# **Minicurso**

1

# Bancos de Dados NoSQL: Conceitos, Ferramentas, Linguagens e Estudos de Casos no Contexto de Big Data

Marcos Rodrigues Vieira<sup>1</sup>, Josiel Maimone de Figueiredo<sup>2</sup>, Gustavo Liberatti<sup>2</sup>, Alvaro Fellipe Mendes Viebrantz<sup>2</sup>

<sup>1</sup>IBM Research Laboratory - Brazil mvieira@br.ibm.com

<sup>2</sup>Instituto de Computação Universidade Federal de Mato Grosso (UFMT) josiel@ic.ufmt.br, {liberatti.gustavo, alvarowolfx}@gmail.com

### Abstract

This lecture presents the data models, tools and languages concepts related to NoSQL databases, with a big emphasis on the Big Data domain. The main goal of this lecture is to provide an introductory, comparative and practical view of the definitions and the main tools and languages for NoSQL products currently available in the market. Since the definitions and tools for NoSQL are evolving rapidly in the past years, here we give a special focus on the most used and well-known NoSQL technologies for manipulating very large volumes of data. As a practical example, we present two use cases of NoSQL applications in different domains: (1) a use case involving sensor datasets from micrometeorological towers; and (2) a use case of NoSQL technologies applied to a large volume of textual dataset representing patent documents. We also discuss the impacts and consequences of the adoption of NoSQL products by the industry, IT staff, and the scientific database community.

### Resumo

Este minicurso apresenta os conceitos relacionados a modelagem, ferramentas e linguagens de manipulação de banco de dados NoSQL, com principal enfoque para Big Data. Este minicurso tem como objetivo principal dar um visão introdutória, comparativa e prática dos conceitos e das principais ferramentas e linguagens NoSQL disponíveis no mercado para diferentes domínios de problemas e aplicações. Como o conceito e ferramentas de NoSQL estão em rápida evolução nos últimos anos, o enfoque principal é voltado para as tecnologias mais adotadas e difundidas de manipulação de grande volume de dados utilizando ferramentas NoSQL. Como exemplo pratico, é apresentado dois estudos de casos de uso NoSQL em diferentes domínios de aplicações: (1) estudo de caso envolvendo dados de sensores presentes em torres micrometeorológicas; e (2) a utilização de tecnologias NoSQL aplicadas para dados textuais em grande bases de patentes. Por fim, são apresentados os impactos da adoção desses produtos na indústria, aos usuários de TI, e também a comunidade científica de banco de dados.

# 1.1. Introdução

Um dos grandes desafios atualmente na área de Computação é a manipulação e processamento de grande quantidade de dados no contexto de *Big Data*. O conceito *Big Data* pode ser resumidamente definido como uma coleção de bases de dados tão complexa e volumosa que se torna muito difícil (ou impossível) e complexa fazer algumas operações simples (e.g., remoção, ordenação, sumarização) de forma eficiente utilizando Sistemas Gerenciadores de Bases de Dados (SGBD) tradicionais. Por causa desse problema, e outros demais, um novo conjunto de plataformas de ferramentas voltadas para *Big Data* tem sido propostas, como por exemplo Apache Hadoop [3].

A quantidade de dados gerada diariamente em vários domínios de aplicação como, por exemplo, da Web, rede sociais, redes de sensores, dados de sensoriamento, entre diversos outros, estão na ordem de algumas dezenas, ou centenas, de Terabytes. Essa imensa quantidade de dados gerados traz novos grandes desafios na forma de manipulação, armazenamento e processamento de consultas em várias áreas de computação, e em especial na área de bases de dados, mineração de dados e recuperação de informação. Nesse contexto, os SGBD tradicionais não são os mais adequados, ou "completos", às necessidades do domínio do problema de *Big Data*, como por exemplo: execução de consultas com baixa latência, tratamento de grandes volumes de dados, escalabilidade elástica horizontal, suporte a modelos flexíveis de armazenamento de dados, e suporte simples a replicação e distribuição dos dados.

Uma das tendências para solucionar os diversos problemas e desafios gerados pelo contexto *Big Data* é o movimento denominado NoSQL (*Not only SQL*). NoSQL promove diversas soluções inovadoras de armazenamento e processamento de grande

volume de dados. Estas soluções foram inicialmente criadas para solucionar problemas gerados por aplicações, por exemplo, Web 2.0 que na sua maioria necessitam operar com grande volume de dados, tenham uma arquitetura que "escale" com grande facilidade de forma horizontal, permitam fornecer mecanismos de inserção de novos dados de forma incremental e eficiente, além da necessidade de persistência dos dados em aplicações nas nuvens (*cloud computing*). Essas diversas soluções vêm sendo utilizadas com muita frequência em inúmeras empresas, como por exemplo, IBM, Twitter, Facebook, Google e Yahoo! para o processamento analítico de dados de *logs* Web, transações convencionais, entre inúmeras outras tarefas.

Atualmente, existe uma grande adoção e difusão de tecnologias NoSQL nos mais diversos domínios de aplicação no contexto de *Big Data*. Esses domínios envolvem, em sua maioria, os quais os SBGD tradicionais ainda são *fortemente* dominantes como, por exemplo, instituições financeiras, agências governamentais, e comercio de produtos de varejo. Isto pode ser explicado pelo fato que existe uma demanda muito grande para soluções que tenham alta flexibilidade, escalabilidade, performance, e suporte a diferentes modelos de dados complexos.

Podemos basicamente resumir as características de Big Data em quatro propriedades: (1) dados na ordem de dezenas ou centenas de Terabytes; (2) poder de crescimento elástico horizontal; (3) fácil distribuição dos dados e/ou processamento; e (4) tipos de dados variados, complexos e/ou semiestruturados. A característica de manipulação de dados na ordem (ou maior) de Terabytes envolve, entre outros aspectos, o requisito de alto poder computacional de processamento, manipulação e armazenamento de dados. O poder de crescimento elástico esta relacionado ao fato de que a quantidade de dados pode variar de alguns Megabytes a várias centenas de Terabytes (e vice-versa) em um espaço de tempo relativamente curto, fazendo com que a estrutura de hardware/software demandada tenha que se adaptar, i.e. seja alocada/desalocada sob demanda da aplicação. A distribuição significa que os dados devem ser distribuídos de forma transparente em vários nós de processamento, o que demanda armazenamento e processamento distribuído. E a quarta característica esta relacionada a adoção de modelos mais apropriados, flexíveis e eficientes para o armazenamento de tipos de dados complexos, variados e semiestruturados. Vale ressaltar que o modelo relacional tradicional não é o mais adequado para tais propriedades acima citadas pois não possui suficiente flexibilidade para o armazenamento de dados e na evolução do modelo de dados.

Em relação a SGBD tradicionais, a distribuição dos dados de forma elástica é inviabilizado pois o modelo de garantia de consistência é *fortemente* baseado no controle transacional ACID (*Atomicity*, *Consistency*, *Isolation* e *Durability*). Esse tipo de controle transacional é praticamente inviável quando os dados e o processamento são distribuídos em vários nós. O teorema CAP (*Consistency*, *Availability* e *Partition tolerance*) mostra que somente duas dessas 3 propriedades podem ser garantidas simultaneamente

em um ambiente de processamento distribuído de grande porte. A partir desse teorema, os produtos NoSQL utilizam o paradigma BASE (*Basically Available*, *Soft-state*, *Eventually consistency*) para o controle de consistência, o que consequentemente traz uma sensível diminuição no custo computacional para a garantia de consistência dos dados em relação a SGBD tradicionais.

Dentro do aspecto do processamento dos dados, o principal paradigma adotado pelos produtos NoSQL é o *MapReduce* [13]. Em resumo, tal paradigma divide o processamento em duas etapas: (1) *Map*, que mapeia e distribui os dados em diversos nós de processamento e armazenamento; e (2) *Reduce*, que agrega e processa os resultados parciais para gerar um resultado final (ou intermediário para outro processo MapReduce). Dentre os vários produtos NoSQL existentes, podemos considerar que o mais representativo é o Apache Hadoop [3], cuja implementação do algoritmo MapReduce é hoje considerada a referência. Atualmente existem diversas grandes e médias empresas que estão utilizando Hadoop para as mais diversas finalidades, como IBM, Google, Twitter, Yahoo!, Netflix, Facebook, e algumas agências financeiras internacionais.

Além do Apache Hadoop, existem vários outros produtos NoSQL disponíveis no mercado, onde eles se diferem principalmente na complexidade para o suporte a diferentes tipos de dados, como por exemplo documentos, dados em forma de *streams*, dados semiestruturados e em forma de grafos, entre outros. Alguns exemplos de serviços baseados em Hadoop são: Amazon Elastic MapReduce, Oracle BigData Appliance, EMC Greenplum, Teradata, Microsoft Windows Azure.

Nesse contexto de grande adoção e uso de sistemas baseados em tecnologias NoSQL, este minicurso tem como objetivo principal introduzir os conceitos relacionados a NoSQL nos mais variados domínios de aplicações de *Big Data*.

