Fuentes de Datos

PIG

Introducción

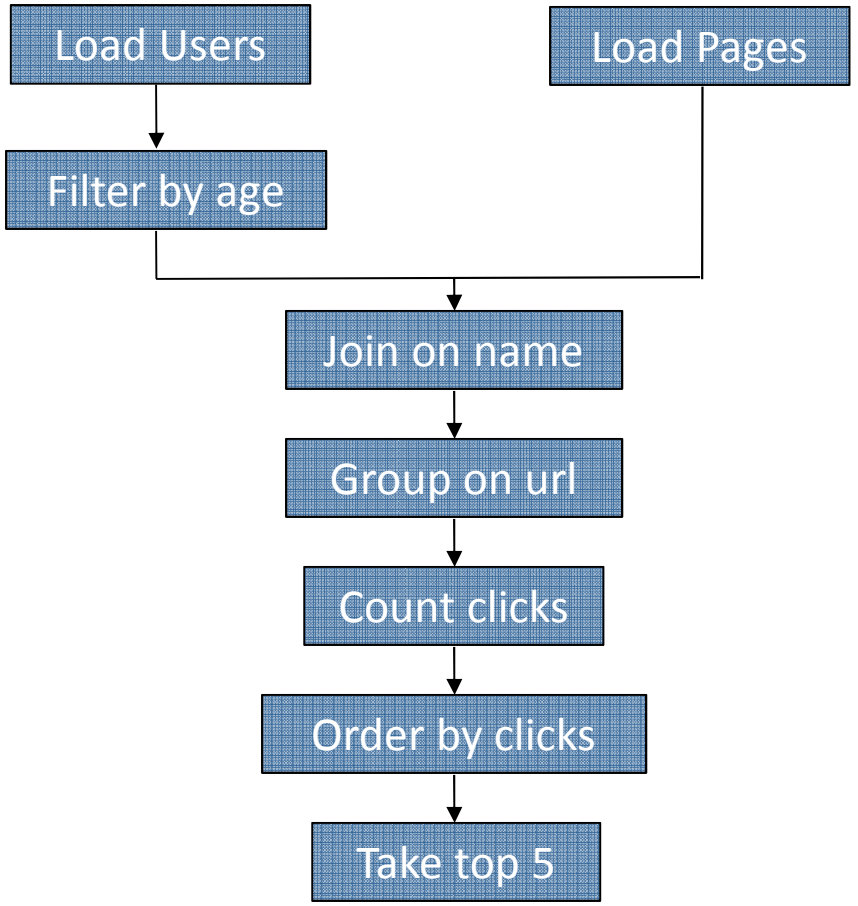
- La manipulación y análisis de datos no triviales en Hadoop, requiere de una serie de fases MapReduce
  - Proceso complejo y ciclo de producción muy lento
- Pig ofrece un nivel de abstracción para expresar el procesamiento de los datos como una serie de declaraciones de alto nivel
  - Las declaraciones de alto nivel se transforman en tareas map y reduce sin intervención del usuario
  - Simplifica la programación para analizar grandes volúmenes de datos
  - … pero mantiene el principio de procesamiento en batch con alta latencia

Pig

- Modelo de datos:
  - "sacos" (bags) de objetos (items)
- Operaciones expresadas como relaciones
  - Lejanamente similar a SQL pero mantiene una lógica de lenguaje procedural
  - Operadores relacionales como JOIN, GROUP, FILTER
- Lenguaje extensible
  - Los usuarios pueden definir sus propias funciones (UDF, *User defined functions*)

Motivación

- Considere que se tienen los datos de usuarios en un archivo, datos de sitios web en otro y se desea identificar cuáles son las 5 páginas más visitadas por usuarios en el rango 18 a 25 años



Fuente: Matei Zaharia

Este sería el código en MapReduce (Java)

```
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 value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.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.setInputFormat(TextInputFormat.class);
        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);
        jc.addJob(limit);
        jc.run();
    }
}
```

