Recitation 6
CS435: Introduction to Big Data

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

October 5, 2018
Recall last recitation
How to approach PA2?

- There are few documents (articles) without article body. (For example: HelloWorld<====>12345<====>) Put a checker before you proceed. Also, ignore such articles while calculating $N$ (total number of articles in the corpus).

- Divide the PA2 into two parts (A and B). From Part A, get the TF-IDF value for each unigram in each document. From Part B, you should get article summary for any given article (provided during demo). Make sure Part A and Part B are separate jars that could be run independently.

- To calculate $N$ and pass its value to a different job, use Hadoop's `Counters` class.

  - References:

- Use multiple reducers wherever possible and implement CustomPartitioner.
Part A - Job1

- Find the frequency of each unigram in each article.
- Similar to Profile2 in PA1.
- Output will be in the form: (docID  unigram  frequency)
Part A - Job2

- Calculate TF of each unigram i in each article j using:
  \[ \text{TF}_{ij} = 0.5 + 0.5 \left( \frac{f_{ij}}{\text{max}_{k} f_{kj}} \right) \]
  where, \( \text{max}_{k} f_{kj} \) is the highest frequency of a unigram in the article j.

- Mapper output will be in the form: \(<K,V> \rightarrow <\text{docID}, (\text{unigram frequency})>\>

- Reducer:
  - First iterate through all the values and find \( \text{max}_{k} f_{kj} \)
  - Next, calculate TF for each unigram
  - Output will be in the form: \(<\text{docID}, (\text{unigram TFvalue})>\>\)
Part A - Job3

- Calculate $n_i$, total number of documents in the whole corpus, where unigram $i$ occurred at least once.
- Mapper output will be in the form: $<K,V> \rightarrow <\text{unigram}, (\text{docID TFvalue})>$
- Reducer:
  - Iterate through all the input values and find $n_i$.
  - Output will be in the form: $<\text{docID}, (\text{unigram TFvalue } n_i)>$
Part A - Job4

- Calculate **TFIDF** of each unigram *i* in each article *j* using:
  
  \[ \text{IDF}_i = \log_{10}(N/ni) \]
  
  \[ \text{TF-IDF}_{ij} = \text{TF}_{ij} \times \text{IDF}_{ij} \]

- Mapper:
  - Calculate IDF for each unigram.
  - Calculate TF-IDF for each unigram.
  - Output will be in the form: \(<\text{docID}, (\text{unigram} \quad \text{TF-IDF value})>\)
  - Note: To calculate IDF, you need to pass \(N\) that you calculate in previous job.

- Reducer: Not Required. (Identity Reducer!)
Part B- Job1

- Generate summary for any given article.
- Multiple inputs required: Output of previous job (PartA-Job4) and given article. You can implement this in either of the two ways:
  - Use `MultipleInputs.addInputPath()` and join them on keys. (Reduce Side Join; Lecture Note: Week4-A)
  - Persist output from previous job in Hadoop’s DistributedCache.
Part B- Job1 continue

**Reduce-side join pattern:**
- Mapper: Two mappers each passing corresponding input file with docID as key.
- Reducer:
  - Load TFIDF data into lookup table (HashMap).
  - Split the given article on a period and a white space to get individual sentences.
  - Assign index to each sentence based on its order of occurrence.
  - Use lookup table to calculate sentence TFIDF. (Use top 5 unigrams from each sentence)
  - Output top 3 sentences preserving its order of occurrence in original article.

**Hadoop's DistributedCache:** [Link]
- Mapper:
  - Single mapper.
  - Load TFIDF data from distributed cache into lookup table.
  - Rest same as reducer above.
- Reducer: No reducer required. (Identity Reducer!)
Key Components in PA2
Counters class

- Create a global struct:

  ```java
  public static enum DocumentsCount {
      NUMDOCS
  }
  ```

- Update the global count in your mapper:

  ```java
  context.getCounter(DriverClass.DocumentCount.NUMDOCS).increment(1);
  ```

- Retrieve the count in driver class before using it in next job:

  ```java
  Counter documentCount = firstJob.getCounters().findCounter(DocCount.NUMDOCS);
  ```

- Declare use of counters in the driver:

  ```java
  IDFJob.getConfiguration().setLong(documentCount.getDisplayName(), documentCount.getValue());
  ```
Counters Class continue

- Access the count value in your mapper/reducer via `setup` method:

  ```java
  N = context.getConfiguration().getLong(DriverClass.DocumentCount.NUMDOCS.name(), 0);
  ```
Combiner class

- Make sure if you are correctly using Combiner.
  - The input of combiner class should match with the output of Mapper class and the output of combiner class should match with the input of Reducer class.
  - Input types (key/value pair) and the output types must be the same: Same as output of mapper as well.
  - Properties of a Combiner Combiner’s functionality must be
    - Commutative
    - Associative
  - Hadoop does not guarantee on how many times a combiner will be called for each output key.
  - Do NOT use Reducer class as your combiner unless sure.
Reduce-side Join Pattern

- [http://www.cs.colostate.edu/~cs435/slides/week4-A.pdf](http://www.cs.colostate.edu/~cs435/slides/week4-A.pdf)
In the next recitation

- Introduce PA3