

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ALLAN ALBINO COLLIER

ESCALABILIDADE DE APLICAÇÕES COM BANCO DE DADOS NÃO RELACIONAL

Monografia apresentada à Faculdade
Natalense para o Desenvolvimento do Rio
Grande do Norte – FARN, como parte dos
requisitos para obtenção do grau de
Bacharelado em Sistemas de Informação.

ESCALABILIDADE DE APLICAÇÕES COM BANCO DE DADOS NÃO RELACIONAL

Orientador: Prof. Msc. Ricardo Wendell
Rodrigues da Silva

NATAL/RN
2011

ALLAN ALBINO COLLIER

ALLAN ALBINO COLLIER

ESCALABILIDADE DE APLICAÇÕES COM BANCO DE DADOS NÃO RELACIONAL

Monografia apresentada à Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte – FARN, como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

ESCALABILIDADE DE APLICAÇÕES COM BANCO DE DADOS NÃO RELACIONAL

BANCA EXAMINADORA

Monografia apresentada à Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte – FARN, como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Msc. Ricardo Wendell Rodrigues da Silveira

NATAL/RN
2011

BIBLIOTECA UNI-RN

004.655.3 C699e - RB: 37284 (M002098)

Título: Escalabilidade de aplicações com

Autoria: Collier, Allan Albino

2011, ex. 1

Setor de Processos Técnicos

Collier, Allan Albino

Escalabilidade de aplicações com banco de dados não relacional / Allan Albino Collier. – Natal, 2011.

53f.

Orientador: Ricardo Wendell Rodrigues da Silveira (Msc.)

Monografia (Bacharelado em Sistemas de Informação) – Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

1. NoSQL - Monografia. 2. Banco de Dados Relacional – Monografia. 3. Escalabilidade – Monografia. 4. Aplicações Web – Monografia. 5. Banco de Dados Não Relacional – Monografia. I. Título.

RN/FARN/BC

CDU 004.655.3

Catálogo na Publicação – Biblioteca da FARN

ALLAN ALBINO COLLIER

ESCALABILIDADE DE APLICAÇÕES COM BANCO DE DADOS NÃO RELACIONAL

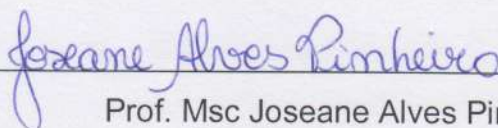
Prof. Ricardo Wendell pela paciência na orientação, incentivo e amadurecimento dos meus conhecimentos que me levaram à execução e conclusão desta monografia.

Monografia apresentada à Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte – FARN, como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em: ____/____/____

BANCA EXAMINADORA

Prof. Msc. Ricardo Wendell Rodrigues da Silveira
Orientador



Prof. Msc Joseane Alves Pinheiro
Membro examinador



Prof. Esp. Romulo Fagundes Catanhede
Membro examinador

AGRADECIMENTOS

A minha família, pois apesar de toda dificuldade sempre me deram forças para continuar batalhando e buscando uma qualificação superior. Ao meu orientador Professor Ricardo Wendell pela paciência na orientação, incentivo e amadurecimento dos meus conhecimentos que me levaram a execução e conclusão desta monografia.

Aos meus amigos e colegas, gostaria de agradecer pelo apoio que vocês me proporcionaram diante dessa caminhada.

"Seu tempo é limitado, então não percam tempo vivendo a vida de outro. Não sejam prisioneiros pelo dogma – que é viver com os resultados do pensamento de outras pessoas. Não deixe o barulho da opinião dos outros abafar sua voz interior. É mais importante, tenha a coragem de seguir seu coração e sua intuição. Eles de alguma forma já sabem o que você realmente quer se tornar. Tudo o mais é secundário."

Steven Paul Jobs

RESUMO

Atualmente o crescimento para muitas empresas pode ser medido através da quantidade de seus usuários de sistema web. Para garantir que essas aplicações estejam sempre disponíveis devem ser considerados diversos fatores, entre eles a escalabilidade. Com o crescimento exponencial de aplicações web surge a necessidade de sistemas que suportem cada vez mais uma carga maior de usuários e dados. Nesse contexto, os bancos de dados NoSQL surgem como uma alternativa eficiente para o armazenamento de dados em larga escala. Como uma alternativa eficiente para o armazenamento de dados em larga escala, os bancos de dados NoSQL oferecem uma série de vantagens em relação aos bancos de dados relacionais. Os bancos de dados NoSQL oferecem uma série de vantagens em relação aos bancos de dados relacionais, tais como: escalabilidade horizontal, alta disponibilidade, flexibilidade de esquema e desempenho superior em operações de leitura e escrita. Além disso, os bancos de dados NoSQL oferecem uma série de vantagens em relação aos bancos de dados relacionais, tais como: escalabilidade horizontal, alta disponibilidade, flexibilidade de esquema e desempenho superior em operações de leitura e escrita.

Palavras-chave: NoSQL, Banco de dados, Escalabilidade, Aplicações web, Banco de dados não relacional.

“Seu tempo é limitado, então não percam tempo vivendo a vida de outro. Não sejam aprisionados pelo dogma – que é viver com os resultados do pensamento de outras pessoas. Não deixe o barulho da opinião dos outros abafar sua voz interior. E mais importante, tenha a coragem de seguir seu coração e sua intuição. Eles de alguma forma já sabem o que você realmente quer se tornar. Tudo o mais é secundário.”

Steven Paul Jobs

RESUMO

Atualmente o crescimento para muitas empresas pode ser medido através da quantidade de seus usuários de sistemas web. Para garantir que esses sistemas estejam sempre disponíveis devem ser considerados diversos fatores, entre eles sua escalabilidade. Com o crescimento exponencial de aplicações web surge a necessidade de que os sistemas suportem cada vez mais uma carga maior de usuários e requisições, mesmo que por pequenos períodos de tempo. Quando se pensa em escalabilidade utilizando sistemas tradicionais de bancos de dados relacionais esse procedimento torna-se dispendioso e trabalhoso. Como uma alternativa eficiente para o problema de disponibilidade em banco de dados relacionais existem soluções, tais como os banco de dados não relacionais ou NoSQL. Os resultados desse estudo poderão ser utilizados como base para auxiliar na decisão entre o uso de bancos de dados relacionais ou banco de dados não relacionais, tendo como base a escolha pela escalabilidade das aplicações web.

Palavras chave: NoSQL. Banco de dados relacional. Escalabilidade. Aplicações web. Banco de dados não relacional.

LISTA ABSTRACT

Currently, the growth of many companies can be measured by its web systems concurrent users number. To ensure that these systems are always available several factors should be considered, including its scalability. With the exponential growth of web applications comes the need for systems that support an increasingly larger load of users and requests, even for short periods of time. When it comes to scalability using traditional relational databases this procedure becomes expensive and laborious. As an efficient alternative to the problem of availability in relational database other solutions exist, such as non-relational databases or NoSQL. The results of this study can be used as a basis to help decide between the use of relational or non-relational non-relational, based on the choice for availability of web applications.

Keywords: NoSQL. Relational database. Scalability. Web applications. Non-relational

database

Figura 9 – Estrutura do Map/Reduce	38
Figura 10 – Escalabilidade vertical com pequenos e grandes servidores	42
Figura 10 – Controle de concorrência: Bloqueio x MVCC	44

LISTA DE ILUSTRAÇÃO

Figura 1 – Exemplo de banco de dados no modelo de rede.	17
Figura 2 – Exemplo de banco de dados no modelo de hierárquico.	18
Figura 3 – Modelo entidade relacionamento.	19
Figura 4 – Tabela de dados no modelo relacional.	20
Figura 5 – Atributos do teorema CAP em relação a escolha dos Banco de dados...33	
Figura 6 – Exemplo simples de sharding sobre uma coleção de usuários37	
Figura 7 – Exemplo de uma configuração de sharding redundante.38	
Figura 8 – Estrutura do Map/Reduce.....39	
Figura 9 – Visão geral da utilização do Map/Reduce.....40	
Gráfico 1 – Escalabilidade vertical com pequenos e grandes servidores.....42	
Figura 10 – Controle de concorrência: Bloqueio x MVCC44	

LISTA DE TABELAS

Tabela 1	– Análise comparativa entre o modelo relacional e NoSQL.	47
-----------------	--	-----------

ACID	Atomicidade, Consistência, Indivisibilidade e Durabilidade
API	Application Programming Interface
BASE	Basically Available, Soft state, Eventual consistency
CAP	Consistency, Availability, Partition tolerance
CPU	Central Processing Unit
DBMS	Date Base Management System
ERD	Diagrama Entidade-Relacionamento
ER	Entidade-Relacionamento
ESD	Entrada e Saída
FFS	Google File System
HD	Hard Disk
JS-ON	Java Script Object Notation
MVCC	Multiversion Concurrency Control
NOSQL	Not Only SQL
RAM	Random Access Memory
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SQL	Structured Query Language

LISTA DE SIGLAS

2PC	Two Phase Commit	12
ACID	Atomicidade, Consistência, Indivisibilidade e Durabilidade	13
API	Application Programming Interface	14
BASE	Basically Available, Soft state, Eventual consistency	14
CAP	Consistência, Availability, Partition tolerance	15
CPU	Central Processing Unit	16
DBMS	Data Base Management System	18
DER	Diagrama Entidade Relacionamento	17
ER	Entidade Relacionamento	17
E/S	Entrada e Saída	19
GFS	Google File System	20
HD	Hard Disk	22
JSON	Java Script Object Notation	24
MVCC	Multiversion Concurrency Control	25
NOSQL	Not Only SQL	26
RAM	Random Access Memory	27
SGBD	Sistema de Gerenciamento de Banco de Dados	28
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional	29
SQL	Structured Query Language	29

4.3 Definição BASE	30
5 TEOREMA CAP	32
6 ESCALABILIDADE	34
6.1 Sistemas com Escalabilidade	35
6.2 Subções de Escalabilidade	35
6.2.1 Sharding	36
6.2.2 MapReduce	39
6.2.3 Escalabilidade Horizontal	41
6.2.4 Escalabilidade Vertical	42
6.3 Controle de Concorrência de Multi-versões	43

SUMÁRIO

1 INTRODUÇÃO	12
1.1 Objetivo	13
1.1.1 Objetivos Específicos	14
1.2 Justificativa	14
1.3 Organização do Trabalho	15
2 REFERENCIAL TEÓRICO	16
2.1 Banco de Dados	16
3 BANCO DE DADOS RELACIONAL	17
3.1 Modelo Relacional	17
3.2 Armazenamento de Dados Relacional	19
3.3 ACID	20
3.4 Normalização de Dados	22
4 BANCOS DE DADOS NÃO RELACIONAL	24
4.1 NoSQL	24
4.2 Tipos de Bancos NoSQL	25
4.2.1 Key Value (Chave Valor)	26
4.2.2 Wide COLUMNS (Tamanho de Coluna)	27
4.2.3 Document Store (Armazenamento de Documentos)	28
4.2.4 Graph Store (Armazenamento de Grafos)	29
4.2.5 Desnormalização	29
4.3 Modelo BASE	30
5 TEOREMA CAP	32
6 ESCALABILIDADE	34
6.1 Problemas com Escalabilidade	35
6.2 Soluções de Escalabilidade	35
6.2.1 Sharding	36
6.2.2 MapReduce	39
6.2.3 Escalabilidade Horizontal	41
6.2.4 Escalabilidade Vertical	42
6.3 Controle de Concorrência de Multi-versões	43

6.4 Escalabilidade entre RDBS e NoSQL	44
7 CONCLUSÕES E PERSPECTIVAS	46
REFERÊNCIAS	49

Atualmente o crescimento para muitas empresas pode ser medido através da quantidade de seus usuários de sistemas web, onde alguns web sites tem que atender milhões de usuários simultâneos e ao mesmo tempo sempre disponíveis sem comprometer o desempenho. Para garantir que esses sistemas estejam sempre disponíveis devem ser avaliados alguns fatores, entre eles sua escalabilidade.

