendo a primeira utilizada em *datacenters* únicos e com apenas um *rack*, e a segunda para uso em múltiplos *datacenters*. Devido ao ambiente restrito do laboratório utilizado, foi escolhida a *SimpleStrategy* como estratégia de replicação dos dados.

Código 5.1: Código CQL para criação do keyspace

```
CREATE KEYSPACE bolsa_familia WITH replication = {'class': '
    ↪ SimpleStrategy', 'replication_factor': 1};
```

Apenas uma tabela foi utilizada para os testes, contendo as colunas definidas anteriormente. O Código 5.2 apresenta o comando CQL utilizado para sua criação. Essa tabela é a única utilizada no banco, sendo criada com todos os campos com seus respectivos tipos, também sendo definida sua chave primária.

Código 5.2: Código CQL para criação da tabela

```
CREATE TABLE bolsa_familia.dados (uf TEXT, periodo
    ↪ TIMESTAMP, valor DOUBLE, nis_favorecido BIGINT,
    ↪ cod_municipio INT, fonte TEXT, nome_favorecido
    ↪ TEXT, nome_municipio TEXT, PRIMARY KEY(
    ↪ nis_favorecido, periodo, valor));
```

Ao se definir uma chave primária no Cassandra é necessária a escolha de uma *Partition Key* (chave de partição) e uma *Clustering Key* (chave de agrupamento). A *Partition Key* é responsável pela distribuição dos dados nos nós do ambiente, enquanto a *Clustering Key* distribui esses dados dentro de cada nó.

Para determinar a localização de um registro dentro do *cluster*, o particionador utilizado realiza um processo de *hash* na chave de partição, sendo recomendável, então, a escolha de uma chave de partição que apresente boa distribuição dos dados dentro do *cluster*.

Para isso foi utilizada a coluna **nis_favorecido**, por ser ela a responsável por identificar unicamente cada beneficiário do programa. Além disso, para complementar a chave primária foram escolhidos os campos **periodo** e **valor**, que identificam unicamente cada

registro, considerando que um mesmo favorecido pelo programa(identificado pela coluna nis_favorecido) pode receber mais de uma parcela em um mesmo mês, ou parcelas em meses distintos. Essa escolha poderia mesmo assim resultar em colisão entre chaves na ocasião do recebimento de benefícios de valores iguais em um mesmo mês, essa situação, entretanto, não foi verificada durante os testes com os arquivos utilizados. A Figura 5.1 traz o modelo utilizado, onde são apresentadas as chaves primárias, separadas em chave de partição e de agrupamento, e as demais colunas que são definidas por elas.

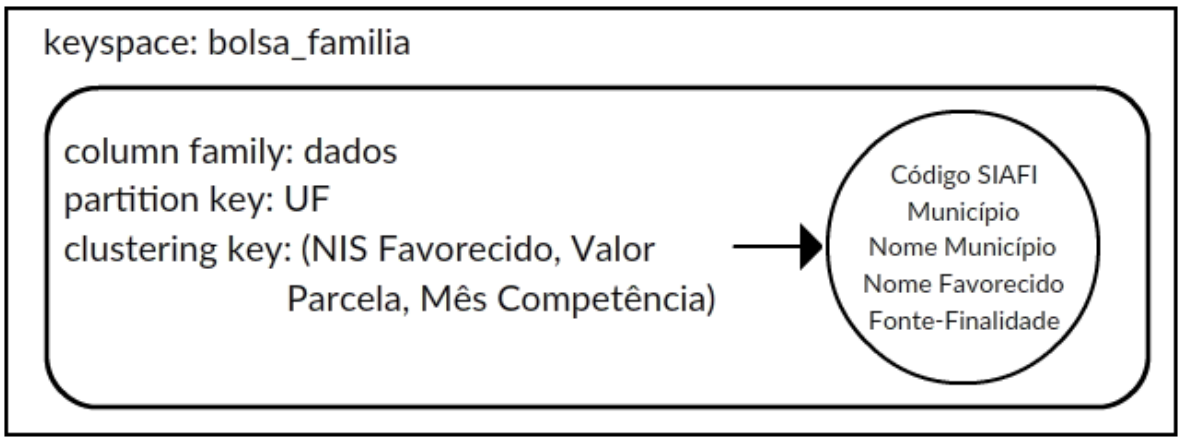

Figura 5.1: Modelo de Dados.

