MapReduce Basics

CHAPTER 2 LIN AND DYER &

HTTP://DEVELOPER.YAHOO.COM/HADOOP/TUTORIAL/

Lin and Dyer's text

Chapter 1: Please read: sets the context for MR

Chapter 2: MR Basics: analysis of a sample problem analysis/walkthrough

Chapter 3: MR Algorithm Design (up to p.60)

Chapter 4: Inverted index for text retrieval

Chapter 5: Graph algorithms (pagerank and other classical algorithms)

Big Ideas

Read Chapter 1

1. Scale out and not scale up
2. Assume failures are common
3. Move processing to data
4. Process sequentially and avoid random access (deterministic)
5. Hide system level details from the developer
6. Seamless scalability

MR: Divide and Conquer Algorithm

How do we break up a large problem into smaller tasks? More specifically, how do we decompose the problem so that the smaller tasks can be executed in parallel?

How do we assign tasks to workers distributed across a potentially large number of machines (while keeping in mind that some workers are better suited to running some tasks than others, e.g., due to available resources, locality constraints, etc.)?

How do we ensure that the workers get the data they need?

How do we coordinate synchronization among the different workers?

How do we share partial results from one worker that is needed by another?

How do we accomplish all of the above in the face of software errors and hardware faults?

Basics

map: (k1,v1) --> [(k2,v2)]

reduce: (k2,[v2])-->[(k3,v3)]

Implicit between map and reduce is the implicit "group by" operation of intermediate keys

Hadoop does not have the GFS restriction that reducer's input key and output key type should be same.

Complete Example

Figure 2.2: Simplified MR process (picture)

Figure 2.3: MR Algorithm (pseudo code)

Figure 2.4: MR process with combiners and partitioners (picture)

Your answers to midterm questions should be in this format.

Lets go through couple of examples: (i) word count (ii) max value for each key

Simple MR

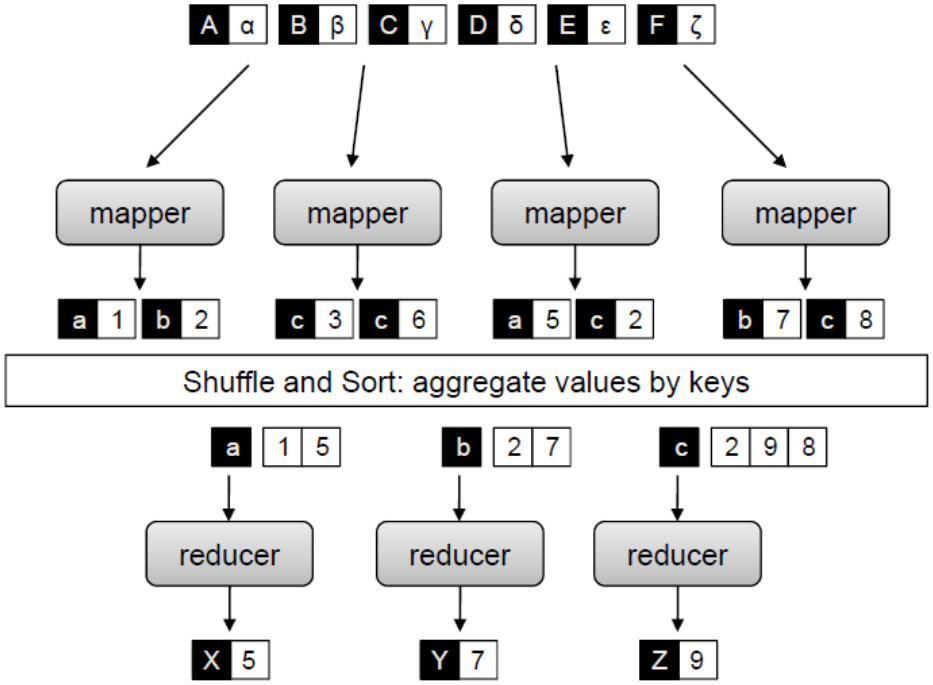


Figure 2.2: Simplified view of MapReduce. Mappers are applied to all input key-value pairs,

