



# MODELO DO MAPREDUCE E A ARQUITETURA DO HADOOP

---

Fundamentos de Big Data  
Prof. Dr. Julio Anjos

Prova Didática — Abril, 2018

Escopo da aula de hoje:

- Discutir o modelo **MapReduce**;
- Apresentar a arquitetura da plataforma **Hadoop**;
- Introduzir a programação MapReduce;

Objetivos de aprendizagem:

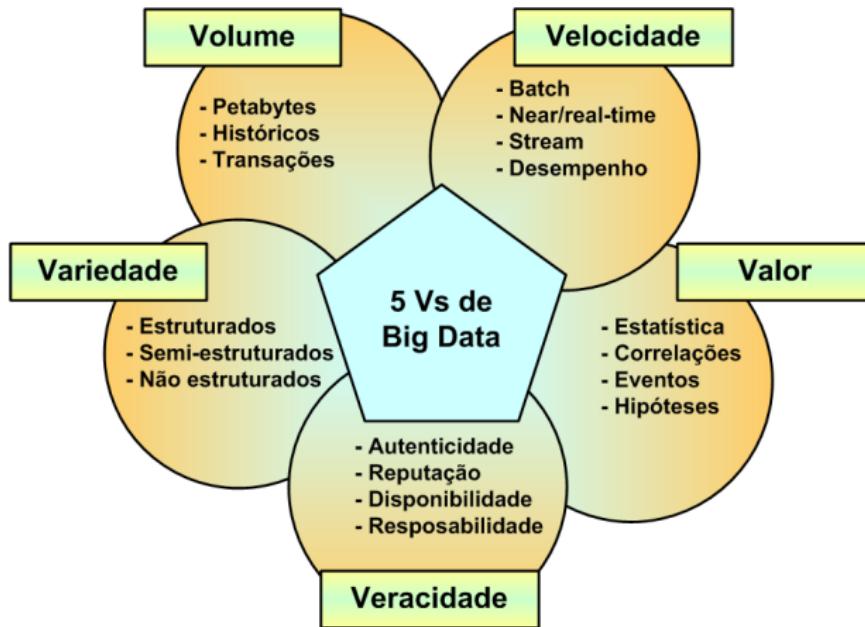
- Compreender o modelo MapReduce e a arquitetura associada;
- Introduzir a programação MapReduce com o Hadoop.

## AGENDA

1. Breve revisão
2. MapReduce
  - Modelo MapReduce
  - Apache Hadoop
3. Onde Aplicar o MapReduce?
  - Aplicações Típicas
  - Exemplos de Aplicações do Hadoop
4. Introdução da Programação MapReduce com Hadoop
  - Entrada e Saída de Dados
  - Programação MapReduce com o Hadoop

1. Breve revisão
2. MapReduce
3. Onde Aplicar o MapReduce?
4. Introdução da Programação MapReduce com Hadoop

## Dimensões de BigData:



[Stonebraker, Madden e Dubey 2013], <<http://www-01.ibm.com/software/data/bigdata/>>

1. Breve revisão
2. MapReduce
3. Onde Aplicar o MapReduce?
4. Introdução da Programação MapReduce com Hadoop

## *MapReduce - Um Framework para Programação Paralela*

- ✓ Abstrai problemas de paralelização, particionamento de dados, sincronização e comunicação.
- ✓ Programador deve definir somente 2 funções **Map** e **Reduce**.
- ✓ Dados são divididos e transformados em *tuplas* (chave, valor).
- ✓ Armazenamento é feito em sistema de arquivos distribuídos.
- ✓ Utilizado em grandes *data centers* como: Google, Amazon, FaceBook, Twitter, Yahoo, IBM, Microsoft, etc.