A configuração do ambiente Cassandra foi feita por meio do cliente *cqlsh*, que permite a realização de consultas e manipulações no banco por meio da linguagem *CQL*.

5.3 Arquitetura do Ambiente

O ambiente utilizado consiste em um *cluster* composto por seis máquinas com Intel i5-4570 3.20GHz, 16GB de RAM, disco rígido de 500GB, com sistema operacional Ubuntu, como representado na Figura 5.2.

O cliente Cassandra, em sua versão 3.0.4, foi instalado em cada uma das seis máquinas utilizadas no teste. Em cada máquina foi necessária a modificação do arquivo de configuração *cassandra.yaml* para possibilitar correta configuração e detecção do *cluster*. O número de nós virtuais foi mantido em 256, valor recomendado para manter um balanceamento correto dos nós, o particionador padrão *Murmur3Partitioner* foi mantido por ser o recomendado para novos *clusters* [14], e como *snitch* foi utilizado o *SimpleSnitch*, pois a implementação desenvolvida faz uso de apenas um *datacenter*.

O arquivo *cassandra.yaml* também define os IPs dos nós *seed*, responsáveis por manter informações sobre os outros nós, bem como o nome do *cluster* utilizado. Esses dados

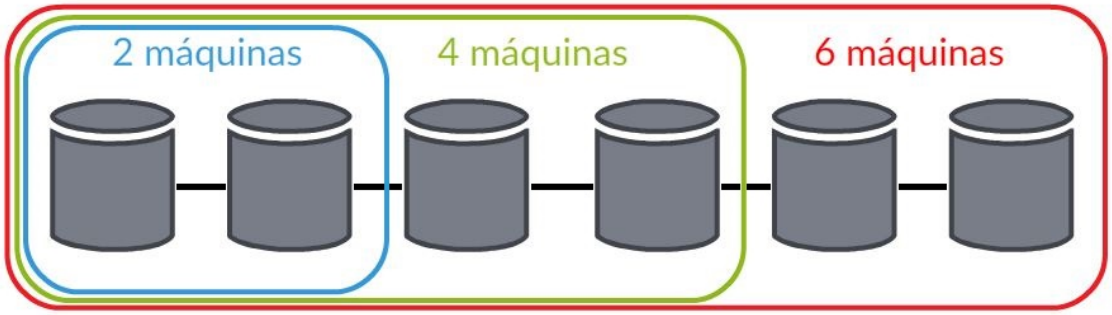

Figura 5.2: Ambiente utilizado.

devem ser configurados de forma idêntica em todas as máquinas para permitir a sua detecção dentro do mesmo *cluster*. A configurações definidas no arquivo são apresentadas no Código 5.3.

Código 5.3: Configuração cassandra.yaml

```
cluster_name: 'BolsaFamilia Cluster C2M FR1'

num_tokens: 256

partitioner: org.apache.cassandra.dht.Murmur3Partitioner

seed_provider:
        - class_name: org.apache.cassandra.locator.
          ↪ SimpleSeedProvider
        parameters:
                - seeds: "164.41.40.35"

endpoint_snitch: SimpleSnitch
```

Além disso, seguindo as orientações em [16], e guardadas as devidas limitações do laboratório, foram realizadas as configurações do Linux:

- Remoção do limite de memória (parâmetro *memlock*)
- Aumento do limite do número de arquivos abertos(parâmetro *nofile*)
- Desativação do *swap*

A duas primeiras configurações buscam evitar erros de recursos insuficientes no Linux, já a desativação do *swap* evita que os nós utilizem o *swap* do sistema, o que causaria piora no desempenho da aplicação, sendo por isso preferível a reinicialização do nó.