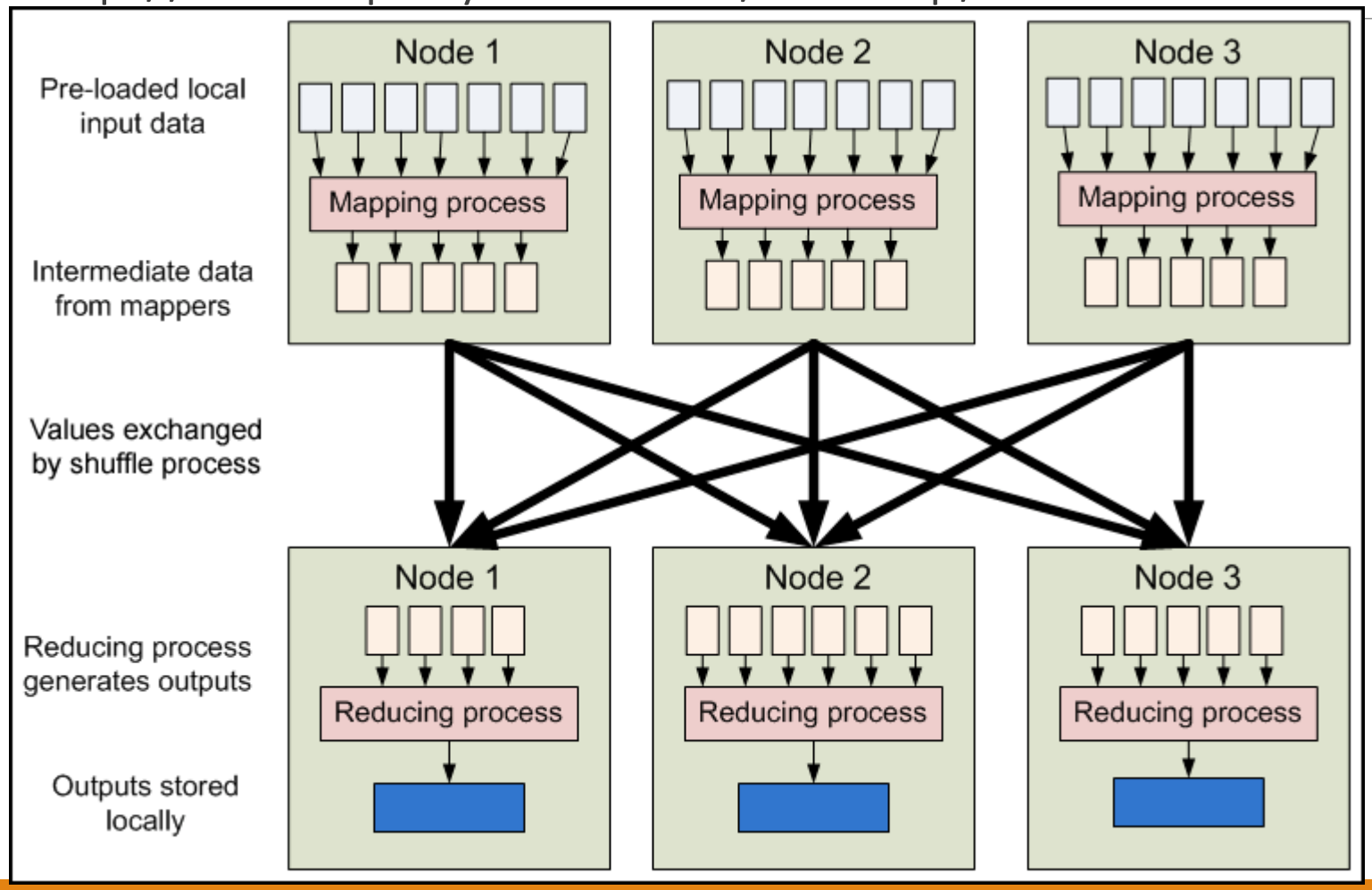
Simple MR Algorithm

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

Figure 2.3: Pseudo-code for the word count algorithm in MapReduce. The mapper emits an intermediate key-value pair for each word in a document. The reducer sums up all counts for each word.

Architecture
http://developer.yahoo.com/hadoop/tutorial



Combiners and Partitioners

Combiner: combines the same “keys” at the output of a mapper

Runs after the Mapper and before the Reducer. Usage of the Combiner is optional.

The Combiner will receive as input all data emitted by the Mapper instances on a given node.

The output from the Combiner is then sent to the Reducers, instead of the output from the Mappers.

The Combiner is a "mini-reduce" process which operates only on data generated by one machine.