## MapReduce

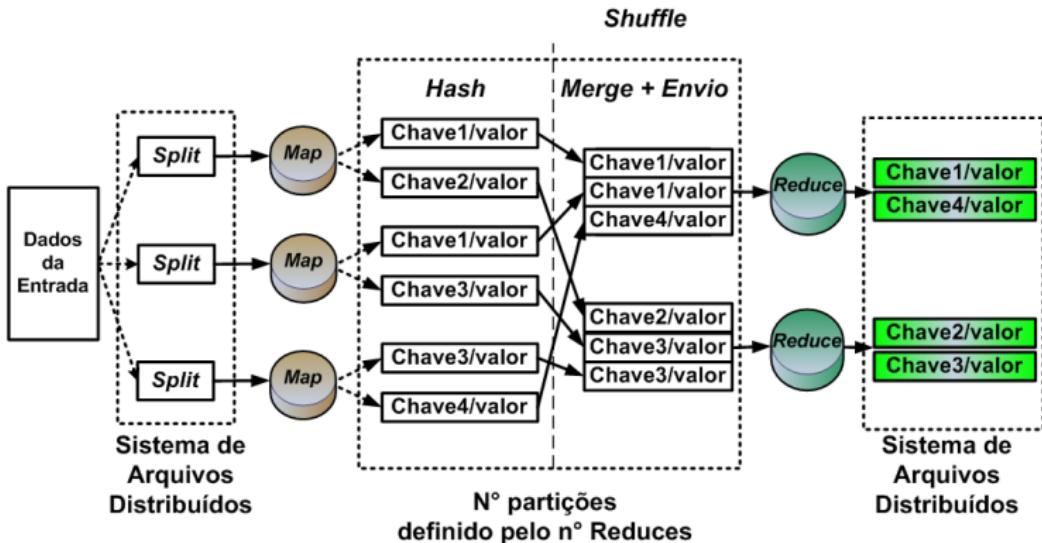
- ✓ Desenvolvido em 2004 pelo Google;
- ✓ Implementação proprietária;

## Apache Hadoop

- ✓ É um Open Source que segue o modelo MapReduce;
- ✓ A Implementação mais próxima do Google;
- ✓ Formado por diversos subprojetos;

[Dean e Ghemawat 2004], [Venner 2009], [White 2012]

# FUNCIONAMENTO DO MAPREDUCE



+ Tratamento de Falhas + Mecanismos Controle/Desempenho

[Dean e Ghemawat 2010], [Lakshman e Malik 2010], [White 2012]

# Apache Hadoop – Arquitetura Base

## Hadoop MapReduce

Processamento de dados distribuídos.

## Hadoop Core

(Utilitários para suportar outros módulos)

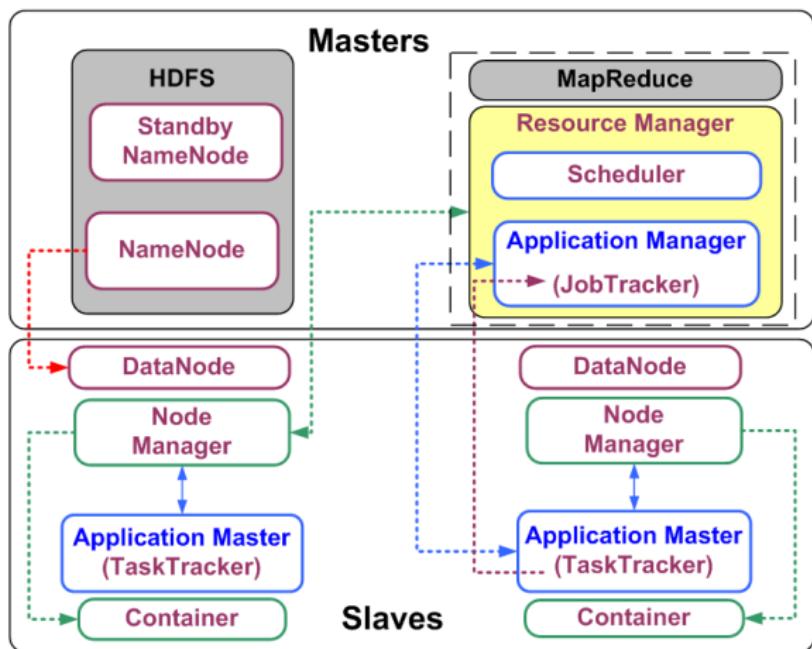
## Hadoop Yarn

(Gerenciamento de *Cluster* e escalonamento *Jobs*)