Com o crescimento exponencial de aplicações web surge a necessidade de que os sistemas suportem cada vez mais uma carga maior de usuários e requisições, sendo que para pequenos períodos de tempo. Quando se pensa em escalabilidade os sistemas tradicionais de banco de dados relacionais esse procedimento tende a ser dispendioso e trabalhoso, onde como uma alternativa eficiente para o problema de escalabilidade de banco de dados relacionais existem soluções, tais como os bancos de dados não relacionais ou NoSQL (ESCALABILIDADE, 2011).

Desde a sua criação no início dos anos 70, o Modelo Relacional de dados tem sido usado em larga escala para grande maioria dos sistemas de gerenciamento de banco de dados, tornando-se padrão para a maioria dos SGBDs (Sistema Gerenciadores de Banco de Dados) tais como o SQL Server, Oracle, PostgreSQL, MySQL, etc. Onde seus elementos básicos são as relações entre os dados, as restrições de integridade e a normalização.

Porém, com o intenso crescimento do número de aplicações, soluções, recursos e tudo que se refere a sistemas computacionais, o volume de dados cresceu em um ritmo muito acelerado. De um modo geral, em cenários onde é necessário fazer o gerenciamento de um grande volume de dados, os Sistemas de Gerenciamento de Banco de Dados Relacionais, ou o próprio modelo relacional, não acompanhou o crescimento acelerado do volume dos dados e por esse motivo já não se mostra tão eficiente (BRITO, 2010).

Utilizando um SGBD relacional, com o aumento do número de usuários o sistema começa a ter uma queda em seu desempenho e para superar esse problema é

1 INTRODUÇÃO

Atualmente o crescimento para muitas empresas pode ser medido através da quantidade de seus usuários de sistemas web, onde alguns *websites* tem que atender milhões de usuários simultâneos e se manter sempre disponível, sem comprometer o seu desempenho. Para garantir que esses sistemas estejam sempre disponíveis devem ser considerados diversos fatores, entre eles sua escalabilidade.

Com o crescimento exponencial de aplicações web surge a necessidade de que os sistemas suportem cada vez mais uma carga maior de usuários e requisições, mesmo que por pequenos períodos de tempo. Quando se pensa em escalabilidade utilizando sistemas tradicionais de bancos de dados relacionais esse procedimento torna-se dispendioso e trabalhoso, onde como uma alternativa eficiente para o problema de escalabilidade em banco de dados relacionais existem soluções, tais como os banco de dados não relacionais ou NoSQL. (ESCALABILIDADE, 2011)

Desde a sua criação no início dos anos 70, o Modelo Relacional de dados tem sido utilizado em larga escala pela grande maioria dos sistemas de gerenciamento de banco de dados, tornando-se padrão para a maioria dos SGBDs (Sistema Gerenciadores de Banco de Dados) tais como o SQL Server, Oracle, PostgreSQL, MySQL, etc. Onde seus elementos básicos são as relações entre os dados, as restrições de integridade e a normalização.

Porém, com o intenso crescimento do número de aplicações, soluções, recursos e tudo que se refere a sistemas computacionais, o volume de dados cresceu em um ritmo muito acelerado. De um modo geral, em cenários onde é necessário fazer o gerenciamento de um grande volume de dados, os Sistemas de Gerenciamento de Bancos de Dados Relacional, ou o próprio modelo relacional, não acompanhou o crescimento acelerado do volume dos dados e por esse motivo já não se mostra tão eficiente. (BRITO, 2010)

Utilizando um SGBD relacional, com o aumento do número de usuários o sistema começa a ter uma queda em seu desempenho e para superar esse problema é

necessário promover uma melhora no servidor ou aumentar a quantidade de servidores. Tal melhoria seria suficiente se o número de usuário não continuasse a crescer em ritmo intenso, pois, mesmo com a melhoria de física dos servidores o problema passa a se concentrar no próprio acesso a base de dados. Entretanto pode-se ainda usar escalabilidade para resolver o problema de acesso a base de dados que consiste em distribuir a base de dados do sistema em várias máquinas, porém, tal abordagem é possível em SGBD relacional mas não é uma tarefa realizada de uma maneira simples devido a natureza estruturada da organização dos dados do Modelo Relacional.

Tendo em vista essa deficiência dos SGBDs, os desenvolvedores começaram a pensar em um modo alternativo de se modelar as bases de dados sem toda a inflexibilidade da estrutura do Modelo Relacional.

Assim surgiram soluções como os simples sistemas de gerenciamento de arquivos, onde por um lado pecavam nas regras de consistência, por outro lado, ganhavam em desempenho, flexibilizando o sistema para as particularidades de cada organização, surgindo assim os Bancos Não Relacionais ou NoSQL, cujo seu propósito não era substituir os bancos de dados relacionais e sim ser útil nos casos em que fossem requeridos maior flexibilidade e disponibilidade.

Consequentemente, mostrando as vantagens e desvantagens entre o Modelo Relacional e o Modelo Não Relacional é possível optar, com ajuda desse estudo, qual o banco de dados determinar quando é necessário escolher por escalabilidade ou integridade. (SOBRINHO, 2008; LEITE, 2010)

1.1 Objetivo

O presente trabalho tem a finalidade de demonstrar os benefícios da escalabilidade para aplicações, principalmente web, nos aspectos relativos a consistência dos dados e disponibilidade, trazendo as vantagens e desvantagens do modelo relacional e não relacional em busca da escalabilidade.

1.1.1 Objetivos Específicos

- Definir as propriedades dos bancos relacionais, tais como sua estrutura de dados;
- Apontar os tipos de bancos de dados não relacionais e seu melhor uso em cada cenário;
- Verificar qual o melhor tipo de banco de dados que se adequa a situação do problema de disponibilidade e distribuição de banco de dados;
- Identificar qual o melhor tipo de escalabilidade para os tipos de bancos descritos.
- Fazer uma comparação de escalabilidade entre os bancos relacionais e não relacionais

1.2 Justificativa

Atualmente para uma aplicação ficar *off-line* por problemas em bancos de dados pode ser bastante dispendioso, onde, com o crescimento do fluxo de usuários os bancos de dados ficam sobrecarregados e tendem a ser um problema para sistemas que tem o seu uso compartilhado por milhares de usuários, sendo assim é necessário usar do artifício da escalabilidade, para que o banco de dados não se torne um problema e sim uma solução para que a aplicação fique mais estável, consistente e disponível.

Esse estudo tem como importância proporcionar aos administradores de bancos de dados o conhecimento de qual método deve ser escolhido para seu sistema, onde dependendo do seu tipo de sistema a escalabilidade vai diminuir os gastos e otimizar o sistema.

1.3 Organização do Trabalho

O trabalho sobre Escalabilidade de Aplicações com Banco de Dados Não Relacional está organizado em forma de capítulos onde se divide da seguinte forma: no capítulo dois terá início o referencial teórico desse trabalho, onde será abordado o conceito de bancos de dados. No terceiro capítulo será exposto a parte dos bancos de dados relacionais como sua história, estrutura, modelo relacional e vários conceitos pertinentes a bancos de dados relacionais. No quarto capítulo será abordado os bancos de dados não relacionais, seus tipos, suas propriedades e o modelo BASE. No quinto capítulo será abordado o teorema CAP que trata da consistência, disponibilidade e tolerância ao particionamento. No sexto capítulo será mostrado os princípios da escalabilidade, tais como problemas e soluções tanto físicas como de software. No sétimo e último capítulo será apresentado as conclusões pertinentes ao trabalho.

De acordo com Silberschatz, Korth e Sudarshan (2005), um banco de dados é uma coleção de dados que contém informações relevantes a uma empresa, onde, gerenciada por um DBMS, fornece uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente.

Pode-se observar que os autores chegam à conclusão de que um banco de dados é o armazenamento de dados de forma relacionada de um modo geral onde são gerenciados por SGBD.

2 REFERENCIAL TEÓRICO

Aqui serão apresentados os conceitos que serviram como base para a análise de escalabilidade entre os bancos relacionais e não relacionais.

2.1 Banco de Dados

Um Banco de Dados, segundo Rob e Coronel (2011), ou uma base de dados, é uma estrutura computacional compartilhada e integrada que armazena um conjunto de dados do usuário final e metadados, ou dados sobre dados e estes dados são gerenciados por um SGBD que consiste em um conjunto de programas que gerenciam a estrutura do banco de dados e controlam o acesso aos dados armazenados.

De acordo com Silberschatz, Korth e Sudarshan (2006), um banco de dados é uma coleção de dados que contém informações relevantes a uma empresa, onde, gerenciada por um DBMS, fornece uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente.

Pode-se observar que os autores chegam a conclusão de que um banco de dados é o armazenamento de dados de forma relacionada de um modo geral onde são gerenciados por SGBD.

Fonte: Korth e Silberschatz (1994)

3 BANCO DE DADOS RELACIONAL

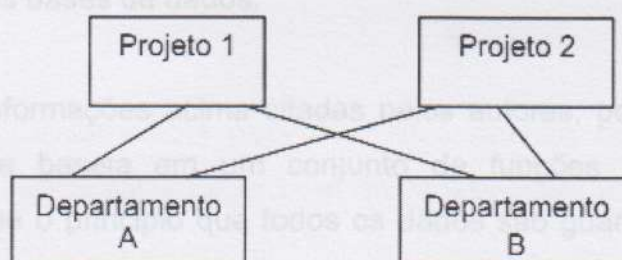
Neste tópico serão apresentadas as características dos bancos de dados relacionais como o próprio modelo relacional, sua estrutura de armazenamento de dados, as propriedades ACID, normalização dos dados e o padrão da indústria.

Fonte: Korth e Silberschatz (1995)

3.1 Modelo Relacional

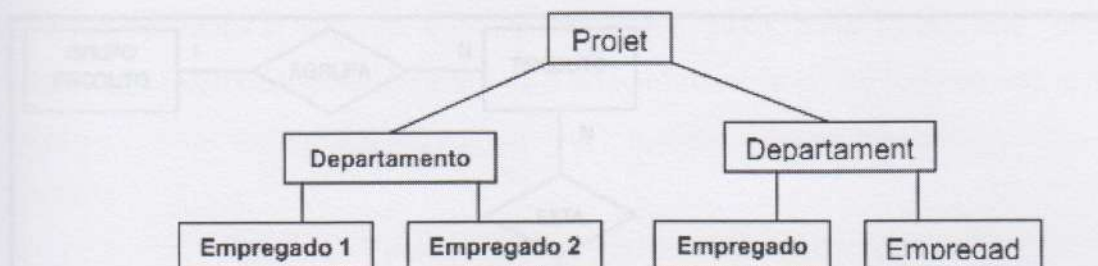
O modelo relacional, segundo Korth e Silberschatz (1995), é relativamente novo e os primeiros sistemas de bancos de dados basearam-se ou no modelo de redes, vide figura 1, ou no modelo hierárquico, vide figura 2, onde esses dois modelos se relacionam melhor com a implementação do banco de dados do que com o modelo relacional propriamente dito.

