What is MapReduce

- Origin from Google, [OSDI'04]
  - MapReduce: Simplified Data Processing on Large Clusters
  - Jeffrey Dean and Sanjay Ghemawat
- Programming model for parallel data processing
- Hadoop can run MapReduce programs written in various languages: e.g. Java, Ruby, Python, C++
- For large-scale data processing
  - Exploits large set of commodity computers
  - Executes process in a distributed manner
  - Offers high availability

Motivation for MapReduce

- There was need for an abstraction that hides many system-level details from the programmer.

- MapReduce addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a ***scalable***, ***robust***, and ***efficient*** manner.

- MapReduce separates the ***what*** from the ***how***

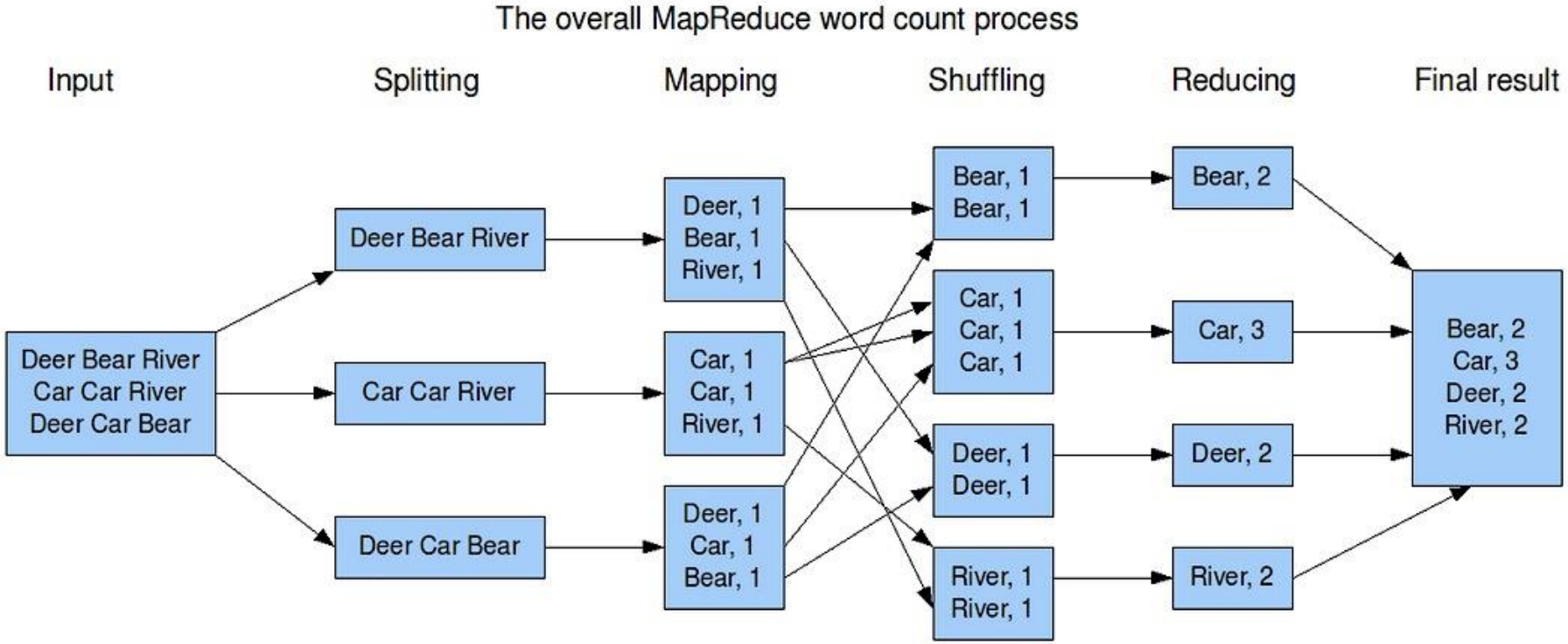
The Idea of MapReduce

- Inspired by the map and reduce functions in functional programming
- We can view map as a transformation over a dataset
  - This transformation is specified by the function *f*
  - Each functional application happens in isolation
  - The application of *f* to each element of a dataset can be parallelized in a straightforward manner
- We can view reduce as an aggregation operation
  - The aggregation is defined by the function *g*
  - Data locality: elements in the list must be “brought together”
  - If we can group elements of the list, also the reduce phase can proceed in parallel
- The framework coordinates the map and reduce phases:
  - Grouping intermediate results happens in parallel

Everything Else?