# 1.1.1. Descrição da Estrutura do Minicurso

Este mini-curso esta estruturado da seguinte forma:

- 1. Introdução, Motivação e Conceitos em Big Data (duração 30 minutos): este módulo faz uma introdução, motivação e descrição dos conceitos relacionados ao contexto de Big Data. Esta parte faz um enfoque nos principais desafios trazidos por esse contexto no qual é demandado o gerenciamento de grande quantidade de informações não estruturadas e semiestruturadas de forma distribuída, com estrutura elástica, e uso em ambiente computacional de nuvem (cloud computing);
- 2. **Descrição e Conceitos de NoSQL (duração 60 minutos)**: este módulo cobre os conceitos relacionados a NoSQL no contexto de *Big Data*, tais como os produtos, linguagens de acesso, manipulação e processamento dos dados. Além disso, esta parte visa dar uma visão geral dos diferentes produtos NoSQL disponíveis

no mercado, como por exemplo: Column Store (e.g., Hadoop, Casandra, Amazon SimpleDB, Hypertable), Document Store (e.g., Apache CouchDB, MongoDB), Key-Value/Tuple Store (e.g., Voldemort, Redis, MemcacheDB), Graph Databases (e.g., Neo4J, HyperGraphDB). Assim, serão abordadas implementações existentes para os problemas de distribuição, transações, mineração e a infraestrutura computacional de nuvem;

- 3. **Estudo de caso 1 (duração 60 minutos)**: este módulo apresenta um estudo de caso de solução NoSQL no contexto de *Big Data* de dados textuais a partir de uma aplicação de bases de patentes. Este módulo descreve as soluções escolhidas, suas vantagens/desvantagens, e suas implementações;
- 4. **Estudo de caso 2 (duração 60 minutos)**: este módulo descreve o uso de *streams* de dados originados de sensores em torres micrometeorológicas. Este módulo descreve a forma de manipular essas informações dentro da solução de produtos NoSQL;
- 5. Desafios, direções futuras, problemas em aberto (duração 30 minutos): este módulo apresenta os desafios de adoção desses novos produtos por ambientes tradicionais de desenvolvimento de sistemas baseados na tecnologia relacional. É também levantado o problema de adaptação do ambiente tradicional, no qual diversas padronizações já consolidadas (e.g., padrões de projetos, bibliotecas, *drivers* de conexão, mapeamento objeto-relacional) precisam ser adaptadas para obtenção das informações originárias dos produtos NoSQL. Será abordado também o impacto desses novos produtos nos atores envolvidos, ou seja, programadores e administradores (DBA).

### 1.2. Conceitos Relacionados a NoSQL

Embora as terminologias relacionadas ao contexto NoSQL ainda são inconsistentes e sem padronização, aqui definimos os termos e conceitos mais utilizados. Para o bom entendimento do contexto NoSQL, é preciso primeiramente a definição de alguns conceitos fundamentais que estão relacionados com essa tecnologia, bem como as influências que direcionam o desenvolvimento de novos produtos relacionados a NoSQL. Portanto, a abordagem utilizada neste minicurso é a de descrever as influências externas e, posteriormente, abordar as consequências internas para produtos NoSQL, ou seja, como as atuais soluções foram implementadas visando solucionar os problemas existentes. Por motivos já citados anteriormente, este minicurso não tem a intenção de cobrir de forma extensiva e completa os conceitos relacionados ao domínio NoSQL. No entanto, aqui fornecemos várias referências para trabalhos que tenham uma explicação e definição mais aprofundada relacionados ao tema aqui exposto.

Os trabalhos disponíveis na literatura relacionados a NoSQL explicam, em sua grande maioria, o surgimento de NoSQL no contexto de grande quantidade de dados gerados em um espaço de tempo (relativamente) curto. Como consequência dessa grande quantidade de dados, sistemas disponíveis para a manipulação desses dados gerados necessitam de um grande poder de processamento de forma eficiente e escalável. Além da alta taxa de geração dos dados, outro fator que influenciou a criação de sistemas NoSQL foi o suporte a tipo de dados complexos, semiestruturados ou não estruturados. Além do volume de geração desses dados ser grande, outro fator predominante é relacionada a dificuldade de modelagem de tais tipos de dados. Esses tipos de dados estão hoje presentes em inúmeros domínios de aplicações, tais como Web 2.0, redes sociais, redes de sensores, entre outros.

Nas últimas décadas os SGBD tradicionais baseados no modelo relacional vêm incorporando novas características que transcendem o modelo relacional original. Exemplos dessas novas funcionalidades são o suporte a outros tipos de dados nativos, como objetos, temporal, espacial, XML, documentos de textos, entre outros.

Essas novas funcionalidades suportadas por SGBD são, basicamente, encapsulamentos de extensões do modelo objeto-relacional. Deste modo, as políticas de transação, replicação, segurança e gerenciamento dos dados continuam, em sua grande maioria, as mesmas pois não existem grandes grandes mudanças estruturais nos SGBD. Por outro lado, uma grande vantagem da adição de novas funcionalidades aos SGBD é que a estrutura corporativa que utiliza os SGBD não sofre grandes mudanças (e.g., os procedimentos de backup, segurança, continuam os mesmos). No entanto, essa abordagem de solução única adotada pelos SGBD relacionais vem sendo criticada por varias pessoas pois novas demandas de mercado requerem abordagens mais adequadas (e.g., a linguagem SQL não consegue representar todas as demandas pelas novas aplicações) [30].

# 1.2.1. Big Data

O termo *Big Data* é bem amplo e ainda não existe um consenso comum em sua definição. Porém, *Big Data* pode ser resumidamente definido como o processamento (eficiente e escalável) analítico de grande volumes de dados complexos produzidos por (várias) aplicações. Exemplos de aplicações no contexto *Big Data* varia bastante, como aplicações científicas e de engenharias, redes sociais, redes de sensores, dados de *Web Click*, dados médicos e biológicos, transações de comércio eletrônico e financeiros, entre inúmeras outras. As semelhanças entre os dados desses exemplos de aplicações incluem: grande quantidade de dados distribuídos, características de escalabilidade sob demanda, operações ETL (*Extract*, *Transform*, *Load* [27]) de dados "brutos" (*raw*) semi- ou não estruturados para dados estruturados e, a necessidade de extrair conhecimento da grande quantidade de dados [12].

Três fatores influenciaram o grande aumento de volume de dados sendo coleta-

dos e armazenados para posterior análise: difusão dos dispositivos captação de dados, dispositivo com armazenamento na ordem de Terabytes e aumento de velocidade de transmissão nas redes. Os dispositivos de aquisição, bem como os dispositivos de armazenamento de grande escala se difundiram principalmente pelo seu barateamento (e.g., redes de sensores, GPS, *smartphones*), enquanto que as redes aumentaram sua velocidade e abrangência geográfica. Outro fator importante é a facilidade de geração e aquisição de dados gerados digitalmente, como máquinas fotográficas digitais, *smartphones*, GPS, etc. Como consequência novas demandas estão surgindo, como a demanda por análise de grande volume de dados em tempo real (*data analytics*), o aumento do detalhamento das informações, bem como plataformas escaláveis e eficientes de baixo custo [10].

Basicamente, podemos resumir as características do contexto Big Data em quatro propriedades: (1) dados na ordem de dezenas ou centenas de Terabytes (podendo chegar a ordem de Petabytes), (2) poder de crescimento elástico, (3) distribuição do processamento dos dados; e (4) tipos de dados variados, complexos e/ou semiestruturados. A característica de análise dos dados na ordem de Terabytes envolve, entre outros aspectos, o requisito de alto poder computacional de armazenamento e processamento dos dados. A elasticidade esta relacionada ao fato de que a quantidade de dados pode variar de alguns Megabytes a vários Terabytes (e vice-versa) em um espaço de tempo relativamente curto, fazendo com que a estrutura de software/hardware adapta-se sob demanda, i.e. seja alocada/desalocada dinamicamente. A distribuição significa que os dados devem ser distribuídos de forma transparente em vários nós espalhados de processamento, o que demanda armazenamento, processamento e controle de falhas distribuído. Finalmente, a quarta característica esta relacionada a adoção de modelos mais apropriados, flexíveis e eficientes para o armazenamento de tipos de dados variados e semiestruturados. Vale ressaltar que o modelo relacional não é o mais adequado pois não possui flexibilidade para o armazenamento de dados e evolução no modelo para tais tipos de dados citados acima.

A análise de dados (*data analytics*) no contexto de *Big Data* normalmente envolve processamento da ordem de Terabytes em dados de baixo valor (i.e., informação original "bruta") que são transformados para dados de maior valor (e.g., valores agregados/sumarizados). Mesmo com a grande quantidade de dados *Big Data* em si não garante a qualidade da informação, pois a análise continua, em grande parte, sendo muito subjetiva. Isso se deve ao fato que os dados em si não são autoexplicativos, onde o processo de limpeza, amostragem, e relacionamento dos dados continua sendo crítico e passível a erros, aproximações e incertezas [14]. Por exemplo, a análise de dados da ordem de Petabytes (ou Exabytes) de cunho científicos (e.g., dados genômicos, física ambiental e simulações numéricas) tem se tornado muito comum hoje em dia, onde é aceitável que o resultado da análise contenham imprecisão (i.e., erro entre certos limites

de erros), porém seja computado de forma (relativamente) rápida e/ou em tempo real.

Recentemente, ambientes de computação em nuvem (*cloud computing*) têm sido utilizados para o gerenciamento de dados em forma de *Big Data*, enfocando principalmente em duas tecnologias: Bases de Dados Como Serviço (*Database as a Service* (DaaS)) e Infraestrutura Como Serviço (*Infrastructure as a service* (IaaS)) (para maiores detalhes, vide [21, 1]). DaaS utiliza um conjunto de ferramentas que permite o gerenciamento remoto dos servidores de dados mantidos por uma infraestrutura externa sob demanda. Essa infraestrutura IaaS fornece elasticidade, pagamento sob demanda, backup automáticos, e rapidez de implantação e entrega.

