빅데이터 분석 및 응용

L03: MapReduce Data Flow

Summer 2020

Kookmin University

Map-Reduce: Environment

Map-Reduce environment takes care of:

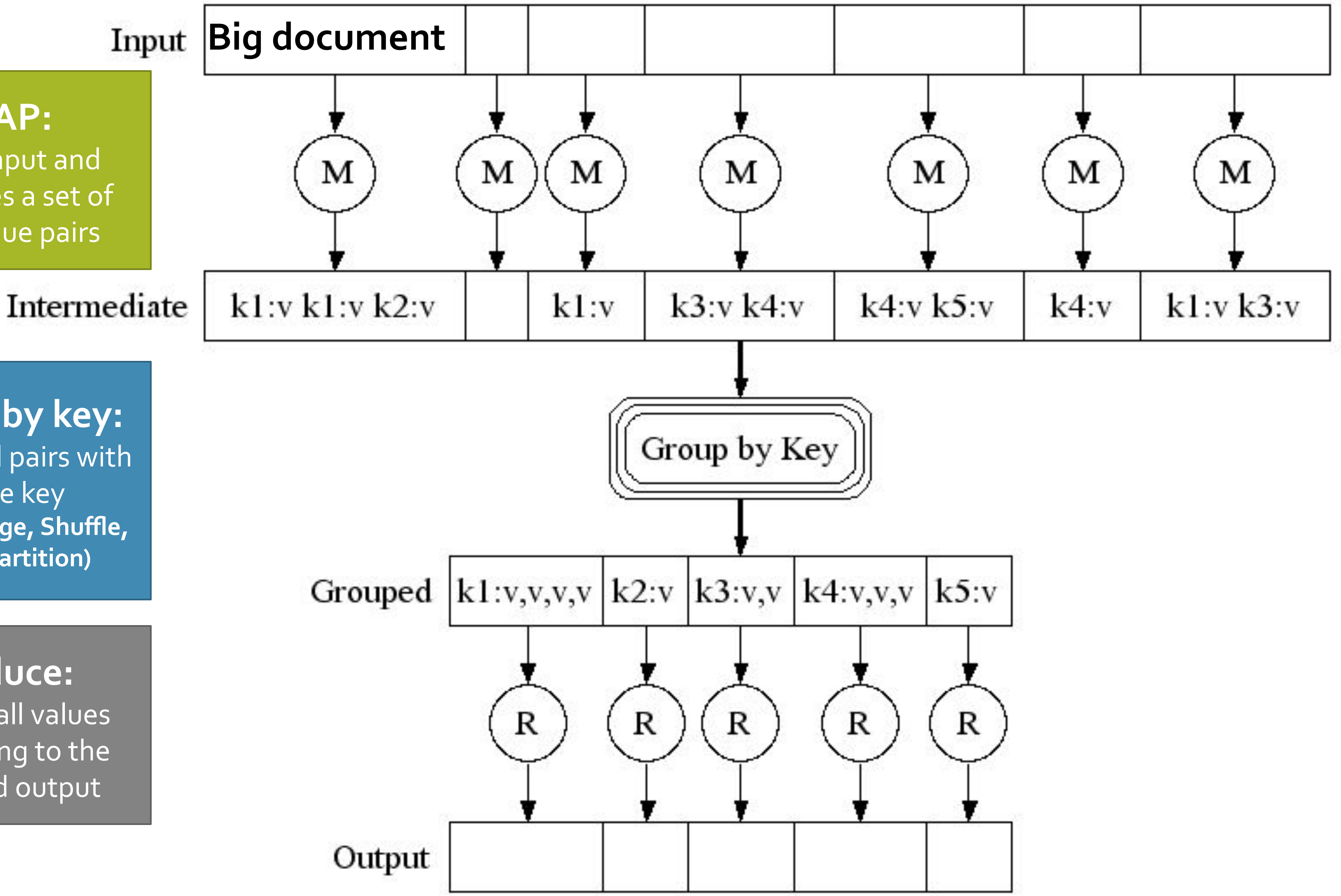
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Map-Reduce: A diagram