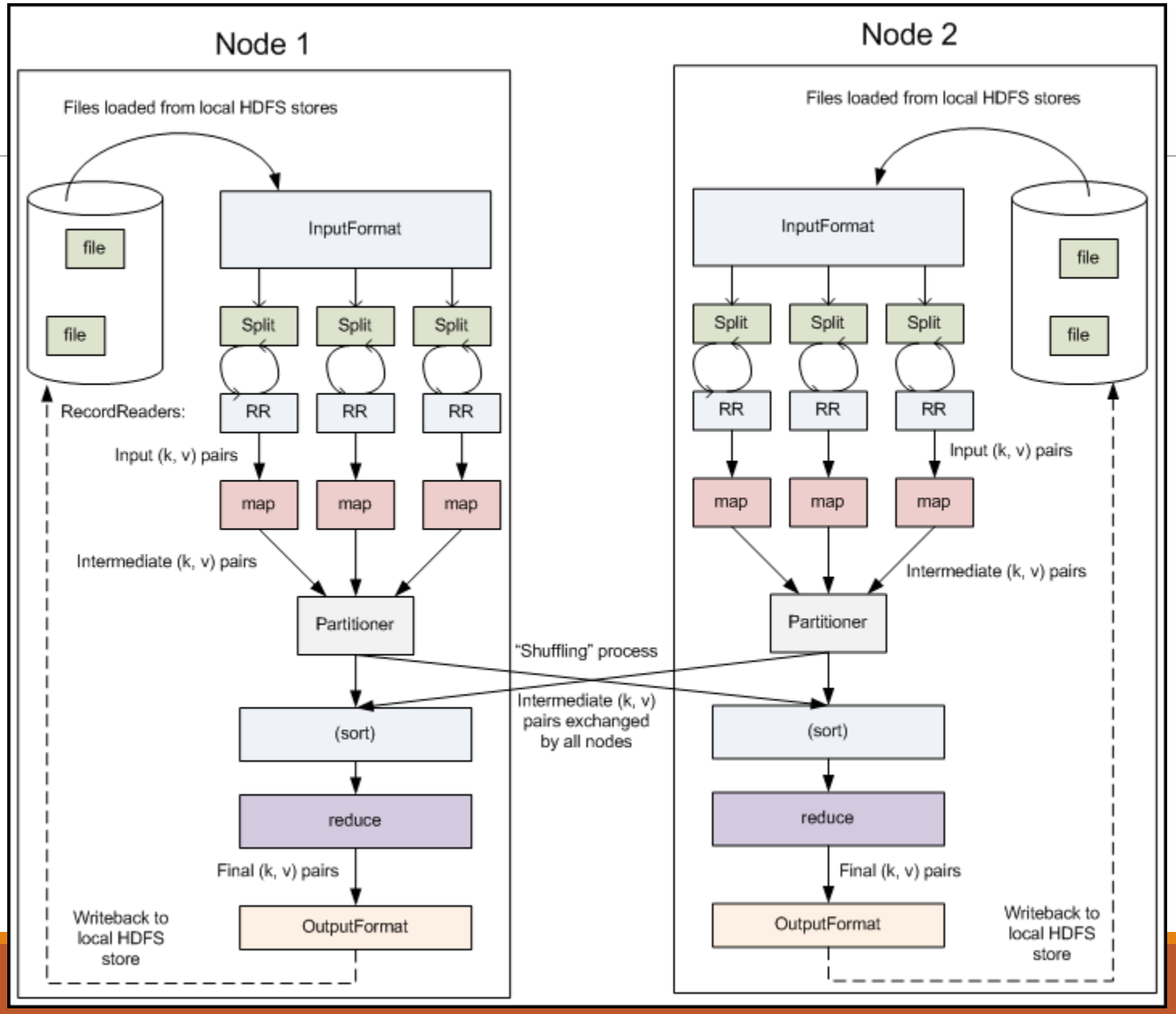
This process of moving map outputs to the reducers is known as shuffling.

A different subset of the intermediate key space is assigned to each reduce node; these subsets (known as "partitions") are the inputs to the reduce tasks.

Default partition function is a hash function.

Each map task may emit (key, value) pairs to any partition; all values for the same key are always reduced together regardless of which mapper is its origin.

MR Workflow


Node 1
Files loaded from local HDFS stores
file
file
InputFormat
Split
Split
Split
RecordReaders:
RR
RR
RR
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes
(sort)
reduce
Final (k, v) pairs
OutputFormat
Writeback to local HDFS store
Node 2
Files loaded from local HDFS stores
InputFormat
file
file
Split
Split
Split
RR
RR
RR
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Partitioner
(sort)
reduce
Final (k, v) pairs
OutputFormat
Writeback to local HDFS store