As principais características que envolvem os ambientes em nuvem são: escalabilidade, elasticidade, tolerância a falhas, auto gerenciamento e a possibilidade de funcionar em hardware *commodity* (comum). Por outro lado, a maioria dos primeiros SGBD relacionais comerciais foram desenvolvidos e concebidos para execução em ambientes corporativos. Em um ambiente de computação em nuvem traz inúmeros desafios do ponto de vista computacional. Por exemplo, o controle de transação na forma tradicional (i.e., definida pelas propriedades ACID) em um ambiente de nuvem é extremamente complexo.

De uma maneira geral, os ambientes em nuvem precisam ter a capacidade de suportar alta carga de atualizações e processos de análises de dados. Os *data centers* são uma das bases da computação em nuvem, pois uma grande estrutura como serviço escalável e dinâmica é fornecida para vários clientes. Um ambiente de gerenciamento de dados escalável (*scalable data management*) pode ser dividido em: (1) uma aplicação complexa com um SGBD gerenciando uma grande quantidade de dados (*single tenant*); e (2) uma aplicação no qual o ambiente deve suportar um grande número de aplicações com dados possivelmente não muito volumosos [2]. A influência direta dessas duas características é que o SGBD deve fornecer um mesmo esquema genérico para inúmeros clientes com diferentes aplicações, termo denominado bases de dados *multitenant*.

É importante lembrar que em ambientes *multitenant* a soma dos *tenant* pode gerar um grande volume de dados armazenado no SGBD. Este característica é apropriadamente gerenciada pelo ambiente em nuvem, onde novas instâncias de SGBD são criadas em novos nós e/ou servidores do ambiente (os dados de diferentes *tenant* são independentes entre si).

Em ambientes tradicionais, quando uma aplicação cresce sua complexidade o SGBD atende às requisições e a escalabilidade do sistema no modo que aumenta o seu poder computacional. Por exemplo, o poder computacional do sistema como um todo cresce a medida que mais memória e/ou número de nós do *cluster* são adicionados ao servidor. No entanto, esta abordagem são caras, dependentes de arquitetura, e normalmente não presentes em SGBD livres (*open sources*).

### 1.2.2. Consistência

Consistência pode ser definido como um contrato entre um processo e um dados armazenado (distribuídos), no qual o dados armazenado especifica precisamente qual é o resultado de operações de escrita e leitura na presença de concorrência. O modelo de consistência pode ser definido em dois casos: centrado nos dados e centrado no cliente [6]. Modelo de consistência centrado nos dados refere-se ao estado interno do sistema de armazenamento, ou seja, a consistência é alcançada no momento em que todas as cópias de um mesmo dado se tornam iguais. Já o modelo de consistência centrado no cliente fornece garantias apenas para um cliente (em específico) em relação a consistência de acesso aos dados do cliente em questão. Neste último caso, quando o cliente verifica o "quanto" eventual a consistência é (o que mede o tamanho da janela de inconsistência), duas perspectivas podem existir: a do ambiente servidor, e a do cliente. Para o servidor pode aplicar o gerenciamento centrado nos dados, enquanto que para o cliente não faz diferença o que está ocorrendo no servidor [32], pois o cliente possui controles próprios para identificar dados "caducos".

O interesse e uso do teorema CAP (*Consistency*, *Availability* e *Partition tole-rance*), também conhecido como Brewer's theorem [7], pelos SGBD têm atingido novas demandas em aplicações que requerem vários nós de processamento. O teorema CAP, resumidamente, afirma que existem três propriedades que são úteis em SGBD: C: consistência, cujo objetivo é permitir que transações distribuídas em vários nós agem com a semântica de "tudo-ou-nada", bem como (quando existirem) das réplicas estarem sempre em um estado consistente; A: disponibilidade, tem como objetivo manter o sistema sempre disponível, e em caso de falha o sistema deve continuar funcionando com alguma réplica dos recursos indisponíveis; P: tolerância a partições, cujo objetivo é manter o sistema operando mesmo no caso de falhas de rede, para isso é dividido o processamento dos nós em grupos que não se comunicam (os subgrupos continuam o processamento independentemente) [29]. O teorema CAP afirma que não é possível alcançar todos os três objetivos simultaneamente quando erros existem, porém uma das propriedades tem que ser desprezada.

Existem duas direções na escolha de A ou C: (1) a primeira necessita de forte consistência como propriedade para tentar maximizar a disponibilidade [18]. A vantagem de existir uma política forte de consistência é que as aplicações podem ser desenvolvidas com mais simplicidade. Por outro lado, controles complexos devem ser implementados nas aplicações caso não exista o controle de consistência; (2) a segunda direção prioriza disponibilidade e tenta maximizar a consistência, sendo que um dos argumentos é que o ambiente indisponível traz perdas financeiras para o usuário do serviço. Portanto o serviço estará sempre com os dados disponíveis, mesmo que em alguns momentos não esteja com os dados consistentes. Em aplicações Web que utilizam a escalabilidade horizontal é necessário decidir entre A ou C, enquanto que os SGBD

tradicionais preferem a propriedade C a A ou P.

No contexto NoSQL, o teorema CAP tem sido usado com a justificativa de desprezo da consistência, de forma que os produtos normalmente não permitem transações que ultrapassam mais de um nó. Ou seja, não há controle de réplicas pois o teorema CAP justifica esse ponto, que é substituída pela "consistência eventual". Nessa abordagem, é garantido que todas as réplicas eventualmente convergem ao mesmo estado no momento que a conectividade for restabelecida e passar o tempo necessário para o sincronismo finalizar. Portanto, a justificativa de desprezar a propriedade C é que as propriedades A e P continuam sendo garantidas.

Com relação a manipulação de dados complexos e não estruturados, o controle de tolerância a falhas pode ser complexo. Esta complexidade é que no contexto < key, value > o modelo de processamento é diferente de modelos como, por exemplo, XML que é fortemente baseado em hierarquia. Com o objetivo de fornecer melhor desempenho e alta escalabilidade, os produtos NoSQL (em contraste com a política de controles de transação do tipo ACID) utilizam a abordagem denominada BASE (Basically Available, Soft state, Eventually consistent). Esta abordagem envolve a eventual propagação de atualizações e a não garantia de consistência nas leituras.

A abordagem BASE é implementada de forma diferente em alguns produtos. Por exemplo, alguns produtos se denominam "eventualmente consistentes", porem fornecem algum tipo de consistência como a política de controle de concorrência multiversionada (MVCC) [9]. O leitor pode encontrar maiores detalhes da consistência eventual em [32].

## 1.2.3. Escalabilidade

Uma propriedade importante nos produtos NoSQL é o poder de escalar horizontalmente de forma não compartilhada (i.e., replicando e particionando os dados em diferentes servidores). Isto permite que um grande volume de operações de leitura/escrita possam ser executadas de forma muito eficiente.

Além dos conceitos de particionamento e distribuição serem bem definidos, não existe a aplicação do conceito de bases de dados federados em produtos NoSQL. Em produtos NoSQL, toda a base é considerada uma só, enquanto que em bases de dados federados é possível administrar e usar separadamente cada base de dados e, em alguns momentos, utilizar todas as bases como se fossem apenas uma única.

O conceito de escalabilidade vertical está relacionada com o uso de vários núcleos/CPU que compartilham memória e discos. Portanto, mais núcleos e/ou memórias podem ser adicionados para aumentar o desempenho do sistema, porém essa abordagem é limitada e normalmente é cara. Já a escalabilidade horizontal está relacionada com a funcionalidade de distribuição de dados e de carga por diversos servidores, sem o com-

partilhamento de memória ou disco. Esta última abordagem permite o uso de máquinas mais baratas e comuns (i.e., hardware *commodity*).

Escalabilidade dinâmica é uma das mais importantes e principais propriedades de um ambiente em nuvem, porém é um grande problema para os SGBD tradicionais. Por exemplo, Websites mais acessados são reconhecidos por sua massiva escalabilidade, baixa latência, possibilidade de aumentar (sob demanda) a capacidade da base de dados, e possuir um modelo de programação relativamente simples. Essas características não são encontradas em SGBD sem pagar um alto custo: somente a escalabilidade vertical é fácil de ser alcançada; já para a escalabilidade horizontal, os SGBD normalmente replicam os dados para manter a sincronização dos dados. Alguns ambientes utilizam grids para fornecerem escalabilidade horizontal em nível de sistema operacional (a escalabilidade e distribuição são mantidas pelo servidor de aplicação em nível de sistema operacional). Realizar essas ações com o SGBD envolvem o uso de soluções caras e que não atendem adequadamente a questão de elasticidade. Assim, essa estratégia de um modelo único de SGBD contemplar todos os modelos de dados está encontrando barreiras para o contexto de Big Data. Se por um lado os produtos objeto-relacional fornecem diversas funcionalidades de extensão e encapsulamento, por outro lado a manutenção da mesma plataforma corporativa está se tornando inviável financeiramente, pois a maioria das soluções são caras e feitas para funcionar em ambientes de *cluster* e nuvem.

SGBD voltados para operações de data warehouse fornece escalabilidade horizontal, contudo as consultas normalmente são complexas envolvendo várias junções com diferentes tabelas e a taxa de leituras sobre escritas é muito alta, ou seja, as operações executadas são em sua grande maioria de leituras.

Já a escalabilidade horizontal em produtos NoSQL (e.g., Apache Cassandra) é alcançada com o particionamento dos dados utilizando a técnica chamada Tabela Hash Distribuída (*Distributed Hash Table* (DHT)) [11]. Nesta técnica, as entidades dos dados são representadas por pares < *key*, *value* >, onde *key* é uma chave que identifica unicamente a entidade representada por *value*. O conjunto de entidades do domínio de dados são organizados em grupos que são colocados em um nó do ambiente.

