# Information Retrieval Example using Map-Reduce

- Information Retrieval is the process of finding information in response to a query from a source of information.
- "*But do you know that, although I have kept the diary [on a phonograph] for months past, it never once struck me how I was going to find any particular part of it in case I wanted to look it up?*"
  —Dr Seward, Beam Stoker's Dracula, 1897
- The source of information can be a text corpus consisting of structured text files such as books or web pages, or it can be meta-data stored in structured format such as XML (for example to search in corpus of sound files or image files).
- In this example, we will look at the calculation of *term-frequency-inverse-document-frequency* (if-ids), which is a basic problem in information retrieval.

# Term Frequency–Inverse Document Frequency (TF-IDF)

- The tf-idf weight (*term frequency–inverse document frequency*) is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.
- The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.
- Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.
- The concept of Inverse Document Frequency was introduced by British computer scientist Karen Spärck Jones.

# TF-IDF Definition (Part 1)

- Let $D$ be the collection of documents in the corpus. Let $T$ be the collection of terms (unique tokens) in the collection $D$.
- The term frequency (tf) for a given term $t_i$ within a particular document $d_j$ is defined as the number of occurrences of that term in the $d_j$th document. We denote this with $n_{i,j}$: the number of occurrences of the term $t_i$ in the document $d_j$.

$$tf_{i,j} = n_{i,j}$$

- The term frequency is often normalized to prevent a bias towards larger documents, as shown below:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where $n_{i,j}$ is, as above, the number of occurrences of the term $t_i$ in the document $d_j$. Note that the denominator is the total number of terms in the document $j$, which is $\sum_k n_{k,j}$, for normalization. Instead, we can use the maximum as well as other values.

# TF-IDF Definition (Part 2)

- The inverse document frequency (idf) is obtained by dividing the total number of documents by the number of documents containing the term $t_i$, and then taking the logarithm of that quotient:

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

with

  - $|D|$: total number of documents in the collection
  - $|\{d : t_i \in d\}|$: number of documents where the term $t_i$ appears. To avoid divide-by-zero, we can use $1 + |\{d : t_i \in d\}|$.

- For a given corpus $D$, then the tf-idf is then defined as:

$$(tf\text{-}idf)_{i,j} = tf_{i,j} \times idf_i$$

- A high weight in *tf-idf* is obtained by a high term frequency and a low inverse document frequency of the term in the collection.
- For common terms, the ratio in *idf* approaches 1, bringing the logarithm closer to 0.

# Sequential Algorithm?


Discuss, in a group, how to calculate the tf-idf sequentially?

# What do we need to compute?

- Given a corpus of text, we want to calculate tf-idf for every document and every term.
- Note that we will use the total number of terms for normalization (as it is simpler to compute) although the definition asks us to use number of terms per document to normalize. This will be adjusted later in the second solution.
- We need to calculate, over the corpus, the following:
  - number of terms,
  - number of unique terms,
  - number of documents,
  - number of occurrences of every term in every document, and
  - the number of documents containing each term.

# New Map-Reduce and Hadoop Techniques

- Use of Hadoop counters.
- Use reducers for data pass-through instead of only reducing the data. This is typically done to add a new dimension to the data.
- Extending FileInputFormat classes to create custom key/value pairs rather than the default ones.

# Example Code

See the full code in the code examples repo at:
CS535-resources/examples/hadoop/tf-idf

The main class TfIdfDriver.java runs three passes of MapReduce in a sequence. The three MapReduce computations are each defined in the following classes.

- TermFrequency.java
- InverseDocumentFrequency.java
- TfIdf.java
- Supporting I/O classes (for defining custom I/O formats):
  - KeyValueTextIntInputFormat.java, KeyValueTextIntRecordReader.java
  - KeyValueTextTextInputFormat.java, KeyValueTextTextRecordReader.java

# Step 1: Number of terms and unique terms

- Run the same algorithm as the word count. Output looks like:

```
the 123455
from 65002
about 33004
.
.
.
```

- Number of unique terms is simply the number of lines in the output. Number of terms is the total of the second column.
- Running an entire pass just to calculate these two values is overkill. These two values can also be calculated as a by-product of Step 2 by using global counters that Hadoop allows a job to increment/decrement.