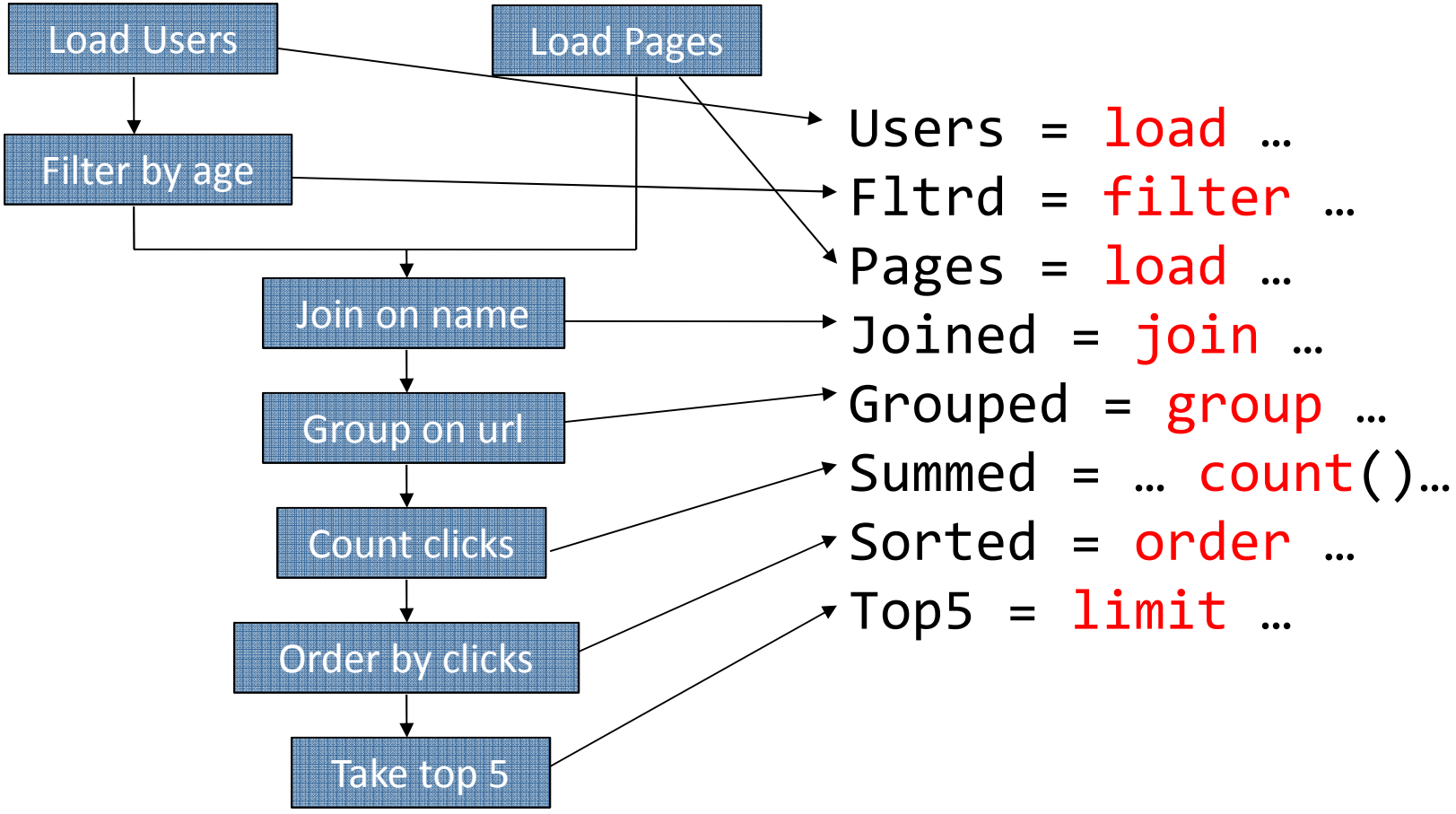

Fuente: Matei Zaharia

Este es el código Pig Latin

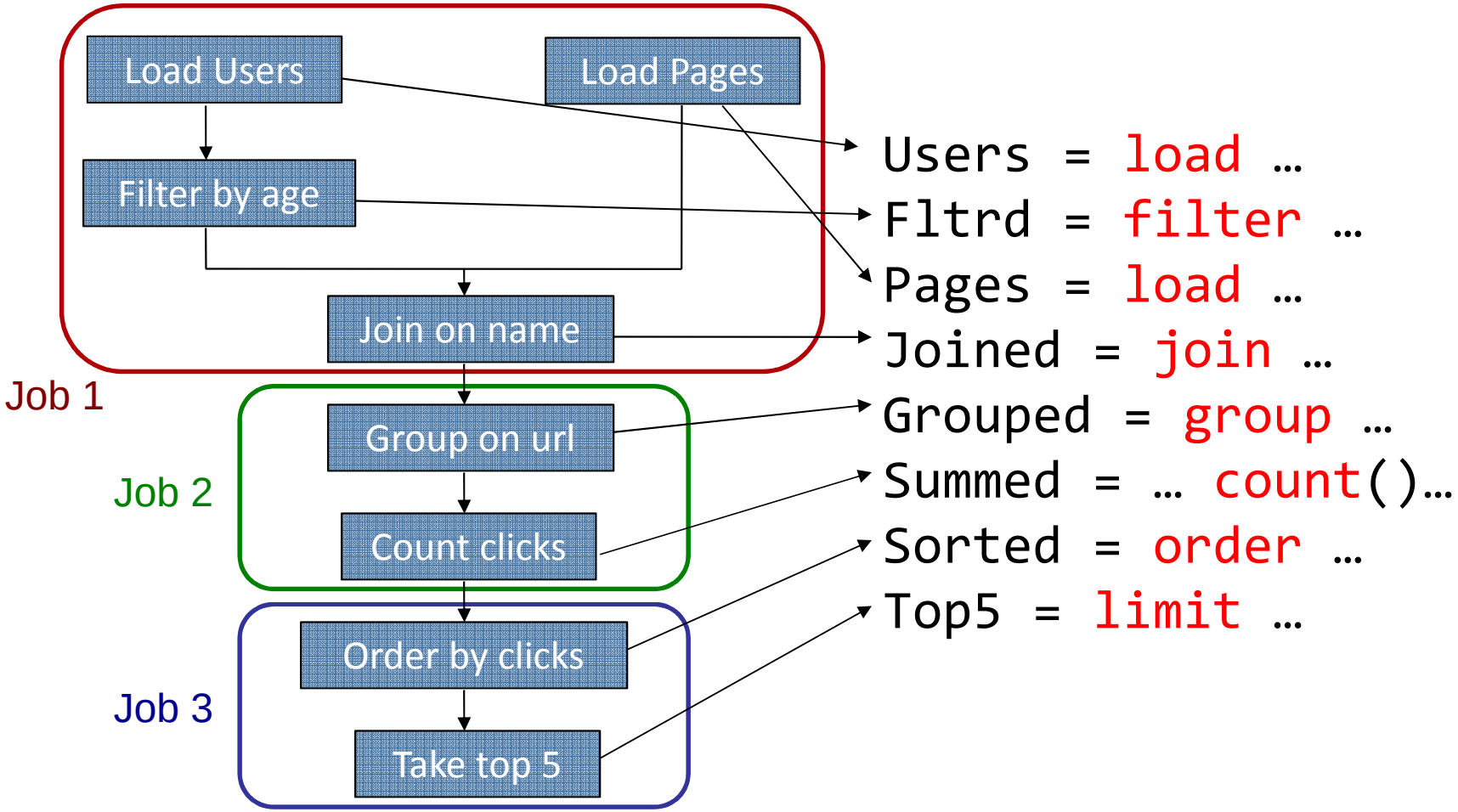
```
Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <=
  25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
                 count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
```

