Conceitos teóricos e práticos, evolução e novas possibilidades

Prof. Dr. Alfredo Goldman

Prof. MS. Ivanilton Polato

Autores

Dr. Alfredo Goldman

Professor DCC IME/USP

Dr. Fabio Kon

Professor DCC IME/USP

Ms. Francisco Pereira Junior

Professor UTFPR-CP

Doutorando DCC IME/USP

Ms. Ivanilton Polato

Professor UTFPR-CM

Doutorando DCC IME/USP

Esp. Rosangela de Fátima Pereira

Professora UTFPR-CP

Roteiro

- Motivação
- Origens do Hadoop e Apache Hadoop
- Vantagens e Desvantagens
- O universo Hadoop
- Sistema de Arquivos HDFS
- O paradigma MapReduce
- Exemplos e práticas

Motivação

- Uso potencial de aplicações BigData
  - Conjuntos de dados na ordem de petabytes
  - Computação intensiva sobre os dados
- Computação paralela não é trivial
  - Divisão das subtarefas
  - Escalonamento das subtarefas
  - Balanceamento de carga

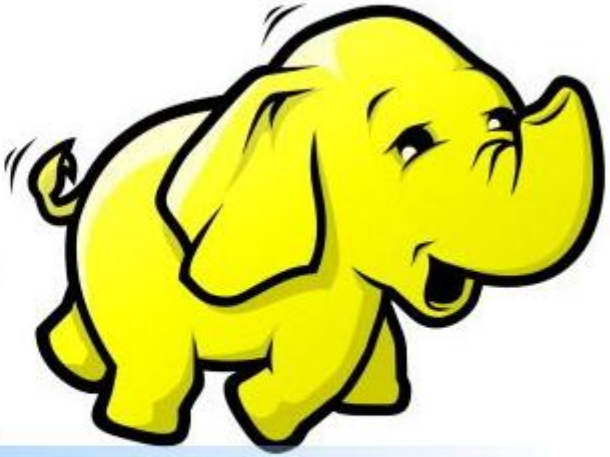
Motivação

- Apache Hadoop
  - Retira a complexidade na computação de alto desempenho
- Custo eficiente
  - Máquinas comuns
  - Rede comum
  - Tolerância a falhas automática
    - Poucos administradores
  - Facilidade de Uso
    - Poucos programadores

Hadoop

- Arcabouço para processamento e armazenamento de dados em larga escala:
  - Código aberto
  - Implementado em Java
  - Inspirado no GFS e MapReduce da Google
  - Projeto principal da Fundação Apache
  - Tecnologia recente, porém já muito utilizada

Histórico



* http://nutch.apache.org/

** http://labs.google.com/papers/mapreduce.html
http://labs.google.com/papers/gfs.html

Como originou?

- 2003: Google publica artigo do GFS
- 2004: Google publica artigo do MapReduce
- 2005: Doug Cutting cria uma versão do MapReduce para o projeto Nutch
- 2006: Hadoop se torna um projeto independente da Fundação Apache

Como originou?

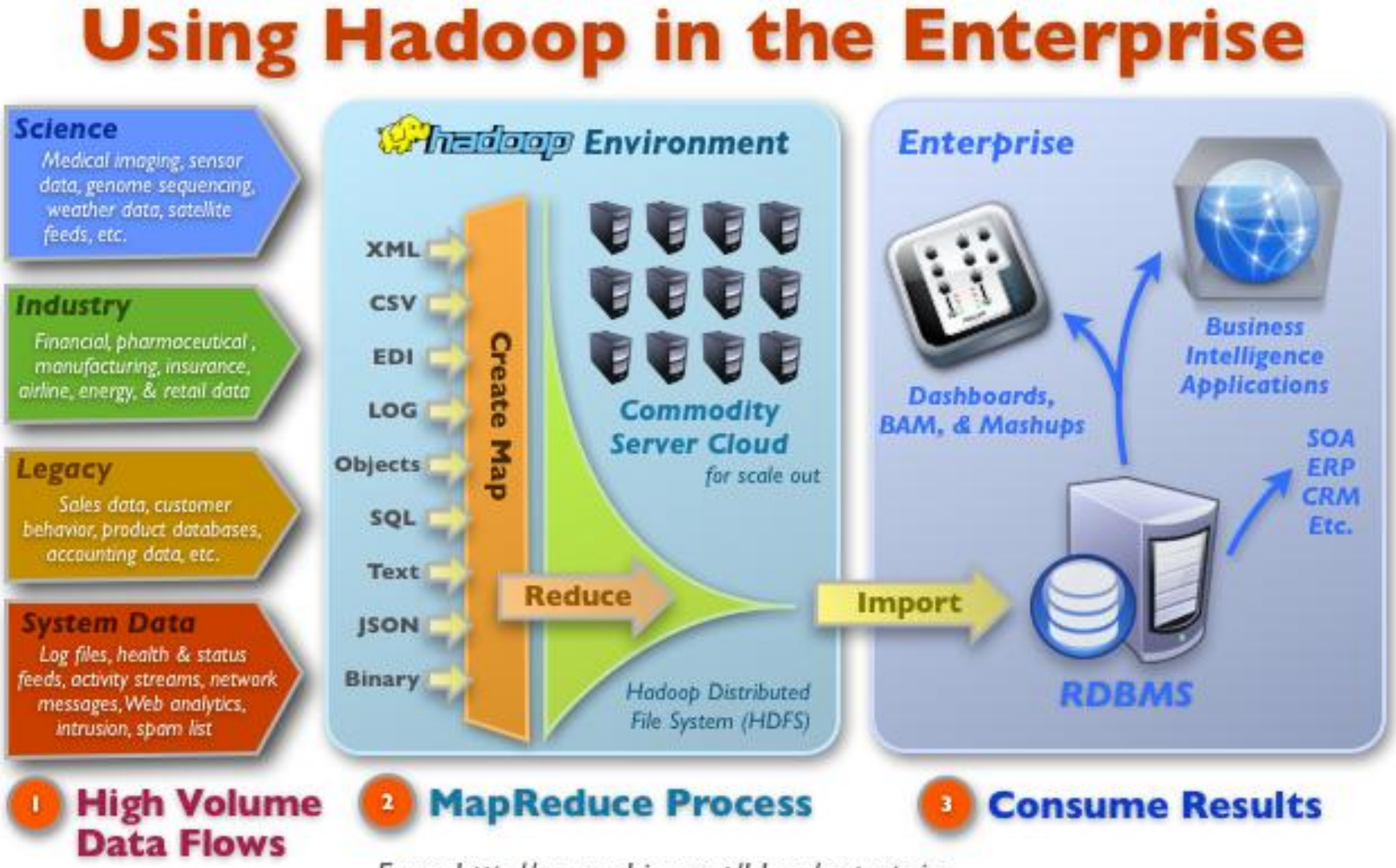
- 2007: Yahoo se torna a maior contribuinte do projeto
- 2008: Hadoop se transforma em um projeto principal da Apache
- 2011: Apache disponibiliza versão 1.0.0

Quem utiliza?

Onde utilizar?

- DataWarehouse
- Business Intelligence
- Aplicações analíticas
- Mídias sociais

Muitas possibilidades...