## Hadoop DFS (HDFS)

Sistema de arquivos distribuídos

## Arquitetura do Hadoop (Yet Another Resource Negotiator)



## Mecanismos para controle e aumento de desempenho

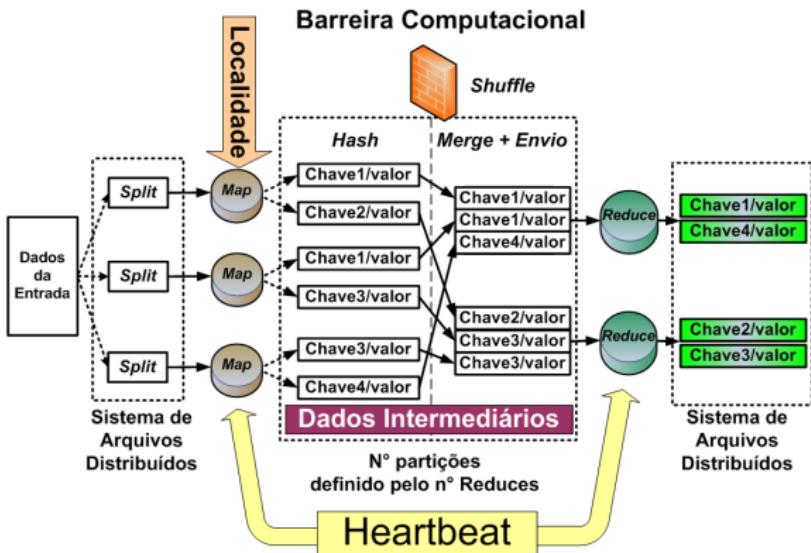
### ✓ Master: Application Manager (JobTracker):

- Gerenciamento de recursos e controle de execução do job;
  - Mecanismo mestre/escravo;
  - **Barreira computacional;**
- Escalonamento de tarefas (observa a **localidade dos dados**);
  - Execução de tarefas como em **ondas**.

### ✓ Slaves: Application Master (TaskTracker):

- Status de execução de tarefas por **Heartbeat**;
- **Tarefa remota** (Transfere dados pelo HDFS);
- **Dados intermediários** armazenados no disco local.

## Mecanismos para controle e aumento de desempenho



## Mecanismos para controle e aumento de desempenho

### ✓ Master: Resource Manager:

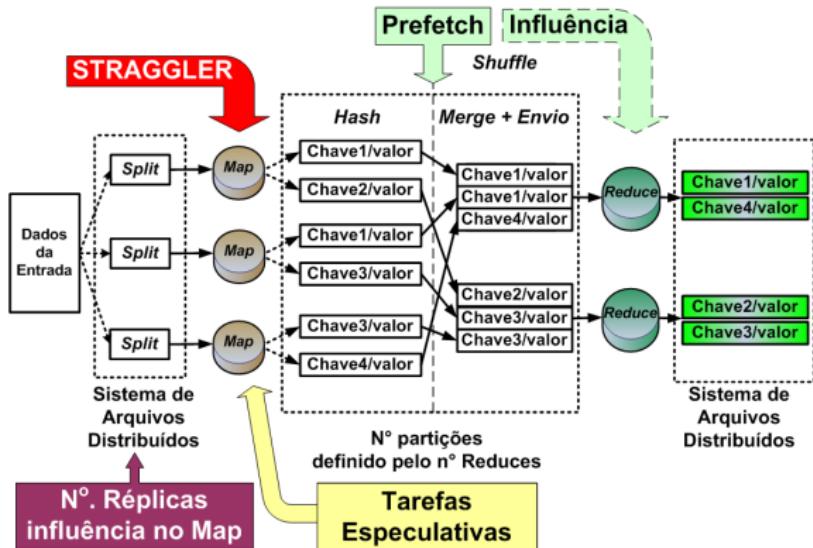
- Controle do progresso das tarefas;
  - Tempo execução > tempo médio de execução → **Straggler**;
  - Lança **tarefas especulativas** (executam após a última onda);
  - Máquina *Straggler* não irá receber mais dados para processar.
- Quando a quantidade de *Maps* concluídos > 5%
  - **Prefetch** de dados intermediários (*Map* → *Reduce*) ;