MR with Combiners & Partitioners

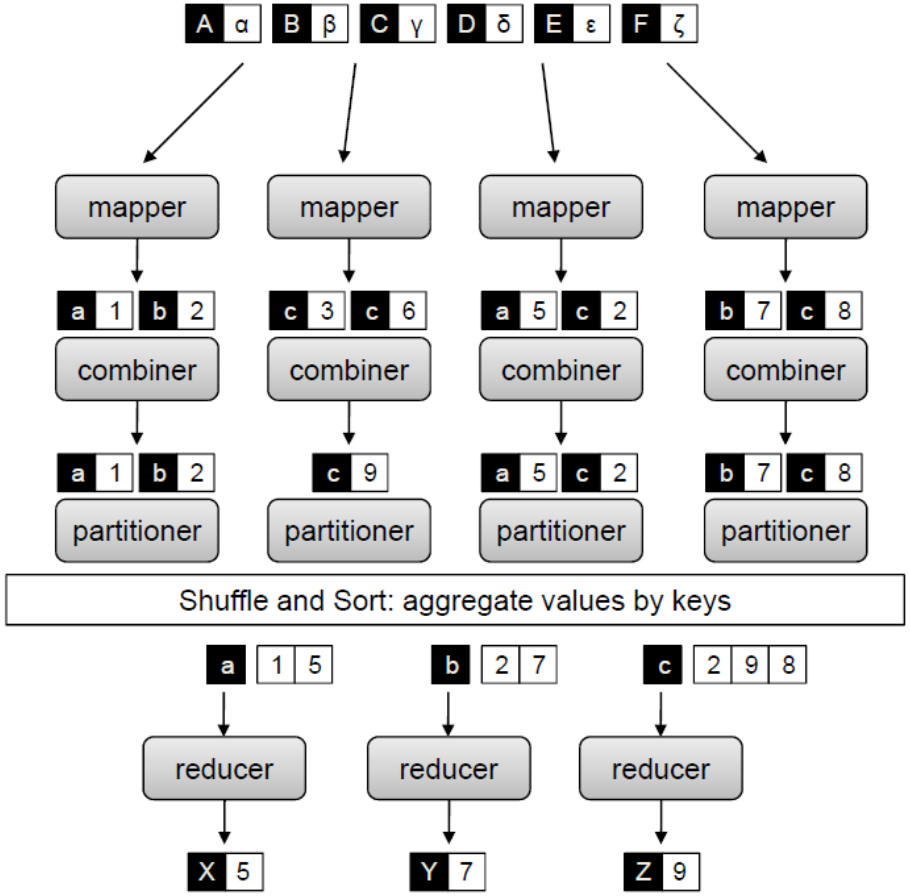


Figure 2.4: Complete view of MapReduce, illustrating combiners and partitioners in addition to mappers and reducers. Combiners can be viewed as "mini-reducers" in the map phase. Partitioners determine which reducer is responsible for a particular key.

HDFS (Recall) : not to scale!

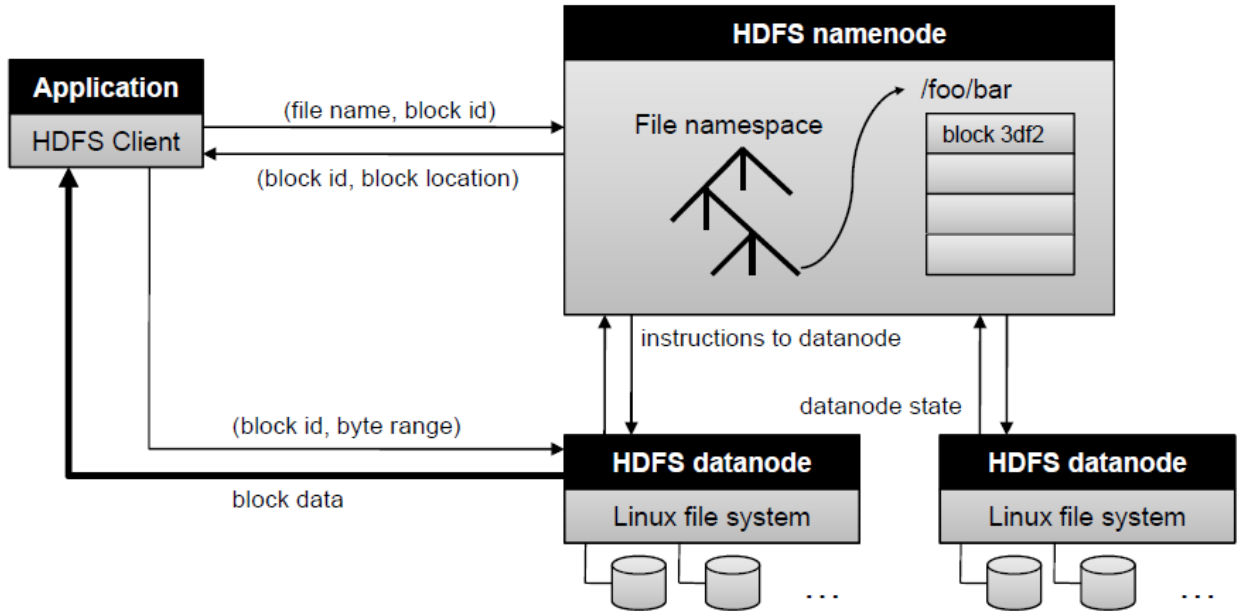


Figure 2.5: The architecture of HDFS. The namenode (master) is responsible for maintaining the file namespace and directing clients to datanodes (slaves) that actually hold data blocks containing user data.

Summary

We learned

Big ideas behind MR. Hadoop

Simple mapreduce for wordcount

Role of combiners and partitioners

HDFS architecture with namenode and datanode