From http://www.ebizq.net/blogs/enterprise

Vantagens

- Por que usar Hadoop?
  - Código aberto
  - Econômico
  - Robusto
  - Escalável
  - Foco na regra de negócio

Vantagens (1)

- Código Aberto
  - Comunidade ativa
  - Apoio de grandes corporações
  - Maiores correções de erros
  - Constante evolução do arcabouço

Vantagens (2)

- Econômico
  - Software livre
  - Uso de máquinas e redes convencionais
  - Aluguel de serviços disponíveis na nuvem
    - Exemplo: Amazon Elastic MapReduce

Vantagens (3)

- Robusto
  - Se em 1 máquina há probabilidade de haver falhas...
    - Tempo médio entre falhas para 1 nó: 3 anos
    - Tempo médio entre falhas para 1000 nós: 1 dia
  - Hadoop proporciona alta tolerância a falhas
  - Estratégias
    - Replicação dos dados
    - Armazenamento de metadados

Vantagens (4)

- Escalável
  - Permite facilmente adicionar máquinas ao aglomerado
  - Adição não implica na alteração do código-fonte
  - Limitação apenas relacionada a quantidade de recursos disponíveis

Vantagens (5)

- Foco na regra de negócio
  - Hadoop realiza o "trabalho duro“
  - Desenvolvedores podem focar apenas na abstração do problema

Desvantagens (1)

- Único nó mestre
  - Ponto único de falha
  - Pode impedir o escalonamento

Desvantagens (2)

- Dificuldade das aplicações paralelas
  - Problemas não paralelizáveis
  - Processamento de arquivos pequenos
  - Muito processamento em poucos dados

Suposições de Projeto (1)

- Os dados que serão processados não cabem em um nó
- Cada nó é hardware comum
- Falhas acontecem

- Ideias e Soluções do Apache Hadoop:
  - Sistema de arquivos distribuído
  - Replicação interna
  - Recuperação de falhas automática

Suposições de Projeto (2)

- Mover dados é caro
- Mover computação é barato
- Computação distribuída é fácil

- Ideias e Soluções do Apache Hadoop:
  - Mover a computação para os dados
  - Escrever programas que são fáceis de se distribuir

Google MapReduce

- O modelo inicial proposto pelo Google apresentou alguns conceitos para facilitar alguns problemas
- Paralelização da computação em um aglomerado de máquinas comuns
  - Centenas/Milhares de CPUs
- Paralelização e distribuição automática da computação deveria ser o mais simples possível
- O sistema de execução se encarrega de:
  - Particionar e distribuir os dados de entrada
  - Escalonar as execuções em um conjunto de máquinas
  - Tratar as falhas
  - Comunicação entre as máquinas

Subprojetos

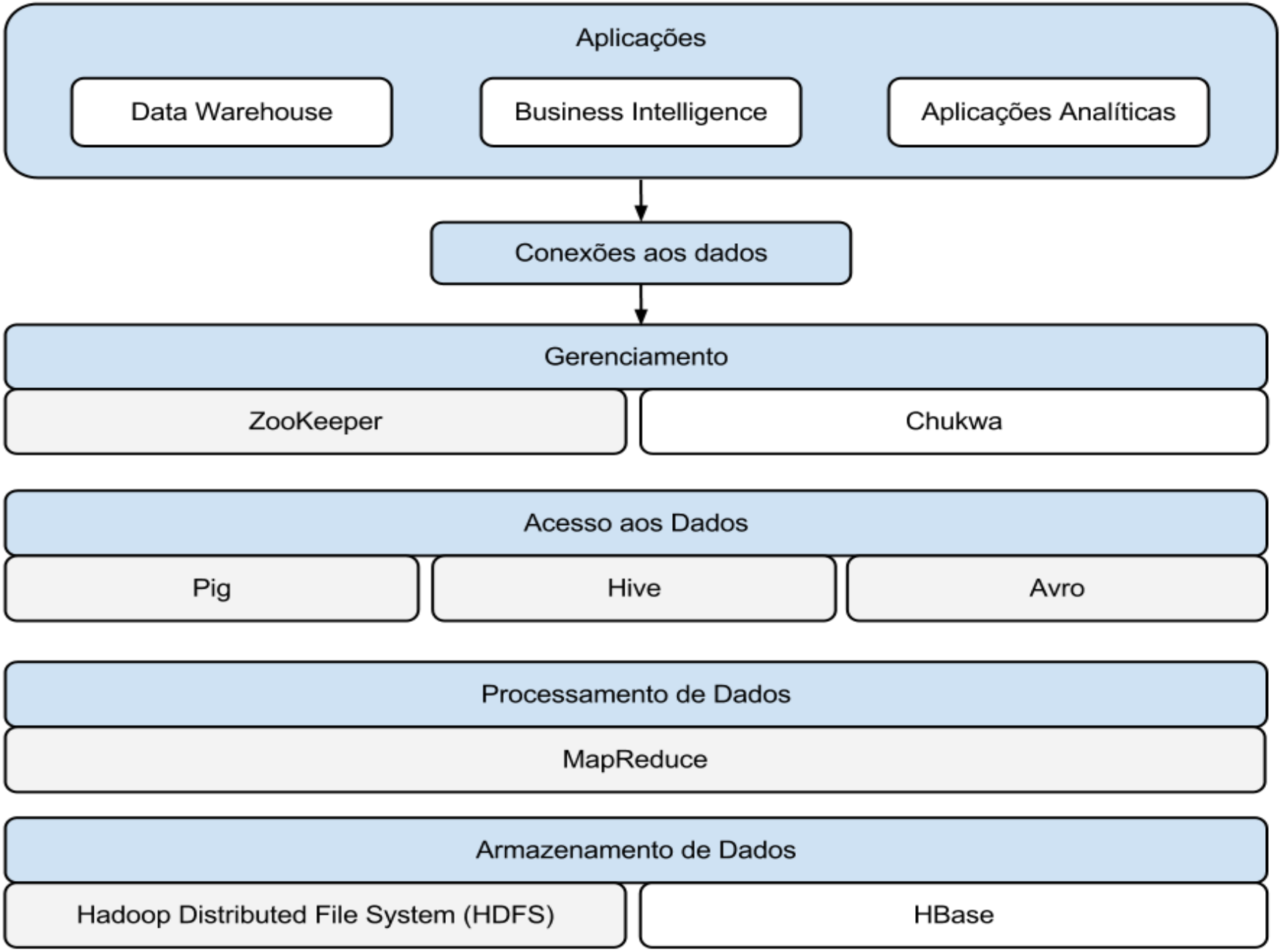
- Principais
  - Hadoop Common
  - Hadoop Distributed File System (HDFS)
  - Hadoop MapReduce

O Hadoop Common

- O Hadoop Common oculta o que os usuários comuns não precisam saber!
  - Paralelização automática
  - Balanceamento de carga
  - Otimização nas transferências de disco e rede
  - Tratamento de falhas
  - Robustez
  - Escalabilidade

Outros Subprojetos

- Avro – Seriação de dados
- Chukwa – Monitoramento de SD
- Hbase – BD distribuído e escalável
- Hive – Infraestrutura de datawarehouse
- Pig – Linguagem de fluxo de dados
- ZooKeeper – Coordenação de serviços


