CS 378 – Big Data Programming

Lecture 5

Summarization Patterns

Review

- Assignment 2 – Questions?

- If you'd like to use guava (Google collections classes)
  - pom.xml available for assignment 2
  - Includes dependency for guava
  - Creates an "uber" JAR for upload to AWS

Summarization

- Other summarizations of interest
  - Min, max, mean

- Suppose we are interested in these metrics for paragraph length (Assignment 2 data)
  - If paragraph lengths are normally distributed, then the median will be very near the mean
  - If the distribution of paragraph lengths is skewed, then the mean and median will be very different

Summarization

- Min and max are straightforward
- For each paragraph, output two values
  - Min length (the length of the current paragraph)
  - Max length (the length of the current paragraph)
  - Key?

- Combiner will get a list of value pairs
  - Select the min, max from the list, output the values
  - Key?
- Reducer does the same

Summarization

- Median
  - Get all the values, sort them, then find the middle
- Since our computation is distributed, no mapper sees all the values
- Should we send them all to one reducer?
  - Not utilizing map-reduce (computation not distributed)
  - Data sizes likely too large to keep in memory

Summarization

- Median
  - Keep the unique paragraph lengths, and
  - The frequency of each length

- Map output:
  - <paragraph length, 1>

- Combiner gets a list of these pairs and updates the count for recurring lengths
- Reducer does the same, then computes the median

Summarization

- Median
  - Hadoop provides the **`SortedMapWritable`** class
  - Can associate a frequency count with a paragraph length
  - Keeps the lengths in sorted order

