CS 378 – Big Data Programming

Lecture 6

Summarization Patterns

Review

- Assignment 2 - WordStatistics

- We'll look at implementation details of:
  - Mapper
  - Combiner
  - Reducer
  - Supporting classes
- Other issues
  - Number precision – how can we control this on output?

File Formats

- In assignments 1 and 2, we have used
  - **`TextInputFormat`**
  - **`TextOutputFormat`**
- Key value pairs:
  - Input: **`LongWritable/Text`**
  - Output (Assign 1): **`Text/LongWritable`**
  - Output (Assign 2): **`Text/WordStatisticsWritable`**
- The input file is just lines of text
  - How does the **`LongWritable`** get generated?

File Formats

- Input formats provide an instance that extends Hadoop class **`RecordReader`**
- **`RecordReader`** methods
  - **`initialize(InputSplit, TaskAttemptContext)`**
  - **`nextKeyValue()`**
  - **`getCurrentKey()`**
  - **`getCurrentValue()`**
  - **`getProgress()`**
  - **`close()`**

File Formats

- What does **`TextInputFormat`** do?
  - Via its **`RecordReader`** implementer

- Identifies the next line of input
  - Text through the next newline
- Creates the **`Text`** object with this content
- Calculates the position of this line in the input split
- Creates the **`LongWritable`** with this number
- Reports progress via `getProgress()`

File Formats

- Key value pairs:
  - Output: **`Text/WordStatisticsWritable`**
- The output file is just lines of text
  - How does this text get generated?
- Similar to input formats, output is controlled by instances that extend **`RecordWriter`**
- **`RecordWriter`** methods
  - **`write(key, value)`**
  - **`close()`**

File Formats

- What does **`TextOutputFormat`** do?
  - Via its **`RecordWriter`** implementer

- Calls **`toString()`** on the key, writes this string
- Writes a tab character
- Calls **`toString()`** on the value, writes this string

- How do we control the format of our results for WordStatistics?

Summarization

- Another summarization of interest
  - Inverted index

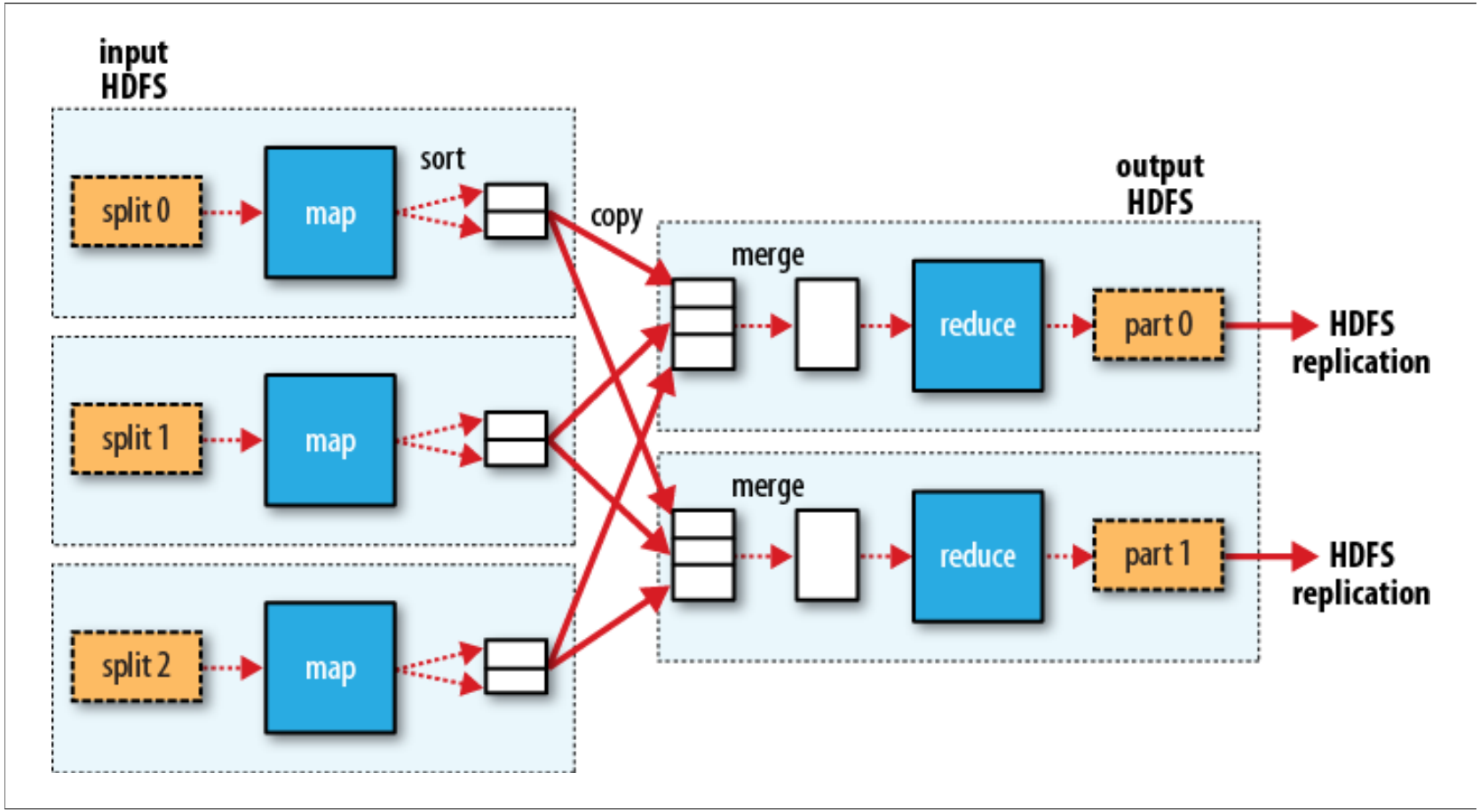
- Suppose we are interested in indexing document(s) by individual words in the document(s)
  - For a given word, which documents contain it
  - Indices are built for search engines to quickly identify which documents are relevant
  - Interesting for anyone investigating the document(s)

Inverted Index

- For an inverted index that represents which documents an individual word appears in:
- What is the final output?
  - Key: word
  - Value: list of documents the word appears in
- Given our data set of document(s)
  - What should the mapper do?
  - What should the reducer do?
  - Can we use a combiner?

MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide



Inverted Index – Assignment 3

- Data set example
  - Genesis:46:14 And the sons of Zebulun; Sered, and Elon, and Jahleel.

- Output for this "document" (one verse)
  - How many "outputs" for this verse?
  - What should the keys be?
  - What should the value(s) be?

Inverted Index – Assignment 3

- Input record parsing
  - Document-ID: `book:chapter:verse`
  - Text string that is the verse
- Verse text parsing:
  - Remove punctuation as in assignment 2
  - Lowercase words
- Output:
  - Word, list of `book:chapter:verse` (no duplicates)
  - Sort the `book:chapter:verse` references

Inverted Index – Assignment 3

- Start with `InvertedIndex.java` (onCanvas)
  - extends `Configured`
  - implements `Tool`

- Split out code from `main()` into `run()` method.

- For more info, see Hadoop, The Definitive Guide
  - pp. 148 – 152.