Aplicações
Data Warehouse
Business Intelligence
Aplicações Analíticas
Conexões aos dados
Gerenciamento
ZooKeeper
Chukwa
Acesso aos Dados
Pig
Hive
Avro
Processamento de Dados
MapReduce
Armazenamento de Dados
Hadoop Distributed File System (HDFS)
HBase

Componentes do Hadoop

- Nó Mestre:
  - NameNode
  - DataNode
  - SecondaryNameNode
- Nó(s) Escravo(s):
  - JobTracker
  - TaskTracker

Componentes do Hadoop

- NameNode
  - Gerencia os metadados dos arquivos
    - FSImage e EditLog
  - Controla a localização das réplicas
  - Encaminha os blocos aos nós escravos
  - Mantém as informações em memória

Componentes do Hadoop

- DataNode
  - Realiza o armazenamento dos dados
  - Permite armazenar diversos blocos
  - Deve se comunicar com o NameNode

Componentes do Hadoop

- SecondaryNameNode
  - Nó auxiliar do HDFS
  - Realiza pontos de checagem em intervalos pré-definidos
  - Permite manter o nível de desempenho do NameNode

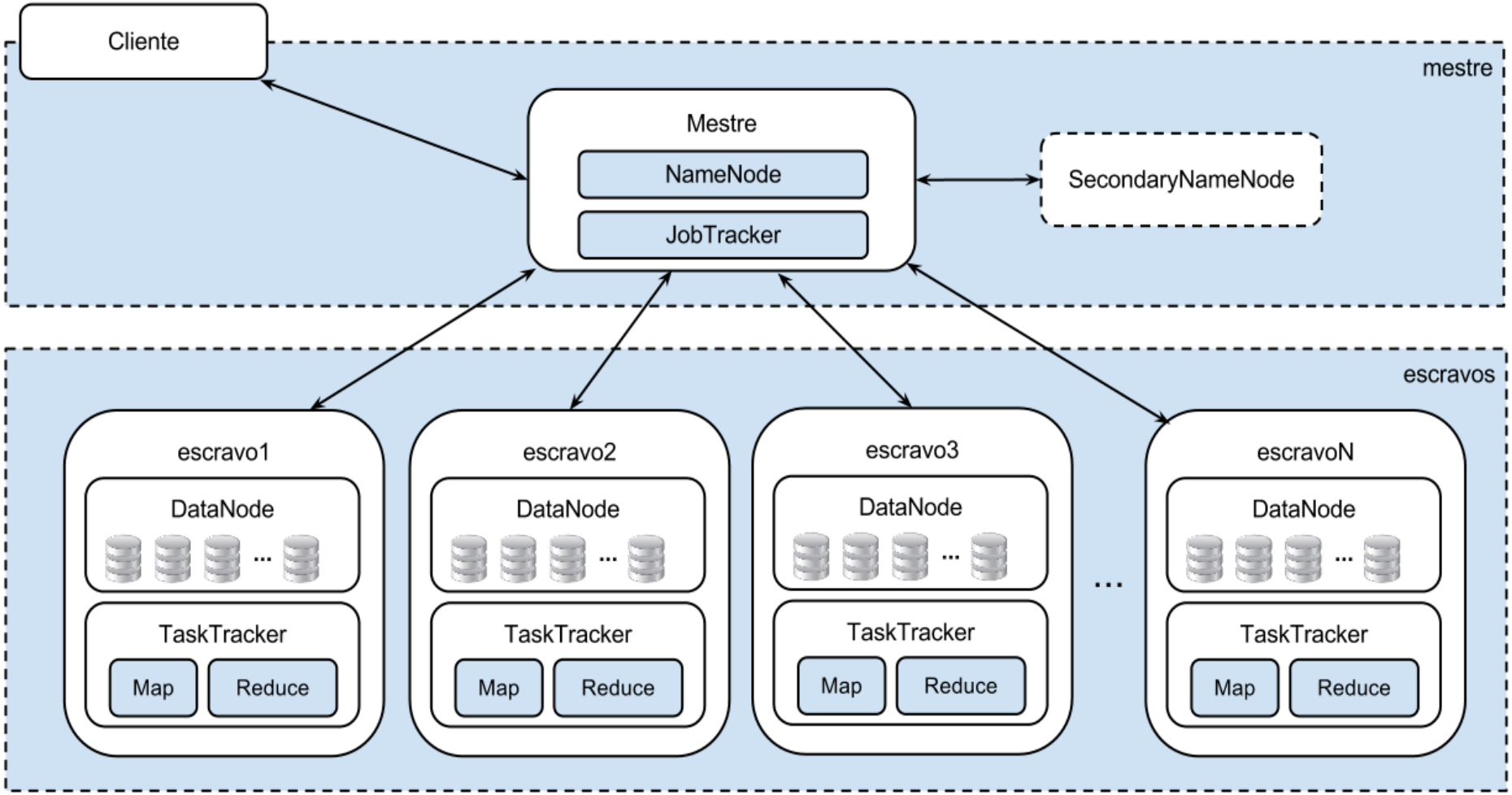
Componentes do Hadoop

- JobTracker
  - Gerencia o plano de execução de tarefas MapReduce
  - Designa as tarefas aos nós escravos
  - Monitora a execução das tarefas, para agir no caso de falhas

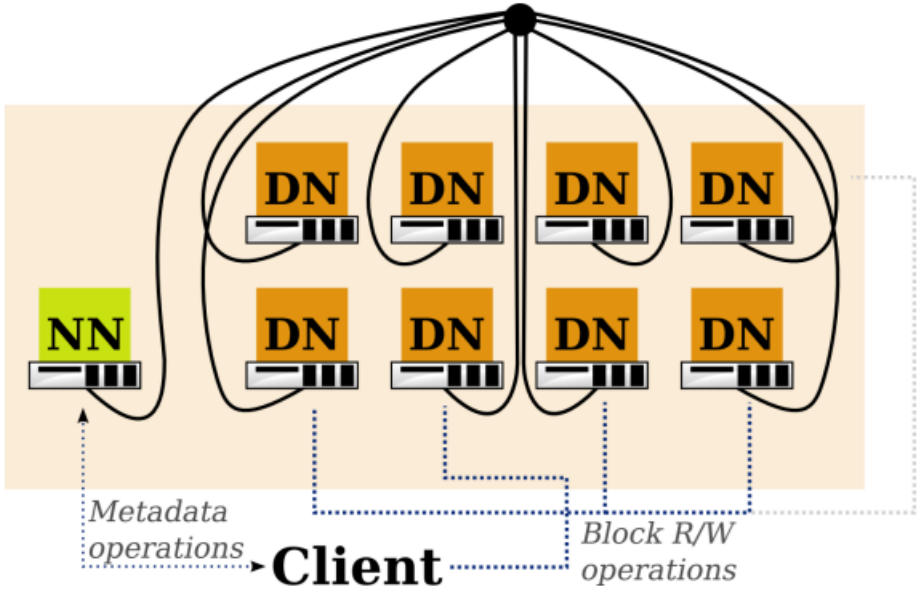
Componentes do Hadoop

- TaskTracker
  - Realiza o processamento das tarefas MapReduce
  - Uma instância em cada nó escravo

Resumindo...