### ✓ Slaves: Application Master (TaskTracker):

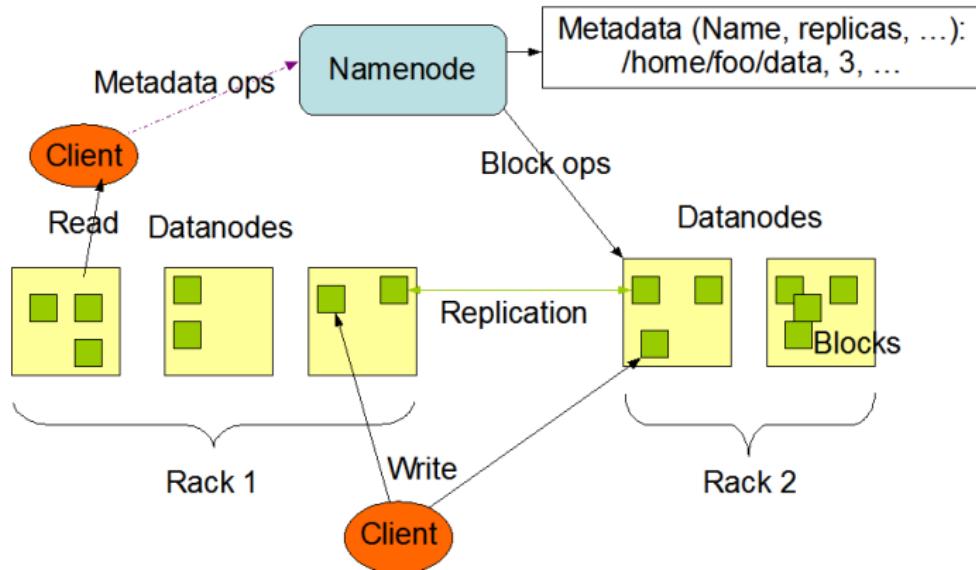
- Slaves trocam dados entre si (**P2P**);

### ✓ NameNode: Controla a quantidade de réplicas de *chunks* no HDFS.

## Mecanismos para controle e aumento de desempenho

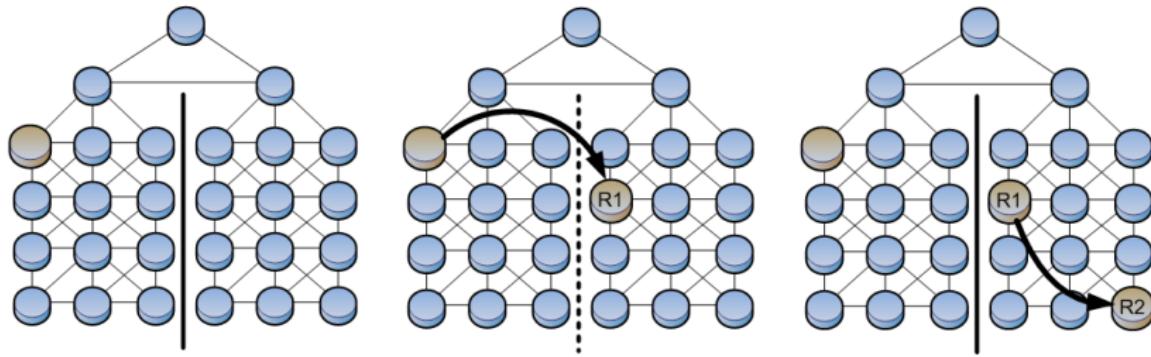


## HDFS - Arquitetura



[White 2012], [Apache 2018]

## HDFS - Funcionamento da replicação de dados no **DataNode**



- ✓ Principais Vantagens:
  - Maior disponibilidade e maior largura de banda de R/W;
- ✓ Custos R/W: no mesmo rack (2), rack diferentes (4) e entre data centers (6). Medida de distância por alg. *k-means*;

1. Breve revisão
2. MapReduce
3. Onde Aplicar o MapReduce?
4. Introdução da Programação MapReduce com Hadoop

## Aplicações do MapReduce

1. Multiplicação de **matriz-matriz** ou **matriz-vetor** (ex.: PageRank)

$$x_i = \sum_{j=1}^n m_{ij} v_j$$

- ✓ Map executa um *chunk* da Matriz  $m$ . Cada elemento  $m_{ij}$  da matriz é multiplicado por um elemento do vetor  $v_j$  que resulta  $(i, m_{ij} v_j)$ ;
- ✓ O Reduce agrupa os termos de  $i$  e produz uma chave  $(i, x_i)$ .

