Apache Hadoop

Conceitos teóricos e práticos, evolução e novas possibilidades

Daniel Cordeiro

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Baseado no curso apresentado no CSBC'12 pelos professores
Alfredo Goldman e Fabio Kon (USP);
Francisco Pereira Jr., Ivanilton Polato e Rosangela de Fátima Pereira (UTFPR)

ERAD/SP – 25 de julho de 2012

Motivação

Uso potencial em 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

Hadoop remove a complexidade da computação de alto desempenho

Custo eficiente

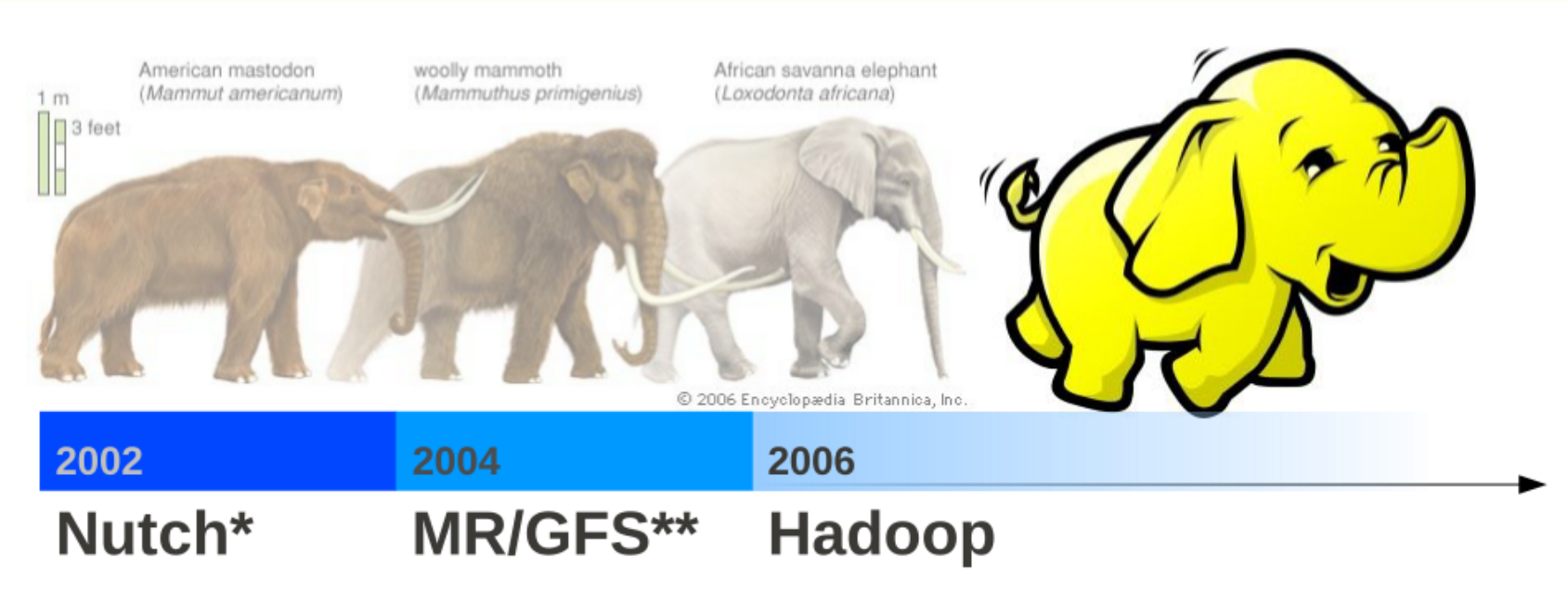
- Máquinas comuns
- Rede comum
- Tolerância a falhas automática
  - Poucos administradores
- Facilidade de uso
  - Poucos programadores

Hadoop

Arcabouço para processamento e armazenamento de dados em larga escala:

- Código aberto
- Implementado em Java
- Inspirado no GFS e MapReduce do Google
- Projeto *top-level* 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

Origem (I)

- 2003 Google publica artigo do GFS (SOSP'03)
- 2004 Google publica artigo do MapReduce (OSDI'04)
- 2005 Doug Cutting cria uma versão do MapReduce para o projeto Nutch
- 2006 Hadoop se torna um subprojeto do Apache Lucene

Origem (II)

2007 Yahoo! Inc. se torna o maior contribuidor e utilizador do projeto (aglomerado com mais de 1.000 nós)

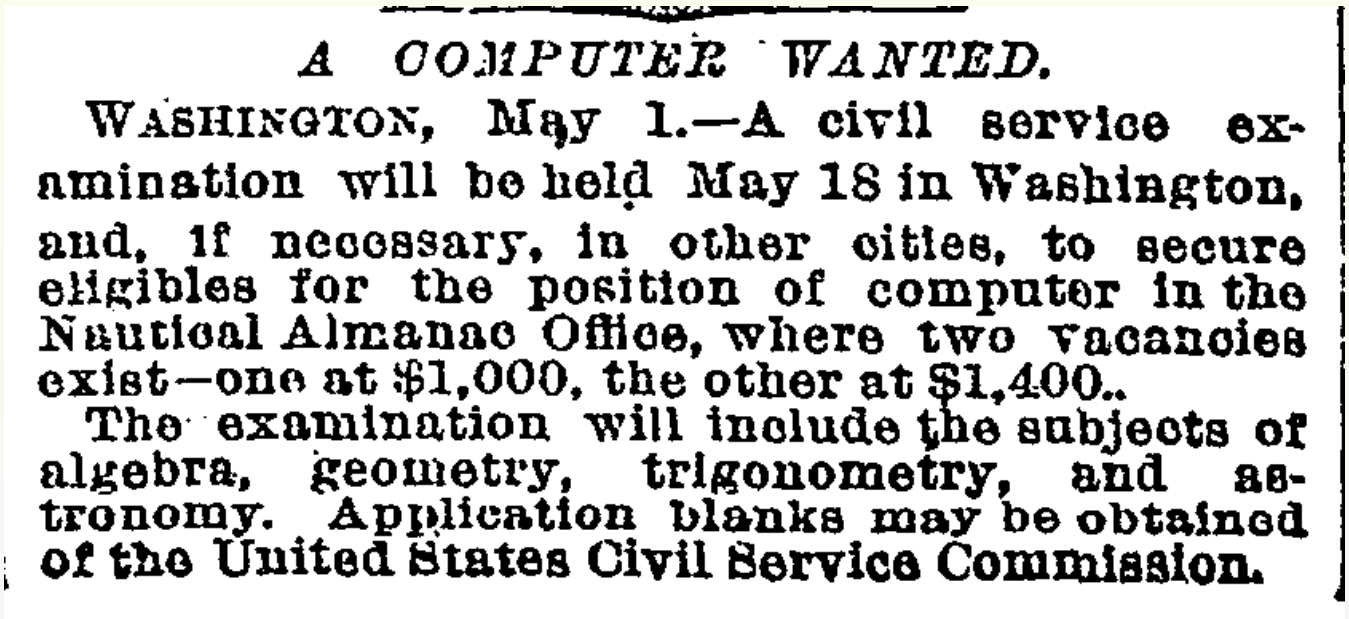
2008 Hadoop deixa a tutela do projeto Lucene e se transforma em um projeto *top-level* da Apache

2010 Facebook anuncia o maior aglomerado Hadoop do mundo (mais de 2.900 nós e 30 petabytes de dados)

2011 Apache disponibiliza a versão 1.0.0

Quem utiliza?

