MapReduce

EE412: Foundation of Big Data Analytics

Announcements

- Todo reminder
  - Register to Classum
  - (Next week) Login to Haedong machine and change your password
    - Please check later an announcement at Classum
- Lecture videos
  - Will be recorded and uploaded from today
- Homeworks
  - Will post HW0 and HW1 next Thursday

apache spark

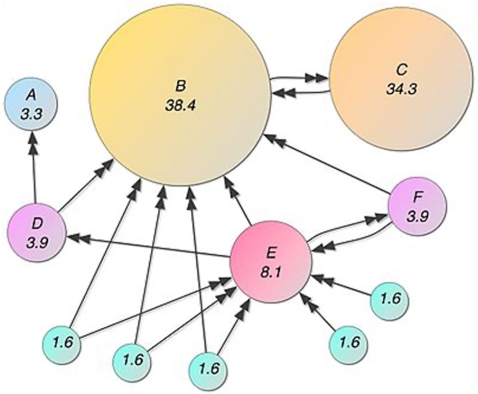
Te 10 assignment

Outline

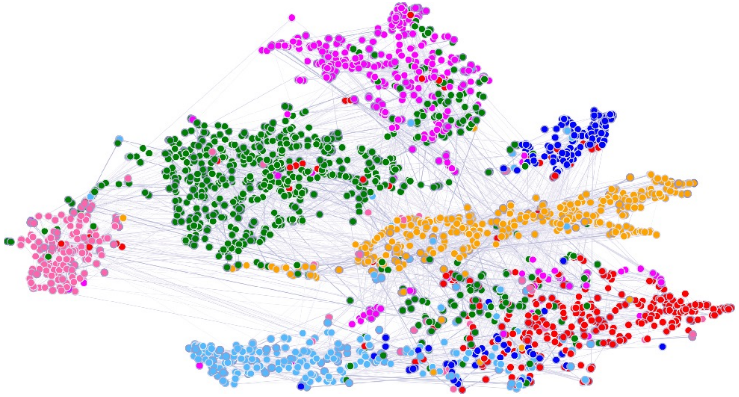
1. **<u>Distributed Computing for Data Mining</u>**
2. MapReduce: Basics
3. MapReduce: Details

Managing Large Amounts of Data Quickly

- Ranking Web pages by importance
  - Iterated matrix-vector multiplication where dimension is many billions
- Search friends in social networks
  - Graphs with hundreds of millions of nodes and many billions of edges



Source: Spatial



Source: Velickovic et al. (2018)

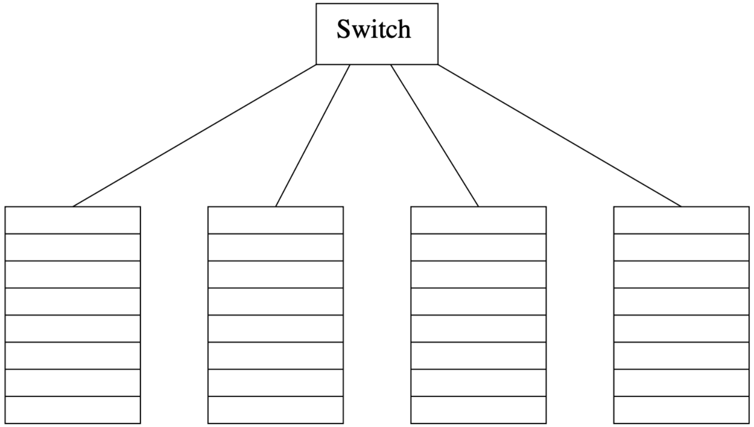
Large-Scale Computing

- Supercomputers are expensive and cannot scale infinitely
- Instead, we have a large **collection of commodity hardware**
  - Connected by Ethernet cables or inexpensive switches

Source: UMBC

Physical Organization of Nodes

- **Parallel-computing architecture**
  - Compute nodes are stored on racks (perhaps 8 – 64 on a rack)
  - The nodes on a single rack are typically connected by gigabit Ethernet
  - There can be many racks connected by a switch



Source: Textbook

Challenges

- How do you distribute computation?
- How can we make it easy to write distributed programs?
- Machines fail:
  - One server may stay up 3 years (≈ 1,000 days)
  - If you have 1,000 servers, expect to lose 1/day
  - With 1M machines, 1,000 machines fail every day!

