The Map Reduce Framework

CompSci 590.03
Instructor: Ashwin Machanavajjhala

Duke UNIVERSITY

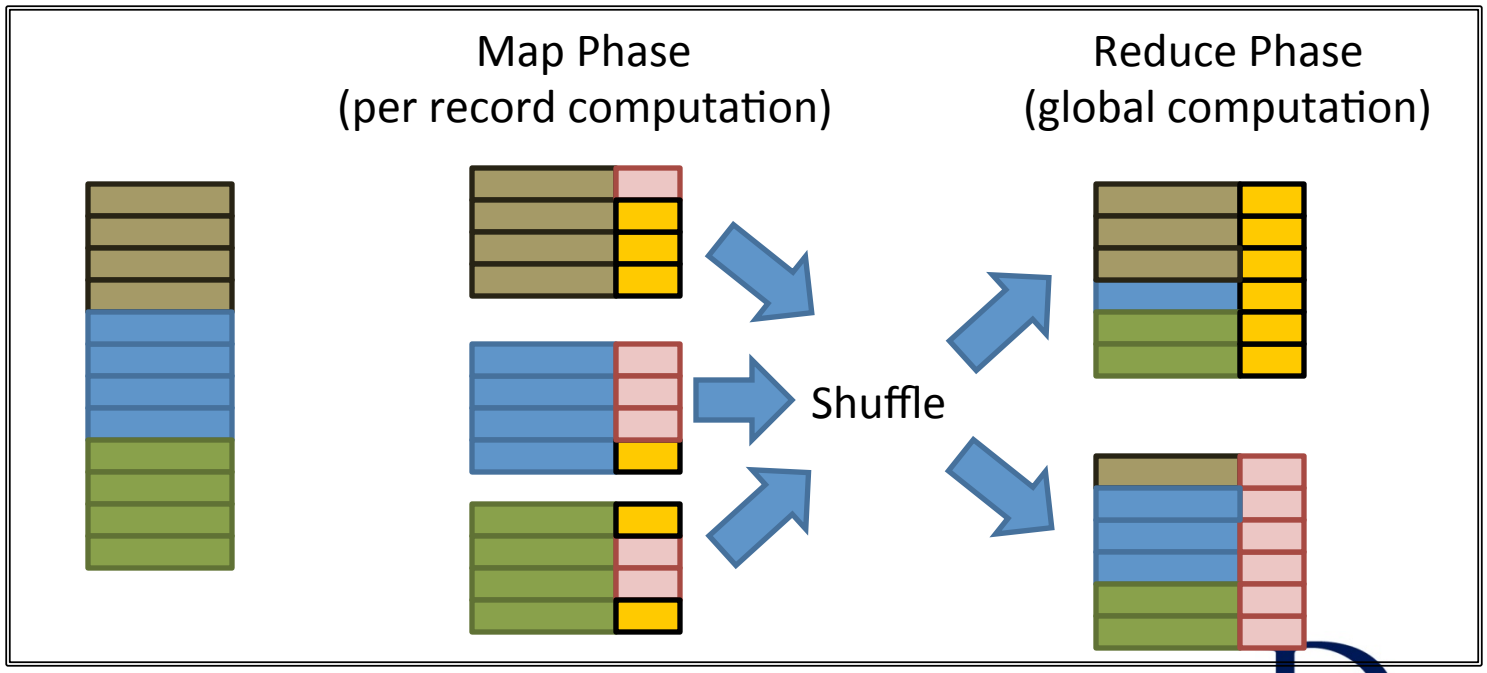
This Class

- Map Reduce Programming Framework
- Map Reduce System Implementation

MAP REDUCE FRAMEWORK

Map-Reduce

```
map     (k1,v1)          → list(k2,v2);
reduce  (k2,list(v2))    → list(k3,v3).
```



Duke
UNIVERSITY

Running Example: Word Count

- Input: A set of documents D, each containing a list of words.
- Output: <w, c>, where c is the number of times word w appears across all documents.

Map Task

- Divides the large file into small chunks.
- One *mapper* is assigned to each chunk (and chunks are typically distributed across machines).

- A UDF (user defined function) is applied to each (key, value) pair in the chunk.
- The UDF creates zero, one or more new (key, value) pairs for every input record.

Word Count

- <docid, {list of words}> → {list of <word, 1>}

  The mapper takes a document d and creates n key value pairs, one for each word in the document.

  The output key is the word

  The output value is 1 (count of each appearance of a word)

Reduce Task

- A reduce task aggregates the keys from different mappers.
- A *reducer* takes all the key value pairs sharing the same key and applies a reduce function to the set of values to generate one output key value pair.
- Shuffle Phase: Reducers are distributed across many machines by hashing key values to different machines.

Word Count

- <word, {list of 1s}> → <word, count>

In this case, the reducer just adds up the count values in each of the tuples with the same key.

Combiner

- For certain types of reduce functions (commutative and associative), one can decrease the communication cost by running the reduce function within the mappers.
  - SUM, COUNT, MAX, MIN …

- <docid, {list of words}> → <word, c>

  where c is the number of times the word appears in the mapper.