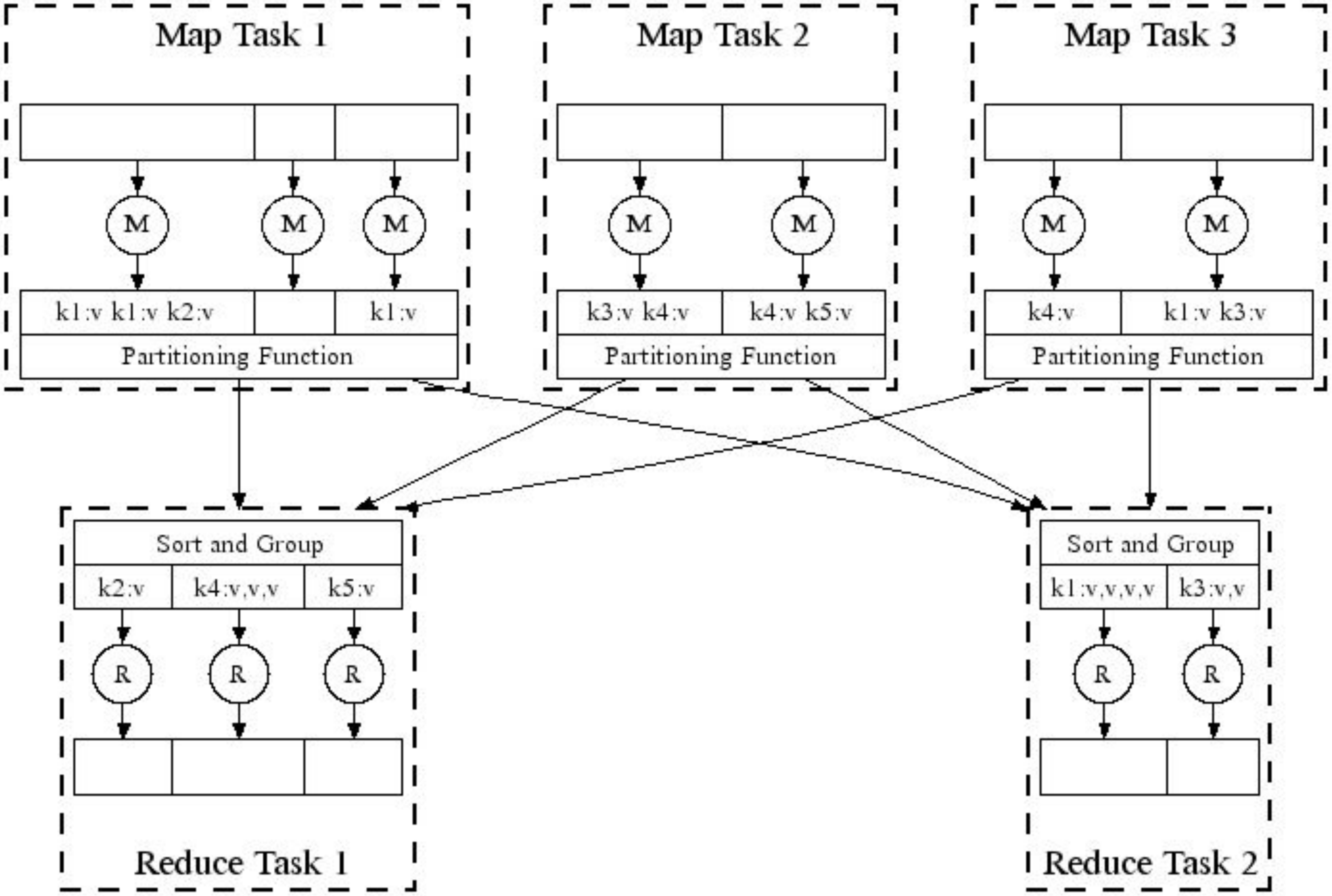
MAP:
Read input and produces a set of key-value pairs

