Hadoop

Tahar Kechadi
School of Computer Science

Outline

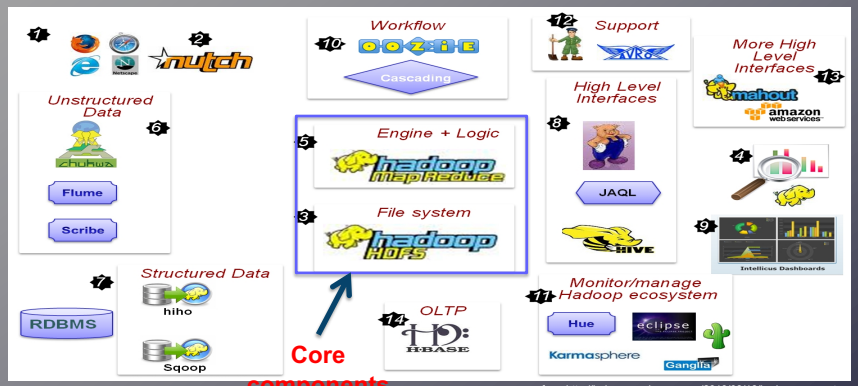
- **Introduction to Hadoop**

- **Pig**, a high-level abstraction layer

  Create MapReduce programs used with Hadoop

Hadoop & the ecosystem

- Hadoop is an open-source implementation of MapReduce
  - September 2007 – release 0.14.1
  - Latest stable release is 2.8.1 (3.0.0-beta1 -> 03/10/2017)



from http://indoos.wordpress.com/2010/08/16/hadoop-ecosystem-world-map/

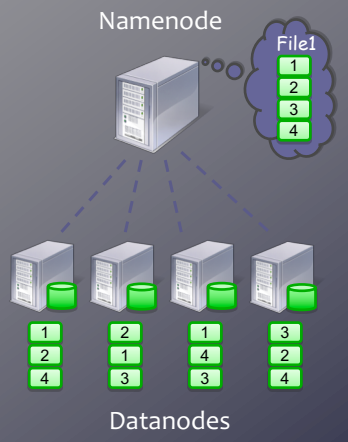
Hadoop & the ecosystem

- The Hadoop ecosystem is quite large and growing fast
- There is already an explosion of business-focused applications and platforms using Hadoop, including:

Datameer, cloudera, CONCURRENT, Appistry, Karmasphere, MicroStrategy Best In Business Intelligence, ZEMENTIS Adaptive Decision Technology, talend open data solutions, Kitenga Reinventing Information, wibidata, REVOLUTION ANALYTICS, CETAS, DIGITAL REASONING

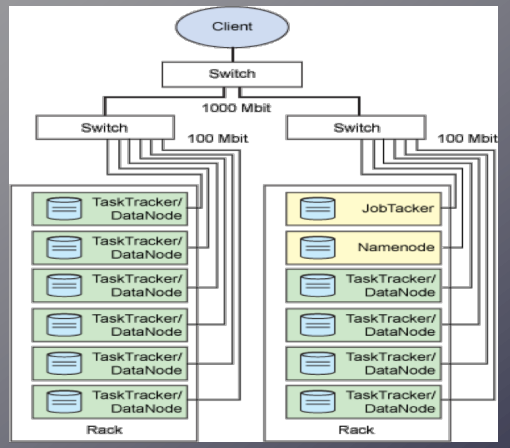
HDFS

- Highly scalable and fault-tolerant
- Blocks replicated across several *datanodes* (usually 3+)
- Single *namenode* stores metadata (file names, block locations, etc.).
- Optimised for large files, sequential reads
- Files written once (no append)
- Files split into 64MB chunks (typical size)



