Report averagewordlength

* 1. Project introduction.
     1. Project is to write a MapReduce job that reads any text input and computes the average length of all words that start with each character. This will include numbers and case sensitive letters.
  2. Description on driver: job configuration, input format, output format, etc.
     1. Job configuration is to take the words of a text input and average the length of all words that start with each character by using the mapper and reducers LetterMapper.java and AverageReducer.java.
     2. Input format is lines of text like Shakespeare’s’ plays format.
     3. Output format is the character type then the average numbers of times it shows up in decimals.
  3. Description on mapper: input key, value types, output key, value types, and method manipulation on the input key value pairs.
     1. Input key is ignored. The value type is text.
     2. The output key is a number. The value type is int representing the length of the word.
     3. The method manipulation is to make the first letter as the key and the length of the word its value.
  4. Description on reducer: input key, value types, output key, value types, and method manipulation on the input key value pairs.
     1. Input key is from mapper and is Text.
     2. Input value is taken from mapper and is an Iterable of mapper values being Iterable<IntWritable>(collection of integers).
     3. Output key is first character of each word, Text object.
     4. Output value is average of each character as an int.
     5. Method of manipulation is to use the built-in shuffle and sort phase of MapReduce and receive the sorted keys and group values of one key together as keyvalue pair. They will be written in decimal format.
  5. Data flow description starting at the input files.
     1. Input files are complied into a .jar file
     2. When job runs on cluster, input file location comes from the first argument and then is given to the mapper by each line.
     3. The mapper maps the first character and assigns value of its length.
     4. The reducer sorts and combines same keys and then reduces by finding the amount of times the key occurs and dividing the length by those occurrences.
  6. Compile procedure description.
     1. In the Eclipse package explorer, export the project, select JAR file.
     2. Specify desired location of JAR file.
  7. Program test procedure
     1. Run the job my using the command $ hadoop jar avgwordlength.jar stubs.AvgWordLength \ shakespeare wordlengths
     2. List results by running $ hadoop fs -ls wordlengths
     3. View results by running $ hadoop fs -cat wordlengths/\*
     4. Results

