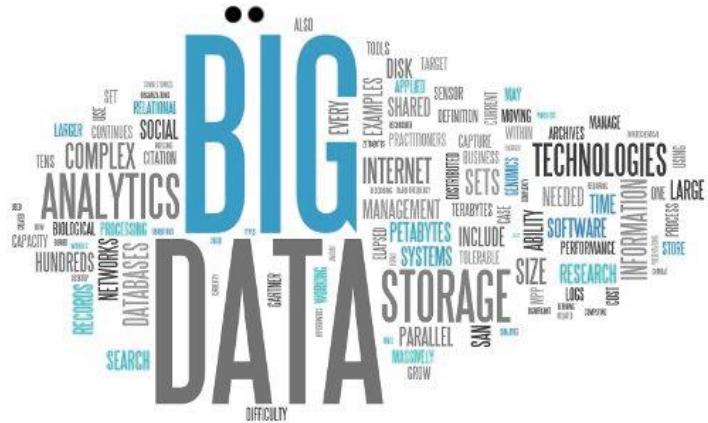


Programa de formación MLDS



Ben Chams - Fotolia

Ben Chams - Fotolia



Módulo BIG DATA Hadoop

Por
Ing. Jorge Camargo, Ph.D.

Agenda



1. Apache Hadoop
2. Motivaciones
3. Arquitectura
4. Sistema de archivos (HDFS)
5. Modelo MapReduce en Hadoop
6. Anatomía de un clúster
7. Programación en Hadoop
8. Casos de estudio

Apache Hadoop



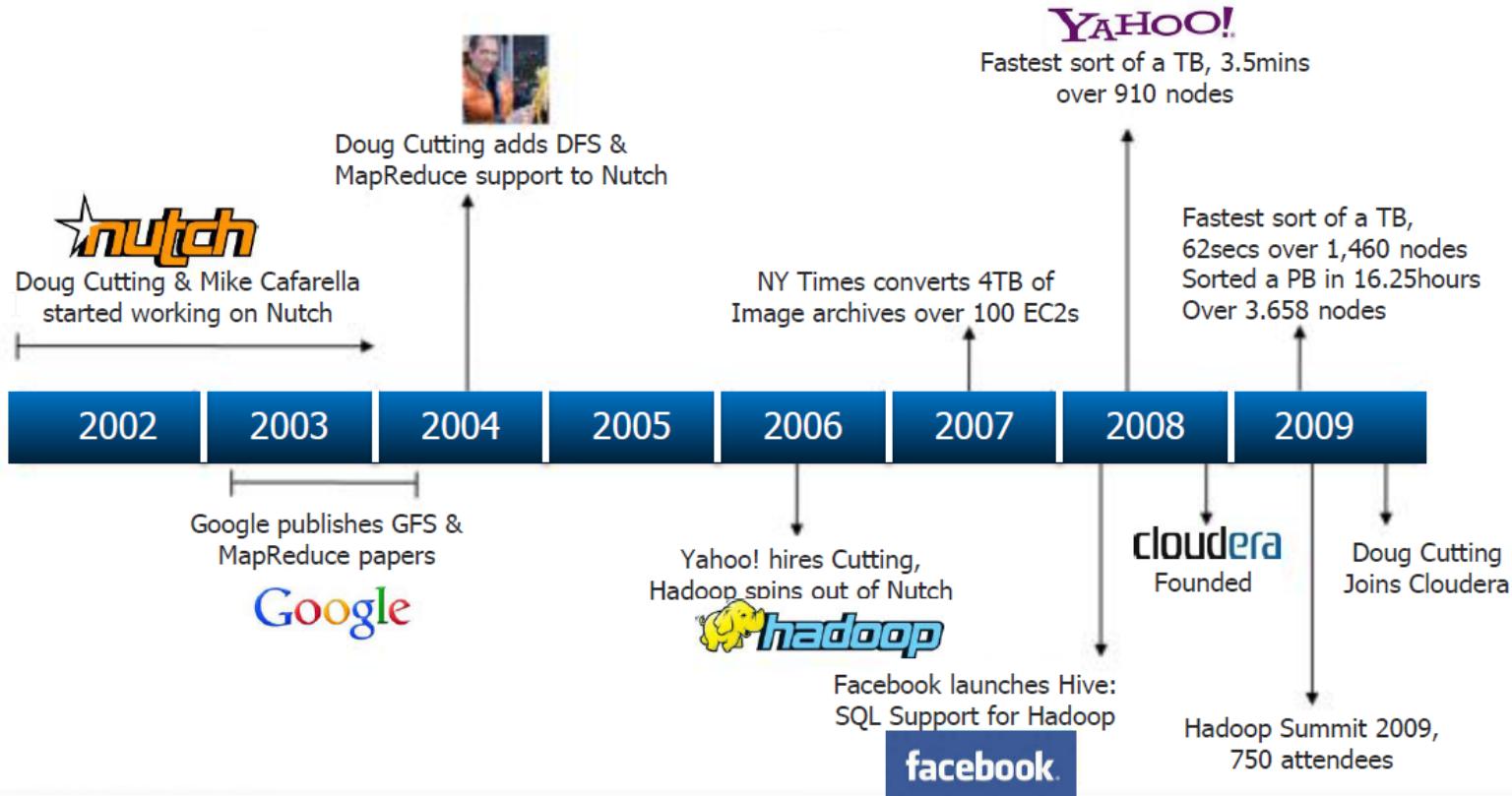
- Framework de código abierto diseñado para almacenar y procesar grandes conjuntos de datos en clúster de máquinas.
- Creado por Doug Cutting y Mike Carafella en 2005
- El nombre proviene del nombre del elefante (juguete) de un hijo Cutting.

Origen de Apache Hadoop



- 2004: Google publica el artículo de GFS
- 2005: Nutch (motor de búsqueda de código abierto) usa el modelo MapReduce
- 2008: Hadoop se convierte en el proyecto top de Apache, seguido por Lucene
- 2009: Yahoo usó Hadoop para ordenar 1TB de datos en 62 segundos
- 2013: Hadoop es utilizado por cientos de compañías

Historia de Apache Hadoop



<http://oracle4ryou.blogspot.com.co/2014/09/hadoop-history.html>

Usos de Hadoop



- Procesamiento de texto
- Procesamiento de grandes cantidades de datos del genoma humano
- Minería de grafos
- Machine learning y minería de datos
- Análisis de redes sociales a gran escala

¿Quiénes lo utilizan?



UNIVERSIDAD
NACIONAL
DE COLOMBIA



eHarmony®



facebook

IBM

twitter

amazon.com®

SAMSUNG

The New York Times

JPMorganChase

intel

NETFLIX

VISA

YAHOO!®

hadoop

Componentes de Hadoop



Hadoop Common

- Contiene librerías y otros módulos

HDFS

- Sistema de archivos distribuido

Hadoop YARN

- Negociador de recursos

Hadoop MapReduce

- Modelo de programación para procesamiento de datos a gran escala

MOTIVACIONES

Motivaciones para Hadoop



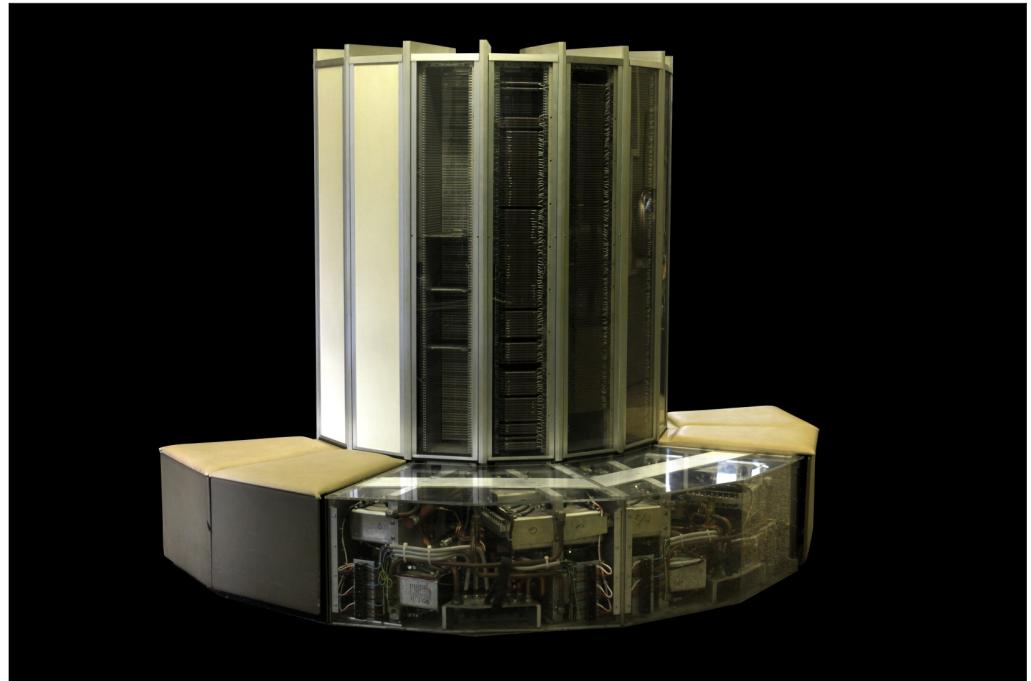
- Limitaciones de modelos de computación a gran escala previos
- El paradigma de *grid computing*
- El paradigma de la computación distribuida
- El paradigma de la computación paralela

Paradigmas previos de computación a gran escala

- Históricamente la computación estuvo centrada en el procesador
 - Volumen de datos relativamente pequeño
 - Cálculos complicados son realizados en estos datos
- Los avances en la tecnología de la computación se centraba en mejorar el poder de una única máquina

Cray-1

- USD \$8MM + \$1MM para discos
- 82 unidades vendidas
- Descontinuado en 1982
- Memoria: 1MM de palabras en memoria principal
- Procesador: 160 Mflops, 64 bits
- Peso: 5.5 toneladas