5.4 Desenvolvimento da Aplicação

Para inserção e busca dos dados no ambiente, foi desenvolvida uma aplicação em Java utilizando o *driver* oferecido pela *Datastax*, empresa que disponibiliza um distribuição gratuita do Apache Cassandra. Essa aplicação é responsável pela leitura de todos os arquivos de entrada e inserção dos campos utilizados no banco de dados Cassandra, assim como a posterior busca desses dados.

A figura 5.3 apresenta um esquema simplificado do processo de inserção de dados realizado pela aplicação.

Os dados são lidos diretamente dos arquivos .csv baixados do Portal da Transparência. A aplicação realiza o tratamento dos campos, com a separação das linhas, a seleção dos campos utilizados e a conversão de valores incompatíveis com o banco. A inserção dos dados é feita de forma assíncrona utilizando os métodos do *driver*.

A leitura dos dados é feita no mesmo programa, após a inserção de todos os dados, também utilizando o driver da *Datastax*.

Todas as interações com o banco, seja na inserção quanto na leitura, foram realizadas por meio da linguagem CQL, que se assemelha a consultas padrões SQL.

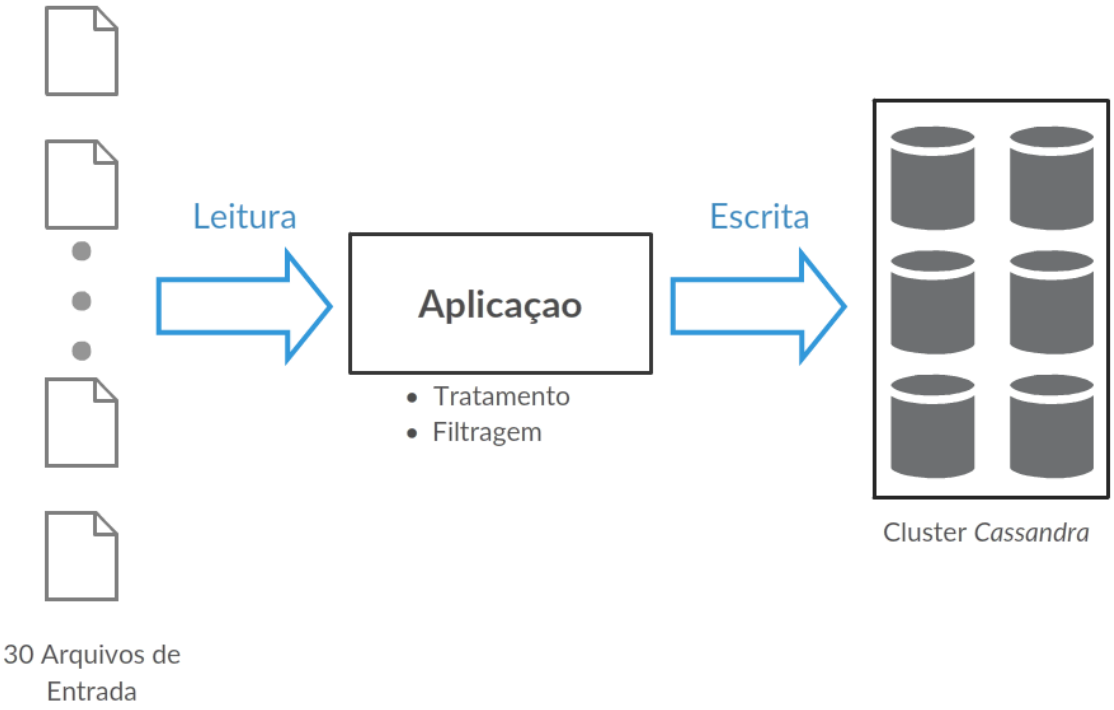

Figura 5.3: Esquema da Aplicação.

Capítulo 6

Resultados

Neste capítulo são apresentados os resultados obtidos nos testes de carga e busca dos dados. Os testes foram realizados em três configurações do *cluster*, com duas, quatro e seis máquinas, a fim de se analisar a melhora do desempenho do SGBD Cassandra ao se aumentar o número de nós. Além disso, em cada configuração de *cluster*, foram realizados testes com dois volumes distintos de dados.

Em cada configuração de teste foram realizadas dez repetições, tanto na inserção quanto na busca, a fim de garantir resultados mais consistentes.

6.1 Carga de Dados

A carga de dados foi realizada a partir da leitura dos arquivos *.csv* de dados do Bolsa Família, com seleção das colunas a serem utilizadas e tratamento de alguns campos. Os dados foram carregados a partir de uma única máquina utilizando a aplicação desenvolvida, sendo o *Cassandra* responsável pela distribuição dos mesmos dentro do *cluster* por meio do particionador *Murmur3Partitioner*. Foi feita também uma comparação entre os tempos observados em cada uma das configurações utilizadas. Esta seção descreve em detalhes essas operações.

6.1.1 Preparação e Carga

Foi desenvolvida uma aplicação Java responsável por toda a operação da inserção dos dados com uso do *driver* disponibilizado pela *Datastax*. A aplicação realiza a leitura dos dados à partir dos arquivos .csv, filtrando apenas os campos que serão utilizados na família de colunas do banco de dados: UF, Código SIAFI Município, Nome Município, NIS Favorecido, Nome Favorecido, Fonte-Finalidade, Valor Parcela, Mês Competência.

Tabela 6.1: Volume de dados

Carga	**Tamanho**
18 meses	8,79 GB
30 meses	14,69 GB

Tabela 6.2: Tempos de inserção

Volume	**2 nós**	**4 nós**	**6 nós**
18 meses	1h	55m	52m
30 meses	2h31m	2h19m	2h06m

Para a correta inserção no banco Cassandra alguns campos tiveram seus valores tratados pela aplicação. No campo Valor Parcela foi necessária a remoção do separador de milhares(',') para se adequar ao tipo *double* do Cassandra. Para o campo Mês Competência, que originalmente segue o padrão MM/AAAA, não é suportado pelo Cassandra, foi necessária a inclusão de um dia para a correta utilização do tipo *timestamp*, sendo então armazenado no formato DD/MM/AAAA.

A carga foi realizada com dois volumes de dados, correspondentes a dezoito meses e trinta meses de informações do Bolsa Família, a fim de se analisar o desempenho do banco com diferentes números de nós ao se aumentar o volume de dados. A Tabela 6.1 apresenta o tamanho total dos dados inseridos em cada uma dessas cargas.

A inserção foi realizada de maneira assíncrona com uso de funções disponibilizadas pelo *driver* da *Datastax*. A *query* em formato CQL, exibida no Código 6.1, é montada e tem seus parâmetros substituidos, e é então executada com o comando *executeAsync* da classe *Session* do *driver*. Essa forma de inserção foi a que apresentou os resultados mais consistentes durante os testes realizados. São feitas operações em seis *threads* simultâneas, valor que apresentou o melhor resultado nos testes realizados.

Os testes foram realizados para as configurações de *cluster* com duas, quatro e seis máquinas, com objetivo de analisar uma possível melhora no desempenho em *clusters* maiores.

Código 6.1: Código CQL para inserção

```
INSERT INTO bolsa_familia.dados (uf, cod_municipio,
    ↪ nome_municipio, nis_favorecido, nome_favorecido, fonte,
    ↪ valor, periodo) VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

A Tabela 6.3 apresenta os tempos obtidos na inserção dos dados nas três configurações de *cluster* e com os dois volumes de dados. O Gráfico 6.1 apresenta os mesmo dados para melhor visualização.

Tabela 6.3: Comparativo dos tempos de inserção

Volume	2 para 4 máquinas	4 para 6 máquinas	Média
18 meses	8,70%	4,22%	**6,46%**
30 meses	8,13%	8,94%	**8,54%**

Com o aumento do número de máquinas, foi observada uma melhora média de *6,46%* quando utilizado o volume de carga correspondente a 18 meses, e de 8,54% nos dados de 30 meses, como exposto na Tabela 6.3. Esse resultado demonstra que ao se utilizar um maior volume de dados nos testes realizados, obteve-se um desempenho proporcionalmente superior do Cassandra.

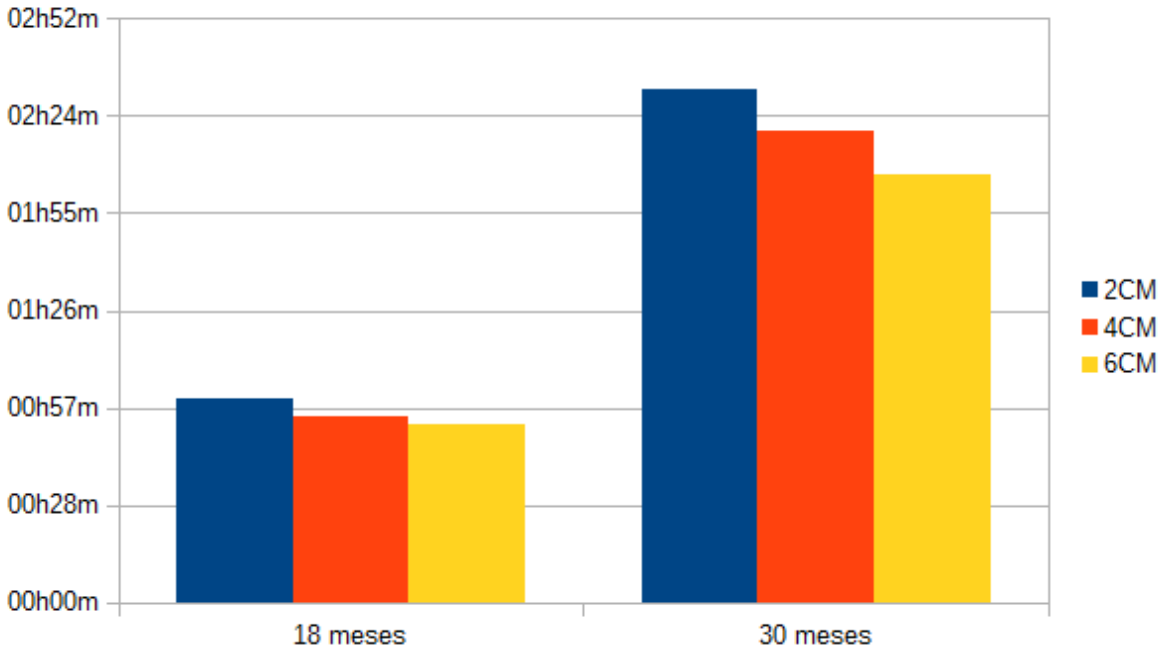

Figura 6.1: Tempos de inserção.

6.2 Extração de dados

Após cada inserção dos dados, nas três configurações de *cluster* (duas, quatro e seis máquinas), e com os dois volumes distintos de dados, foram realizadas consultas para verificar o desempenho de um banco de dados Cassandra ao se realizar extração de dados.

Tabela 6.4: Comparativo dos tempos de busca

Volume	2 para 4 máquinas	4 para 6 máquinas	Média
18 meses	81,00%	11,41%	**46,20%**
30 meses	66,21%	67,48%	**66,85%**

6.2.1 Consultas

Assim como para a carga de dados, foi desenvolvida uma aplicação Java para a realização das consultas no banco de dados, com uso do *driver* da *Datastax*. Cada teste de consulta envolveu a execução *SELECTs* no banco, sendo realizas consultas com uso da linguagem CQL a trinta registros específicos, selecionados de forma aleatória. Essa consulta, exemplificada no Código 6.2, contém toda a totalidade da chave primária da tabela, requisito imposto pelo banco de dados Cassandra. Cada teste foi repetido dez vezes a fim de se obter uma média dos resultados encontrados.

Código 6.2: Consulta CQL

```
SELECT * FROM dados WHERE nis_favorecido = 00020915229557 AND
    ↪ periodo = '2014-07-01' AND valor = 147.00
```

A Tabela 6.5 apresenta os tempos obtidos na consulta de registros nos diferentes ambientes e volumes de dados testados. O Gráfico 6.3 apresenta os mesmos dados para melhor visualização.

Com o aumento no número de máquinas, observou-se uma melhora média de 46,20% no tempo de busca utilizando-se o volume de dados correspondente a 18 meses, e de 66,85% utilizando-se o volume de dados correspondente a 30 meses. Nota-se, ainda, uma significativa melhora no tempo de busca ao se aumentar o número de máquinas de duas para quatro, tendo-se obtido uma melhora de 81% no primeiro volume de dados.

O Gráfico 6.2 apresenta os resultados das dez consultas efetuadas para cada configuração do *cluster* com volume de dados de 18 meses. Pode-se observar, principalmente na configuração com duas máquinas, que algumas execuções das consultas apresentam tempos bem distintos da média encontrada. Esse resultado pode ser explicado por variações da rede no ambiente utilizado, justificando a execução de múltiplos testes para cada configuração de *cluster* e volume de dados.

6.2.2 Comparação dos Ambientes

Nos testes com volume de dados de 18 meses do programa observou-se uma melhora de 81% ao se aumentar o número de máquinas no *cluster* de duas para quatro, e de 11,41% ao se aumentar o número de máquinas de quatro para seis, uma média de 46,20%.

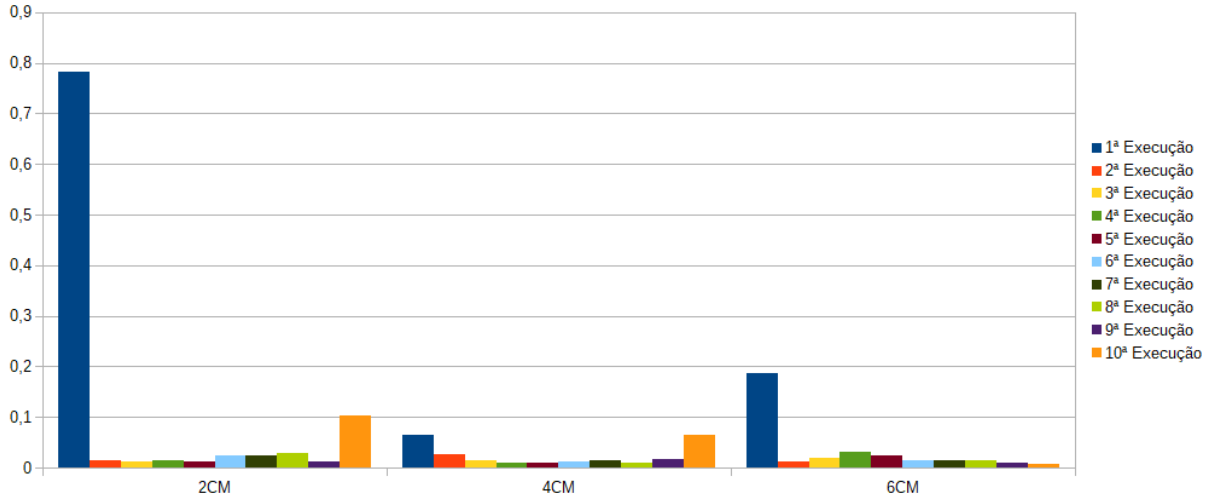

Figura 6.2: Detalhamento dos tempos de busca.

Tabela 6.5: Tempos de busca

Tamanho	**2 nós**	**4 nós**	**6 nós**
18 meses	10,26 s	1,95 s	1,73 s
30 meses	12,98 s	4,38 s	1,43 s

Para o volume de dados correspondente a 30 meses, obteve-se uma melhora no tempo da busca de dados de 66,21% ao se aumentar o número de máquinas de dois para quatro, e de 67,48% de quatro para seis máquinas, uma média de 66,85%.

Os resultados obtidos demonstram a tendência de melhora no desempenho de um banco Cassandra ao se aumentar o número de nós em um *cluster*, observando-se um resultado proporcionalmente superior ao se trabalhar com um volume crescente de dados.

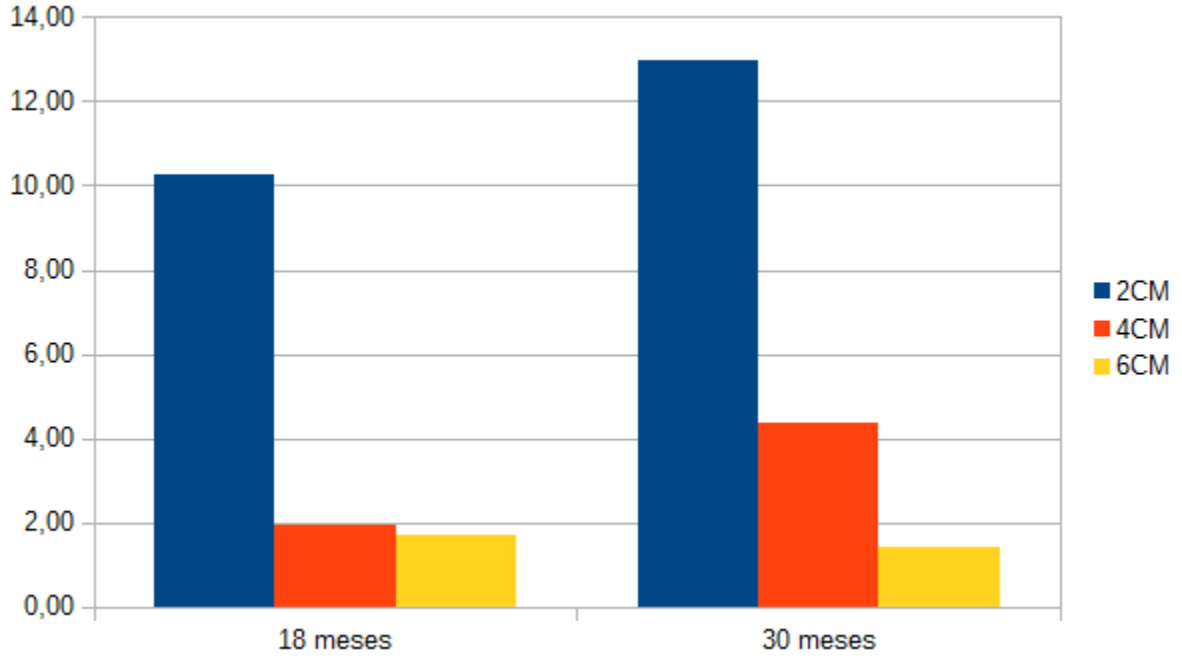

Figura 6.3: Tempos de busca.

Capítulo 7

Conclusão

Esta monografia apresentou um estudo de caso para análise do desempenho de um banco de dados não relacional, especificamente o Cassandra, no armazenamento e análise de dados públicos abertos, sendo para isso utilizados os dados dos pagamentos do programa Bolsa Família. A análise foi feita com a criação de um modelo de dados no SGBD Cassandra e posteriores testes de inserção e busca com diferentes configurações de *cluster* e volume de dados.

Foram realizados testes em três configurações diferentes de ambientes, com duas, quatro e seis máquinas, e dois volumes de dados. A inserção e a busca de dados foram realizadas por meio de uma aplicação Java desenvolvida com utilização dos *drivers* disponibilizados pela *Datastax*. Foi feita uma comparação entre os diferentes resultados de desempenho, tanto na inserção quanto na busca de dados, obtendo-se uma melhora média de 7,5% na inserção e de 56,53% na busca de dados ao se aumentar o número de máquinas no *cluster*.

Apesar dos testes com Cassandra apresentarem uma melhora de desempenho com o aumento do número de máquinas, possíveis limitações de rede no ambiente utilizado não permitiram uma análise isolada do desempenho com aumento do número de máquina. Um possível trabalho futuro poderia tentar isolar as máquinas utilizadas em uma rede dedicada, de forma a analisar o desempenho em um ambiente de servidor mais realista. Também seria interessante realizar a mesma análise com outros bancos de dados, especialmente com o uso de um banco de dados relacional, de forma a se verificar uma real vantagem de um banco de dados NoSQL para o volume de dados utilizado.

Além disso, a modelagem de um banco Cassandra, por ser baseado nas consultas dos dados, possui forte influência sobre o desempenho da aplicação. Outros trabalhos futuros poderiam explorar diferentes modelagens para as tabelas, incluindo consultas diferenciadas por modelagem.

Referências

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

[2] Bolsa família. http://www.caixa.gov.br/programas-sociais/bolsa-familia/Paginas/default.aspx. Acessado em 20 de maio de 2017. 31

[3] Companies using nosql apache cassandra. http://www.planetcassandra.org/companies/. Acessado em 30 de maio de 2016. 20

[4] 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. 23

[5] 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. 18, 24, 25, 26, 27, 28, 29

[6] Definição de conhecimento aberto. http://opendefinition.org/od/2.0/pt-br/. Acessado em 08 de abril de 2016. 1, 4

[7] Governo destina r$ 2,5 bilhões aos beneficiários do bolsa família em setembro. http://www.brasil.gov.br/cidadania-e-justica/2016/09/governo-destina-r-2-5-bilhoes-aos-beneficiarios-do-bolsa-familia-em-setembro. Acessado em 20 de maio de 2017. 31

[8] 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. 14

[9] 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. 9

[10] Introduction to cassandra query language. http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html. Acessado em 02 de julho de 2017. 30

[11] Mais de 163,8 milhões receberam benefícios sociais da caixa. http://www.brasil.gov.br/economia-e-emprego/2017/04/mais-de-163-8-milhoes-receberam-beneficios-sociais-da-caixa. Acessado em 20 de maio de 2017. 31

[12] Manual dos dados abertos: Governo. http://www.w3c.br/pub/Materiais/PublicacoesW3C/Manual_Dados_Abertos_WEB.pdf. Acessado em 01 de julho de 2016. 9

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

[14] Partitioners. http://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architecturePartitionerAbout_c.html. Acessado em 08 de julho de 2017. 34

[15] Portal da transparência - sobre o portal. http://www.portaltransparencia.gov.br/sobre/. Acessado em 07 de julho de 2017. 31

[16] Recommended production settings for linux. https://docs.datastax.com/en/cassandra/3.0/cassandra/install/installRecommendSettings.html. Acessado em 23 de março de 2017. 35

[17] Virtual nodes in cassandra 1.2. https://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2. Acessado em 15 de novembro de 2017. 27, 28

[18] Eric Brewer. Cap twelve years later: How the "rules"have changed. *Computer*, 45(2):23–29, 2012. 14

[19] 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. 12, 16, 20, 22

[20] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 10

[21] C.J. Date. *An Introduction to Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 8 edition, 2003. 1, 10, 11

[22] 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. 9

[23] 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. 12, 20

[24] Peter Deutsch. The eight fallacies of distributed computing. http://today.java.net/jag/Fallacies.html. Acessado em 28 de maio de 2016. 14

[25] 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. 7

[26] 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. 13

[27] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based scalable network services. 31(5), 1997. 11, 15

[28] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007:1–16, 2012. 4

[29] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983. 11

[30] Jan L. Harrington. *Relational database design and implementation : clearly explained.* Morgan Kaufmann, 2009. 10, 11, 12

[31] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. pages 336–341, 2011. 16, 17, 18

[32] Carlos A. Heuser. *Projeto de Banco de Dados.* 4 edition, 1998. 10

[33] Eben Hewitt. *Cassandra: The Definitive Guide.* O'Reilly Media, 2016. 20, 21, 22, 23, 24, 25, 26, 29, 30

[34] Seiji Isotani and Ig I. Bittencourt. *Dados Abertos Conectados.* Novatec, 2015. 4, 5, 6, 8

[35] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010. 20, 22, 23

[36] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010. 12, 13

[37] 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. 10

[38] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008. 15

[39] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems.* Osborne/McGraw-Hill, 3 edition, 2003. 11

[40] Pramod J. Sadalage and Martin Fowler. *NoSQL Essencial.* Novatec, 2013. 1, 10, 11, 12, 13, 16, 17, 18

[41] Christof Strauch. Nosql databases. *Stuttgart Media University*, 2011. 12, 13, 15, 16