Pig Latin es el lenguaje de Pig

El flujo de ejecución se traduce fácilmente en operadores de Pig



… y éstos, en jobs MapReduce



Conceptos básicos

- Formado por dos componentes
  - Ambiente de ejecución
    - Puede ser local (una sola VM Java) o ejecución en cluster (MapReduce)
    - Puede invocarse de forma interactiva (para depuración) o para ejecutar un script
      - En forma interactiva, se accede al intérprete de comandos Grunt
  - Lenguaje de programación
    - Pig Latin (o simplemente Pig)

- Las declaraciones forman un *plan lógico* (secuencia de ejecución) que se transforman en un *plan físico* (Jobs MapReduce) cuando se va a ejecutar

Arquitectura

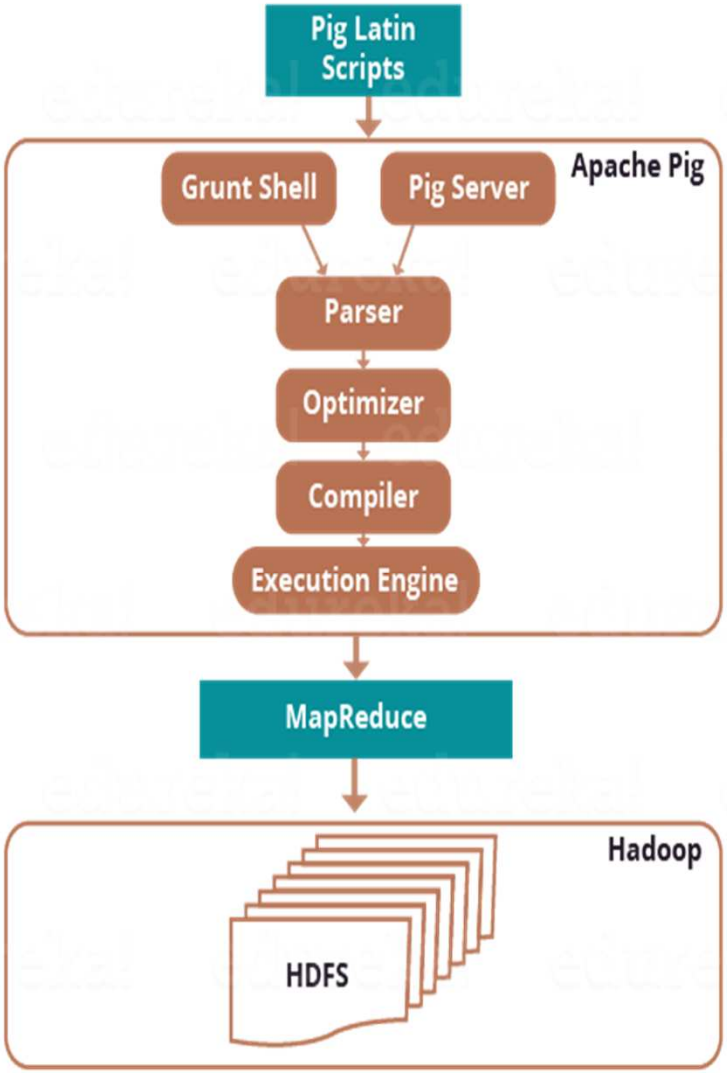


Figure: Apache Pig Architecture

Conceptos básicos

```
$ pig –x local
grunt>
```

Modo interactivo local
Se ejecuta en una jvm
Utiliza el sistema de archivos local

```
$ pig –x
grunt>
```

Modo interactivo MapReduce
Se ejecuta en cluster Hadoop
Utiliza el sistema de archivos HDFS

```
$ pig –x local miScript.pig

$ pig miScript.pig
```

Ejecutan script (no interactivo) de forma local en una jvm o en modo MapReduce en un cluster Hadoop

Tipos de datos

scalar:
int 10
chararray diez
double 10.0
etc.

tuple:
(15, juan, 10.3)

Agrupación de escalares de cualquier tipo

bag:
(15, juan, 10.3)
(Luis, 32.4, 17)
(7.4, 11)

Agrupación de tuplas de cualquier longitud

relation:
(Lupe, {leon, gato, 7})
({Pedro, 6}, Ana)
(25, {Raul})

Los campos de una tupla pueden ser a su vez una *bag* que tiene otras tuplas.

map:
[1#rojo,2#azul,3#verde]

Asociaciones <key,value>

Declaraciones

- Un programa en Pig consiste en un conjunto de declaraciones (operadores de Pig o comandos de shell)
- Las declaraciones terminan con ";"
- La mayoría de los operadores toman una *relation* como entrada y generan una *relation*. Una relation es un conjunto de tuplas
  - Las excepciones son LOAD que no tiene una relation de entrada, STORE y DUMP que no tienen una relation de salida
- Se usan comentarios tipo c (/* …*/) o doble guión (- -) para comentar el resto de una línea
- Las palabras reservadas no son sensibles a mayúsculas; los nombres de relations, de campos y de funciones sí son sensibles

Un programa no se ejecuta (el plan físico no se lanza) hasta que encuentre una declaración DUMP o STORE

Operadores básicos

Aritméticos

+: a + b, “a” + ”b”
-: a - b, -a
*: a * b
/: a / b
%: modulo
?: operador ternario

