Map Reduce

Aaron Palumbo

9/6/2015

Contents

2.3.1 1
2.3.1 (a) 2
2.3.1 (b) 2
2.3.1 (c) 2
2.3.1 (d) 3
2.3.2 3

```
library(rmr2)
```

```
## Warning: S3 methods 'gorder.default', 'gorder.factor', 'gorder.data.frame',
## 'gorder.matrix', 'gorder.raw' were declared in NAMESPACE but not found
```

```
## Please review your hadoop settings. See help(hadoop.settings)
```

```
library(rhdfs)
```

```
## Loading required package: rJava
##
## HADOOP_CMD=/home/apalumbo/workspace/cuny_msda_is622/hadoop-2.7.1/bin/hadoop
##
## Be sure to run hdfs.init()
```

```
hdfs.init()
```

From Mining Massive Datasets:

- 2.3.1, 2.3.2 (in section 2.3.11)

2.3.1

Design MapReduce algorithms to take a very large file of integers and produce as output:

2.3.1 (a)

The largest integer.

We have the map job find the max value of it's chunck, then send the results of all the chuncks to a reduce job.

Map function: Take inputs and returns key=1, value=max(values)

Reduce function: All map jobs return with a key of 1. Now with the reduce function we just need to return the max of key=1.

Implementation:

```
mapper <- function(null, vec){
  return(keyval(1, max(vec)))
}

reducer <- function(key, val.list){
  return(keyval(1, max(val.list)))
}

vec <- as.integer(rnorm(1000, mean=500, sd=100))
rand.ints <- to.dfs(vec)
mr <- mapreduce(input=rand.ints, map=mapper, reduce=reducer)
result <- from.dfs(mr)

print(paste0("result from map reduce:  ", result$val))
```

```
## [1] "result from map reduce:  807"
```

```
print(paste0("Actual value:  ", max(vec)))
```

```
## [1] "Actual value:  807"
```

2.3.1 (b)

The average of all the integers.

Here we do an average in the map step, then a weighted average in the reduce step.

Map: Take inputs and return a key of 1 and a value of (w, a) where w, weight, is the count of the numbers sent to the map job and a is the average of the numbers sent to the map job.

Reduce: We have a constant key value, so now we just take our value list and do a weighted average:

$$\frac{\sum w * a}{\sum w}$$

2.3.1 (c)

The same set of integers, but with each integer appearing only once.

Here we can just use the key as a filter:

Map: Take inputs and for each input return key=input, value=input.

Reduce: The list of keys is the desired output.

2.3.1 (d)

The count of the number of distinct integers in the input file.

I'm not sure about speed tradeoffs here, but you could just pass the output of part (c) to another map reduce job:

Map: Take inputs and return a key of 1 and a value equal to the input.

Reduce: The number of distinct integers is now just a count of the values with key = 1.

2.3.2

Our formulation of matrix-vector multiplication assumed that the matrix M was square. Generalize the algorithm to the case where M is an r-by-c matrix for some number of rows r and columns c.

We will assume matrix multiplication of the form $M \cdot v$. This means that v must have c components.

We will also assume v will fit into memory. If not, it is possible to adapt this technique to work in that case as well.

Map: Take all matrix values M_{rc} as input. Map produces a key=c and a value equal to $M_{rc} * v_c$.

Reduce: Each key corresponds to a component of the vector v whose value is just the sum of the values associated with each key.