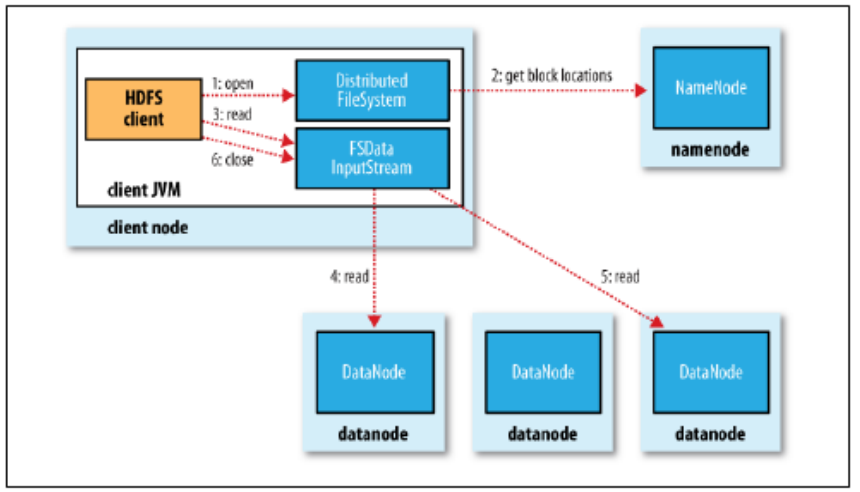
General architecture

- MapReduce layer
  - JobTracker
  - TaskTrackers
- HDFS layer
  - Namenode
  - Datanode



Example of a physical distribution within a hadoop cluster

Reading data



HDFS

- Advantages
  - Very large files
  - Streaming data access
  - Commodity hardware
  - Fault-tolerance
- Disadvantages
  - Low-latency data access
  - Lots of small files
  - Multiple writers

Getting Started…

- Download the raw Apache version, or one of the numerous existing distributions
  - hadoop.apache.org
  - www.cloudera.com - A set of VMs is also provided
  - http://www.karmasphere.com/
- Three ways to write jobs:
  - Java API
  - Hadoop Streaming (for Python, Perl, etc.)
  - Pipes API (C++)

Hadoop Setup

- Prerequisite: Java
- Create Hadoop Group and User
- Setup ssh certificate
- Setup hadoop
- Setup Hadoop Environment Variables
- Configure Hadoop
- Format namenode
- Start Hadoop service

File operations

- Basic commands
  - Syntax: hadoop fs -*cmd* <*args*>
  - Ex:

```
hadoop  fs  -ls
hadoop  fs  -mkdir  /users/data
hadoop  fs  -put  log  /users/data
hadoop  fs  -get  log  /users/data
hadoop  fs  -rm   /users/data/log
```

Hadoop Data Types

Class	Description
BooleanWritable	Wrapper for a standard Boolean variable
ByteWritable	Wrapper for a single byte
DoubleWritable	Wrapper for a Double
FloatWritable	Wrapper for a Float
IntWritable	Wrapper for a Integer
LongWritable	Wrapper for a Long
Text	Wrapper to store text using the UTF8 format
NullWritable	Placeholder when the key or value is not needed

Word Count in Java

```
public static void main(String[] args) throws Exception {
   JobConf conf = new JobConf(WordCount.class);
   conf.setJobName("wordcount");

   conf.setMapperClass(MapClass.class);
   conf.setCombinerClass(ReduceClass.class);
   conf.setReducerClass(ReduceClass.class);

   FileInputFormat.setInputPaths(conf, args[0]);
   FileOutputFormat.setOutputPath(conf, new Path(args[1]));

   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);

   JobClient.runJob(conf);
 }
```

Word Count in Java – mapper

```
public class MapClass extends MapReduceBase
   implements Mapper<LongWritable, Text, Text, IntWritable> {

   private final static IntWritable ONE = new IntWritable(1);

   public void map(LongWritable key, Text value,
                   OutputCollector<Text, IntWritable> out,
                   Reporter reporter) throws IOException {
     String line = value.toString();
     StringTokenizer itr = new StringTokenizer(line);
     while (itr.hasMoreTokens()) {
       out.collect(new text(itr.nextToken()), ONE);
     }
   }
 }
```

Word Count in Java – reducer

```
public class ReduceClass extends MapReduceBase
   implements Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterator<IntWritable> values,
                      OutputCollector<Text, IntWritable> out,
                      Reporter reporter) throws IOException {
     int sum = 0;
     while (values.hasNext()) {
       sum += values.next().get();
     }
     out.collect(key, new IntWritable(sum));
   }
 }
```

Hadoop Streaming

Mapper.py:

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
  for word in line.split(): print (word, 1)
```

Reducer.py:

```
#!/usr/bin/env python
import sys
dict={}
for line in sys.stdin:
  word, count = line.split()
  if word in dict.keys(): dict[word] += int(count)
  else: dict[word] = 1
for word in dict.keys():
  print (word, dict[word])
```

You can locally test your code on the command line:

```
$> cat data | mapper | sort | reducer
```

Hadoop Streaming Example

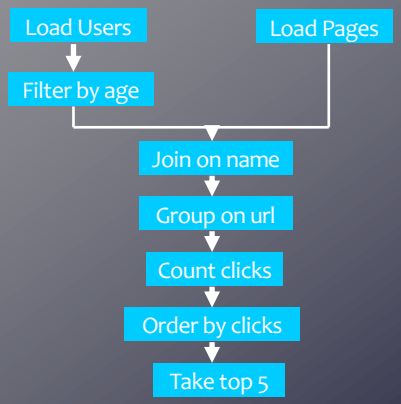
```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar
-file /home/hduser/mapper.py    -mapper /home/hduser/mapper.py \
-file /home/hduser/reducer.py   -reducer /home/hduser/reducer.py \
-input /user/hduser/gutenberg/* -output /user/hduser/gutenberg-output
```

High-level tools

- MapReduce is fairly low-level: must think about keys, values, partitioning, etc.
- Many parallel algorithms can be expressed by a series of MapReduce jobs
  - Can we capture common 'job building blocks'?
- Different use cases require different tools as well

Example

Let's find the top 5 most visited pages by users aged 18 – 25. Input: user data file, and page view data file.



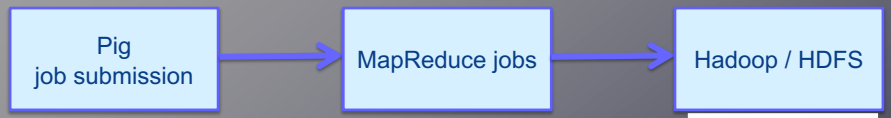
In MapReduce!

```
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.JobC
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" +
            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.
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
            }

            // 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, 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 k
                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
            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
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pag
        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.cla
        FileInputFormat.addInputPath(lfu, new
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_us
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputForm
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(
        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.
            Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFor
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.cl
        group.setOutputFormat(SequenceFileOutputF
        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.cl
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFo
        top100.setOutputKeyClass(LongWritable.cla
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutput
        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
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Pig

A high-level programming interface

- Apache Pig
  - is a platform raising a level of abstraction for processing large datasets. Its language, Pig Latin is a simple query algebra expressing data transformations and applying functions to records



- Started at Yahoo! Research,
  - More than 60% of Hadoop jobs within Yahoo! are Pig jobs

Motivations

- MapReduce requires a Java programmer
  - Solution was to abstract it and create a system where users are familiar with scripting languages
- Other than very trivial applications, MapReduce requires multiple stages, leading to long development cycles
  - Rapid prototyping. Increased productivity
- In MapReduce users have to reinvent common functionality (join, filter, etc.)
  - Pig provides them

Used for

- Rapid prototyping of algorithms for processing large datasets
- Log analysis
- Ad hoc queries across various large datasets
- Analytics (including through sampling)
- **Pig Mix** provides a set of performance and scalability benchmarks. Currently 1.1 times MapReduce speed.

Using Pig

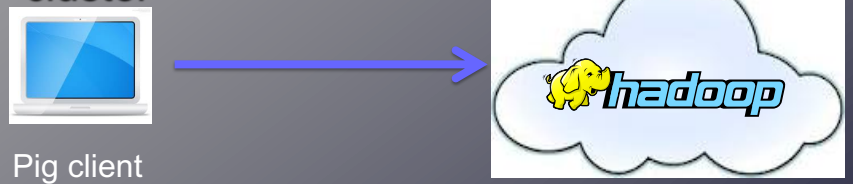
- Grunt, the Pig shell
- Executing scripts directly
- Embedding Pig in Java (using PigServer, similar to SQL using JDBC), or Python
- A range of tools including Eclipse plug-ins
  - PigPen, Pig Editor…

Execution modes

- Pig has two execution types or modes: local mode and Hadoop mode
- *Local*
  - Pig runs in a single JVM and accesses the local file system. Starting form v0.7 it uses the Hadoop job runner.
- *Hadoop mode*
  - Pig runs on a Hadoop cluster (you need to tell Pig about the version and point it to your Namenode and Jobtracker

Running Pig

- Pig resides on the user's machine and can be independent from the Hadoop cluster
- Pig is written in Java and is portable
  - Compiles into map reduce jobs and submit them to the cluster
- No need to install anything extra on the cluster



Pig client

How does it work?

- Pig defines a **DAG**. A step-by-step set of operations, each performing a transformation
- Pig defines a logical plan for these transformations:

```
A = LOAD 'file' as (line);
B = FOREACH A GENERATE
FLATTEN(TOKENIZE(line)) AS
word;
C = GROUP B BY word;
D = FOREACH C GENERATE
group, COUNT(words);
STORE D INTO 'output'
```

- Parses, checks, & optimises
- Plan the execution
  - Maps & Reduces
- Passes the jar to Hadoop
- Monitor the progress

Operators

- Read/Write operators
  - LOAD:
    - alias = LOAD 'file' [USING function] [AS schema];
  - LIMIT
    - alias = LIMIT alias n;
  - DUMP
    - DUMP alias;
  - STORE
    - STORE alias INTO 'directory' [USING function];
  - PigStorage: parses a line of input into fields using a character delimiter
  - BinStorage: loads and stores data in machine-readable format
  - TextLoader: Loads unstructured data in UTF-8 format
  - …

Read/Write Example

```
grunt> a = LOAD 'A' using PigStorage(',') AS
                        (a1, a2, a3);
grunt> b = LOAD 'B' using PigStorage(',') AS
                        (b1, b2, b3);
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
```

Read/Write Example (cont.)

- *tutorial/data/excite-small.log (~4500)*

```
3F8AAC2372F6941C    970916093724    minors in possession
C5460576B58BB1CC    970916194352    hacking telenet
9E1707EE57C96C1E    970916073214    buffalo mob crime family
06878125BE78B42C    970916183900    how to make ecstacy
```

```
grunt> log  = LOAD 'tutorial/data/excite-small.log' AS (user, time, query);
grunt> lmt = LIMIT log 4;
grunt> DUMP lmt;
(2A9EABFB35F5B954,970916105432L,+md foods +proteins)
(BED75271605EBD0C,970916001949L,yahoo chat)
(BED75271605EBD0C,970916001954L,yahoo chat)
(BED75271605EBD0C,970916003523L,yahoo chat)
```

grunt> STORE lmt INTO 'log4';

Data types & expressions

- Scalar type:
  - int, long, float, double, chararray, bytearray
- Complex type representing nested structures:
  - Tuple: sequence of fields of any type
    - Ex: (1.2, apple, 10)
  - Bag: an unordered collection of tuples
    - Ex: {(1.2, apple, 10), (3.2, orange, 5)}
  - Map: a set of key-value pairs. Keys must be atoms, values may be any type

Read/Write Example

```
grunt> a = load 'A'  as (a1:int,  a2:int,  a3:int);
grunt> b = load 'B' as (b1:int,  b2:int,  b3:int);
grunt> DUMP  a;
(0,1,2)
(1,3,4)
grunt> DUMP  b;
(0,5,2)
(1,7,8)
```

```
grunt> log  = LOAD 'tutorial/data/excite-small.log'
            ➥ AS (user:chararray, time:long, query:chararray);
```