The New York Times



A COMPUTER WANTED.

WASHINGTON, May 1.—A civil service examination will be held May 18 in Washington, and, if necessary, in other cities, to secure eligibles for the position of computer in the Nautical Almanac Office, where two vacancies exist—one at $1,000, the other at $1,400..

The examination will include the subjects of algebra, geometry, trigonometry, and astronomy. Application blanks may be obtained of the United States Civil Service Commission.

The New York Times

Published: May 2, 1892

http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/

The New York Times

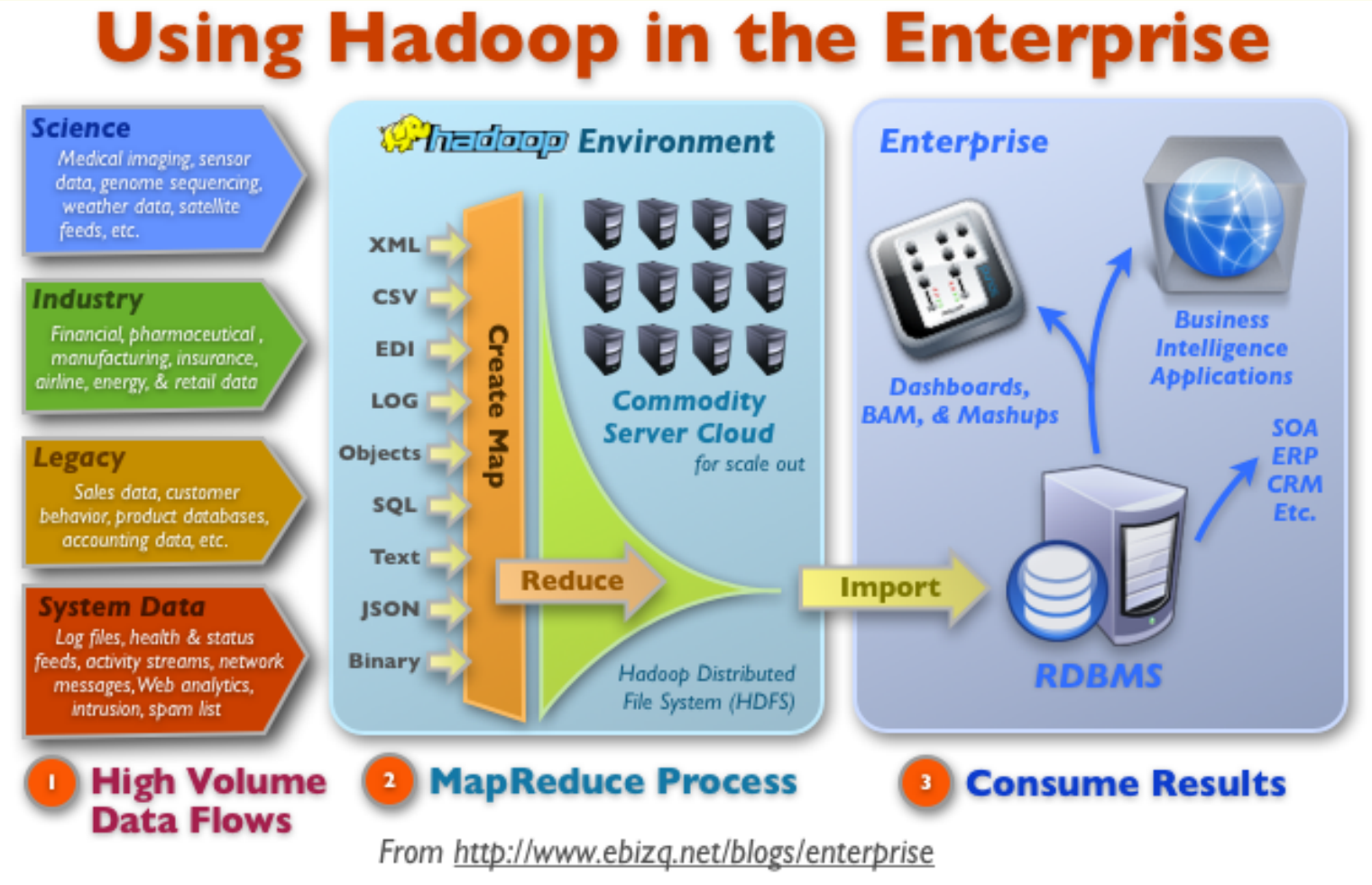
- Em 2007, o jornal The New York Times converteu para PDF todos seus os artigos publicados entre 1851 e 1980
- Cada artigo é composto por várias imagens previamente digitalizadas que precisavam ser posicionadas e redimensionadas de forma coerente pra a criação do PDF
- O Hadoop foi utilizado para converter 4 TB de imagens TIFF em 11 milhões de arquivos PDF
- 100 instâncias EC2 da Amazon foram utilizadas durante 24 horas para gerar 1,5 TB de arquivos PDF, a um custo de aproximadamente US$ 240,00

Onde o Hadoop é utilizado?

Solução para:

- Data warehouse
- Business intelligence
- Aplicações analíticas
- Mídias sociais

Muitas possibilidades...



Vantagens

Por que usar Hadoop?

- Código aberto
- Econômico
- Robusto
- Escalável
- Foco na regra de negócio

Vantagem I

Código Aberto

- Comunidade ativa
- Apoio de grandes corporações
- Correções de erros frequentes
- Constante evolução do arcabouço

Vantagem II

Econômico

- Software livre
- Uso de máquinas e redes convencionais
- Aluguel de serviços disponíveis na nuvem:
  - Amazon Elastic MapReduce
  - Google App Engine MapReduce
  - etc.

Vantagem III

Robusto

- Se em 1 máquina há probabilidade de haver falhas...
  - Tempo médio entre falhas para 1 nó: 3 anos
  - Tempo médio entre falhas para 1.000 nós: 1 dia

Estratégias

- Replicação dos dados
- Armazenamento de metadados

Vantagem IV

Escalável

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

Vantagem V

Foco na regra de negócio

- Hadoop realiza todo o "trabalho duro"
- Desenvolvedores podem focar apenas na abstração do problema

Desvantagens

Único nó mestre

- Ponto único de falha
- Pode impedir o escalonamento

Dificuldade das aplicações paralelas

- Problemas não paralelizáveis
- Processamento de arquivos pequenos
- Muito processamento em um pequeno conjunto de dados

Suposições do projeto (I)

Problemas

- Os dados que serão processados não cabem em um nó
- Cada nó é composto por hardware comum
- Falhas podem (e irão) acontecer

Ideias e soluções do Apache Hadoop

- Sistema de arquivos distribuído
- Replicação interna
- Recuperação de falhas automática

Suposições do projeto (II)

Problemas

- Mover dados é caro (largura de banda pequena)
- Mover computação é barato
- Programação paralela e distribuída é difícil