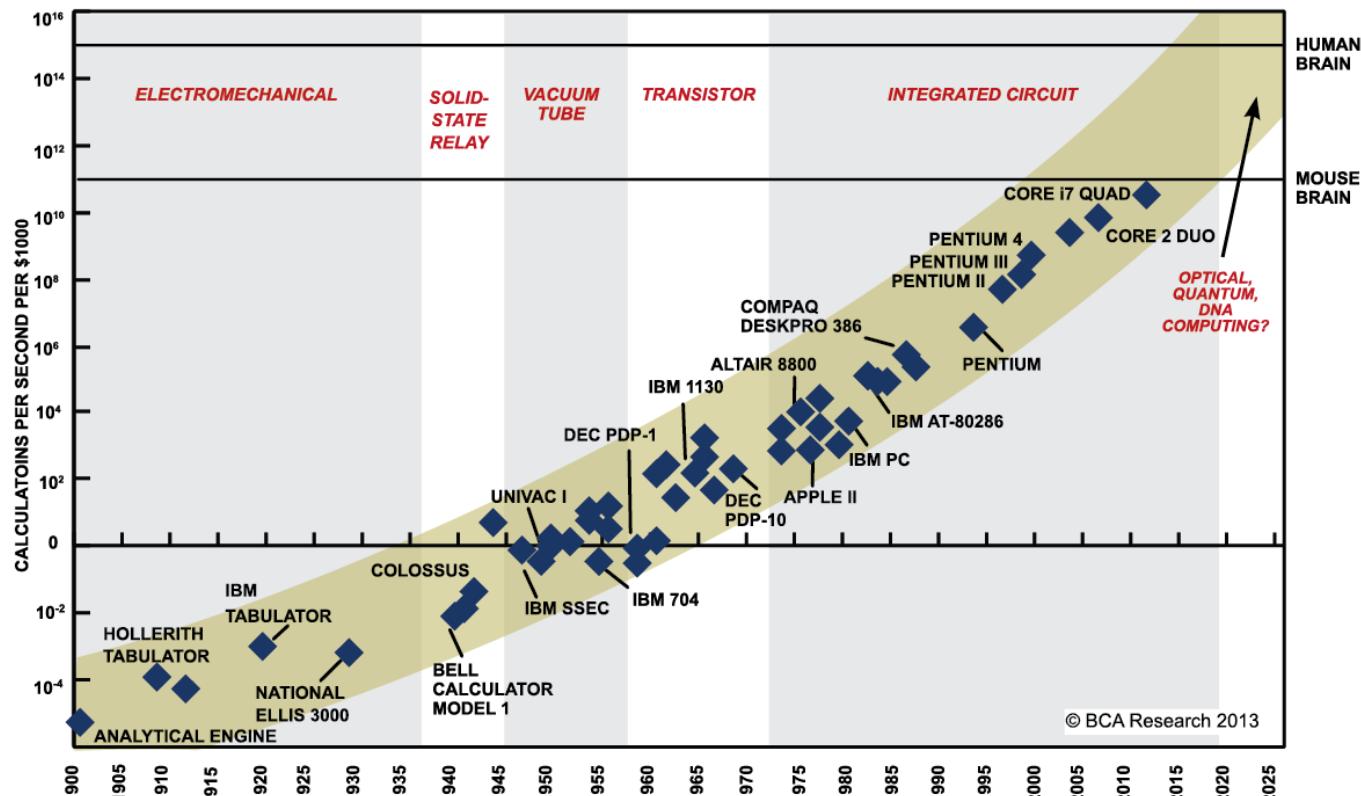
Avances en las CPUs



- Ley de Moore
 - La cantidad de transistores en un circuito integrado se duplica cada dos años
- Computación de un solo *core (single-core)* no puede escalar con las necesidades de cómputo requeridas

Limitaciones de *single-core*

- El consumo de energía limita la velocidad de crecimiento a causa de la densidad de transistores



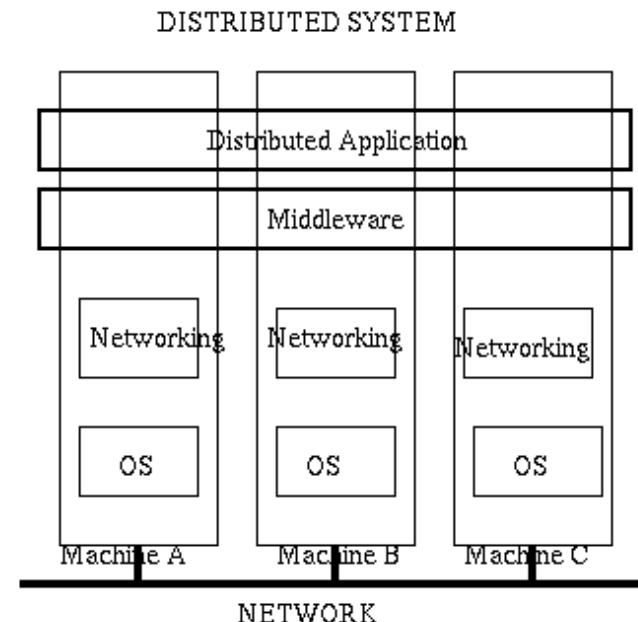
Sistemas Distribuidos

- Permite a los desarrolladores utilizar múltiples máquinas para una única tarea



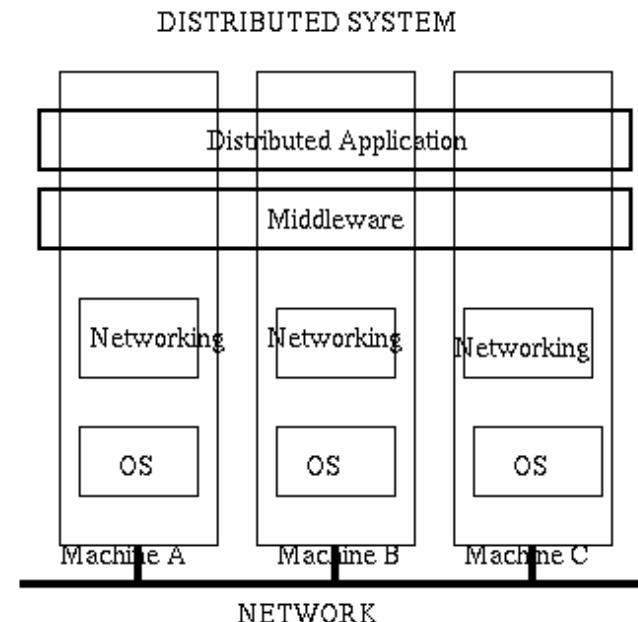
Sistemas distribuidos: problemas

- La programación en un sistema distribuido es muy compleja
 - Sincronización en el intercambio de datos
 - Administración de ancho de banda finito
 - El control del tiempo de cómputo es complicado



Sistemas distribuidos: almacenamiento de datos

- Típicamente dividido en *Data Nodes* y *Compute Nodes*
- En tiempo de ejecución, los datos son copiados a los *Compute Nodes*
- Funciona bien para cantidades de datos no muy grandes



Cúantos datos?

- Facebook
 - 500TB por día
- Yahoo
 - 170TB por día
- eBay
 - Cerca de 6 PB por día
- Llevar los datos a los procesadores se convierte en un cuello de botella

Requerimientos para Hadoop

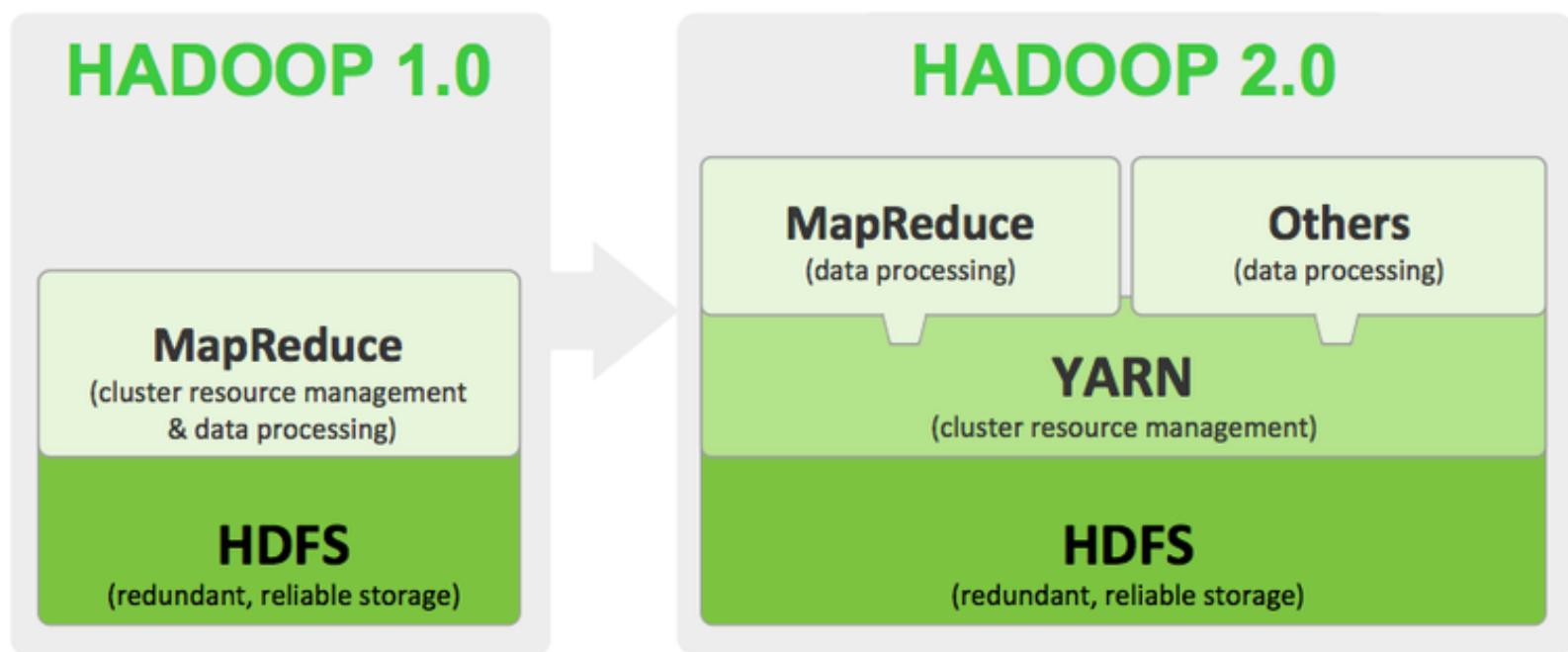
- Debe soportar fallas parciales
 - La falla en un componente no debe causar la falla del sistema entero (solamente degradación en el desempeño de la aplicación)
 - La falla no debería causar pérdida de datos
 - Recuperación de componentes
- Debe ser escalable
 - Al incrementar los recursos se debería incrementar la capacidad de carga del sistema

