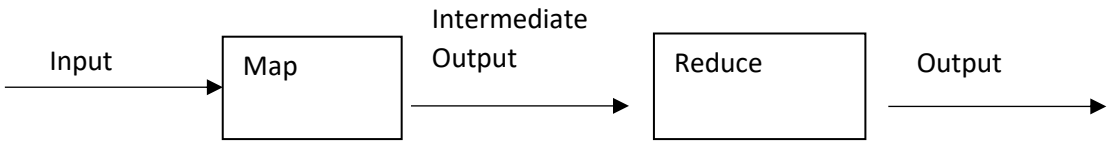
ENSF 612 Assignment 1
Sieu Eric Diep



1) Producing a random sub-sample of a big dataset using Hadoop MapReduce:

Input: key = document name, value = document text
Intermediate output: key = document name, value = 10% of the text
Output: key = document name, value = 10% of the text

Map(document name, document text):

count the total number of lines of the document text (N)

for 0.1*N times: //looping 0.1* N times
while(true):
generate a random number R between 1 and N
// assume the line number starts at 1, end at N
if this random number has not been generated previously:
select a line correspond to this random number and store it into a list L
store this random number in a list
break

emit(document name, list L = 10% of the line)

Reduce(document name, list L) (Note: this is an identity function that doesn't do anything but emit the output of the map)

emit(document name, list L = 10% of the line)

Hadoop Command:

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myPythonScript.py \
    -reducer /bin/wc \
    -file myPythonScript.py
```

Explanation:

For a document with N number of lines, the mapper would loop 0.1* N times. Each time, it would generate a unique random number corresponding to the line number of each line in the document and pick out the content of this line. The intermediate output from the mapper would be approximately 10% of the line number in the document. The reducer is simply an identity function that emits the same result.

The Hadoop command above specifies the input location for the mapper, output location for the reducer. The –file options provides a script to the mapper, reducer, or combiner to run

2) Building n-grams using Hadoop MapReduce. (Marks: 5)

Input: key = document name, value = document text
Intermediate output: key = 2 word di-gram (w1,w2), value = di-gram count, which should be 1
Output: key = 2 word di-gram (w1,w2), value = the **Total/sum** count # of the di-gram

Map(document name DN, document text DT):

Cleaning the document text:

- Removing punctuation and special characters except dash as in di-gram, assuming words connected with dash is one word
- Run a grammar check and spelling check so that the words are correctly spell. This to prevent a situation such as spell and spel are considered as two different words
- Set all the words to lowercase

for each word w1 in the document text DT
combine w1 with its next neighbor w2 to form a tuple (w1,w2) as a key
emit((w1,w2), 1)

Note: the result of the map would be automatically group by the framework which is not shown here, before feeding to reduce

Reduce(di-gram (w1,w2), di-gram count =1)

TotalCount = 0
For each count value v in di-gram count:
TotalCount += v
emit(di-gram (w1,w2), TotalCount)

Hadoop Command:

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myPythonScript.py \
    -reducer /bin/wc \
    -file myPythonScript.py
    -combiner streamingCommand or JavaClassName
```

Explanation:

The mapper would loop through each word w1 in the document, and emit a key which is a tuple that include the word w1 and its next neighbor w2. The value of this key is 1. Then the reducer will sum up all of these values and emit a key (w1,w2) and its total number of occurrence.

The Hadoop command will be the same as in question 1 with an extra option –combiner which provide a script to combine the output of the mappers in the event many mappers are used.

3) Building an inverted index of a text corpus using Hadoop MapReduce:

Input: key = directory name, value = documents in the directory
Intermediate output: key = unique word W in the document, value = name of the document (file name)
Output: key = unique word W, value = a list of the document names that contains W

Map (directory name, the documents in the directory):

Cleaning the content of the documents:

- Removing punctuation and special characters except dash as in di-gram, assuming words connected with dash is one word
- Run a grammar check and spelling check so that the words are correctly spell. This to prevent a situation such as spell and spel are considered as two different words
- Set all the words to lowercase

for each word W in each document:
    if W is unique: // that is comparing W with the rest of the word in the document
        emit(W, document name)

Note: the result of the map would be automatically group by the framework which is not shown here, before feeding to reduce

Reduce(unique word W, document name)
    documentList = []; //an empty list to contain all the document name
    For each unique word W:
        Append the document name into the documentList
    emit(unique word W, documentList)

Note**:** many map would be used. Each map would take a number of documents in the directory

Hadoop Command:

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myPythonScript.py \
    -reducer /bin/wc \
    -file myPythonScript.py
    -outputformat JavaClassName
```

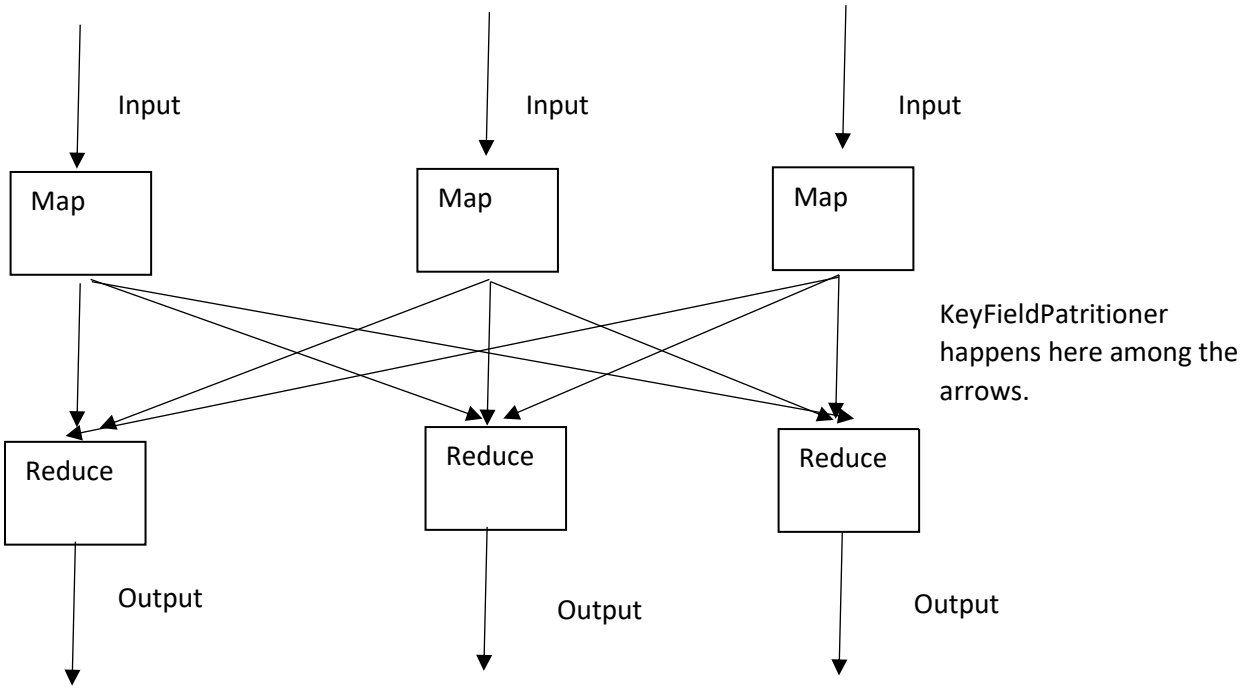
Explanation:

The mapper would loop through each word in the document and identify if this word is unique by comparing it to all other word in the document. If it is unique, then it emits this word as a key with the document name as a value. Then, the reducer will append all of the document name into a list, and emit the same key with this value.

The Hadoop command is the same as question 1, with an extra –outputformat option that would format the key, value pair output

4) Sorting using Hadoop MapReduce:

Input: key = document name, value = chunks of document text
Intermediate output: key = each word w in the document text, value = 1
Output: key = each word w in the document text, value = total word count of w. These key, value pair is emitted in an ascending order



Map(document name, chunk of document text)
Cleaning the document text:

- Removing punctuation and special characters except dash as in di-gram, assuming words connected with dash is one word
- Run a grammar check and spelling check so that the words are correctly spell. This to prevent a situation such as spell and spel are considered as two different words
- Set all the words to lowercase

For each word w in the document text:
emit(w,1)

Partitioning of the mapper output into the reducer:

Then, a proper hash map would be used to partition these intermediate key (word W) into the proper reducer. All the key (word W) with a similar rank should be partitioned into the same reducer because there hash code would be in a similar range.

An example of the hash function would be to modulus the ASCII value of the first letter in the key word, by 97, which is the ASCII value of letter 'a'. As a result, the hash code of all the words in the document will be ranging from 0 to 25 corresponding to the alphabet a-z.

For example, the hash code of the word "apple" will be 0 since 'a' % 97 = 97 % 97= 0
Similarly, the hash code of the word "banana" will be 1 since 'b' % 97 = 98% 97 = 1, etc

The number of reducers can be set to 26. One reducer to contain all the words start with each letter in the alphabet.

Reduce(word W, 1):
sum = 0;
For each count v of word W:

sum += 1

then sort all the (word W, sum) pairs in ascending order according to the key word W.
Emit(word W, sum) in this ascending order

Hadoop Partitioner Class: KeyFieldBasedPartitioner,

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -D mapred.reduce.tasks=26 \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myScript \
    -reducer myScript \
    -partitioner myHashfunction \
    -file myscript
```

Explanation:

This question is similar to a word count problem where the mapper would simply emit the word as a key with a value of 1. Many mapper would be used to process different chunks of the text. Then the output of each mapper will be partitioning into multiple reducers using a proper hash function. This hash function would ensure

Each reducer will then summing up the total count of each word to form a unique key, value pair. Then, the reducer will sort these unique key (word W) in an ascending order and emit them according to this order. This way all the words will be displayed in an ascending order.

The Hadoop command is similar to the previous questions with the following extra options:

`-D mapred.reduce.tasks=26` to set the number of reducer to 26

`-partitioner myHashfunction` to set up the partitioner with the proper hashfunction