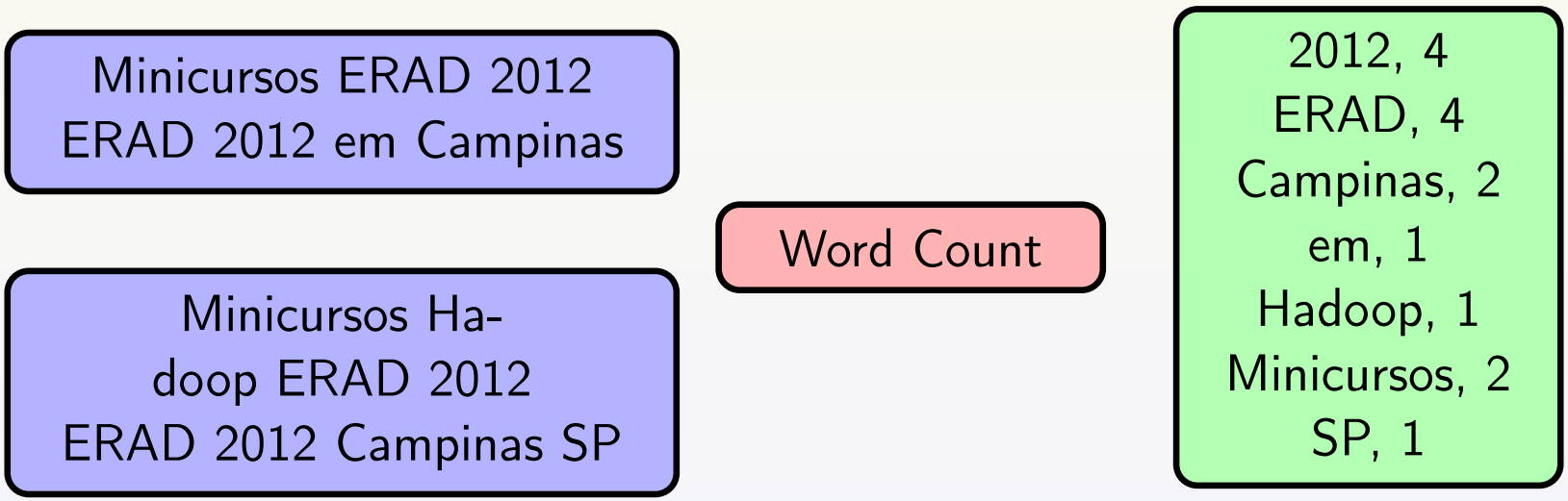
Ideias e soluções do Apache Hadoop

- Mover a computação para onde estão os dados
- Escrever programas que são fáceis de se distribuir
- Paralelismo de dados utilizando conceitos de linguagem funcional

O exemplo clássico: contagem de palavras

Word Count

Gerar uma lista de frequência das palavras em um conjunto **grande** de arquivos: ordem de *terabytes*!



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
}
```

Fácil, mas provavelmente a execução demorará um longo tempo, pois a entrada é da ordem de *terabytes*

Em um mundo paralelo qualquer

```
Mutex lock;  // protege 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!

Google MapReduce

- O modelo inicial proposto pelo Google apresentou conceitos para simplificar alguns problemas
- Paralelização da computação em um aglomerado de máquinas comuns (com centenas/milhares de CPUs)
- Paralelização e distribuição automática de 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

Ideia básica do MapReduce

O modelo de programação paralela MapReduce aborda os problemas da seguinte forma:

1. Leia uma grande quantidade de dados
2. Aplique a função **MAP**: extrai alguma informação de valor!
3. Fase intermediária: Shuffle & Sort
4. Aplique a função **REDUCE**: reúne, compila, filtra, transforma, etc.
5. Grave os resultados

MapReduce

- A ideia do modelo de programação Map e Reduce não é nova
- Presente em linguagens funcionais há mais de 40 anos!
- No Hadoop é a parte do arcabouço responsável pelo processamento distribuído (paralelo) de grandes conjuntos de dados
- Usa padrões já conhecidos:

cat	| grep	| sort	| uniq	>	arquivo
entrada	| map	| shuffle	| reduce	>	saída

A natureza do Map

Map em programação funcional

```
map({1,2,3,4}, (×2)) -> {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}, (×)) -> {24}
```

- Todos os elementos da 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 modelo implementado

- O modelo MapReduce é adequado para trabalhar com grandes quantidades de dados
- Realiza computação sobre os dados (pouca movimentação de dados)
- Os dados são compartilhados através de um *sistema de arquivos distribuído*

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

	Entrada	**Saída**
map	`<k1, v1>`	`lista(<k2, v2>)`
reduce	`<k2, lista(v2)>`	`lista(<k3, v3>)`

De volta ao exemplo do Word Count

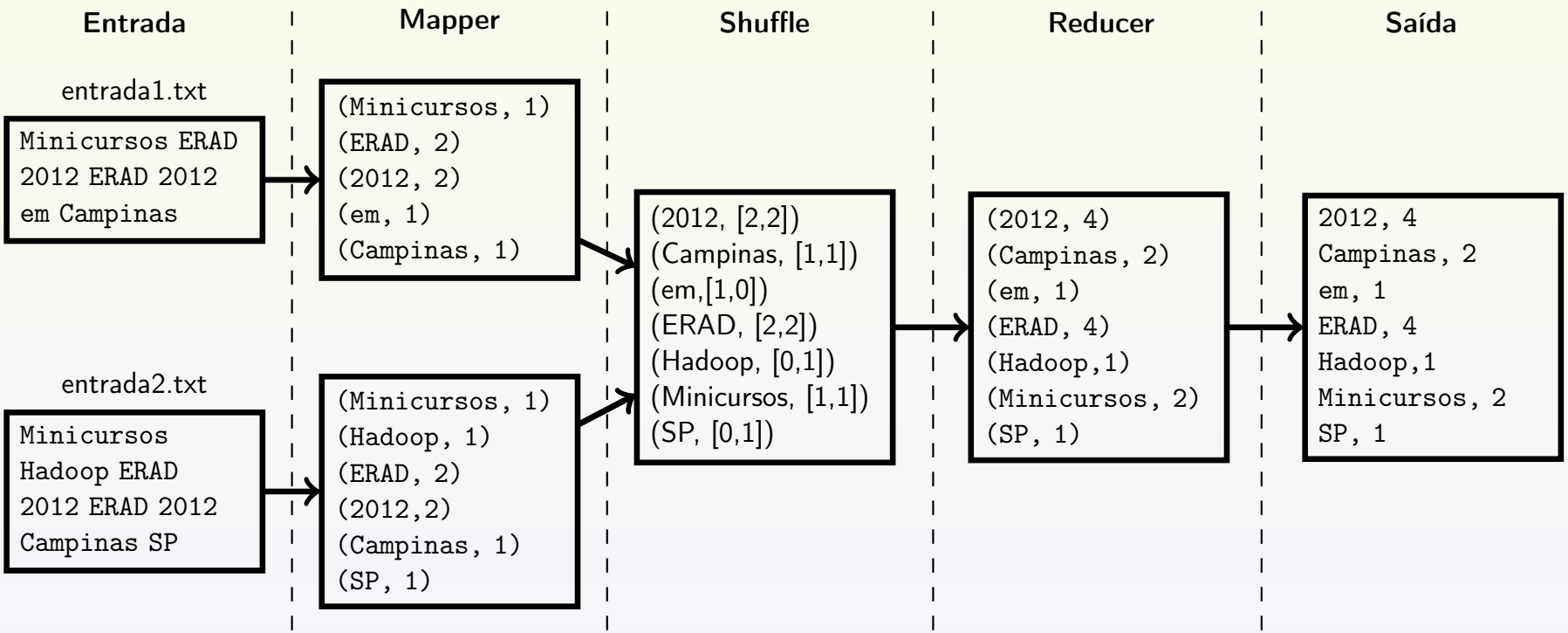
- Lê arquivos texto e conta a frequência das palavras
  - **Entrada**: arquivos texto
  - **Saída**: arquivo texto
  - **Cada linha**: palavra, separador (tab), quantidade
- **Map**: gera pares (**palavra**, **quantidade**)
- **Reduce**: para cada palavra, soma as quantidades