Hadoop vs RDMS

| | Traditional RDMS | Hadoop MapReduce |
|------------------|---------------------------|--|
| Data size | Gigabytes | Petabytes |
| Access | Interactive | Batch |
| Updates | Read and Write many times | Write once, Read many times |
| Structure | Static schema | Dynamic schema |
| Integrity | High | Low in simple setup, can be improved with additional servers |
| Scaling | Non-linear | Linear(up to 10,000 machines as of Dec 2012) |

ARQUITECTURA DE HADOOP

Evolución de Hadoop



Evolución de Hadoop

| Attributes | Hadoop 2.x | Hadoop 3.x |
|-----------------------------|----------------------------|---|
| Fault-Tolerance | Through replication | Through erasure coding |
| Storage | Consumes 200% in HDFS | Only 50% consumption in HDFS |
| Scalability | Limited up to 10.000 nodes | Over 10.000 in a cluster |
| File system | DFS, FTP and Amazon S3 | All features plus Microsoft Azure Data Lake File System |
| Manual intervention | Not needed | Not needed |
| Cluster resource management | YARN | YARN |
| Data balancing | Uses HDFS balancer | Uses Intra-data balancer |

HDFS

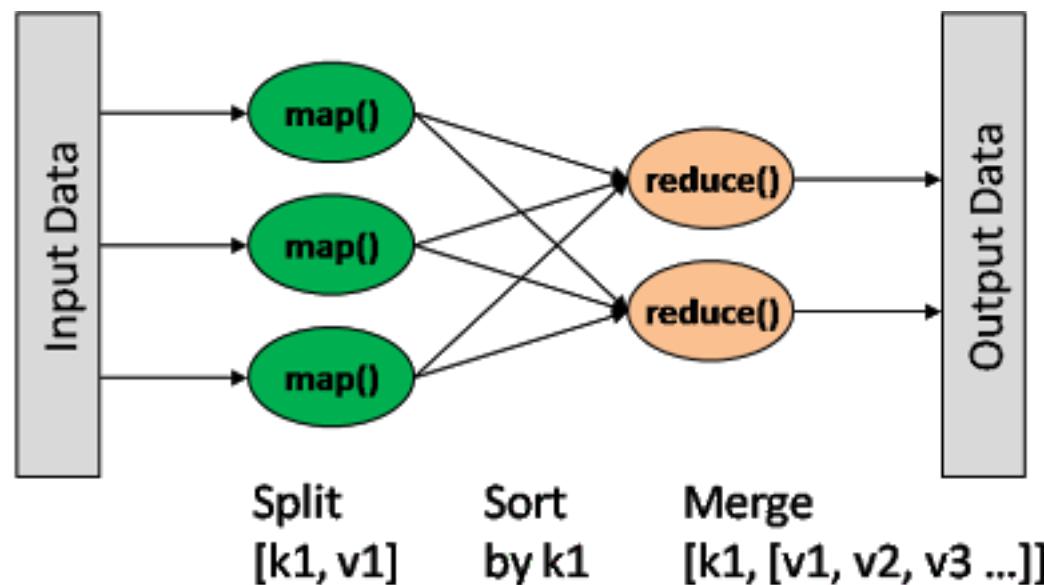
- Sistema de archivos distribuido
- Escalabilidad horizontal
- Servicio de *namespaces*
 - Operaciones en archivos y directorios
- Servicio de almacenamiento de bloques
 - Gestión de clúster, operaciones sobre bloques y replicación

Yarn

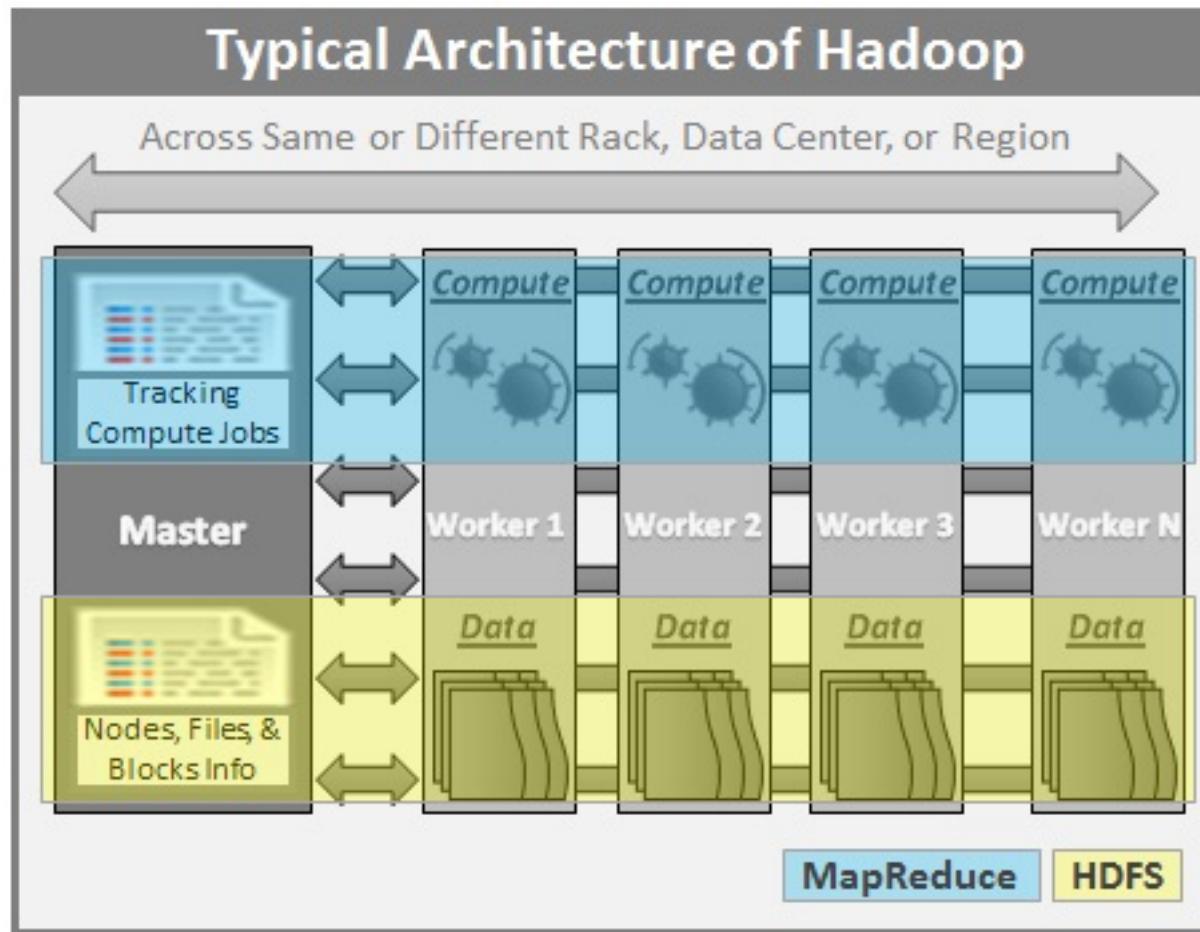
- Gestor de recursos
- Fue creado para separar las responsabilidades del motor de procesamiento y el gestor de recursos
- Se le conoce como el sistema operativo de Hadoop: monitoreo de carga, controles de seguridad, gestor multi-usuarios, administración de alta disponibilidad
- Soporta múltiples modelos de procesamiento (incluido MapReduce)

MapReduce

- Programación por lotes de datos
- Modelo de programación distribuida



Arquitectura general



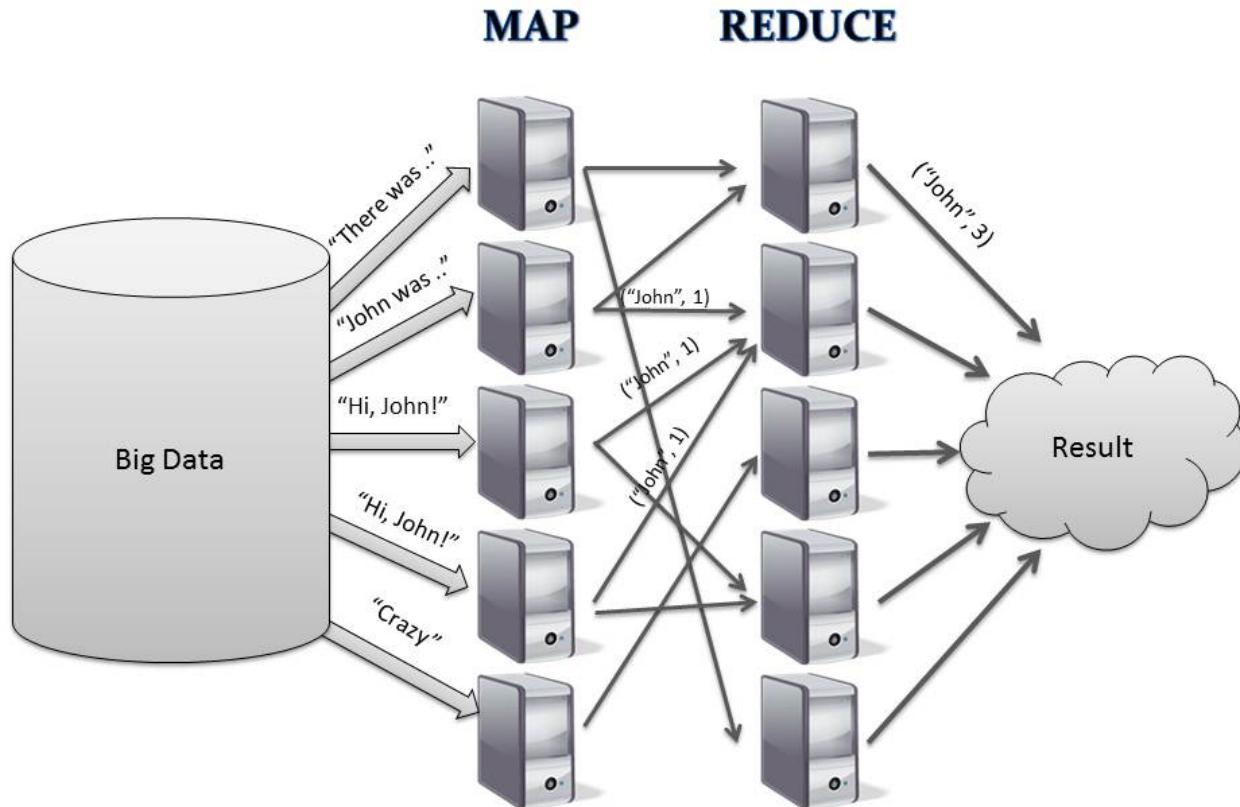
Conceptos en Hadoop

- Las aplicaciones son escritas en un lenguaje de programación de alto nivel (no se necesita programación a nivel de red)
- Los nodos deberían comunicarse entre ellos lo menor posible (*shared nothing architecture*)
- Los datos son ubicados previamente entre las máquinas (cálculo sobre datos ya almacenados)