Booleanos

and: a and b
or: a or b
not: a and not b
not (a and b)

Comparación

==: a and b
!=: a or b
<, <=: a < b, a <= b
>, >=: a > b, a >= b
is null
is not null

Operadores relacionales

Carga y guarda

LOAD
STORE
DUMP

Filtrado

FILTER
DISTINCT
FOREACH…
GENERATE
MAPREDUCE
STREAM
SAMPLE

Agrupamiento

JOIN
COGROUP
GROUP
CROSS

Ordenamiento

ORDER
LIMIT

Combina y separa

UNION
SPLIT

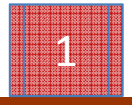
LOAD: Carga de datos

- Tiene varias funciones para la lectura de distintos tipos de datos:
  - BinStorage() para formato interno, PigStorage() para csv, JsonLoader() para estructuras JSON, TextLoader() para texto no estructurado
  - Para datos en un formato propietario, se pueden definir UDFs

A = LOAD 'miArch' using PigStorage(',') as (clave: int, nombre:chararray, edad: int)

A es una relation donde se accede a los datos leídos
using PigStorage es opcional y especifica el delimitador de los campos. Por omisión es "\t"
as (…) es opcional y permite especificar un "schema", a los datos de entrada, asignando nombre y tipo a los campos.
Si se omite, los campos se pueden acceder por posición ($0, $2,…)

Si la entrada es un directorio, se leen todos los archivos en él



Salida: STORE, DUMP

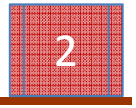
- Ya hemos usado DUMP. Despliega en la consola
  - Utilizada para depurar en modo interactivo
- STORE manda resultados al sistema de archivos.
  - Igual que LOAD, tiene varias funciones para el manejo de distintos tipos de datos: BinStorage(), PigStorage(),PigDump(), JsonStorage()

```
STORE A into 'miDir';
```

Por omisión, el separador de campos es '\t'. Si se desea otro separador, utilizamos la función PigStorage

Los datos se guardan en un archivo debajo de la carpeta especificada

Si la carpeta ya existe, marca un error



FILTER, ORDER BY

- Filter selecciona tuplas de una relation con base en algún criterio
- Order by ordena de forma ascendente (ASC) o descendente (DESC) sobre uno o varios campos

```
B = FILTER A BY clave == 241;
B = FILTER A BY $2 == “Luis”;

B = ORDER A BY edad DESC;
```

Por supuesto, se necesita que la relación tenga un schema para poder identificar los campos por nombre; de lo contrario, se pueden especificar campos por posición

GROUP

- Agrupa tuplas con la misma llave
  - Genera una relation con dos campos:
    - El primero (*group*) contiene la llave
    - El segundo es una *bag* con los campos agrupados, manteniendo el *schema* de la relation original

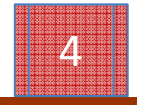
```
B = GROUP A BY $0
```

```
DUMP A;
(1,2,3)
(4,5,6)
(1,8,9)
(4,3,2)

DUMP B;
(1,{(1,2,3),(1,8,9)})
(4,{(4,5,6),(4,3,2)})
```

La llave para agrupar puede contener varios campos y hasta alguna expresión regular. Por ejemplo GROUP A BY SIZE($3) agruparía en función del número de caracteres del 4º campo

GROUP A ALL; (sin cláusula BY) agrupa todas las tuplas en una relation. Se usa frecuentemente para contar el número de tuplas



FOREACH…GENERATE

- Permite crear una nueva relation agregando o eliminando campos de una existente (una proyección de la relation)
  - Puede operar en un bloque o en bloques anidados

```
B = FOREACH A GENERATE $0, $1+$2 as f1:int, 'Cte';
```

El schema es opcional

```
DUMP A;
(1,2,3)
(4,5,6)
(1,8,9)
(4,3,2)

DUMP B;
(1,5,Cte)
(4,11,Cte)
(1,17,Cte)
(4,5,Cte)
```

En este caso, se está accediendo a los campos por posición.

5

FOREACH en una *grouped relation*

```
A = load 'arch' as (x1:int, x2:int, x3:int);
B = group A by x1;
C = foreach B generate group, A.x2, A.x3;
```

```
DUMP A;
(1,2,3)
(4,5,6)
(1,8,9)
(4,3,2)
```

```
DUMP B;
(1,{(1,2,3),(1,8,9)})
(4,{(4,5,6),(4,3,2)})
```

```
DUMP C;
(1, {(2),(8)},{(3),(9)})
(4, {(5),(3)},{(6),(2)})
```

- Las palabras clave no son sensibles a mayúsculas
- El primer campo en una tupla agrupada, se llama *group*
- Para acceder a los miembros del segundo campo, se utiliza un **operador de derreferencia** con la notación x.y

FOREACH en un bloque anidado

```
A = load 'arch' as (x1:int, x2:int, x3:int);
B = group A by x1;
C = foreach B {
        fdata = filter A by x1 < 3;  pdata = fdata.x1;
        generate group,COUNT(pdata);};
```

```
DUMP A;
(1,2,3)
(4,5,6)
(1,8,9)
(4,3,2)

DUMP B;
(1,{(1,2,3),(1,8,9)})
(4,{(4,5,6),(4,3,2)})

DUMP C;
(1, 2)
(4, 0)
```

- Las declaraciones del bloque anidado se delimitan entre llaves
- En la tupla con *group* =1, hay dos tuplas; la tupla con *group*=4, no cumple el criterio de filtrado, por lo que tiene 0 tuplas

FLATTEN

- Sustituye un campo de una tupla por la tupla o desanida una *bag*

```
DUMP A;
(1,(2,3))
(4,(5,6))
(1,(8,9))
(4,(3,2))
```

```
B = FOREACH A GENERATE $0, FLATTEN($1);
```

```
DUMP B;
(1,2,3)
(4, 5,6)
(1,8,9)
(4,3,2)
```

```
DUMP A;
(1,{(1,2,3),(1,4,5)})
(4, {(4,5,6),(4,7,8)})
(1, {(7,8,9),(7,9,3)})
```

```
B = FOREACH A GENERATE $0, FLATTEN($1);
```

```
DUMP B;
(1,1,2,3)
(4,4,5,6)
(1,7,8,9)
```

DISTINCT, UNION

- DISTINCT elimina tuplas duplicadas en una relation
- UNION combina el contenido de dos o más relation

```
DUMP A;
(1,2,3)
(4,5,6)
(1,2,3)
(4,3,2)
```

```
B = DISTINCT A;
```

```
DUMP B;
(1,2,3)
(4,5,6)
(4,3,2)
```

```
DUMP A;
(1,2,3)
(4,5,6)

DUMP B;
(7,8)
(9,3)
```

```
C = UNION A, B;
```

```
DUMP C;
(1,2,3)
(4,5,6)
(7,8)
(9,3)
```

Pig es más flexible que SQL: Admite registros con distintas longitudes

SPLIT, CROSS

SPLIT Divide una relation en dos o más con base en alguna condición

```
DUMP A;
(1,2,3)
(4,5,6)
(1,2,3)
(4,3,2)
```

```
SPLIT A INTO B IF $0 <3,
      C IF $0 > 3;
```

```
DUMP B;
(1,2,3)
(1,2,3)

DUMP C;
(4,5,6)
(4,3,2)
```

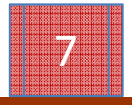
CROSS calcula el producto cruz de dos o más relations

```
DUMP A;
(1,2,3)
(4,5,6)

DUMP B;
(7,8)
(5,3)
```

```
C = CROSS A, B;
```

```
DUMP C;
(1,2,3,7,8)
(1,2,3,5,3)
(4,5,6,7,8)
(4,5,6,5,3)
```



JOIN

- Union de dos o más relation. Tiene muchas variantes
  - Inner join: las relation tienen un campo común

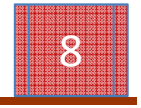
```
DUMP A;
(1,2)
(1,6)
(2,1)

DUMP B;
(3,1,5)
(4,2,6)
```

```
C = JOIN A BY $0, B BY $1;
```

```
DUMP C;
(1,2,3,1,5)
(1,6,3,1,5)
(2,1,4,2,6)
```

  - Outer join
    - Sólo se usa cuando se unen dos relations, y permite incluir entradas que no coincidan con el criterio, de la primer relation (LEFT), de la segunda (RIGHT) o de ambas (FULL)



Funciones de evaluación

- Pig cuenta con una amplia gama de funciones de evaluación, como AVG, COUNT, CONCAT, SUM, MAX, MIN, SIZE, TOKENIZE

```
A = load 'arch' as (dep:chararray,
emp:chararray, sueldo:float);

DUMP A;
Ventas, Juan, 65000.00
Ventas, Maria, 73500.00
Ventas, Luis, 70600.00
Mkt, Saul, 54700.00
Mkt, Ana, 63750.00
Mkt Blanca, 55600.00
```

```
B = foreach A generate group,AVG(A.sueldo);
```

```
(Ventas,69700.00)
(Mkt, 58016.00)
```

```
B = foreach A generate group,MAX(A.sueldo);
```

```
(Ventas,73500.00)
(Mkt, 63750.00)
```

```
B = foreach A generate group,SUM(A.sueldo);
```

```
(Ventas,209100.00)
(Mkt, 174050.00)
```

```
B = foreach A generate group,COUNT(A.sueldo);
```

```
(Ventas,3)
(Mkt, 3)
```

- Pig tiene muchas otras funciones
  - Matemáticas (ABS, CEIL, SIN, ..) basadas en la clase Math de Java
  - Para manejo de strings (SUBSTRING, STRSPLIT, …)
  - Para convertir expresiones a tuplas, bags y maps (TOTUPLE, TOBAG, TOMAP)
  - Para acceder a operaciones externas (STREAM, MAPREDUCE, UDF)
  - DESCRIBE para mostrar el schema
  - EXPLAIN para describir el plan lógico y físico que se desarrolla,
  - …

Fuentes de Datos

HIVE

Introducción

- Fue creado en 2007 por Facebook (Pig fue creado por Yahoo!)
  - Base de datos de 700 TB muy lenta en procesarse en su data warehouse
  - Migraron a Hadoop pero el modelo MapReduce era muy poco amigable
- Principio de diseño
  - Retener los conceptos familiares para la explotación de bases de datos (y DWH) manteniendo la extensibilidad, flexibilidad y bajo costo de Hadoop

En sus orígenes, Pig era más útil para diseño de scripts interactivos, como lenguaje de flujo de datos y para manejo de datos semi-estructurados. Hive fue concebido para manejo eficiente de DWH con datos estructurados.

En su evolución se han ido traslapando, por lo que hay muchas discusiones sobre cuándo usar Pig o Hive y hasta por qué se tienen los dos.