1 1.02

2 1.0588235294117647

3 1.0

4 1.5

5 1.5

6 1.5

7 1.0

8 1.5

9 1.0

A 3.891394576646375

B 5.139302507836991

C 6.629694233531706

D 5.201834862385321

E 5.514263685427911

F 5.255528255528255

G 5.809792180345192

H 4.42107243650047

I 1.4526860926284046

J 4.984008528784648

K 4.657106838953672

L 5.115881561238224

M 5.44646530258742

N 3.9848387785607517

O 2.8794768365725463

P 6.505740766357726

Q 5.5216426193118755

R 5.929275069461985

S 5.293126010314833

T 3.959143714919723

U 5.325

V 5.194537815126051

W 4.464014043300176

X 3.1650485436893203

Y 3.4432244242099626

Z 6.1

a 3.0776554817818575

b 4.245396808453862

c 6.041441229514624

d 4.146387533448764

e 5.182465923172243

f 4.778552071234998

g 4.938916799411837

h 3.8777881295555434

i 2.7292957500654507

j 5.329446064139941

k 4.607202914798206

l 4.272777716124736

m 3.7182168186423508

n 3.7032013944985334

o 2.7875536480686693

p 6.10748861047836

q 6.025462962962963

r 5.829150579150579

s 4.327014649237208

t 3.733261651336357

u 4.4905590522028875

v 5.726228030644434

w 4.3475752474027844

y 3.5292446231858716

z 4.672727272727273

Report log\_file\_analysis

1. Project introduction.
   1. Analyze the log file from a web server and count the number of hits made from each unique IP address.
2. Description on driver: job configuration, input format, output format, etc.
   1. It takes the logs, read and store the IP address, count the number of times those address pop up, and assign key value pair on total times it appears by using LogFileMapper and SumReducer.
3. Description on mapper: input key, value types, output key, value types, and method manipulation on the input key value pairs.
   1. Input key is type LongWritable.
   2. Input Value type is text.
   3. Output key is IP address.
   4. Output value is IntWritable.
   5. The input is split into spaces, we use first field, ip address, as key. Assign the value of IP address to 1.
4. Description on reducer: input key, value types, output key, value types, and method manipulation on the input key value pairs.
   1. Input key is type Text.
   2. Input value is type IntWritable and is integer
   3. Output key is IP address as Text object.
   4. Output value is total number of hits made with that IP address as IntWritable.
   5. Each of the IP address occurrences are totaled and value is written as keyvalue pair, key as IP address.
5. Data flow description starting at the input files.
   1. Input files are compiled into JAR file
   2. Job runs on cluster, input file location is taken from the first argument and given to mapper by each line.
   3. Mapper maps the IP address to 1.
   4. Reducer will sort and combine the same keys and find total occurrences for each key.
6. Compile procedure description.
   1. Within Eclipse package explorer, export project and select JAR file.
   2. Choose location for the JAR file.
7. Program test procedure and the results.
   1. Run the map reduce job by this command: $ hadoop jar processlogs.jar stubs.ProcessLogs \weblog processlogs
   2. View results by doing : $ hadoop fs -cat processlogs/\*
   3. Results(last couple. Too many):

10.99.79.45 14

10.99.79.77 13

10.99.8.111 1

10.99.8.131 50

10.99.8.152 2

10.99.8.171 1

10.99.8.205 1

10.99.8.226 1

10.99.8.58 13

10.99.8.63 21

10.99.8.73 47

10.99.8.80 1

10.99.80.140 1

10.99.80.156 1

10.99.80.207 1

10.99.80.21 12

10.99.80.85 2

10.99.81.154 1

10.99.81.35 1

10.99.81.44 1

10.99.82.103 12

10.99.82.139 18

10.99.82.148 1

10.99.82.193 26

10.99.82.199 1

10.99.82.219 2

10.99.82.224 2

10.99.82.237 1

10.99.82.238 1

10.99.82.39 1

10.99.82.49 1

10.99.82.8 3

10.99.82.93 4

10.99.83.102 43

10.99.83.119 11

10.99.83.127 1

10.99.83.192 1

10.99.83.249 17

10.99.83.58 3

10.99.84.169 26

10.99.84.185 1

10.99.84.47 1

10.99.84.72 1

10.99.85.130 1

10.99.85.238 4

10.99.85.48 2

10.99.85.55 1

10.99.85.83 1

10.99.86.132 1

10.99.86.171 14

10.99.86.181 1

10.99.86.36 1

10.99.86.51 1

10.99.86.86 3

10.99.87.100 1

10.99.87.149 1

10.99.87.182 2

10.99.87.192 1

10.99.87.199 1

10.99.87.202 1

10.99.87.68 1

10.99.88.171 1

10.99.88.197 365

10.99.88.241 12

10.99.88.35 1

10.99.88.57 8

10.99.88.9 1

10.99.89.137 1

10.99.89.167 1

10.99.89.221 7

10.99.89.232 1

10.99.89.44 1

10.99.89.63 1

10.99.89.66 1

10.99.9.111 1

10.99.9.148 1

10.99.9.32 1

10.99.9.74 2

10.99.9.75 30

10.99.9.79 2

10.99.90.120 1

10.99.90.16 1

10.99.90.228 1

10.99.90.30 1

10.99.91.111 1

10.99.91.143 1

10.99.91.159 26

10.99.91.221 1

10.99.91.23 2

10.99.91.234 1

10.99.91.52 18

10.99.91.84 3

10.99.91.88 1

10.99.92.13 1

10.99.92.145 1

10.99.92.152 2

10.99.92.178 1

10.99.93.12 1

10.99.93.157 1

10.99.93.19 1

10.99.93.197 2

10.99.93.224 6

10.99.93.229 54

10.99.93.247 8

10.99.93.25 1

10.99.93.55 1

10.99.93.99 1

10.99.94.182 1

10.99.94.197 1

10.99.94.236 1

10.99.94.53 1

10.99.95.116 1

10.99.95.132 1

10.99.95.213 39

10.99.95.22 1

10.99.96.129 1

10.99.96.193 1

10.99.96.220 1

10.99.96.84 1

10.99.97.112 1

10.99.97.130 1

10.99.97.14 6

10.99.97.193 5

10.99.97.211 19

10.99.97.217 1

10.99.97.32 1

10.99.97.34 65

10.99.97.41 1

10.99.97.54 1

10.99.97.8 1

10.99.98.150 2

10.99.98.229 2

10.99.98.77 1

10.99.99.127 1

10.99.99.186 6

10.99.99.247 1

10.99.99.58 21