Parte I

Capítulo

4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Big Data

**4.1 Introdução**

Big Data é um termo que vem chamando a atenção pela acelerada escalada em que volumes cada vez maiores de dados são criados pela sociedade. Fala-se comumente em petabytes de dados gerados a cada dia, e zetabytes começa a ser uma escala real e não mais futurista. A uma década atrás terabyres era uma quantidade futurista, agora temos em nosso próprios computadores. Muito tem sido escrito sobre Big Data e como ele pode servir como base para a inovação, diferenciação e crescimento da análise de dados em grandes massa de dados (Kolb, 2013).

De acordo com Raj (2013), as tecnologias que sustentam o Big Data, podem ser analisadas sob duas óticas: as envolvidas com análise de dados, tendo Hadoop e Map-Reduce como as principais e as tecnologias de infraestrutura, que armazenam e processam os dados. Neste aspecto, destacam-se os banco de dados NoSQL (Not Only SQL).

O termo Big Date está diretamente ligado a questões como volume, variedade, velocidade, complexidade e valor (Mayer-Schönberger, 2013).

* **Volume** – o volume está claro. Geramos petabytes de dados a cada dia. E estima-se que este volume dobre a cada 18 meses.
* **Variedade** - Variedade também, pois estes dados vêm de sistemas estruturados (hoje minoria) e não estruturados (a imensa maioria), gerados por e-mails, mídias sociais (Facebook, Twitter, YouTube e outros), documentos eletrônicos, apresentações estilo Powerpoint, mensagens instantâneas, sensores, etiquetas RFID, câmeras de vídeo, etc.
* **Velocidade** - De acordo com o Gartner, velocidade significa tanto o quão rápido os dados estão sendo produzidos quanto o quão rápido os dados devem ser tratados para atender a demanda. Etiquetas RFID e contadores inteligentes estão impulsionando uma necessidade crescente de lidar com torrentes de dados em tempo quase real. Reagir rápido o suficiente para lidar com a velocidade é um desafio para a maioria das organizações.
* **Valor** - E valor porque é absolutamente necessário que a organização que implementa projetos de Big Data obtenha retorno destes investimentos. Um exemplo poderia ser a área de seguros, onde a análise de fraudes poderia ser imensamente melhorada, minimizando-se os riscos, utilizando-se, por exemplo, de análise de dados que estão fora das bases estruturadas das seguradoras, como os dados que estão circulando diariamente nas mídias sociais.
* **Complexidade** - Quando você lida com grandes volumes de dados, eles vêm de diversas fontes. É um grande desafio vincular, correlacionar, limpar e transformar os dados de um sistema. No entanto, é necessário conectar e correlacionar interações, hierarquias e vínculos múltiplos de informação ou então os dados podem rapidamente sair de controle. Governança de dados pode ajudar a determinar como os dados díspares se relacionam com definições comuns e como integrar sistematicamente os ativos de dados estruturados e não estruturados para produzir informações de alta qualidade, uteis, adequadas e atualizadas.

Em última análise, independentemente dos fatores envolvidos, acreditamos que o termo Big Data é relativo e se aplica (por avaliação do Gartner) sempre que a capacidade da organização de gerenciar, armazenar e analisar os dados exceder sua capacidade atual.

**4.2 O Uso do Big Data**

Os modelos relacionais, quando proposto por Edgar F. Codd, atenderam muito bem, a demanda era acessar dados estruturados, de acordo com (Elmasri & Navathe (2005), gerados pelos sistemas internos das corporações. Estes modelos não foram desenhados para tratar dados não estruturados e nem para volumes de dados na casa dos petabytes de dados.

Para tratar dados na escala de volume, variedade e velocidade do Big Data precisamos de outros modelos. Surgem os softwares de banco de dados NoSQL, desenhados para tratar imensos volumes de dados estruturados e não estruturados. Existem diversos modelos como sistemas colunares como o [*Big Table*](http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/pt-BR/archive/bigtable-osdi06.pdf)(Usado internamente pelo Google),o modelo Key/value como [*DynamoDB da Amazon*](http://aws.amazon.com/pt/dynamodb/), o modelo “*document database*” baseado no conceito proposto pelo Lotus Notes da IBM e aplicado em softwares como MongoDB, e o modelo baseado em grafos como o [Neo4j](http://neo4j.org/), etc (Kolb, 2013).