Idea and Solution

- **Issue:** Copying data over a network takes time
- **Idea:** *Bring computation to data* result 만 공유하고, raw data exchange X
  모든 노드에 모든 data 없는 (subset이 존재)
- **Hadoop/Spark** address these problems
  - **Storage infrastructure (file system):** GFS and HDFS
  - **Programming model:** MapReduce and Spark

philosophy

not the real software

real sw

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently? → 영속화 어케 하노?.
    - Loss of single node (e.g., disk crashes)
    - Loss of an entire rack (e.g., network fails)
- **Answer: Distributed File System (DFS)**
- **Typical usage pattern:**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place → 수정은 rarely
    는 dependent 하므로 비싸다.
  - Reads and appends are common

읽기, 쓰기는 common

쓰기는 기존 데이터에) dependent X

Distributed File System (DFS)

- **Chunk servers**
  - Files are divided into chunks, which are typically 16 – 64MB
  - Chunks are **replicated** (usually 2x or 3x) at different nodes
    - Try to keep replicas in different racks 다른 rack에 replica
  - Chunk servers also serve as compute servers // (result 계산을 위함)
- **Master node** (or a name node) manage all info (파일이 어디에 있나?)
  - Another small file storing metadata about where files are stored
  - Master node itself is replicated
    - File system directory knows where to find the master node copies

Outline

1. Distributed Computing for Data Mining
2. **MapReduce: Basics**
3. MapReduce: Details

MapReduce

- **MapReduce** is a style of programming designed for:
  1. Easy parallel programming
  2. Invisible management of hardware and software failures
  3. Easy management of very-large-scale data
- It has several implementations, including Hadoop, Spark, Flink, and the original Google implementation just called “MapReduce”

Example: Word Counting

each machine have partial part

각 머신에서 계산을 해서 결과를

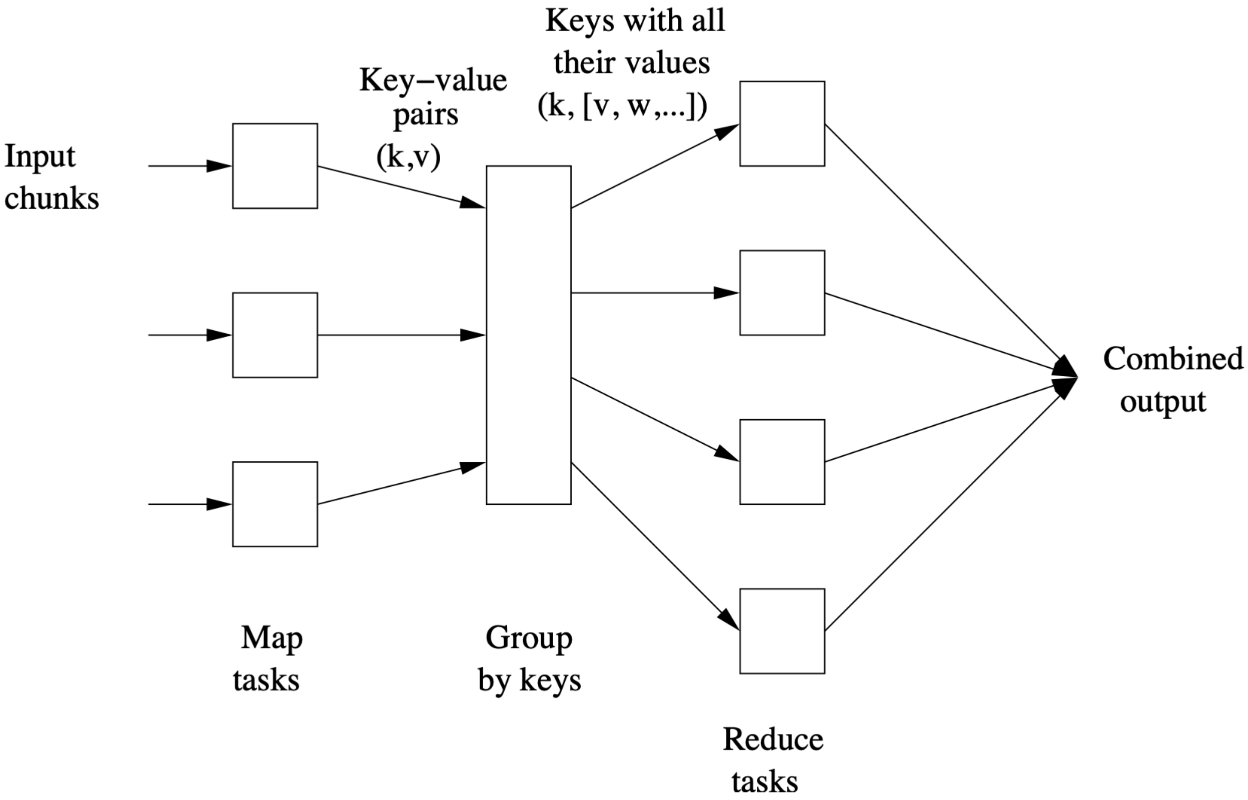
- Suppose we have a huge text document
- Count the number of times each distinct word appears in the file
  - E.g., (ball, 30), (basket, 10), (great, 5), and so on
- **Many applications:**
  - Analyze web server logs to find popular URLs
  - Statistical machine translation
    - Need to count the frequency of all 5-word sequences in documents

MapReduce: Components

ex) document 그냥 line 등

- **Map:**
  - Apply a user-written *Map function* to each input element
  - The output of the Map function is a set of key-value pairs //
- **Group by key:** Sort and shuffle
  - System sorts all the key-value pairs by key
  - Outputs key-(list of values) pairs
- **Reduce:**
  - User-written *Reduce function* is applied to each key-(list of values)

MapReduce Pattern



Source: Textbook

Map

- **Input**
  - Chunks consisting of elements, which are key-value pairs
    - To allow composition of several MapReduce processes
    - Keys of the input (i.e., initial) elements can be ignored
- **Output**
  - Zero or more key-value pairs

Group by Key

각 machine 에 다 저장하지 못할 수도 있다.

