2.2.1 Word Count Using MapReduce

- **map(key, value)**:

```
// key: document name; value: text of the document
        for each word w in value:
                emit(w, 1)
```

- **reduce(key, values):**

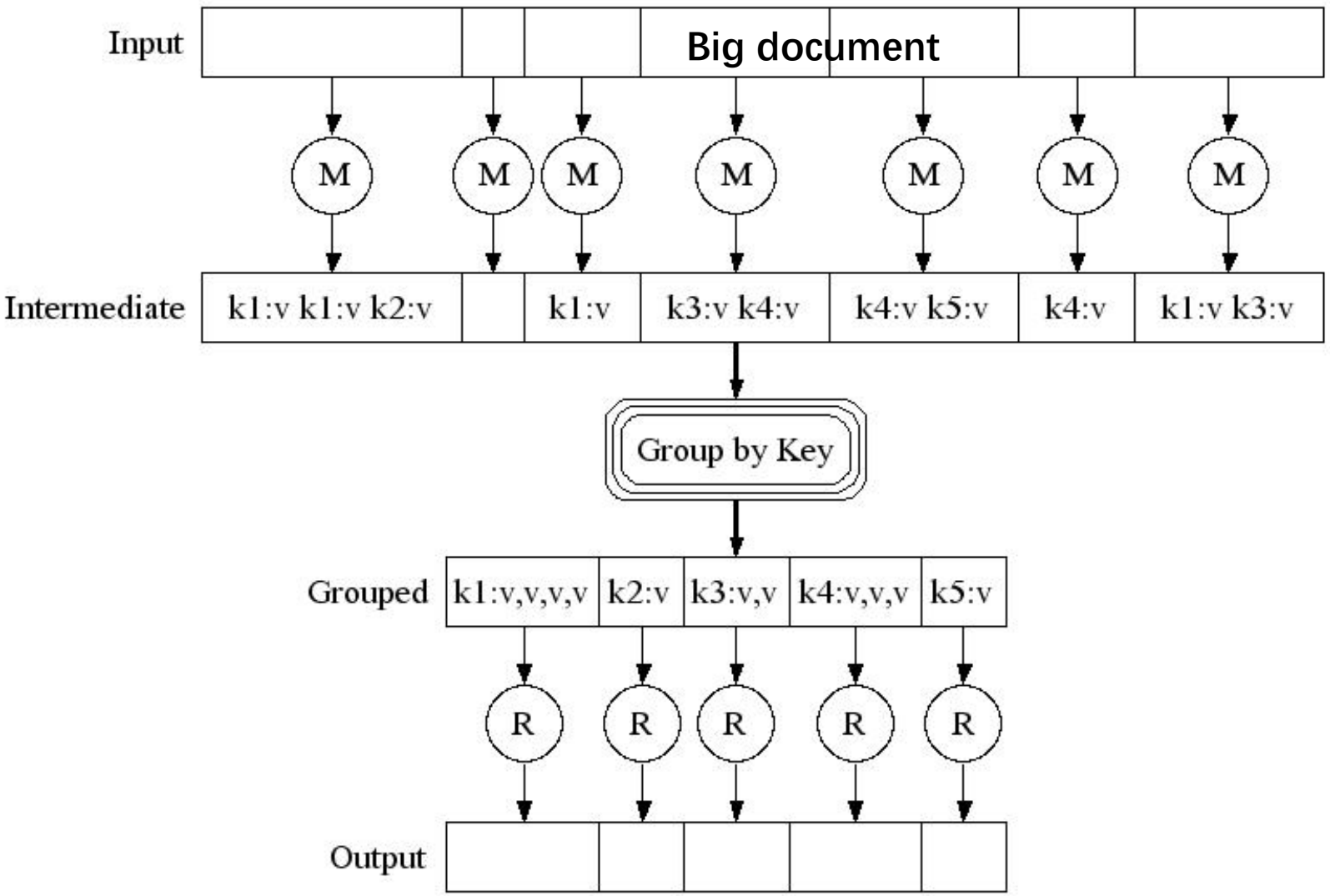
```
// key: a word; value: an iterator over counts
        result = 0
        for each count v in values:
                result += v
        emit(key, result)
```