Aplicações modernas de mineração de dados, frequentemente chamada "Big-Data Analytics", exigir-nos gerenciar grande quantidade de dados rapidamente e em muitas dessas aplicações, exige-se um amplo paralelismo (Kolb, 2013).

Para lidar com aplicações tais como essas, novos tipos de software tem surgido. Estes sistema de programação são projetados para obter o máximo do paralelismo. O novo tipo de software começa com uma nova forma de sistema de arquivos, chamada "Sistema de arquivos distribuídos", que contam com unidades muito maiores do que os blocos de disco dos sistemas operacionais convencionais. Além do mais, os sistemas distribuídos também fornecem replicação de dados ou redundância para proteger os dados, contra falhas frequentes de mídias, que ocorrem quando o dado é distribuído para milhões de nós de computadores (Kolb, 2013).

No topo destes sistemas de arquivos, diversos sistema de alto nível de programação foram desenvolvidos. No centro do novo software está o sistema de programação chamada ***Map-Reduce***. Implementações de ***Map-Reduce*** permite que os cálculos sob os dados em grande escala, sejam executados em clusters de computação de forma eficiente e tolerante a falhas de hardware (kolb, 2013).

**4.3 Map-Reduce**

Map-reduce não é um produto ou um software especifico, mas sim uma tecnologia desenvolvida pelo Google para lidar com grande quantidade de dados, cortando-os e combinando-os no final (Kolb, 2013).

A ideia básica é que os dados que precisam ser processados, entram no sistema, e é cortado em pedaços chamados de chunks. Essas peças de software que é responsável em fazer esses cortes é chamado de “***Mapper***”. Os chunks são então enviados para outra peças de software para fazer o processamento requerido sobre eles, e então eles são ainda enviados para outra peça de software chamado “***Reducers***” que combina o resultado final para a saída. isto tudo está ilustrado na Figura 4.1 (Kolb, 2013).

**Figura 4.1 Um esquema da função Map-Reduce**. **Fonte**: The Big Data Revolution, Kolb, 2013.

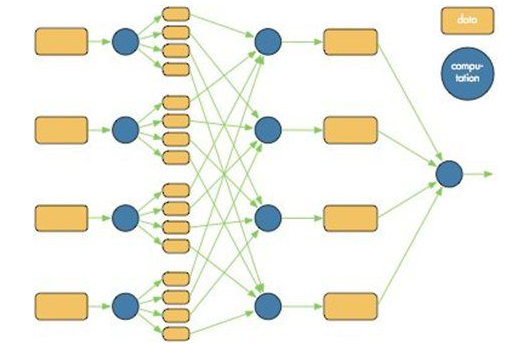
CombinedOutput

Input (chuncks)

Reduce (tasks)

Map (tasks)

Group by keys



Input (chuncks)

Todas essas peças de software - mapeador, processador, e redutor – tipicamente rodam no mesmo servidor de uma só vez. Dessa forma, a carga de processamento dos dados mapeados podem ser espalhados por todo servir disponível, e mais servidores podem ser adicionado em tempo real, se for necessário um resultado mais rápido (Kolb, 2013).

O importante aqui, é você entender que, a tecnologia de Map-Reduce é capaz de pegar uma grande quantidade de dados, que seria muito dispendioso rodar em apenas um servidor, e poder distribui-lo por vários servidores. Este é um novo paradigma de programação. Existem algumas ferramentas que implementam esse novo paradigma, entre elas cito, o **Hadoop** do *Apache Foundation* (Kolb, 2013).