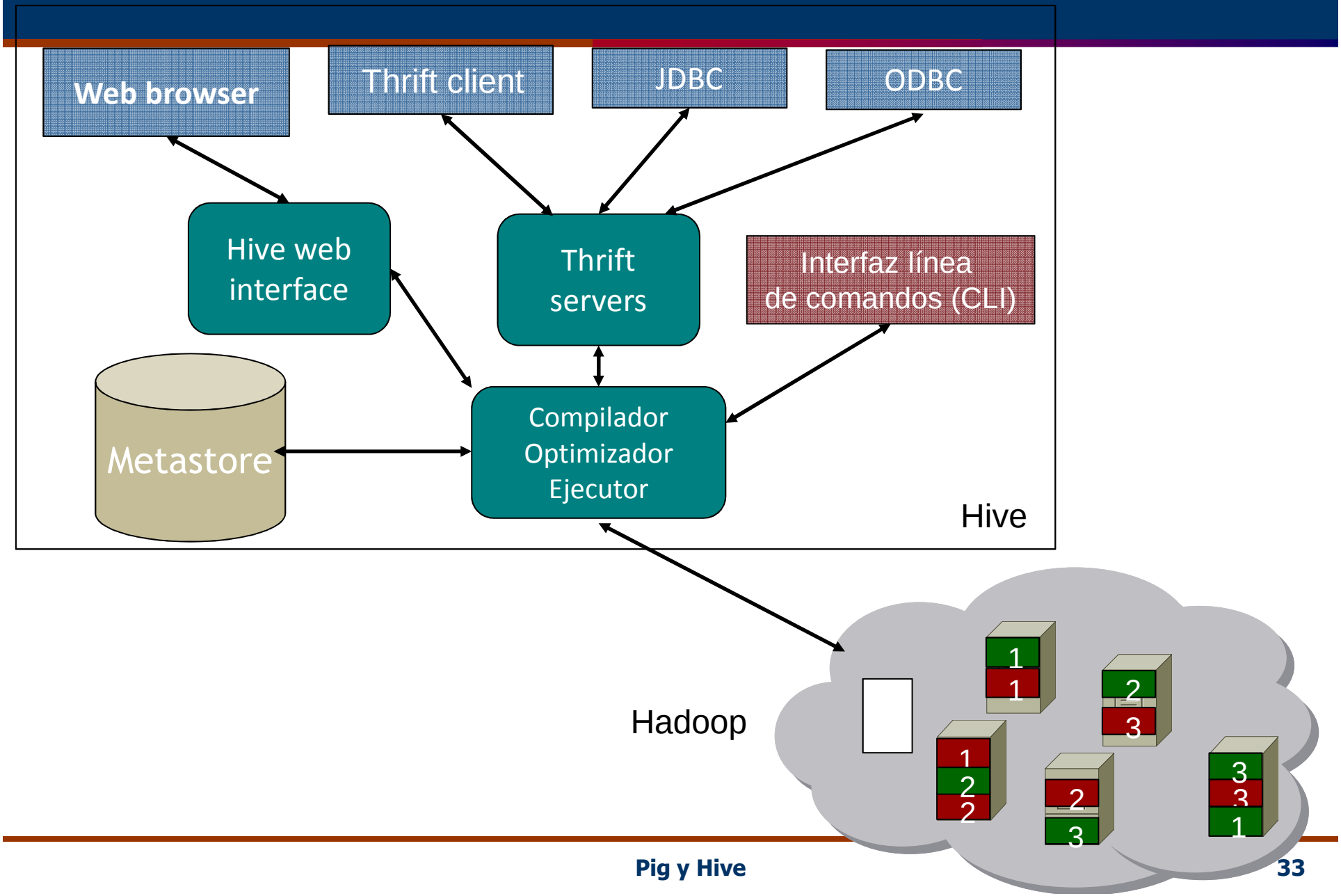
Hive

- Hive es:
  - Un datawarehouse sobre Hadoop
  - Facilita sumarización, queries adhHoc, análisis de datasets almacenados en Hadoop
  - Interfaz SQL (HQL). Data definition languange y data manipulation language
  - Permite proyectar estructura sobre datasets en Hadoop
  - Catalog metastore. Mapea estructura de archivos a forma tabular
- Hive NO ES:
  - Un Manejador de Base de Datos completo
  - Para procesamiento en tiempo real: Latencias mucho mayores que RDBMS. Schema on Read = carga rápida pero query costoso
  - No soporta modelos transaccionales

Sentencias

- Data Definition Language (DDL)
  - Relacionadas con el *schema* (la estructura) de las tablas
    - CREATE, ALTER, DROP, DESCRIBE, TRUNCATE, SHOW, …
- Data Manipulation Language (DDL)
  - Relacionadas con el procesamiento de las tablas
    - LOAD INSERT, UPDATE, JOIN, SELECT, GROUP BY, …
- User Defined Functions (UDF)
  - Permiten extender la funcionalidad de Hive