- See the example in Chapter 2 of *Map-Reduce Design Patterns*

- How could we compute all in one pass over the data?
  - min, max, median

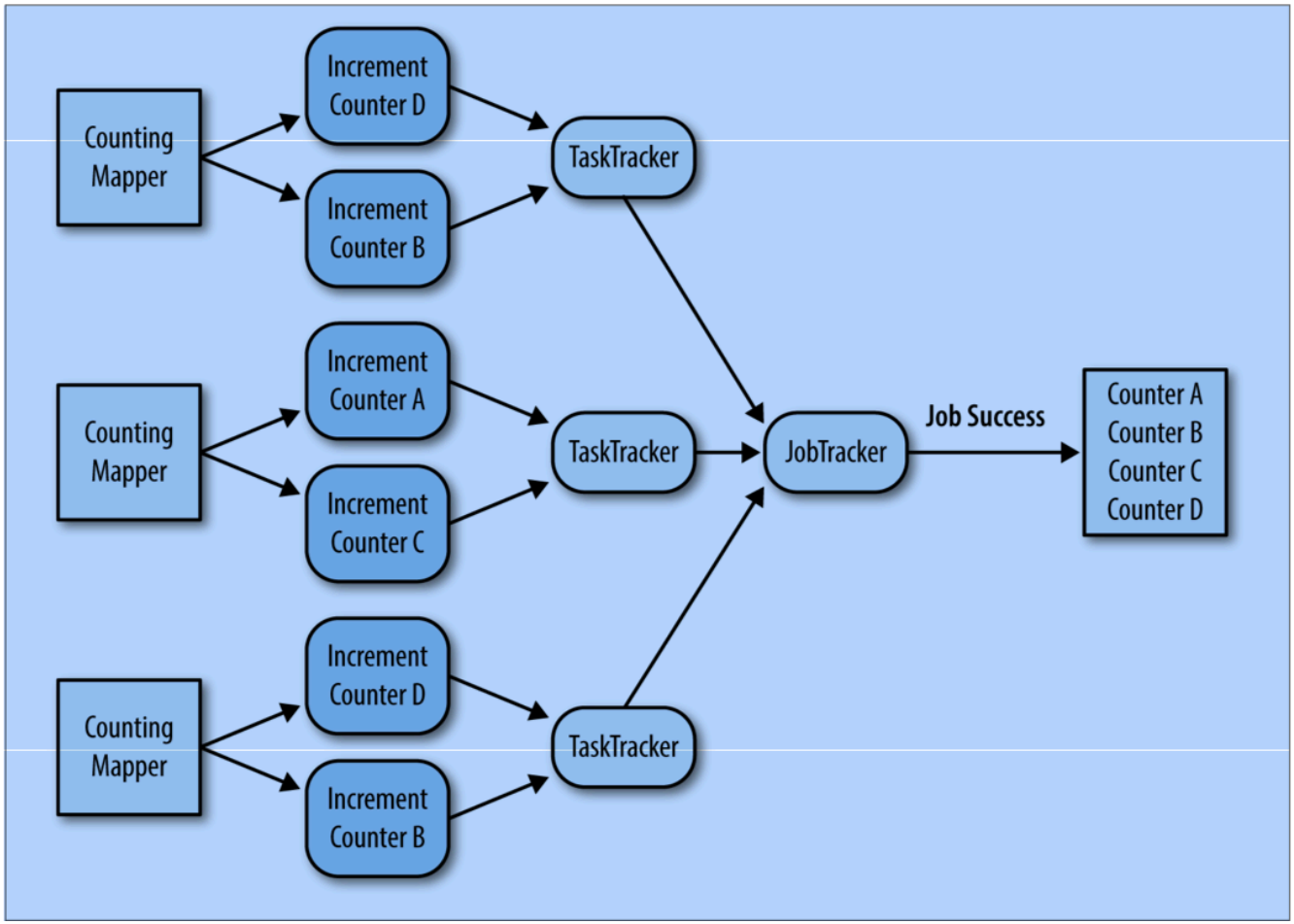
Counters

- Hadoop map-reduce infrastructure provides counters
  - Accessed by group name
  - Cannot have a large number of counters
    - For example, can't use counters to solve WordCount
  - A few tens of counters can be used

- Counters are stored in memory on **`JobTracker`**

Counters

Figure 2-6, MapReduce Design Patterns



How Hadoop MapReduce Works

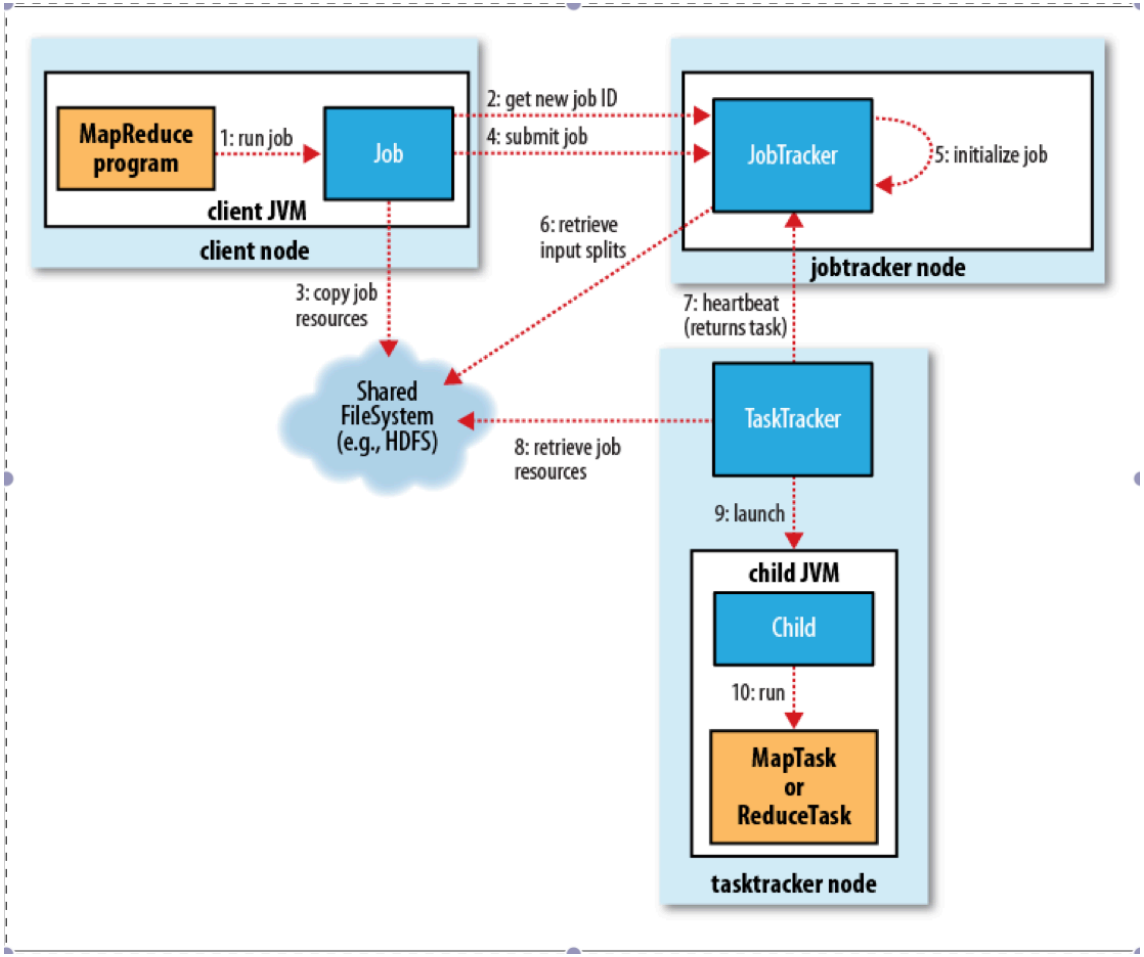
- We've seen some terms like:
  - Job
  - JobTracker
  - TaskTracker

- Let's look at what they do

- Details from Chapter 6, *Hadoop: The Definitive Guide 3rd Edition*

How Hadoop MapReduce Works

Figure 6-1, Hadoop: The Definitive Guide 3rd Edition



Job Submission

- Job submission
  - Input files exist?
  - Output directory exist?
    - If yes, it fails. Hadoop expects to create this directory
  - Copy resources to HDFS
    - JAR files
    - Configuration file
    - Computed file splits