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

Reduce:
Collect all values belonging to the key and output



Map–Reduce: In Parallel



All phases are distributed with many tasks doing the work

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

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

Dealing with Failures

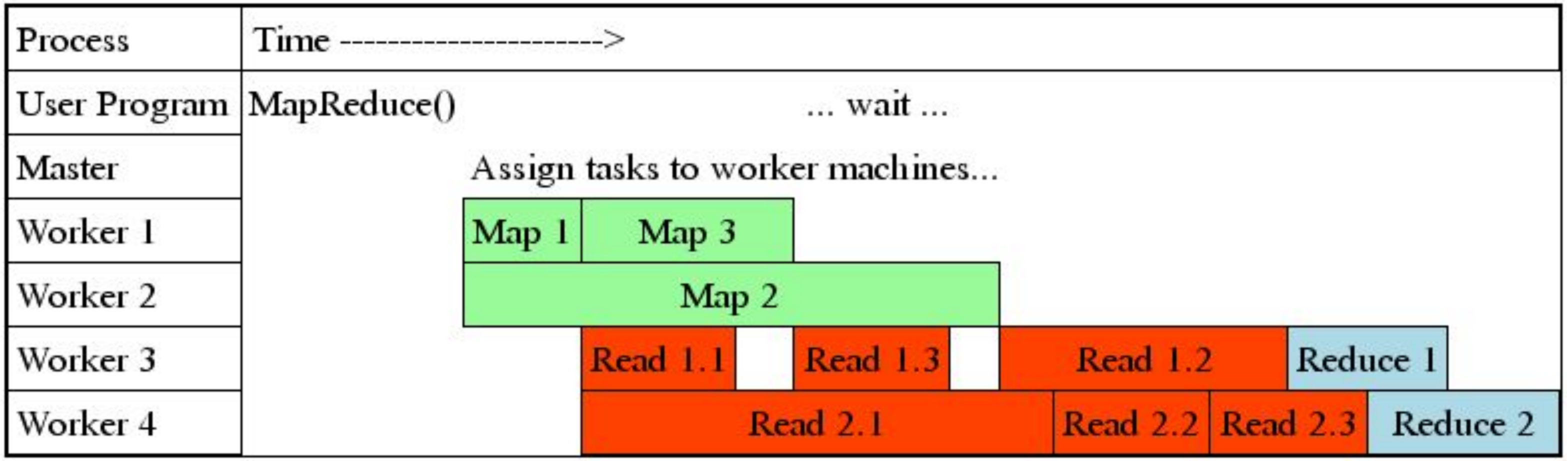
- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- Master failure
  - MapReduce task is aborted and client is notified

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

Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks >> machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing



Process | Time ---------------------->

User Program | MapReduce() ... wait ...

Master | Assign tasks to worker machines...

Worker 1 | Map 1 | Map 3

Worker 2 | Map 2

Worker 3 | Read 1.1 | Read 1.3 | Read 1.2 | Reduce 1

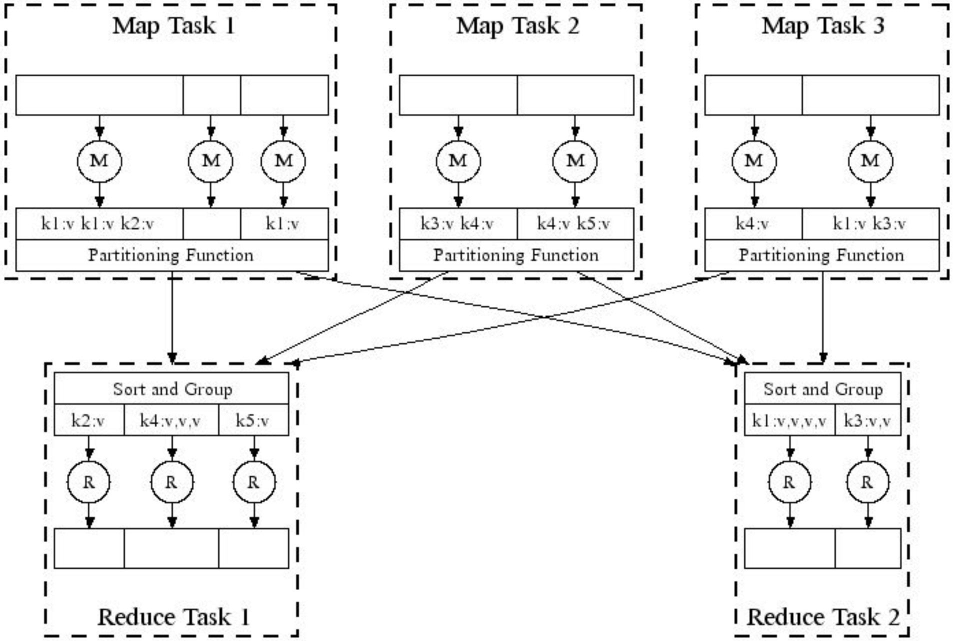
Worker 4 | Read 2.1 | Read 2.2 | Read 2.3 | Reduce 2