2.2.2 Map-Reduce: A diagram

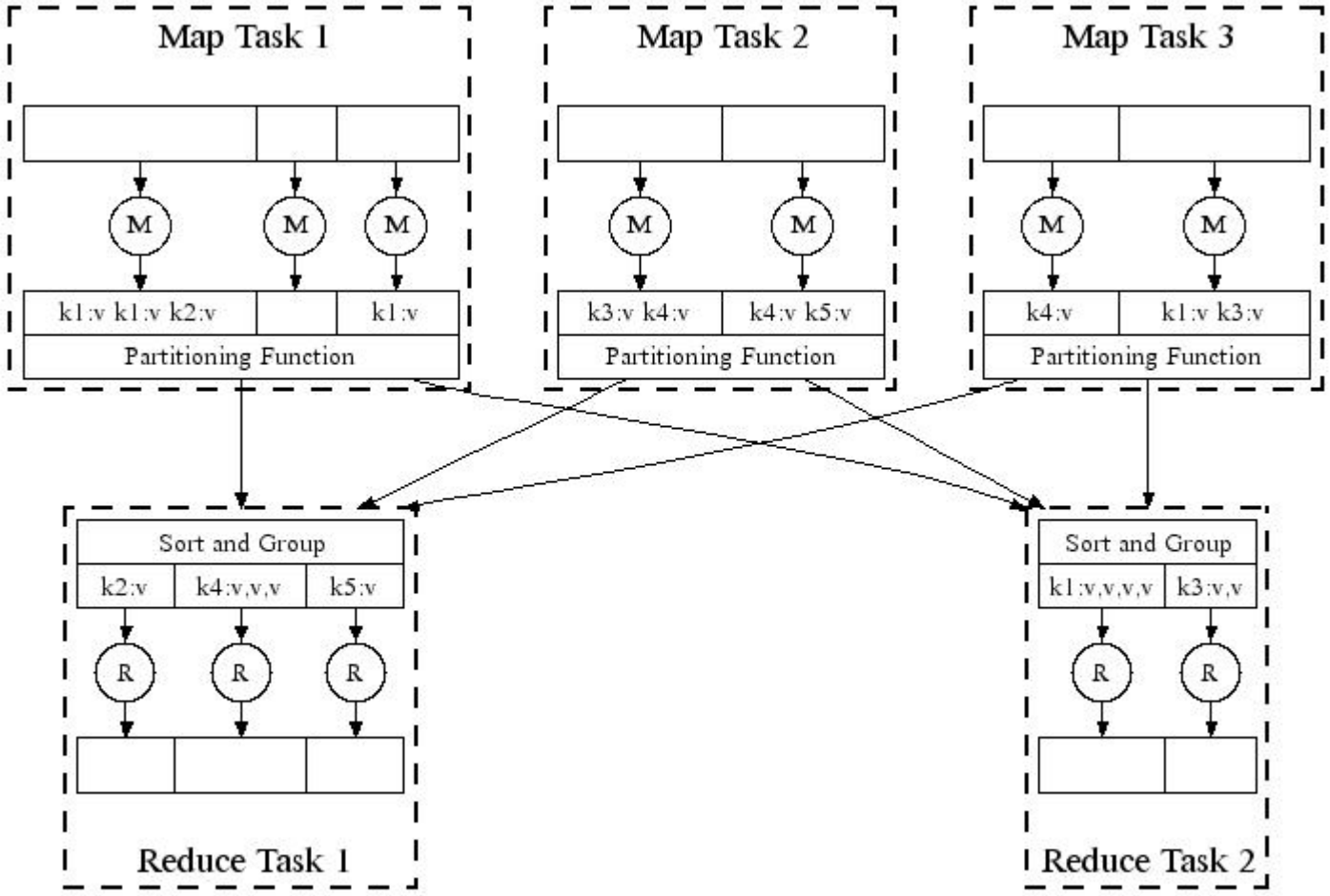
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



2.2.2 Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

2.2.2 Map-Reduce: Environment

- Map-Reduce environment takes care of:
  - Partitioning the input data
  - Scheduling the program's execution across a set of machines
  - Performing the group by key
  - Handling machine failures
  - Managing required inter-machine communication

2.2.2 Data Flow

- Input and final output are stored on a **distributed file system (DFS)**:
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on **local FS** of Map and Reduce workers
- Output is often input to another MapReduce task

2.2.2 Coordination: Master

- Master node takes care of coordination:
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers

- Master pings workers periodically to detect failures.
  - How to deal with failures?

2.2.2 Dealing with Failures

- Map worker failure
  - Map tasks completed or in-progress at worker are **reset to idle**
  - Reduce workers are notified when task is rescheduled on another worker

- Reduce worker failure
  - Only **in-progress tasks are reset** to idle
  - Reduce task is restarted

- Master failure
  - MapReduce task is aborted and client is notified

2.2.2 How many Map and Reduce jobs?

- M map tasks, R reduce tasks
- Rule of a thumb:
  - Make M much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
  - Usually R is smaller than M, because output is spread across R files