Componentes

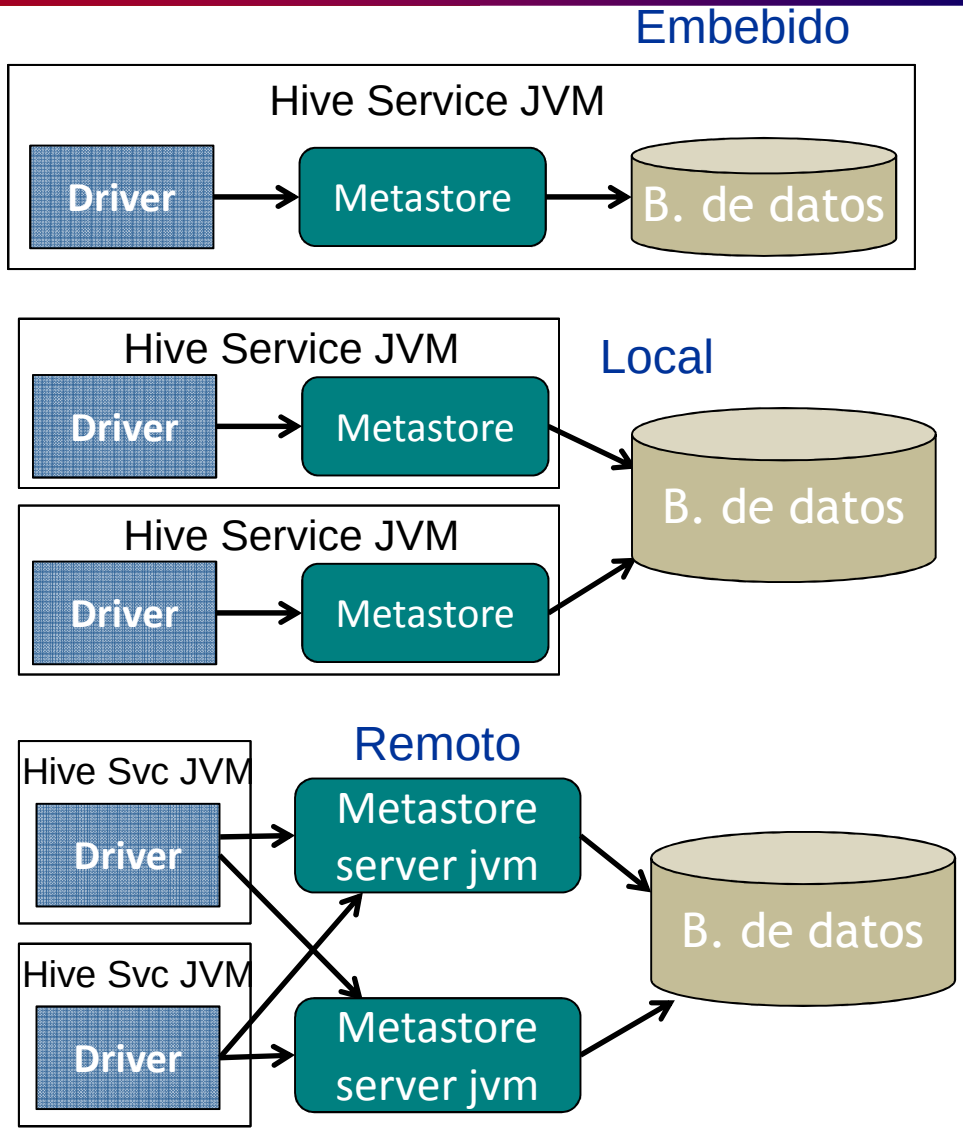


Metastore

Almacena el catálogo del sistema y metadata sobre tablas, columnas, particiones, etc.

Permite mapear estructuras de archivos a formas tabulares.

- Embebido. Ejecuta el código en el mismo proceso que Hive. La BD está en el mismo proceso. Típicamente para ambientes de pruebas
- Local. La BD está separada en otro proceso pero el código se ejecuta en el mismo proceso de Hive
- Remoto. Se ejecuta en un proceso independiente. Puede ser compartido por otros usuarios y procesos. Es el más común en ambientes de producción.



Componentes

HCatalog es un nuevo componente encima del metastore y ofrece interfaces de lectura y escritura para Pig y MapReduce.
Usa CLI para emitir definición de datos y comandos para explorar metadata.
HCatalog facilita a usuarios de Pig, MapReduce y Hive leer y escribir datos en una malla.
El driver, compilador, Optimizador y Ejecutor de Hive trabajan juntos para convertir un query en un conjunto de jobs.

- El Driver maneja el ciclo de vida de un comando HQL. Mantiene un handle y estadísticas para la sesión.
- El compilador convierte queries en un grafo dirigido de tareas MapReduce que son monitoreadas por la *execution engine*.

Organización de datos. DDL

En Hive los datos se organizan jerárquicamente en las siguientes estructuras (definidas por orden de granularidad)

Database –Namespace que separa tablas y otras unidades de datos

Table –Unidades *homogéneas*, con el mismo *schema*

Partition –Opcional, es una columna virtual que define cómo se almacena en el HDFS. Muy útil para optimización

Bucket – Opcional, son divisiones con base en un valor hash de una columna permite optimizar *joins*

Almacenamiento

- Los archivos se almacenan directamente en HDFS. Se puede utilizar una amplia variedad de formatos para los registros. El formato interno en realidad cambiará de tabla en tabla dependiendo de cómo se configure.

  Por default se almacenan en la carpeta warehouse del archivo de configuración. Se crea un directorio para la BD y subdirectorio para cada tabla, subsubdir para partition, y el archivo se almacena ahí o en varios bucketfiles si se decidió usarlos.

Entidad	Ejemplo	Ubicación
B. de datos	Midb	/apps/midb.db
Tabla	T	/apps/midb.db/T
Partición	Date='230602017	/apps/midb.db/T/date=23062017
Bucket col.	userId	/apps/midb.db/T/date=2306201700000_0 … /apps/midb.db/T/date=2306201700017_0

Tipos de datos

- Integer
  - TINYINT – 1 byte
  - SMALLINT – 2 bytes
  - INT – 4 bytes
  - BIGINT – 8 bytes
- Boolean
  - TRUE/FALSE
- Float
  - FLOAT, DOUBLE, DECIMAL
- String
  - STRING
  - VARCHAR (especifica long)
- Date/Time
  - TIMESTAMP YYYY-MM-DD:HH:MM:SS:ffffff
  - DATE –YYYY-MM-DD
- Binary
- Array
  - Cualquier tipo primitivo
  - Se indexan a partir de 0
- Struct
  - Colección de elementos de distinto tipo
  - Se acceden con notación punto
- Map
  - Colección de tuplas <key-value>
  - Se accede por Name[key]
  - Key debe ser tipo primitivo
- Union
  - Puede variar el tipo de dato que contiene pero en un momento determinado solo puede tener exactamente uno de los tipos definidos