## 2. Operações da álgebra relacional

- ✓ Seleção;
- ✓ Projeção;
- ✓ União, intersecção e diferença;
- ✓ Join;
- ✓ Agrupamento e agregação.

## Exemplos de Aplicações do Hadoop para BigData Analytics

Local	Nome	Entrada de Dados	Aplicação	Objetivos
	Walmart	8 PB /ano	Aprendizagem de Máquina	Estratégias de marketing
	LSST	30 TB /noite	Reconhecimento de Padrões	Compreensão do universo
	LHC	3 PB /mês	Reconhecimento de Padrões	Compreensão do universo

**LSST** = Large Synoptic Survey Telescope

**LHC** = Large Hadron Collider

[Chen e Zhang 2014], [LSST 2016], [CERN 2018]

1. Breve revisão
2. MapReduce
3. Onde Aplicar o MapReduce?
4. Introdução da Programação MapReduce com Hadoop

Considerações sobre os dados a serem analisados:

Entrada:

Um conjunto de textos;

Objetivo:

Saber a quantidade de palavras existentes nestes textos;

Saída:

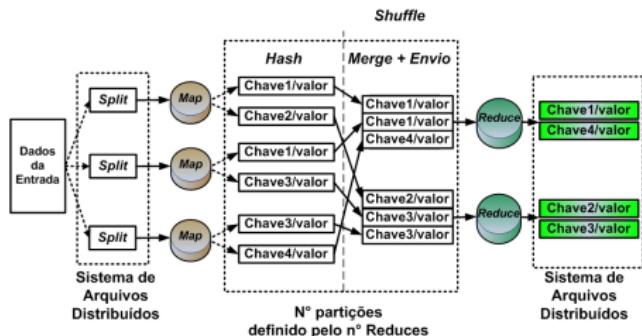
Uma lista de palavras com suas incidências nos textos;

Uso de APP de Programação chamada WordCount.

Exemplo de pseudo-código MapReduce para o Word Count

**map** (line\_number, text):  
**for** each word in word\_list: **do**  
    emit (word, 1)

**reduce** (word, values[]):  
    word\_count = 0  
**for** each v in values: **do**  
        word\_count += v  
    emit (word, word\_count)



Exemplo de pseudo-código MapReduce para o Word Count

**map** (line\_number, text):  
**for** each word in word\_list: **do**  
    emit (word, 1)

**reduce** (word, values[]):  
word\_count = 0  
**for** each v in values: **do**  
    word\_count+ = v  
emit (word, word\_count)

A função *Map* lê cada linha de uma entrada em formato texto e para cada palavra encontrada, emite uma chave intermediária (palavra,1)

Exemplo de pseudo-código MapReduce para o Word Count

**map** (line\_number, text):  
**for** each word in word\_list: **do**  
    **emit** (word, 1)

**reduce** (word, values[]):  
word\_count = 0  
**for** each v in values: **do**  
    word\_count+ = v  
**emit** (word, word\_count)

A função *Map* lê cada linha de uma entrada em formato texto e para cada palavra encontrada, **emite uma chave intermediária (palavra,1)**

Exemplo de pseudo-código MapReduce para o Word Count

**map** (line\_number, text):  
**for** each word in word\_list: **do**  
    emit (word, 1)

**reduce** (word, values[]):  
word\_count = 0  
**for** each v in values: **do**  
    word\_count+ = v  
emit (word, word\_count)

A função *Reduce* “soma” as chaves de mesmo índice e emite nova chave (palavra, valor)

Exemplo de pseudo-código MapReduce para o Word Count

**map** (line\_number, text):  
**for** each word in word\_list: **do**  
    emit (word, 1)

**reduce** (word, values[]):  
word\_count = 0  
**for** each v in values: **do**  
    word\_count+ = v  
emit (word, word\_count)

A função *Reduce* “soma” as chaves de mesmo índice e emite nova chave (palavra, valor)

Operação da álgebra relacional de: **agregação**.