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed file system (HDFS)
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

MapReduce Example - WordCount



- Hadoop MapReduce is an implementation of MapReduce
  - MapReduce is a computing paradigm (Google)
  - Hadoop MapReduce is an open-source software

Data Structures in MapReduce

- Key-value pairs are the basic data structure in MapReduce
  - Keys and values can be: integers, float, strings, raw bytes
  - They can also be arbitrary data structures, but must be comparable (for sorting)

- The design of MapReduce algorithms involves:
  - Imposing the key-value structure on arbitrary datasets
    - E.g.: for a collection of Web pages, input keys may be URLs and values may be the HTML content
  - In some algorithms, input keys are not used (e.g., wordcount), in others they uniquely identify a record
  - Keys can be combined in complex ways to design various algorithms

Map and Reduce Functions

- Programmers specify two functions:
  - **map** $(k_1, v_1) \rightarrow$ list $[<k_2, v_2>]$
    - Map transforms the input into key-value pairs to process
  - **reduce** $(k_2,$ list $[v_2]) \rightarrow [<k_3, v_3>]$
    - Reduce aggregates the list of values for each key
    - All values with the same key are sent to the same reducer
  - list $[<k_2, v_2>]$ will be grouped according to key k_2 as $(k_2,$ list $[v_2])$

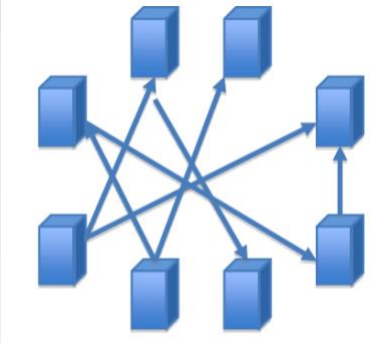
- The MapReduce environment takes in charge of everything else…

- A complex program can be decomposed as a succession of Map and Reduce tasks

Understanding MapReduce

- Map>>
  - (K1, V1) →
    - Info in
    - Input Split
  - list (K2, V2)
    - Key / Value out (intermediate values)
    - One list per local node
    - Can implement local Reducer (or Combiner)

Shuffle/Sort>>



Reduce

(K2, list(V2)) →

- Shuffle / Sort phase precedes Reduce phase
- Combines Map output into a list

list (K3, V3)

- Usually aggregates intermediate values

(input) <k1, v1> → **map** → <k2, v2> → **combine** → <k2, list(V2)> → **reduce** → <k3, v3> *(output)*

WordCount – Mapper/Reducer

- Let's count number of each word in documents (e.g., Tweets/Blogs)
  - Reads input pair <k1,v1>
    - The input to the mapper is in format of <docID, docText>:

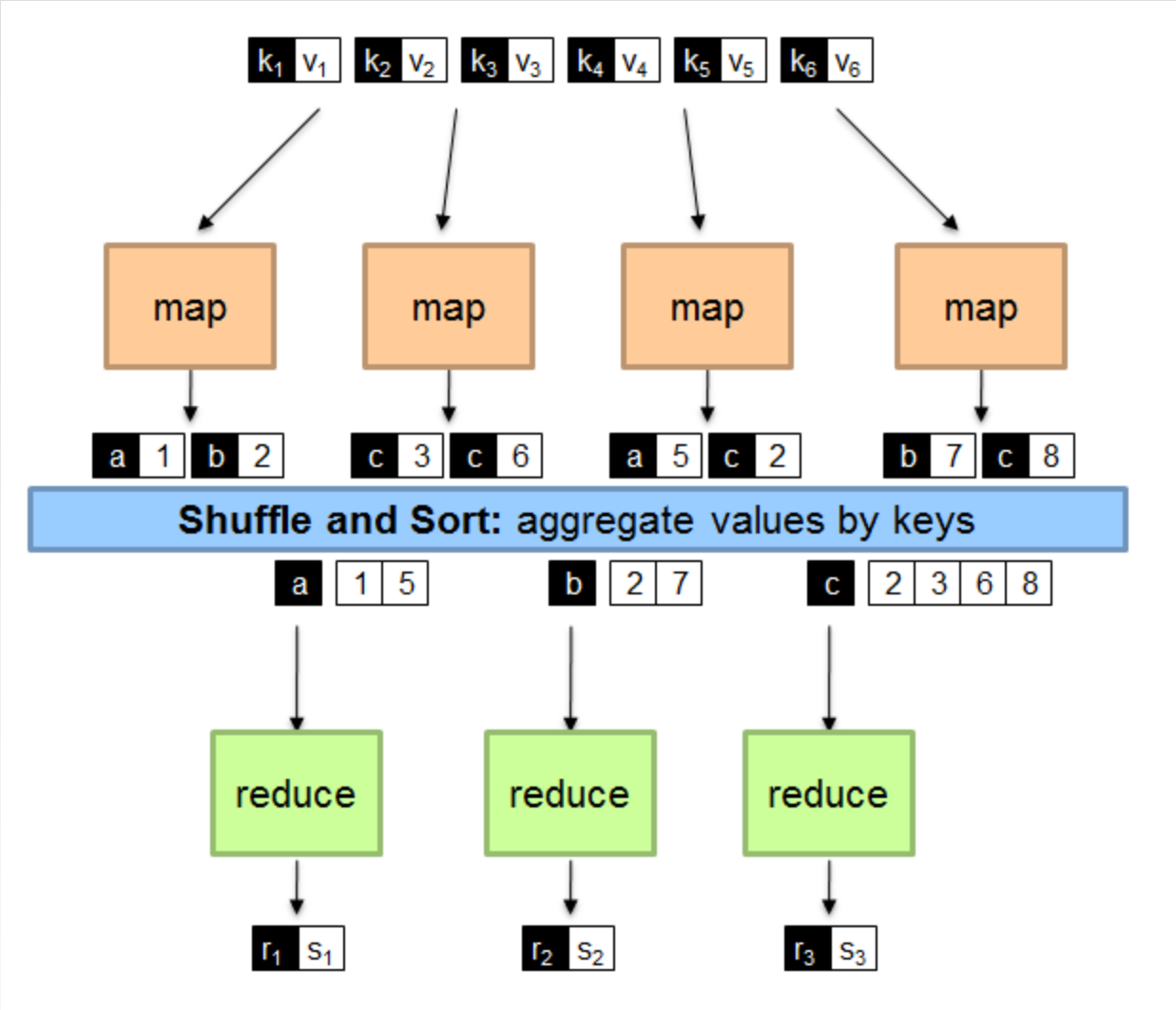
    <D1,"Hello World" >,<D2,"Hello Hadoop Bye Hadoop" >
  - Outputs pairs <k2, v2>
    - The output of the mapper is in format of <term, 1>:

  <Hello, 1><World, 1><Hello, 1><Hadoop, 1><Bye, 1><Hadoop, 1>
  - After shuffling and sort, reducer receives <k2, list(v2)>

  <Hello, {1, 1}><World, {1}><Hadoop, {1, 1}><Bye, {1}>
  - The output is in format of <k3, v3>:

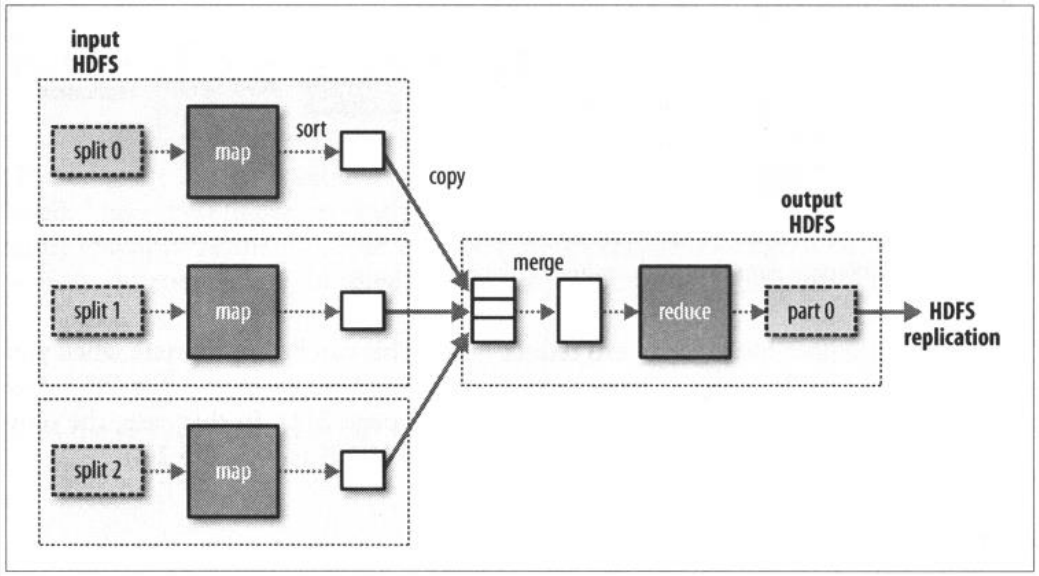
  <Hello, 2><World, 1><Hadoop, 2><Bye, 1>

A Brief View of MapReduce



Shuffle and Sort

- Shuffle
  - Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.
- Sort
  - The framework groups Reducer inputs by keys (since different Mappers may have output the same key) in this stage.
- Hadoop framework handles the Shuffle and Sort step .



“Hello World” in MapReduce

- Input:
  - Key-value pairs: (docid, doc) of a file stored on the distributed filesystem
  - docid : unique identifier of a document
  - doc: is the text of the document itself
- Mapper:
  - Takes an input key-value pair, tokenize the line
  - Emits intermediate key-value pairs: the word is the key, and the integer is the value
- The framework:
  - Guarantees all values associated with the same key (the word) are brought to the same reducer
- The reducer:
  - Receives all values associated to some keys
  - Sums the values and writes output key-value pairs: the key is the word, and the value is the number of occurrences