Recitation 2
CS435: Introduction to Big Data

GTAs: Bibek R. Shrestha and Aaron Pereira
Email: cs435@cs.colostate.edu

September 7, 2018
Today...

- We will go through code snippets for each profile.
Dataset

- ~1.5 million Wikipedia articles (summarized)
- ~1 GB size
- Format:

  Title_of_Article-1<===>DocumentID<===>Text_of_Article-1
  NEWLINE
  NEWLINE

  Title_of_Article-2<===>DocumentID<===>Text_of_Article-2
  NEWLINE
  NEWLINE
Data Preprocessing

- Consider only alphabetic and numeric text.
- Convert upper cases to lower cases.
- Example (Original → processed → toLowerCase)
  
  U.S.A. → USA → usa
  west. → west → west
  (USA) → USA → usa
  D.C., → DC → dc
  9.8 → 98 → 98
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    if (!value.toString().isEmpty()) {
        StringTokenizer token = new StringTokenizer(value.toString().split("<====>")[2]);
        while (token.hasMoreTokens()) {
            String out = token.nextToken().replaceAll("[^A-Za-z0-9]","").toLowerCase();
            ...
        }
    }
    ...
}
Profile 1

- First 500 unigrams in the dataset
- Must be sorted in alphabetical order (ascending)
- Eliminate duplicates.
- Use of combiner to eliminate local duplicates can be important (one of the approaches that can be taken).
Profile 1 Solution

- This is very similar to WordCount program except you do not have to count the word frequency in the reducer.
- Emit the key received from the mapper function.
- Mapper function emits (word, 1) or (word, null)
- In Reducer, `context.write(key, null);`
- Why do we need Reducer then? - >For sorting the mapper output.
- If number of reducer more than 1, as defined using `job.setNumReduceTasks(5);` in driver program, we will get sorted output per reducer, not the globally sorted output as required!
- Use Partitioner to get globally sorted output (defined in driver): `job.setPartitionerClass(profile1Partitioner.class)`
Partitioner code snippet

```java
@Override
public int getPartition(Text key, IntWritable value, int numReduceTasks) {
    if (numReduceTasks == 5) {
        Character partitionKey = key.toString().toLowerCase().charAt(0);
        if (partitionKey >= 'a' && partitionKey <= 'd')
            return 0;
        else if (partitionKey >= 'e' && partitionKey <= 'k')
            return 1;
    }
    ...
```
Profile 2

- A list of top 500 unigrams and their frequencies within each article
- Similar to word-count program
- Each article has unique integral Document ID.
- Output should be grouped by Document ID.
- Output files will correspond to the number of reducer used.
- Requires two jobs to be executed in a sequence
- Also, you may use the concept of CompositeKey in MapReduce.
Profile 2 Solution

- Output of Mapper will be in the form:
  \( \{\text{DocumentID, unigram}, 1\} \)

- Output of Reducer will be in the form:
  \( \{\text{DocumentID, unigram}, \text{frequency}\} \)

- Use TopK concept to output based on frequency.
Profile 3

- A list of top 500 unigrams and their frequencies in the corpus
- List should be sorted from most frequent unigrams to least frequent ones.
- Solution to generate profile 3 is a combination of profile 1 and 2.
- Here, we list unigram with its total occurrence in the complete dataset (of course, without considering per article construct as in profile 2).
Profile 3 Solution

- Similar to Profile1
- Except, count the frequency in reducer as you did in WordCount program in PA0
- And, sorting is based on the value of (key, value) pair output from reducer
- So you need second mapreduce job.
- And, to sort by value, reverse key-value pair being sent by first mapreduce job.
In next recitation...

- We will go wrap-up PA1.
- We will start discussion on PA2.