Cliente
mestre
Mestre
NameNode
JobTracker
SecondaryNameNode
escravos
escravo1
DataNode
TaskTracker
Map
Reduce
escravo2
DataNode
TaskTracker
Map
Reduce
escravo3
DataNode
TaskTracker
Map
Reduce
escravoN
DataNode
TaskTracker
Map
Reduce

NameNode e DataNodes no HDFS



Fonte: Evert Lammerts (SARA.nl)

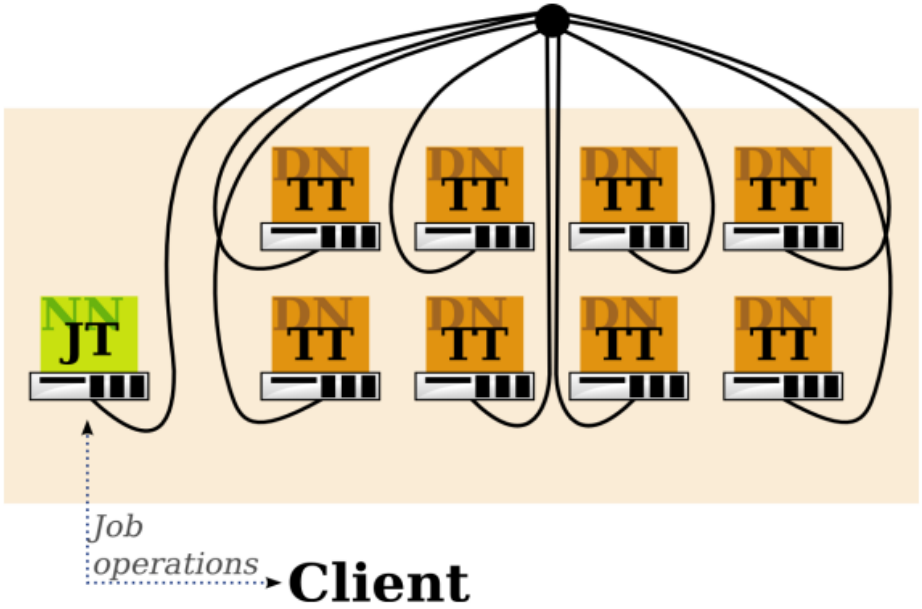
NameNode (NN)

- Gerencia o namespace do sistema de arquivos
- Mapeia nomes de arquivos para blocos
- Mapeia blocos para DataNodes
- Gerenciamento de replicação

DataNode (DN)

- Servidor de blocos que armazena
- Dados no sistema de arquivos local
- Metadados dos blocos (hash)
- Disponibiliza metadados para clientes

JobTracker e TaskTrackers no MR



Fonte: Evert Lammerts (SARA.nl)

- **JobTracker (JT)**
  - Controla os metadados
  - Status de um job
  - Status de Tasks nos TTs
  - Decide o escalonamento

TaskTrackers (TT)

- Solicita trabalho do JT
- Busca código para executar do DFS
- Aplica configurações específicas dos jobs
- Comunicam-se com o JT nas tasks
- Enviar saídas, atualizações de tasks, matar tasks, ...

Formas de Execução

- Local
- Pseudo-distribuído
- Completamente distribuído

Formas de Execução

- Local:
  - Configuração padrão
  - Recomendável para a fase de desenvolvimento e testes
  - Aplicação é executada na máquina local

Formas de Execução

- Pseudo-distribuído
  - "Cluster" de uma máquina só
  - Configuração similar à do processamento em um cluster...
  - ... porém o processamento continua sendo executado na máquina local

Formas de Execução

- Completamente distribuído
  - Processamento real de uma aplicação Hadoop
  - Deve indicar quais máquinas irão efetivamente executar os componentes Hadoop

HDFS

HDFS

- Hadoop Distributed Filesystem
  - Características
  - Divisão em blocos
  - Replicação de dados

HDFS

- Características
- Sistema de arquivos distribuídos
- Arquitetura Mestre/Escravo
- Inspirado no Google FileSystem (GFS)

Características

- Implementado em Java
- Armazenamento de grandes volumes de dados
- Recuperação de dados transparente ao usuário

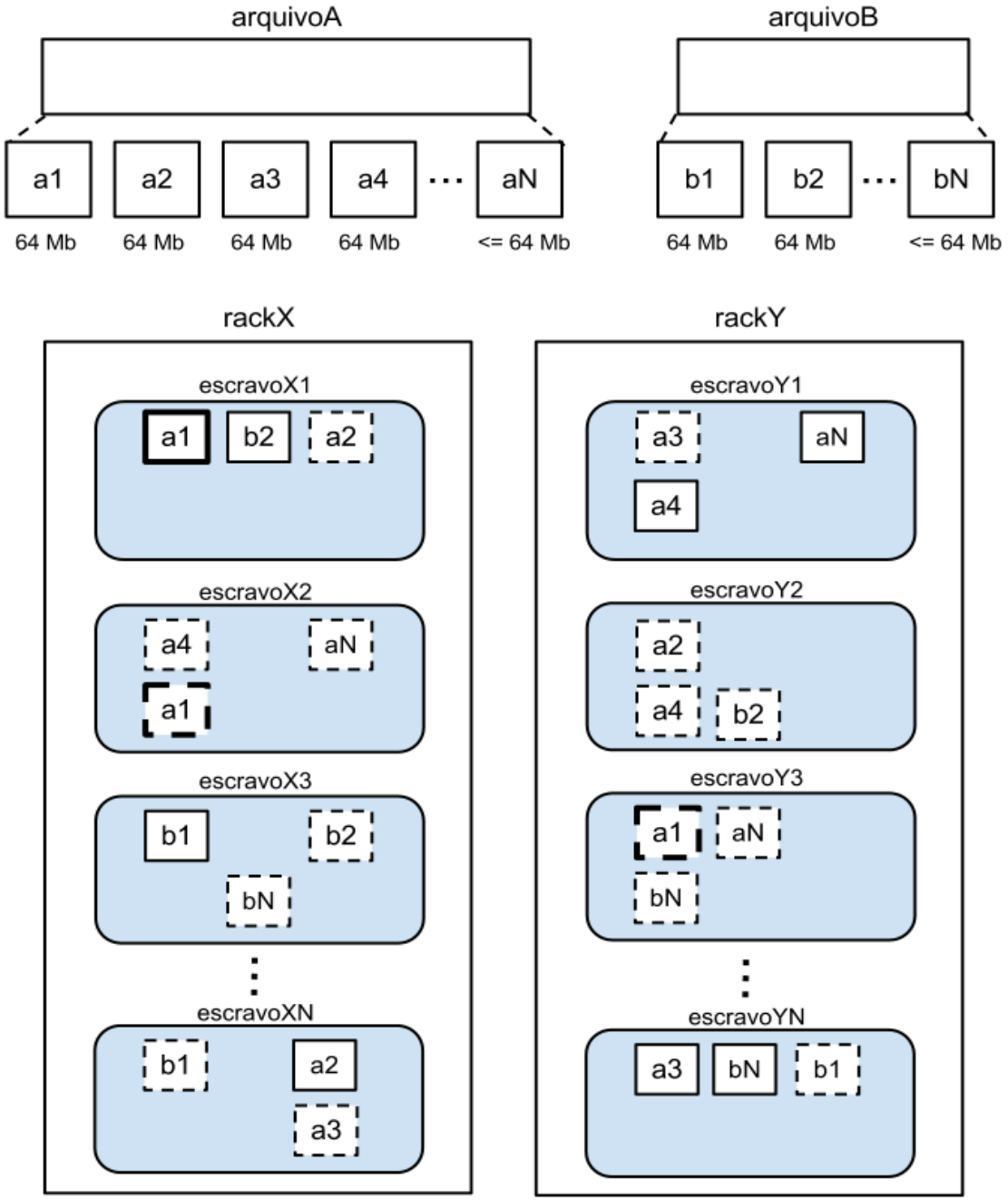
Divisão em Blocos

- Disco rígido pode não suportar o tamanho de um arquivo
  - Principalmente em soluções BigData
- HDFS divide os arquivos em blocos de mesmo tamanho
  - 64 MB por padrão