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

Execução do Word Count

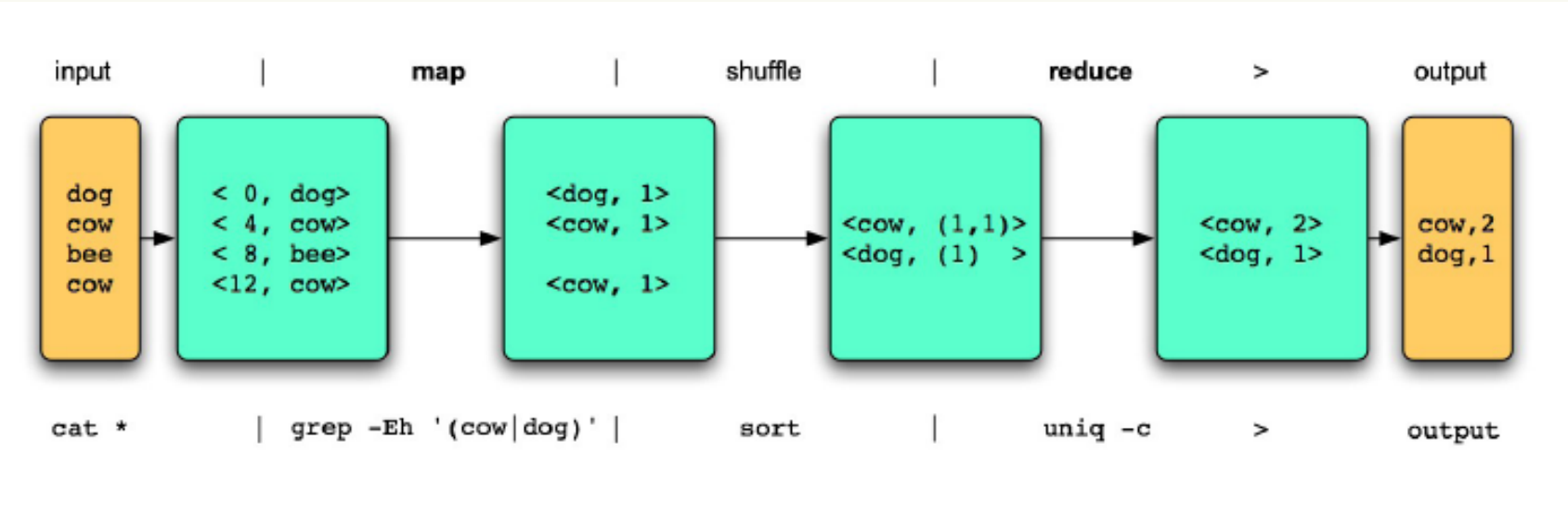


Outros exemplos: Grep

- 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

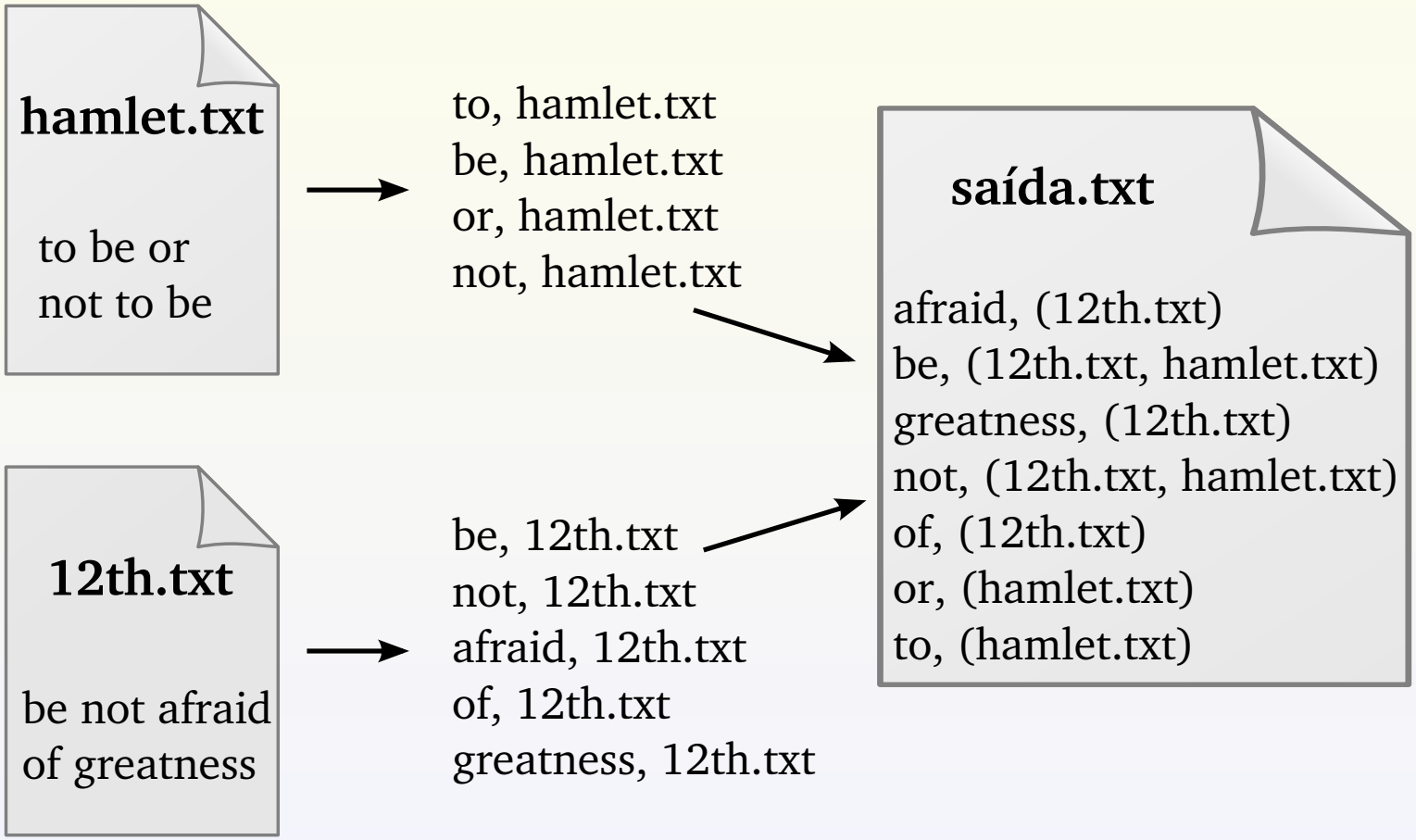
cat	|	grep	|	sort	|	uniq	>	arquivo
entrada	|	map	|	shuffle	|	reduce	>	saída