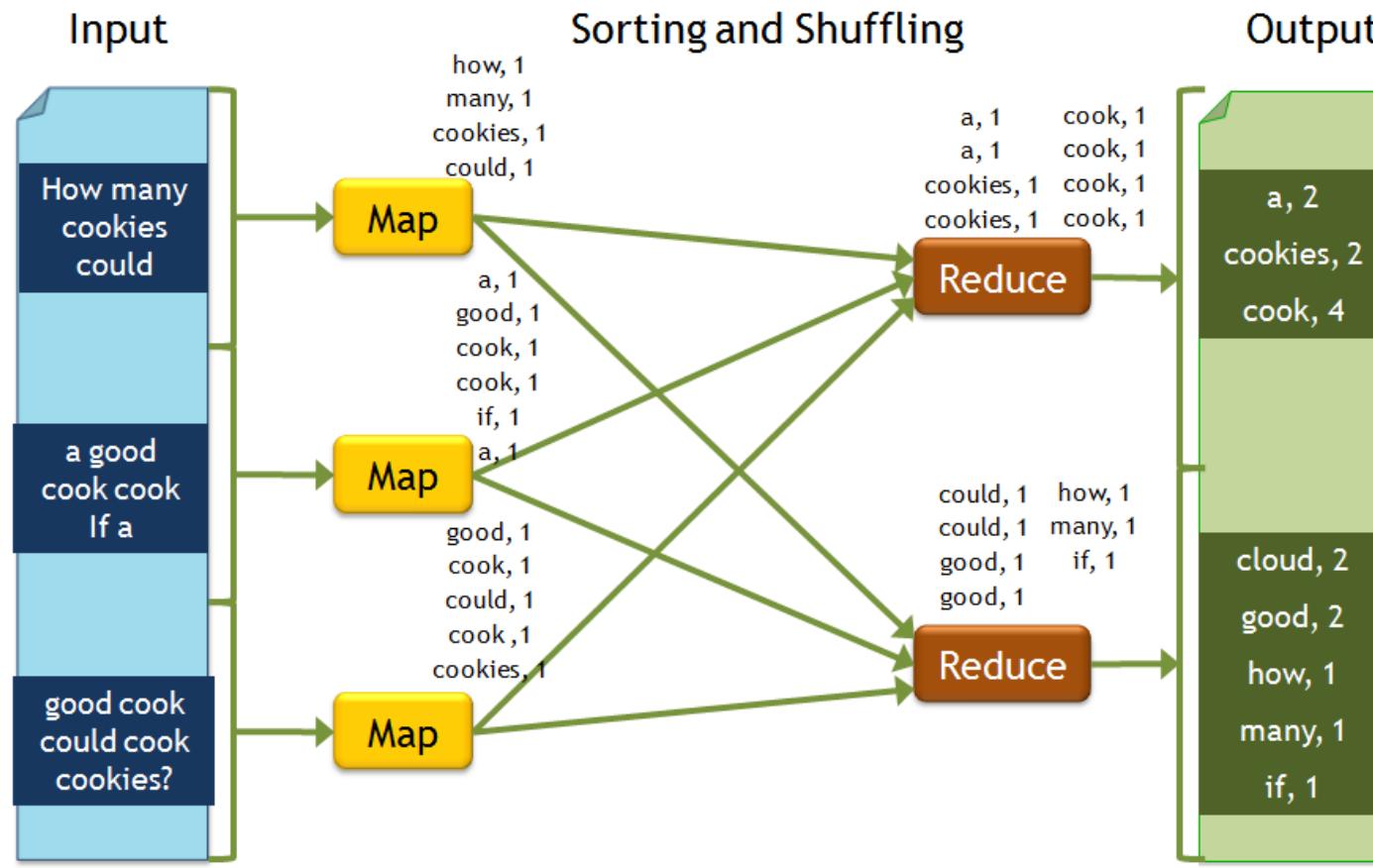
Procesamiento en Hadoop

- Cuando los datos están cargados en el sistema, éstos son divididos en bloques
 - Típicamente de 64MB o 128MB
- Las tareas se dividen en dos fases
 - Tareas *Map* que son realizadas en pequeñas porciones de datos
 - Tareas *Reduce* que combinan datos producidos en la salida final
- Un programa master asigna trabajo a nodos individuales

Procesamiento en Hadoop



Procesamiento en Hadoop: ejemplo



By Manaranjan Pradhan

HDFS

Generalidades de HDFS

- Responsable de almacenar los datos en el clúster
- Archivos de datos son divididos en bloques y distribuidos a través de los nodos del clúster
- Cada bloque es replicado múltiples veces

Conceptos Básicos de HDFS

- HDFS trabaja mejor con un pequeño número de archivos grandes
 - Millones de archivos
 - Típicamente 100MB o más por cada archivo
- Los archivos en HDFS se escriben una sola vez
- Optimizados para lecturas en *streaming* de archivos grandes (no lecturas aleatorias)

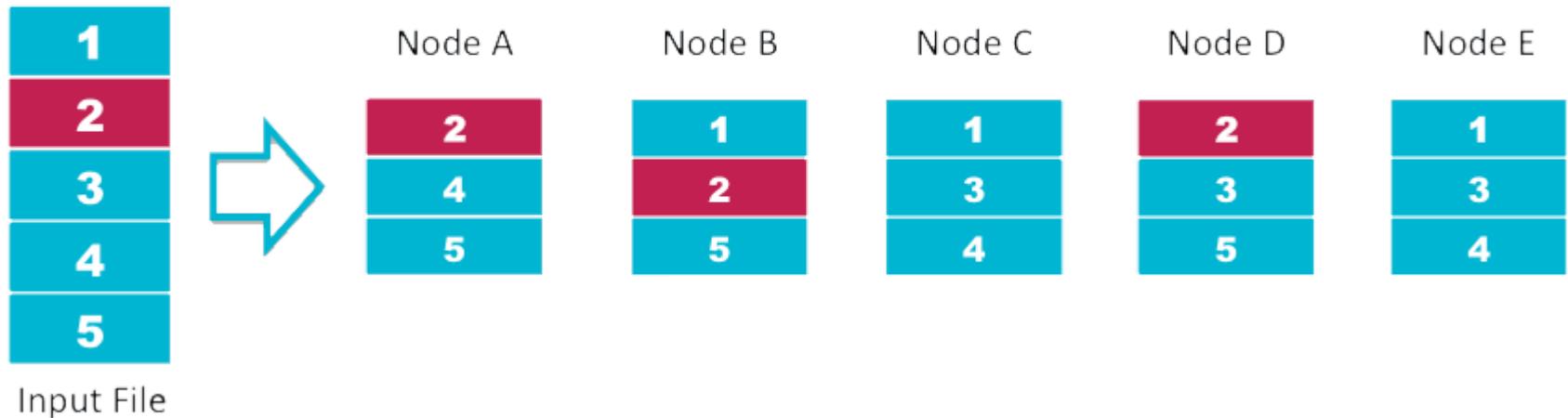
¿Cómo se almacenan los archivos en HDFS?



- Los archivos se parten en bloques
- Los bloques son distribuidos en muchas máquinas en tiempo de carga
 - Diferentes bloques de un mismo archivo serán almacenados en diferentes máquinas
- Los bloques se replican a través de múltiples máquinas
- El *NameNode* mantiene la traza de cuáles bloques componen un archivo y en dónde están almacenados

¿Replicación de datos en HDFS?

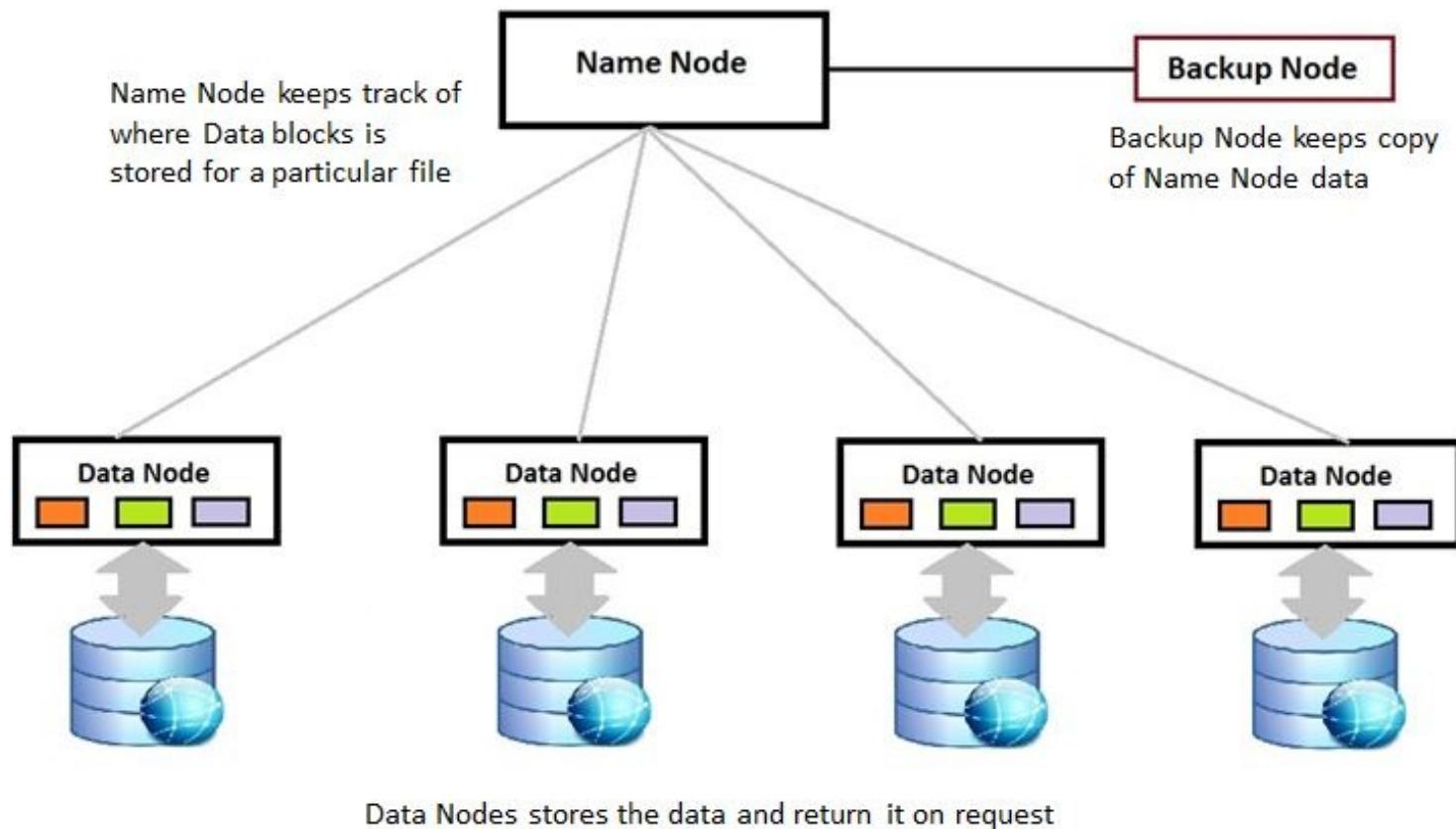
HDFS Data Distribution



Recuperación de datos en HDFS

- Cuando un cliente desea recuperar datos
 - Se comunica con el *NameNode* para determinar cuáles bloques componen un archivo y en qué nodos de datos (*DataNodes*) estos bloques están almacenados
 - Una vez comunicado directamente, lee los datos desde el *DataNode*

