UNIVERSITY OF HELSINKI
DEPARTMENT OF COMPUTER SCIENCE

Distributed System Project Assigment two Report

Mohammad Bakharzy
ID: 014083589
February 4, 2014

1 Solution and answers for question one

1.1 Answers

Maximum:	73708965.73706365
Minimum:	10.25634718
Average:	9027230.275509289
Variance:	37567548924327.3640504616574393

Answering to this question was straightforward. I used both Java example code with modifications and Pig. Firstly, I explain how I found the answers with Pig and afterwards with Java. The Java code is available in the attachment file in q1 folder.

1.2 Pig solution

After running Pig shell command line (grunt) in mapReduce mode, I loaded the dataset to the Pig using the following command:

```
grunt> data = LOAD 'hdfs://86.50.20.81:9000/data/distr.txt' AS (index:int,value:double);
```

in which, the index is saved as Integer value and the value is saved as Double value. Then I grouped-all the data with the following command:

```
grunt> data_grouped = GROUP data ALL;
```

Now I can find MAX, MIN, COUNT and AVERAGE easily by one of the following commands:

```
grunt>  max = FOREACH data_grouped GENERATE MAX(data.value);

grunt>   min = FOREACH data_grouped GENERATE MIN(data.value);

grunt> count = FOREACH data_grouped GENERATE COUNT(data);

grunt> avg = FOREACH data_grouped GENERATE AVG(data.value);
```

Finally, by DUMP command on each of these results, the mapReduce will start calculating the results. For example, the following command and result will be given by mapReduce.

```
grunt> DUMP max;
result: (7.370896573706365E7)
```

which shows that 7.370896573706365E7 is the maximum number in the dataset. Each DUMP command took about 3 minutes for me (sometimes about 2 minutes) to finish. For Variance I did not use Pig, instead I used the Java code for Variance calculation.

1.3 JAVA SOLUTION

For Java solution, I combined my solutions to one code. Minimum and Maximum values are calculated as the example which was provided. For calculating the average, I calculated the sum of all the numbers called 'sumOfAll' in the output as well as the number counts called 'numberCount' in the output. Then, I divided the former value by the latter value and calculate the Average value which was the same as Average from Pig. For calculating Variance, I used a simple formula as below:

$$\sigma^2 = \frac{\sum_{i=1}^{N} (eachValue - avg)^2}{N}$$

I calculated the part over fraction(before dividing by N) in Java and get the output which is called 'varianceSum', then I divided it by N to find the variance value.The output of Java code (in q1 folder in attachment) is as bellow:

```
maxValue :      7.370896573706365E7
minValue :      10.25634718
numberCount :   5.36870912E8
sumOfAll :      4.846457350846692E15
varianceSum :   2.0168924215040694E22
```

I separated the combiner and reducer because for calculating Average or Variance the reducer does not respect the commutative and associative properties.

2 SOLUTION AND ANSWERS FOR QUESTION TWO

2.1 ANSWERS(WITH JAVA)

Median1: 7832134.12364261
Median2: 7832134.17029064
Median = Average(Median1, Median2) = 7832134.146966625

Algorithm: I think you already remember my basic solution algorithm for finding median without sorting the dataset. It starts with a guess(can be average or any other good guess or even does not matter) value as median. This value must be in the range of the values in the dataset. If this value is median, then half of the values in the dataset are less than it and the other half are more than this guessed value. Therefore, we go through the dataset and count the numbers less than guessed value(let them be LM), if this number is equal to Median Index = (total count of numbers in dataset / 2) then that is median. Otherwise, if LM < P we need to narrow the range by increasing the minimum to guessed value and if LM > P then we narrow the range by decreasing the maximum to the value of guessed number and continue until we find the Median.

2.2 PIG SOLUTION

Firstly, I used Pig with this basic algorithm to iterate and find the median. To show how it works, after loading data to a relation called 'data' (same as above), I filtered the data which are less than guessed median value:

```
grunt> k1r = FILTER data BY (value < guessedValueHere );
grunt> k1g = GROUP k1r ALL;
grunt> k1c = FOREACH k1g GENERATE COUNT(k1r);
grunt> DUMP k1c;
```

which the result is LM. Names of the relations are just for my calculation, for example, digit one in k1r means it is the first iteration. It took me about 20 iterations to find a median value which is 7832134.13 and now that I used Java, I know this is not exactly the median.

After talking with Liang about making this algorithm better, he suggested to extend this method by breaking the data into more intervals and calculate each interval frequency in one iteration. The more the number of intervals, the less iterations to do for finding the median. Since, I found it a bit complex to do it with Pig, I started coding with Java. First, I want to explain a way of finding median which in theory it can find median in one or two iteration. I did not implement it completely, so I do not know if it is working or not. The solution benefits from sorted output of Reducer in mapReduce. If in the mapper, we map each line with keys equal to the string of their values as below:

String strOfValue = String.valueOf(observation);
output.collect(new Text(strOfValue), something usefull?);
The Reducer will sort the dataset. Then it will not be that hard to find the median from the sorted dataset which is the output of Reducer. I do not know about memory usage but it is just a plain idea.

2.3 JAVA SOLUTION

In Java solution, I am using the mapper to help me with narrowing down the range of the dataset. For example, in the beginning the Average value can be used to find an approximate range than the median will fall into that. Once we have a range, which the numbers which are less than start of the range value is less than Median Index then we can continue with narrowing down the range from up and down. The key of the output in mapper for values less than start of the range is 'lessThanRange' and the value of each output in mapper is digit 1, since we are counting the number of values with the key 'lessThanRange' in reducer. Digit 1 is acting as a counter in the reducer. For values which are in the range we defined, the output key is changing by each iteration helping us to filter values based on their common digits. For example, we can group values by their first two or three digits and then find the count for each group and compare it to the Median Index and narrow the range more and more by continuing iterations. As the number of values in the dataset is even, we have two Medians and the Median itself is the average of two medians. The Java code should be checked to illustrate my explanations. The code is available in the file and associated with questions 2.

3 SOLUTION AND ANSWERS FOR QUESTION THREE

3.1 ANSWERS(WITH JAVA)

Quartile1:	4357676.98235004
Quartile2:	4357677.01040075
Quartile =	Average(Quartile1, Quartile2) = 4357676.996375395
Maximum:	7.370896573706365E7
Minimum:	4357677.01040075
Average:	11196193.2677048246065776
Variance:	35536715121671.8852519989013672

For finding quartile I just used Java with the same code as Median with modifications. So there is no other code for quartile as it is the same as Median. We know that the quartile index = 536870912/4 = 134217728 For answering other questions, I restricted the dataset to values more than Quartile value by adding the below if statement in the same code as question 1(in mapper):

if(observation > 4357676.996375395)
output.collect(new Text("lines"), new DoubleWritable(observation));

so we are just outputing those values that we want and do not consider values less than quartile.
The output of the program is as follow:

maxValue : 7.370896573706365E7
minValue : 4357677.01040075
numberCount : 4.02653184E8
sumOfAll : 4.508182867920712E15
varianceSum : 1.4308971492642132E22