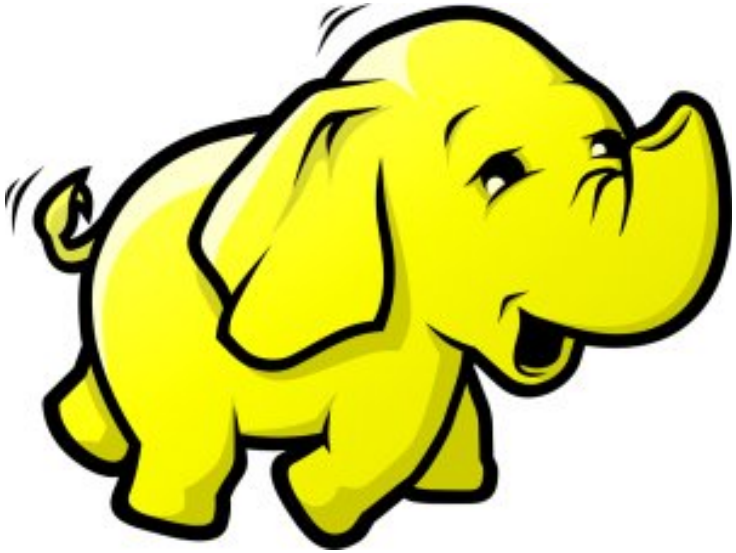
# Extreme Computing

## Hadoop

Stratis Viglas

School of Informatics
University of Edinburgh
sviglas@inf.ed.ac.uk

# Hadoop



Overview

Examples

Environment

# History

*Nutch* started in 2002 by Doug Cutting and Mike Cazfarella

- Early open-source web-search engine
- Written in Java
- Realisation that it would not scale for the Web
  - 2004: Google MR and GFS papers appeared
  - 2005: Working MR implementation for Nutch
  - 2006: Hadoop became standalone project
- 2008: Hadoop broke world record for sorting 1TB of data

# Overview

Set of components (Java), implementing most of MR-related ecosystem:

- MapReduce
- HDFS (Hadoop distributed filesytem)
- Services on top:
  - HBase (BigTable)
  - Pig (sql-like job control)
- Job-control

# Overview

Hadoop supports a variety of ways to implement MR jobs:

- Natively, as java
- Using the 'streaming' interface
  - Mappers and reducers can be in any language
  - Performance penalty, restricted functionality
- C++ hooks etc

We will use the Streaming interface

# Aside: Map-Reduce in one line

Under Unix, you can quickly test a MR job:

**% cat input | mapper | sort -0 +1 | reducer > output**

**mapper** is your Mapper and **reducer** is the Reducer

# Word Counting

Word counting using Hadoop:

- Use HDFS
- Specify the MR program
- Run the job

Note: all commands are for Hadoop 0.19

# Word Counting

First need to upload data to HDFS

- Hadoop has a set of filesystem-like commands to do this:
  - **hadoop dfs -mkdir data**
  - **hadoop dfs -put file.text data/**
- This creates a new directory and uploads the file **file.txt** to HDFS
- We can verify that it is there:
  - **hadoop dfs -ls data/**

# Word Counting

Mapper:

- Using Streaming, a Mapper reads from STDIN and writes to STDOUT
- Keys and Values are delimited (by default) using tabs.
- Records are split using newlines

# Word Counting

Mapper:

```
1 while !eof(STDIN) do
2 |   line = readLine(STDIN)
3 |   wordList = split(line)
4 |   foreach word in wordList do
5 |   |   print word TAB 1 NEWLINE
6 |   end
7 end
```

# Word Counting

Reducer

```
1  prevWord = "" ; valueTotal = 0
2  while !eof(STDIN) do
3      line = readLine(STDIN); (word,value) = split(line)
4      if word eq prevWord or prevWord eq "" then
5          valueTotal += value
6          prevWord = word
7      else
8          print prevWord valueTotal NEWLINE
9          prevWord = word; valueTotal = value
10     end
11 end
12 print word valueTotal NEWLINE
```

# Word Counting

## Improving the Mapper

```
1  wordsCounts = {}
2  while !eof(STDIN) do
3  |   line = readLine(STDIN)
4  |   wordList = split(line)
5  |   foreach word in wordList do
6  |   |   wordCounts{word}++
7  |   end
8  end
9  foreach word in keys(wordCounts) do
10 |   count = wordCounts{word}
11 |   print word TAB count NEWLINE
12 end
```

# Word Counting

Our improved Mapper:

- Only emits one word-count pair, per word and shard
- This uses a *Combiner* technique
- Uses an unbounded amount of memory

Word counting 2 million tokens (Unix MR simulation)

| Mapper | Time |
|---|---|
| Naive | 1 minute 5 sec |
| Combiner | 10 sec |

How can you change it to use a *bounded* amount of memory?

# Secondary Sorting

At times, we may want to resort the Reducer input:

- Hadoop only guarantees that the same keys are grouped together
- We may want to ensure that some key occurs before other ones

A *secondary sort can rearrange the Reducer input*

# Secondary Sorting

Suppose we want to join two databases (White, p 234):

- Records in the DBs share the same set of IDs
- Want to merge records X in DB1 with records X in DB2
- All items in DB1 must be before DB2

# Secondary Sorting

DB1

| Station ID | Station Name |
|---|---|
| A1 | Zebra |
| A2 | Goat |

DB2

| Station ID | Time | Temperature |
|---|---|---|
| A1 | 10:00 | 12 |
| A1 | 11:00 | 11 |
| A1 | 12:00 | 9 |
| A2 | 10:00 | 12 |

Combined:

| Station ID | Time | Station Name | Temperature |
|---|---|---|---|
| A1 | 10:00 | Zebra | 12 |
| A1 | 11:00 | Zebra | 11 |
| A1 | 12:00 | Zebra | 9 |
| A2 | 10:00 | Goat | 12 |

# Secondary Sorting

Approach:

- Key mapper output by record ID
- Assign a secondary key to each output
- Resort output by first and second key
  - All records with the same ID are still together
  - Now, DB1 can be before DB2

# Running a job

To run a streaming job:

```
hadoop jar \
contrib/streaming/hadoop-*-streaming.jar \
-mapper map -reducer red -input in \
-output out -file map -file red
```

- **map** and **red** are the actual MR program files
- **in** is the input DFS and **out** is the output DFS directory

# Job Control

There is a graphical UI:

**http://crom.inf.ed.ac.uk:50030/jobtracker.jsp**

And also a command-line interface:

- **hadoop job -list**
- **hadoop job -status jobid**
- **hadoop job -kill jobid**

# Summary

- History
- Looked at major components
- Two examples
- Job control