Replicação de Dados

- 3 réplicas para cada bloco
  - Aumento de segurança e disponibilidade
- Cada réplica em um diferente nó
  - 2 em um mesmo armário (rack) e 1 em um armário diferente
- Re-Replicação
  - Em casos de corromper uma das réplicas

Exemplo…


arquivoA
a1
a2
a3
a4
aN
64 Mb
64 Mb
64 Mb
64 Mb
<= 64 Mb
arquivoB
b1
b2
bN
64 Mb
64 Mb
<= 64 Mb
rackX
escravoX1
a1
b2
a2
escravoX2
a4
aN
a1
escravoX3
b1
b2
bN
escravoXN
b1
a2
a3
rackY
escravoY1
a3
aN
a4
escravoY2
a2
a4
b2
escravoY3
a1
aN
bN
escravoYN
a3
bN
b1

MapReduce

Um pequeno grande exemplo

- Word Count (Conta Palavras)
- Gera uma lista da frequência das palavras em um conjunto de arquivos.
  - Conjunto de arquivos: terabytes!

CSBC JAI 2012
CSBC 2012 em Curitiba

Minicurso Hadoop JAI 2012
CSBC 2012 Curitiba Paraná

Word Count

2012, 4
CSBC, 3
Curitiba, 2
em, 1
JAI, 2
Hadoop, 1
Minicurso, 1
Paraná, 1

Em um mundo não paralelo!

- Assuma que a máquina tem memória suficiente (1+ Tb?)

```
word-count ( ) {
  for each document d {
        for each word w in d {
              w_count[w]++;
    }
  }
  save w_count to persistent storage
}
```

- Provavelmente a execução demorará um longo tempo (dias, semanas...) pois a entrada é da ordem de terabytes

Em um mundo paralelo qualquer!

```
Mutex lock; // protects w_count
word-count ( ) {
     for each document d in parallel {
          for each word w in d {
               lock.Lock();
                    w_count[w]++;
       lock.Unlock();
          }
       }
       save w_count to persistent storage }
```

- Problemas: utiliza uma estrutura de dados única e global.
  - Recursos compartilhados: seção crítica!

No Mundo Hadoop

- Usando o MapReduce podemos resolver problemas da seguinte forma:
  - Leia uma grande quantidade de dados
  - Aplique a função **MAP**: extrai alguma informação de valor!
  - Fase intermediária: Shuffle & Sort
  - Aplique a função **REDUCE**: reúne, compila, filtra, transforma,...
  - Grava os resultados

MapReduce

- A ideia do paradigma de programação Map e Reduce não é nova
- Provavelmente 40+ anos!
- No Hadoop é a parte do arcabouço responsável pelo processamento distribuído (paralelo) de grandes conjuntos de dados.
- Provê um modelo de programação
- Usa padrões já conhecidos:

```
cat   | grep | sort    | unique  >  file
input | map  | shuffle | reduce  >  output
```

A natureza do Map

- Map em programação funcional

map({1,2,3,4}, (x2)) -> {2,4,6,8}

- Todos os elementos são processados por um método e os elementos não afetam uns aos outros

A natureza do Reduce

- Reduce em programação funcional

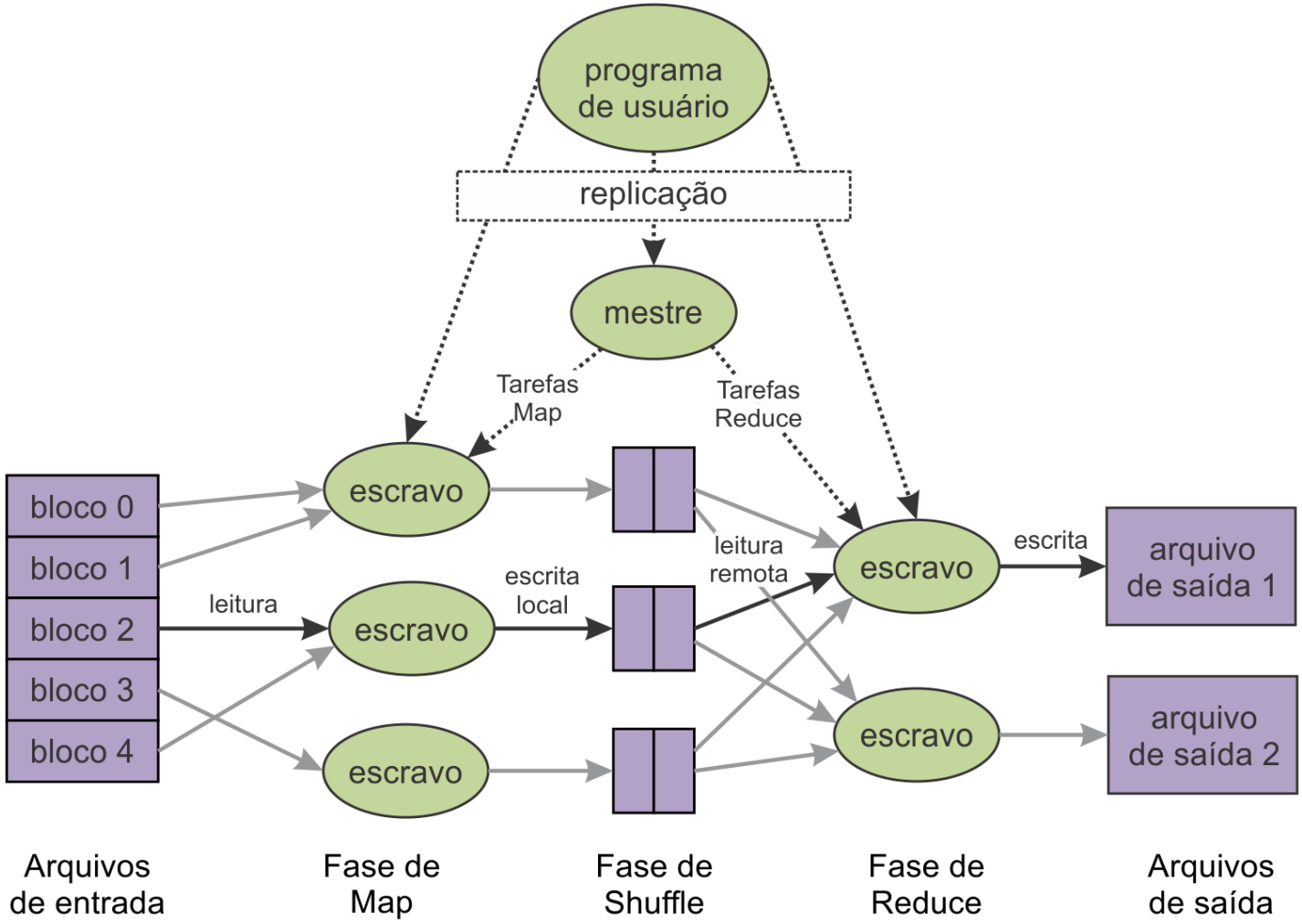
reduce({1,2,3,4}, (x)) -> {24}

- Todos elementos na lista são processados juntos
- Tanto em Map quanto em Reduce: a entrada é fixa (imutável), e a saída é uma nova lista (em geral)