## Modelo de Entrada e Saída da Dados

**map** :  $(k_1, v_1) \rightarrow \text{list}(k_2, v_2) \rightarrow \text{combine} : (k_2, \text{list}(v_2))$   
**reduce** :  $(k_2, \text{list}(v_2)) \rightarrow \text{list}(k_3, v_3)$

## Classe Mapper

Class Mapper<KEY<sub>in</sub>, VALUE<sub>in</sub>, KEY<sub>out</sub>, VALUE<sub>out</sub>>

## Classe Reducer

Class Reducer<KEY<sub>in</sub>, VALUE<sub>in</sub>, KEY<sub>out</sub>, VALUE<sub>out</sub>>

## Pacotes do Hadoop

Mapreduce

Mapred

## Tipos do Hadoop comparados com primitivas Java

Tipo Hadoop	Primitiva Java eq.	Tamanho (Bytes)
BooleanWritable	boolean	1
ByteWritable	byte	1
IntWritable	int	2
VIntWritable		1-5
FloatWritable	float	4
LongWritable	long	8
VLongWritable		1-9
DoubleWritable	double	8
Text	java.lang.String	2 GB

Exemplo: WordCount

## **Programação MapReduce estrutura básica**

```
public class WordCount {  
    public static class Mapper {}  
    public static class Reducer {}  
    public static void main(String[] args) throws Exception {}  
}
```

## Exemplo de Programação MapReduce - Imports

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;
```

## Exemplo de Programação MapReduce - Classe Mapper

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

## Exemplo de Programação MapReduce - Classe Reducer

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

## Exemplo de Programação MapReduce - Definição da main

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

    if (otherArgs.length < 2) {
        System.err.println("Usage: wordcount <in> [<in>...] <out>");
        System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    :
```

## Exemplo de Programação MapReduce - Definição da main

```
:  
job.setCombinerClass(IntSumReducer.class);  
job.setReducerClass(IntSumReducer.class);  
job.setOutputKeyClass(Text.class);  
job.setOutputValueClass(IntWritable.class);  
for (int i = 0; i < otherArgs.length - 1; ++i) {  
    FileInputFormat.addInputPath(job, new Path(otherArgs[i]));  
}  
FileOutputFormat.setOutputPath(job,  
    new Path(otherArgs[otherArgs.length - 1]));  
System.exit(job.waitForCompletion(true) ? 0 : 1);  
}  
}
```

## Descrição da Entrada de Dados

- Suponha que você tenha uma entrada de 9GB cujo arquivo possua a seguinte estrutura.

#	node_name	event_type	event_start_time	event_end_time
0	100416828	1	1211524407.906	1211524417.922
1	100416828	0	1211524417.922	1211524440.312
2	100416828	1	1211524440.312	1211615915.156
3	100416828	0	1211615915.156	1211616057.031
4	100416861	1	1211616057.031	1211837707.998
5	100416861	0	1211837707.998	1211837740.186
6	100416828	1	1211837740.186	1211849805.748
7	100416861	0	1211849805.748	1211867762.312
8	100416861	1	1211867762.312	1212269739.859
9	100416828	0	1212269739.859	1212269772.047

- Cada linha representa o tempo que uma máquina está ligada [1] e desligada [0] em **event\_type**. O nome da máquina é dado por **node\_name**. O *timestamp* (tempo) é dado respectivamente por **start\_time** e **end\_time** de cada evento.

Desejamos saber:

- 1.** Quanto tempo cada máquina fica ligada?
  
- 2.** Se a máquina estiver ativa por 300 dias. Quais máquinas tem um tempo médio maior ou igual a 1 hora por dia? (Informar: nome da máquina, tempo médio, tempo de início e de fim).