**4.4 Sistemas Distribuídos**

A maioria dos computadores são feitos com um único processador, com sua memória principal, cache, e disco local (um nó de computação). No passado, grandes cálculos científicos, eram realizadas em aplicações chamada de processamento paralelo, feitos com propósito especial em computadores de uso paralelo com muitos processadores e hardware especializado. No entanto, a prevalência de serviços Web de grande escala tem levado a computação a ser realizada em instalações com milhares de nós de computação operando mais ou menos de forma independente. Nestas instalações, os nós de computação são hardware de commodity, que reduz os custos comparado com as máquinas paralelas (Stonebraker et Al., 2010).

Esta nova arquitetura de computação tem dado origem a uma nova geração de sistemas de programação. Estes sistemas tiram máximo proveito do paralelismo e são tolerantes a falhas, quer seja de software ou de hardware (Stonebraker et al., 201o).

**4.4.1 Organização Física de Nós de Computação**

A nova arquitetura de computação paralela, algumas vezes chamada de computação em cluster, é organizada como segue. Os nós de computação são armazenados em racks, talvez de 8-64 nós em um rack. Os nós em um simples rack são conectados por uma rede, tipicamente uma Ethernet gigabits. Pode haver muitos racks de nós de computador, e racks são interconectados outros níveis de rede ou a um switch (Dean & Ghemawat, 2008).

**4.4.2 Organização de Sistemas de Arquivos em Grande Escala**

Para explorar computação em cluster, o arquivo deve parecer e se comportar um pouco diferente de sistemas de arquivos convencionais encontrados em computadores simples. Este novo sistema de arquivos, geralmente chamado de *distributed file system* ou **DFS** (embora este termos tenha tido outro significado no passado), é tipicamente usado como segue (Dean & Ghemawat, 2008):

* Os arquivos são divididos em chunks, que são tipicamente de 64 megabytes de tamanho. Esses chunks são replicados, talvez três vezes, para três diferentes nós de computação. Além disso, os nós que armazenam um chunk deve estar localizado em racks diferentes, dessa forma não se perde todas as cópias devido uma falha de rack. Normalmente, o tamanho do chunk e grau de replicação podem ser decidido pelo usuário.
* Para encontrar os chunks de um arquivo, existe um outro pequeno arquivo chamado nó **master** ou nó de nome para o arquivo. O nó master é ele mesmo replicado, e o diretório do sistema de arquivos como um todo sabe onde encontrar suas cópias. O diretório pode ser replicado e todos os participantes usando o DFS sabem onde estão suas cópias no diretório.

**4.5 As Tarefas de Mapeamento**

De acordo com Kolb (2013), o arquivo de entrada para uma tarefa Mapeamento, consiste de elementos, que podem ser de qualquer tipo: uma tupla ou um documento, por exemplo. Um chunk é uma coleção de elementos, e nenhum elemento é armazenado em dois chunks. Tecnicamente, todas as entradas para as tarefas de mapeamento (The Map Tasks) e saídas para as tarefas Redução (The Reduce Tasks) são os pares na forma chave-valor (key-value), geradas por uma função hash. Essa forma de entradas e saídas são motivadas pelo desejo de permitir a composição de vários processos Map-Reduce (Kolb, 2013).

A função de mapeamento (Map) recebe um elemento com seus argumentos e produz zero ou mais pares chave-valor. Os tipos de chaves e valores são arbitrários. Mais, as chaves não são "chaves" no sentido usual; elas não precisam ser únicas. Mais uma tarefa de mapeamento pode produzir vários pares chave-valor com a mesma chave, mesmo a partir do mesmo elemento (Kolb, 2013).