Figura 1 - Exemplo de banco de dados no modelo de rede



Fonte: Korth e Silberschatz (1995)

Figura 2 - Exemplo de banco de dados no modelo hierárquico



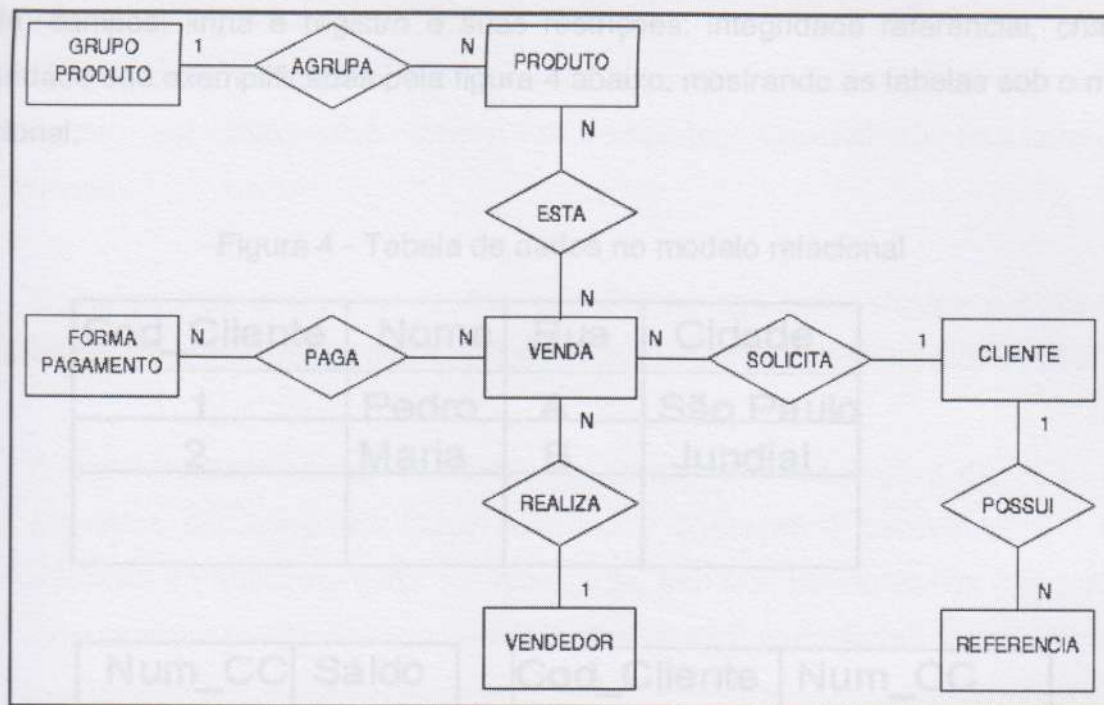
Fonte: Korth e Silberschatz (1995)

Um banco de dados relacional consiste em uma coleção de tabelas, cada qual designada por um nome único. Cada tabela tem uma estrutura similar a de uma representação de um Diagrama de Entidade Relacionamento (DER). Uma linha em uma tabela representa um relacionamento entre um conjunto de valores. Uma vez que uma tabela é uma coleção de tais relacionamentos, existe uma correspondência entre o conceito de tabela e o conceito matemático de relação, a partir da qual o modelo de dados relacional tira o seu nome (KORTH e SILBERSCHATZ, 1995, p. 58).

O Modelo relacional, segundo Takai, Italiano e Ferreira (2005), surgiu mediante a necessidade de prover um conjunto de funções apoiadas em álgebra relacional para o armazenamento e recuperação de dados, tendo como base a teoria dos conjuntos e álgebra relacional revelou ser o mais flexível e adequado para solucionar os problemas de implementação das bases de dados.

Através das informações acima citadas pelos autores, pode-se entender que o modelo relacional se baseia em um conjunto de funções apoiadas em álgebra relacional, onde segue o princípio que todos os dados são guardados em tabelas que são representadas por um relacionamento entre um conjunto de valores são estruturados por um modelo de entidade relacionamento (ER), vide figura 3.

Figura 3 - Modelo entidade relacionamento



Fonte: Barcelar (2011)

3.2 Armazenamento de Dados Relacional

Os elementos fundamentais para o armazenamento de dados nos modelos relacionais possuem responsabilidades bem definidas, como mencionadas em Takai, Italiano e Ferreira (2005).

A estrutura fundamental do modelo relacional é a relação (tabela). Uma relação é constituída por um ou mais atributos (campos) que traduzem o tipo de dados a armazenar. Cada instância do esquema (linha) é chamada de tupla (registro). O modelo relacional implementa estruturas de dados organizadas em relações. Porém, para trabalhar com essas tabelas, algumas restrições precisaram ser impostas para evitar aspectos indesejáveis, como: Repetição de informação, incapacidade de representar parte da informação e perda de informação. Essas restrições são: integridade referencial, chaves e integridade de junções de relações (TAKAI, ITALIANO e FERREIRA, 2005, p. 8).

Ainda de acordo com, Takai, Italiano e Ferreira (2005), essa estrutura de dados: tabela, campos, linha e registro e suas restrições: integridade referencial, chaves e integridade são exemplificadas pela figura 4 abaixo, mostrando as tabelas sob o modelo relacional.

Figura 4 - Tabela de dados no modelo relacional

Cod_Cliente	Nome	Rua	Cidade
1	Pedro	A	São Paulo
2	Maria	B	Jundiai

Num_CC	Saldo
20121	1200
21582	1320
21352	652

Cod_Cliente	Num_CC
1	20121
2	21582
2	21352

Fonte: Takai, Italiano e Ferreira (2005)

3.3 ACID

As propriedades referidas pelo modelo ACID descrito por Rob e Coronel (2011), consiste em, toda transição individual deve apresentar indivisibilidade, consistência, isolamento e durabilidade, onde o SGBD deve escalonar as operações concorrentes e esse escalonamento deve seguir as propriedades do modelo ACID.

Uma transação é normalmente iniciada por um programa escrito em uma linguagem de programação de alto nível, segundo Silberschatz, Korth e Sudarhan (2006), onde essas transações são executadas normalmente usando SQL, linguagem

de banco de dados que permite a criação de bancos de dados, estruturas de tabelas e comandos de gerenciamento de dados.

Para garantir a integridade dos dados, é necessário que o SGBD mantenha as propriedades de transações como: Atomicidade, Consistência, Isolamento e Durabilidade.

Segundo Rob e Coronel (2011), a definição das propriedades citadas acima como Atomicidade (Indivisibilidade), Consistência, Isolamento e Durabilidade são:

- A indivisibilidade: exige que todas as operações (solicitações de SQL) sejam gravadas por completo. Caso contrário a operação é cancelada. Em outras palavras a transação, após ser executada, tem que ser concluída integralmente sem nenhum problema (erro de execução) ou será cancelada.
- A consistência: indica a permanência do estado consistente do banco de dados. A transação leva o banco de dados de um estado consistente a outro, quando a transação é concluída o banco deve estar em um estado consistente. Em outras palavras quando os dados são gravados no banco de dados eles vão estar íntegros em seu estado.
- O isolamento: significa que os dados utilizados durante a execução de uma transação não podem ser utilizados por uma segunda transação até a primeira ser concluída, essa propriedade é especialmente útil em ambientes com bancos multiusuários, pois vários usuários podem acessar o banco simultaneamente. Em outras palavras quando um dado é acessado nenhum outro usuário pode acessar esse mesmo dado para manter a integridade do mesmo.
- A durabilidade: garante que uma vez feita as alterações elas não podem ser desfeitas ou perdidas, mesmo em caso de falha do sistema. Em outras palavras quando um dado é gravado ele persiste na base de dados mesmo o sistema apresentando falhas ou ficando desabilitado.

3.3 Padrão da Indústria

Diante da definição dos autores acima, pode-se afirmar que as várias propriedades denominadas ACID (Atomicidade, Consistência, Isolamento e Durabilidade) são de fundamental importância para a integridade dos dados sendo esta uma dos pontos fortes do modelo relacional.

3.4 Normalização de Dados

A necessidade da normalização dos dados é considerada um elemento fundamental para os bancos de dados relacionais como afirma Elmasri e Navathe (2011).

A normalização de dados pode ser considerada um processo de analisar os esquemas de relação dados como base suas dependências funcionais e chaves primárias para conseguir as propriedades desejadas de minimização da redundância e minimização das anomalias de inserção exclusão e atualização (ELMASRI e NAVATHE, p. 348, 2011).

Como afirma Barcelar (2011), o processo de normalização é encarado como o método no qual são eliminados esquemas de relações (tabelas) não satisfatórias, decompondo-as, através da separação de seus atributos em esquemas de relações não tão complexas, mas que satisfaçam as propriedades desejadas.

Esse processo conduz um esquema de relação através de uma bateria de testes para certificar se o mesmo encontra-se na 1ª, 2ª e 3ª Formas Normais. Essas três Formas Normais são baseadas nas dependências dos atributos da relação.

Como se pode observar a afirmação de Barcelar (2011), completa a afirmação de Elmasri e Navathe (2011), onde o processo de normalização analisa as relações com base nas dependências funcionais, no qual, eliminam os esquemas de relação não satisfatórios através de testes como as Formas Normais.

3.5 Padrão da Indústria NÃO RELACIONAL

Como afirma Brito (2010), “Desde a sua criação no início dos anos 1970, o Modelo Relacional de dados tem sido utilizado em larga escala pela maioria dos sistemas de gerenciamento de banco de dados”. Os bancos de dados relacionais oferecem aos usuários diversos atributos que facilitaram a vida dos desenvolvedores de aplicações, fazendo com que os mesmos tivessem um foco maior com a aplicação como: validação, verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas, segurança, entre outros.

O conceito do modelo relacional com a associação entre as tabelas do banco de dados são amplamente utilizadas.

Todos esses recursos ajudaram a manter os bancos relacionais em posição de destaque entre os mais diversos tipos de ambientes computacionais, mas não impediram o surgimento de certos problemas, principalmente devido ao crescimento vertiginoso do volume de dados presentes nos bancos de certas organizações (BRITO, p. 2, 2010).

De acordo com o autor, bancos de dados relacionais têm relativamente um período não tão longo de vida e esse modelo tem se aprimorado ao longo do tempo, onde se tornou o sistema de gerenciamento de banco de dados mais utilizado devido às suas características, porém com o aumento do volume de dados o modelo relacional dos SGBDs tradicionais não conseguiram acompanhar o ritmo e assim passaram a não ser tão eficientes para os sistemas que necessitavam de uma demanda por escalabilidade cada vez mais frequente.

Assim sendo, segundo Leavitt (2010), devido à sua natureza estruturada os desenvolvedores começaram a perceber que teriam muita dificuldade ao se organizar o modelo relacional em um sistema distribuído trabalhando com particionamento de dados e foi justamente nesse ponto que surgiram as ideias para as soluções os bancos de dados não relacionais.

4 BANCOS DE DADOS NÃO RELACIONAL

Nesse capítulo será abordada a história dos bancos não relacionais assim como sua usabilidade e o seus tipos.

4.1 NoSQL

O termo NoSQL foi usado pela primeira vez em 1998 por um banco de dados que omitia o uso de SQL e foi usado novamente em 2009 em uma conferência de defensores dos bancos não relacionais por Jon Oskarsson e posteriormente foi descrito por ser "a alternativa que você procura para resolver os problemas onde os bancos de dados relacionais são uma péssima solução" (STRAUCH, 2011).

Os *softwares* interativos têm mudado de maneira fundamental ao longo dos últimos 35 anos. Os sistemas online da década de 1970 têm passado por uma série de transformações para atender as necessidades da web nos dias de hoje e para aplicações moveis. Estes sistemas visam resolver novos problemas para as populações usuário potencialmente muito maior, e elas são executadas em cima de uma infraestrutura computacional que mudou ainda mais radicalmente ao longo dos anos.