DataNodes y NameNodes en HDFS



MODELO MAPREDUCE EN HADOOP

Generalidades de MapReduce

- Método para distribuir cómputo a través de múltiples nodos
- Cada nodo procesa los datos que tiene almacenados
- Consiste en dos fases
 - *Map*
 - *Reduce*

Generalidades de MapReduce

- Paralelización y distribución automática
- Tolerancia a fallos
- Provee una abstracción para los programadores

El Mapper

- Lee datos en forma de pares clave/valor
 - Frecuentemente la clave se descarta
- Genera salidas de cero o más pares clave/valor

Shuffle and Sort

- La salida de cada *Mapper* es ordenada por clave
- Todos los valores con la misma clave van a una misma máquina

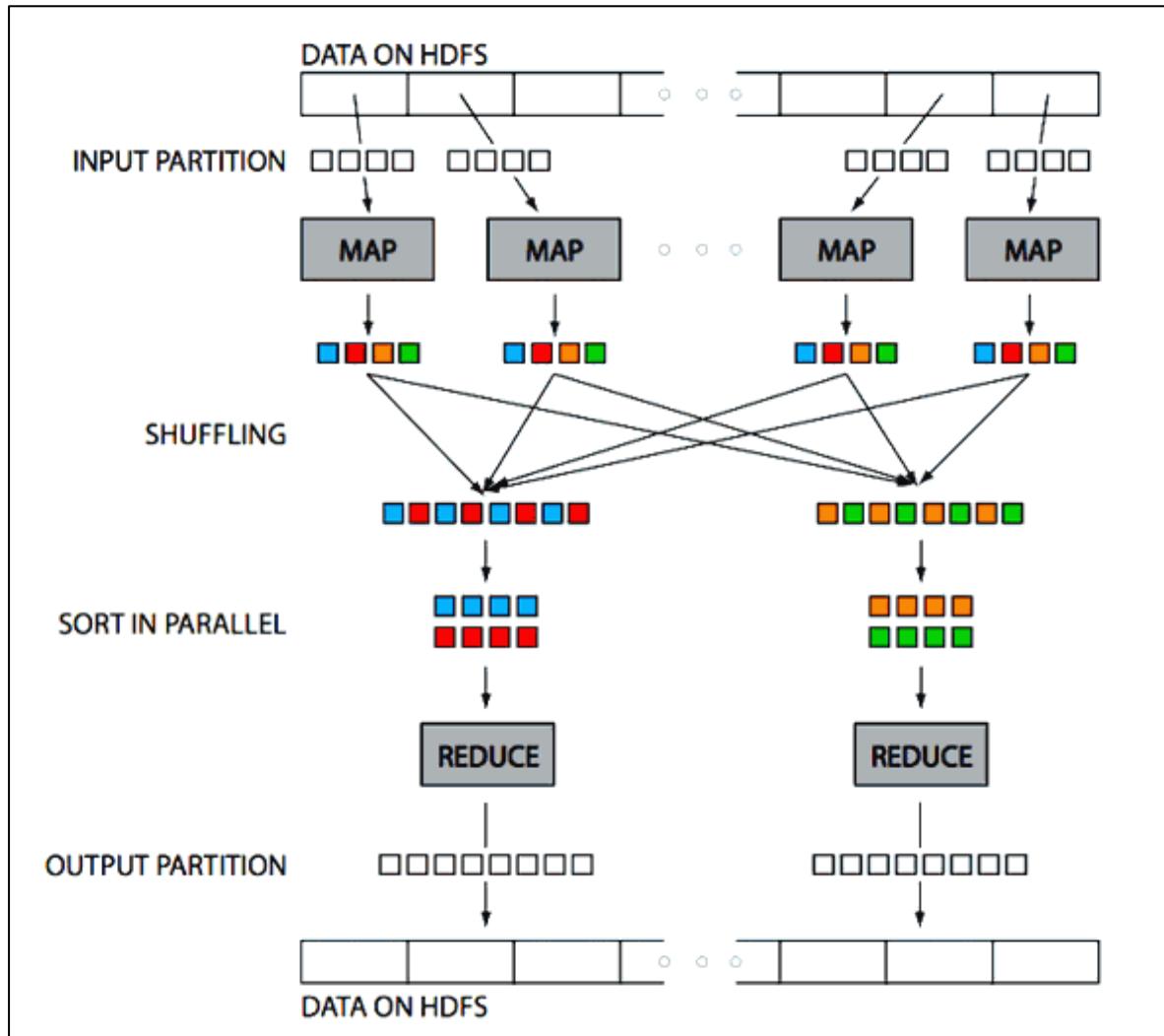
El Reducer

- Llamado una vez para cada clave única
- Obtiene una lista de todos los valores asociados a una llave como entrada
- Produce como salida cero o más pares clave/valor
 - Usualmente solo una salida por cada clave

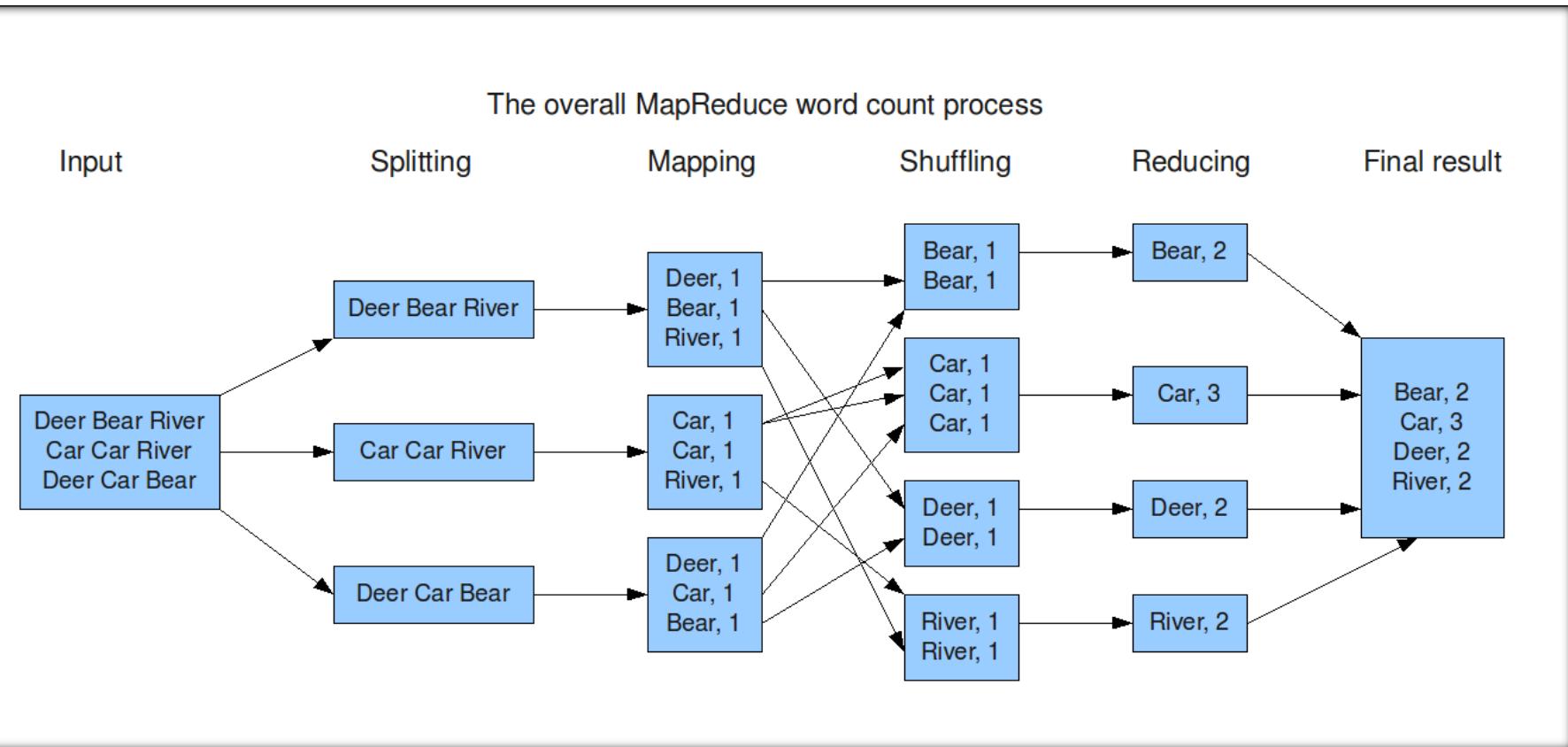
HDFS, Mapper, Shuffle and Sort, y Reduce



UNIVERSIDAD
NACIONAL
DE COLOMBIA



Calculando la ocurrencia de palabras en archivos



ANATOMÍA DE UN CLUSTER

Componentes de un Clúster en Hadoop

- *NameNode*
 - Mantiene la metadata para HDFS
- *Secondary NameNode*
 - Realiza funciones *housekeeping* para el *NameNode*
- *DataNode*
 - Almacena los actuales bloques de datos HDFS
- *JobTracker*
 - Administra los jobs *MapReduce*
- *TaskTracker*
 - Monitorea las tareas *Map* y *Reduce*

El *NameNode*



- Almacena la información del sistema de archivos HDFS en una *fsimage*
- Actualizaciones al sistema de archivos (add/remove blocks)
 - Escribe en un archivo de log
- Cuando inicia el *NameNode* se carga el archivo *fsimage* y se aplican los cambios en el archivo del archivo log

El *Secondary NameNode*

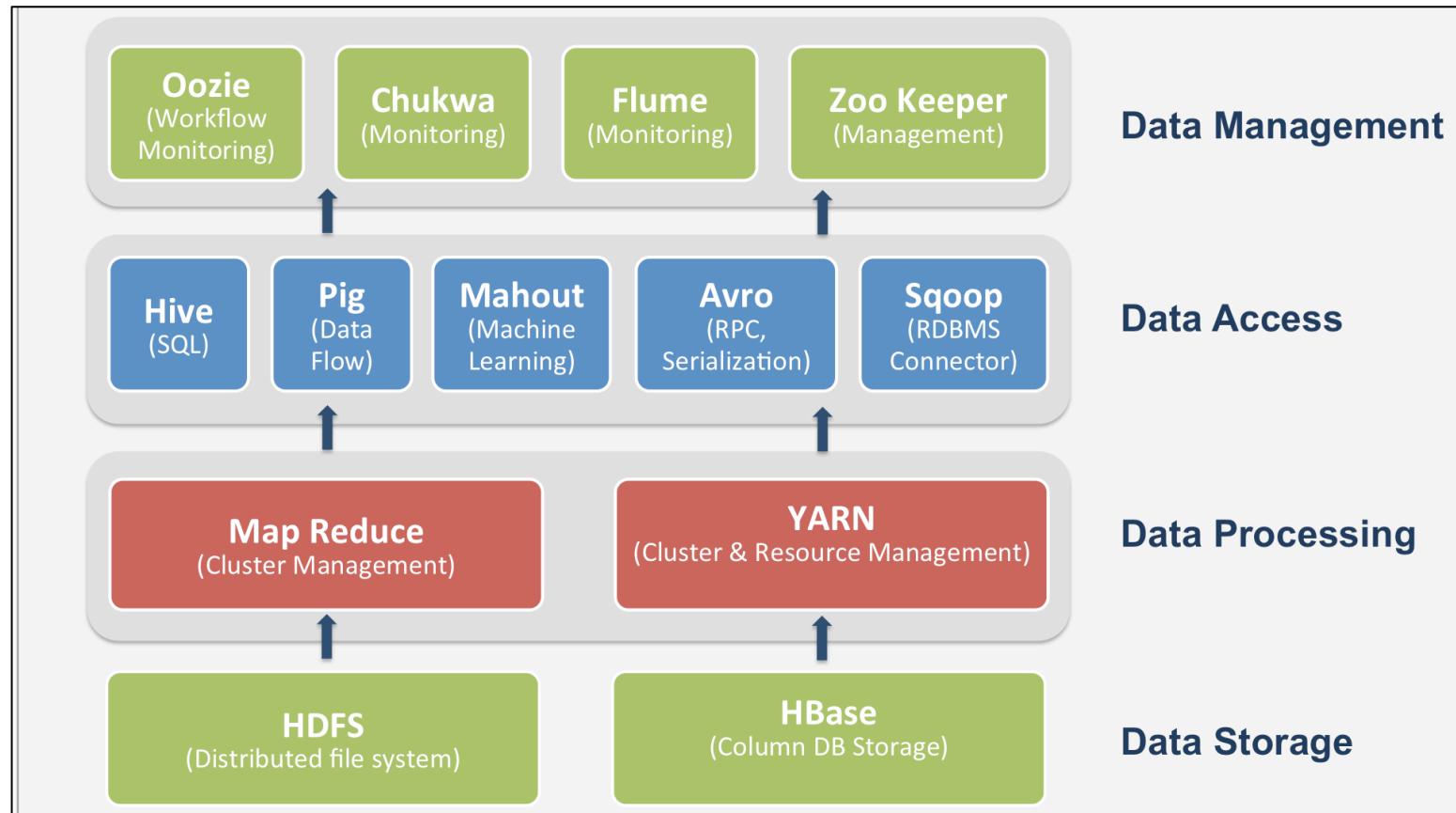
- No es una copia de respaldo del *NameNode*
- Periódicamente lee el archivo de log y aplica los cambios al archivo *fsimage* para actualizarlo
- Permite que el *NameNode* reinice más rápido cuando se requiera

El *JobTracker* y *TaskTracker*

- *JobTracker*
 - Determina el plan de ejecución para el job
 - Asigna tareas individuales
- *TaskTracker*
 - Mantiene la trazabilidad del desempeño de cada *mapper* o *reducer*



Ecosistema Hadoop



PROGRAMACIÓN EN HADOOP

Ejemplo: Word Count (Mapper)

```
01 package com.javacodegeeks.examples.wordcount;
02
03 import java.io.IOException;
04 import java.util.StringTokenizer;
05
06 import org.apache.hadoop.io.IntWritable;
07 import org.apache.hadoop.io.LongWritable;
08 import org.apache.hadoop.io.Text;
09 import org.apache.hadoop.mapreduce.Mapper;
10
11 public class MapClass extends Mapper<LongWritable, Text, Text, IntWritable>{
12
13     private final static IntWritable one = new IntWritable(1);
14     private Text word = new Text();
15
16     @Override
17     protected void map(LongWritable key, Text value,
18                         Context context)
19                         throws IOException, InterruptedException {
20
21         String line = value.toString();
22         StringTokenizer st = new StringTokenizer(line, " ");
23
24         while(st.hasMoreTokens()){
25             word.set(st.nextToken());
26             context.write(word,one);
27         }
28     }
29 }
30 }
```

Ejemplo: Word Count (Reducer)

```
01 package com.javacodegeeks.examples.wordcount;
02
03 import java.io.IOException;
04 import java.util.Iterator;
05
06 import org.apache.hadoop.io.IntWritable;
07 import org.apache.hadoop.io.Text;
08 import org.apache.hadoop.mapreduce.Reducer;
09
10 public class ReduceClass extends Reducer{
11
12     @Override
13     protected void reduce(Text key, Iterable values,
14                          Context context)
15             throws IOException, InterruptedException {
16
17         int sum = 0;
18         Iterator valuesIt = values.iterator();
19
20         while(valuesIt.hasNext()){
21             sum = sum + valuesIt.next().get();
22         }
23
24         context.write(key, new IntWritable(sum));
25     }
26 }
```

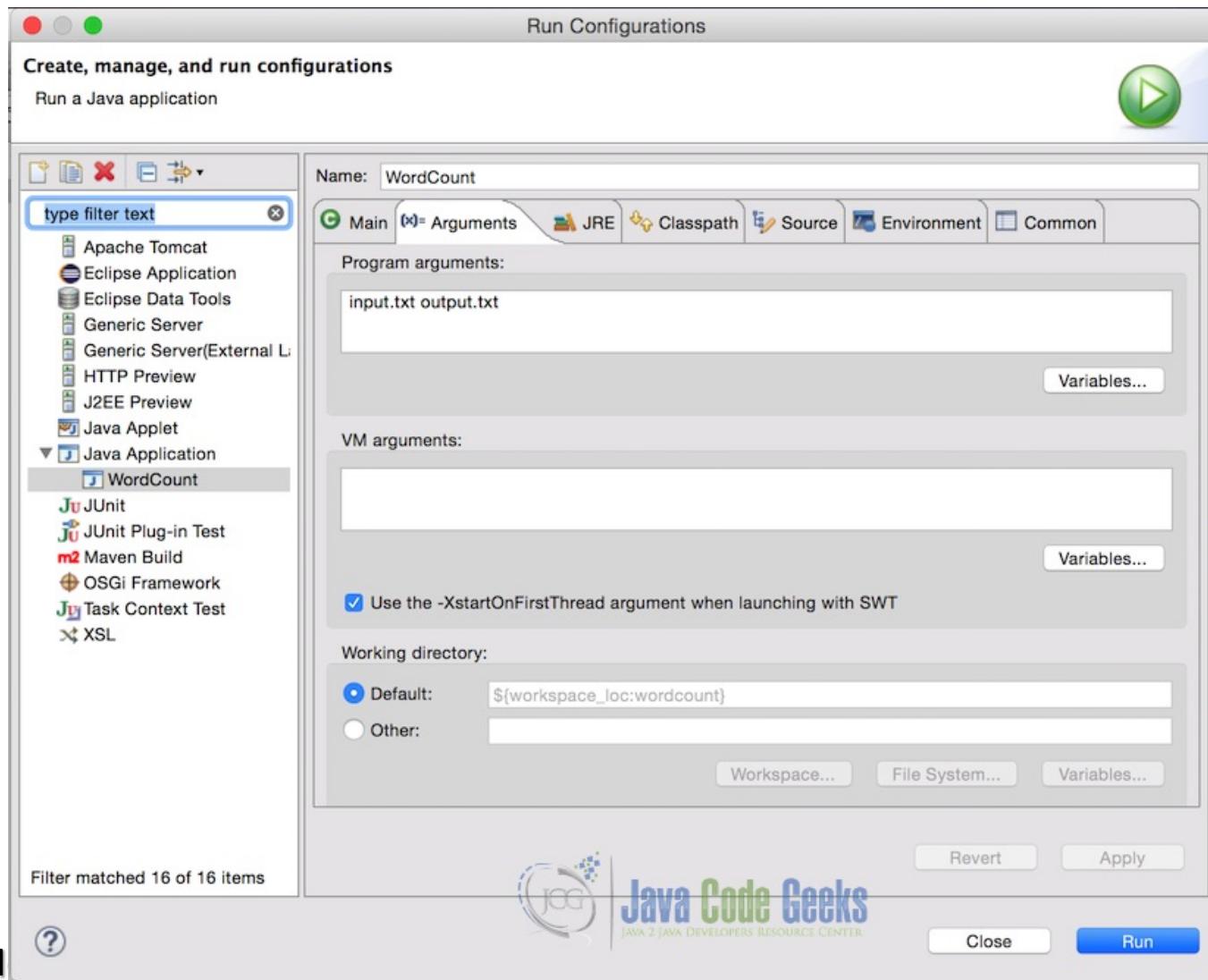


Ejemplo: Word Count (Job)

```
01 package com.javacodegeeks.examples.wordcount;
02
03 import org.apache.hadoop.conf.Configured;
04 import org.apache.hadoop.fs.Path;
05 import org.apache.hadoop.io.IntWritable;
06 import org.apache.hadoop.io.Text;
07 import org.apache.hadoop.mapreduce.Job;
08 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
09 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
11 import org.apache.hadoop.util.Tool;
12 import org.apache.hadoop.util.ToolRunner;
13
14 public class WordCount extends Configured implements Tool{
15
16     public static void main(String[] args) throws Exception{
17         int exitCode = ToolRunner.run(new WordCount(), args);
18         System.exit(exitCode);
19     }
20
21     public int run(String[] args) throws Exception {
22         if (args.length != 2) {
23             System.err.printf("Usage: %s needs two arguments, input and output
24 files\n", getClass().getSimpleName());
25             return -1;
26         }
27
28         Job job = new Job();
29         job.setJarByClass(WordCount.class);
30         job.setJobName("WordCounter");
31
32         FileInputFormat.addInputPath(job, new Path(args[0]));
33         FileOutputFormat.setOutputPath(job, new Path(args[1]));
34
35         job.setOutputKeyClass(Text.class);
36         job.setOutputValueClass(IntWritable.class);
37         job.setOutputFormatClass(TextOutputFormat.class);
38
39         job.setMapperClass(MapClass.class);
40         job.setReducerClass(ReduceClass.class);
41
42         int returnValue = job.waitForCompletion(true) ? 0:1;
43
44         if(job.isSuccessful()) {
45             System.out.println("Job was successful");
46         } else if(!job.isSuccessful()) {
47             System.out.println("Job was not successful");
48         }
49
50         return returnValue;
51     }
52 }
```



Ejemplo: Word Count (Ejecución)



Ejemplo: Word Count (Ejecución)

```
Problems @ Javadoc Declaration Console X
<terminated> WordCount [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (N
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Reduce output records=43
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Map output records=58
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Combine input records=0
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Total committed heap usage (bytes)=514850816
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: File Input Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Bytes Read=322
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: FileSystemCounters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: FILE_BYTES_WRITTEN=103805
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: FILE_BYTES_READ=1676
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: File Output Format Counters
Nov 24, 2015 3:19:49 AM org.apache.hadoop.mapred.Counters log
INFO: Bytes Written=351
Job was successful
```



Ejemplo: Word Count (Salida)

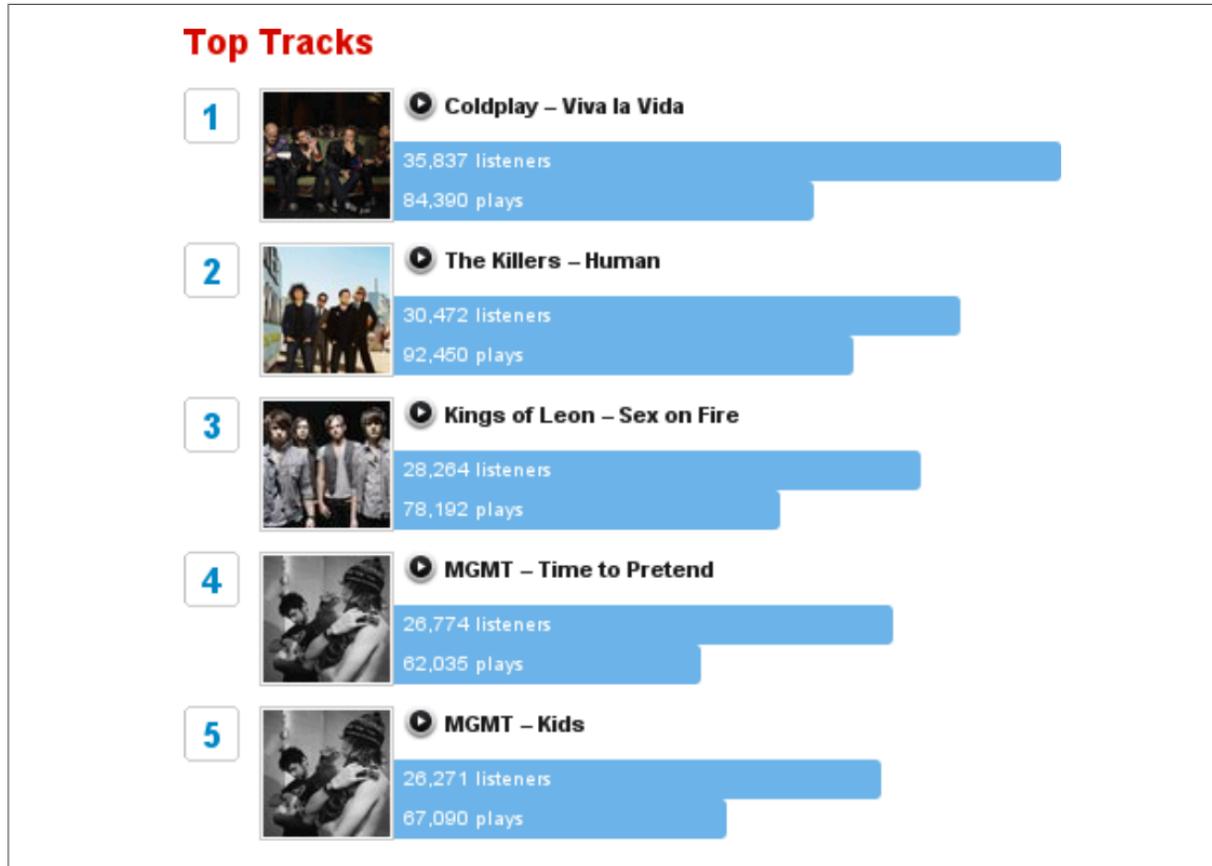
```
01 Hadoop 1
02 The 2
03 This 2
04 above 1
05 all 1
06 alphabets. 1
07 also 1
08 article 1
09 as 1
10 brown 1
11 code 1
12 contains 1
13 count 1
14 dog. 1
15 ecosystem. 1
16 english 1
17 example 4
18 examples 1
19 famous 1
20 file 1
21 for 2
22 fox 1
23 geek 1
24 hello 1
25 is 3
26 java 1
27 jumps 1
28 knows 1
29 language 1
30 lazy 1
31 line 1
32 lines 1
33 most 1
34 of 3
35 one 1
36 over 1
37 quick 1
38 text 1
39 the 6
40 which 1
41 word 1
42 world 1
43 written 1
```

CASOS DE ESTUDIO

Caso 1: LastFM

LastFM desea encontrar la cantidad de usuarios que han escuchado un track.
Solo se cuenta una vez por cada usuario.

LastFM



Datos

UserId|TrackId|Shared|Radio|Skip

111115|222|0|1|0

111113|225|1|0|0

111117|223|0|1|1

111115|225|1|0|0

Datos

| Line of file | UserId | TrackId | Scrobbled | Radio play | Skip |
|--------------|-------------|-------------|-----------|------------|---------|
| LongWritable | IntWritable | IntWritable | Boolean | Boolean | Boolean |
| 0 | 11115 | 222 | 0 | 1 | 0 |
| 1 | 11113 | 225 | 1 | 0 | 0 |
| 2 | 11117 | 223 | 0 | 1 | 1 |
| 3 | 11115 | 225 | 1 | 0 | 0 |

Constantes

```
public class LastFMConstants {  
  
    public static final int USER_ID = 0;  
    public static final int TRACK_ID = 1;  
    public static final int IS_SHARED = 2;  
    public static final int RADIO = 3;  
    public static final int IS_SKIPPED = 4;  
  
}
```

Mapper

```
public static class UniqueListenersMapper extends
Mapper< Object , Text, IntWritable, IntWritable > {
    IntWritable trackId = new IntWritable();
    IntWritable userId = new IntWritable();

    public void map(Object key, Text value,
        Mapper< Object , Text, IntWritable, IntWritable, Context context)
        throws IOException, InterruptedException {

        String[] parts = value.toString().split("[|]");
        trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
        userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
        if (parts.length == 5) {
            context.write(trackId, userId);
        } else {
            // add counter for invalid records
            context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
        }
    }
}
```

Reducer

```
public static class UniqueListenersReducer extends
    Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
    public void reduce(
        IntWritable trackId,
        Iterable< IntWritable > userIds,
        Reducer< IntWritable , IntWritable, IntWritable,
        IntWritable>.Context context)
    throws IOException, InterruptedException {
    Set< Integer > userIdSet = new HashSet< Integer >();
    for (IntWritable userId : userIds) {
        userIdSet.add(userId.get());
    }
    IntWritable size = new IntWritable(userIdSet.size());
    context.write(trackId, size);
}
}
```

Job

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    if (args.length != 2) {
        System.err.println("Usage: uniquelisteners < in > < out >");
        System.exit(2);
    }
    Job job = new Job(conf, "Unique listeners per track");
    job.setJarByClass(UniqueListeners.class);
    job.setMapperClass(UniqueListenersMapper.class);
    job.setReducerClass(UniqueListenersReducer.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
    org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
    System.out.println("No. of Invalid Records :"
        + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
        .getValue());
}
```

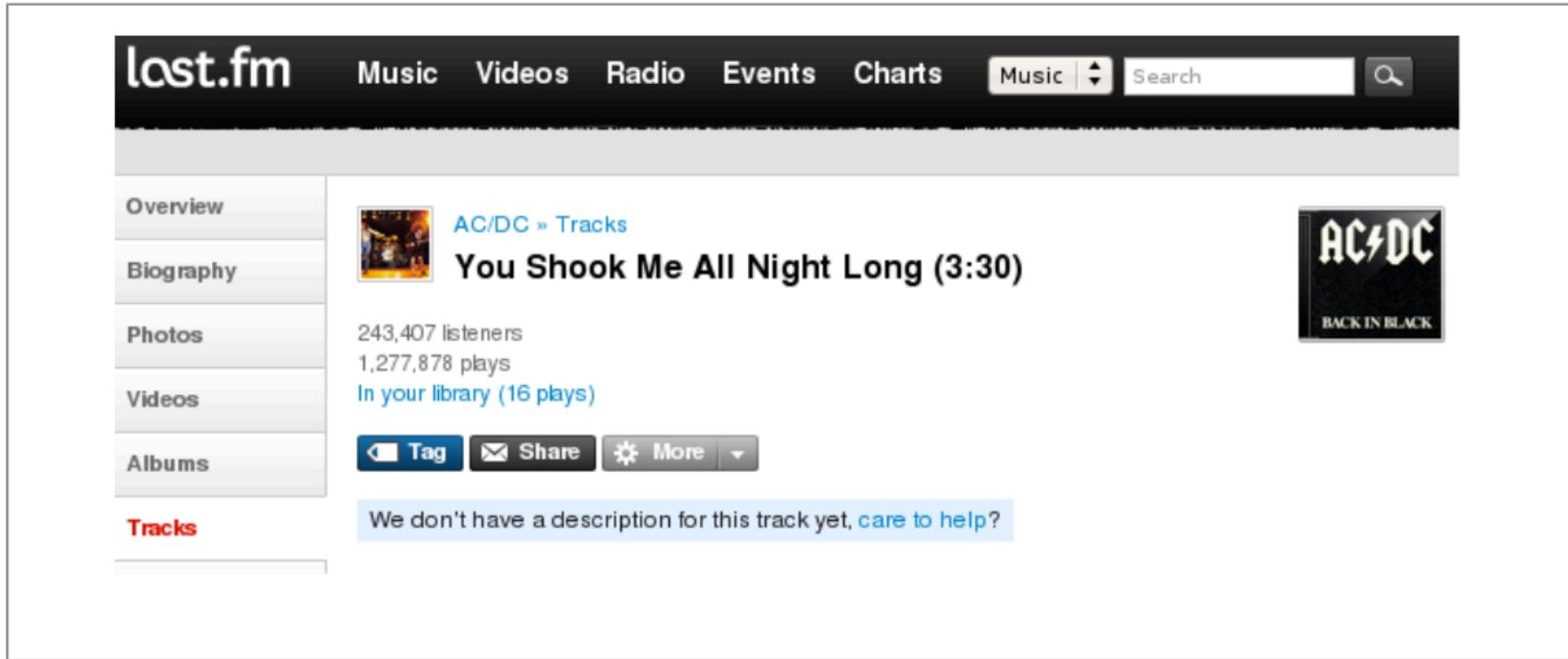
Salida del Mapper

| TrackId | UserId |
|-------------|-------------|
| IntWritable | IntWritable |
| 222 | 11115 |
| 225 | 11113 |
| 223 | 11117 |
| 225 | 11115 |

Salida del Reducer

| TrackId | #listeners |
|-------------|-------------|
| IntWritable | IntWritable |
| 222 | 1 |
| 225 | 2 |
| 223 | 1 |

Resultado final



The screenshot shows a last.fm track page for the song "You Shook Me All Night Long" by AC/DC. The top navigation bar includes links for Music, Videos, Radio, Events, Charts, and a dropdown menu. A search bar and a magnifying glass icon are also present. On the left, a sidebar lists Overview, Biography, Photos, Videos, Albums, and Tracks, with Tracks being the active tab. The main content area displays the album art for "BACK IN BLACK" and the song title "You Shook Me All Night Long (3:30)". It shows statistics: 243,407 listeners and 1,277,878 plays. Below that, it says "In your library (16 plays)". There are buttons for Tag, Share, and More. A note at the bottom states, "We don't have a description for this track yet, care to help?"

Caso 2: Compañía Telco

Una Telco desea encontrar en los CDRs cuáles abonados están haciendo llamadas de larga distancia de mas de 60 minutos para ofrecerles un mejor plan.

Datos

FromPhoneNumber|ToPhoneNumber|CallStartTime|CallEndTime|STDFlag
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128505|8983006310|2015-03-01 07:08:10|2015-03-01 08:12:15|0
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 09:12:15|1
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|0
9665128506|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128507|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1

Constantes

```
package in.co.hadoop.tutorials;

public class CDRConstants {

    public static int fromPhoneNumber = 0;
    public static int toPhoneNumber = 1;
    public static int callStartTime = 2;
    public static int callEndTime = 3;
    public static int STDFlag = 4;

}
```

Mapper

```

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

    Text phoneNumber = new Text();
    LongWritable durationInMinutes = new LongWritable();

    public void map(Object key, Text value,
                    Mapper< Object , Text, Text, LongWritable, Context context>
                    throws IOException, InterruptedException {
        String[] parts = value.toString().split("[|]");
        if (parts[CDRConstants.STDFlag].equalsIgnoreCase("1")) {

            phoneNumber.set(parts[CDRConstants.fromPhoneNumber]);
            String callEndTime = parts[CDRConstants.callEndTime];
            String callStartTime = parts[CDRConstants.callStartTime];
            long duration = toMillis(callEndTime) - toMillis(callStartTime);
            durationInMinutes.set(duration / (1000 * 60));
            context.write(phoneNumber, durationInMinutes);
        }
    }

    private long toMillis(String date) {

        SimpleDateFormat format = new SimpleDateFormat(
            "yyyy-MM-dd HH:mm:ss");
        Date dateFrm = null;
        try {
            dateFrm = format.parse(date);

        } catch (ParseException e) {

            e.printStackTrace();
        }
        return dateFrm.getTime();
    }
}

```

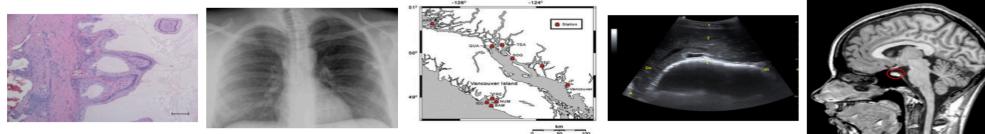
Reducer

```
public static class SumReducer extends  
    Reducer< Text , LongWritable, Text, LongWritable> {  
    private LongWritable result = new LongWritable();  
  
    public void reduce(Text key, Iterable< LongWritable> values,  
                      Reducer< Text , LongWritable, Text, LongWritable>.Context context)  
        throws IOException, InterruptedException {  
        long sum = 0;  
        for (LongWritable val : values) {  
            sum += val.get();  
        }  
        this.result.set(sum);  
        if (sum >= 60) {  
            context.write(key, this.result);  
        }  
    }  
}
```

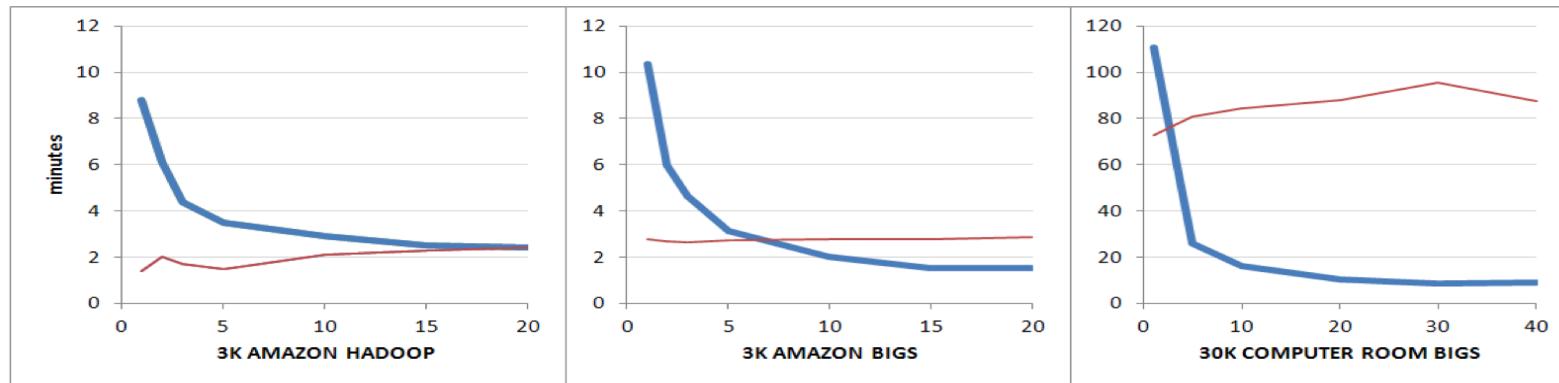
Job

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration();  
    if (args.length != 2) {  
        System.err.println("Usage: stdsubscriber < in> < out>");  
        System.exit(2);  
    }  
    Job job = new Job(conf, "STD Subscribers");  
    job.setJarByClass(STDSubscribers.class);  
    job.setMapperClass(TokenizerMapper.class);  
    job.setCombinerClass(SumReducer.class);  
    job.setReducerClass(SumReducer.class);  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(LongWritable.class);  
    FileInputFormat.addInputPath(job, new Path(args[0]));  
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  
    System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

Caso 3: ImageCLEF2012 - Clasificación de imágenes médicas



- 300K images
- Hadoop
- Hbase
- BIGS



Ramos-Pollan, R.; González, F.A.; Caicedo, J.C.; Cruz-Roa, A.; **Camargo, J.E.**; Vanegas, J.A.; Pérez, S.A.; Bermeo, J.D.; Otálora, J.S.; Rozo, P.K.; Arevalo, J.E., "BIGS: A framework for large-scale image processing and analysis over distributed and heterogeneous computing resources," E-Science (e-Science), 2012

Amazon Elastic Map Reduce

Amazon EMR

- Cluster list
- Security configurations
- VPC subnets
- Help

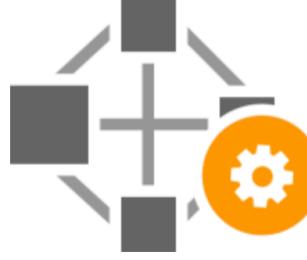
Welcome to Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

You do not appear to have any clusters. Create one now:

[Create cluster](#)

How Elastic MapReduce Works

| Upload | Create | Monitor |
|--|---|--|
|  |  |  |
| Upload your data and processing application to S3. | Configure and create your cluster by specifying data inputs, outputs, cluster size, security settings, etc. | Monitor the health and progress of your cluster. Retrieve the output in S3. |

Referencias

- Documentación Apache Hadoop,
<http://hadoop.apache.org/docs/stable>
- Daniel Leblanc, "*Hadoop Lecture Information*", Portland State University, <http://web.cecs.pdx.edu/~dleblanc/>
- Ramos-Pollan, R.; González, F.A.; Caicedo, J.C.; Cruz-Roa, A.; Camargo, J.E.; Vanegas, J.A.; Pérez, S.A.; Bermeo, J.D.; Otálora, J.S.; Rozo, P.K.; Arevalo, J.E., "*BIGS: A framework for large-scale image processing and analysis over distributed and heterogeneous computing resources,*" E-Science (e-Science), 2012
- Tom White, Hadoop: The definitive Guide. Third Edition, O'reilly, 2012

¿Preguntas?

jecamargom@unal.edu.co

[http://www.ingenieria.unal.edu.co/min
lab/](http://www.ingenieria.unal.edu.co/minlab/)