Mapeo HQL a MapReduce: Join

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

X

user

userid	age	gender
111	25	female
222	32	male

=

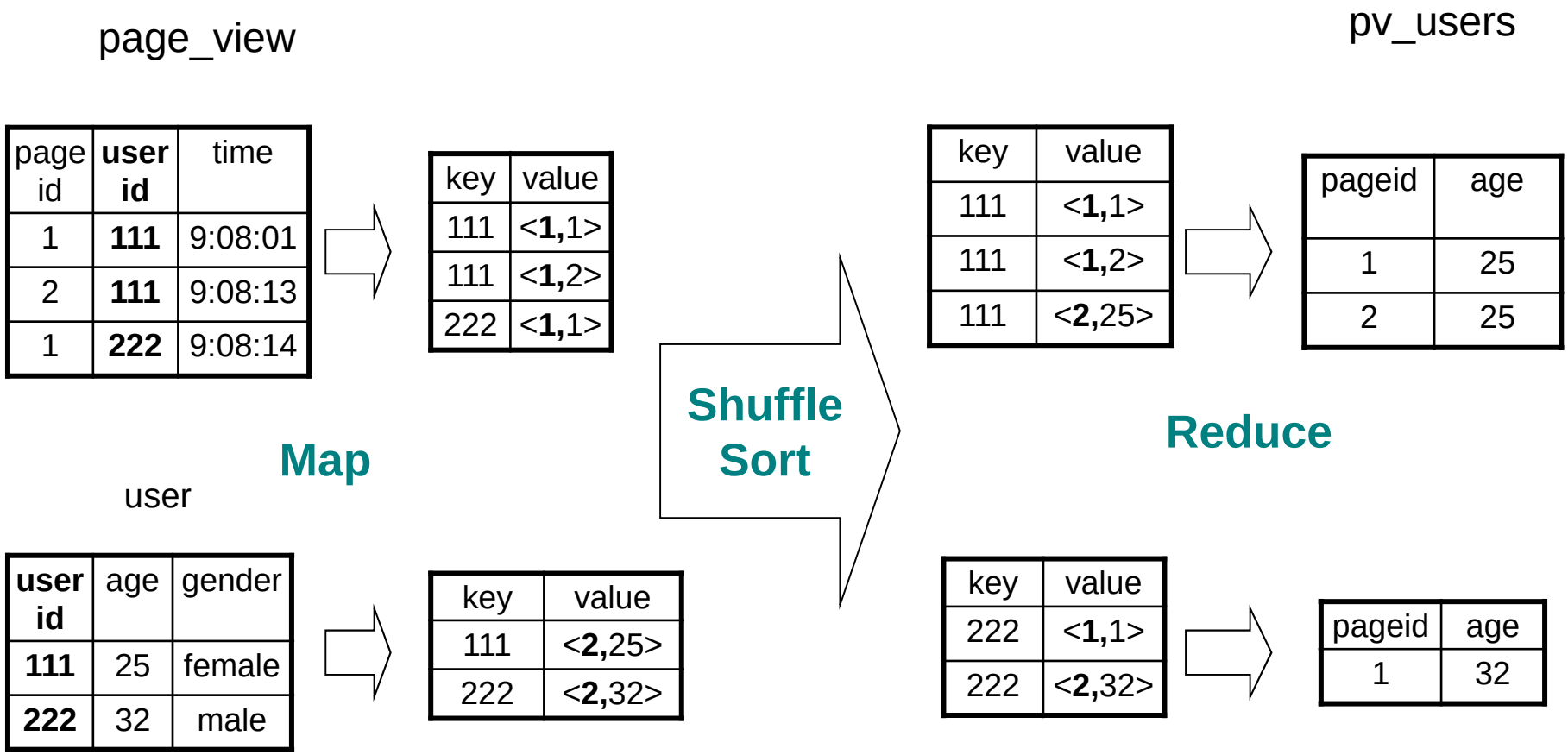
pv_users

pageid	age
1	25
2	25
1	32

HQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

Mapeo HQL a MapReduce: Join



Incremento en desempeño Hive

- How to optimize Hive Performance (www.bigdataanalust.in)
  - Cost based optimization – Optimiza plan de ejecución https://es.hortonworks.com/blog/hive-0-14-cost-based-optimizer-cbo-technical-overview/
  - Privilegia Tez sobre MapReduce como ambiente de ejecución
  - Skewed tables – separa valores que ocurren con mucha frecuencia en una tabla separada
  - Vectorización – Permite combinar varias filas en una sola
  - Comprime con ORC_table
  - Usa SORT by en vez de ORDER by
  - En JOIN, pon tabla grande a la derecha y usa Map_side join
  - Almacena bloques comprimidos de 256 MB (o el tamaño de los fragmentos de HDFS
- Practical Hive (http://www.apress.com/jp/book/9781484202722)
- 10 ways to optimize Hive queries http://sanjivblogs.blogspot.mx/2015/05/10-ways-to-optimizing-hive-queries.html

Apache Hive vs Apache Pig

- Lenguaje declarativo tipo SQL (HiveQL)
- Más apropiado para datos estructurados
- Fácil de utilizar, sobre todo para los familiarizados con SQL para la generación de reportes
- Tareas de Datawarehouse

- Lenguaje procedural para describir flujos de procesos (Pig Latin)
- Más apropiado para datos semi-estructurados
- Creación de *pipelines* de datos
  - Desarrolladores tienen puntos de validación
- Cambios incrementales