Schemas

- Schemas enable you to associate names and types of the fields in the relation
- Schemas are optional but recommended whenever possible; type declarations result in better parse-time error checking and more efficient code execution
- They are defined using the AS keyword with operators
- Schema definition for simple data types:

```
> records = LOAD 'input/data' AS (id:int, date:chararray);
```

Statements and aliases

- Each statement, defining a data processing operator / relation, produces a dataset with an alias

```
grunt>records = LOAD 'input/data' AS (id:int, date:chararray);
```

- LOAD returns a tuple, which elements can be referenced by name
- Very useful operators (diagnostic) are DUMP, ILLUSTRATE, and DESCRIBE

```
grunt> DESCRIBE log;
log: {user: chararray,time: long,query: chararray}
```

Filtering data

- FILTER is user to work with tuples and rows of data
- Select data you want, or remove the data you are not interested in
- Filtering early in the processing pipeline minimises the amount of data flowing through the system, which can improve efficiency

```
grunt>filtered_records = FILTER records BY id == 234;
```

Expressions

- Used in Pig as a part of a statement:
  - field name: users, ipaddress, date
  - arithmetic: + - * / %
  - conditional: if , (x ? y : z)
  - comparison: < > >= <= == !=
  - Boolean: and or not
  - Pattern matching: x matches, regex
    - Ex: REGEX_EXTRACT('192.168.1.5:8020', '(.*):(.*)', 1)
      (\\d+\\.\\d+\\.\\d+)
    - logs_base = FOREACH raw_logs GENERATE FLATTEN (REGEX_EXTRACT_ALL(line,'^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] "(.+?)" (\\S+) (\\S+) "([^"]*)" "([^"]*)"')) AS (remoteAddr: chararray, remoteLogname: chararray, user: chararray, time: chararray, request: chararray, status: int, bytes_string: chararray, referrer: chararray, browser: chararray);
  - etc.

FOREACH ... GENERATE

- `FOREACH .. GENERATE` acts on columns on every row in a relation

```
grunt> ids = FOREACH records GENERATE id;
```

- ***Positional reference***. This statement has the same output

```
grunt> ids = FOREACH records GENERATE $0;
```

- The elements of 'ids' however are not named 'id' unless you add 'AS id' at the end of your statement

```
grunt> ids = FOREACH records GENERATE $0 AS id;
```

Grouping

- `GROUP .. BY` makes an output bag containing grouped fields with the same schema using a grouping key

```
grunt> log  = LOAD 'tutorial/data/excite-small.log'
           ➥ AS (user:chararray, time:long, query:chararray);
grunt> grpd = GROUP log BY user;
```

```
grunt> log  = LOAD 'tutorial/data/excite-small.log'
           ➥ AS (user:chararray, time:long, query:chararray);
grunt> grpd = GROUP log BY user;
grunt> cntd = FOREACH grpd GENERATE group, COUNT(log);
grunt> STORE cntd INTO 'output';
```

ILLUSTRATE

- Sample run to show a step-by-step process on how Pig would compute the relation

```
grunt> ILLUSTRATE cntd;
```

```
-------------------------------------------------------------------------------
| log     | user: chararray  | time: long    | query: chararray    |
-------------------------------------------------------------------------------
|         | 0567639EB8F3751C | 970916161410  | "conan o'brien"     |
|         | 0567639EB8F3751C | 970916161413  | "conan o'brien"     |
|         | 972F13CE9A8E2FA3 | 970916063540  | finger AND download |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
| grpd    | group: chararray | log: bag({user: chararray,time: long,      |
|         |                  |           query: chararray})               |
-------------------------------------------------------------------------------
|         | 0567639EB8F3751C | {(0567639EB8F3751C, 970916161410,          |
|         |                  |   "conan o'brien"),                        |
|         |                  |  (0567639EB8F3751C,970916161413,           |
|         |                  |   "conan o'brien")}                        |
|         | 972F13CE9A8E2FA3 | {(972F13CE9A8E2FA3, 970916063540,          |
|         |                  |   finger AND download)}                    |
-------------------------------------------------------------------------------

-----------------------------------
| cntd    | group: chararray | long |
-----------------------------------
|         | 0567639EB8F3751C | 2    |
|         | 972F13CE9A8E2FA3 | 1    |
-----------------------------------
```