**Exemplo 4.1**: Suponha que deseja-se contar o número de ocorrências para cada palavra em uma coleção de documentos. Neste exemplo, o arquivo de entrada é um repositório de documentos, e cada documento é um elemento. A função de mapeamento para este exemplo usa chaves que são do tipo String (a palavra) e valores que são inteiros. A tarefa de mapeamento lê um documento e quebra ele em uma sequência de palavras w1, w2, w3, ..., wn. Ela então emite um sequência de pares de chave-valor onde o valor é sempre 1. Isto é, a saída da tarefa de mapeamento para este documento é a sequência de pares chave-valor:

(w1,1),(w2,1),....,(wn,1)

Note que uma simples tarefa de mapeamento irá processar muitos documentos - todos os documentos em um ou mais chunks. Assim, a saída produzida será mais do que a sequência para o documento sugerida acima. Note também que se uma palavra w aparece m vezes entre todos os documentos atribuídos a esse processo, então haverá m pares chave-valor (w,1) entre sua saída. Uma opção para resolver esse problema é usar agrupamento e agregação, que é combinar esses m pares em um simples par (w, m), isso só é possível porque as tarefas Redução, aplica uma operação associativa e comutativa, para os valores.

**4.5.1 Agrupamento e Agregação**

O processo controlador mestre sabe quantas tarefas Reduce haverá, digamos r tarefas, pois o usuário normalmente informa ao sistema map-reduce quais são as r tarefas. Então o controlador mestre aplica uma função hash e produz uma tabela de chaves de números (códigos) de 0 até r-1. Cada chave produzida pela tarefa Map é um hash e seus pares chave-valor são colocados em um arquivo local. Cada arquivo é destinado para um das tarefas Reduce (Kolb, 2013).

Após todas as tarefas Map terem completadas com sucesso, o controlador mestre junta os arquivos de cada tarefa Map que são destinados para uma particular tarefa e alimenta o arquivo resultante com uma lista de pares chave-valor. Isto é, para chave k, a entrada para a tarefa Reduce que manipula a chave k é um par da forma (k, [v1, v2, ..., vn]), onde (k, v1), (k, v2), ..., (k, vn) são todos pares chave-valor e k, vindo de todas as tarefas Map.

**4.6 As Tarefas de Redução**

Os argumentos da função Reduce é um par consistindo de uma chave e sua lista de valores associados. A saída da função Reduce é uma sequência de zero ou mais pares chave-valor. Esses pares chave-valor pode ser de tipo diferente daqueles enviados das tarefas map para as tarefas Reduce, mais normalmente elas são do mesmo tipo. Referimo-nos a aplicação da função Reduce que reduz para uma simples chave e seus valores associados de redutor (Kolb, 2013).

Uma tarefa reduce recebe uma ou mais chaves e sua lista de valores associados. Isto é, uma tarefa reduce executa um ou mais redutores. As saídas de todas as tarefas reduce são juntas em um simples arquivo. Redutores podem ser divididos em tarefas reduce menores e a função hash associa cada chave com um dos códigos da tabela hash (Kolb, 2013).

**Exemplo 4.2**: Vamos continuar com o exemplo conta palavras do Exemplo 4.1. A função Reduce simplesmente agrega todos os valores. A saída de um redutor consiste da palavra e da soma. Isto é, a saída de todos as tarefas Reduce é uma sequência de pares (w,m), onde *w* é uma palavra que aparece pelo menos uma vez entre todos os documentos e *m* é o total de ocorrências de *w* em todos os documentos.

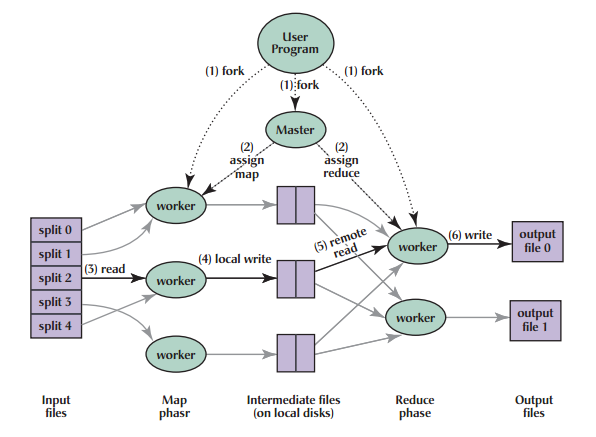
**4.7 Detalhes de Execução de Map-Reduce**

A Figura 4.2 oferece um esboço de como processo, tarefas, e arquivos interagem. Aproveitando uma biblioteca fornecida por um sistema map-reduce tal como Hadoop, o programa do usuário bifurca o processo controlador mestre e alguns dos processos Worker para diferentes nós de computação. Normalmente, um Worker manipula suas tarefas Map (um Map worker) ou tarefas Reduce (um Reduce worker), mas não ambos (Kolb, 2013).