Refinement: 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

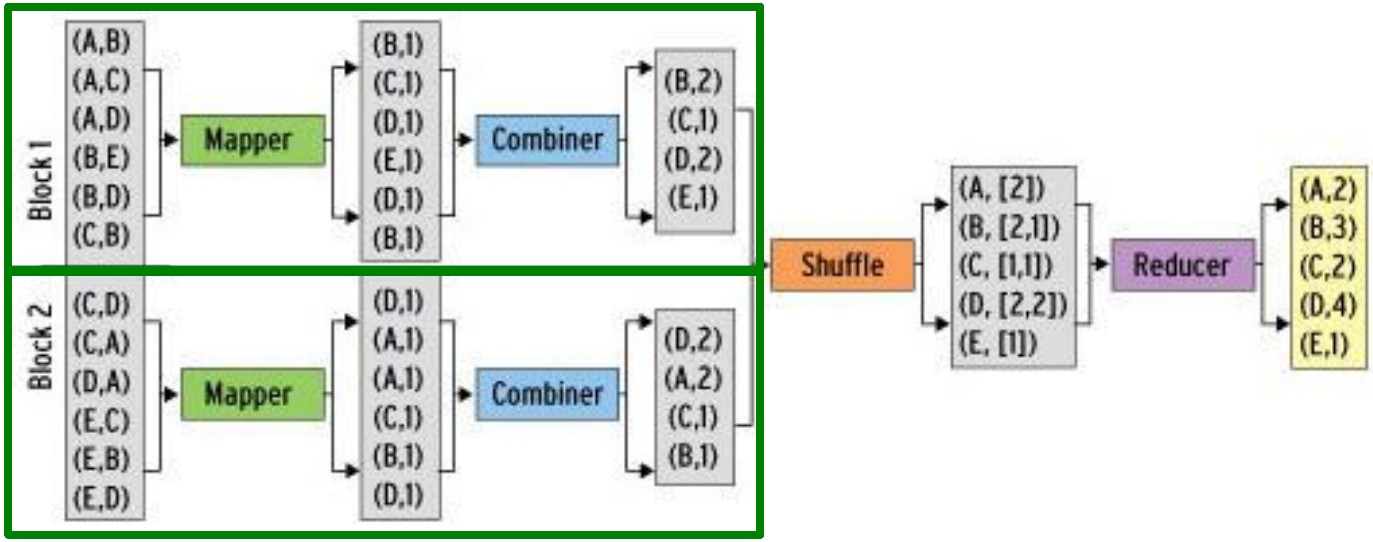
Refinement: Combiners

- Often a Map task will produce many pairs of the form (k,v_1), (k,v_2), ⋯ 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(v_1)) → v_2
  - Combiner is usually same as the reduce function



Refinement: Combiners

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



  - d!

Refinement: Combiners

- Combiner trick works only if reduce function is commutative and associative
  교환법칙 결합법칙
- Sum (o)
  - a + b = b + a, (a + b) + c = a + (b + c)
- Average (△)
  - avg(a,b) = avg(b,a), avg(avg(a,b), c) != avg(a, avg(b,c))
    결합 법칙 적용 불가
  - Combination of sum and count
- Median (x)

Refinement: Partition Function

- Want to control how keys get partitioned
  - The set of keys that go to a single reduce 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

Implementations

- Google MapReduce
  - Uses Google File System (GFS) for stable storage
  - Not available outside Google

- Hadoop
  - Open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://hadoop.apache.org/

Questions?