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



Subprojetos do Hadoop

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

O Hadoop Common

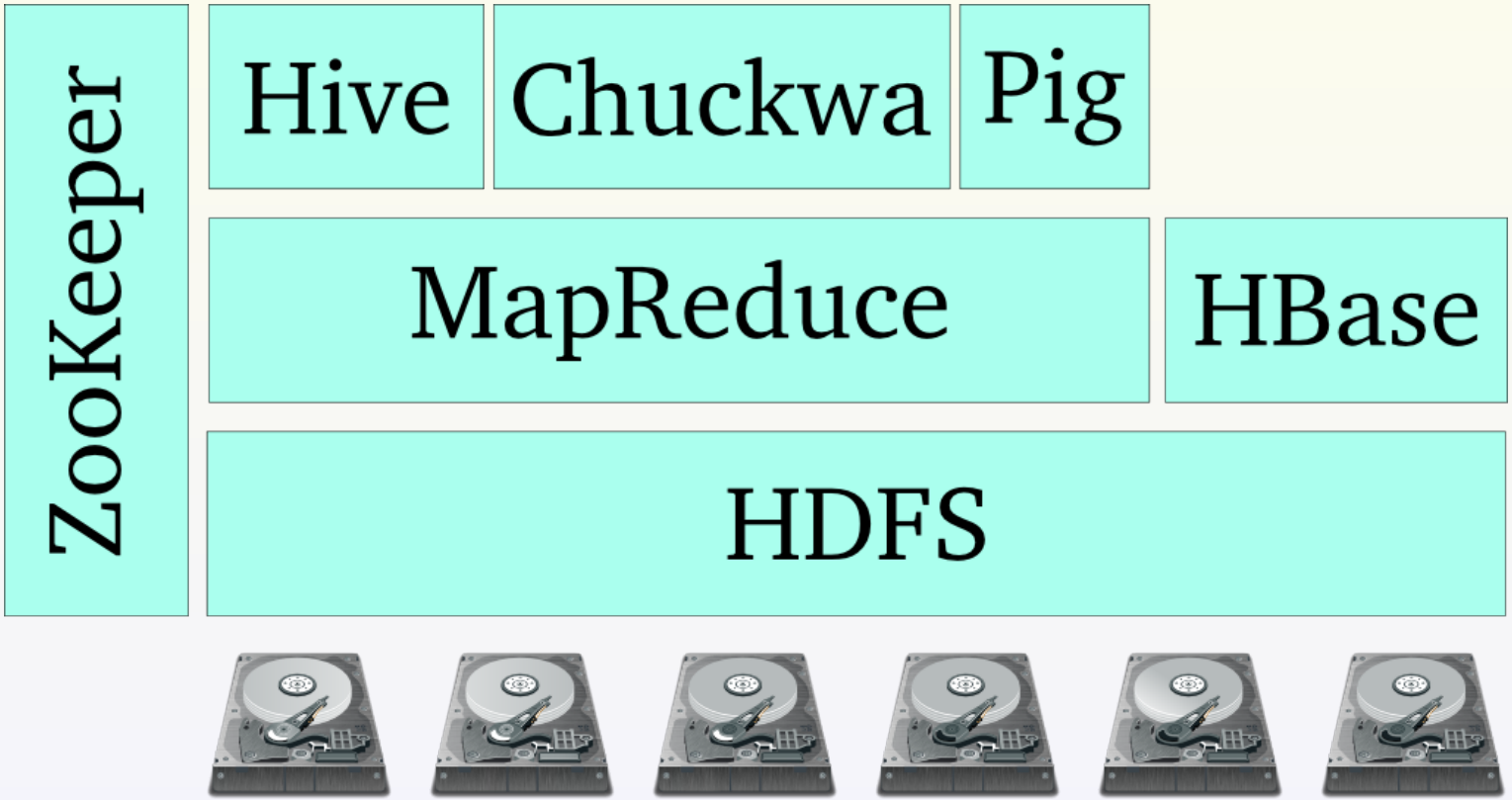
O Hadoop Common oculta o que os usuários comuns não precisam saber!

- Paralelização automática
- Balanceamento de carga
- Otimização nas transferências de disco e rede
- Tratamento de falhas
- Robustez
- Escalabilidade

Outros projetos Apache relacionados

Avro	seriação de dados e chamada a procedimentos remotos (*Remote Procedure Call*)
Cassandra	banco de dados NoSQL, tuplas <chave,valor>
Chukwa	monitoramento e coleta de dados de sistemas distribuídos
HBase	banco de dados não-relacional distribuído e escalável (baseado no Google Bigtable)
Hive	infraestrutura de *data warehouse* (relacional, SQL-like)
Mahout	biblioteca para *machine learning* e *data mining*
Pig	plataforma de análise de dados e linguagem de fluxo de dados (*Pig Latin*)
ZooKeeper	coordenação de serviços distribuídos (configurações, nomes, sincronização, etc.)

A pilha de software do Hadoop



Componentes do Hadoop

Nó Mestre

- NameNode
- SecondaryNameNode
- JobTracker

Nós Escravos

- DataNode
- TaskTracker

NameNode

- Gerencia os metadados dos arquivos
  - FSImage (*checkpointing*) e EditLog (lista das operações)
- Controla a localização das réplicas
- Encaminha os blocos aos nós escravos
- Mantém as informações em memória

