**CS5543 Real-Time Big Data Analytics**

**MapReduce & Spark/Storm Programing**

ICE-5

9/27/2016

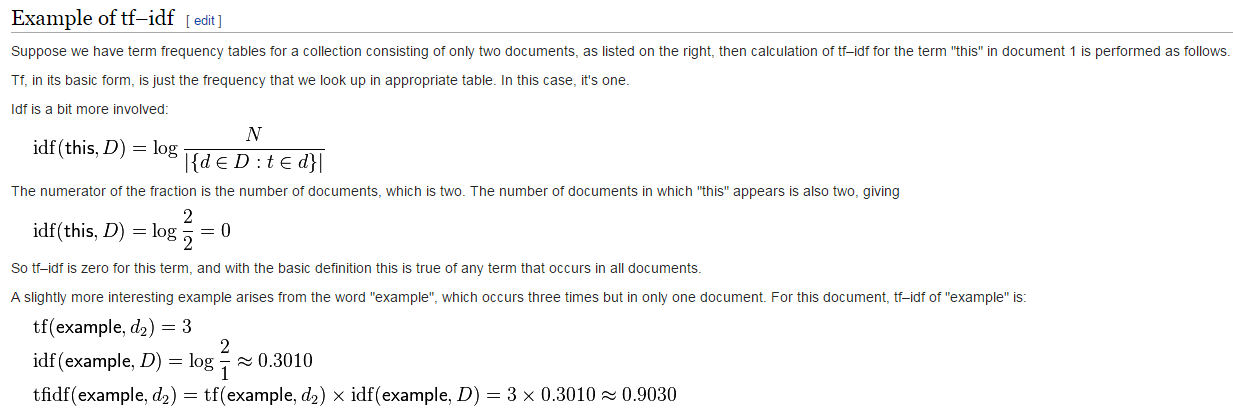
Name:

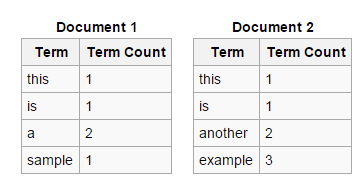
Class ID:

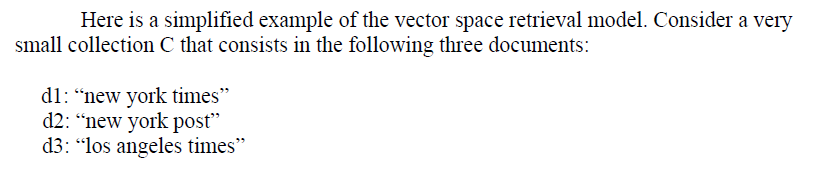
**MapReduce & Spark Programming: Compute TF-IDF**   
  
Extract the terms from a document**.** To keep the term simple, each word itself is a term in our example below. **Multiple rounds of Map/Reduce** to gradually compute are required as follows:

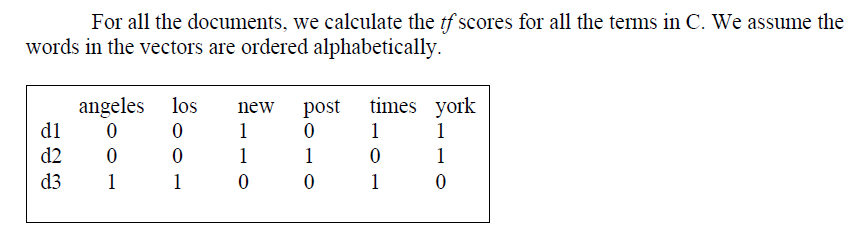
1. the word count of per word/doc combination
2. the total number of words per doc
3. the total number of docs per word.
4. And then finally compute the TF-IDF