# Hadoop Global Counters

- Declare an `enum` to represent the counters. The enum name is the group of the counter, and each field of the enum is the name of the counter that will be reported in this same group.

```
public class TfIdfDriver {
    static enum Counters { DOCUMENTS, TERMS }
...
}
```

- Increment the desired counters from the map and reduce methods through the Context object. For example (`context` is an instance of a `Context` object):

```
context.getCounter(TfIdfDriver.Counters.TERMS).increment(1);
context.getCounter(TfIdfDriver.Counters.DOCUMENTS).increment(1);
```

- Access the value of the counter from the Job object. For example:

```
int numTerms = job.getCounters().findCounter(
                                TftIdfDriver.Counters.TERMS).getValue();
```

- For other methods, see the class `org.apache.hadoop.mapreduce.Counter`

# Step 2: Term Frequency

- Change the output from previous map step to produce "docId_term" pairs.

```
map1:          map2:          map3:
1_the 1        2_the 1        37_london 1
1_the 1        2_the 1        .
1_the 1        2_from 1       .
1_to 1         .              .
.              .
.              .
.
```

- Reducer simply adds up the values with the same key and we have the term frequency.

```
1_the 3
1_to 1
...
2_the 2
2_from 1
...
37_london 1
```

- We also get the number of terms and number of documents in this step using global counters.
- See the code at tf-idf/TermFrequency.java

# Step 3: Inverse Document Frequency (Try 1)

- Use the output from the previous step as the input. Take every key from the input data, split it to (1, the) and write to map's output (the, 1) to denote that we have at least one document containing the word `the`:

```
the 1
to 1
...
the 1
from 1
...
london 1
```

- Reduce simply computes the length of the list and outputs:

```
the 2
to 1
from 1
london 1
...
```

- Now we have two outputs: one with TF values and another with IDF values but no way to combine them.... Let's try another way.

# Step 3: Inverse Document Frequency (Try 2)

- We will use a MapReduce technique known as data pass-through. The output from the mapper in the previous attempt has more information that we can pass along for later use. We write `docId_tf` instead of simply writing "1" as before.

```
the 1_3
to 1_3
...
the 2_2
from 2_1
...
london 37_1
```

- Reduce computes the length of the list and outputs key/value pairs shown below (where the format of the value is (docCount, docId, countInDoc)):

```
the 2_1_3
the 2_2_2
to 1_1_3
from 1_2_1
london 1_37_1
...
```

  where the key/value 2_1_3 reads as: there are 2 documents that contain the term `the`, and the document with id 1 contains 3 such terms.
- *Note that the reducer didn't reduce the data but added a new dimension to it.*
- See the code at tf-idf/InverseDocumentFrequency.java

# Step 4: Calculating the tf-idf

- Assume we have counted the number of documents (`totalDocCount`) and the number of terms (`totalTermCount`), we can write a mapper (with no need for reduce) to finish the process.
- Mapper input:

```
term = key
(docCount, docId, countInDoc) = value.split("_")
```

- Mapper output:

```
tf = countInDoc/totalTermCount
idf = log(totalDocCount/docCount)
result = tf * idf
output(key = docId + "_" + term, value = result)
```