A arquitetura desses sistemas de software também tem se transformado. A *web* moderna pode ter aplicações que suportem milhões de usuários simultâneos, espalhando carga através de uma coleção de servidores de aplicativos atrás de um balanceador de carga. Mudanças no comportamento do aplicativo pode ser implementado de forma incremental sem a necessidade de inatividade do aplicativo gradualmente substituindo o software em servidores individuais. Ajustes para capacidade de aplicação podem ser feitas facilmente, alterando o número de servidores de aplicação.

Mas a tecnologia de banco de dados não acompanhou o ritmo. A tecnologia de banco de dados relacional, inventado na década de 1970 e ainda em uso difundido hoje, foi otimizada para as aplicações, usuários e infraestrutura. Enquanto um número de arranjos técnicos estendeu a vida útil da tecnologia (*sharding* horizontal e vertical, *caching* distribuído e desnormalização de dados), essas táticas anulam os principais benefícios do modelo relacional, enquanto aumenta o custo total do sistema e sua complexidade.

Em resposta à falta de alternativas disponíveis no mercado, organizações como Google (BigTable) e Amazon (Dynamo) foram, por necessidade, forçados a inventar novas abordagens para o gerenciamento de dados. NoSQL ou tecnologias não-relacionais de banco de dados são uma melhor adequação às necessidades dos modernos sistemas de software interativo, onde, com sua tecnologia permitiram a gestão econômica de dados por trás da web moderna e aplicações móveis. (COUCHBASE, 2011)

4.2 Tipos de Bancos NoSQL

Os bancos de dados NoSQL são categorizados por serem não relacionais, distribuídos, código aberto e escalável horizontalmente, onde, tem como principais características: livre de esquemas, fácil suporte na replicação, API simples, eventualmente consistente (BASE), suporte a uma enorme quantidade de dados, entre outras. E são divididos em: Wide Column Store, Document Store, Key Value, Graph Databases, entre outros, segundo Nosql (2011).

Fazendo uma análise dos modelos existentes é importante deixar claro que um modelo não deve ser considerado melhor que o outro, uma vez que cada tipo de modelo pode ser mais adequado para determinadas aplicações. Onde, para manipulação de dados estatísticos, frequentemente escritos, mas raramente lidos, pode ser usado um banco de dados orientado a chave/valor ou um banco de dados orientado a documentos. Aplicativos de alta disponibilidade, onde é necessária a minimização da

inatividade é fundamental, pode-se utilizar um banco de dados orientado a colunas. Aplicações que exigem alto desempenho em consultas com muitas junções podem adotar um banco orientado a grafos.

A importância de conhecer cada solução e adotar aquela que for mais adequada poderá contribuir para a diminuição do custo de criação do banco de dados, bem como para o aumento da eficiência no processamento dos dados. Serão apresentados a seguir os tipos mais conhecidos.

4.2.1 Key Value (Chave Valor)

Segundo Decandia et al. (2007), os sistemas distribuídos nessa categoria são conhecidos como tabelas de *hash* distribuídas, onde armazenam objetos indexados por chave e possibilitam as buscas por esses objetos a partir de suas chaves.

“O banco de dados é composto por um conjunto de chaves, as quais estão associadas um único valor, que pode ser uma *string* ou um binário.” (LÓSCIO, OLIVEIRA e PONTES, 2011) Ainda segundo Lóscio, Oliveira e Pontes (2011), este modelo permite que os dados sejam acessados rapidamente pela chave, principalmente em sistemas que possuem alta escalabilidade, fazendo com que aumente a disponibilidade de acesso aos dados. A desvantagem deste modelo é que não permite a recuperação de consultas mais complexas.

São exemplos de bancos que seguem o conceito de chave-valor: Redis¹, Riak² e Memcached³.

¹ <http://try.redis-db.com>

² <http://wiki.basho.com>

³ <http://memcachedb.org>

4.2.2 Wide Columns (Tamanho de Coluna)

Um dos modelos mais utilizados de bancos de dados não relacional é o Wide Columns, onde será abordado nesse quesito.

Este modelo é um pouco mais complexo que o modelo chave-valor e, neste caso, muda-se o paradigma de orientação a registros ou tuplas (como no modelo relacional) para orientação a atributos ou colunas (modelo NoSQL). Neste modelo os dados são indexados por uma tripla (linha, coluna e timestamp), onde linhas e colunas são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado, Lóscio, Oliveira e Pontes, p. 6 (2011).

As operações de leitura e escrita são atômicas, em outras palavras, todos os valores associados a uma linha são considerados na execução das operações, independente das colunas que estão sendo lidas ou escritas. Este modelo de dados surgiu com o BigTable da Google e dentre suas características pode-se citar que permite o particionamento de dados e oferece forte consistências, porém não garante alta disponibilidade, como afirma Lóscio, Oliveira e Pontes (2011).

Os bancos mais conhecidos seguindo esse formato são: Cassandra⁴, HBase⁵ e Hypertable⁶.

Os bancos NoSQL mais populares no quesito documento são MongoDB⁷ e CouchDB⁸.

⁴ <http://cassandra.apache.org/>

⁵ <http://hbase.apache.org/>

⁶ <http://hypertable.org/>

4.2.3 Document Store (Armazenamento de Documentos)

Os bancos de dados orientados a documentos são coleções de atributos e valores, como afirma Diana e Gerosa (2010), onde um atributo pode ser multivalorado. Geralmente os bancos orientados a documentos não possuem esquema, fazendo com que os documentos armazenados não precisem possuir uma estrutura comum, essa característica faz desse tipo de banco uma boa opção para o armazenamento de dados semiestruturados.

Nos bancos de dados orientados a documentos é necessário usar um formato de armazenamento de dados que suporte uma estrutura dentro de outra, em outras palavras embutir um documento em outro, como o JSON, onde ao se fazer esse procedimento muitas vezes ocorre a duplicação de dados, essa duplicação pode ocasionar problemas de consistência causados pela atualização ou deleção.

A normalização presente nos bancos relacionais tenta evitar esse tipo de problema, mas considerando a perda da integridade ao se fazer *sharding* em bancos relacionais a duplicação em bancos orientados a documentos não é um problema. A duplicação de dados facilita a distribuição do sistema, já que a quantidade de nós a serem consultados é menor caso estejam próximos, tendo em vista que a consulta ocorre em um único nó. Esse fato mostra uma característica presente não só nos bancos orientado a documentos, mas nos bancos NoSQL de um modo geral (DIANA e GEROSA, 2010; CHODOROW e DIROLF, 2010; ANDERSON, LEHNARDT e SLATER 2010).

Os bancos NoSQL mais populares no quesito documento são MongoDB⁷ e CouchDB⁸.

⁷ <http://www.mongodb.org/>

⁸ <http://couchdb.apache.org/>

4.2.4 Graph Store (Armazenamento de Grafos)

“Diferentemente de outros tipos de bancos de dados NoSQL, esse está diretamente relacionado a um modelo de dados estabelecido, o modelo de grafos” (DIANA e GEROSA, 2010). O modelo orientado a grafos é constituído por três componentes básicos os nós (vértices), os relacionamentos (arestas) e as propriedades (atributos), deste modo o banco de dados pode ser enxergado como um multigrafo rotulado e direcionado, onde cada par de nós é conectado por mais de uma aresta.

A vantagem da utilização do modelo baseado em grafos fica bastante clara quando as consultas exigidas pelos usuários se tornam bastante complexas. Comparando ao modelo relacional, o modelo orientado a grafo tem um ganho de performance, permitindo um melhor desempenho das aplicações, de acordo com Lóscio, Oliveira e Pontes (2011).

Neo4j⁹, AllegroGraph¹⁰ e Virtuoso¹¹ são exemplos de bancos de dados baseado em grafos.

4.3 Modelo BASE

4.2.5 Desnormalização

Antes de armazenar dados em um SGBDR, um esquema deve ser criado e definindo precisamente que dados podem ser armazenados no banco de dados e a relação entre os elementos dos dados. Os dados são decompostos em uma forma normal e espalhados por muitas tabelas interligadas, normalmente.

⁹ <http://neo4j.org/>

¹⁰ <http://www.franz.com/agraph/allegrograph/>

¹¹ <http://www.openlinksw.com/>

Para atualizar um registro todas essas tabelas devem ser bloqueadas, depois, ser feita a atualização atômica para que o banco de dados não fique corrompido. Esse modelo limita substancialmente a latência e taxa de transferência de atualizações simultâneas. Para suportar a concorrência e sharding, os dados são frequentemente armazenados de uma forma desnormalizada quanto aos SGBDR. Essa abordagem duplica potencialmente dados nos bancos de dados quando é necessário fazer uma atualização que requeiram várias tabelas, porém essa abordagem reduz a quantidade de bloqueios necessários e melhora a concorrência.

A desnormalização abandona quase que inteiramente o esquema relacional, onde com dados armazenados em forma de chave-valor, uma chave primária está ligada a um conjunto de dados que pode conter todos os dados. Deste modo o tipo das informações que estão sendo armazenadas no banco de dados podem sofrer alterações sem a necessidade que seja feita mudança no esquema, isto faz o com que o sharding seja muito mais fácil e permite uma rápida mudança no modelo de dados, com isso todos os benefícios dos bancos de dados relacionais está perdida (COUCHBASE, 2011).

4.3 Modelo BASE

O conceito denominado pelo modelo BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) vai de encontro ao paradigma do modelo ACID, onde se caracteriza por ser basicamente disponível em seu estado leve com uma consistência eventual, ou seja, o sistema estará disponível o tempo todo, mas não será consistente o tempo todo, tornando-o consistente no momento devido, de acordo com BRITO (2010)

O BASE é otimista em relação ao ACID. Este segundo é forçado a consistência em 100% e caso ocorra algum erro na transação ela é cancelada. Por outro lado o modelo BASE aceita que a consistência dos dados estará no estado do fluxo. O BASE suporta falhas parciais ao invés de ocorrer uma falha total no sistema, onde em um exemplo de um sistema distribuído rodando em cinco servidores, quando ocorre falha em um servidor, o sistema vai funcionar com 80% da sua capacidade mas não vai parar

de funcionar e voltará a sua capacidade normal quando estado da máquina for normalizado.

A disponibilidade no modelo BASE está relacionada ao fato de que a queda de uma máquina do sistema não leva o sistema inteiro a ser comprometido ou interrompido, representando apenas uma máquina a menos disponível.

A consistência eventual ocorre, pois, no modelo BASE é usado o esquema de filas de mensagem para as transações sendo designado como 2PC (two phase commit), onde o sistema tentará gravar os dados em duas fases, e em cada uma dessas fases será emitida uma mensagem para a transação, sendo a primeira a tentativa de salvar os dados no banco e em caso positivo entrará na segunda fase, que será emitido a mensagem de confirmação que os dados foram salvos, caso ocorra algum erro na primeira fase a segunda fase mostrará uma mensagem mostrando que os dados não foram salvos (PRITCHETT, 2008).

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, como exemplo Google e Facebook.

A tolerância ao particionamento segundo Gilbert e Lynch (2002) 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. A divisão das propriedades do sistema CAP pode ser visto na figura 5.

5 TEOREMA CAP

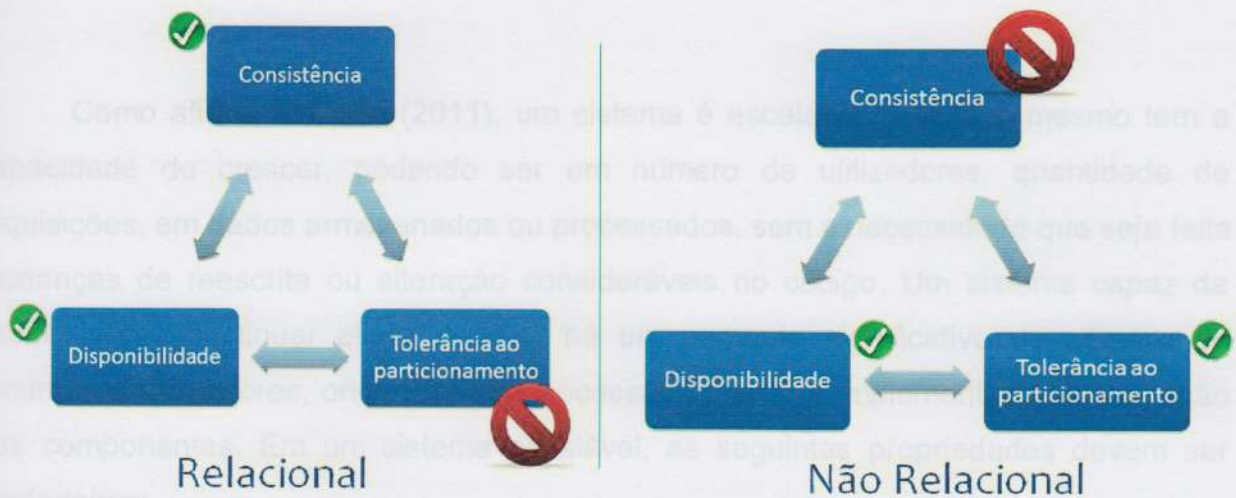
De acordo com Brewer (cit. in LEITE, 2010), em um sistema computacional distribuído é desejável obter consistência, disponibilidade e tolerância ao particionamento. Entretanto, essas propriedades são impossíveis de serem obtidas ao mesmo tempo. De acordo com o teorema, que ficou conhecido como teorema CAP (Consistency, Availability e Partition Tolerance), um sistema distribuído só pode garantir apenas duas dessas propriedades simultaneamente.

Como afirma Wei, Pierre e Chi (2009), 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.

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, como exemplo Google e Facebook.

A tolerância ao particionamento segundo Gilbert e Lynch (2002) 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. A divisão das propriedades do teorema CAP pode ser visto na figura 5.

Figura 5 – Atributos do teorema CAP em relação a escolha do banco de dados.



Fonte: Elemar (2011)

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.

De acordo com Fox e Brewer (1999), em sistemas que priorizam as propriedades de consistência e disponibilidade podemos citar os bancos de dados associados ao Modelo Relacional, o que só é possível com a não existência de particionamentos na rede. Em relação aos bancos de dados que adotam a disponibilidade e tolerância a particionamento podemos citar os bancos NoSQL, onde, disponibilizam uma consistência eventual para adotar a disponibilidade.

6 ESCALABILIDADE

Como afirma Mazetto (2011), um sistema é escalável quando o mesmo tem a capacidade de crescer, podendo ser em número de utilizadores, quantidade de requisições, em dados armazenados ou processados, sem a necessidade que seja feita mudanças de reescrita ou alteração consideráveis no código. Um sistema capaz de escalar deve continuar eficaz quando há um aumento significativo do número de recursos e utilizadores, onde não seja necessário alterar a implementação e interação dos componentes. Em um sistema escalável, as seguintes propriedades devem ser verdadeiras:

- Os recursos envolvidos devem aumentar linearmente com a dimensão do sistema.
- O Aumento da dimensão dos dados/recurso não deve levar a diminuição significativa do desempenho.

Segundo Henderson (cit. in SOBRINHO, 2009), escalabilidade não tem relação com performance e sim com a capacidade de acomodar o aumento do uso do sistema e da quantidade de dados adicionando hardware, sem perder sua manutenibilidade. Entretanto escalar não implica necessariamente adição de mais *hardware*, pois a melhor utilização dos recursos computacionais existentes na arquitetura de software também pode ser uma possível solução ao problema de escalabilidade.

O fato de um sistema ser escalável é uma característica momentânea, ou seja, um sistema pode suportar cada vez mais acessos apenas adicionando mais hardware ou melhorando os algoritmos de cache. Mas a partir de um determinado número de acessos, a solução começa a não ter mais ganhos, seja por um problema de hardware ou software, o sistema encontra seu limite de escalabilidade e deixa de ser escalável para aquele novo cenário. Para continuar escalando, é necessário encontrar o limitador

e substituí-lo por uma nova solução, de acordo com Schlossnagle (cit. in SOBRINHO, 2009).

6.1 Problemas com Escalabilidade

No âmbito dos contextos das aplicações *web* os problemas relacionados com escalabilidade podem ocorrer de diversas maneiras e atingem principalmente aplicações que precisam de um processamento intenso, que necessitam de uma enorme quantidade de armazenamento e que precisam garantir o acesso de um grande número de usuários.

Considerando um cenário de uma aplicação *web* que realize diversas tarefas que exijam um processamento mais intenso deve-se pensar na escalabilidade acrescentando mais hardware até atingir o limite da arquitetura ou distribuir a carga em diversas máquinas, nesse contexto é aplicado escalabilidade horizontal ou vertical. No cenário de aplicações com enorme quantidade de armazenamento que precisam armazenar arquivos de mídia tais como fotos, vídeos, planilhas, etc., tem um volume de dados gigante onde poderá ser usado a escalabilidade vertical, porém as consultas ao banco de dados ficarão lentas, então, podemos optar pelo sharding que é a distribuição do banco de dados em várias máquinas, esta solução também é considerada uma boa prática para cenários onde se tem um grande número de acesso de usuários (MAZETTO, 2011).

6.2 Soluções de Escalabilidade

Aqui serão apresentadas técnicas possíveis para escalabilidade e suas melhores aplicações como suas vantagens e desvantagens.

6.2.1 Sharding

De acordo com Plugge, Membrey e Hawkins (2010), o *sharding* é uma técnica que permite a você espalhar os seus dados ou banco de dados através de múltiplas máquinas, ou seja, distribuir o processamento do seu banco de dados em diversos servidores paralelizando as consultas.

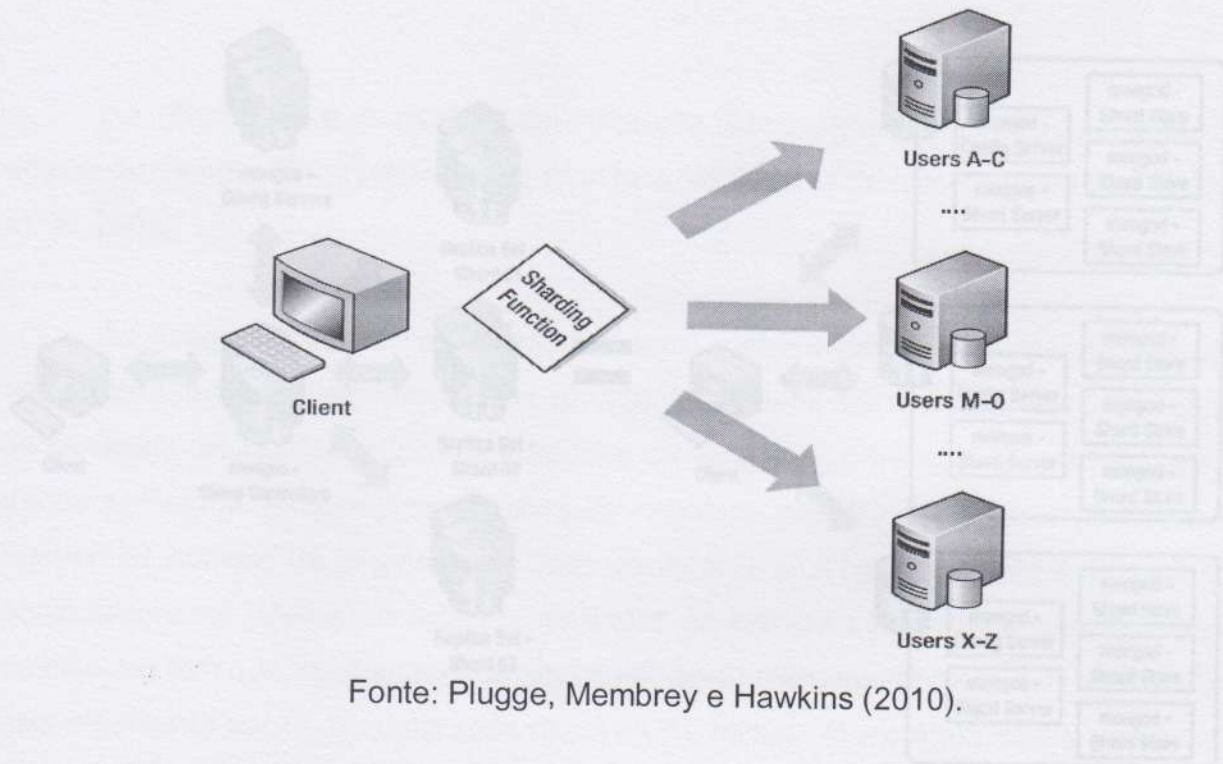
O modelo de dados dos bancos relacionais tem uma mecânica de operação assumidamente centralizada no modelo computacional - CPU compartilhada, memória e HD. Se os dados não couberem em um único servidor ou se o servidor for incapaz de manter E/S de transferência necessária para atender a muitos usuários simultaneamente então será necessário o uso de uma técnica conhecida como sharding, Couchbase, p. 8 (2011).

“O termo Sharding se refere ao processo de divisão de dados e armazenamento de diferentes partes dos dados em diferentes máquinas, o termo particionamento também é usado para descrever esse conceito” (CHODOROW e DIROLF, 2010). Ainda segundo o conceito de Chodorow e Dirolf (2010), a divisão de dados entre diferentes máquinas, torna-se possível armazenar mais dados e lidar com mais carga sem a necessidade de adquirir máquinas melhores ou mais poderosas.

Quando seu banco de dados não é armazenado em uma única máquina, mas em nós de cluster, sua vantagem é que os nós podem ser adicionados ao cluster para aumentar a capacidade e o desempenho de leitura e escrita de operações sem a necessidade de modificar a aplicação, onde, também é possível reduzir o tamanho de um cluster de banco de dados (shards) quando as exigências diminuem (OREND, 2010).

Quando é feita a configuração de um sharding deve-se escolher uma chave para a coleção e usar essa chave para dividir os dados. Essa chave é chamada “shard key”. Na figura 6 temos um exemplo da execução de um sharding que contém bancos distintos, onde, documentos como nomes iniciados de A-C são armazenados no primeiro banco de dados, o próximo shard ficará com os nomes iniciados de D-F e o último shard com os nomes X-Z, Chodorow e Dirolf (2010).

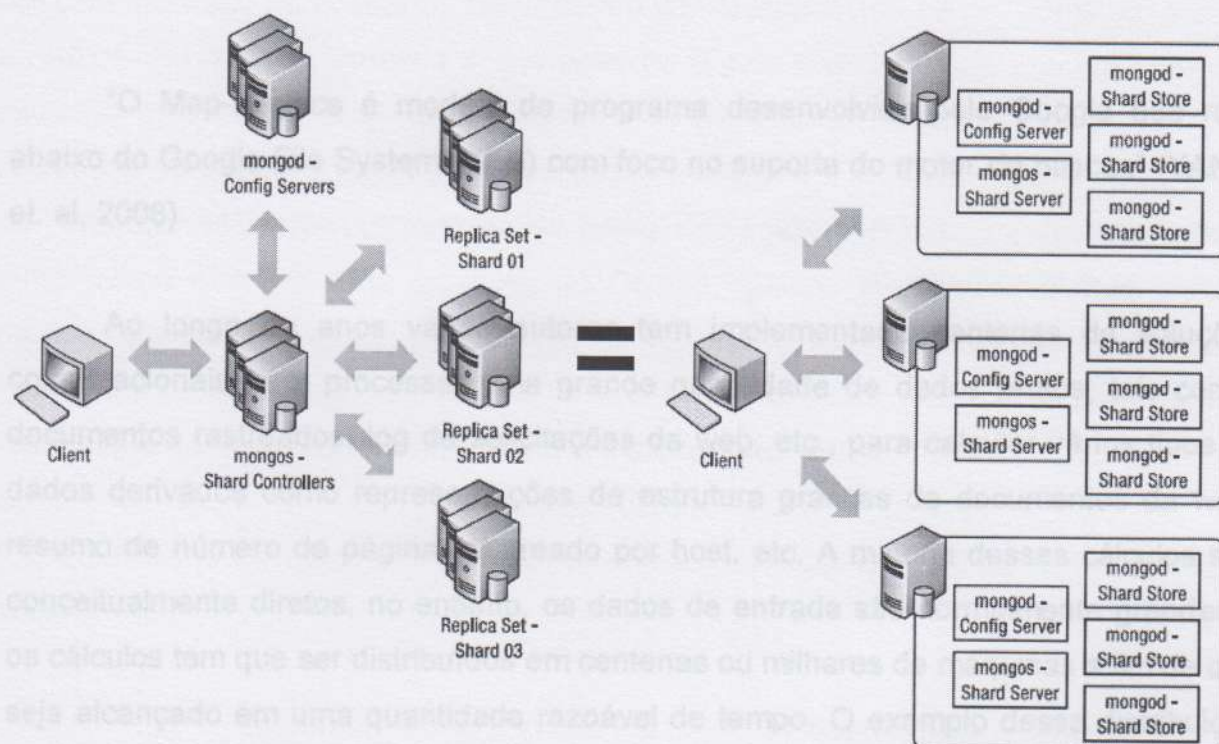
Figura 6 - Exemplo simples de sharding sobre uma coleção de usuários.



Fonte: Plugge, Membrey e Hawkins (2010).

O sharding pode ser feito manual em quase todos os softwares de banco de dados, isso ocorre quando a aplicação mantém conexões para vários servidores de bancos de dados diferentes, cada um dos quais são completamente independentes. O código da aplicação é quem gerencia o armazenamento de dados em diferentes servidores, fazendo a consulta no servidor apropriado e obtendo a resposta, essa abordagem pode funcionar bem, mas se torna complicado de manter quando a adição ou remoção de nós do cluster ou distribuição de dados. Alguns bancos de dados suportam o autosharding diminuindo o trabalho ao se fazer essa tarefa manualmente (CHODOROW e DIROLF, 2010). Na figura 7 pode-se observar a execução de sharding com redundância onde com o uso do autosharding será acrescentado mais nós ao cluster de bancos de dados, assim como suas replicas.

Figura 7 - Exemplo de uma configuração de sharding redundante.



Fonte: Plugge, Membrey e Hawkins (2010).

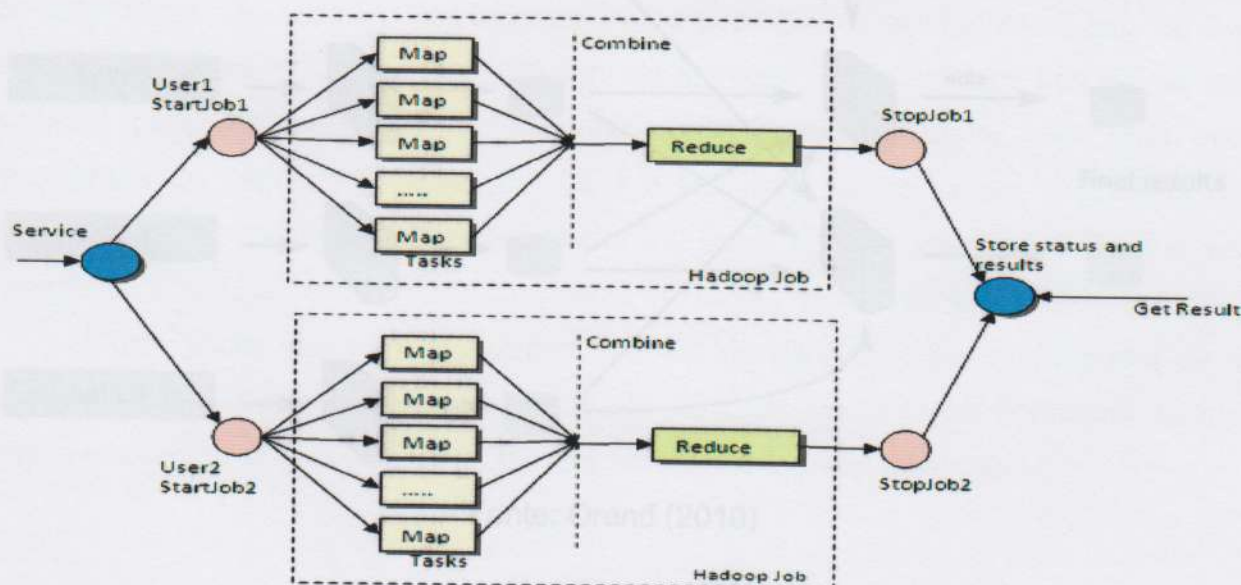
De acordo com Orend (2010), a técnica de sharding também tem suas desvantagens, onde, algumas operações típicas e eficientes de banco de dados relacional se tornam complexas e ineficientes. Uma das operações mais importantes nas bases de dados relacional é o operador de junção (JOIN), que é usado para materializar as relações dos dados. A operação de junção funciona em dois conjuntos de dados, um esquerdo (LEFT JOIN) e um direito (RIGHT JOIN), que são conectados por um par de atributos. Uma junção distribuída em um shard de banco de dados exige que o banco de dados procure no conjunto do lado direito todos os itens que estão associados a cada item no conjunto da esquerda. Isso exige muitas requisições para todas as máquinas que armazenam itens de dados de um dos dois conjuntos, que causaria um grande tráfego de rede. Devido a isso, a maioria dos bancos de dados que fazem shard não suportam operações de junção (OREND, 2010).

6.2.2 MapReduce

“O Map-Reduce é modelo de programa desenvolvido pelo Google que roda abaixo do Google File System (GFS) com foco no suporte do motor de buscas” (YANG, et. al, 2008)

Ao longo de anos vários autores tem implementado centenas de soluções computacionais para processar uma grande quantidade de dados brutos, tais como, documentos rastreados, log de solicitações da web, etc., para calcular vários tipos de dados derivados como representações de estrutura gráficas de documentos da web, resumo de número de páginas rastreado por host, etc. A maioria desses cálculos são conceitualmente diretos, no entanto, os dados de entrada são normalmente grandes e os cálculos tem que ser distribuídos em centenas ou milhares de máquinas a fim de que seja alcançado em uma quantidade razoável de tempo. O exemplo dessa distribuição de máquinas pode ser observado na figura 8, onde cada map significa uma máquina onde está sendo feito a consulta.

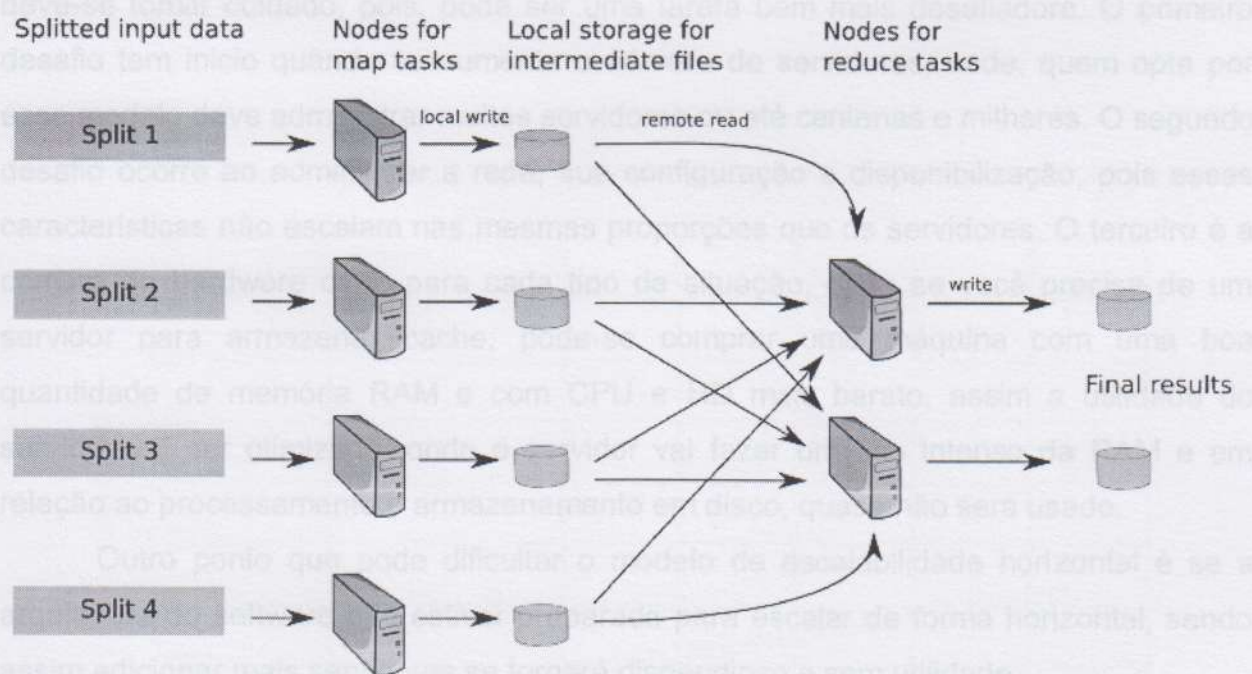
Figura 8 - Estrutura do Map/Reduce



Fonte: MAPREDUCE (2011)

Com toda essa complexidade foi desenvolvido uma forma de paralelizar esses cálculos inspirado na abstração presente em várias linguagens de programação o *map* e *reduce*. No *map* foi aplicada a operação de gravar nos dados de entrada com um intuito de calcular um conjunto intermediário de chave-valor e então aplicar a operação de *reduce* para reduzir todos os valores que compartilham a mesma chave-valor, a fim de combinar os resultados obtidos de forma adequada, em outras palavras, vamos tomar como exemplo uma pesquisa no Google, quando a pesquisa é feita o map procura pelos nós que compartilham a mesma chave-valor dividindo a busca (paralelizando a busca), depois que os nós fazem a busca eles convergem para o *reduce* que vai organizar os resultados e exibir para o usuário (DEAN e GHEMAWAT, 2004). Na figura 8 temos uma demonstração de é executado o algoritmo do Map/Reduce, já na figura 9 o paralelismo é feito a divisão de uma consulta usando o Map/Reduce.

Figura 9 - Visão geral da utilização do Map/Reduce



Fonte: Orend (2010)

6.2.3 Escalabilidade Horizontal

O princípio por trás da escalabilidade horizontal é semelhante ao modelo vertical em adicionar mais hardware. Porém ocorre de uma maneira diferente do modelo de escalabilidade vertical, onde não tem a necessidade de se colocar uma máquina mais poderosa, basta apenas colocar mais uma máquina semelhante a que já existe. (HENDERSON, 2006)

O principal motivador de se usar o modelo de escalabilidade horizontal é o custo, onde quando é exigido um poder computacional maior a adição de novos servidores se torna mais barato do que fazer a substituição do servidor por um mais robusto, além do baixo custo não se perde o investimento feito no hardware atual quando se escala horizontalmente.

Como afirma Sobrinho (2009), ao se usar o modelo de escalabilidade horizontal deve-se tomar cuidado, pois, pode ser uma tarefa bem mais desafiadora. O primeiro desafio tem início quando se aumenta o número de servidores, onde, quem opta por esse modelo deve administrar muitos servidores ou até centenas e milhares. O segundo desafio ocorre ao administrar a rede, sua configuração e disponibilização, pois essas características não escalam nas mesmas proporções que os servidores. O terceiro é a compra do hardware certo para cada tipo de situação, onde se você precisa de um servidor para armazenar cache, pode-se comprar uma máquina com uma boa quantidade de memória RAM e com CPU e HD mais barato, assim a utilidade do servidor vai ser otimizada, onde o servidor vai fazer um uso intenso da RAM e em relação ao processamento e armazenamento em disco, quase não será usado.

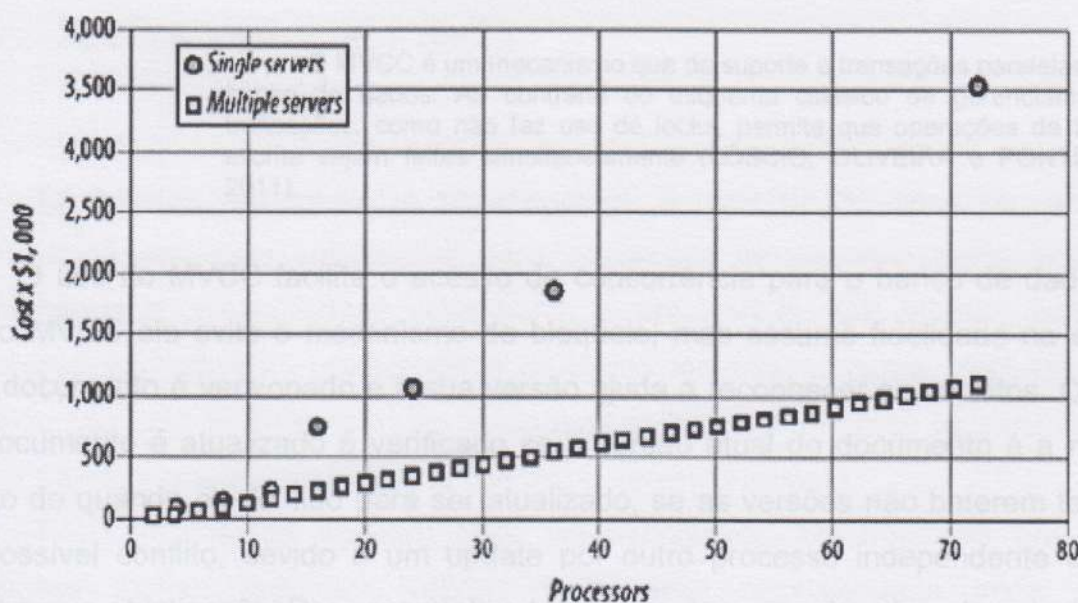
Outro ponto que pode dificultar o modelo de escalabilidade horizontal é se a arquitetura do software não estiver preparada para escalar de forma horizontal, sendo assim adicionar mais servidores se tornará dispendioso e sem utilidade.

6.2.4 Escalabilidade Vertical

Chama-se de escalabilidade vertical quando um sistema aumenta sua capacidade física, seja, trocando o servidor que o hospeda por um mais potente ou trocando partes internas como memória RAM, HD ou CPU (SOBRINHO, 2009).

De acordo com Henderson (2006), o princípio de escalabilidade vertical é simples, se começa com uma configuração de máquina básica e quando essa configuração chega ao seu limite será substituído por outra configuração e assim sucessivamente (HENDERSON, 2006). Esse tipo de escalabilidade resolve o problema com a melhora do hardware atual, porém nele reside um problema o custo desse modelo não é linear, fazendo uma comparação somente com o aumento do número de processadores um servidor com quatro CPUs custa aproximadamente o mesmo que quatro servidores com apenas uma CPU, mas quando se escala para um servidor de 72 CPUS o custo aumenta aproximadamente 3,5 vezes a mais do que 72 servidores com apenas uma CPU, vide gráfico 1.

Gráfico 1 - Escalabilidade vertical com pequenos e grandes servidores



Fonte: Henderson (2006)

Figura 10 – Controle de Concorrência: Bloqueios x MVCC

Segundo Sobrinho (2009), mesmo que o custo dessas melhorias não seja um limitador, em um determinado ponto se alcançaria o limite imposto pelo fabricante, seja no número de CPUs ou quantidade de memória, que um único servidor pode suportar. Outro ponto importante é, mesmo que o hardware seja eficiente a aplicação e sua arquitetura também devem ser, onde, quando o hardware cresce em números de CPUs e o sistema tem poucos processos paralelos, o que geralmente leva a baixa utilização das múltiplas CPUs. Quando se usa escalabilidade vertical visando aumentar o desempenho no banco de dados temos outro limitador, por exemplo, por mais hardware que seja usado o problema pode estar na conexão com o banco, por esse motivo por mais que seja aumentado a potencia do hardware pode não resolver o problema.

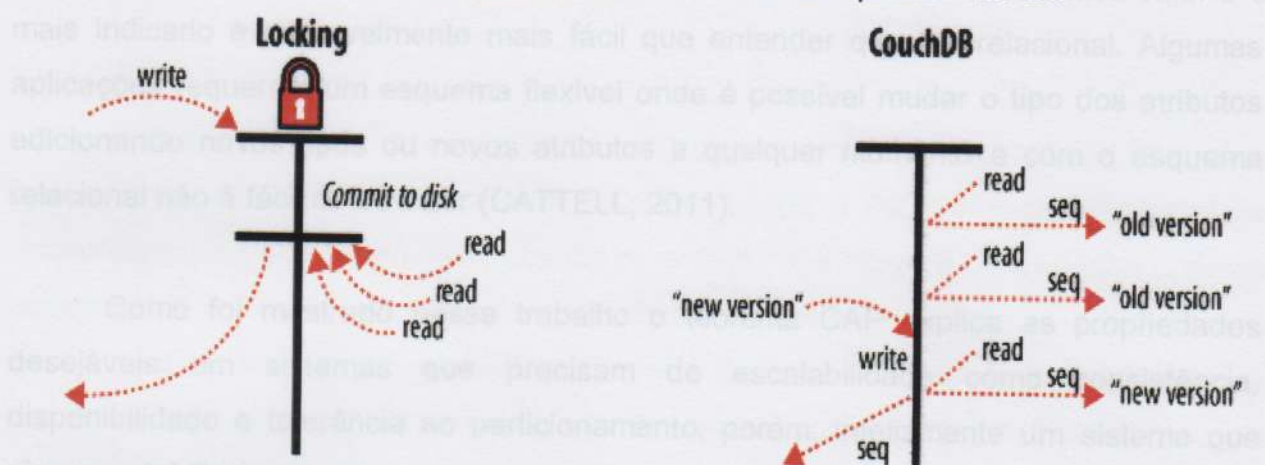
6.3 Controle de Concorrência de Multi-versões

De acordo com Orend (2010), o MVCC (Multiversion Concurrency Control) é um método eficiente para deixar múltiplos processos acessarem o mesmo dado em paralelo sem corromper o dado, porém há uma possibilidade de *deadlock*.

O MVCC é um mecanismo que dá suporte a transações paralelas em um banco de dados. Ao contrário do esquema clássico de gerenciamento de transações, como não faz uso de locks, permite que operações de leitura e escrita sejam feitas simultaneamente (LÓSCIO, OLIVEIRA e PONTES, p.5, 2011).

O uso do MVCC facilita o acesso de concorrência para o banco de dados. Ao usar o MVCC ele evita o mecanismo de bloqueio, mas assume fidelidade na escrita, cada documento é versionado e a sua versão ajuda a reconhecer os conflitos. Quando um documento é atualizado é verificado se a versão atual do documento é a mesma versão de quando ele foi lido para ser atualizado, se as versões não baterem teremos um possível conflito, devido a um update por outro processo independente entre a leitura e sua atualização. Por esse motivo temos um desempenho elevado e problemas de consistência (TIWARI, 2011).

Figura 10 – Controle de Concorrência: Bloqueios x MVCC



Fonte: Brito (2010)

A figura 10 ilustra como o controle de concorrência funciona em relação ao sistema de bloqueios dos modelos relacionais, onde com a estratégia do MVCC permite que as requisições de dados possam ser realizadas em paralelo, onde a ideia é criar múltiplas versões dos documentos e permitir a atualização de uma dessas versões, mantendo ainda a versão desatualizada, desse modo não há necessidade de bloquear os dados.

6.4 Escalabilidade entre RDBS e NoSQL

Os bancos relacionais já se encontram a mais de 30 anos no mercado e atendem as principais solicitações para aplicações de pequeno porte, oferecendo desempenho e escalabilidade semelhante a um NoSQL. Os SGBDR podem ser escalados verticalmente resolvendo o problema instantaneamente. Tem um padrão definido com uso de SQL que dão vantagens as junções tornando as consultas mais fáceis devido a grande aceitação do uso pelos desenvolvedores.

Em contrapartida nenhum teste de benchmarks mostrou que um SGBDR escalasse melhor que um NoSQL, onde essa escalabilidade pode alcançar desempenhos como o sistema BigTable do Google. Se você necessita de uma

pesquisa baseada em uma chave única, então um banco de dados de chave-valor é o mais indicado e provavelmente mais fácil que entender que um relacional. Algumas aplicações requerem um esquema flexível onde é possível mudar o tipo dos atributos adicionando novos tipos ou novos atributos a qualquer momento e com o esquema relacional não é fácil de se fazer (CATTELL, 2011).

Como foi mostrado nesse trabalho o teorema CAP explica as propriedades desejáveis em sistemas que precisam de escalabilidade como consistência, disponibilidade e tolerância ao particionamento, porém, infelizmente um sistema que visa escalabilidade somente pode ter duas dessas três características, sendo as três impossíveis de ter ao mesmo tempo (SOBRINHO, 2009). Quando um sistema necessita que a consistência seja prioridade ele deve escolher entre disponibilidade e consistência ao particionamento, onde terá que arcar com as consequências de que o sistema não estará disponível em determinados momentos, esse modelo usará o ACID para manter a consistência. Ao contrário de sistemas que tem como prioridade a disponibilidade, onde devem optar pela tolerância ao particionamento deixando de lado a consistência dos dados, este modelo usa o BASE para priorizar disponibilidade.

Porém ao decidir escolher uma solução em banco de dados deve-se fazer uma pergunta: O seu sistema precisa ser altamente disponível ou manter consistência todo o tempo? Dependendo da resposta a essa simples pergunta pode chegar a conclusão que de fato irá solucionar o seu problema descrito no projeto do sistema.

Adicionalmente, deve-se ter em mente que a escalabilidade em alto grau é necessária apenas em bancos de dados de grande porte, onde a sua disponibilidade é imprescindível, a escolha de soluções não relacionais para o caso que a disponibilidade não seja o fator crucial para o problema a ser solucionado deverá ser avaliada e trabalhada com bastante atenção, onde se pode ter sérios problemas ao se escolher uma solução errada. Pode-se observar a Tabela 1 a seguir para ver os pontos fortes e fracos de cada modelo.

7 CONCLUSÕES E PERSPECTIVAS

A decisão de optar por soluções de abordagens NoSQL para suprir as limitações que os bancos de dados relacionais possuem devido a sua natureza estruturada e normalizada, deve levar em consideração os critérios relevantes a necessidade onde as abordagens descritas nesse trabalho poderão estabelecer a solução ideal para o problema a ser solucionado. Importantes pontos devem ser considerados no âmbito da escalabilidade são eles: a necessidade de prover consistência, disponibilidade do sistema ou tolerância ao particionamento para uma melhor escalabilidade e também até mesmo a necessidade de se fazer consultas em ambientes distribuídos.

Apesar de existirem muitas opções no mercado atualmente e serem das mais variadas, a seleção de uma solução NoSQL ou relacional não atenderia todos os requisitos necessários para uma solução ideal, com escalabilidade e consistência.

Nesse trabalho foram apresentadas as vantagens e desvantagens dos modelos relacionais e não relacionais. Um modelo visando atender as necessidades onde outro modelo ainda não consegue atingir, porém ainda deixa a desejar em partes como consistência e outro modelo que já tem uma longa experiência no mercado e que atualmente atende as necessidades da maioria onde a robustez do sistema ainda não amadureceu.

Porém ao decidir escolher uma solução em banco de dados deve-se fazer uma pergunta: O seu sistema pretende ser, altamente disponível ou manter consistência todo o tempo? Dependendo da resposta a essa simples pergunta pode chegar a conclusão que de fato irá solucionar o seu problema descritos no projeto do sistema.

Adicionalmente, deve-se ter em mente que a escalabilidade em alto grau faz-se necessária apenas em bancos de dados de grande porte, onde a sua disponibilidade é imprescindível, a escolha de soluções não relacionais para o caso que a disponibilidade não seja o fator crucial para o problema a ser solucionado deverá ser estudada e trabalhada com bastante atenção, onde se pode ter sérios problemas ao se escolher uma solução errada. Pode-se observar a tabela 1 a seguir para ver os pontos fortes e fracos de cada modelo.

Tabela 1 – Análise Comparativa entre o Modelo Relacional e NoSQL

	Relacional	NoSQL
Escalonamento	Possível, mas complexo, devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos nós no grid não é realizada de modo natural.	Uma das principais vantagens desse modelo, pois, não possui nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos.
Consistência	Ponto mais forte do modelo relacional, onde as regras de consistência presentes propiciam um maior grau de rigor quanto a consistência das informações.	Realizada de modo eventual no modelo: só garante que se nenhuma atualização for realizada sobre o item de dados, todos os acessos a esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco de dados.	Outro fator fundamental do sucesso desse modelo. O Alto grau de distribuição dos dados propicia um maior número de solicitações aos dados seja atendida por parte do sistema e que o sistema fique menos tempo não disponível.

Fonte: Brito (2010)

Uma perspectiva para o futuro dos bancos de dados relacionais:

- Tem uma participação significativa no mercado de dados escalável se suas transações e consultas forem limitadas a nós únicos, dessa maneira eles serão uma escolha perfeita por causa do ACID (CATTELL, 2011).

Uma perspectiva para o futuro dos bancos de dados não relacionais:

- Os bancos de dados NoSQL não são uma moda passageira, onde sua simplicidade, flexibilidade e escalabilidade preenchem um nicho do mercado de sistemas onde é necessário milhões de acesso com leitura e escrita (CATTELL, 2011).

BROWNE, Julian. Brewer's CAP Theorem. 2009. Disponível em: <<http://www.jllianbrowne.com/article/viewer/brewers-cap-theorem>>. Acesso em: 15 nov. 2010.

CATTELL, Rick. Scalable SQL and NoSQL Data Stores. 2011. Disponível em: <<http://www.catteall.net/datastores/Datastores.pdf>>. Acesso em: 13 nov. 2011.

CHODURA, Kristina; DIROLI, Nosh. MongoDB: The Definitive Guide. California: No Starch Press, 2010. 218p.

COUCHBASE. NoSQL Database Technology: Post-relational data management for interactive software systems. Disponível em: <<http://www.couchbase.com/files/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf>>. Acesso em: 02 nov. 2011.

DEAN, Jeffrey; GHEMAWAT, Sanjoy. MapReduce: simplified data processing on large clusters. ACM Queue, vol. 6, 2004. Disponível em: <<http://dl.acm.org/citation.cfm?id=1251264>>. Acesso em: 13 nov. 2011.

DECANDIA, Giuseppe; HASTORUN, O.; 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 5, Dezembro 2007, SOSP '07, p. 205-220.

REFERÊNCIAS

- DIANA, Mauricio D.; GEROSA, Marco Aurelio. **NoSQL na Web 2.0: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0**. Disponível em: <<http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>>. Acesso em: 05 nov. 2011.
- ANDERSON, J. Chris; LEHNARDT, Jan; SLATER, Noah. **CouchDB: The Definitive Guide**. Califórnia: Editora O'Reilly Media Inc., 2010. 272p.
- BARCELAR, Ricardo. R. **Banco de Dados**: Introdução ao estudo de banco de dados. Disponível em: <http://www.ricardobarcelar.com.br/aulas/bd/ebook_bdV1-2011.pdf>. Acesso em: 05 nov. 2011.
- BRITO, Ricardo W. **Banco de Dados NoSQL x SGBDs Relacionais**: Análise Comparativa. Faculdade Farias de Brito e Universidade de Fortaleza, 2010. Disponível em: <<http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>>. Acesso em: 05 nov. 2011.
- BROWNE, Julian. **Brewer's CAP Theorem**, 2009. Disponível em: <<http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>>. Acesso em: 15 nov. 2010.
- CATTEL, Rick. **Scalable SQL and NoSQL Data Stores**. 2011. Disponível em: <<http://www.cattell.net/datastores/Datastores.pdf>>. Acesso em: 13 nov. 2011.
- CHODOROW, Kristina; DIROLF, Noah. **MongoDB: The Definitive Guide**. Califórnia: Editora O'Reilly Media Inc., 2010. 216p.
- COUCHBASE. **NoSQL Database Technology**: Post-relational data management for interactive software systems. Disponível em: <<http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf>>. Acesso em: 02 nov. 2011.
- DEAN, Jeffrey; GHEMAWAT, Sanjay. **MapReduce**: simplified data processing on large clusters. ACM Queue, vol. 6, 2004. Disponível em: <<http://dl.acm.org/citation.cfm?id=1251264>>. Acesso em: 13 nov. 2011.
- 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.

DIANA, Mauricio D.; GEROSA, Marco Aurélio. **NOSQL na Web 2.0: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0.** Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wtdbd/2010/sbbd_wtd_12.pdf>. Acesso em: 17 nov. 2011.

ELEMAR. **Cap Theorem e alternativa para o ACID.** Disponível em: <<http://elemarjr.net/2011/08/11/cap-theorem-e-alternativa-para-o-acid/>>. Acesso em: 20 nov. 2011.

ELMASRI R.; NAVATHE S. B. **Sistemas de banco de dados.** Tradução por: Daniel Vieira, 6. ed. São Paulo: Editora Pearson, 2011. 770p.

ESCALABILIDADE. **Introdução ao NoSQL parte I.** Disponível em: <<http://escalabilidade.com/2010/03/08/introducao-ao-nosql-parte-i/>>. Acesso em 19 nov. 2011

FOX, Armando.; BREWER, Eric. A.; **Harvest, Yield and Scalable Tolerant Systems.** In: Workshop on Hot Topics in Operating Systems, p. 174-178. 1999. Disponível em: <<http://radlab.cs.berkeley.edu/people/fox/static/pubs/pdf/c18.pdf>>. Acesso em: 13 nov. 2011.

GILBERT, Seth; LYNCH, Nancy. **Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services.** 2002. Disponível em: <<http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf>>. Acesso: em: 12 nov. 2011.

HENDERSON Cal. **Building Scalable Web Sites.** California: Editora O'Reilly Media Inc., 2006. 348p.

KORTH, Henry F.; SILBERSCHATZ, Abraham. **Sistema de Banco de Dados.** Tradução por: Maurício Heihachiro G. Abe, 2. ed. São Paulo: Editora MAKRON Books, 1995. 754p.

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

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN S. **Sistema de Banco de Dados.** Tradução por: Daniel Vieira, 5. ed. São Paulo: Editora Campus, 2006. 761p.

LEITE, Gleidson Sobreira. **Análise Comparativa do Teorema CAP Entre Bancos de Dados NoSQL e Bancos de Dados Relacionais**. 2010. 49p. Dissertação (Monografia em Ciência da Computação). Faculdade Farias Brito, Fortaleza, 2010.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues; PONTES, Jonas C. de Souza. **NoSQL no desenvolvimento de aplicações Web colaborativas**. Disponível em: <http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf>. Acesso em: 13 nov. 2011.

TAKAI, O. K.; ITALIANO, L. C.; FERREIRA, J. E. **Introdução a Banco de Dados**. MAPREDUCE. **Programming Model Overview**. Disponível em: <<http://map-reduce.wikispaces.asu.edu/>>. Acesso em: 22 nov. 2011.

MAZETTO, Gabriel Gonçalves Nunes. **Ruby on Rails Escalável: Estudo e Soluções**. 2011. 77p. Dissertação (Trabalho de Conclusão de Curso para Bacharel em Sistemas de Informação). Universidade Federal de Santa Catarina, Florianópolis, 2011.

WEL, Zhou; PIERRE, Guillaume; CHI, Chih-hung. **Scalable Transactions for Web**. NOSQL. **Your Ultimate Guide to the Non - Relational Universe**. Disponível em: <<http://nosql-database.org/>>. Acesso em: 15 nov. 2011.

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

PLUGGE, Eelco; MEMBREY, Peter; HAWKINS, Tim. **The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing**. New York: Editora Apress, 2010. 329p.

PRITCHETT, D. **BASE: An Acid Alternative**. ACM Queue, vol. 6, no. 3, Julho, 2008. Disponível em: <<http://queue.acm.org/detail.cfm?id=1394128>>. Acesso em: 13 nov. 2011.

ROB, Peter; CORONEL, Carlos. **Sistemas de Banco de Dados: Projeto, Implementação e Administração**. Tradução por: All Tasks, 8. ed. São Paulo: Editora Cengage Learning, 2011. 711p.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN S. **Sistema de Banco de Dados**. Tradução por: Daniel Vieira, 5. ed. São Paulo: Editora Campus, 2006. 781p.

SOBRINHO, David Lojudice. **Avaliação de Modelos de Arquiteturas de Web Sites de Alta Escalabilidade**. 2009. 69p. Dissertação (Pós-graduação em Tecnologia em Análise e Projeto de Sistemas). Faculdade de Tecnologia de São Paulo, São Paulo, 2009.

STRAUCH, Christof. **NoSQL Databases**. Disponível em: <<http://www.christof-strauch.de/nosql dbs.pdf>>. Acesso em 15 nov. 2011.

TAKAI, O. K.; ITALIANO, I. C.; FERREIRA, J. E. **Introdução a Banco de Dados**. Departamento de Ciência e Tecnologia, São Paulo fev. 2005. Disponível em: <<http://www.ime.usp.br/~jef/apostila.pdf>>. Acesso em: 05 nov. 2011.

TIWARI, Shashank. **Professional NoSQL**. Indianapolis: Editora John Wiley & Sons, Inc., 2011. 386p.

WEI, Zhou; PIERRE, Guillaume; CHI, Chi-Hung. **Scalable Transactions for Web Applications in the Cloud**. In proc. of the Euro-Par Conference, Janeiro 2009. Disponível em: <http://www.globule.org/publi/STWAC_europar2009.pdf>. Acesso em: 04 nov. 2011.

YANG, Hung-chih; DASDAN, Ali; HSIAO, Ruey-Lung; PARKER, D. Stott. **Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters**. 2008. Disponível em: <<http://www.systems.ethz.ch/education/past-courses/hs08/map-reduce/reading/mapreducemerge-sigmod07.pdf>> Acessado em 03 nov. 2011.