O paradigma implementado

- O paradigma MapReduce é adequado para trabalhar com grandes quantidades de dados
- Realiza computação sobre os dados (pouca movimentação de dados)
- Utiliza os blocos armazenados no DFS, logo não necessita divisão dos dados

Ilustrando a ideia original



MapReduce no Hadoop

- A função Map atua sobre um conjunto de entrada com chaves e valores, produzindo uma lista de chaves e valores
- A função Reduce atua sobre os valores intermediários produzidos pelo Map para, normalmente, agrupar os valores e produzir a saída

	Input	Output
map	<k1, v1>	lista(<k2, v2>)
reduce	<k2, lista(v2)>	lista(<k3, v3>)

Exemplos: Word Count

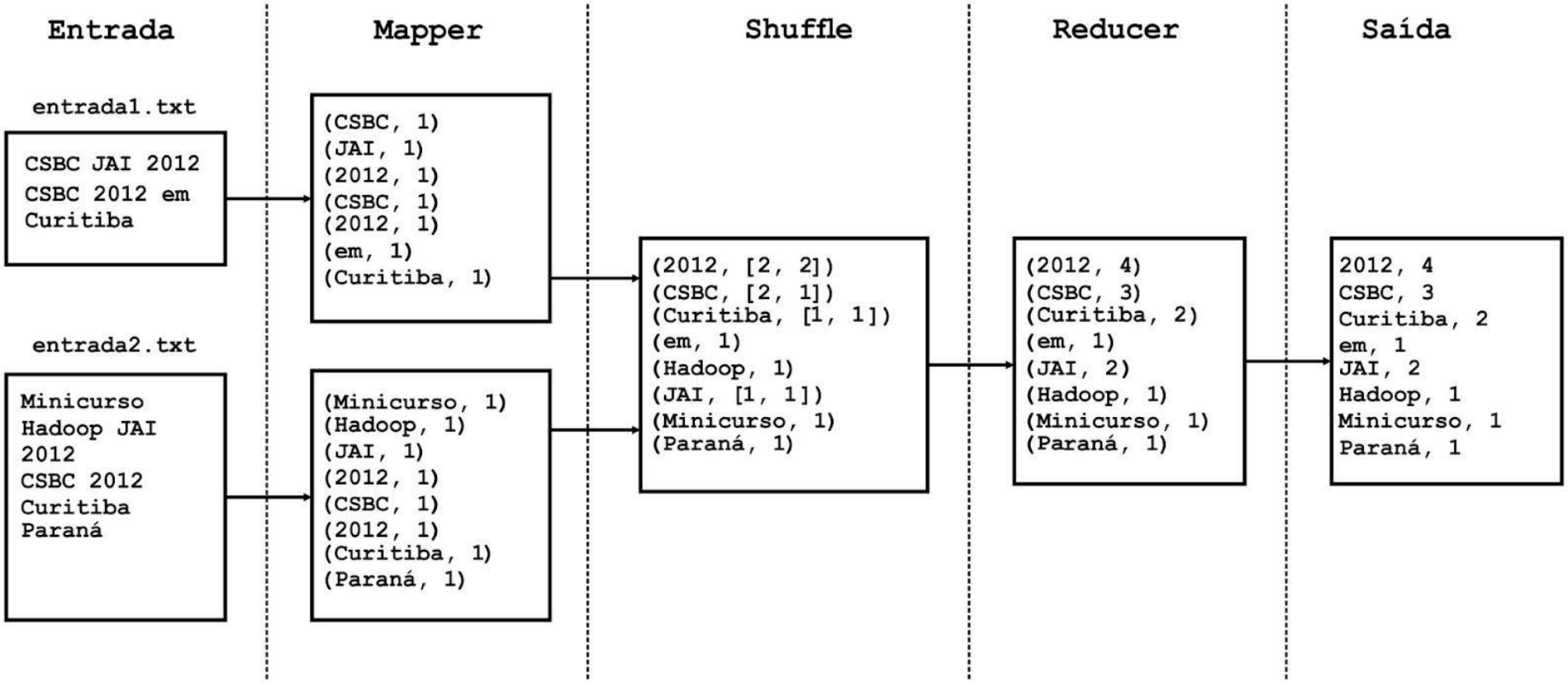
- Lê arquivos texto e conta a frequência das palavras
  - **Entrada**: arquivos texto
  - **Saída**: arquivo texto
  - **Cada linha**: palavra, separador (tab), quantidade
- **Map**: gera pares de (palavra, quantidade)
- **Reduce**: para cada palavra, soma as quantidades

Word Count (Pseudo-código)

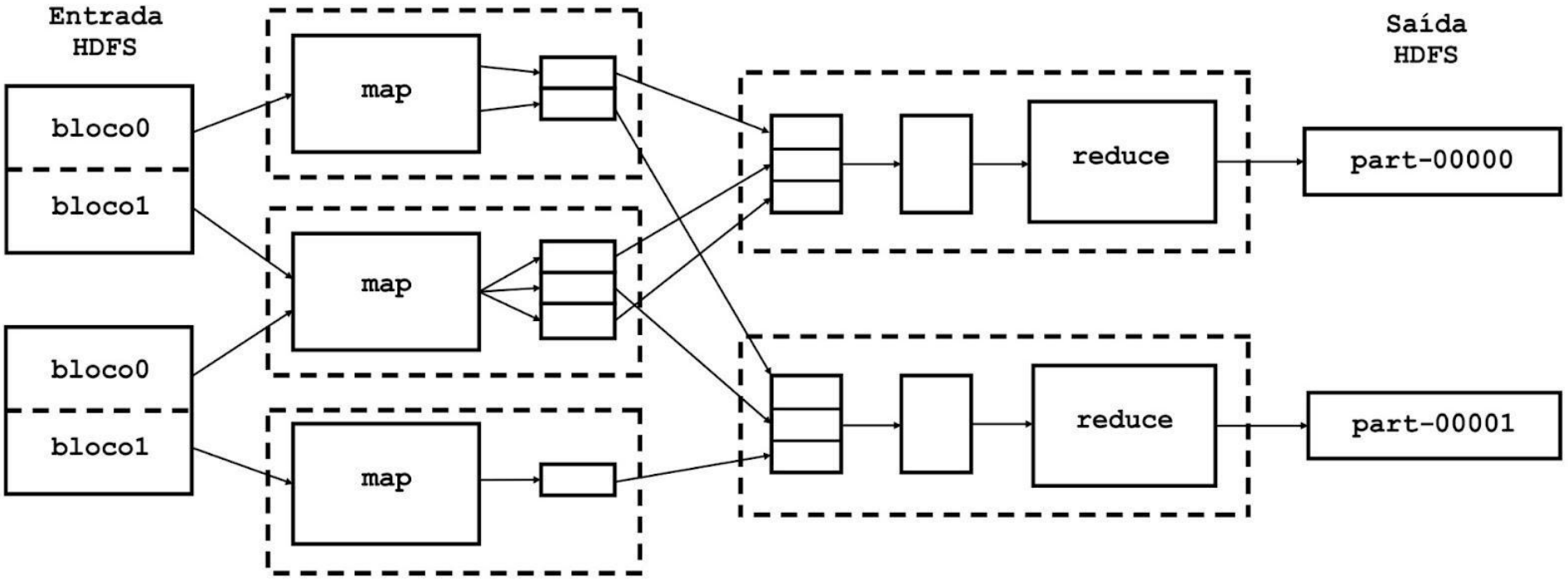
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
       EmitIntermediate(w, “1”);