DataNode

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

SecondaryNameNode

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

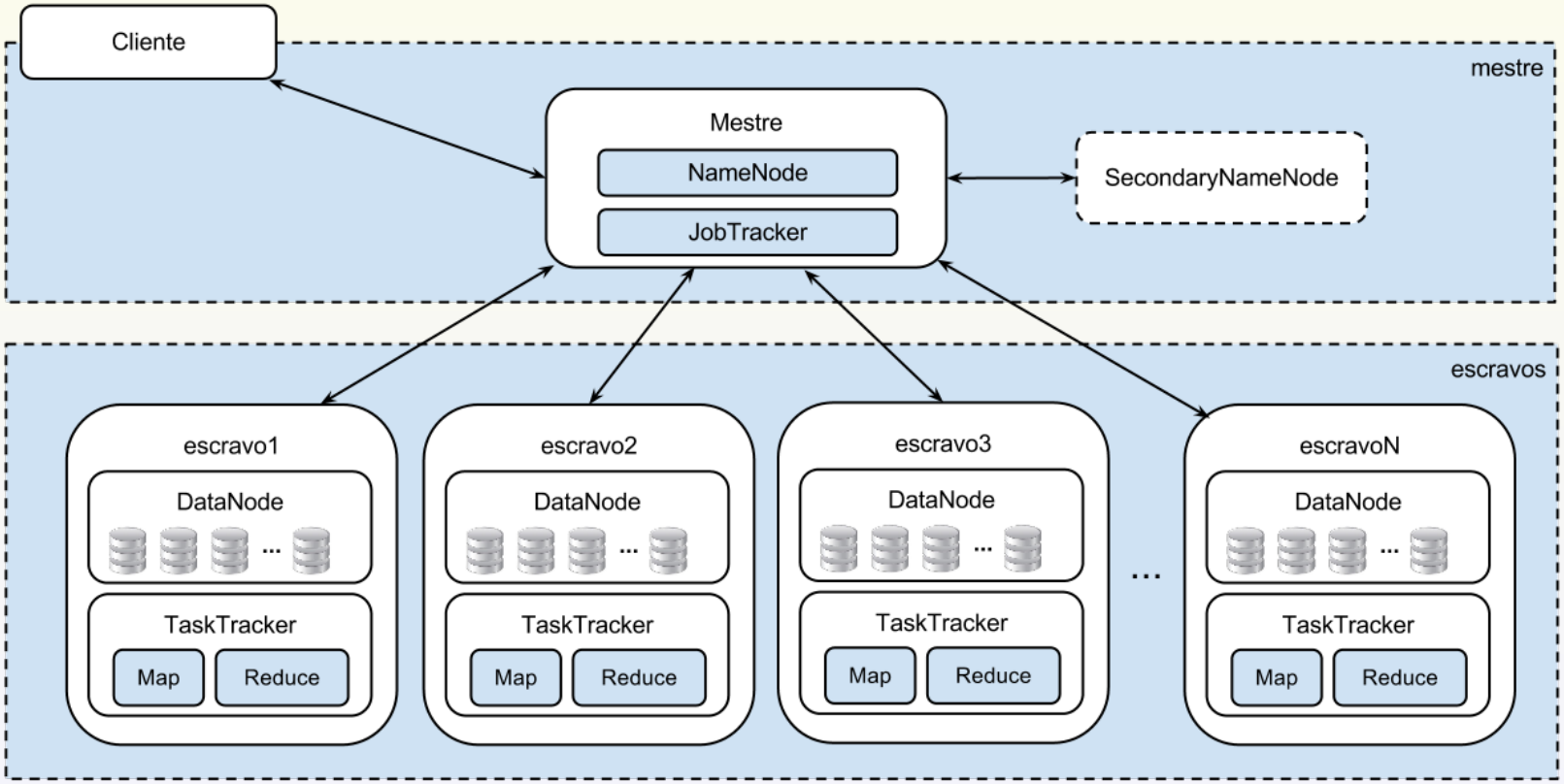
JobTracker

- Gerencia o plano de execução de tarefas MapReduce
- Designa as tarefas aos nós escravos
- Monitora a execução das tarefas para agir em caso de falhas

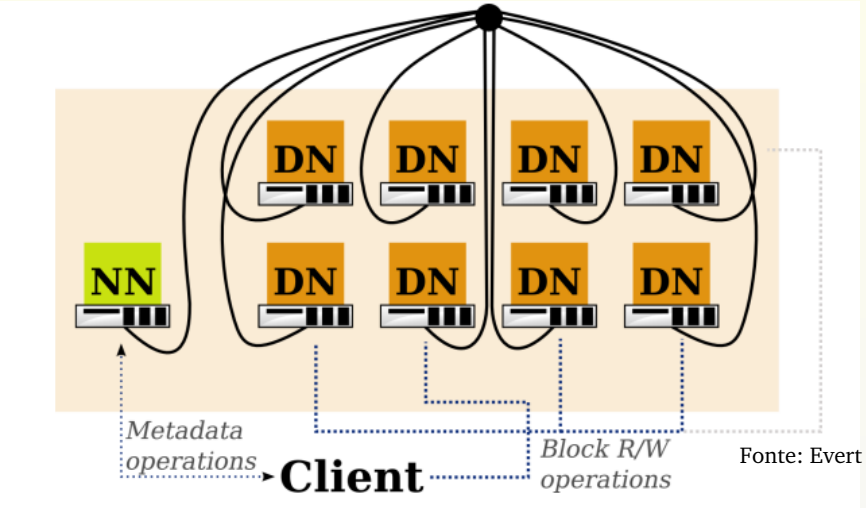
TaskTracker

- Realiza o processamento das tarefas MapReduce
- Cada nó escravo possui uma única instância

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 Lammers (SARA.nl)

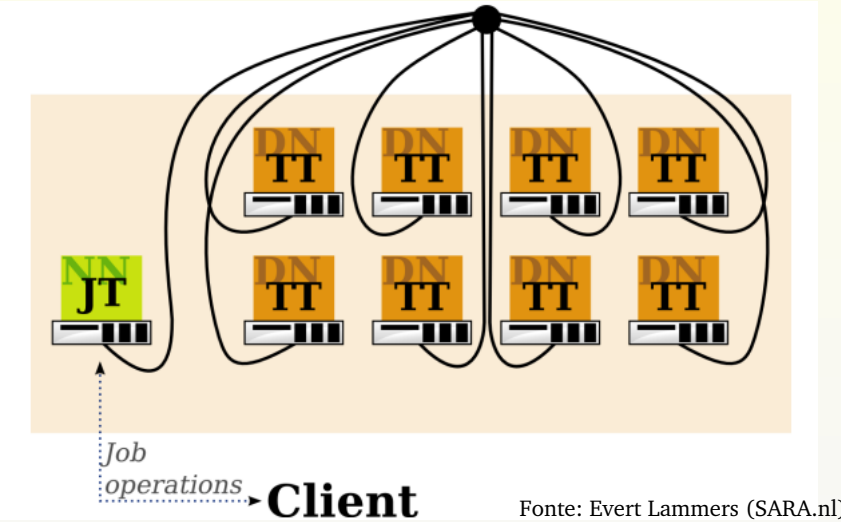
NameNode (NN)

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

DataNode (DN)

- Armazena dados no sistema de arquivos local
- Mantém informações para checar integridade dos blocos (*CRC*)

JobTracker e TaskTrackers no MapReduce



Fonte: Evert Lammers (SARA.nl)

JobTracker (JT)

- Controla os metadados
  - status de um job
  - status de tasks nos TTs
- Decide como será o escalonamento

TaskTrackers (TT)

- Solicita trabalho no JT
  - busca código para executar do DFS
  - aplica configurações específicas nos jobs
- Comunicam-se com o JT nas tasks
  - enviam saídas, sinais, atualizações ...

Formas de execução

- Local
- Pseudo-distribuída
- Completamente distribuída

Formas de execução

Execução local:

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

Formas de execução

Execução pseudo-distribuída:

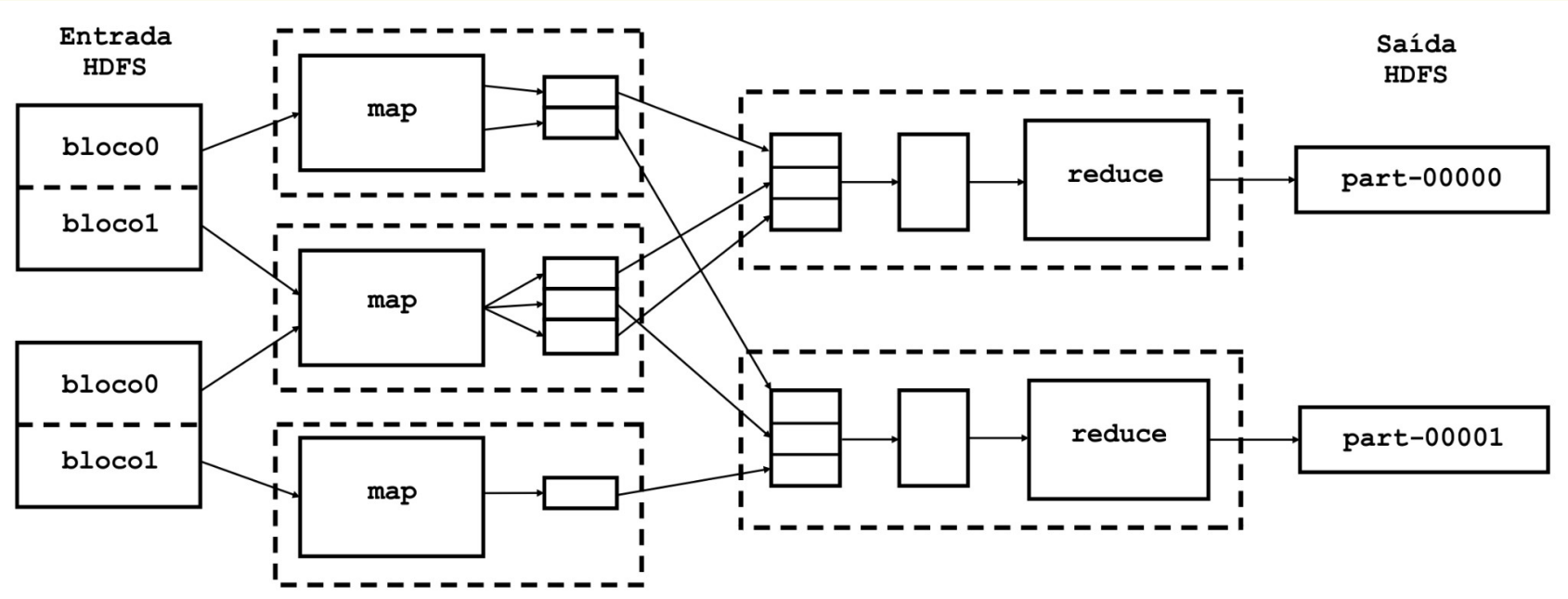
- "Cluster" de uma máquina só
- Configuração similar à do processamento em um cluster...
- ... porém, o processamento continua sendo executado na máquina local

Formas de execução

Execução completamente distribuída:

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

Execução de tarefas MapReduce no Hadoop



Demo

Demo do Word Count

HDFS

Características

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

Características

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

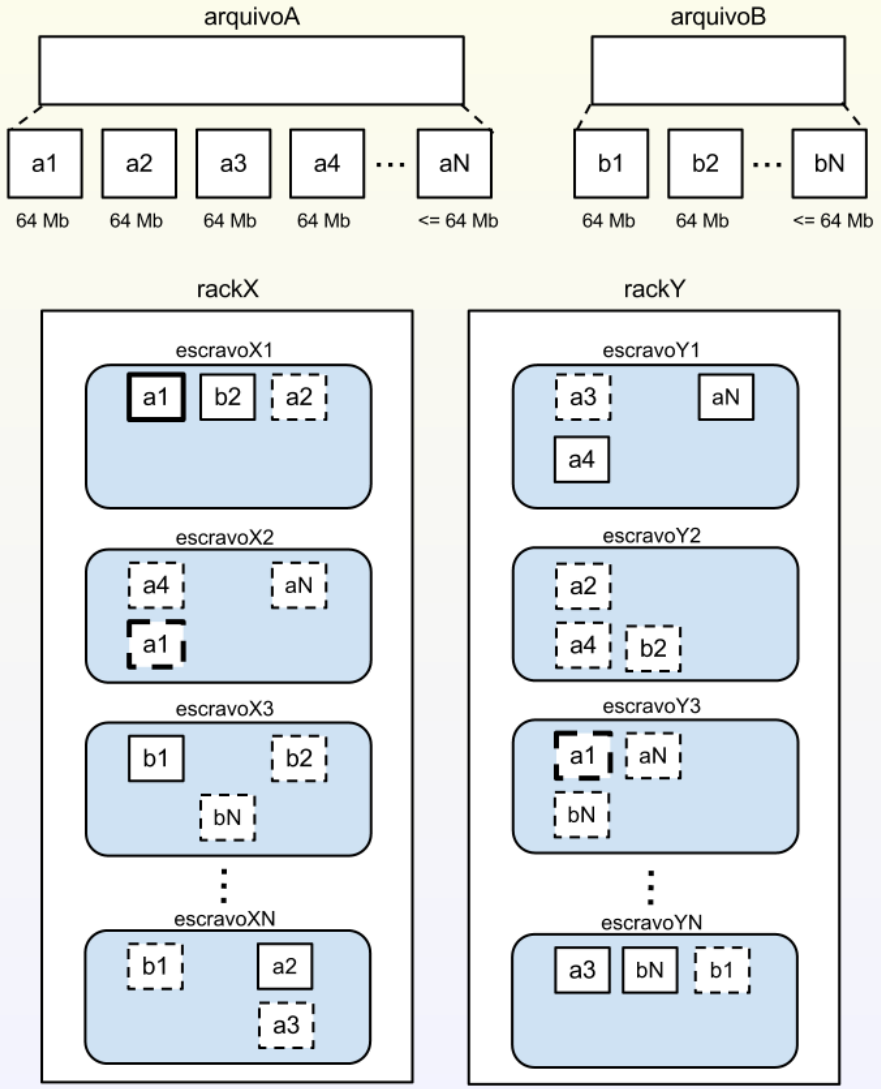
Divisão em blocos

- 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 (em geral) para cada bloco
  - aumento de segurança e disponibilidade
- Cada réplica em um nó diferente
  - 2 em um mesmo *rack* de rede e 1 em um *rack* diferente
- Re-replicação
  - para o caso de uma réplica se tornar corrompida

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

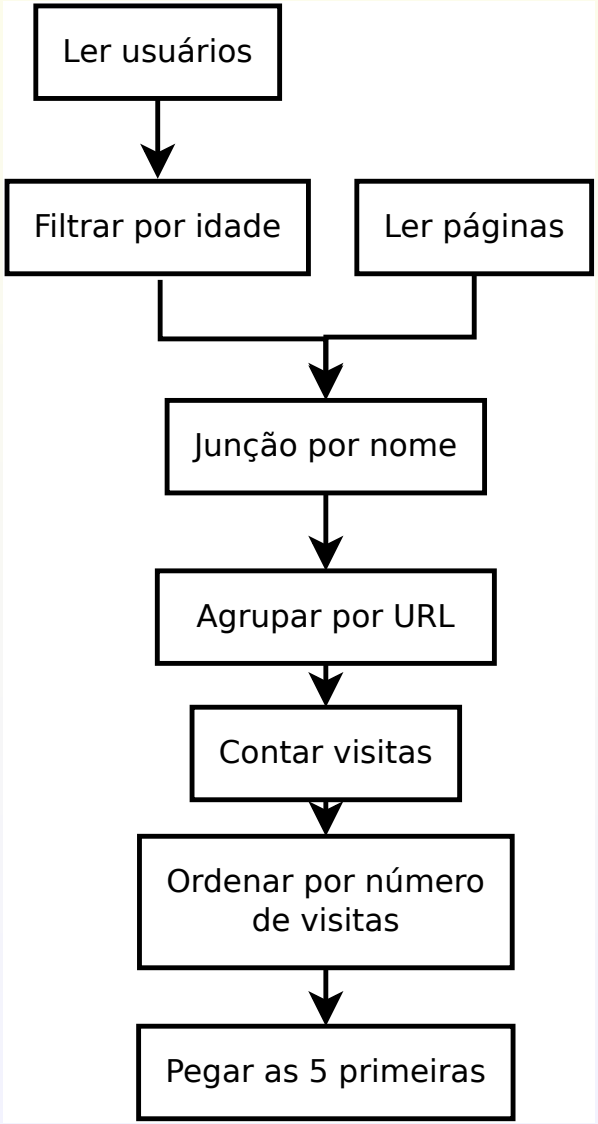
Apache Pig

O que é o Apache Pig?

