MapReduce

Hadoop Seminar, TUT, 2014-10-22
Antti Nieminen
MapReduce

- MapReduce is a programming model for distributed processing of large data sets
- Scales ~linearly
  - Twice as many nodes -> twice as fast
  - Achieved by exploiting data locality
    - Data processing where the data is
- Simple programming model
  - Programmer only needs to write two functions: Map and Reduce
Map & Reduce

• The programmer writes two functions:
  – **Map** maps input data to key/value pairs
  – **Reduce** processes the list of values for a given key

• The MapReduce framework (such as Hadoop) does the rest
  – Distributes the job among nodes
  – Moves the data to/from nodes
  – Handles node failures
  – etc.
MapReduce

### MAP
- TRE-1, 2°C
- TKU-1, 5°C
- HKI-1, 7°C
- TRE-2, 1°C
- HKI-1, 5°C
- HKI-2, 9°C
- HKI-2, 6°C
- ...

### SHUFFLE
- TRE 2
- TKU 5
- HKI 7
- TRE 5
- HKI 5
- HKI 9
- HKI 6
- HKI 4
- HKI 4
- TRE 3
- TKU 2
- TKU 2
- TKU 8
- TKU 8
- TRE 9
- TRE 8

### REDUCE
- TRE 9
- TKU 8
- HKI 9
MapReduce

MAP

Map(k1, v1) → list(k2, v2)

REDUCE

Reduce(k2, list(v2)) → list(v3)
Map & Reduce in Hadoop

• In Hadoop, Map and Reduce functions can be written in
  – Java
    • org.apache.hadoop.mapreduce.lib
  – C++ using Hadoop Pipes
  – any language, using Hadoop Streaming
• Also a number of third party programming frameworks for Hadoop MapReduce
  – For Java, Scala, Python, Ruby, PHP, ...
  – See eg. this blog post
Mapper Java example

public class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) {
        String city = cityFromValue(value);
        int temp = tempFromValue(value);
        context.write(new Text(city), new IntWritable(temp));
    }
}

• The Mapper input types depend on the defined InputFormat
• By default TextInputFormat
  • Key (LongWritable): position in the file
  • Value (Text): the line
Reducer Java example

public class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int maxValue = Integer.MIN_VALUE;
        for (IntWritable value : values) {
            maxValue = Math.max(maxValue, value.get());
        }
        context.write(key, new IntWritable(maxValue));
    }
}"
Run MapReduce example

```java
Job job = new Job();
job.setJarByClass(MyClass.class);
job.setJobName("Max temperature");

FileInputFormat.addInputPath(job, new Path("~/input");
FileInputFormat.setOutputPath(job, new Path("~/output");

job.setMapperClass(MyMapper.class);
job.setReducerClass(MyReducer.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

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

job.waitForCompletion(true);
```
Hadoop Streaming

• Map and Reduce functions can be implemented in any language with the Hadoop Streaming API
• Input is read from standard input
• Output is written to standard output
• Input/output items are lines of the form key\tvalue
  – \t is the tabulator character
• Reducer input lines are grouped by key
  – One reducer instance may receive multiple keys
Python example

• mapper.py

import sys
for line in sys.stdin:
    city, temp = city_temp_from_line(line)
    print('%%s\t%s' % (city, temp))

• reducer.py

import sys
last_key = None
max_val = None  # < anything
for line in sys.stdin:
    key, val = line.strip().split(' \t')
    if key != last_key:
        print('%%s\t%s' % (last_key, max_val))
        max_val = None
    last_key = key
    max_val = max(val, max_val)
if last_key:
    print('%%s\t%s' % (last_key, max_val))
Run Hadoop Streaming

• Debug using Unix pipes:

```
cat sample.txt | ./mapper.py | sort | ./reducer.py
```

• On Hadoop:

```
hadoop jar $HADOOP_INSTALL/contrib/streaming/hadoop-*-streaming.jar \
  -input sample.txt \
  -output output \
  -mapper ./mapper.py \
  -reducer ./reducer.py
```
Combiners

Map node 1

Reduce node for key A

A  1
A  5
A  3
B  2
Combiners

Map node 1

Reduce node for key A

Combiner
Combiners

• Combiner can “compress” data on a mapper node before sending it forward
• Combiner input/output types must equal the mapper output types
• In Hadoop Java, Combiners use the Reducer interface

```java
job.setCombinerClass(MyReducer.class);
```
Reducer as a Combiner

• Reducer can be used as a Combiner if it is **commutative** and **associative**
  – Eg. max is
    • $\max(1, 2, \max(3, 4, 5)) = \max(\max(2, 4), \max(1, 5, 3))$
    • true for any order of function applications...
  – Eg. avg is not
    • $\avg(1, 2, \avg(3, 4, 5)) = 2.33333 \neq \avg(\avg(2, 4), \avg(1, 5, 3)) = 3$

• Note: if Reducer is not c&a, Combiners can still be used
  – The Combiner just has to be different from the Reducer and designed for the specific case
MapReduce example

• Find the number of unique purchasing locations for each product

• Data:
  – Users
    • (user id, name, location, ...)
  – Transactions
    • (transaction id, product id, product name, user id, ...)

• The example stolen from here
  – Here are some more examples...
### MAP 1

**Users**
- u1, Antti, FI
- u2, Bob, US
- u3, Carola, SE

**Transactions**
- p1, Apple, u1
- p1, Apple, u2
- p2, Banana, u2

### REDUCE 1

**MAP 1**
- u1 → p1
- u2 → p1
- u3 → p2

**REDUCE 1**
- u1 → p1, FI
- u2 → p1, US
- u3 → p2, US

### MAP 2

**Transactions**
- p1, Apple, u1
- p1, US
- p2, US

### REDUCE 2

**MAP 2**
- p1 → p1, FI
- p1 → p1, US
- p2 → p2, US

**REDUCE 2**
- p1 → p1, 2
- p2 → p2, 1
The End

Questions? Comments?