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
       result += ParseInt(v);
  Emit(AsString(result));
```

Execução do WordCount



Implementação do Hadoop

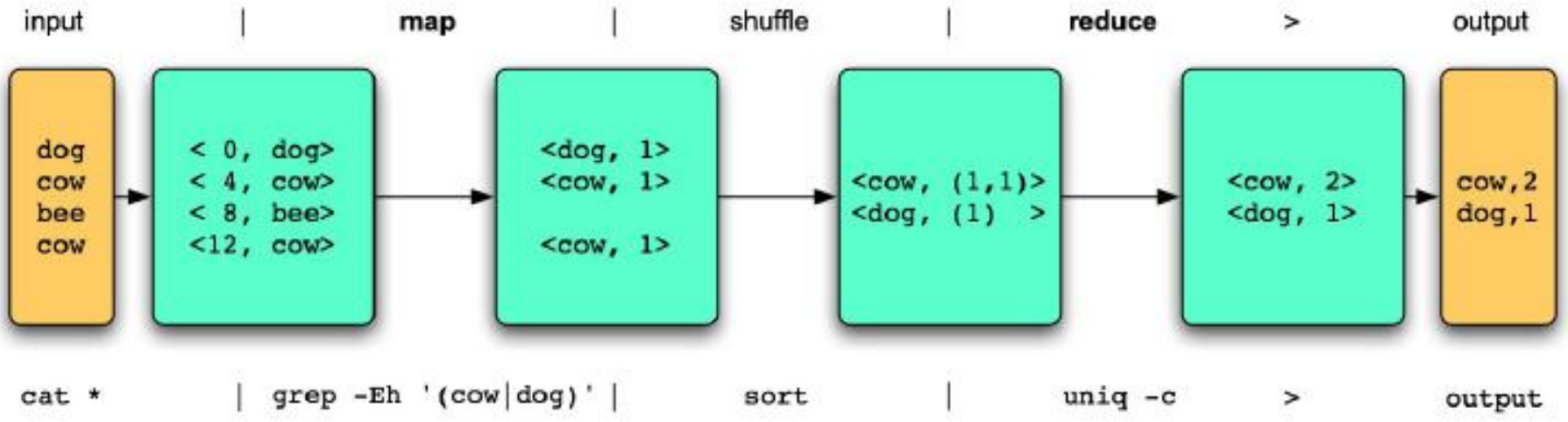


Outros exemplos: Grep

- Procura nos arquivos de entrada por um dado padrão
- **Map**: emite uma linha se um padrão é encontrado
- **Reduce**: copia os resultados para a saída

Ilustrando o Grep

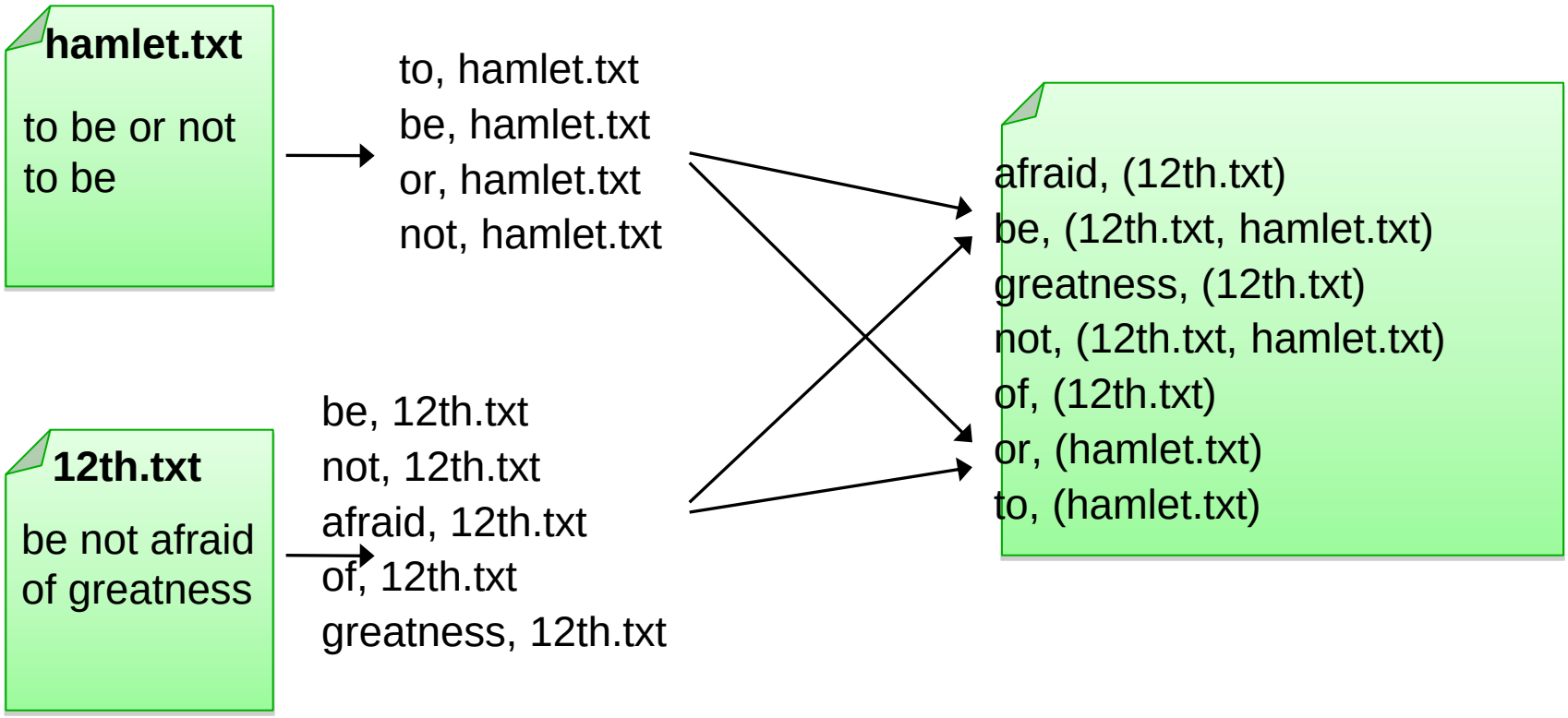
cat | grep | sort | unique > file
input | map | shuffle | reduce > output



Outros exemplos: Índice Invertido

- Gerar o índice invertido das palavras de um conjunto de arquivos dado
- **Map**: faz a análise dos documentos e gera pares de (**`palavra, docId`**)
- **Reduce**: recebe todos os pares de uma palavra, organiza os valores docId, e gera um par (**`palavra, lista(docId)`**)

Ilustrando o Índice Invertido



Apache Mahout

- É uma biblioteca de algoritmos de aprendizagem de máquina
- É um projeto da Apache Software Foundation
- Software Livre (Licença Apache)
- Principal objetivo é ser escalável para manipular volume gigantesco de dados

Onde usar o Mahout?

- Trabalha com:
  - Matrizes e vetores
  - Estruturas esparsas e densas
  - Agrupamento
  - Cobertura
  - K-Means
  - Análise de densidade de funções
  - Filtragem colaborativa
- Mahout pode ser usado com o Hadoop explorando sua escalabilidade para processar os dados

Gerando recomendações

- Construir uma matriz que relaciona os itens:
- Matriz de co-ocorrência
- Computa o número de vezes que cada par de itens aparecem juntos na lista de preferências de algum usuário
- Se existem 9 usuários que expressam preferência pelos itens X e Y, então X e Y co-ocorrem 9 vezes
- Co-ocorrência é como similaridade, quanto mais dois itens aparecem juntos, mais provável que sejam similares

Gerando recomendações

	101	102	103	104	105	106	107
101	5	3	4	4	2	2	1
102	3	3	3	2	1	1	0
103	4	3	4	3	1	2	0
104	4	2	3	4	2	2	1
105	2	1	1	2	2	1	1
106	2	1	2	2	1	2	0
107	1	0	0	1	1	0	1

Gerando recomendações

- Computando o vetor de cada usuário
- Um vetor para cada usuário
- Com n itens na base de dados, o vetor de preferências terá n dimensões
- Se o usuário não exprime nenhuma preferência por um determinado item, o valor correspondente no vetor será zero
- Neste exemplo, o vetor do usuário três é [2.0, 0.0, 0.0, 4.0, 4.5, 0.0, 5.0]

Gerando recomendações

	101	102	103	104	105	106	107		U3		R
101	5	3	4	4	2	2	1		2.0		40.0
102	3	3	3	2	1	1	0		0.0		18.5
103	**4**	**3**	**4**	**3**	**1**	**2**	**0**	**X**	**0.0**	**=**	**24.5**
104	4	2	3	4	2	2	1		4.0		40.0
105	2	1	1	2	2	1	1		4.5		26.0
106	2	1	2	2	1	2	0		0.0		16.5
107	1	0	0	1	1	0	1		5.0		15.5

Multiplicando a matriz de co-ocorrência com o vetor de preferências do usuário três para chegar ao vetor que nos leva às recomendações

Gerando recomendações

- Intuitivamente, olhando para a linha 3 da tabela, se o item desta linha co-ocorre com muitos itens que o usuário 3 expressou sua preferência, então é provável que seja algo que o usuário 3 goste

Integrando o Mahout no Hadoop

- Precisamos do RecommenderJob
- Modo Newbie (Novato!):
  - Apenas coloque o JAR pré-compilado da distribuição do Mahout no diretório do hadoop.
  - mahout-core-0.6-job.jar
- Modo Expert:
  - Faça as alterações necessárias no Mahout para sua persolnalização, gere o JAR e coloque no diretório do hadoop.

Chamada do Mahout no Hadoop