Joining

- **JOIN** performs inner of two or more relations based on common field values.
- You can also perform outer joins using keywords **left**, **right** and **full**

```
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
```

```
grunt> j = JOIN a BY $2, b BY $2;
grunt> dump j;
(0,1,2,0,5,2)
```

Combining, splitting…

- the UNION operator to merge the contents of two or more relations
- SPLIT partitions a relation into two or more relations

```
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
```

```
grunt> c = UNION a, b;
grunt> DUMP c;
(0,1,2)
(0,5,2)
(1,3,4)
(1,7,8)
```

```
grunt> SPLIT c INTO d IF $0 == 0, e IF $0 == 1;
grunt> DUMP d;
(0,1,2)
(0,5,2)
grunt> DUMP e;
(1,3,4)
(1,7,8)
```

Ordering, Sampling…

- ORDER imposes an order on the output to sort a relation by one or more fields
- The LIMIT statement limits the number of results
- the SAMPLE operator selects a random data sample with the stated sample size

```
grunt>sample_records = SAMPLE records 0.01;
```

Functions

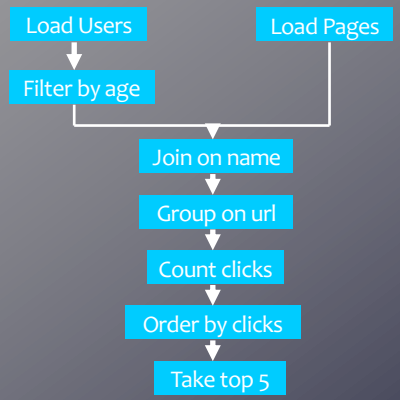
- Filter
  - A special type of eval function used by the FILTER operator. IsEmpty is a built-in function
- Evaluation
  - Many built-in functions MAX, COUNT, SUM, DIFF, SIZE…
- Comparison
  - Function used in ORDER statement; ASC | DESC

Built-in Functions

AVG	Calculate the average of numeric values in a single-column bag.
CONCAT	Concatenate two strings (`chararray`) or two `bytearrays`.
COUNT	Calculate the number of tuples in a bag. See SIZE for other data types.
DIFF	Compare two fields in a tuple. If the two fields are bags, it will return tuples that are in one bag but not the other. If the two fields are values, it will emit tuples where the values don't match.
MAX	Calculate the maximum value in a single-column bag. The column must be a numeric type or a `chararray`.
MIN	Calculate the minimum value in a single-column bag. The column must be a numeric type or a `chararray`.
SIZE	Calculate the number of elements. For a bag it counts the number of tuples. For a tuple it counts the number of elements. For a `chararray` it counts the number of characters. For a `bytearray` it counts the number of bytes. For numeric scalars it always returns 1.
SUM	Calculate the sum of numeric values in a single-column bag.
TOKENIZE	Split a string (`chararray`) into a bag of words (each word is a tuple in the bag). Word separators are space, double quote ("), comma, parentheses, and asterisk (*).
IsEmpty	Check if a bag or map is empty.

An Example

Let's find the top 5 most visited pages by users aged 18 – 25. Input: user data file, and page view data file.