Distributed Grep

- Map:
  <lineid, string> → <lineid, string> *//if string matches pattern*

- Reduce: *Identity function.*

Matrix Multiplication

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

Matrix Multiplication

- Map:

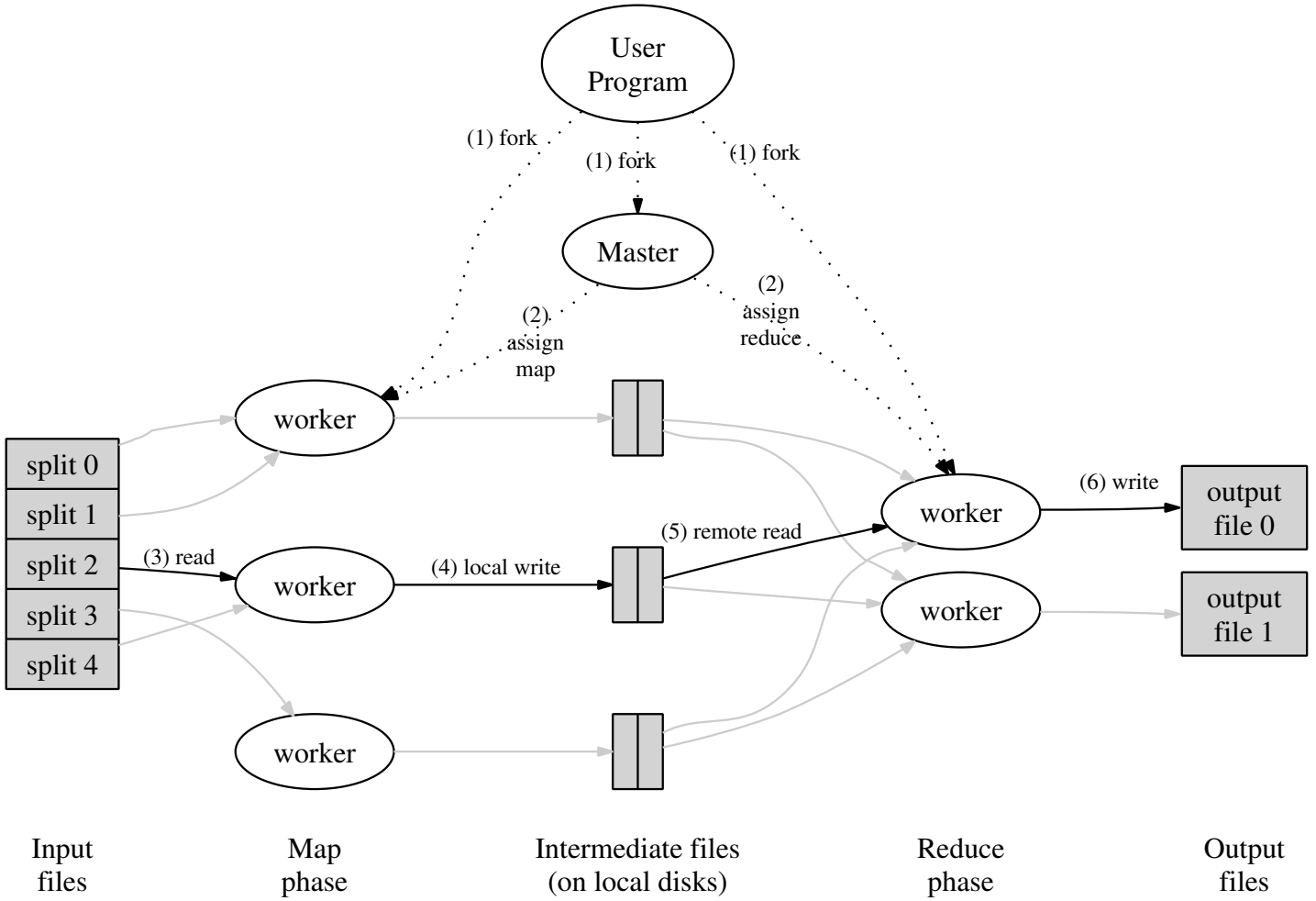
  $<(i,j), m_{ij}> \rightarrow <j, (M, i, m_{ij})>$
  $<(j,k), n_{jk}> \rightarrow <j, (N, k, n_{jk})>$

- Reduce:

  $<j, \{(M, I, m_{ij}), (N, k, n_{jk}) \ldots\}> \rightarrow \{ <(i,k), \Sigma m_{ij}n_{jk}>\}$

MAP REDUCE SYSTEM

System



Duke UNIVERSITY

System

- Workers/machines are typically commodity hardware
  - Arranged on racks
  - Connected by gigabit ethernets
- Storage is inexpensive hard disks
- Since there are thousands of machines in a cluster, failures are to be expected.

Fault Tolerance

- Map reduce runs over a distributed file system (GFS or HDFS) which ensure availability by replicating the data (e.g., 3x).
- Master (or job tracker) pings each worker periodically.
- If a mapper or a reducer fails midway, then the task is restarted.
- Map (or reduce) phase waits for all the mappers (or reducers) to complete successfully.

Map Execution

- Data is divided into M splits, and one work is assigned to each split
- Worker applies map function to each record in the split.
- Resulting key-value pairs are stored into disk divided into R partitions.

Reduce Execution

- Keys output by the map phase are divided into R partitions using a *partition function*
  - E.g., hash(key) mod R
- The i^{th} reduce worker reads the i^{th} partition output by each map using remote procedure calls

- Data is sorted based on the keys so that all occurrences of the same key are close to each other.

- Reducer iterates over the sorted data and passes all records from the same key to the reduce function.