FACULDADE FARIAS BRITO

CIÊNCIA DA COMPUTAÇÃO

Gleidson Sobreira Leite

Análise Comparativa do Teorema CAP Entre Bancos de Dados NoSQL e Bancos de Dados Relacionais

Fortaleza, 2010

Gleidson Sobreira Leite

Análise Comparativa do Teorema CAP Entre Bancos de Dados NoSQL e Bancos de Dados Relacionais

Monografia apresentada para obtenção dos créditos da disciplina Trabalho de Conclusão do Curso da Faculdade Farias Brito, como parte das exigências para graduação no Curso de Ciência da Computação.

Orientador: Ricardo Wagner Cavalcante Brito.

Fortaleza, 2010

ANÁLISE COMPARATIVA DO TEOREMA CAP ENTRE BANCOS DE DADOS NOSQL E BANCOS DE DADOS RELACIONAIS

Gleidson Sobreira Leite

NOTA: FINAL (0 – 10): _______

Data: ____/____/_________

BANCA EXAMINADORA:

____

Me. Ricardo Wagner Cavalcante Brito

(Orientador)

____

Me. José Helano Matos Nogueira

(Examinador)

____

Me. Leopoldo Soares de Melo Júnior

(Examinador)

RESUMO

Como um dos modelos mais utilizados nas últimas décadas, o Modelo Relacional vem sendo utilizado pela maioria dos bancos de dados do mercado, sendo aprimorado cada vez mais com o passar do tempo. Porém, com o intenso crescimento da quantidade de dados com os quais as organizações têm que trabalhar e do número de usuários dos sistemas, o uso de bancos de dados relacionais têm apresentado certos fatores limitantes, principalmente devido a sua natureza estruturada que não permite muita flexibilidade ao realizar a estruturação desses dados. Assim, diversas soluções estão aparecendo no mercado com o objetivo de suprir essas deficiências, principalmente no que se refere a questões de escalabilidade e disponibilidade do sistema. Este trabalho tem como principal objetivo realizar uma análise comparativa a respeito do Teorema CAP (consistência, disponibilidade e tolerância ao particionamento) entre bancos de dados NoSQL, que surgiram como um dos representantes das soluções alternativas ao Modelo Relacional, e bancos de dados relacionais. A análise comparativa do Teorama será realizada entre um banco de dados PostgreSQL e um banco de dados CouchDB, com o intuito de mostrar quais propriedades desse teorema estão sendo flexibilizadas em prol de outras e como eles provêem soluções para disponibilizar essas propriedades. Essas análises representam a contribuição desse trabalho, provendo uma análise comparativa teórica de forma que também possibilitará a realização de futuras implementações práticas dessas análises.

SUMÁRIO

INTRODUÇÃO **7**

1. SISTEMAS DISTRIBUÍDOS **9**

2. TEOREMA CAP **11**

3. BANCO DE DADOS RELACIONAIS **13**

4. POSTGRESQL **17**

5. NOSQL **21**

6. COUCHDB **27**

6.1 MODELO DE DADOS 28

6.2 RECUPERAÇÃO DE DADOS 29

6.2.1 MapReduce *29*

6.2.2 Views *30*

6.3 CONSIDERAÇÕES FINAIS 31

7. ANÁLISE COMPARATIVA DO TEOREMA CAP ENTRE POSTGRESQL E COUCHDB **33**

7.1 CONSISTÊNCIA 33

7.1.1 Consistência no CouchDB *33*

7.1.2 Consistência no PostgreSQL *36*

7.2 DISPONIBILIDADE 38

7.2.1 Diponibilidade no CouchDB *38*

7.2.2 Disponibilidade no PostgreSQL *39*

7.3 TOLERÂNCIA AO PARTICIONAMENTO 41

7.3.1 Tolerância ao particionamento no CouchDB *41*

7.3.2 Tolerância ao particionamento no PostgreSQL *42*

7.4 CONSIDERAÇÕES FINAIS 43

8. CONCLUSÃO **45**

9. REFERÊNCIAS BIBLIOGRÁFICAS **47**

LISTA DE FIGURAS

Figura 1. Exemplo de bloqueio em uma transação relacional..19

Figura 2. Exemplo de uma transação que utiliza o modelo MVCC................................19

Figura 3. Distribuição relativa de alguns bancos de dados quanto ao Teorema CAP.....24

Figura 4. Exemplo de um documento do CouchDB...28

Figura 5. Assinaturas dos métodos do framework MapReduce da Google.....................30

Figura 6. Exemplo de replicação no banco de dados CouchDB.....................................38

LISTA DE TABELAS

Tabela 1. Comparativo entre bancos de dados relacionais e bancos de dados NoSQL...........25

Tabela 2. Resumo da análise comparativa do Teorema CAP entre bancos de dados NoSQL e PostgreSQL...44

INTRODUÇÃO

Com o passar dos anos e com a popularização da *Internet*, o aumento do número de usuários das diversas aplicações computacionais tem causado uma necessidade cada vez maior de garantir a escalabilidade em bancos de dados, ou seja, garantir que eles continuem operando à medida que recebem quantidades cada vez maiores de requisições e processamentos de dados.

Podemos citar como um exemplo o fato de muitas aplicações Web atuais estarem enfrentando o desafio de garantir os seus serviços a dezenas ou centenas de milhões de usuários distribuídos ao redor do mundo, e que vem aumentando cada vez mais.

Por causa de seu rico conjunto de recursos, capacidade de gerenciar transações e consultas, além de diversas outras características, o Modelo Relacional vem sendo amplamente utilizado pela grande maioria dos sistemas de bancos de dados nas últimas décadas.

Porém, junto com seu rico conjunto de recursos, cresce também a complexidade em construir Sistemas Gerenciadores de Bancos de Dados (SGBDs) Relacionais em ambientes distribuídos, cuja necessidade vem crescendo cada vez mais com o crescimento do volume de dados.

Motivados principalmente pela necessidade de escalabilidade em banco de dados, diversas alternativas ao Modelo Relacional surgiram para suprir as restrições relativas à complexidade ao realizar a distribuição de dados, das quais a que mais ganhou destaque foi o paradigma NoSQL (*Not only SQL*).

Grande parte dos bancos de dados NoSQL que são considerados bancos de dados não relacionais foram desenvolvidos com a idéia de que, por melhor que um banco de dados seja, ele não servirá para tudo. Assim, dependendo da demanda exigida, os bancos de dados NoSQL podem ser uma alternativa eficiente ao Modelo Relacional.

Assim, baseando-se nessa idéia e no fato de as soluções alternativas ao Modelo Relacional estarem sendo cada vez mais utilizadas no mercado, este trabalho tem como objetivo principal a realização de uma análise comparativa entre os modelos de bancos de dados Relacional e NoSQL.

Essa análise envolve a comparação de três principais características, as quais de acordo com o Teorema CAP que será abordado no capítulo específico, não poderão ser garantidas de forma absoluta e simultânea pelos modelos de dados. Dessa forma, cada modelo deverá optar por garantir de forma plena apenas duas dessas propriedades.

Assim, mostrando quais dessas propriedades cada um dos modelos analisados estarão abrindo mão em prol das outras, este trabalho também propõe uma análise entre dois bancos de dados que representarão cada um dos modelos citados.

Para atingir o objetivo proposto neste trabalho, o mesmo será estruturado da seguinte maneira:

- Capítulo 1: Este capítulo apresenta os conceitos de sistemas distribuídos, suas vantagens e desvantagens, e cita algumas características importantes ao realizar o compartilhamento de dados de forma distribuída.

- Capítulo 2: É neste capítulo onde é explicado o Teorema CAP, que é o aspecto central da análise a ser feita entre os bancos de dados relacionais e os bancos de dados NoSQL, neste trabalho.

- Capítulo 3 e 4: Esses capítulos abordam os conceitos de bancos relacionais e suas características, assim como é mais bem abordado o PostgreSQL, que é um dos bancos de dados que fará parte da análise comparativa.

- Capítulo 5: Aqui são abordados os conceitos dos bancos de dados NoSQL, fazendo uma análise comparativa com bancos de dados relacionais à medida que as características desses bancos vão sendo mencionadas.

- Capítulo 6: Este capítulo aborda o CouchDB, que foi escolhido como representante dos bancos de dados NoSQL, onde serão mencionados aspectos como seu modelo de dados, forma de recuperação dos mesmos, entre outros.

- Capítulo 7: É neste capítulo que é realizada a análise comparativa entre o banco de dados PostgreSQL e o banco de dados CouchDB, objeto de estudo desse trabalho.

1. SISTEMAS DISTRIBUÍDOS

Um sistema distribuído, segundo Coulouris (2007), é aquele no qual os componentes localizados em diferentes computadores, interconectados através da rede, se comunicam e coordenam suas ações através do envio de mensagens possibilitando o compartilhamento de recursos.

Uma das principais motivações para a construção e uso de sistemas distribuídos é, de acordo com Coulouris (2007), o desejo de compartilhamento de recursos, onde estes podem ser gerenciados por servidores e acessados por clientes, ou encapsulados como objetos e acessados por outros objetos clientes.

Dentre as vantagens existentes no uso de sistemas distribuídos podemos citar a heterogeneidade, que possibilita aos usuários o acesso a serviços e a execução de aplicativos por meio de um conjunto heterogêneo de computadores e redes; a escalabilidade, que possibilita um aumento significativo do número de usuários e recursos na rede; e o tratamento de falhas, onde a ocorrência de falhas em alguns componentes do sistema não causa a falha completa do sistema (COULOURIS, 2007).

Apesar de um sistema distribuído possibilitar o compartilhamento de recursos entre computadores que estejam separados por grandes distâncias, existem diversas desvantagens, dentre as quais podemos citar a inexistência de um relógio global, do qual os programas dependem para que haja uma noção compartilhada de tempo em que as ações dos programas ocorrem, e a segurança, pelo fato de a rede estar sujeita (vulnerável) a ataques de negação de serviço (COULOURIS, 2007).

Com relação ao compartilhamento distribuído de dados, algumas características devem ser garantidas ao se realizar a distribuição de dados. Dentre as características que um sistema computacional distribuído deve garantir podemos citar a consistência, a disponibilidade e a tolerância ao particionamento, onde, segundo Eric Brewer em seu teorema CAP que será abordado mais adiante, um sistema distribuído só pode garantir duas dessas características.

Sistemas distribuídos normalmente costumam ser fortemente consistentes ou disponibilizar alguma forma de prover consistência de modo mais fraco. Um exemplo bem conhecido de forte consistência em sistemas de banco de dados é o padrão ACID (*Atomicity, Consistency, Isolation, Durability*) que é comumente associado a bancos de dados relacionais,

assunto que será mais bem abordado no capítulo de bancos de dados relacionais (BRITO, 2010).

Como exemplo de consistência que, como citado por Brito (2010), entra em contraste com o paradigma ACID por possuir um caráter eventual, podemos citar o paradigma BASE (*Basically Available, Soft state, Eventual consistency*). A característica eventual que é utilizada em bancos de dados NoSQL, cujo assunto será abordado mais adiante, permite que o sistema seja basicamente disponível sem a necessidade de estar consistente o tempo todo.

2. TEOREMA CAP

No Simpósio sobre princípios da computação distribuída (*Principles of Distributed Computing* PODC) que ocorreu em julho de 2000, o professor, Eric Brewer, da Universidade da Califórnia realizou uma afirmação que mais tarde, no ano de 2002, foi comprovada por Seth Gilbert e Nancy Lynch (2002) surgindo assim o teorema de Brewer (BREWER, 2000).

Segundo a hipótese de Brewer (2000), em um sistema computacional distribuído é desejável obter consistência, disponibilidade e tolerância ao particionamento. Porém, no mundo real, essas propriedades são impossíveis de serem obtidas ao mesmo tempo. De acordo com esse teorema, que ficou conhecido como teorema CAP (*Consistency, Availability e Partition Tolerance*), um sistema distribuído só pode garantir apenas duas dessas propriedades simultaneamente, conforme é discutido a seguir.

Para que um sistema seja considerado consistente, deve-se garantir que, após a realização de uma determinada transação, o sistema deve manter a integridade dos dados, ou seja, a transação não pode quebrar as regras do banco de dados (WEI, 2009).

A disponibilidade que, segundo Browne (2009), deve garantir que um sistema esteja sempre fornecendo acesso e funcionalidades a seus usuários, é uma propriedade indispensável para sistemas de empresas que precisam disponibilizar seus serviços continuamente, por exemplo, empresas como a Google ou a Microsoft.

A tolerância ao particionamento está relacionada à habilidade de um sistema continuar operando após ocorrerem particionamentos na rede. Assim, se um sistema disponibilizar essa propriedade, ele ainda será capaz de realizar operações como leitura e escrita mesmo após serem realizados diversos particionamentos na rede (GILBERT, 2002).

Porém, segundo Orend (2010), um sistema que disponibiliza a propriedade de tolerância a particionamento somente pode prover forte consistência com a realização de alguns cortes em sua disponibilidade. Isso ocorre porque ele deve garantir que cada operação de escrita só termine se o dado for replicado para todos os devidos nós, o que pode ser inviável em situações em que ocorreram quedas de conexão ou outros problemas de hardware, por exemplo.

A tentativa de se garantir as três propriedades mencionadas anteriormente acarretaria, segundo Monson-Haefel (2009), em um aumento significativo dos custos computacionais,

assim como o aumento da complexidade, porém ainda não haveria garantia do alcance do efeito desejado.

Assim, se os dados de um determinado sistema devem estar distribuídos e disponíveis, alcançar a sua consistência poderia ser extremamente cara ou até mesmo impossível. Assim como, se um sistema deve ser distribuído e consistente, eventualmente ele terá problemas de performance e indisponibilidade (MONSON-HAEFEL, 2009).

Como exemplo de sistemas que priorizam as propriedades de consistência e disponibilidade, de acordo com Fox e Brewer (1999), podemos citar os bancos de dados associados ao Modelo Relacional, o que só seria possível com a não existência de particionamentos na rede, e que causa a separação de seus nós.

Na prática, muitas aplicações tendem a disponibilizar as propriedades de consistência e disponibilidades de forma reduzida ao prover seus serviços. Como um exemplo de sistemas de bancos de dados distribuídos com consistência reduzida, podemos citar, segundo Fox (2009), o Bayou, que é um sistema que suporta compartilhamento de dados entre dispositivos móveis.

Quanto a bancos de dados que disponibilizam uma consistência eventual por adotar a disponibilidade em prol da consistência podemos citar os bancos NoSQL que serão abordados mais adiante.

3. BANCO DE DADOS RELACIONAIS

Um banco de dados é uma coleção de dados inter-relacionados que tem como objetivo atender as necessidades de um conjunto de usuários, apresentando algum grau de coerência lógica entre seus componentes que podem ser armazenados sob alguma forma física (KROENKE, 1999).

Constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados, um Sistema Gerenciador de Banco de Dados (SGBD) tem como principal objetivo, segundo Silberschartz (1999), proporcionar um ambiente conveniente e eficiente para recuperar e armazenar informações em um banco de dados, permitindo uma facilitação quanto à manutenção de dados possibilitando seu compartilhamento por diversas aplicações.

Uma das principais contribuições dos primeiros Sistemas Gerenciadores de Bancos de Dados foi a introdução da separação entre dados armazenados e a descrição de sua estrutura, que é feita através de esquemas, onde restrições sobre tipos de dados e esquemas constituem o modelo de dados de um sistema (RICARTE, 2008).

O Modelo Relacional surgiu como sucessor dos modelos hierárquico e de rede e tem sido utilizado pela grande maioria dos Sistemas Gerenciadores de Bancos de Dados tornando-se um padrão para os mesmos, onde como exemplo podemos citar o SQL Server, o ORACLE, o PostgreSQL, dentre outros (BRITO, 2010).

Considerado um grande marco na evolução dos modelos de dados, o modelo de dados relacional oferece uma visão do conjunto de dados como um conjunto de tabelas (ou relacionamentos), onde cada tabela é composta por múltiplas linhas (ou tuplas) e colunas (ou atributos), possibilitando a realização de consultas e manipulação de bases de dados utilizando linguagens independentes de sistema como o SQL (*Structured Query Language*) (SILBERSCHARTZ, 1999).

Incluindo a possibilidade de os dados em uma base de dados serem estruturados conforme o Modelo Relacional, uma outra característica importante desse modelo, segundo Silberschartz (1999), é a utilização de restrições de integridade que, mais comumente, envolve o uso de chaves e garante que a integridade dos dados seja mantida.

Essas chaves podem ser divididas em chaves primárias, que possibilitam a identificação de forma única das tuplas (linhas) de uma tabela, assim como a criação de

índices que possibilitam ganho na performance de consultas, e chaves estrangeiras, que permitem o estabelecimento de relações entre tabelas distintas (SILBERSCHARTZ, 1999).

Os SGBDs Relacionais possuem diversos recursos que os colocam em uma situação de destaque com relação aos diversos tipos de ambientes computacionais, assim como vem facilitando a vida de desenvolvedores de aplicações de forma que eles passaram a se preocupar mais com suas aplicações deixando a preocupação com outros aspectos como verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas e segurança para os SGBDs (SILBERSCHARTZ, 1999).

Com relação a transações em SGBDs relacionais que, segundo Ramakrishnan (2008), são execuções de programas de usuários que são vistas pelos SGBDs como um séries de operações de leitura e escrita, quatro propriedades importantes devem ser garantidas, com o objetivo de manter dados mediante acesso concorrente e falhas no sistema.

Essas propriedades que, como mencionado anteriormente, são comumente associadas a bancos de dados relacionais, são a atomicidade, a consistência, o isolamento e a durabilidade, e, de acordo com Ramakrishnan (2008), são muitas vezes referidas pelo acrônimo ACID (*Atomicity, Consistency, Isolation, Durability*).

A propriedade de atomicidade, que é garantida por um SGBD ao desfazer transações incompletas, determina que uma determinada transação deve ser executada por completo ou não ser executada (RAMAKRISHNAN, 2008).

O isolamento é uma propriedade responsável por assegurar que, mesmo que as ações de diversas transações sejam executadas de forma intercalada, o resultado dessas execuções deve ser idêntico a executar cada transação uma após a outra, ou seja, de forma serial (RAMAKRISHNAN, 2008).

A consistência do banco de dados é a propriedade em que toda transação enxerga uma instância íntegra do banco de dados, ou seja, quando um determinado usuário executa uma transação, ele deve garantir a integridade do banco de dados (RAMAKRISHNAN, 2008).

Por fim, após diversas transações terem sido realizadas com sucesso, um SGBD relacional deve assegurar que o efeito dessas transações persista, mesmo que o sistema falhe antes que as alterações resultadas dessas transações sejam refletidas em disco, propriedade conhecida por durabilidade (RAMAKRISHNAN, 2008).

Para sistemas distribuídos, segundo Orend (2010), o termo durabilidade pode ser estendido pelo fato de que alguns bancos de dados não só garantem que dados que foram "confirmados" (*committed*) sejam persistentes em uma única máquina, mas também garantem que esses dados sejam replicados em um determinado número de máquinas ou em diferentes centros de dados.

Apesar das vantagens disponibilizadas pelo uso de SGBDs relacionais, em cenários nos quais é necessário o gerenciamento de grandes volumes de dados conciliando-se com a demanda cada vez mais freqüente por escalabilidade, a utilização de SGBDs relacionais já não é considerada tão eficiente (BRITO, 2010).

Devido a sua natureza estruturada, onde normalmente a estrutura de seus dados é predefinida pelo esboço das tabelas assim com nomes e tipos das colunas, começou-se a perceber a dificuldade na organização de grandes volumes de dados no Modelo Relacional em sistemas que trabalham com o particionamento desses dados (BRITO, 2010).

Seja, por exemplo, uma situação em que uma determinada aplicação Web de uma grande empresa esteja sendo executada em um banco de dados relacional. No momento em que o número de usuários começa a crescer consideravelmente faz-se necessário tornar o banco de dados relacional cada vez mais escalável, de forma que ele suporte o constante aumento do número de usuários dessa aplicação.

De acordo com Leavitt (2010), pode-se tornar um banco de dados relacional mais escalável executando-o em computadores mais poderosos, ou seja, através da técnica de escalonamento vertical em um servidor de bancos de dados, onde seria feita uma atualização (*upgrade*) do mesmo. Porém em algum momento, como por exemplo, uma situação em que o volume dos dados que um servidor deverá gerenciar se tornar muito grande poderá ser necessária a distribuição do banco em diversos servidores, situação em que o uso de bancos de dados NoSQL poderia ser mais viável.

O problema, segundo Leavitt (2010), é que bancos de dados relacionais não foram construídos com a finalidade de funcionar com particionamento de dados, tendo um dos motivos o fato desses bancos não trabalharem muito bem de maneira distribuída por causa da dificuldade de junções (*join*) entre suas tabelas.

Assim, parcialmente em resposta ao crescimento das limitações oferecidas pelo Modelo Relacional, tem-se concentrado cada vez mais, segundo Brito (2010), um maior foco

em soluções não relacionais, como, por exemplo, o uso de bancos de dados NoSQL, assunto que será comentado mais adiante.

4. POSTGRESQL

Como mencionando anteriormente, este trabalho tem como objetivo fazer uma análise comparativa do Teorema CAP entre bancos de dados relacionais e bancos de dados NoSQL, onde, como exemplo, o PostgreSQL foi escolhido como o representante dos bancos de dados relacionais.

Dentre algumas das razões dessa escolha podemos citar o fato de ser gratuito e possuir seus códigos fonte disponíveis e abertos, sendo também, de acordo com Barvick (2006), considerado um dos bancos de dados de código aberto disponíveis no mercado que possui o conjunto mais complexo de recursos, assim como o fato de ser um banco de dados com grande potencial para estender suas funcionalidades.

De acordo com Barvick (2006), o PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional que é baseado no POSTGRES, que foi desenvolvido na Universidade da Califórnia (Berkeley), e que, a partir de 1996, passou a possuir uma versão estável.

Apesar de o Modelo Relacional ter substituído com sucesso alguns modelos anteriores, principalmente devido a sua simplicidade, acabou tornando difícil a implementação de certas aplicações, onde, como exemplo, podemos citar as situações de aplicações Web distribuídas mencionadas anteriormente, casos em que é exigida uma maior escalabilidade dos bancos.

Assim, embora o PostgreSQL se encontre na mesma situação dos bancos de dados relacionais quanto a questão da necessidade de alta escalabilidade, ele oferece um substancial poder adicional devido a incorporação de certos conceitos, como a possibilidade de extensão de suas funcionalidades, e que permitem ao usuário estender o sistema.

Por ser um banco de dados relacional ele também garante as propriedades ACID ao realizar suas transações, assim como possui suporte a transações complexas, gatilhos (*triggers*), visões (*views*), regras (*rules*), integridade relacional, e, para situações de acesso concorrente, ele também realiza controle de concorrência e de transações (BARVICK, 2006).

O fato do PostgreSQL possuir grande potencial para estender suas funcionalidades, refere-se a seu mecanismo de extensibilidade, onde, segundo Queiroz (2002), ele disponibiliza aos usuários a possibilidade de criação de tipos de dados, funções, operadores, entre outros.

Como citado em Puchalski (2005), o PostgreSQL também dá suporte aos conceitos de objetos, OIDs (Object Identifiers), objetos compostos, assim como herança múltipla entre as

tabelas, onde ele também permite a utilização de linguagens procedurais como Perl e Python, além do suporte a linguagem SQL.

De acordo com Puchalski (2005), OIDs são valores gerado pelos sistemas gerenciadores de bancos de dados, e que são associados a objetos em um banco de dados de forma que o identifique e diferencie de outros objetos, onde esse valor é único durante toda a sua existência.

Outra aspecto importante do banco de dados PostgreSQL é a sua portabilidade, onde, de acordo com Queiroz (2002), atualmente existem versões para quase todos os sistemas operacionais disponíveis no mercado.

Essas funcionalidades, dentre outras, colocam o PostgreSQL dentro de uma categoria referida como objeto-relacional, como mencionado anteriormente. Assim, embora esse banco possua funcionalidades de orientação a objetos, ele está firmemente ligado ao Modelo Relacional.

Com respeito a ambientes distribuídos, a manutenção de várias cópias de dados em locais diferentes, ou seja, a replicação de dados, é algo de extrema importância e que possibilita um balanceamento de carga do sistema, assim como possibilita a existência de *backups* dos dados.

Para a realização da replicação de seus dados, os bancos de dados PostgreSQL utilizam ferramentas que são responsáveis pela manutenção dos dados que são replicados, onde, como exemplo, podemos citar o Postgres-R e o Mammoth PostgreSQL + Replication (PEDRONI, 2006).

Como utilizado pela maioria dos bancos de dados distribuídos e por alguns bancos de dados relacionais, o PostreSQL também utiliza um método de controle de concorrência chamado MVCC (*Multiversion Concurrency Control*) que aumenta o grau de concorrência do sistema e representa um dos principais fatores de aumento de escalabilidade oferecido por esse modelo.

O MVCC, que pode ser traduzido como controle de concorrência multi-versão, é, segundo Orend (2010), um mecanismo eficiente que permite que vários processos acessem um determinado dado paralelamente sem corrompê-lo, e sem oferecer a possibilidade de ocorrerem *deadlocks*.

Assim, ao invés de permitir que um processo que queira acessar um determinado dado tenha que fazê-lo de forma exclusiva e por um determinado período de tempo, como é feito na abordagem baseada em bloqueios onde, para acessar um determinado dado, um processo deve primeiro fazer um LOCK (bloqueio ou travamento) do mesmo, o MVCC permite que diferentes processos acessem esse dado paralelamente (OREND, 2010).

As figuras abaixo, retiradas de Eloy (2009), mostram como é realizado o processo de bloqueio em uma transação relacional, e como é realizada uma transação seguindo o modelo MVCC que não só é utilizado pelo PostgreSQL, como também é utilizado pelo CouchDB, que também será comentado mais adiante.

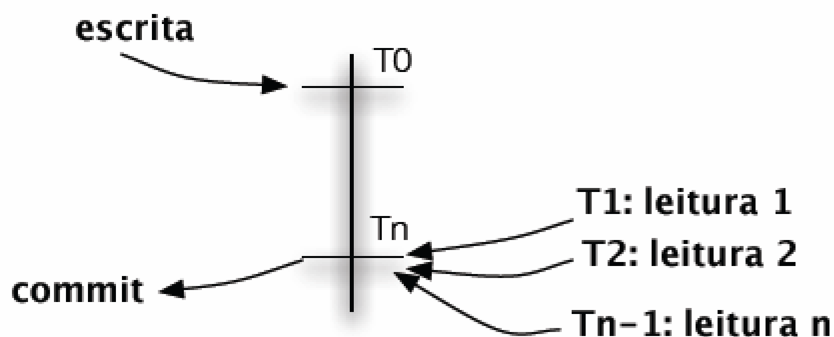

Figura 1: Exemplo de bloqueio em uma transação relacional

Como mostrado na Figura 1, um cliente que tenha requisitado um determinado recurso, obtém o acesso a ele bloqueando-o para outras leituras e escritas, onde outros clientes terão que aguardar até que o recurso tenha sido liberado para obter uma versão atualizada do mesmo. Essa é a abordagem tradicional utilizada em diversos bancos de dados relacionais existentes no mercado.

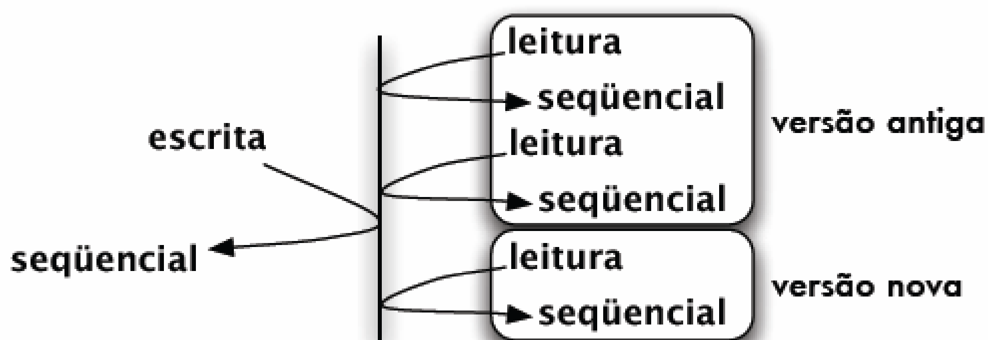

Figura 2: Exemplo de uma transação que utiliza o modelo MVCC

A Figura 2 ilustra uma transação em que as escritas e leituras são feitas de forma paralela permitindo aumentar a capacidade de atender um número maior de requisições pelo fato de não existir mais a necessidade de os clientes terem de esperar para efetuar a leitura dos dados. De uma maneira geral, isso ocorre porque no momento em que um processo deseja realizar a atualização (escrita) de um determinado item de dados, o mecanismo promove a geração de uma nova versão desse item, a qual é passada ao processo. A versão original do item continua existente e disponível para que outros processos também possam ler.

Mais adiante, na análise comparativa entre banco de dados PostgreSQL e o banco de dados CouchDB, será informado como o banco de dados PostgreSQL provê consistência em um ambiente multi-usuário utilizando o MVCC, assim como o CouchDB, que como banco de dados orientado a documentos, também utiliza o MVCC.

Com respeito ao aumento da performance na realização de consultas, o PostgreSQL, por ser um banco de dados relacional, também utiliza índices sobre as tabelas do banco de dados, estando sujeito assim as mesmas restrições existentes no Modelo Relacional com relação a distribuição e particionamento dos dados.

5. NOSQL

Com o objetivo de se propor soluções alternativas ao uso do Modelo Relacional, tendo como um dos principais motivos a estrutura pouco flexível utilizada nesse modelo, diversos projetistas de bancos de dados de grandes organizações passaram a desenvolver novas estratégias de desenvolvimento onde pudessem flexibilizar certas estruturas e regras existentes em bancos de dados relacionais (BRITO, 2010).

Embora baseado em uma arquitetura relacional, em 1998 surgiu o termo NoSQL (*Not only SQL*) a partir de uma solução de banco de dados que não disponibilizava uma interface SQL, onde, segundo Brito (2010), posteriormente esse termo passou a representar soluções caracterizadas como uma alternativa ao já bastante utilizado e consolidado Modelo Relacional.

Apesar de as primeiras implementações de bancos de dados NoSQL terem sido realizadas por grandes empresas privadas como a Amazon, ao armazenar seus dados corporativos em bancos de dados não relacionais como o Dynamo, a maioria dos bancos de dados NoSQL atuais são, segundo Leavitt (2010), *open source*, ou seja, possuem seu código aberto.

Com relação aos modelos de bancos de dados NoSQL, as categorias básicas, de acordo com Brito (2010), são:

- Sistemas baseados em amarzenamento chave-valor que, segundo Leavitt (2010), são sistemas que armazenam valores indexados por chaves para posterior recuperação;

- Sistemas orientados a documentos, que armazenam e organizam os dados como coleções de documentos ao invés de tabelas estruturadas como é feito no Modelo Relacional (LEAVITT, 2010).

- Sistemas orientados a colunas, onde a principal diferença de bancos de dados relacionais é que os dados são armazenados em colunas ao invés de tuplas (linhas).

- Sistemas baseados em grafos, cujos dados são armazenados em nós de um grafo e as associações entre os dados são representadas através de suas arestas.

Como exemplo de sistemas baseados em armazenamento chave-valor, podemos citar o Dynamo da Amazon, já mencionado anteriormente, cuja característica básica, segundo Decandia (2007), é ser um banco de dados não relacional de alta disponibilidade.

O Cassandra que foi utilizado pelo site Facebook cujo projeto tinha como objetivo ser um banco de dados distribuído de alta disponibilidade, é um exemplo de sistema orientado a colunas (LAKSMAN, 2009).

Para sistemas baseados em grafos temos como exemplo, como citado por Brito (2010), o Neo4j e o InfoGrid, enquanto nos casos de sistemas orientados a documentos temos o MongoDB e o Apache CouchDB, cujo assunto será abordado no capítulo sobre CouchDB.

Apesar de possuírem algumas características em comum como, por exemplo, uma maior escalabilidade em relação a bancos de dados relacionais, alta disponibilidade e serem livres de esquema, os bancos de dados NoSQL possuem algumas singularidades como a distribuição de dados (BRITO, 2010).

Apesar da maioria das soluções NoSQL ser distribuída, como é o caso de alguns dos exemplos citados anteriormente, existem certos sistemas, como o MongoDB ou CouchDB, que promovem o particionamento e a replicação dos dados, enquanto outros, segundo Brito (2010), podem deixar essa tarefa para o cliente, como é o caso do Amazon SimpleDB que também é livre de esquema e distribuído.

Outra característica importante existente em bancos de dados NoSQL é a questão da consistência. Como mencionado anteriormente, bancos de dados NoSQL disponibilizam uma consistência eventual por priorizarem a disponibilidade em detrimento da consistência, prioridades que, de acordo com o teorema de Brewer, não são possíveis de serem obtidas de forma absoluta e simultânea com a tolerância ao particionamento.

A consistência eventual, segundo Orend (2010), não garante que dois ou mais processos enxerguem a mesma versão de um determinado item de dado ao mesmo tempo, comportamento esse que é normalmente causado pela replicação de dados em diversos nós.

Assim, com a consistência eventual não se pode garantir que uma escrita realizada em um banco de dados em um dado momento ocasione leituras atualizadas desses dados por outros processos.

Apesar de que a consistência muitas vezes não seja garantida de forma absoluta em bancos de dados NoSQL, como no caso do CouchDB que garante somente uma consistência eventual, ao optar pela sua utilização em substituição aos SGBDs relacionais, um administrador de bancos de dados permitiria ao seu sistemauma maior flexibilidade, disponibilidade e performance, devido ao fato de os dados estarem distribuídos (ou replicados) em diversos servidores.

De acordo com Brito (2010), esse conceito de consistência eventual, que é característica de bancos de dados NoSQL, é estendido do paradigma BASE (*Basically Available, Soft state, Eventual consistency*) entrando em contraste com o paradigma ACID, já mencionado anteriormente, e que está associado a bancos de dados relacionais.

Assim, enquanto os bancos de dados que estendem do paradigma BASE garantem uma consistência de forma eventual, os bancos de dados que estão associados com o paradigma ACID garantem a propriedade de consistência de forma absoluta deixando de garantir a propriedade de disponibilidade ou tolerância ao particionamento simultaneamente e de forma absoluta (PRITCHETT, 2008).

Segundo o paradigma BASE que, de acordo com Brito (2010), se caracteriza por ser basicamente disponível, um sistema não precisa estar necessariamente em estado consistente o tempo todo, ele se tornaria consistente no devido momento.

Ou seja, ao contrário do paradigma ACID, que funciona, segundo Pritchett (2008), de forma pessimista forçando a consistência ao final de cada operação, o paradigma BASE aceita que o sistema fique, em um certo momento, em um estado inconsistente. Se por um determinado período não ocorrerem atualizações em um determinado dado, eventualmente todas as instâncias daquele dado estarão consistentes, ou seja, todos os clientes enxergarão um mesmo valor referente aquele dado.

Por conseguinte, enquanto no modelo ACID as transações garantem que o banco de dados esteja em um estado consistente, antes e após o fim de suas operações, o modelo BASE, segundo Brito (2010), permite que o banco de dados esteja eventualmente em um estado consistente.

Na Figura 3, adaptada de Orend (2010), pode ser visualizada a distribuição relativa de alguns dos bancos de dados NoSQL mencionados anteriormente, dentro das três dimensões que representam as propriedades do teorema CAP (*Consistency, Availability e Partition Tolerance*), incluindo alguns bancos de dados relacionais.

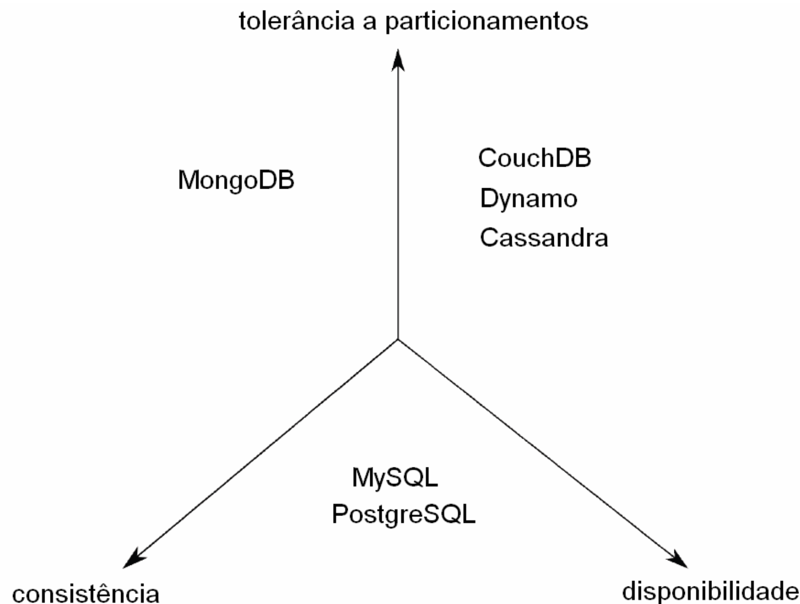

Figura 3: Distribuição relativa de alguns bancos de dados quanto ao Teorema CAP

Ainda a respeito das vantagens da utilização de bancos de dados NoSQL com relação a bancos de dados relacionais, existe a questão do escalonamento, que é inclusive um dos motivos pelos quais esses bancos foram criados.

Em cenários em que o escalonamento se faz necessário, como, por exemplo, em situações em que grandes empresas disponibilizam seus serviços de forma distribuída através de aplicações Web, de acordo com Brito (2010), os bancos de dados relacionais possuem uma estrutura pouco flexível, o que os torna menos adaptáveis para esses cenários.

São nessas situações ou cenários, cuja exigência por maior escalabilidade é maior, é que os bancos de dados NoSQL podem ser utilizados oferecendo uma maior performance, alta escalabilidade e assim suprindo as restrições existentes em bancos de dados relacionais onde, em situações como as mencionadas anteriormente, eles acabam não sendo suficientemente eficientes.

Tendo como exemplo, situações em que aplicações Web devem prover serviços para um número crescente de usuários, os responsáveis pelo sistema poderiam solucionar a questão

da escalabilidade optando pelo escalonamento vertical ou pelo escalonamento horizontal de seu sistema, sendo esta bastante utilizada em sistemas de bancos de dados NoSQL.

O escalonamento vertical é uma opção que envolve o *upgrade* do servidor, estratégia que é mais utilizada em bancos de dados relacionais. O escalonamento horizontal envolve o aumento da quantidade de servidores que disponibilizarão os dados de forma paralela, onde a grande quantidade de clientes do exemplo citado poderá ter acesso a esses dados de forma mais facilitada, garantindo também que a queda de um determinado servidor não acarrete na falta de disponibilidade dos dados a seus clientes (BRITO, 2010).

A Tabela 1 apresenta informações a respeito de comparações entre algumas características dos bancos de dados NoSQL e bancos de dados relacionais.

	Modelo Relacional	NoSQL
Consistência	Pelo fato de possuir uma estrutura mais rígida e garantir em suas transações a existência dessa propriedade, as diversas regras existentes nesse modelo possibilitam uma maior rigidez quanto a garantia de consistência das informações, considerado o ponto mais forte desse modelo.	A consistência nesse modelo possui um caráter eventual, o que não garante que uma determinada atualização, em um dado momento, seja percebida por todos os nós, mas se nenhuma atualização for realizada sobre um item de dados, os acessos a esse item retornarão seu último valor atualizado.
Disponibilidade	Devido à dificuldade de se trabalhar de forma eficiente com a distribuição de dados por causa de sua natureza estruturada, situações em que exigem uma maior demanda de um sistema que utilizem esse modelo podem não ser bem suportadas por ele.	Considerada outra das grandes vantagens do modelo, essa propriedade, junto com o alto grau de distribuição desse modelo, possibilita que o sistema fique um menor período de tempo não-disponível, assim como também permite que solicitação aos dados por um número crescente de clientes seja atendida.

Tolerância ao particionamento	Pelo fato de não terem sido construídos com a finalidade de trabalharem com particionamento de dados, bancos de dados que utilizam esse modelo não possuem um grau muito alto de tolerência ao particionamento, cuja razão principal seria a dificuldade de junções entre as tabelas.	Pela facilidade de se trabalhar de forma eficiente com a distribuição de dados, esse modelo é capaz de suportar grandes demandas de dados assim como possui alta tolerância ao particionamento dos mesmos entre os nós.
Escalonamento	Devido à natureza estrutural do modelo, o escalonamento de bancos tende a ser uma tarefa complexa, onde tem-se como um dos motivos o fato de que a adição de novos nós não é realizada de modo natural.	Pelo fato de ser livre de esquemas, esse modelo possui uma maior flexibilidade, favorecendo assim a inclusão de uma quantidade crescente de nós, onde essa alta escalabilidade é considerada uma das principais vantagens desse modelo.

Tabela 1: Comparativo entre bancos de dados relacionais e bancos de dados NoSQL.

6. COUCHDB

Considerado por Eloy (2009) uma das principais implementações de bancos de dados orientados a documentos (BDOD), sendo essa uma das razões da escolha desse banco para a análise do trabalho, o CouchDB é um banco de dados livre de esquema e distribuído, possuindo como princípio arquitetural a Web.

Como citado por Eloy (2009), o termo "Couch" é um acrônimo de "*Cluster of Unreliable Commodity Hardware*" (Conjunto de Hardware Commodity Não-Confiáveis). Esse sistema disponibiliza uma série de características que permitem a sua utilização de forma viável em *hardware commodity*, ou seja, em máquinas utilizadas como servidores que possuem *hardware* de baixa qualidade ou antigos.

Logo, mesmo quando executado em um *hardware* que esteja suscetível a falhas, o banco de dados CouchDB tem como objetivo prover uma alta disponibilidade e confiabilidade, mesmo que eventual, assim como ser extremamente escalável (ELOY, 2009).

Como outras características do CouchDB que também contribuíram para sua escolha como representante dos bancos de dados NoSQL, podemos citar a sua facilidade com programação distribuída, tolerância a falhas, e o fato de possuir seu código fonte liberado de acordo com a licença Apache v2.0, o que permite a livre modificação e uso desse código (ELOY, 2009).

De acordo com Eloy (2009), o CouchDB foi desenvolvido tendo como foco principal a Web, onde se torna acessível através de uma *Application Programming Interface* (API) denominada REST (*Representational State Transfer*), que trabalha com dados utilizando a notação JavaScript Object Notation (JSON) e utiliza o HTTP como parte integrante de sua arquitetura.

Assim, além de o CouchDB ser um banco de dados distribuído que possui um mecanismo de replicação com detecção e gerenciamento de conflitos por também implementar o MVCC, ele está disposto a receber requisições de qualquer cliente que implemente o protocolo HTTP, como, por exemplo, requisições de um *browser* (ELOY, 2009).

6.1 MODELO DE DADOS

Os dados no banco de dados CouchDB são organizados através de documentos, cada um podendo conter qualquer quantidade de atributos que podem conter listas e até mesmo objetos (OREND, 2010).

Como o CouchDB utiliza o JSON como notação para estruturar seus dados, seus documentos são, de acordo com Orend (2009), armazenados e acessados como objetos JSON, permitindo assim o suporte a cadeias de caracteres, números, datas e até mesmo listas e *hashmaps*.

O JSON é, segundo Eloy (2009), um padrão leve de intercâmbio de dados que foi projetado para facilitar a leitura e escrita de estruturas de dados, onde estas são baseadas em um subconjunto da especificação Javascript e que permite a construção de estruturas baseadas nos conceitos de coleções de pares de chave/valor e uma lista ordenada de valores.

Na Figura 4 pode-se observar um exemplo de um documento do CouchDB como um objeto JSON.

```
“Subject”: “Bancos de dados orientados a documentos”,
“Author”: “Gleidson Sobreira Leite”,
“Date”: “04/10/2010”,
“Tags”: [“BDOD”, “CouchDB”,”NOSQL”],
“Content”: “Este é um exemplo de um documento do CouchDB”
```

Figura 4: Exemplo de um documento do CouchDB

Em cada documento, como no exemplo acima, existe um identificador único e um identificador de revisão, onde este é utilizado, segundo Orend (2010), como controle de concorrência, sendo atualizado cada vez que o documento é reescrito, permitindo assim detectar se outro usuário realizou alguma alteração.

De acordo com Eloy (2009), as atualizações em documentos são feitas de forma otimista e sem *locks*, assim caso um usuário queira realizar alguma alteração em um documento ele deve primeiro carregar o documento, realizar a devida alteração e então enviar todo o documento de volta para o banco de dados.

Um outra característica interessante do banco de dados CouchDB, é a possibilidade de especificar funções de validação em Javascript em seus documentos, onde, se um determinado

documento é criado ou alterado, essas funções poderão ser utilizadas para validar essas operações no respectivo documento (OREND, 2010).

Porém, por causa da natureza distribuída do CouchDB, a validação dessas funções não poderão causar efeitos colaterais ou secundários, assim cada função em Javascript só poderá ler os documentos que lhes são passados como parâmetros (OREND, 2010).

Dentre uma das vantagens da utilização de documentos podemos citar o fato de os dados que estejam relacionados já estarem prontos para armazenamento, ao invés de distribuídos em diferentes tabelas como é o caso de bancos de dados relacionais.

Assim, enquanto que em um banco de dados relacional os dados precisam estar devidamente modelados antes de serem utilizados, em um banco de dados orientado a documentos, os documentos já estão prontos para agregarem informações após serem criados, porém eles permitem a existência de redundância e inconsistência dos dados sem que exista uma camada semântica que permita o acesso à somente uma versão dos dados (ELOY, 2009).

6.2 RECUPERAÇÃO DE DADOS

Segundo Orend (2009), o modelo de consultas do CouchDB consiste no conceito de *Views*, que são construídas utilizando funções de *MapReduce*, e a API de consulta HTTP, que permite os usuários acessarem e consultarem as *Views*.

6.2.1 MapReduce

O *MapReduce* é, de acordo com Eloy (2009), um modelo para processamento de um grande conjunto de dados, onde sua construção teve como premissas básicas a alta escalabilidade, a possibilidade do uso de *hardware commodity*, e a possibilidade de aplicações que são construídas dentro do *framework MapReduce* poderem ser distribuídas em vários nós sem a necessidade de o desenvolvedor escrever código para realizar sincronização ou paralelização.

Esse modelo para computação distribuída consiste na especificação de uma função de mapeamento (*map*), que processa um conjunto de pares chave/valor e retorna um conjunto intermediário de pares chave/valor, e uma função de redução (*reduce*), que usa os conjuntos chave/valor retornados pela função de mapeamento para mesclar todos os valores

intermediários que possuem a mesma chave disponibilizando assim um resultado para um determinada consulta (ELOY, 2009).

A Figura 5 ilustra um exemplo retirado de Orend (2010), que mostra as assinaturas dos métodos do *framework MapReduce* desenvolvido pela Google.

$$\begin{aligned} map(key, value) &\rightarrow list(ikey, ivalue) \\ reduce(ikey, list(ivalue)) &\rightarrow list(fvalue) \end{aligned}$$

Figura 5: Assinaturas dos métodos do framework MapReduce da Google

Como mostrado na figura acima, o par (*key, value*) é um registro de um dado de entrada tendo o (*ikey; ivalue*) como um par chave/valor intermediário, e o *fvalue* um valor do resultado final.

Dentre as vantagens da utilização do modelo *MapReduce* podemos citar o fato de ele simplificar o desenvolvimento de algoritmos distribuídos escondendo como é feita essa distribuição do desenvolvedor, e auxiliar no processamento de grandes quantidades de dados por possuir uma arquitetura pronta para paralelizar o processamento dos mesmos (OREND, 2010).

6.2.2 Views

Como apresentado na seção anterior, o *MapReduce* é um modelo que ajuda na divisão de tarefas através do uso de uma função de mapeamento, que processa conjuntos de registros gerando valores intermediários, e uma função de redução, que agrupa esses registros baseando-se nas suas chaves disponibilizando uma saída para uma determinada consulta.

Com a utilização desse modelo o CouchDB realiza consulta aos dados através de uma estrutura intermediária chamada *view*, onde, segundo Orend (2010), ela é basicamente um conjunto de pares chave/valor que são ordenados por suas chaves.

De acordo com Orend (2010), as *views* são construídas através de funções *MapReduce* especificadas pelos usuários, onde estas são chamadas e incrementadas toda vez que um documento do CouchDB é criado ou alterado.

Como as views devem ser especificadas antes da execução, a introdução de novas *views* requer que suas funções *MapReduce* sejam invocadas para cada documento no CouchDB, sendo esse um dos motivos de ele não suportar consultas dinâmicas (OREND, 2010).

Segundo Eloy (2009), as *views* são, para o CouchDB, uma ferramenta para consultas e relatórios de documentos, onde possuem funções objetivas como a extração de dados de documentos, construção e utilização de índices para encontrar documentos e representar relacionamentos fracos entre eles, assim como permite a realização de diversos tipos de cálculos nos dados contidos nos documentos.

Para um melhor esclarecimento do conceito de *views* em um banco de dados orientado a documentos, vamos fazer uma associação comparativa das *views* do BDOD com as tabelas de um banco de dados relacional.

Como citado em Eloy (2009), dado um conjunto de tabelas de um banco de dados com suas respectivas colunas, vamos imaginar que todas as tuplas (linhas) de cada tabela faça parte de um único documento do CouchDB, e que estejam associadas a um mesmo banco de dados sem que exista hierarquia.

No caso acima, as *views* seriam as responsáveis pela filtragem e agregação dos documentos com o objetivo de criar algo equivalente a uma tabela.

6.3 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o MapReduce, que é uma técnica de programação utilizada pelo CouchDB para recuperação de dados e para facilitar o processamento paralelo de grandes volumes de dados, e as *views* que, como mencionado anteriormente, são uma estrutura intermediária que permite a criação de programas de mapeamento e redução para filtragem e agregação de dados numa base de dados do CouchDB.

Apresentamos também o modelo de dados do CouchDB, onde os dados são representados por documentos que, ao contrario da estrutura rígida existente no Modelo Relacional, são livres de esquema, permitindo o armazenamento de dados de forma semi-estruturada, sendo também constituídos para o contexto da Web.

Com respeito às características do CouchDB relativas as propriedades como consistência, disponibilidade e tolerância a particionamentos, as mesmas serão abordadas no

próximo capítulo, onde será feita uma analise comparativa do banco de dados PostgreSQL e o banco de dados CouchDB.

Assim, o CouchDB, do mesmo modo que os bancos de dados NoSQL, oferece uma alternativa eficiente para situações que exigem uma maior distribuição dos dados e uma maior escalabilidade do banco de dados, onde, com nos exemplos de situações já mencionados anteriormente, que exigem uma maior distribuição e particionamento dos dados, o Modelo Relacional não é considerado tão eficiente.

7. ANÁLISE COMPARATIVA DO TEOREMA CAP ENTRE POSTGRESQL E COUCHDB

Este capítulo, que tem como objetivo realizar uma análise comparativa das propriedades do Teorema de Brewer entre o banco de dados PostgreSQL e o banco de dados CouchDB, será dividido em três principais tópicos, onde cada um representará uma das propriedades do teorema, indicando como cada um dos bancos citados garantem ou deixam de garantir de forma absoluta essas propriedades.

Por fim, nas considerações finais, apresentaremos uma tabela de resumo das propriedades que foram analisadas nesse estudo.

7.1 CONSISTÊNCIA

Como mencionado anteriormente, um sistema distribuído é um sistema que opera em redes robustas onde os equipamentos que estão conectados nelas podem desconectar a qualquer momento.

Hoje existem diversas estratégias para gerenciar esse tipo de segmentação da rede, onde, em oposição aos sistemas de bancos de dados relacionais como o PostgreSQL , que garantem uma consistência forte, o CouchDB se diferencia por aceitar uma consistência eventual.

7.1.1 Consistência no CouchDB

Em casos em que é estritamente necessário que os usuários de um banco de dados o visualizem em um estado consistente, como é o caso de aplicações que envolvem transações bancárias, o usuário de um determinado nó (servidor do banco) teria que esperar que outros nós entrem em acordo antes de poder ler ou escrever no banco de dados, situação esta em que a consistência é priorizada em relação à disponibilidade.

Porém se um determinado administrador que esteja preocupado com a escalabilidade, por exemplo, optar por algoritmos que forçarão os usuários a esperar por acordos entre nós para que os mesmos se atualizem com os dados que foram recentemente atualizados no banco, eventualmente ocorrerão gargalos (ANDERSON, 2010).

Para essas situações o CouchDB, segundo Anderson (2010), simplifica a construção de aplicações que sacrificam uma consistência forte em prol da disponibilidade, garantindo assim uma consistência eventual em que os usuários podem atualizar um determinado nó (servidor) de um banco de dados distribuído sem ter que esperar que outros nós tenham que realizar acordos, pois o banco trataria de realizar posteriormente as devidas atualizações entre os nós.

Diferentemente de bancos de dados relacionais, que realizam diversas verificações de consistência após ocorrerem operações no banco, bancos de dados CouchDB facilitam a construção de aplicações que sacrificam a consistência garantindo uma maior performance em bancos de dados distribuídos (ANDERSON, 2010).

O CouchDB, de acordo co Anderson (2010), possui uma árvore B (*B-Tree*) como mecanismo de armazenamento, onde a mesma apresenta uma estrutura de dados ordenados que permite buscas, inserções e exclusões, de forma que esse mecanismo de armazenamento é utilizado para todo dado interno do banco, documentos e visões (views).

Como mencionado no capítulo do CouchDB, o mesmo utiliza duas funções ("map" e "reduce") sobre os documentos armazenados de forma que elas produzem pares chave-valor que podem ser inseridos no mecanismo de armazenamento (*B-Tree*) do CouchDB ordenados pelas chaves.

Assim, é possível acessar, incluir e fazer alterações em documentos e *views* na *B-Tree* do banco de dados através do uso de chaves, sendo esta uma restrição que permite ganhos de performance devido ao fato de o acesso passar a ser feito diretamente pela chave ou pela variação entre chaves, o que também permite o particionamento dos dados entre vários nós (ANDERSON, 2010).

Com relação ao controle de concorrência do acesso aos dados, como mencionado anteriormente, o CouchDB utiliza o mecanismo chamado *Multi-Version Concurrency Control* (MVCC).

Diferentemente de bancos de dados relacionais que gerenciam a concorrência ao acesso as tabelas de dados através do uso de *locks,* o CouchDB permite o acesso aos dados sem haver necessidade de um cliente ter que bloquear (*lock*) o dado (ou tabela) para si enquanto está fazendo alguma operação. Assim, se muitos clientes quiserem acessar algum dado, não há necessidade de fazer os outros clientes esperarem até que o dado seja liberado.

O uso do MVCC permite ao CouchDB funcionar em sua capacidade máxima mesmo em situações de altas cargas de acesso, devido ao fato de permitir acesso paralelos aos dados sem a necessidade de esperas de processamentos individuais (ANDERSON, 2010).

Como mencionado no capítulo do CouchDB, os documentos são armazenados nesse banco em versões, ou seja, se um cliente quiser realizar alguma alteração em um documento, ele vai ter que criar uma nova versão do documento e salvar por cima da versão anterior, da mesma forma como é feito em um repositório de controle de versões.

Para citar exemplo de como essa utilização de versões de documentos armazenados oferece uma melhoria com relação ao uso de *locks*, veja o caso de três clientes que querem acessar ou alterar um determinado documento.

Enquanto um cliente está realizando uma consulta em um documento e uma segunda requisição vinda de outro cliente queira efetuar uma alteração nesse documento, essa alteração ocasionará na criação de uma nova versão do documento onde o CouchDB simplesmente acrescenta essa nova versão no banco.

Caso ocorram novas requisições a esse documento, elas enxergaram somente a nova versão do documento, enquanto o cliente que havia feito a primeira requisição para uma consulta continua enxergando a versão anterior do documento, onde esse uso de versões permite que nenhum cliente tenha que esperar para que sua requisição seja iniciada.

Como as operações no CouchDB são realizadas dentro do contexto de um documento, ao utilizar vários nós (servidores) de bases de dados, não há a preocupação de esses nós estarem em constante comunicação pois o CouchDB alcança uma consistência eventual entres esses nós através do uso de replicação incremental (ANDERSON, 2010).

Fornecendo uma forma de manter consistência em sistemas distribuídos, onde existem vários servidores de bancos de dados, o uso da replicação incremental, de acordo com Anderson (2010), permite que as alterações em documentos sejam periodicamente copiadas entres os servidores.

Com o uso dessa replicação, é possível, segundo Anderson (2010), a construção de grupos de servidores (nós), onde cada um funciona independentemente dos outros, processando as requisições dos clientes e deixando que o CouchDB realize as sincronizações devidas posteriormente.

Em casos em que ocorrerem alterações de um mesmo documento em diferentes réplicas que se encontram em servidores distintos, o sistema de replicação do CouchDB é responsável pela detecção e resolução automática desses conflitos (ANDERSON, 2010).

Ou seja, se foi detectado que um documento foi alterado em dois servidores, esse documento será marcado indicando a existência de conflito, da mesma forma que é feito em sistemas de controle de versões, e a versão vitoriosa será salva como a mais atual deixando a outra versão salva como uma versão anterior do histórico do documento (ANDERSON, 2010).

A forma como é escolhida a versão vitoriosa em casos de conflitos poderá ser feita por um analista ou um administrador de banco de dados, por exemplo, sendo esta uma escolha que faça mais sentido para aplicação.

Assim, após feita essa detecção e resolução dos conflitos, poderá ser feita uma fusão das versões ou, dependendo da escolha mais adequada para a aplicação, a versão mais antiga poderá ser escolhida.

7.1.2 Consistência no PostgreSQL

Como mencionado anteriormente, o banco de dados PostgreSQL, assim como o CouchDB, efetua o controle de concorrência do acesso aos dados utilizando o mecanismo MVCC, que permite acesso simultâneo aos dados, porém nada impede que possam ser feitos *locks* caso necessário.

Com o objetivo de prover a consistência dos dados em um ambiente distribuído, o CouchDB, para proteger as transações de enxergarem dados inconsistentes causados por alterações de outras transações, permite que uma determinada transação enxergue uma versão do banco de dados no momento em que ela é iniciada onde, se ocorrerem alterações posteriores, essas alterações não serão vistas pela transação corrente (MACH, 2008).

Apesar de utilizar o mesmo mecanismo de controle de concorrência para garantir a consistência dos dados que o CouchDB, o PostgreSQL trabalha com esse mecanismo de forma diferente, pois, como mencionado anteriormente, a forma de armazenamento desse banco é feita atreves do uso de tabelas, enquanto o banco de dados CouchDB armazena informações em documentos.

A forma como o MVCC é implantando pelo PostgreSQL é feita, segundo Mach (2008), através da gerência de armazenamento dos dados de forma que novas alterações não acarretam a sobrescrita dos mesmos, onde ao realizar, por exemplo, uma alteração em uma linha da tabela, essa alteração não é feita diretamente.

Ao invés disso, na tupla original é feita uma cópia com a mudança desejada, e a mesma é acrescentada à tabela tendo a linha original marcada (guardando o ID da transação) como removida, e, após isso, é feita uma ligação (*link*) da tupla original para a nova versão da tupla (MACH, 2008).

Assim, são armazenadas várias versões de uma tupla que vem sendo alterada com o tempo, e versões anteriores só serão removidas quando a sua exclusão tiver sido confirmada (*commit*) após um tempo suficiente em que nenhuma transação possa ter acesso a essa versão da tupla (MACH, 2008).

Para realizar a remoção de tuplas que a um longo tempo foram marcadas como removidas, o CouchDB provê um comando chamado VACCUM, que permite o reuso do espaço alocado para tuplas que foram marcadas como removidas a um longo tempo, permitindo assim um controle sobre o espaço alocado para as diversas versões que uma tupla pode possuir (MACH, 2008).

Um outro aspecto relativo a garantia de consistência que o banco de dados PostgreSQL provê, segundo Mach (2008), é o uso de um log, Write Ahead Log (WAL), que realiza um registro das operações que são confirmadas (*commit*), de forma que se ocorrer uma queda no banco as mesmas não serão perdidas. Essa abordagem é comumente conhecida como Estratégia Adiada de Escrita.

O uso do WAL permite ao PostgreSQL, em caso de problemas nas transações ou até mesmo uma queda no banco, recuperar as mudanças confirmadas (*commit*) diretamente do log, garantindo também que o banco de dados mantenha a informação que lhe foi enviada por uma transação que realizou um *commit* da operação (MACH, 2008).

Assim, apesar de o banco de dados PostgreSQL também utilizar o MVCC como forma de controle de concorrência, ele também permite, em casos em que forem necessários o uso de *locks*, visando prover a consistência de uma forma mais absoluta em comparação da consistência eventual garantida pelo CouchDB.

7.2 DISPONIBILIDADE

Com relação à propriedade de disponibilidade, o CouchDB adota um prioridade diferente em relação ao PostgreSQL, onde este possui diversas soluções para prover essa propriedade, sendo a escolha delas dependente da situação da rede onde se encontram os servidores, ou diversos outros fatores que levam a escolha de soluções específicas.

7.2.1 Diponibilidade no CouchDB

O CouchDB objetiva prover uma alta disponibilidade em prol da consistência através do uso da replicação de dados entre os servidores de bancos de dados distribuídos, onde, como mencionado anteriormente, passa a disponibilizar uma consistência eventual.

O CouchDB usa a replicação como forma de propagar as mudanças ocorridas em um determinado documento, por exemplo, para outros servidores, de forma que os clientes possam ter acesso a alguma versão do documento requisitado (BHAT, 2010).

A Figura 6, retirada de Mach (2008), ilustra um exemplo de replicação de documentos entre servidores (nós) de bancos de dados, onde quando um determinado nó recebe alguma alteração ou entrada (*put*) de um documento, ele realiza a replicação desse documento entre outros nós.

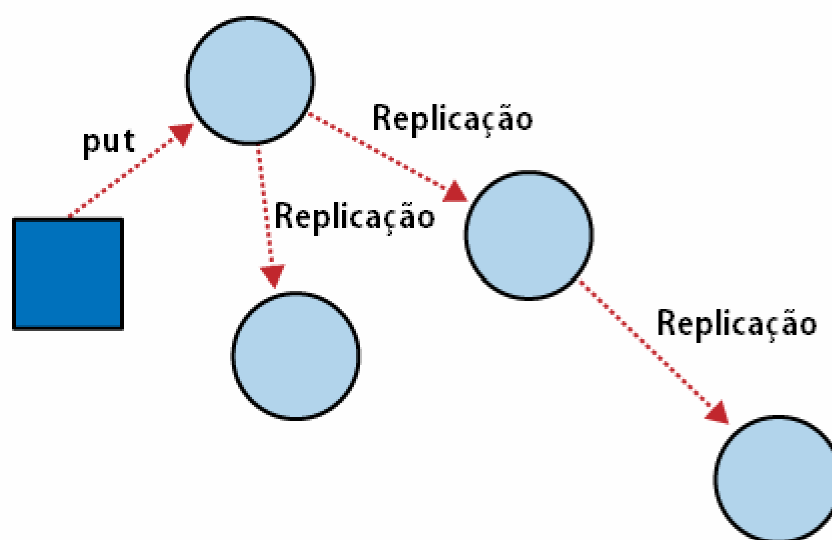

Figura 6: Exemplo de replicação no banco de dados CouchDB

Assim, o CouchDB usa a replicação dos dados tanto como forma de prover uma alta disponibilidade em um ambiente distribuído, como para, através da replicação incremental, prover uma consistência eventual dos dados (MACH, 2008).

7.2.2 Disponibilidade no PostgreSQL

Para prover a disponibilidade dos dados em um ambiente distribuído, os servidores de bancos de dados podem trabalhar em conjunto, mas para trabalhar dessa forma a maior dificuldade encontrada por servidores que trabalham em conjunto é a questão da sincronização (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Para lidar com a questão da sincronização o PostgreSQL provê diversas soluções para diminuir o impacto do problema da sincronização em um ambiente distribuído, porém nenhuma dessas soluções resolve o problema completamente, pois cada solução lida com o problema de uma forma diferente procurando minimizar o impacto desse problema em diferente situações e diferentes cargas de trabalho (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Algumas soluções, como as que realizam balanceamento de carga e as que permitem a troca automática de servidores ativos que sofreram alguma falha por outros servidores, são síncronas, enquanto soluções que permitem algum atraso entre o tempo em que uma transação foi confirmada e o tempo que essa confirmação foi propagada são chamadas de soluções assíncronas (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Soluções síncronas, segundo o *PostgreSQL Global Development Group* (2010), são aquelas em que, até que todos os servidores não tenham confirmado (*commit*) uma determinada transação que tenha realizado alguma alteração de dados, a mesma não será considerada confirmada, o que possibilita a consistência entre diferentes servidores.

Como o uso de soluções assíncronas pode acarretar na existência de dados obsoletos em alguns servidores em um dado momento pelo fato dessa solução permitir atrasos quanto à propagação de dados confirmados, a decisão quanto uso de soluções síncronas ou assíncronas vai depender de qual problema deve ser solucionado (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Para casos em que o uso de soluções síncronas acaba tornando muito lenta a comunicação entres servidores, por exemplo, uma solução assíncrona poderia ser mais adequada (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Um outro aspecto que também deve ser considerado quanto à escolha de soluções síncronas ou assíncronas de comunicação entre servidores é a questão da performance.

Por exemplo, em situações em que servidores estejam funcionando sobre uma rede com baixa taxa de transmissão, a escolha de soluções síncronas poderia afetar gravemente a performance, onde nesse caso o uso de uma solução assíncrona seria mais aconselhável por não causar um impacto tão grande na performance. Um dos motivos é o fato de possibilitar que não haja necessidade de que os todos os servidores que contenham um determinado dado confirmem uma alteração.

Assim, não há necessidade de espera dessa confirmação conjunta para que a própria transação seja confirmada.

Como exemplo de soluções assíncronas podemos citar o uso da replicação mestre-escravo (*Master-Slave Replication*), que visa mandar todas as requisições de alterações de dados para o servidor mestre, onde este envia as alterações para os servidores escravos de forma assíncrona, possibilitando apenas que os servidores escravos somente respondam a requisições de consultas dos clientes deixando as requisições de alterações para serem tratadas e distribuídas pelo servidor mestre (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Um exemplo de uma solução síncrona utilizada em bancos de dados PostgreSQL é o uso de replicação síncrona entre múltiplos servidores mestres (*Synchronous Multi-Master Replication*), que, de acordo com *PostgreSQL Global Development Group* (2010), permite que cada servidor mestre aceite requisições de alterações de dados (*write*) de forma que a alteração seja transmitida do servidor original para os outros servidores mestres que também devem confirmar a alteração.

Assim, com exceção das situações em que o uso de soluções assíncronas é priorizado, a maioria das soluções utilizadas visa garantir como prioridade a consistência, sendo esta priorizada no uso de soluções síncronas.

7.3 TOLERÂNCIA AO PARTICIONAMENTO

A tolerância ao particionamento, que é uma das propriedades que o CouchDB visa em prol da consistencia, não é garantida de forma absoluta pelo PostgreSQL, visto que o mesmo tem como pioridade prover consistência e disponibilidade dos dados.

7.3.1 Tolerância ao particionamento no CouchDB

Como mencionado anteriormente, o CouchDB é um banco de dados orientado a documentos e livre de esquema que também possui alta tolerância a particionamentos.

Atualmente uma das principais soluções para realização de particionamentos e agrupamentos (*clustering*) no CouchDB é o uso do *framework* Lounge que é, segundo Anderson (2010), um *framework* para particionamento e agrupamento baseado em proxy.

Para possibilitar a realização de particionamentos e agrupamentos no CouchDB, o Launge possui dois principais componentes, onde um (*dumbproxy*) lida com requisições simples de GET e PUT nos documentos, enquanto o outro componente (*smartproxy*) realiza as distribuições de requisições de *views* (visões) (ANDERSON, 2010).

O *dumbproxy*, segundo Anderson (2010), lida com as requisições simples que não sejam de *views* do CouchDB, enquanto o *smartproxy* é responsável apenas pelas requisições de *views,* distribuindo-as para os outros servidores de forma que realize a distribuição da carga de trabalho. Assim, o *smartproxy*, após receber o resultado da distribuição dessas requisições vinda dos outros nós, realiza o agrupamento dos resultados e retorna o resultado aos clientes.

Como mencionado anteriormente, o CouchDB permite salvar e recuperar documentos armazenados através de IDs que identificam unicamente esses documentos.

A partir desses IDs, o Lounge permite que, através do uso de um método hash (*Consistent Hashing*), possa ser determinada qual partição uma determinada requisição deve ser enviada. Ou seja, mesmo que seja efetuado o balanceamento de carga de dados entre diversos servidores distribuídos, o Lounge permite encontrar qualquer documento que tenha sido requisitado, assim como realizar a alocação de documentos nos devidos nós (ANDERSON, 2010).

Apesar de que o uso de *Consistent Hashing*, de acordo com Anderson (2010), resolva o problema do particionamento de itens de um banco de dados entres servidores distribuídos garantindo que futuras requisições possam ter acesso a esses dados, ainda existe o problema de os dados que foram armazenados sejam perdidos por alguma falha de *software* ou *hardware*.

Para lidar com esse tipo de situação, por questões de segurança, é realizada a replicação dos dados de forma que um dado não seja considerado seguro até que, segundo Anderson (2010), pelo menos possua alguma cópia em outro nó.

7.3.2 Tolerância ao particionamento no PostgreSQL

Embora a tolerância ao particionamento não seja uma propriedade que o PostgreSQL visa prover de forma absoluta, o mesmo suporta o partiocionamento básico de tabelas, que é feito através da herança (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

No PostgreSQL um tabela pode herdar de uma ou mais tabelas, onde uma consulta pode referenciar tanto as tuplas de uma tabela como as de suas tabelas filhas, e estas podem herdar de mais de uma tabela (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

Normalmente o momento ou situação que levaria o uso de uma tabela a se beneficiar do uso de particionamentos dependeria da aplicação, mas isso, segundo *PostgreSQL Global Development Group* (2010), geralmente ocorre em casos em que o uso de particionamentos não pode ser evitado. Esse tipo de situação normalmente ocorre quando a quantidade de dados que são armazenados na tabela é muito grande.

A forma como o PostgreSQL visa prover particionamentos de tabelas através do uso de herança pode ser implementada, por exemplo, através de particionamentos por variações (*Range Partitioning*) ou particionamento por listas (*List Partitioning*) (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2010).

No *Range Partitioning*, de acordo com *PostgreSQL Global Development Group* (2010), uma tabela pode ser particionada em variações das chaves de colunas ou grupos de chaves, onde, citando um exemplo, um administrador de banco de dados poderia criar uma partição por variação de datas.

O particioamento por lista é feito, segundo *PostgreSQL Global Development Group* (2010), através da especificação explícita de quais valores de chaves aparecerão em cada partição.

Podemos citar, como exemplo de como seria criado um particionamento de uma tabela através do uso de herança, o processo de criação de uma tabela mestre a partir de onde outras tabelas herdarão. Após isso segue a inserção, nessa tabela mestra, de uma trigger ou *rule* que faria o devido redirecionamento dos dados entra suas tabelas filhas (partições) a partir de regras como o *Range Partitioning*.

Assim, apesar de o PostgreSQL não visar o provimento da propriedade de tolerância ao particionamento de forma absoluta, em situações em que o particionamento é necessário ele também provê soluções que, embora possam afetar parcialmente a consistência, resolveria alguns problemas, como os casos de grandes quantidades de dados em um único servidor que atingiu seu limite de armazenamento.

7.4 CONSIDERAÇÕES FINAIS

Neste estudo foi mostrada a forma como os bancos de dados PostgreSQL e CouchDB provêm as propriedades de consistência, disponibilidade e tolerância a particionamentos, onde cada um dos bancos, apesar de que priorizem algumas propriedades em prol de outras, visam prover soluções que possibilitem também oferecer a alguma forma de conceder de forma eventual a propriedade no qual não prioriza.

Com relação a essas propriedades, enquanto o PostgreSQL visa prover a consistência e disponibilidade de forma absoluta, o CouchDB prioriza a propriedades de disponibilidade e tolerância a particionamentos, abrindo mão da consistência de forma que também possibilita altos ganhos de performance.

Assim, para uma situação onde a decisão de qual solução seria mais adequada implementar numa empresa, por exemplo, a escolha do banco de dados vai depender da necessidade da aplicação ou da forma como se objetiva disponibilizar os dados de um sistema, onde também é possível o uso das duas soluções para suprir as diversas necessidades de provimento de serviços.

A Tabela 2 apresenta um resumo da análise comparativa das propriedades do Teorema CAP entre os bancos de dados PostgreSQL e CouchDB, bancos escolhidos como exemplos para a análise comparativa proposta neste trabalho.

	CouchDB	PostgreSQL
Consistência	Garantida em caráter eventual através do uso da replicação incremental. Possui como mecanismo de armazenamento uma B-tree, onde, com o uso do MVCC, gerencia o acesso concorrente aos nós dessa árvore.	Priorizada. Apesar de também utilizar o MVCC, possibilita o uso de locks. Novas alterações nas tabelas não são sobrescritas, permitindo a existência de várias versões de uma tupla.
Disponibilidade	Disponibilizada em alto grau através do uso de replicação de dados entre servidores distribuídos. Provida de forma que os clientes têm acesso a alguma versão do dado.	Provida através da sincronização síncrona ou assíncrona entre servidores. Visa priorizá-la de forma a garantir paralelamente uma boa performance.
Tolerância ao particionamento	Possui alta tolerância. Agrupamentos e particionamentos são mais bem gerenciados com o uso de *frameworks* específicos como o Lounge. Utiliza replicação para garantir segurança de dados particionados.	Não é provida de forma absoluta. Possibilita a realização de particionamentos através da herança entre tabelas. Particionamentos são feitos em casos extremos ou situações que exigem sua realização.

Tabela 2: Resumo da análise comparativa do Teorema CAP entre bancos de dados NoSQL e PostgreSQL.

8. CONCLUSÃO

A decisão de optar pelo uso de abordagens NoSQL para suprir as limitações que os bancos de dados relacionais possuem devido sua natureza estruturada, deve levar em consideração fatores que devem ser usados como critérios relevantes e que auxiliarão na escolha de uma dessas abordagens, ou na escolha pelo uso conjunto de ambas.

Importantes pontos também devem ser levados em consideração como: a escalabilidade, a necessidade de prover consistência, e até mesmo questões de existência ou não de facilidades para realizar consultas em ambientes distribuídos.

Apesar de que as possibilidades de escolha de soluções a serem utilizadas sejam variadas, a seleção de uma solução NoSQL ou relacional não abrangeria todos os problemas, onde a escolha de um determinado banco de dados as ser usado dependeria dos requerimentos do projeto.

Neste trabalho foi apresentada uma análise comparativa entre duas abordagens que estão crescendo cada vez mais no mercado, a abordagem NoSQL, que visa disponibilizar uma outra forma de priorizar certas propriedades com o intuito de suprir limitações de outros modelos, e a abordagem relacional, continua sendo a mais amplamente utilizada no mercado.

Como um dos pontos positivos do uso do Modelo Relacional, foi possível observar que o mesmo se apresenta como uma solução mais madura, que vem sendo utilizada e aperfeiçoada há muito mais tempo que a abordagem NoSQL, enquanto esta ainda não apresenta modelos completamente definidos e padronizados, de forma que a sua utilização ainda não é tão popularizada no mercado.

Neste trabalho também foi apresentado um estudo de caso que teve como objetivo realizar uma análise comparativa de como os bancos de dados PostgreSQL e CouchDB priorizam ou deixam de priorizar as propriedades do Teorema CAP, mostrando também como cada um desses bancos procuram disponibilizar soluções para prover as propriedades que não dão uma maior prioridade.

Assim, apesar de que esses bancos não disponibilizam de forma absoluta as propriedades de consistência, disponibilidade e tolerância a particionamentos simultaneamente, cada um procura disponibilizar soluções para suprir as necessidades de um projeto.

Adicionalmente, deve-se ter em mente que, como a escalabilidade em alto grau faz-se necessária apenas em bancos de dados de grande porte, cuja disponibilidade é imprescindível, a escolha de soluções NoSQL em casos em que a disponibilidade não seja um fator essencial a ser provido ainda é uma abordagem a ser discutida e trabalhada.

Um ponto interessante a ser tratado em trabalhos futuros poderia ser a realização de uma análise prática de como cada um dos bancos que foram objetos do estudo de caso deste trabalho realizam o provimento de cada uma das propriedades do Teorema CAP.

9. REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSON, J. Chirs; LEBNARDT, Jan; Slater Noab. **CouchDB**: The Definitive Guide. 1.ed. California, O'Reilly Media, Inc., 2010.

BARVICK, Pryscila Guttoski. **Otimização de consultas no PostgreeSQL utilizando o algoritmo de Kruskal**. 2006. 105f. Dissertação (Mestrado em Informática na área de ciências exatas). Universidade Federal do Paraná, Curitiba.

BHAT, Uma; SHRADDHA, Jadhav. **Moving Towards Non-Relational Databases**. *In*: 2010 International Journal of Computer Applications (0975 – 8887) Volume 1 – No. 13, 2010.

BREWER, E. A. **Towards robust distributed systems**. (Invited Talk). Principles of Distributed Computing (PODC), Portland, Oregon, Julho 2000.

BRITO, Ricardo W.. **Bancos de Dados NoSQL x SGBDs Relacionais**: Análise Comparativa. Faculdade Farias Brito e Universidade de Fortaleza, 2010. Disponível em : < http://www.infobrasil.inf.br/pagina/anais-2010 >. Acesso em: 16 set. 2010.

BROWNE Julian. **Brewer's CAP Theorem**, 2009. Disponível em: < http://www.julianbrowne.com/article/viewer/brewers-cap-theorem >. Acesso em: 16 set. 2010.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; **Sistemas distribuídos**: conceitos e projeto. Tradução por: João Tortello. 4 ed. Porto Alegre, Bookman, 2007.

DECANDIA, Giuseppe; HASTORUN, D.; JAMPANI, M.; KAKULAPATI G.; LAKSHMAN A.; PILCHIN A.; SIVASUBRAMANIAN S.; VOSSHALL P.; VOGELS W.. **Dynamo**: Amazon's Highly Available Key-Value Store. *In*: ACM SIGOPS Operating Systems Review, Volume 41, Edição 6, Dezembro 2007, SOSP '07, p.205–220.

ELOY, Leonardo Abranques de Oliveira. **O Estado da Arte em Bancos de Dados Orientados a Documentos**. 2009. 78f. Trabalho de conclusão de curso (Graduação). Universidade de Fortaleza, Ceará.

MONSON-HAEFEL, Richard. **97 Things Every Software Architect Should Know.** Collective Wisdom from the Experts, O'Reilly Media, 1.ed., p. 116-117, 2009.

FOX, A.; BREWER, E. A.; Harvest, yield and scalable tolerant systems. In: Workshop on Hot Topics in Operating Systems, p. 174-178 , 1999.

GILBERT Seth; LYNCH Nancy, **Brewer's conjecture and the feasability of consistent, available, partition-tolerant web services**, Sigact News 33(2), 51-59, Junho 2002.

KROENKE, David M.. **Banco de dados**: fundamentos, projeto e implementação. 6.ed. São Paulo: Livros Técnicos e Científicos, 1999.

LAKSMAN A.; MALIK P.. **Cassandra - A Decentralized Structured Storage System**. LADIS, 2009. Disponível em : < http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf >. Acesso em: 21 out. 2010.

LEAVITT, Neal; **Will NoSQL Databases Live Up to Their Promise?**, Computer, vol. 43, no. 2, p. 12-14, Fev. 2010.

MACH, Robert. **Design of Integrity Check And Repair Algorithms For PostgreSQL Data File**. 2008. 92f. Proposta de Mestrado em Ciência da Computação e Engenharia. Faculdade de Engenharia Elétrica da Universidade de Praga, Praga.

OREND, Kai. **Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer**. 2010. 100f. Dissertação (Mestrado em Engenharia de Software para Sistemas de Informação Empresariais). Universidade técnica de Munique, Baviera.

PEDRONI, Lisiane M.; Ribeiro Helena G. **Replicação em Banco de Dados PostgreSQL**. Universidade de Caxias do Sul, Caxias do Sul, 2006. Disponível em : < http://www.upf.br/erbd/download/16138.pdf >. Acesso em: 1 out. 2010.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL 9.0.1 Documentation**. Disponível em : < http://www.postgresql.org/docs/9.0/interactive/index.html >. Acesso em: 15 nov. 2010.

PRITCHETT Dan. **BASE: An Acid Alternative**. ACM Queue, vol. 6, no. 3, Julho, 2008. Disponível em : < http://queue.acm.org/detail.cfm?id=1394128 >. Acesso em: 25 out. 2010.

PUCHALSKI, Nádia Kozievitch. **Dados Meteorológicos**: Um estudo de viabilidade utilizando um SGBD em plataforma de baixo custo. 2005. 70f. Dissertação (Mestrado em Informática na área de ciências exatas). Universidade Federal do Paraná, Curitiba.

QUEIROZ, Gilberto Ribeiro. **Extensão do SGBD PostgreSQL com operadores espaciais**. 2002. 53f. Proposta de Mestrado em Computação Aplicada. Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

RAMAKRISHNAN Raghu; Gehrke, Johannes; **Sistemas de Gerenciamento de Bancos de Dados**. Tradução por: Célia Taniwake; João Eduardo Nóbrega Tortello. 3.ed. São Paulo, McGraw-Hill Interamericana do Brasil Ltda, 2008.

RICARTE, Ivan Luiz Marques. **Sistemas de bancos de dados.** Faculdade de Engenharia Elétrica e de Computação, Unicamp, Campinas, 2008. Disponível em : < ftp://ftp.dca.fee.unicamp.br/pub/docs/ricarte/apostilas/mc_sbdoo.pdf >. Acesso em: 26 out. 2009.

SILBERSCHARTZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.. **Sistema de Banco de dados**. Tradução por: Marília Guimarães Pinheiro; Cláudio César Canhette. 3.ed. São Paulo, Pearson Makron Books, 1999.

WEI Zhou; PIERRE Guillaume; CHI Chi-Hung. **Scalable Transactions for Web Applications in the Cloud**. In proc. of the Euro-Par Conference, Janeiro 2009.