1. Quanto tempo cada máquina fica ligada?

**! Usar a estrutura básica do WordCount.**

```
public class WordCount {  
  
    public static class Mapper {}  
  
    public static class Reducer {}  
  
    public static void main(String[] args) throws Exception {}  
}
```

## Definição de imports

```
package org.exec1;

import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

import org.apache.hadoop.util.GenericOptionsParser;
```

## Modelo do Map para chave tipo LongWritable

```
public class Nome extends MapReduceBase implements Mapper<K, V, K, V> {  
    .....  
    public void map(K key, V val, OutputCollector<K, V> output, Reporter reporter)  
        throws IOException {  
        .....  
        output.collect(key, val);  
    }  
}
```

## Código para o Map

```
public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, LongWritable, Text> {

    private LongWritable k = new LongWritable();
    private Text v = new Text();

    public void map(LongWritable key, Text value,
                    OutputCollector<LongWritable, Text> output, Reporter reporter)
            throws IOException {
        String[] tokens = value.toString().split("\\s");
        if (tokens[0].charAt(0) != '#') {
            Long machine = new Long(tokens[1]);
            if (tokens[2].equals("1")) {
                k.set(machine);
                v.set(tokens[3] + ":" + tokens[4]);
                output.collect(k, v);
            }
        }
    }
}
```

## Modelo do Reduce para chave tipo LongWritable

```
public class Nome extends MapReduceBase implements Reducer<K, V, K, V> {  
    .....  
    public void reduce(K key, Iterator<V> values,  
                      OutputCollector<K, V> output, Reporter reporter)  
        throws IOException {  
        .....  
        output.collect(key, val);  
    }  
}
```

## Código para o Reduce

```
public static class Reduce extends MapReduceBase
    implements Reducer<LongWritable, Text, LongWritable, Text> {

    private Text val = new Text();

    public void reduce(LongWritable key, Iterator<Text> values,
                      OutputCollector<LongWritable, Text> output, Reporter reporter)
        throws IOException {
        Long sum = new Long(0);
        Long traceStart = new Long(Long.MAX_VALUE);
        Long traceEnd = new Long(0);
        Long start = new Long(0);
        Long end = new Long(0);
        while (values.hasNext()) { ...
```

## Código para o Reduce

```
while (values.hasNext()) {  
    String line = values.next().toString();  
    String[] tokens = line.split(":");  
    start = new Double(tokens[0]).longValue();  
    end = new Double(tokens[1]).longValue();  
  
    if (start < traceStart) {  
        traceStart = start;  
    }  
    if (end > traceEnd) {  
        traceEnd = end;  
    }  
    sum += (end - start);  
}  
val = new Text (Long.toString(sum));  
output.collect(key, val);  
}
```

## Main

```
public static void main (String[ ] args) throws Exception {  
    JobConf conf = new JobConf(WordCount.class);  
    conf.setJobName("tempocount");  
  
    conf.setNumReduceTasks(Integer.parseInt(args[2]))  
  
    conf.setOutputKeyClass(LongWritable.class);  
    conf.setOutputValueClass(Text.class);  
  
    conf.setMapperClass(Map.class);  
    conf.setReducerClass(Reduce.class);  
  
    conf.setInputFormat(TextInputFormat.class);  
    conf.setOutputFormat(TextOutputFormat.class);  
  
    FileInputFormat.setInputPaths(conf, new Path(args[0]));  
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
  
    JobClient.runJob(conf);  
}
```

Comandos para execução do Job

`./hadoop jar /path/WordCount.jar URL input output 2`

Hadoop normalmente instala em /local/hadoop/bin/, verificar na sua instalação;

Onde:

1. /path = caminho onde está o arquivo;
2. URL = **org.exec1.WordCount**;
3. input = local no HDFS onde estão os arquivos  
verificar pelo comando: `./hadoop dfs -ls`

Cópia do arquivo local para o HDFS

`./hadoop dfs -copyFromLocal /path/filtered_event_trace.txt input`

## Execução do Job

```
Map input records=2879797
Map output records=1441505
Map output bytes=54777190
Map output materialized bytes=57660212
Input split bytes=90
Combine input records=0
Spilled Records=1441505
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=168
Total committed heap usage (bytes)=536346624
File Input Format Counters
    Bytes Read=134221824
18/01/22 14:33:04 INFO mapred.LocalJobRunner: Finishing task: attempt_local2138217572_0001_m_000031_0
18/01/22 14:33:04 INFO mapred.LocalJobRunner: Starting task: attempt_local2138217572_0001_m_000032_0
18/01/22 14:33:04 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
18/01/22 14:33:04 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
18/01/22 14:33:05 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
18/01/22 14:33:05 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
18/01/22 14:33:05 INFO mapred.MapTask: soft limit at 83886080
18/01/22 14:33:05 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
18/01/22 14:33:05 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
18/01/22 14:33:05 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
18/01/22 14:33:05 INFO mapreduce.Job: map 100% reduce 0%
18/01/22 14:33:09 INFO mapred.LocalJobRunner:
18/01/22 14:33:09 INFO mapred.MapTask: Starting flush of map output
18/01/22 14:33:09 INFO mapred.MapTask: Spilling map output
18/01/22 14:33:09 INFO mapred.MapTask: bufstart = 0; bufend = 54840954; bufvoid = 104857600
18/01/22 14:33:09 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 20441668(81766672); length = 5772729/6553600
18/01/22 14:33:10 INFO mapreduce.Job: map 45% reduce 0%
18/01/22 14:33:10 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/root/input:4294967296+134217728 > sort
```