A técnica *sharding* é outra técnica para particionamento horizontal dos dados em uma arquitetura sem compartilhamento de recursos. Diferentes das técnicas de divisão dos dados por colunas (i.e., técnicas de normalização e particionalmente vertical), na técnica *sharding* os dados de uma tabela são divididos por tuplas (*rows*. Cada partição forma parte de um *shard*, onde pode ser recuperada a partir de um SGBD específico. Existem inúmeras vantagens de particionamento usando esta técnica, por exemplo, como as tabelas estão divididas e distribuídas em múltiplos servidores, o número total de tuplas em cada tabela de cada servidor é reduzido. Consequentemente, o tamanho dos

índices também são reduzidos, o que geralmente melhora o desempenho de consultas. Outra vantagem é que uma base de dados onde o *shard* foi aplicado pode ser separado em diferentes máquinas, e múltiplos *shards* podem ser alocados em diferentes máquinas. Com isso, é possível distribuir uma base de dados em um número grande de máquinas, o que significa que o desempenho do SGBD pode ser espalhado por diferentes máquinas. No entanto, a técnica de *sharding* é relativamente difícil de ser implementada, pois a divisão dos dados muito geralmente é feita de forma estática e com conhecimento prévio da distribuição dos dados. Uma alternativa para este problema é a utilização de *hashing* para distribuição dos dados de uma forma mais dinâmica.

# 1.2.4. O Paradigma de Programação MapReduce

Nos últimos anos, o paradigma chamado *MapReduce* [13] tem ganhado grande atenção e muitos adeptos tanto na indústria quanto na área acadêmica. Este paradigma foi inicialmente proposto para simplificar o processamento de grande volume de dados em arquiteturas paralelas e distribuídas (e.g., clusters). O enfoque principal do *MapReduce* é tornar transparente os detalhes da distribuição dos dados e balanceamento de carga, permitindo que o programador enfoque somente nos aspectos de tratamento dos dados [24]. *MapReduce* é voltado para uso em *clusters* de computadores *commodities*, onde os dados são distribuídos e armazenados utilizando como pares < *key*, *value* > (vários valores podem ser armazenados para a mesma chave). Ambas as funções *Map* e *Reduce* são definidas com relação aos dados estruturados em pares < *key*, *value* >. Como consequência desta arquitetura, o paradigma *MapReduce* fornece uma abordagem altamente escalável voltado para o processamento intensivo de grande volume de dados (i.e., o alto desempenho da arquitetura é conseguida através da alocação dos dados e computação em um mesmo nó).

Provavelmente uma das maiores vantagens deste paradigma é a sua simplicidade, onde a manipulação dos dados é feita pelo uso de duas funções básicas: *Map* (função de mapeamento) e *Reduce* (função de redução). Ambas funções são codificadas pelo programador na linguagem do ambiente, que normalmente é uma linguagem procedural. Porém, alguns produtos possibilitam o uso de linguagens declarativas com SQL (e.g, Hive [4, 31]), que são depois transformadas em funções *Map* e *Reduce*.

Resumidamente, a função Map tem como entrada um par de dados, com o tipo in one domínio de dados, e retorna uma lista de pares em um domínio diferente. Isto é, a função  $Map(k_1,v_1) \rightarrow list(k_2,v_2)$ . A função Map é aplicada em paralela para cada par do conjunto de dados de entrada, gerando uma lista de pares para cada chamada). Em uma segunda etapa, o processo MapReduce coleta e agrupa todos os pares com a mesma key das listas, criando um grupo para cada diferente key gerada. Este conjunto de pares é construído utilizando uma função de mapeamento (hash) utilizando os valores key dos

pares, que então são distribuído em diversos nós para serem utilizado na fase Reduce.

A função Reduce é então aplicada a cada grupo em paralelo, onde é produzido uma coleção de valores em um mesmo domínio:  $Reduce(k_2, list(v_2)) \rightarrow list(v_3)$ . O conjunto de pares < key, value > de cada conjunto intermediário são processados por jobs independentes. Este processamento na fase Reduce gera, para cada nó, resultados intermediários ordenados por key, e que são direcionados para um arquivo no sistema de arquivos. Esse conjunto de arquivos deve ser então processados (i.e., agregados em um único resultado) para gerar o resultado final. A Figura 1.1 ilustra o paradigma MapReduce para contar palavras em um arquivo grande de texto.

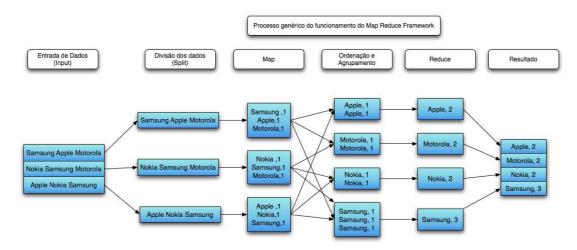


Figura 1.1: Exemplo de um processo *MapReduce* para contar palavras.

No momento em que um job é iniciado em um nó, os pares < key, value > são transferidos em partes aos nós de mapeamento. Uma nova instância de mapeamento é gerada para cada registro de entrada. Esses pares são coletados localmente no nó de mapeamento através de uma função em um nó de redução. Após receber os dados, os nós de redução ordenam os pares e geram novos pares do tipo  $< k_i, lista(v_i) >$ , cujo resultado é gravado em um arquivo.

Geralmente, a implementação do paradigma *MapReduce* envolve um sistema de arquivos distribuído, um *engine* distribuído que permite a execução de tarefas de mapeamento e redução nos nós hospedeiros, e a implementação de uma arquitetura de programação (e.g., formato de entrada/saída, funções de particionamento) [19]. Os detalhes de paralelização, organização de tarefas, acesso a dados concorrentes e controle de falhas são transparentes ao usuário. Portanto, esse nível de abstração permite que o usuário enfoque no algoritmo em si, em vez de, por exemplo, aspectos de controle de falhas e concorrência dos processos.

As implementações de *MapReduce* normalmente incluem uma API simples que descreve qual parte do processamento é feito em paralelo (etapa de **mapeamento**), e qual parte do processamento é feito depois do agrupamento em um nó (etapa de **redução**). O paradigma *MapReduce* não obriga o uso de um modelo de dados específico, o que permite que virtualmente qualquer tipo de dado possa ser utilizado [17].

Alguns exemplos de uso de *MapReduce* mais comuns são no contexto de *data analytics* (e.g., Web logs), onde normalmente todos os registros do conjunto de dados necessitam serem processados (a leitura dos dados é feito sequencialmente sem muita complexidade no acesso). No entanto, o paradigma *MapReduce* não é eficiente em tarefas onde o acesso é feito em subconjuntos dos dados (e.g., *sort-merge based grouping*) [16]. Análises complexas de dados em *MapReduce* são fornecidas por produtos que seguem enfoques direcionados, como PigLatin [5, 25] que tem desempenho adequado para agrupamentos e Jaql que tem integração com a plataforma R. Podemos considerar que o poder do paradigma *MapReduce* tem íntima relação com o poder de processamento existente em ambientes em nuvem. Ambientes *MapReduce* têm maior facilidade na carga de informações quando comparados com ambientes de SGBD tradicionais, pois os dados são armazenados em seu estado original (i.e., sem transformações), enquanto que a transformação dos dados é feita no momento do processamento da consultas.

Em relação ao contexto de SGBD, o paradigma MapReduce é considerado por alguns pesquisadores um retrocesso no processamento paralelo de dados [20]. Já os envolvidos nesse paradigma defendem que MapReduce não é um SGBD. Nos últimos anos, várias comparações foram feitas entre MapReduce e SGBD, com alguns casos de sucesso para ambos os lados (por exemplo, Hadoop sendo de 2-50 vezes mais lento que SGBD paralelos, ganhando somente nas operações de carga [26]). Hadoop é muito criticado por possuir uma baixa eficiência por nó (em torno de 5MB/s): o foco principal é na escalabilidade e tolerância a falhas, em detrimento a eficiência. Um exemplo de desempenho alcançado pelo Hadoop envolveu o uso de 3.800 nós para ordenamento de 100TB de dados, o que não levou em conta por exemplo aspectos como gasto de energia. Já os SGBD paralelos têm seu foco na eficiência explorando o pipeline de resultados intermediários entre operações de consulta. Exemplos apresentados em [28] ilustram que a maioria dos processamentos analíticos processam relativamente pouca quantidade de dados, na ordem de Gigabytes, como por exemplo 90% dos processos em um cluster Facebook tem a quantidade de dados menor que 100GB. Outro aspecto interessante é que inúmeros algoritmos são muito complexos para serem escaláveis e adaptados com o paradigma MapReduce, como no caso de algoritmos de aprendizado de máquina que são executados em processamentos analíticos.

Normalmente a entrada para o processamento *MapReduce* é um arquivo armazenados em blocos contínuos em um sistema de arquivos distribuído, onde as funções *Map* 

e *Reduce* processam cada arquivo em paralelo. Esse enfoque funciona adequadamente porque o contexto envolve a leitura sequencial de cada arquivo concorrente em cada nó. No entanto, o acesso a dados contínuos fisicamente não ocorre em diversas situações de processamento analítico de dados científicos (existe uma grande limitação e dificuldade no uso do paradigma *MapReduce* para dados científicos). Um exemplo disso é o armazenamento de matrizes, que logicamente uma posição linha *vs* coluna próximas pode não ser da mesma forma em relação ao armazenamento físico. Algumas soluções como SciHadoop tentam levar em conta ambas as organizações físicas e lógicas [8].