- O Apache Pig é uma plataforma para a análise de grandes quantidades de dados composta por:
  - uma linguagem de alto-nível para expressar programas de análise de dados (Pig Latin)
  - e uma infra-estrutura para a execução desses programas
- A plataforma gera, otimiza e compila automaticamente programas MapReduce em tempo de execução

Motivação com um exemplo

Problema:
Suponha que você tenha dados dos seus usuários em um arquivo, *logs* de acesso a *sites* em outro, e você quer saber quais são os 5 sites mais visitados por usuários com idades entre 18 e 25 anos.



Código em MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String s = t.toString();
                if (s.charAt(0) == '1')
                    first.add(s.substring(1));
                else second.add(s.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");

                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }

    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
```

Mesmo código em Pig Latin

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
    COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

Execução

Ao executar o script Pig, a plataforma se encarrega de:

- fazer o *parse* do arquivo
- verificar erros de sintaxe
- otimizar o código do script
- criar um plano de execução — quais tarefas Map e Reduce serão necessárias e qual a melhor ordem para executá-las?
- enviar todos os arquivos necessários para o HDFS
- monitorar os processos em execução

Pig vs. Hive

Por que não usar SQL em vez do Pig?

Pig

- *Pipelines*
- Processamento iterativo
- Pesquisa

Hive

- Ferramentas de Business Intelligence
- Análise a posteriori

Destaques do Pig

- Funções definidas pelo usuário (UDFs) são elementos de primeira ordem da linguagem. Podem ser escritos para transformações em colunas (`toUpper()`) ou agregação (`sum()`)
- Quatro tipo de joins diferentes: hash, fragment-replicate, merge e skewed
- *Multi-query*: Pig irá combinar certos tipos de operações em um único *pipeline* para reduzir o número de vezes que um mesmo dado precisa ser analisado
- *Order by* provê ordem total entre os "reducers"
- Piggybank, uma coleção de funções UDF disponibilizadas pela comunidade de usuários

Funções algébricas e de acumulação

Eval functions:

- AVG
- CONCAT
- COUNT
- COUNT_STAR
- DIFF
- IsEmpty
- MAX
- MIN
- SIZE
- SUM
- TOKENIZE

Funções matemáticas

- ABS
- ACOS
- ASIN
- ATAN
- CBRT
- CEIL
- COS
- COSH
- EXP
- FLOOR
- LOG
- LOG10
- RANDOM
- ROUND
- SIN
- SINH
- SQRT
- TAN
- TANH

Quem usa o Pig?

- Em 2010, tarefas MapReduce geradas pelo Pig correspondiam a 70% das tarefas executadas no Yahoo!
- O Pig também é usado pelo Twitter, LinkedIn, Ebay, AOL, etc.
- Usos comuns:
  - Processamento de *logs* de servidores web
  - Construção de modelos de predição de comportamento de usuários
  - Processamento de imagens
  - Construção de índices de páginas da web
  - Pesquisa em conjuntos de dados "brutos"

Leitura dos arquivos

A leitura dos arquivos pode ser feita utilizando:

- a classe PigStorage, que fornece um modo conveniente de ler arquivos com entradas separadas por um delimitador especificado com uma expressão regular, ou
- uma classe Java personalizada

Acessando o Pig

Modos de execução

- **Grunt Shell**: modo iterativo, comandos são digitados manualmente usando um *shell* iterativo
- **Arquivo de script**: os comandos são definidos em um arquivo de *script*
- **Modo embutido**: os comandos do Pig podem ser executados de dentro de um outro programa

Modos de distribuição

- Modo local, as tarefas MapReduce são executadas na máquina local
- Modo Hadoop (MapReduce): a plataforma executa as tarefas MapReduce em uma instalação do Hadoop e do HDFS remota

Um script em PIG simples

Script

```
A = load 'passwd' using PigStorage(':');
B = foreach A generate $0 as id;
dump B;
store B into `id.out';
```

Um script em PIG simples

Modo embutido

```
public class idlocal{
public static void main(String[] args) {
try {
  PigServer pigServ = new PigServer("mapreduce");
  runIdQuery(pigServer, "passwd");
} catch(Exception e) {}
}
public static void runIdQuery(PigServer pigServ, String inputFile)
                                                    throws IOException {
  pigServ.registerQuery("A = load '" + inputFile + "' using PigStorage(':');");
  pigServ.registerQuery("B = foreach A generate $0 as id;");
  pigServ.store("B", "id.out");
 }
}
```

Pig Demo

Pig Demo

Apache Mahout

Apache Mahout

- É uma biblioteca de algoritmos de aprendizado de máquina
- É um projeto da Fundação Apache
- Software livre (licença Apache)
- Principal objetivo: ser escalável para manipular grandes volumes de dados

Onde usar o Mahout?

O Mahout é utilizado quando se é preciso trabalhar com:

- Matrizes e vetores
- Estruturas esparsas e densas
- Agrupamento
- Cobertura
- K-Means
- Análise de densidade de funções
- Filtragem colaborativa

Mahout + Hadoop

Opcional, mas se utilizado com o Hadoop o Mahout pode explorar a escalabilidade do modelo MapReduce para processar os dados

Quem usa o Mahout?

Adobe	Adobe Media Player usa o Mahout para gerar recomendações de vídeos para seus usuários
Amazon	Amazon's Personalization Platform
AOL	recomendações de compras
Foursquare	sistema de recomendações de lugares
Mendeley	sistema de recomendações de artigos científicos
Twitter	modelagem de "interesses" de usuários
etc.	

Gerando recomendações

Exemplo: Large-scale Parallel Collaborative Filtering for the Netflix Prize (AAIM'08 – Zhou et al., HP Labs)

- Constrói uma 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 pelo itens X e Y, então X e Y co-ocorrem 9 vezes
- Co-ocorrência é como similaridade, quanto mais dois itens aparecerem 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 de que o usuário 3 goste.

Demo

Recomendação de filmes usando o conjunto de dados "MovieLens" da Universidade de Minnesota

Entrada: `UserID::MovieID::Rating::Timestamp`

Saída: `UserID [MovieID:Recommendation,...]`
"`6040 [1941:5.0,1904:5.0,2859:5.0,3811:5.0,...]`"

Veja o exemplo completo em:

cwiki.apache.org/MAHOUT/recommendationexamples.html

Referências

Livros

- Hadoop: The Definitive Guide (Tom White, Yahoo Press)
- Hadoop in Action (Chuck Lam, Manning Publications)

Web

- `http://wiki.apache.org/hadoop/`
- `http://developer.yahoo.com/hadoop/tutorial/`
- `http://pig.apache.org/`
- `http://mahout.apache.org/`

Material extra

- Profa. Luciana Arantes (LIP6, Paris)