Section 2.3: MapReduce Refinements

Content

1 Backup Tasks

2 Combiners

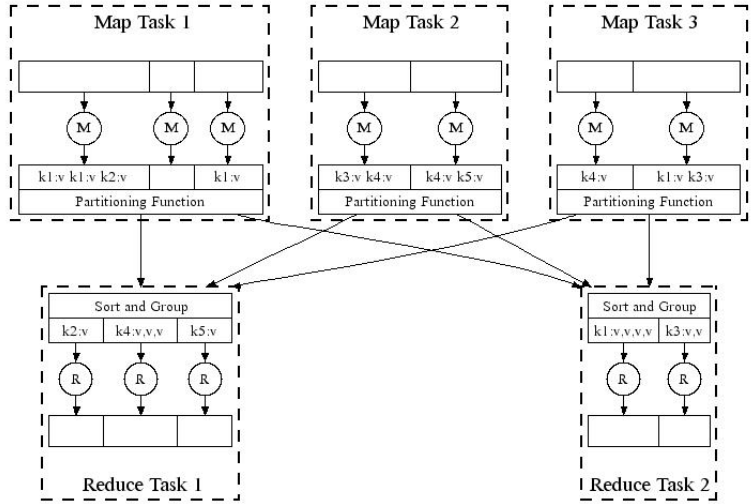
3 Partition Function

2.3.1 Refinements: Backup Tasks

- Problem
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- Solution
  - Near end of phase, spawn **backup** copies of tasks
    - Whichever one finishes first "wins"
- Effect
  - Dramatically shortens job completion time

2.3.2 Refinements: Combiners(组合器)

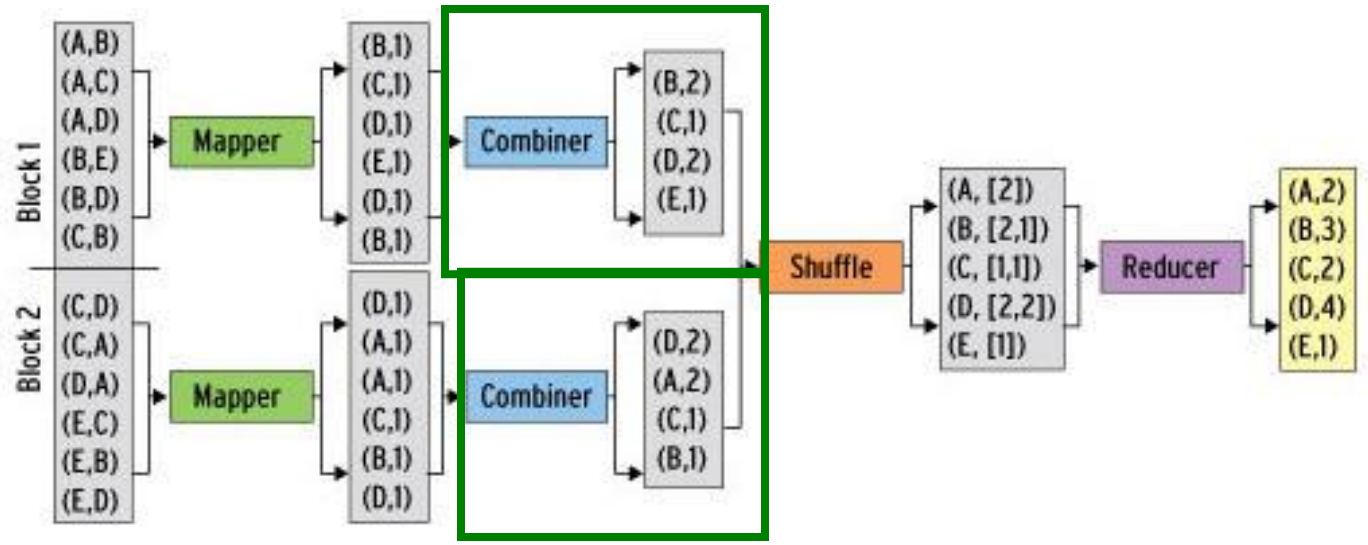
- Often a Map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in the word count example
- Can save network time by pre-aggregating values in the mapper:
  - combine(k, list(v1)) → v2
  - Combiner is usually same as the reduce function



- Works only if reduce function is commutative and associative(交换律和结合律), e.g., sum

2.3.2 Refinements: Combiners(组合器)

- Back to our word counting example:
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

2.3.3 Refinements: Partition Function(分区)

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - **hash(key) mod R**
- Sometimes useful to **override the hash function**:
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

2.3.3 Example: Host size

- **Suppose we have a large web corpus (语料库) with a metadata file** formatted as follows:
  - Each record of the form: (URL, size, date, …)
- **We want to: For each host (not each URL), we want to find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host
- Map: For each record, output(hostname(URL),size)
- Reduce: sum the size of each host

2.3.3 Example: Join By Map-Reduce

- Compute the natural join R(A,B) ⋈ S(B,C). R and S are each stored in files. Tuples are pairs (a,b) or (b,c)

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

- **Map**: (b,(R,a))for each tuple on R; (b,(S,c))for each tuple on S
- **Reduce**:same key with (R,a) or (S,c), then output only (a,c). key is irrelevant.

Chapter 2: Summary

- MapReduce uses **parallelization + aggregation** (并行和聚集, 或者并行及串行) to schedule applications across clusters
  - Plenty of ongoing research work in scheduling and fault-tolerance for Mapreduce

- MapReduce **problems**:
  - Many problems aren't easily described as MapReduce
  - Persistence to disk typically slower than in-memory work

- Alternative: **Apache Spark**
  - a general-purpose processing engine
  - 有兴趣的同学课外自学相关内容