Existem também algumas soluções que propõem o uso integrado de SGBD Relacional e NoSQL. Em resumo, estas soluções armazenam os dados utilizando arquiteturas NoSQL, e utilizam o paradigma *MapReduce* para processos ETL, por exemplo geração de cubos em produtos Data Warehouse tradicionais. Existem diversas linguagens de consulta construídas sobre uma plataforma MapReduce, como Hive [4, 31] ou PigLatin [5, 25]. Essas linguagens possuem limitações para os programadores quando comparadas com o processamento *MapReduce* tradicional, incluindo otimizações voltadas para diminuir a transferência de dados pela rede.

### 1.3. Ferramentas

Os produtos NoSQL possuem várias características comuns entre si (vide Seção 1.2), porém se diferenciam quanto ao modelo de dados utilizados (i.e., os produtos são classificados pela representação dos dados). Atualmente, os principais produtos NoSQL disponíveis<sup>1</sup>, são organizados segundo seu modelo de dados a seguir:

- Baseado em Columa (*Column Stores*): Hbase, Cassandra, Hypertable, Accumulo, Amazon SimpleDB, Cloudata, Cloudera, *SciDB*, HPCC, Stratosphere;
- Baseado em Documentos (*Document Stores*): MongoDB, CouchDB, *BigCouch*, RavenDB, Clusterpoint Server, ThruDB, TerraStore, RaptorDB, JasDB, SisoDB, SDB, SchemaFreeDB, djondb;
- Baseado em Grafos (*Graph-Based Stores*): Neo4J, Infinite Graph, Sones, InfoGrid, HyperGraphDB, DEX, Trinity, AllegroGraph, BrightStarDB, BigData, Meronymy, OpenLink Virtuoso, VertexDB, FlockDB;
- Baseado em Chave-Valor (Key-Value Stores): Dynamo, Azure Table Storage, Couchbase Server, Riak, Redis, LevelDB, Chordless, GenieDB, Scalaris, Tokyo Cabinet/Tyrant, GT.M, Scalien, Berkeley DB, Voldemort, Dynomite, KAI, MemcacheDB, Faircom C-Tree, HamsterDB, STSdb, Tarantool/Box, Maxtable, Pin-

<sup>&</sup>lt;sup>1</sup>listagem obtida em www.nosql-database.org

caster, RaptorDB, TIBCO Active Spaces, allegro-C, nessDB, HyperDex, Mnesia,LightCloud, Hibari, BangDB.

Dentre esses vários produtos NoSQL, a seguir detalhamos os principais.

# 1.3.1. Apache Hadoop e IBM BigInsights

O projeto Apache Hadoop [3] é atualmente a referência do paradigma *MapReduce*. Este projeto envolve um conjunto de ferramentas voltadas para o processamento de dados de forma escalável, confiável e distribuída. O conjunto de projetos relacionados ao Hadoop são ilustrados na Figura 1.2.

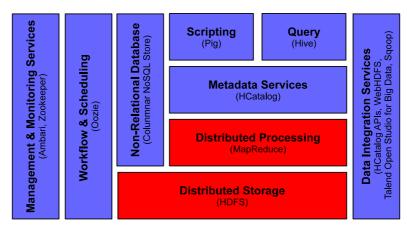


Figura 1.2: Conjunto de subprojetos relacionados ao projeto Apache Hadoop.

O Hadoop é constituído de duas camadas muito importantes: a camada de armazenamento, Hadoop Distributed File System (HDFS), e a de processamento dos dados, Hadoop MapReduce Framework. Essas duas camadas são minimas requiridas para o funcionamento do paradigma *MapReduce*.

O produto IBM InfoSphere BigInsights [15] é uma plataforma flexível destinada ao processamento de grandes volumes de dados. Esta plataforma tem como diferencial a integração do Apache Hadoop com configuração automatizada pelo instalador, além de fornecer um terminal de administração.

# 1.3.1.1. Apache Hadoop

O Apache Hadoop é um *framework MapReduce* que facilita o desenvolvimento de aplicações para o processamento de grande volume de dados de forma distribuída, paralela, e com tolerância a falhas. A ideia principal para o desenvolvimento de um processamento do tipo (*Job*) *MapReduce* envolve o particionamento dos dados em partes independentes. Estas partes são processadas por uma função de mapeamento de forma totalmente paralela em cada nó do *cluster* de computadores. Este *framework* ordena as saídas das funções de mapeamento e que servem de entrada para as funções de redução. Na maioria dos casos, as entradas e saídas desses processo são armazenadas no sistema de arquivos HDFS. O Apache Hadoop também pode organizar a ordem de execução dos processos, e monitoramento e re-execução em caso de falhas nas tarefas.

# 1.3.1.2. Hadoop Distributed File System (HDFS)

HDFS é um sistema de arquivos distribuídos voltado para o processamento de volume de dados na ordem de Terabytes (ou Petabytes). Este sistema também suporta controle de falhas e acesso paralelo, sendo que os arquivos são automaticamente armazenados de forma redundante e transparente em vários nós. Para que qualquer dado seja processado por um dos produtos Hadoop, é preciso que os dados esteja armazenados no sistema de arquivos visualizado pelo ambiente. Isto é, apesar do HDFS ser o mais usual, também é possível utilizar outros sistemas de arquivos, como o NFS. Os controles do HDFS fazem com que em cada nó os dados sejam armazenados em blocos contínuos, o que facilita muito o acesso aos dados de forma sequencial. Outra importante característica é o não cacheamento dos arquivos, dado que estes podem chegar a tamanhos extremamente grandes.

### **1.3.1.3.** Apache Hive

O sistema Apache Hive [4] incorpora ao framework Hadoop funcionalidades de *data warehouse*, i.e., o Hive facilita operações de sumarização e consultas *ad-hoc* voltadas para análise de grandes volumes de dados. O desenvolvedor pode definir consultas utilizando funções *Map/Reduce* ou utilizar a linguagem Hive QL. Esta linguaguem é baseada em SQL e encapsula o processamento *MapReduce*. A organização dos dados pelo Hive é feita pelas seguintes estruturas:

- *Databases*: usado como *namespace* para resolver conflitos de nomes;
- Table: dados dentro de um mesmo esquema;
- *Partition*: especifica a divisão dos dados de uma *Table* para o armazenamento;
- *Bucket*: forma de organização de um *Partition*, i.e., um grupo de *Bucket* forma uma partição.

# 1.3.2. Apache CouchDB e BigCouch

O produto Apache CouchDB, implementado em Erlang, é voltado para trabalhar com documentos no padrão JSON. Sua interface de comunicação se dá através do padrão HTTP, fornecendo acesso no formato de um *serviço Web* no padrão *Restful* e as consultas e transformações são feitas com a linguagem Javascript, mas ele pode ser estendido, funcionando por exemplo com outras linguagens de *script* como CoffeScript e Python. O controle de concorrência é realizada seguindo a politica MVCC (*Multi-Version Concurrency Control*) e conflitos são resolvidos em nível de aplicação.

Para permitir a escalabilidade horizontal do CouchDB, foi desenvolvido o produto BigCouch que facilita a construção de *cluster* elástico de instancias de CouchDB. Para este curso foram desenvolvidos exemplos utilizando o produto BigCouch.

### 1.3.3. SciDB

O produto SciDB é voltado para análise de dados em contextos comerciais ou científicos, e utiliza um modelo de dados de Vetor Multidimensional. Sua plataforma é voltada para analise de grande quantidade de dados, sendo escalável para execução em *cluster* comum ou em ambiente de nuvem. Apesar de não utilizar o paradigma *MapReduce* está sendo incluído neste Curso para ilustrar um modelo de dados diferenciado.

Seu modelo de *vetor* funciona de forma que um vetor (*array*) é composto de dimensões e atributos. Um vetor de dimensão n tem  $d_1, d_2, ..., d_n$  dimensões. O tamanho de uma dimensão é a quantidade de valores ordenados nessa dimensão. Por exemplo, um vetor bidimensional (i, j) pode possuir os valores i = (1, 2, ..., 10) e j = (1, 2, ..., 30), ou seja, i possui 10 atributos e j possui 30 atributos.

Para processamento das informações a arquitetura do SciDB envolve três componentes principais: o *nó coordenador*, o *nó de trabalho* e o *catálogo*. O *nó coordenador* organiza o processamento e interage diretamente com o cliente que fez a requisição. O *nó de trabalho* armazena os dados (de forma não compartilhada) e faz o processamento da consulta. E o nó de *catálogo* utiliza o SGBD Postgresql para armazenar as informações sobre o *cluster*, bem como da distribuição dos dados. Um dos parâmetros utilizados para distribuição dos dados entre os nós é o denominado *chunk*, que estabelece para cada dimensão a quantidade de registros distribuídos em cada nó.

O SciDB provê duas interfaces de programação, a AQL (*Array Query Language*) e a AFL (*Array Functional Language*). A linguagem AFL possui as mesmas funcionalidades da AFL porem com uma abordagem funcional, além de prover recursos adicionais para manipulação de metadados e arrays. Para exemplificar o uso do SciDB foi implementado um exemplo com dados ambientais, conforme apresentado no Tópico 1.4.2.

### 1.4. Estudo de Caso

Para este curso foram preparados dois tipos de exemplos e seu uso em produtos NoSQL diferentes. Dessa forma, o enfoque não eh ser aprofundar em questões de desempenho ou detalhes de administração de cada produto, mas sim demonstrar as funcionalidades e conceitos básicos no uso desses produtos.

### 1.4.1. Dados de Patentes

De uma forma geral no Brasil as informações contidas no sistema de propriedade industrial ainda não têm sido bem utilizadas pela indústria ou pela academia, quer seja para proteção dos conhecimentos, quer seja como fonte de informação tecnológica.