```
:~$ bin/hadoop jar mahout-core-0.6-job.jar
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-Dmapred.input.dir=input/movieRec10M.txt
-Dmapred.output.dir=output
--usersFile input/movieUsers.txt
--numRecommendations 10
--maxPrefsPerUser 100
--similarityClassname SIMILARITY_COSINE
```

Formato dos dados

O recomendador do Mahout espera que os dados estejam da forma:

userID, itemID [,preferencevalue]

- UserID é um Long
- ItemID é um Long
- Preferencevalue é um Double

Carregar os dados no HDFS

Comando:

hadoop fs -put <caminho_local_do_arquivo> <caminho_do_arquivo_no_hdfs>

Ex:

:~$ hadoop fs -put moviesRecommendation1M.txt input/moviesRec.txt

Executando o RecomenderJob

Parâmetros de chamada:

--usersFile(path): (xor) arquivo contendo os Ids dos usuários considerados na computação

--itemsFile(path): (xor) arquivo contendo os Ids dos itens;

--numRecommendations(integer): número de recomendações computadas por usuário (padrão:10)

--booleanData(boolean): tratar a entrada como não tendo valores de preferência (padrão:falso)

--maxPrefsPerUser(integer): número máximo de preferências consideradas por usuário (padrão:10)

Executando o RecomenderJob

Parâmetros de chamada:

--similarityClassname(classname): (obrigatório) medida de similaridade;

- SIMILARITY_COOCCURRENCE
- SIMILARITY_LOGLIKELIHOOD
- SIMILARITY_TANIMOTO_COEFFICIENT
- SIMILARITY_CITY_BLOCK
- SIMILARITY_COSINE
- SIMILARITY_PEARSON_CORRELATION
- SIMILARITY_EUCLIDEAN_DISTANCE

Demonstração

- Rodar o Hadoop na máquina local
  - Pseudo-Distributed mode
- Colocar rating para 10 filmes que já assistiu
- Enviar arquivos para o HDFS
- Rodar o recomendador
- Aplicar script python nos resultados

Preenchendo com suas recomendações

Edite o arquivo yourRec.txt e coloque suas recomendações nele:

- Crie um id único a partir de 1000 (chute um valor);
- Procure no arquivo filmes.txt por filmes que você já assistiu e classifique-os com nota de 1 a 5.
- Cada linha de seu arquivo deverá conter a seguinte estrutura:
- <user_id_criado>,<id_filme>,<rating>
- ex: 1977,123,4
- Faça isto para no mínimo dez filmes.

Preenchendo com suas recomendações

- Agora vamos juntar suas recomendações com o arquivo de recomendações de outros usuários com o comando:

:~$ cat yourRec.txt >> rec.txt

- Edite o arquivo user.txt que contenha apenas uma linha com seu user_id criado nela.

Enviando arquivos ao HDFS

- Agora que os arquivos estão preenchidos, envie seus dois arquivos para o fs com os comandos:

```
:~$ bin/hadoop fs -put rec.txt input/rec.txt

:~$ bin/hadoop fs -put user.txt input/user.txt
```

Rodando o RecommenderJob

Verificar se o arquivo mahout-core-0.6-job.jar está no diretório raiz do hadoop.

- A partir do diretório raiz do hadoop, executar o comando:

:~$ bin/hadoop jar mahout-core-0.6-job.jar
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-Dmapred.input.dir=input/rec.txt
-Dmapred.output.dir=output
--usersFile input/user.txt
--numRecommendations 10
--maxPrefsPerUser 100
--similarityClassname SIMILARITY_COSINE

Resultados!

- Acesse em seu navegador: http://localhost:50070/
  - Opção: “Browse the filesystem”

- Verificar os filmes recomendados para você usando o script imprimeFilmes.py com o seguinte comando:

:~$ python imprimeFilmes <arquivo_resultado> <arquivo_filmes> <id_usuario>

Referências!

- Livros:
- Hadoop – The Definitive Guide
  - Tom White – 2ª Ed.
- Hadoop in Action
  - Chuck Lam – 1ª Ed.
- Web:
  - http://wiki.apache.org/hadoop/
- Materiais extras:
  - Luciana Arantes