O mestre tem muitas responsabilidades. Uma é criar um certo número de tarefas Map e algumas tarefas Reduce, este número sendo selecionado pelo programa do usuário. Estas tarefas serão atribuídas para o processo Worker pelo Mestre. É razoável criar uma tarefa Map para cada chunk de arquivo de entrada, mas pode-se desejar criar poucas tarefas Reduce. A razão para limitar o número de tarefas Reduce é que é necessário para cada tarefa Map criar um arquivo intermediário para cada tarefa Reduce, e se existe muitas tarefas Reduce o número de arquivos intermediários aumenta bastante (Kolb, 2013).

O Mestre (Master) se matem informado do estado de cada tarefa Map e Reduce (ocioso, executando um particular worker, ou concluído). Um processo Worker relata para o Mestre quando ele termina uma tarefa, e uma nova tarefa é agendada pelo Mestre para esse processo Worker (Kolb, 2013).

**Figura 4.2** **Esboço de Iteração de um Processo Map-Reduce. Fonte:** Dean & Ghemawat (2008).



**4.8 Lindando com Falha de Nós**

A pior coisa que pode acontecer é quando o nó de computação que está executando o Mestre falha. Neste caso, todo carga de entrada do map-reduce deve ser reiniciado. Mas somente este nó pode derrubar um processo inteiro; outras falhas iram ser gerenciadas pelo Mestre, e o trabalho do map-reduce irá completar eventualmente (Kolb, 2013).

Suponha que o nó de computação no qual reside o Map worker falha. Esta falha irá ser detectada pelo Mestre, porque ele periodicamente pings o processo Worker. Todas as tarefas Map que foram atribuídas para este Worker terão que ser refeitas, mesmo que tivessem concluído. A razão para refazer completamente as tarefas Map é que sua saída destinada as tarefas Reduce residem no nó de computação, e devido falha, está indisponível para as tarefas Reduce. O Mestre configura o estado de cada uma das tarefas Map para ociosa e reprograma-o para um Worker quando se tornar disponível. O Mestre também informa cada tarefa Reduce que a localização de suas entrada, que são as tarefas Map, mudaram (Kolb, 2013).

Lidar com uma falha para um Reduce Worker é simples. O Mestre, simplesmente, configura os estados de suas correntes tarefas Reduce em execução, para ociosa. Estas serão agendadas para outro reduce worker mais tarde (Kolb, 2013).

4.9 **Conclusões**

Concluindo, Map-Reduce é um modelo de programação, e framework introduzido pelo Google para suportar computações paralelas em grandes coleções de dados em clusters de computadores. Agora Map-Reduce é considerado um novo modelo computacional distribuído, inspirado pelas funções map e reduce usadas comumente em programação funcional. Map-Reduce é um “Data-Oriented” que processa dados em duas fases primárias: Map e Reduce. A filosofia por trás do Map-Reduce é: Diferentemente de data-stores centrais, como um banco de dados, você não pode assumir que todos os dados residem em um lugar central portanto você não pode executar uma query e esperar obter os resultados em uma operação síncrona. Em vez disso, você precisa executar a query em cada fonte de dados simultaneamente. O processo de mapear a requisição do originador para o data source é chamado de ‘Map’, e o processo de agregação do resultado em um resultado consolidado é chamado de ‘Reduce’.

Hoje existem diversas implementações de Map-Reduce, como: Hadoop, Disco, Skynet, FileMap e Greenplum. Hadoop é a implementação mais famosa.

A tecnologia de big data não apenas suporta a habilidade de coletar grandes volumes de dados como também provê a habilidade de compreendê-los e tirar proveito de seu valor.