Trabalhos elaborados por Marmor [22] e pela Organização Mundial da Propriedade Intelectual (OMPI) [33] mostram que cerca de dois terços de todas as publicações técnicas são apresentadas somente através do sistema de patentes. Essa característica, por si só, as tornariam imprescindíveis, ou no mínimo desejáveis, no desenvolvimento de qualquer atividade criativa envolvendo as áreas técnicas. Além de fornecer informações técnicas inéditas, o conhecimento dessas informações propicia a eliminação de possíveis coincidências, representando economia de tempo e de recursos financeiros, além de ser fonte de inspiração para novos desenvolvimentos, contudo estudos demonstram que a utilização do sistema de patentes, ainda é muito pouco conhecida e percebida pelo setor de pesquisa do país [23].

Uma das dificuldades observadas é o custo na obtenção das informações contidas no sistema de patentes, que claramente tem sido um empecilho para sua utilização, principalmente na fase inicial da pesquisa onde o usuário de uma forma geral não tem muita intimidade com o sistema de informação. Muitos desses usuários, ainda que tenham participado dos cursos de busca ora disponíveis, não dominam as técnicas de buscas capazes de assegurar a obtenção das informações corretas que em tese poderiam conter as respostas aos seus problemas. O IC-UFMT desenvolveu em conjunto com a Coordenação de Estudos e Programas do Centro de Disseminação da Informação Tecnológica do Instituto Nacional de Propriedade Industrial (CEDIN-INPI) a ferramenta INPITec que pretende auxiliar na solução desse problema de uso de bases de patentes, por ser baseado na filosofia de software livre a relevância da ferramenta fundamentase na possibilidade da sociedade em geral ser beneficiada com um software gratuito que utiliza as informações contidas no sistema de Propriedade Industrial para analisar setores tecnológicos de interesse.

A partir dos dados importados principais bases disponíveis, o sistema gera um grande número de relatórios e gráficos, automatizando a seleção dos documentos para posterior análise, que constitui a etapa fundamental no processo de elaboração de estudos envolvendo dados de patente.

### 1.4.1.1. Descrição do Estudo de Caso

Os dados utilizados neste trabalho foram da base de patentes do USPTO, que são fornecidas gratuitamente para download pelo Google Patents, do período de 2005 a 2012. Esses arquivos tem por volta de 500MB e estão no formato XML, portanto foram utilizados 200GB de dados de patentes como objeto de estudo.

Esses arquivos XML possuem vários patentes juntas e cada patente possui informações importantes como, classificação internacional, título, resumo, números e datas de publicação e depósito e informações adicionais dos depositantes e inventores. Esses dados são suficientes para serem geradas informações que auxiliem no processo de prospecção tecnológica que são muito importantes para empresas.

### 1.4.1.2. Ambiente de Processamento

Para este minicurso, foi utilizada a estrutura do IC-UFMT. O *cluster* de Hadoop foi criado com 4 máquinas utilizando o produto IBM BigInsights Basic Edition (isto facilitou muito a criação e manutenção do *cluster*).

# 1.4.1.3. Configuração do Ambiente de Programação

Para reprodução dos exemplos de uso do Hadoop são necessários conhecimentos básicos da linguagem Java e seu uso na ferramenta Eclipse, bem como a criação de projetos utilizando Maven. Neste curso as seguintes versões foram utilizadas: a JDK 1.6, Maven 3.0.4, Hadoop 0.20 no ambiente de desenvolvimento Eclipse Juno.

Para auxiliar na manipulação dos arquivos XML foi utilizada a biblioteca XS-tream da Codehaus <sup>2</sup> que fornece uma maneira simples de fazer um mapeamento XML-Objeto. Também é recomendada a instalação de algum *plugin* no Eclipse para Maven, neste caso foi usado o M2Eclipse.

# 1.4.1.4. Implementando um Exemplo de WordCount

Para exemplificar o uso das funções *Map Reduce* diretamente no *framework*, um exemplo de processamento de contagem de palavras eh apresentado. Nesse contexto de patentes, foi utilizado somente o resumo da patente e fazer a contagem de palavras no mesmo. Os passos utilizados neste exemplo seguem: (1) a implementação de três classes: *Map*, *Reduce* e *GooglePatentWordCount*. Esta ultima realiza a preparação dos dados

<sup>&</sup>lt;sup>2</sup>http://xstream.codehaus.org

e faz a chamada das outras duas classes; (2) gerar um arquivo . jar, que empacota todo o código construído, e implantá-lo no ambiente Hadoop.

Devemos escrever duas classes que estendam respectivamente das seguintes classes: *Mapper* e *Reducer*. Essas classes são genéricas e devem ser parametrizadas com os tipos de entradas e saídas das funções de *Map* e *Reduce*, sendo que esses parâmetros devem implementar a interface *Writable* (o Hadoop já fornece alguns tipos básicos que seguem essa interface).

Figura 1.3: Classe *Map* para contagem de palavras contidas no resumo das patentes.

A classe *Map*, ilustrada na Figura 1.3, estendeu de *Mapper* com os parâmetros que indicam a entrada e a saída da função *Map*, que no caso a entrada vai ser uma entidade XML e vamos ter como saída uma chave com a palavra e o numero 1 para cada palavra que depois vai ser somada na função de *Reduce*. No contexto usado, foi necessário extrair as marcações *html* contidas no resumo da patente e separar todas as palavras no texto para serem contadas.

Figura 1.4: Classe *Reduce* para contagem de palavras contidas no resumo das patentes.

A classe *Reduce*, que estende de *Reducer*, parametrizaas saídas da função *Map* (i.e., uma palavra seguida de um número) e os tipos da saída do método *Reduce* que serão os mesmo tipos, mas vão representar a contagem total das palavras. A função é bem simples, como já foi visto, teremos uma lista de números que foram emitidos na função de map para cada palavra, então nosso trabalho aqui é apenas agregar esses valores, neste caso apenas fazer a soma dos valores.

Outra questão importante é que os dados originais precisam ser ajustados, pois o Hadoop por padrão particiona os arquivos a serem processados por linha do arquivo. Esse comportamento obviamente não funciona com arquivos XML pois precisamos que cada função *Map* receba uma entidade XML para que ela consiga extrair informações de cada patente. Para tal vamos utilizar um classe que estende *InputFormat*, ou seja, ela determina qual é o formato de entrada dos dados e como ele deve ser particionado. O projeto Apache Mahout já desenvolveu uma classe *XMLInputFormat* <sup>3</sup> que faz exatamente o que desejamos, sendo necessária apenas que seja informado quais são as *strings* de inicio e fim da entidade XML.

Para finalizar, vamos construir a parte final do *Job*, que determina todas as configurações iniciais, as delimitações do XML, as entradas e saídas do *Job*. Os caminhos das entradas e saídas vão ser passados como parâmetro na execução do *Job*. Segue o exemplo de tal configuração.

```
public static class GooglePatentWordCount{
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        conf.set("xmlinput.start", "<us-patent-application");
        conf.set("xmlinput.end", "</us-patent-application >");

        Job job = new Job(conf, "Google Patent Word Count");

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setJarByClass(GooglePatentWordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(XmlInputFormat.class);
        job.setInputFormatClass(TextOutputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputFormat.setInputPaths(job, new Path(args[1]));
        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        job.waitForCompletion(true); } }
```

Figura 1.5: Classe que prepara os dados e chamadas às funções *Map/Reduce*.

Para gerar o arquivo *jar* executável dentro do Hadoop, iremos utilizar o Maven para dar *build* no nosso projeto. Para isso temos que descrever como será esse *build*, para que ele construa o *jar* contendo as *libs* que nós utilizamos no projeto. Vamos utilizar o seguinte XML de configuração de *build*. Após essa fase é preciso adicionar no arquivo pom.xml o caminho para o arquivo, ilustrado na Figura 1.6 (job.xml), e as configurações de *build*. Segue o conteúdo que deve ser adicionado ao pom.xml.

Agora para gerar o jar é muito simples. Pelo terminal, na pasta do projeto, é preciso executar os comandos: mvnclean e mvnpackage, o que gera na pasta target do projeto um arquivo jar com o nomedoprojeto + verso + job. Agora podemos submeter

<sup>&</sup>lt;sup>3</sup>https://github.com/apache/mahout/blob/ad84344e4055b1e6adff5779339a33fa2 9e1265d/examples/src/main/java/org/apache/mahout/classifier/bayes/XmlInputFormat.java

```
1 <assembly
         sembly
xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly plugin/assembly/1.1.0
http://maven.apache.org/xsd/assembly-1.1.0.xsd">
        < id > job < /id >
        <formats>
             <format>jar </format>
         </formats>
10
        <\!includeBaseDirectory>\!false<\!/includeBaseDirectory>\!
11
        <dependencySets>
12
13
             <dependencySet>
<unpack>false </unpack>
                 <scope>runtime </scope>
<outputDirectory>lib </outputDirectory>
                 <scope>runtime </s
16
17
                <excludes >
  <exclude > ${ artifact . groupId }: ${ artifact . artifactId } </exclude >
                 </excludes
18
             </dependencySet>
            <dependencySet>
<unpack>false </unpack>
20
21
22
                 <scope > system </scope
                 <outputDirectory>lib </outputDirectory>
24
                <excludes:
25
                     <exclude>${ artifact.groupId }:${ artifact.artifactId }</exclude>
                 </excludes
26
             </dependencySet>
28
        </dependencySets>
29
30
            <fileSet>
                 <directory >\${ basedir }/ target / classes </ directory >
32
                 <outputDirectory >/ </outputDirectory >
33
34
                    <exclude > *. iar </exclude >
35
                 </excludes
             </fileSet>
36
37 </ file S e
38 </ assembly >
        </ file S et s >
```