Job Tracker

- Creates tasks (work to be done)
  - Map task for each input split
  - Requested number of reducer tasks
  - Job setup, job cleanup tasks

- Map tasks are assigned to task trackers that are “close” to the input split location
  - Data local preferred
  - Rack local next

- Reduce task can go anywhere. Why?
- Scheduling algorithm orders the tasks

Task Tracker

- Configured for several map and reduce tasks
- Periodically sends a “heartbeat” to job tracker
  - “I’m still alive”
  - “Ready for new task”

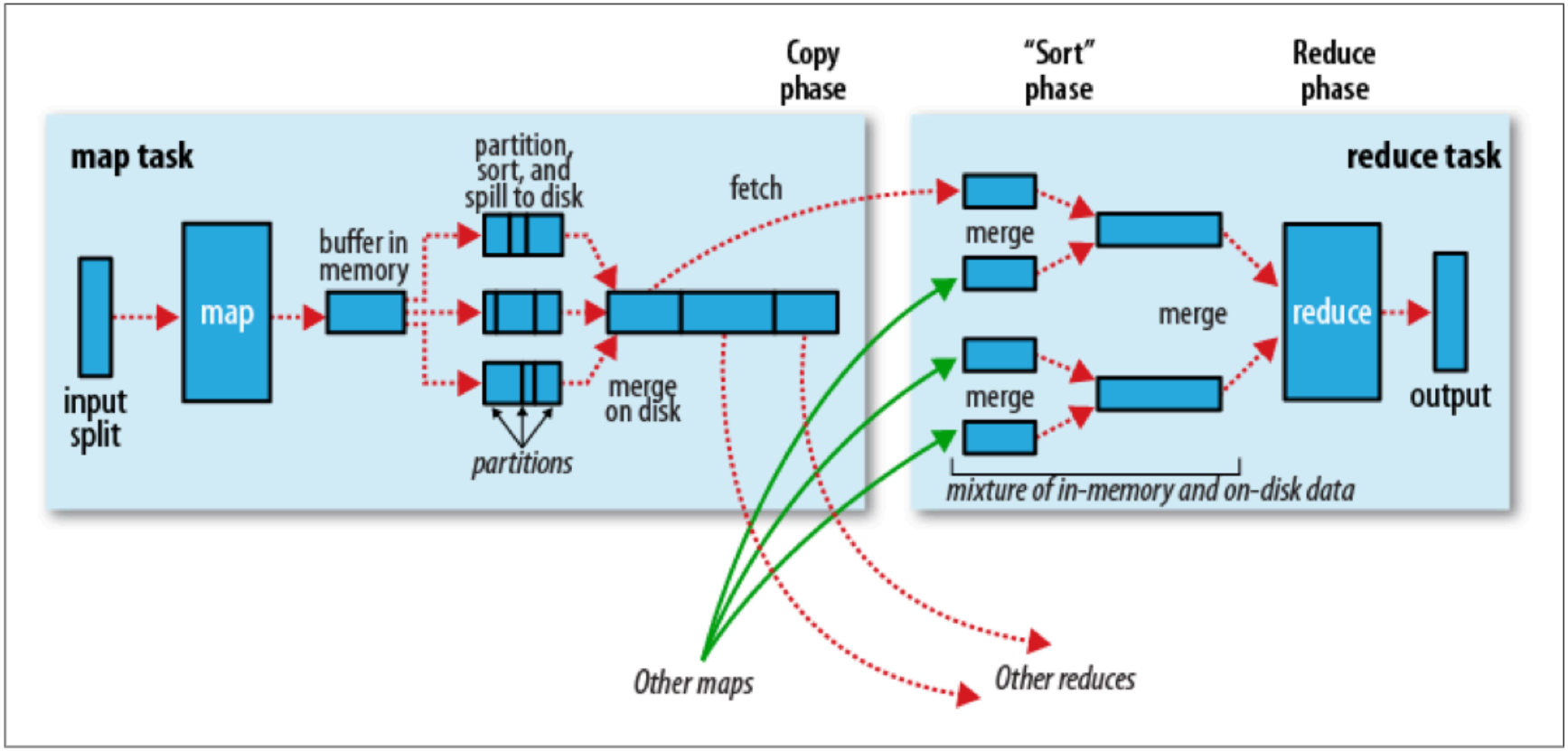
- For a new task
  - Copy files to local file system (JAR, configuration)
  - Launch a new JVM (**`TaskRunner`**)
  - Load the mapper/reducer class and call its method
  - Update the task tracker progress

Task Progress

- Mapper
  - What portion of the input has been processed
- Reducer – more complicated
  - Sort, shuffle and reduce are considered here
  - Progress is an estimate of how much of the total work has been done

Shuffle

Figure 6-6, Hadoop: The Definitive Guide 3rd Edition



MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide