Recitation 3
CS435: Introduction to Big Data

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

September 14, 2018
Today...

- Discussion on Programming Assignment 1
Profile 1

- First 500 unigrams in the dataset
- Must be sorted in alphabetical order (ascending)
- Eliminate duplicates.
A. Profile 1 Solution using TopN

- Follow the TopN Design Pattern
  
  - Mapper:
    - Initialize TreeMap<String, String>
    - Store and update first 500 unigrams in the TreeMap
    - Use cleanup() method to send Local 500 unigrams from each mapper. (Context write with nullWritable as key)
  
  - Reducer:
    - Initialize TreeMap<String, String>
    - Store and update TreeMap to get global first 500 unigrams
B. Profile 1 Solution using TopN (modified)

- Instead of passing local 500 unigrams from mapper, filter global 500 unigrams at reducer-side. (NOT very efficient)

- Mapper:
  - Just like wordcount program, Context write <unigram, 1> or <unigram, null>

- Reducer:
  - Initialize TreeMap<String, String>
  - Store and update TreeMap to get global first 500 unigrams

- Note: explicitly set number of reducers to 1 in driver as job.setNumReduceTasks(1)
Profile 2

- A list of top 500 unigrams with highest frequencies within each article
- Each article has unique integral Document ID.
- Output should be grouped by Document ID.
- Output files will correspond to the number of reducer used.
A. Profile 2 Solution using single job

- Instead of passing local 500 unigrams from mapper, filter global 500 unigrams at reducer-side. (NOT very efficient)

- Mapper:
  - <docID, unigram>

- Reducer:
  - Store and update HashMap to get frequency
  - Context write top 500 unigrams with highest frequencies
  - Output should contain docID, unigram and its frequency

- Note: For efficiency, you can set the number of reducers to appropriate number \( n \) in driver as `job.setNumReduceTasks(n)`
B. Profile 2 Solution using single mapper?

- Observe the solution (B) in previous slide.
- Do we need reducer? Why? Why not?
C.1. Profile 2 Solution using multiple jobs with TopN

**Job1**
- Mapper1:
  - Context write <(DocID, unigram), 1>
- Reducer1:
  - Count the frequency of (DocID, unigram) pair
  - Context write <DocID, (unigram, frequency)>
C.2. Profile 2 Solution using multiple jobs with TopN

**Job2**

- **Mapper2:**
  - Store and update top 500 unigrams (based on frequency) in the TreeMap.
  - Use cleanup() method to send Local top 500 unigrams from each mapper (Context write with docID as key).

- **Reducer:**
  - Store and update TreeMap to get global top 500 unigrams.
  - Output should contain docID, unigram and its frequency.
D.1. Profile 2 Solution using compositeKey with sortComparator

Job1
- Mapper1:
  - Context write <(DocID, unigram), 1>
- Reducer1:
  - Count the frequency of (DocID, unigram) pair
  - Context write <DocID, (unigram, frequency)>
D.2. Profile 2 Solution using compositeKey with SortComparator

**Job2**
- Mapper2:
  - Context write <(docID, unigram, frequency), null>
- SortComparator:
  - Extend WriteComparator such that: one can independently sort each of the three items in the key. (Code snippet [Link])
- Reducer:
  - Store and update TreeMap to get global top 500 unigrams.
  - Output should contain docID, unigram and its 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.
A. Profile 3 Solution using TopN pattern

- Like WordCount program
- Mapper:
  - Context write <unigram, 1>
- Reducer:
  - Store and update HashMap to get frequency of each unigram
  - Context write top 500 unigrams with highest frequencies
- Note: explicitly set number of reducers to 1 in driver as `job.setNumReduceTasks(1)`
B.1. Profile 3 Solution using multiple jobs

**Job1**
- Mapper1:
  - Context write <unigram, 1>
- Reducer1:
  - Context write <unigrams, frequency>
- Note: For efficiency, you can set the number of reducers to appropriate number $n$ in driver as `job.setNumReduceTasks(n)`
B.2. Profile 3 Solution using multiple jobs

**Job2**
- Follow the TopN Design Pattern
- Mapper2:
  - Initialize TreeMap\textless \text{String}, \text{String}\textgreater
  - Store and update the 500 unigrams in the TreeMap
  - Use cleanup() method to send Local top 500 unigrams from each mapper. (Context write with nullWritable as key)
- Reducer2:
  - Initialize TreeMap\textless \text{String}, \text{String}\textgreater
  - Store and update TreeMap to get global top 500 unigrams
B.3. Profile 3 Solution using multiple jobs without using TopN

**Job2 (alternative)**
- Mapper2:
  - Context write <frequency, unigram>
- sortComparator:
  - Set descending order sort on key (frequency)
- Reducer2:
  - Context write first 500 input
Demo

- Running Hadoop job in IDE
- Using Logger class to get logs in cluster mode.
In next recitation...

- Introduction on PA2.