Figura 1.6: Arquivo job.xml contendo a configuração do Maven para inserir as classes geradas no Hadoop.

```
<build>
                                            < plu g i n s >
                                                        <plugin>
                                                                  <artifactId >maven-assembly-plugin </artifactId >
                                                                  <version >2.2.1 </ version >
                                                                  <\!configuration>
                                                                            <descriptors>
                                                                             <descriptor > src / main / java / job . xml </descriptor >
</descriptors >
                                                                                       <manifest>
11
                                                                                                     <\!mainClass\!>\!br.ufmt.patent.wordcount.job.GooglePatentWordCount<\!/mainClass\!>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/mainClass>\!patentWordCount<\!/main
13
                                                                                          </manifest> </archive> </configuration>
                                                                  < e x e c u t i o n s >
15
                                                                              <execution>
16
                                                                                        <id>make-assembly </id>
17
                                                                                        <phase > package </phase >
<goals >
                                                                                                      <goal>single </goal>
20
                                                                                          </goals> </execution> </executions> </plugin> </plugins> </build>
```

Figura 1.7: Conteúdo do arquivo pom. xml que referencia job. xml.

o arquivo *jar* para que nosso *job* possa ser executado, para isso é preciso copiar o *jar* para a máquina que é o *Namenode* no *cluster* e guardar o destino do arquivo, pois o mesmo deve ser chamado para execução do processamento.

Para que o Hadoop possa executar o *job* é preciso utilizar a interface fornecida pela ferramenta da IBM, que em uma instalação padrão vai responder pela porta 8080 do *Namenode*. Nessa interface, na aba *Jobs*, é possível submeter o *job* clicando no botão "*CreateJarJob...*".

# 1.4.1.5. Utilizando o Hadoop como Ferramenta de ETL

Aqui tratamos o seguinte problema: Como executar consultas nos dados de patentes e como armazenar estes dados que suporte consultas eficientes. Para a resolução deste problema vamos utilizar o Apache Hive. O Hive consegue utilizar a estrutura do Hadoop para armazenar os dados e abstrai o *MapReduce* em forma de consulta SQL. No momento do processamento a consulta SQL é transformada em um *job MapReducer* que é executado no *cluster*.

Mas temos que adequar nossos dados a estrutura que ele fornece para trabalhar. O formato mais simples é um formato de arquivo em que cada registro se encontra e uma linha do arquivo e os campos estão delimitados por algum caractere especial. Isso nos lembra o formato de um arquivo CSV, então nosso trabalho aqui é transformar os dados XML em um arquivo com formatação CSV e importar os dados para dentro do Hive. Para tal tarefa vamos aproveitar a estrutura do Hadoop para trabalhar com uma grande quantidade de dados e de forma paralela.

Nossa função *Map* vai ter como objetivo converter as entidades XML em objetos Java e emitir para cada patente (usando como chave o atributo *datadepublicao*), criar todas as representações do objeto no formato CSV. Neste exemplo é criada uma tabela única que contem todas as combinações de classificações *vs.* depositantes, conforme o código ilustrado na Figura 1.8.

Figura 1.8: Preparação dos dados pela função *Map* a serem inseridos no Hive.

Explicando um pouco o código: Recebemos uma entidade em XML, converte-

mos para um objeto java contendo os atributos de interesse, usando a biblioteca *XStream*, e depois geramos todas as combinações com as classificações e depositantes dentro das patentes para a função *Reduce*. Agora o código da função *Reduce* é bem simples. Como não temos nenhuma agregação a fazer, apenas iteramos pelos valores gerados para cada chave e passamos para a saída do Job.

```
public class GooglePatentReducer extends Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterable <Text> values, Context context)

throws IOException, InterruptedException {

for (Text val : values) {

context.write(key, val);

} }
```

Figura 1.9: Preparação dos dados pela função *Reduce* a serem inseridos no Hive.

### 1.4.1.6. Utilizando o CouchDB para Processar Patentes

Para a construção do exemplo de uso do CouchDB com informações de patentes foram realizados os seguintes passos:

- 1. Transformação do formato XML para o formato JSON;
- 2. Inserção dos dados no CouchDB;
- 3. Construção de consulta que com o uso de agregações faz a contagem de patentes e as organiza na hierarquia definida pelo padrão da classificação internacional.

O CouchDB manipula as informações no formato JSON, assim é preciso fazer a transformação dos dados das patentes do formato XML para serem enviados para o armazenamento. Apesar dessa transformação ser relativamente simples o maior problema é trabalhar com arquivos XML grandes (500MB), que acabam por consumir muito memória durante seu processamento.

Para este exemplo foi construído um *script* em NodeJS que ler os documentos XML, converter cada entidade de patente para o formato JSON e após isso enviar para o CouchDB. Foi escolhida essa linguagem por possuir uma grande variedade de bibliotecas desenvolvidas pela comunidade e que vão suprir as necessidades para a transformação. As ferramentas utilizadas para este exemplo foram:

- 1. NodeJS para implementação;
- 2. Biblioteca Cradle para transformação<sup>4</sup>;

<sup>&</sup>lt;sup>4</sup>Endereço: https://github.com/cloudhead/cradle

- 3. Biblioteca Node-BufferedReader para carregamento dos arquivos<sup>5</sup>;
- 4. Biblioteca Node-xml2js<sup>6</sup>.

A Figura 1.10 apresenta o código em NodeJS que faz a transformação de XML para JSON e insere em bloco de 100 patentes no CouchDB. Como pode ser visto, o valor do identificador padrão utilizado pelo CouchDB foi substituído pelo campo *numerode publicacao* para garantir que não exista documentos duplicados dentro do banco, já que o CouchDB não tem restrições de unicidades para os campos além do *id*.

```
var xml2js = require('xml2js');
   var parser = new xml2js.Parser();
//usa 'patente' que foi lida pelo BufferedReader no codigo anterior
             parser.parseString(patente, function (err, result)
                         if (err){
                                    console.log('Erro no parse de XML -> JSON');
                                   console.log(err);
                        }else{
                   // Modificamos a id da patente
                                   var docID = result['us-bibliographic-data-application']['publication-reference']['document-id'];
var pub = docID['country'] + docID['doc-number'] + ' ' + docID['date'];
result['_id'] = pub;
patentes.push(result);
if( networks leader);
10
12
13
14
15
16
17
18
                                    if ( patentes.length >= 100 ) {
                                              var tempPatentes = patentes;
                                              patentes = []:
                                              db.save(tempPatentes, function (err, res) {
                                                         if(err){
                                                                   console.log('Erro ao salvar no CouchDB');
19
20
21
22
                                                                    console.log(err);
                                                         }else{ tempPatentes = []; } ); } });
              patente = '':
```

Figura 1.10: Conversão e inserção em lote no CouchDB das patentes no formato JSON.

A forma escolhida para gerar um exemplo de estatística foi a criação de uma visão no CouchDB que usa agregação para mostrar a contagem do numero de patentes dentro da hierarquia estabelecida pela classificação internacional. Uma visão é guardada internamente como um *designdocument* e é executada no momento em que é feita a consulta à visão. Para construção da visão foram criadas as funções *Map* e *Reduce*. Na função *Map*, ilustrada na Figura 1.11, são organizadas os campos da classificação, de forma que recebe apenas um parâmetro, que é o documento em que está sendo executado. Partindo desse documento os dados são extraídos e com a função *Emit*, são definidos quais dados vão ser passados para a função de *Reduce*. A função *Emit* recebe dois parâmetros, uma *chave* e um *valor*, podendo ser chamada mais de uma vez, como no exemplo estão sendo enviadas todas as classificações dentro de uma patente para a função de *Reduce* (se tiver mais de uma classificação). Outro ponto a ser observado é que as chaves e os valores passados para essas funções podem ser listas, no exemplo as classificações são separadas nos seus campos principais.

<sup>&</sup>lt;sup>5</sup>Endereço: https://github.com/Gagle/Node-BufferedReader

<sup>&</sup>lt;sup>6</sup>Endereço: https://github.com/Leonidas-from-XIV/node-xml2js

```
oc['us-bibliographic-data-application']['classifications-ipcr']['classification-ipcr'];
                         // Verifica se este objeto eh um Array
                        if(Object.prototype.toString.call(ipcrs) == '[object Array]'){
    for(var i in ipcrs){
                                                                             var section = ipcrs[i]['section'];
var clazz = ipcrs[i]['class'];
                                                                             var clazz = ipcrs[1]['class'];
var subclass = ipcrs[i]['subclass'];
var group = ipcrs[i]['main-group'];
var subgroup = ipcrs[i]['subgroup'];
var ipc = section + clazz + subclass + group + subgroup'];
var ipc = section + clazz + subclass + group + subgroup'];
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + group + subgroup';
var ipc = section + clazz + subclass + gro
 10
 11
                                                                                                                                                                                                                                                                  + group + subgroup;
 12
 13
                              // Senao trata como um objeto simples
                                                                          var section = ipcrs['section'];
16
17
                                                                          var clazz = ipcrs['class']
                                                                          var subclass = ipcrs['subclass'];
                                                                         var group = ipcrs['main_group'];
var subgroup = ipcrs['subgroup'];
var ipc = section + clazz + subclass + group + subgroup;
 18
 20
                                                    emit([section, clazz, subclass, group, subgroup, ipc],1); } }
```

Figura 1.11: Função *Map* para a visão por classificação das patentes.

A função *Reduce*, ilustrada na Figura 1.12, aplica uma função de agregação nas chaves e valores enviados pela função *Map*, que nesse caso é apenas a soma dos valores para cada chave, obtendo assim a contagem de classificações.

```
1 function(keys, values, rereduce){ return sum(values); }
```

Figura 1.12: Função *Reduce* para a visão por classificação das patentes.

Como pode ser observado, a função *Reduce* recebe 3 parâmetros: *keys*, *values* e *rereduce*. A chave recebida pela função pode vir no formato de listas, já que é possível fazer isso na função *Map*. A variável *values* corresponde a todos os valores associados com uma determinada chave. Já a variável *rereduce* é um booleano que diz se essa função *Reduce* já foi executa em uma mesma chave. Isto deve ser levado em consideração em determinados casos nos quais a função de agregação não deve ser aplicada da mesma forma se a função *Reduce* passar duas vezes pela mesma chave.

# 1.4.2. Dados Micrometeorológicos

Os dados obtidos através de simulações científicas, experimentos e observações em larga escala, facilmente chegam a terabytes, devido ao aumento do número de transistores dentro dos chip's, conhecido como a lei de Moore e o barateamento dos mesmos. Entre esses equipamentos encontra-se aqueles que são voltados para medir dados micrometeorológicos, sendo esses coletados através de vários sensores efetuando medições de múltiplas variáveis.

Assim, a quantidade de dados é muito significativa, pois vários sensores podem medir várias variáveis micrometeorológicas em um tempo curto, por exemplo, medir a temperatura entre 10 e 10 segundos, assim a quantidade de dados cresce 3153600

por ano e por variável medida. Após a obtenção dos dados alguns cálculos realizados sobre os mesmos pode demorar muito tempo para seu processamento, no contexto do Programa de Pós-Graduação em Física Ambiental (PGPFA-UFMT), existem vários cálculos que demandam esse tempo, por exemplo, o cálculo para obter dimensão fractal, o expoente de Lyapunov, a análise de recorrência, entre outros. Para otimizar esse tempo pode-se utilizar algumas soluções com enfoque em paralelismo, como *clusters* de computadores, ou GPU (*Graphics Processing Unit*).

Apesar deste Curso não abordar esses cálculos complexos, a ideia é demonstrar o uso do produto SciDB para armazenamento e processamento de variáveis micrometeorológicas. Dessa forma, o exemplo aqui apresentado envolve variáveis (*evaporação*, *insolação*, *precipitação*, *temperatura máxima*, *temperatura minima*, *umidade relativa do ar* e *velocidade do vento*) captadas em 4 locais diferentes no Estado de MT entre os anos de 1985 a 2010. Os seguintes passos foram necessários para construir o exemplo: (1): conversão dos dados do formato CSV para o formato SciDB; (2): inserção dos dados na base SciDB em forma de vetor unidimensional; (3): conversão do vetor unidimensional para vetores multidimensionais, ou seja, conversão de atributos para dimensões. Para inserção de dados o SciDB fornece o aplicativo *csv2scidb* que converte do formato CSV (separado por ",") para o formato de leitura do SciDB, assim, depois de convertidos usa-se a função *Load* (em AFL) para inserção dos dados na base.

# 1.5. Considerações Finais

Este minicurso apresentou os conceitos, linguagens e ferramentas das principais tecnologias NoSQL. Também foi apresentado a aplicação de produtos NoSQL em dois estudos de casos, como os produtos Hadoop, Hive, BigCouch, e SciDB. Podemos observar que, pela grande quantidade de produtos NoSQL disponíveis no mercado, o desenvolvedor possui uma grande variedade de produtos para a escolha. Esta vantagem pode as vezes ser um problema, principalmente se o desenvolvedor não tiver um conhecimento profundo dos diversos produtos. Outro importante impacto para a atividade do desenvolvedor é a variedade de linguagens envolvidas, onde cada produto possui sua linguagem específica, seu formato de entrada, seus comandos, entre outras características. Com isso, vemos que manter um ambiente com vários produtos é a uma tendência frequentemente praticada nos sistemas atuais.

# 1.6. Descrição dos Palestrantes

Marcos Rodrigues Vieira: possui graduação em Engenharia de Computação pela Univ. Federal de São Carlos (UFSCar) (2001), mestrado em Ciências da Computação e Matemática Computacional pela Univ. de São Paulo (USP), São Carlos (2004), e Doutorado em Ciências da Computação pela University of California, Riverside (UCR), EUA (2011). Atualmente é pesquisador da IBM Research/Brazil, e colaborador do Computer

Science Department da UCR e do Instituto de Ciências Matemáticas e de Computação (USP, São Carlos).

Josiel Maimone de Figueiredo: possui graduação em Engenharia de Computação pela UFSCar (1998), mestrado em Ciência da Computação pela UFSCar (2000) e doutorado em Ciências da Computação e Matemática Computacional, com ênfase em Banco de Dados, pela USP, São Carlos (2005). Atualmente é professor adjunto do Instituto de Computação da Univ. Federal de Mato Grosso (UFMT), sendo credenciado no Programa de Pós-Graduação em Física Ambiental do Instituto de Física da UFMT.

**Gustavo Liberatti**: aluno de graduação do curso de Bacharelado em Ciência da Computação pela UFMT desde 2008, onde atua principalmente nos seguintes temas: processamento de imagens, geo-referenciamento, serviços Web e Linux. É bolsista de Iniciação Científica em projeto de tratamento textual de dados.

**Alvaro Fellipe Mendes Viebrantz**: aluno de graduação do curso de Bacharelado em Ciência da Computação pela UFMT desde 2009. Atualmente é bolsista de Iniciação em Desenvolvimento Tecnológico e Inovação com atuação em sistemas de manipulação de informações de patentes.

### Referências

- [1] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: New wine or just new bottles? *Proc. VLDB Endow.*, 3(2):1647–1648, 2010.
- [2] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: current state and future opportunities. In *EDBT*, page 530, 2011.
- [3] Apache Hadoop. http://hadoop.apache.org, 2012.
- [4] Apache Hive. hive.apache.org, 2012.
- [5] Apache Pig. http://pig.apache.org, 2012.
- [6] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. In *Workshop on Middleware for Service Oriented Computing*, 2011.
- [7] E. A. Brewer. Towards robust distributed systems (abstract). In *Symp. on Principles of Distributed Computing*, PODC, page 7, New York, NY, USA, 2000.
- [8] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt. SciHadoop: Array-Based Query Processing in Hadoop. In *SC*, page 66, 2011.
- [9] R. Cattell. Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4):12, May 2011.
- [10] S. Chaudhuri. What next? A Half-Dozen Data Management Research Goals for Big Data and the Cloud. In *PODS*, page 1, 2012.
- [11] I. Clarke. A distributed decentralised information storage and retrieval system. In *PhD Thesis*. University of Edinburgh, 1999.
- [12] A. Cuzzocrea, I.-y. Song, and K. C. Davis. Analytics over large-scale multidimensional data: the big data revolution! In *Int'l Workshop on Data Warehousing and OLAP (DOLAP)*, page 101, 2011.

- [13] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI)*, 2004.
- [14] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with Big Data Analytics. *Interactions*, 19(3):50, May 2012.
- [15] IBM InfoSphere BigInsights. www.ibm.com/software/data/infosphere/biginsights, 2012.
- [16] M.-y. Iu and W. Zwaenepoel. HadoopToSQL: a mapReduce query optimizer. In EuroSys, 2010.
- [17] S. Khatchadourian, M. Consens, and J. Simeon. Web data processing on the cloud. *Conf. of the Center for Advanced Studies on Collaborative Research (CASCON)*, page 356, 2010.
- [18] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas. TIRA-MOLA: Elastic NoSQL Provisioning Through a Cloud Management Platform. In *SIGMOD*, 2012.
- [19] V. Kumar, H. Andrade, B. Gedik, and K.-l. Wu. DEDUCE: at the intersection of MapReduce and stream processing. In *EDBT*, page 657, 2010.
- [20] K.-H. Lee, Y.-j. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with MapReduce: a survey. *ACM SIGMOD Record*, 40(4):11, Jan. 2012.
- [21] D. B. Lomet. Data management on cloud computing platforms. In S. P. Beng Chin Ooi, editor, *IEEE Data Engineering Bulletin*, volume 32, pages 1–82. 2009.
- [22] A. Marmor, W. Lawson, and J. Terapane. The Technology Assessment and Forecast Program of the United States Patent and Trademark Office. *World Patent Information*, 1(1):15–23, 1979.
- [23] J. S. Nunes and L. Goulart. Universidades Brasileiras Utilização do Sistema de Patentes de 2000 a 2004. Technical report, INPI/CEDIN/DIESPRO, 2007.
- [24] A. Okcan and M. Riedewald. Processing theta-joins using MapReduce. SIGMOD, page 949, 2011.
- [25] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, page 1099, 2008.
- [26] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. *SIGMOD*, page 165, 2009.
- [27] J. C. Ralph Kimball. DW ETL toolkitThe Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley, 2004.
- [28] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas. Nobody ever got fired for using Hadoop on a cluster. *Int'l Workshop on Hot Topics in Cloud Data Processing*, 2012.
- [29] M. Stonebraker. In Search of Database Consistency. Commun. ACM, 53(10):8, Oct. 2010.
- [30] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). *VLDB*, pages 1150–1160, 2007.
- [31] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, R. Murthy, S. Anthony, P. Wyckoff, F. Data, and I. Team. Hive a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.
- [32] W. Vogels. Eventually Consistent. Queue, 6(6):14, Oct. 2008.
- [33] WIPO. Using Patent Information for the Benefit of Your Small and Medium-sized Enterprise, 2012.