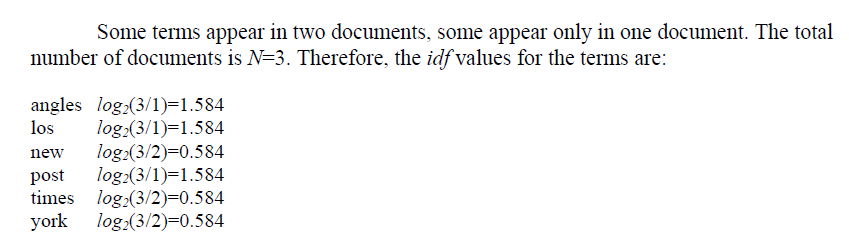
Given the problem description,

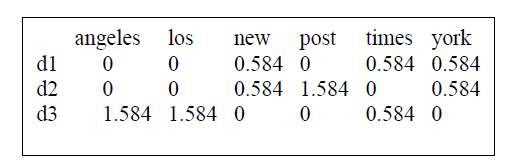
1. Draw the MapReduce Diagram.
2. Sketch the MapReduce Algorithm.
3. Sketch the Spark Scala implementation
4. Sketch the Storm Topology

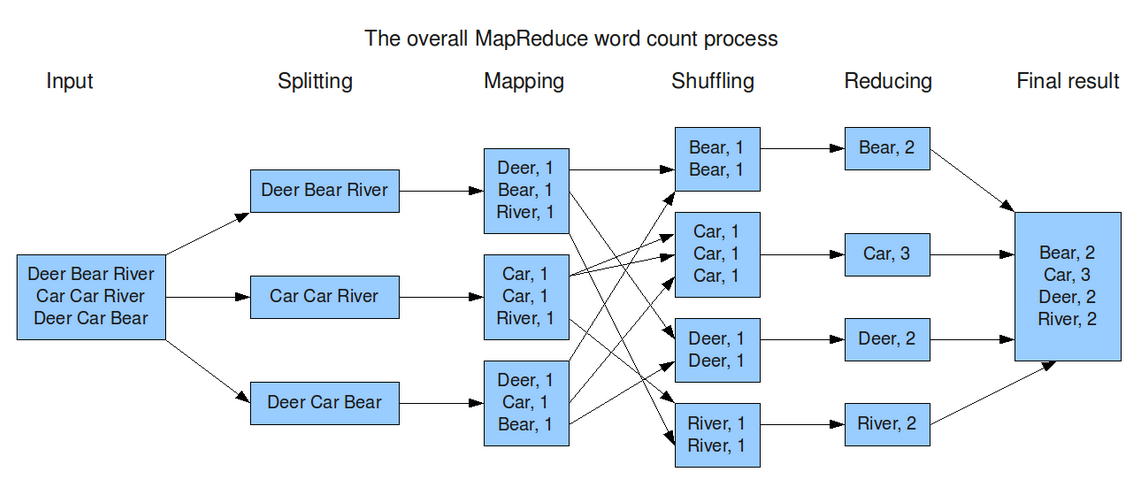


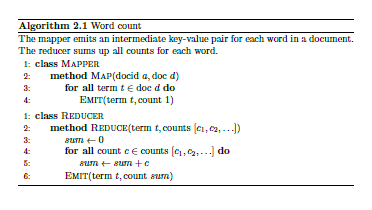
**Example**



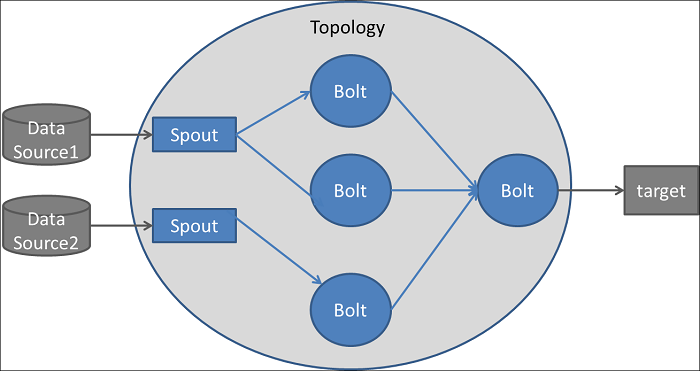


 Now we compute tf-idf by multiply the *tf* scores by the *idf* values of each term, obtaining the following matrix of documents-by-terms

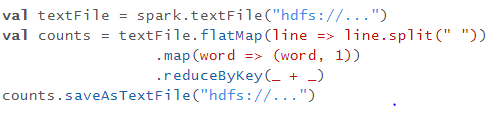




Storm Topology Example



**Spark Scala Code for WordCount**



|  |  |
| --- | --- |
| **flatMap(func)** | Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item). |
| **reduceByKey**(*func*, [*numTasks*]) | When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument. |