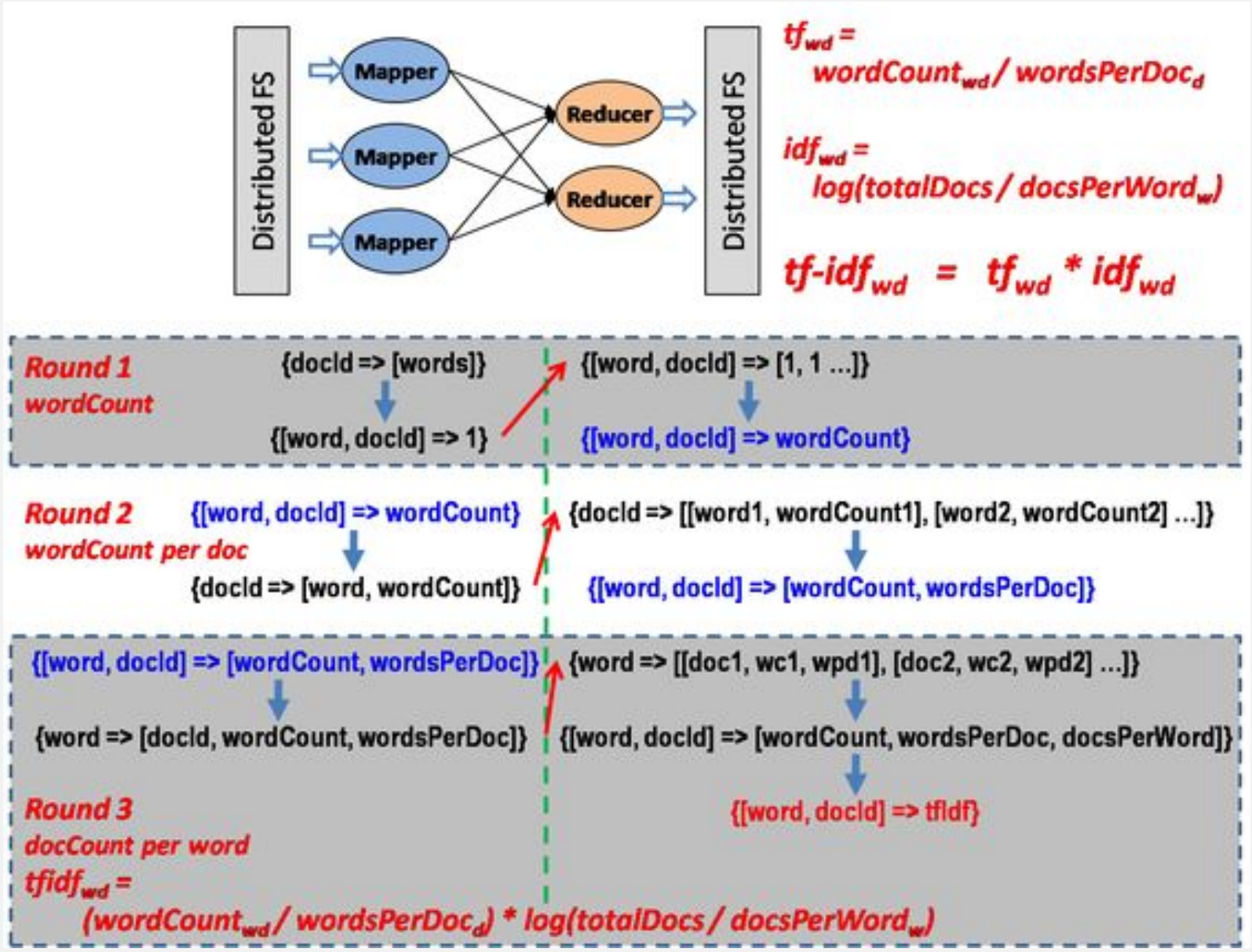
- See the code at tf-idf/TfIdf.java
- Note that we only need two MapReduce passes as we can calculate the TF-IDF at the end of Step 3. And we are avoiding a pass for Step 1 by using global counters (with the assumption that we are normalizing by dividing with the total number of terms)

# TF-IDF with better normalization

- In order to normalize the term frequency for term $i$ in the $j$th document by the number of terms (words) in that document, we need three MapReduce passes.
- This is because we cannot deduce the number of terms in the $j$th document that contains term $i$ from what we had calculated up to Step 3. Why?
  - After Step 3, we have *docCount* (how many documents contain the $i$th term), we have *countInDoc* (how many instances of the $i$th term are in the $j$ document) but we have no idea of how many total terms are in the $j$ document!
- See next slide for a three pass MapReduce algorithm.

# 3-Pass TF-IDF


In your group, discuss how the following algorithm works.



# Supporting I/O Classes

- KeyValueTextIntInputFormat.java, KeyValueTextIntRecordReader.java. These classes are used by the `InverseDocumentFrequency` mapper for input since we want the first field in the input to be returned as a `Text` key (instead of the default, which is the byte offset of current line in the input file) and the second field to be returned as an integer value.
- KeyValueTextTextInputFormat.java, KeyValueTextTextRecordReader.java. These classes are used by the `TfIdf` mapper for input since we want the first field in the input line to be returned as a `Text` key and the second filed to be returned as a `Text` object.

# References


- *Tf-idf*. http://en.wikipedia.org/wiki/Tf-idf.
- *MapReduce for real problems: Calculating TF-IDF using Hadoop*. Explains how to use MapReduce to calculate tf-idf but does not provide any code. http://romankirillov.info/hadoop.pdf. The website is no longer in existence but forms the basis for these slides.