Hadoop Streaming: An Introduction

Elia Bruni
Claudio Martella

9 November 2011
Hadoop Streaming

- HS is a utility that with the Hadoop distribution
- HS allows you to use arbitrary programs for the Mapper and Reducer phases of a MapReduce job
- Both Mappers and Reducers receive their input on stdin and emit output (key, value) pairs on stdout
- Input and output are always represented textually in Streaming
Python MapReduce code

- We use python's `sys.stdin` to read input data and print our own output to `sys.stdout`
- That's all we need to do because HadoopStreaming will take care of everything else!
Hadoop streaming

- Basically, write some code that runs like:

  \$ cat data | mapper.py | sort | reducer.py
Ex. 1: wordcount

- We write a simple **MapReduce** program for Hadoop in **Python**
- Our program reads text files and counts how often words occur
- The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab
Mapper

- It will read data from STDIN, split it into words and output a list of lines mapping words to their (intermediate) counts to STDOUT
- The Map script will not compute an (intermediate) sum of a word’s occurrences
- Instead, it will output “<word> 1” immediately – even though the <word> might occur multiple times in the input – and just let the subsequent Reduce step do the final sum count
Reducer

- It will read the results of the Mapper from STDIN, and sum the occurrences of each word to a final count, and output its results to STDOUT
Pseudo-code

1: class Mapper
2:    method MAP(docid a, doc d)
3:        for all term t ∈ doc d do
4:            Emit(term t, count 1)

1: class Reducer
2:    method REDUCE(term t, counts [c₁, c₂, ...])
3:        sum ← 0
4:        for all count c ∈ counts [c₁, c₂, ...] do
5:            sum ← sum + c
6:        Emit(term t, count sum)
Ex. 2: co-occurrences

- We want to build a word co-occurrence matrix from a corpus where
  - The co-occurrence matrix is a square $n \times n$ matrix where $n$ is the number of unique words in the corpus (i.e., the vocabulary size)
  - A cell $m_{ij}$ contains the number of times word $w_i$ co-occurs with $w_j$ within a specific context – a certain window of $m$ words in our case
Mapper

- It will read data from STDIN, split it into words and output a list of lines mapping pair of words to their (intermediate) counts to STDOUT
- The Map script will **not** compute an (intermediate) sum of a word’s co-occurrences
- Instead, it will output “<pair> 1” immediately – even though the <pair> might occur multiple times in the input – and just let the subsequent Reduce step do the final sum count
Reducer

- It will read the results of the Mapper from STDIN, and sum the co-occurrences of each pair to a final count, and output its results to STDOUT
Pseudo-code

1: class Mapper
2:     method MAP(docid $a$, doc $d$)
3:         for all term $w \in$ doc $d$ do
4:             for all term $u \in$ NEIGHBORS($w$) do
5:                 EMIT(pair $(w, u)$, count 1) \(\triangleright\) Emit count for each co-occurrence

1: class Reducer
2:     method REDUCE(pair $p$, counts $[c_1, c_2, \ldots]$)
3:         $s \leftarrow 0$
4:         for all count $c \in$ counts $[c_1, c_2, \ldots]$ do
5:             $s \leftarrow s + c$ \(\triangleright\) Sum co-occurrence counts
6:         EMIT(pair $p$, count $s$)
Run

- $ cat data | mapper.py | sort | reducer.py
- If that works, run it live
  - Put the data into hadoop's Distributed File System (DFS)
  - Run hadoop
  - Read the output data in the DFS
Run Hadoop

Stream data through these two files, saving the output back to HDFS:

```
$HADOOP_HOME/bin/hadoop jar \n$HADOOP_HOME/hadoop-streaming.jar \n  -input input_dir \n  -output output_dir \n  -mapper mapper.py \n  -reducer reducer.py \n  -file mapper.py -file reducer.py
```
View output

- View output files:
  - `$ hadoop dfs -ls output_dir`

- Note multiple output files ("part-00000", "part-00001", etc)

- View output file contents:
  - `$ hadoop dfs -cat output_dir/part-00000`
A more complicated problem: Eva

- Compute cosine similarity between each pair of vectors of two matrices
- Given two vectors A and B, the cosine similarity is represented using a dot product and magnitude as

\[
\frac{\sum_{i=1}^{n} a_i \times b_i}{\sqrt{\sum_{i=1}^{n} a^2} \times \sqrt{\sum_{i=1}^{n} b^2}}
\]

- In python:
  - \( c = \text{dot}(v,w) / (\text{norm}(v) \times \text{norm}(w)) \)
# The two matrices

- **reduced.matrix**
  - `A-level-n \ -148.59 \ 053.17 ... 0.46`
  - `zoom-n \ -081.18 \ 038.22 ... -2.61`

- **add_an-reduced.matrix**
  - `hot-j_blood-n \ -839.47 \ -106.30 ... -0.22`
  - `hot-j_woman-n \ -972.28 \ -172.25 ... 1.19`
Hints

- The problem is fully parallelizable: you don't need a Reducer.
- To simplify, add_an-reduced.matrix is small and can be kept in memory, while reduced.matrix not (not a necessary step, but useful to win the competition!)
Presentation and code

- https://github.com/eliabruni/hadoop-tutorial
Have fun!