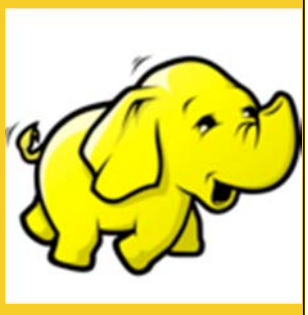
Hadoop Streaming

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

© 2012 coreservlets.com and Dima May



Agenda

- **Implement a Streaming Job**
- **Contrast with Java Code**
- **Create counts in Streaming application**

Hadoop Streaming

- **Develop MapReduce jobs in practically any language**
- **Uses Unix Streams as communication mechanism between Hadoop and your code**
  - Any language that can read standard input and write standard output will work
- **Few good use-cases:**
  - Text processing
    - scripting languages do well in text analysis
  - Utilities and/or expertise in languages other than Java

Streaming and MapReduce

- **Map input passed over standard input**
- **Map processes input line-by-line**
- **Map writes output to standard output**
  - Key-value pairs separate by tab (‘\t’)
- **Reduce input passed over standard input**
  - Same as mapper output – key-value pairs separated by tab
  - Input is sorted by key
- **Reduce writes output to standard output**

Implementing Streaming Job

1. **Choose a language**
   - Examples are in Python
2. **Implement Map function**
   - Read from standard input
   - Write to standard out - keys-value pairs separated by tab
3. **Implement Reduce function**
   - Read key-value from standard input
   - Write out to standard output
4. **Run via Streaming Framework**
   - Use $yarn command

1: Choose a Language

- **Any language that is capable of**
  - Reading from standard input
  - Writing to standard output
- **The following example is in Python**
- **Let's re-implement StartsWithCountJob in Python**

2: Implement Map Code - countMap.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token: print token[0] + '\t1'
```

1. Read one line at a time from standard input

2. tokenize

3. Emit first letter, tab, then a count of 1

3: Implement Reduce Code

- **Reduce is a little different from Java MapReduce framework**
  - Each line is a key-value pair
  - Differs from Java API
    - Values are already grouped by key
    - Iterator is provided for each key
  - You have to figure out group boundaries yourself
- **MapReduce Streaming will still sort by key**

3: Implement Reduce Code - countReduce.py

```
#!/usr/bin/python

import sys

(lastKey, sum)=(None, 0)


for line in sys.stdin:
    (key, value) = line.strip().split("\t")


    if lastKey and lastKey != key:
        print lastKey + '\t' + str(sum)
        (lastKey, sum) = (key, int(value))
    else:
        (lastKey, sum) = (key, sum + int(value))

if lastKey:
    print lastKey + '\t' + str(sum)
```

Variables to manage key group boundaries

Process one line at a time by reading from standard input

If key is different emit current group and start new

4: Run via Streaming Framework

- **Before running on a cluster it's very easy to express MapReduce Job via unit pipes**

```
$ cat testText.txt | countMap.py | sort | countReduce.py
a       1
h       1
i       4
s       1
t       5
v       1
```

- **Excellent option to test and develop**

4: Run via Streaming Framework

```
yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
       -D mapred.job.name="Count Job via Streaming" \
       -files $HADOOP_SAMPLES_SRC/scripts/countMap.py,\
                     $HADOOP_SAMPLES_SRC/scripts/countReduce.py \
       -input /training/data/hamlet.txt \
       -output /training/playArea/wordCount/ \
       -mapper countMap.py \
       -combiner countReduce.py \
       -reducer countReduce.py
```

Name the job

-files options makes scripts available on the cluster for MapReduce

Python vs. Java Map Implementation

Python

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token: print token[0] + '\t1'
```

Java

```
package mr.wordcount;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

public class StartsWithCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable countOne = new IntWritable(1);
    private final Text reusableText = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context)
                    throws IOException, InterruptedException {

        StringTokenizer tokenizer = new StringTokenizer(value.toString());
        while (tokenizer.hasMoreTokens()) {
            reusableText.set(tokenizer.nextToken().substring(0, 1));
            context.write(reusableText, countOne);
        }
    }
}
```


Python vs. Java Reduce Implementation

Python

```
#!/usr/bin/python

import sys

(lastKey, sum)=(None, 0)
for line in sys.stdin:
    (key, value) = line.strip().split("\t")

    if lastKey and lastKey != key:
        print lastKey + '\t' + str(sum)
        (lastKey, sum) = (key, int(value))
    else:
        (lastKey, sum) = (key, sum + int(value))

if lastKey:
    print lastKey + '\t' + str(sum)
```

Java

```
package mr.wordcount;

import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

public class StartsWithCountReducer extends
                Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    protected void reduce(Text token, Iterable<IntWritable> counts,
                Context context) throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable count : counts) {
            sum+= count.get();
        }
        context.write(token, new IntWritable(sum));
    }
}
```

Reporting in Streaming

- **Streaming code can increment counters and update statuses**
- **Write string to standard error in "streaming reporter" format**
- **To increment a counter:**

```
reporter:counter:<counter_group>,<counter>,<increment_by>
```


countMap_withReporting.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token:
            sys.stderr.write("reporter:counter:Tokens,Total,1\n")
            print token[0] + '\t1'
```

Print counter information in "reporter protocol" to standard error

Wrap-Up

Summary

- **Learned how to**
  - Implement a Streaming Job
  - Create counts in Streaming application
  - Contrast with Java Code

Questions?