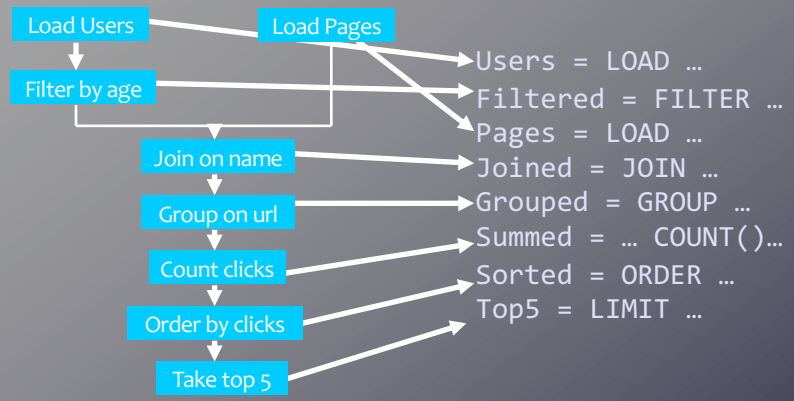
In MapReduce!

```
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.JobC
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
            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.to
                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, 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 k,
                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_pag
        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.cla
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
                new Path("/user/gates/tmp/filtered_us
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputForm
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap
        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(
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFor
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.cl
        group.setOutputFormat(SequenceFileOutput
        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.cl
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFo
        top100.setOutputKeyClass(LongWritable.cla
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutput
        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
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

A simple script


```
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';
```

Ease of Translation



Stream

- The `Stream` operator allows to transform data in a relation using an external program or script

```
grunt> C = STREAM A THROUGH `cut -f 2`;
```

  - Extract the second field of A using `cut`

- The scripts are shipped to the cluster using

```
grunt> DEFINE script `script.py` SHIP ('script.py');
grunt> D = STREAM C THROUGH script AS (…);
```

User defined functions

- Support and a community of user-defined functions (UDFs)
- UDFs can encapsulate users processing logic in filtering, comparison, evaluation, grouping, or storage
  - `filter` functions for instance are all subclasses of `FilterFunc`, which itself is a subclass of `EvalFunc`
- **PiggyBank**: the Pig community sharing their UDFs
- **DataFu**: Linkedin's collection of Pig UDFs

Hive

- Initially developed at Facebook
- http://hive.apache.org/
- 'Relational database' built on top of Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HQL)
  - Can call Hadoop Streaming scripts from HQL
  - Supports table partitioning, clustering, complex data types, …

Example of Hive Queries

- Find top 5 pages visited by users aged 18-25:

```
SELECT p.url, COUNT(1) as clicks
FROM users u JOIN page_views p ON (u.name = p.user)
WHERE u.age >= 18 AND u.age <= 25
GROUP BY p.url
ORDER BY clicks
LIMIT 5;
```

- Filter page views through Python script:

```
SELECT TRANSFORM(p.user, p.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views p;
```

Workflow

- A workflow system provides an infrastructure to set up & manage a sequence of interdependent jobs / set of jobs
- The hadoop ecosystem includes a set of workflow tools to run applications over MapReduce processes or High-level languages
  - Cascading (http://www.cascading.org/). A java library defining data processing workflows and rendering them to MapReduce jobs
  - Oozie (http://yahoo.github.com/oozie/)
  - …

Some interesting links

- Interesting papers:
  - http://bit.ly/rskJho - Original MapReduce paper
  - http://bit.ly/KvFXxT - Pig paper: 'Building a High-Level Dataflow System on top of MapReduce: The Pig Experience'
- http://code.google.com/edu/parallel/mapreduce-tutorial.html
- http://wiki.apache.org (Hadoop and related projects pages)
- http://hadoop.apache.org
- http://pig.apache.org/
- https://cwiki.apache.org/confluence/display/PIG/Index
- PiggyBank: https://cwiki.apache.org/confluence/display/PIG/PiggyBank
- DataFu: https://github.com/linkedin/datafu
- Pygmalion: https://github.com/jeromatron/pygmalion
- http://code.google.com/edu/parallel/mapreduce-tutorial.html
- Video tutorials from Cloudera: http://www.cloudera.com/hadoop-training

A simple eval UDF example

```
package myudfs;

import …

public class UPPER extends EvalFunc<String>
{
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try{
            String str = (String) input.get(0);
            return str.toUpperCase();
        }catch(Exception e){
            throw WrappedIOException.wrap("Caught exception processing input row ",
e);
        }
    }
}
```