## Tarefa - no Moodle - Entrega próxima aula

2. Se a máquina estiver ativa por 300 dias. Quais máquinas tem um tempo médio maior ou igual a 1 hora por dia? (Informar: nome da máquina, tempo médio, tempo de início e de fim).

### Atividades:

- Baixar e executar o Hadoop versão 2.9 como Single node:  
<https://hadoop.apache.org/docs/>
- Baixar o arquivo com input em:  
[http://www.inf.ufrgs.br/~jcsanjos/BigData/filtered\\_event\\_trace\\_9.tar.gz](http://www.inf.ufrgs.br/~jcsanjos/BigData/filtered_event_trace_9.tar.gz)
- Rodar o exercício 1 e ver os resultados;
- Propor uma solução para o exercício 2, fazer o print dos resultados e execuções e postar no Moodle;

#### Introdução a Sistemas Big Data

##### Introdução a Sistemas Big Data

Aula expositiva e prática sobre Big Data (Modelo MapReduce e arquitetura dos frameworks envolvidos tais como: o Apache Hadoop)

Slides sobre Big Data

Exercícios

Entrega da Atividade

Temas para próxima aula:

- Algoritmos usando MapReduce;
- Extensões do modelo de programação;
- Custo de Comunicação;
- Exemplos avançados de programas MapReduce;

-  APACHE. *Apache Zookeeper*. Apache Software Foundation, jan. 2018. Disponível em: <<http://hadoop.apache.org/versioning.html>>.
-  CERN. *European Organization for Nuclear Research*. mar 2018. Disponível em: <<http://public.web.cern.ch/public>>.
-  CHEN, C. P.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, v. 275, p. 314–347, 2014. ISSN 0020-0255. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0020025514000346>>.
-  DEAN, J.; GHEMAWAT, S. MapReduce - Simplified Data Processing on Large Clusters. In: *OSDI*. [S.I.: s.n.], 2004. p. 137–150.
-  DEAN, J.; GHEMAWAT, S. MapReduce - A Flexible Data Processing Tool. *Communications of the ACM*, ACM, New York, NY, USA, v. 53, n. 1, p. 72–77, 2010. ISSN 0001-0782.
-  LAKSHMAN, A.; MALIK, P. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 35–40, abr. 2010. ISSN 0163-5980. Disponível em: <<http://doi.acm.org/10.1145/1773912.1773922>>.
-  LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. *Mining of Massive Datasets*. 2nd. ed. [S.I.]: Cambridge University Press, 2016. 467 p. ISBN 978-1-107-07723-2.
-  LSST. *Data Management*. oct 2016. Available in <<http://www.lsst.org/about/dm/>> Accessed in October 2016. Disponível em: <<http://www.lsst.org>>.

-  STONEBRAKER, M.; MADDEN, S.; DUBEY, P. Intel "Big Data"Science and Technology Center Vision and Execution Plan. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 42, n. 1, p. 44–49, maio 2013. ISSN 0163-5808. Disponível em: <<http://doi.acm.org/10.1145/2481528.2481537>>.
-  VENNER, J. *Pro Hadoop - Build Scalable, Distributed Applications in the Cloud*. 1st. ed. [S.I.]: Apress, Inc., 2009. ISBN 9781430219422.
-  WHITE, T. *Hadoop - The Definitive Guide*. 3rd. ed. California, USA: O'Reilly Media, Inc., 2012. 688 p. ISBN 978-1-4493-1152-0.