- **Input**
  - Key-value pairs from Map tasks
- **Output**
  - The master controller knows the number of reducer tasks (r)
  - Applies a hash function to the keys, producing bucket numbers 0 to $r - 1$
  - The key value pairs are put into r local files
    - Each local file is destined for a specific Reduce task
  - The master controller merges the files and returns $(k, [v_1, v_2, \ldots, v_n])$

key에 대해 hashing을 해서 목적지를 설정

Reduce

- **Input**
  - One or more keys and their associated value lists
    - One Reducer can take several keys
- **Output**
  - The Reduce function is applied to each key-(value list) pair
  - The outputs from all the Reduce tasks are merged into a single file

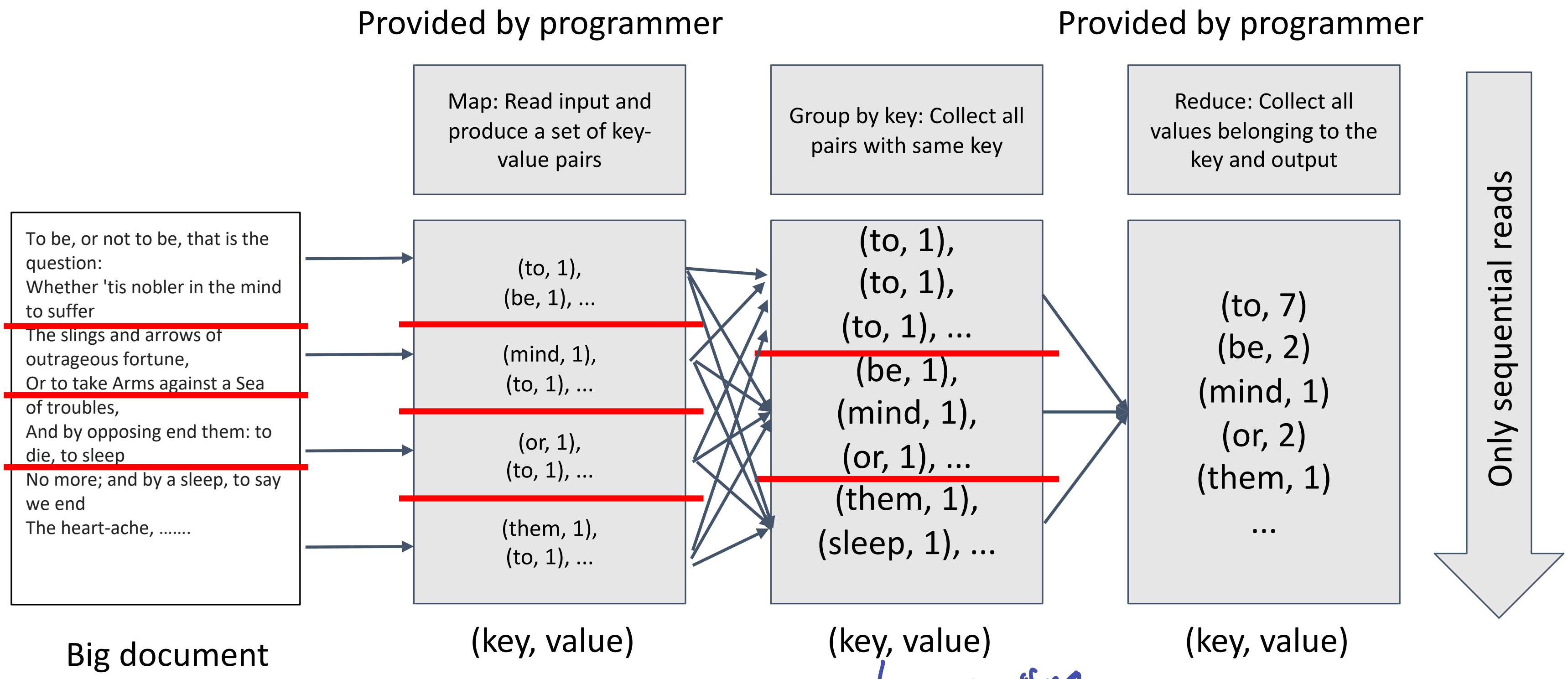
Map and Reduce for Word Counting

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

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

reduce뒤에 또 map을 또붙이고
붙이는거 가능

MapReduce for Word Counting



↳ sorting
key 별로 bitwise sorting을 하면

그래도 빠르게 진행할거고, 분산되지 않을까?

Combiners

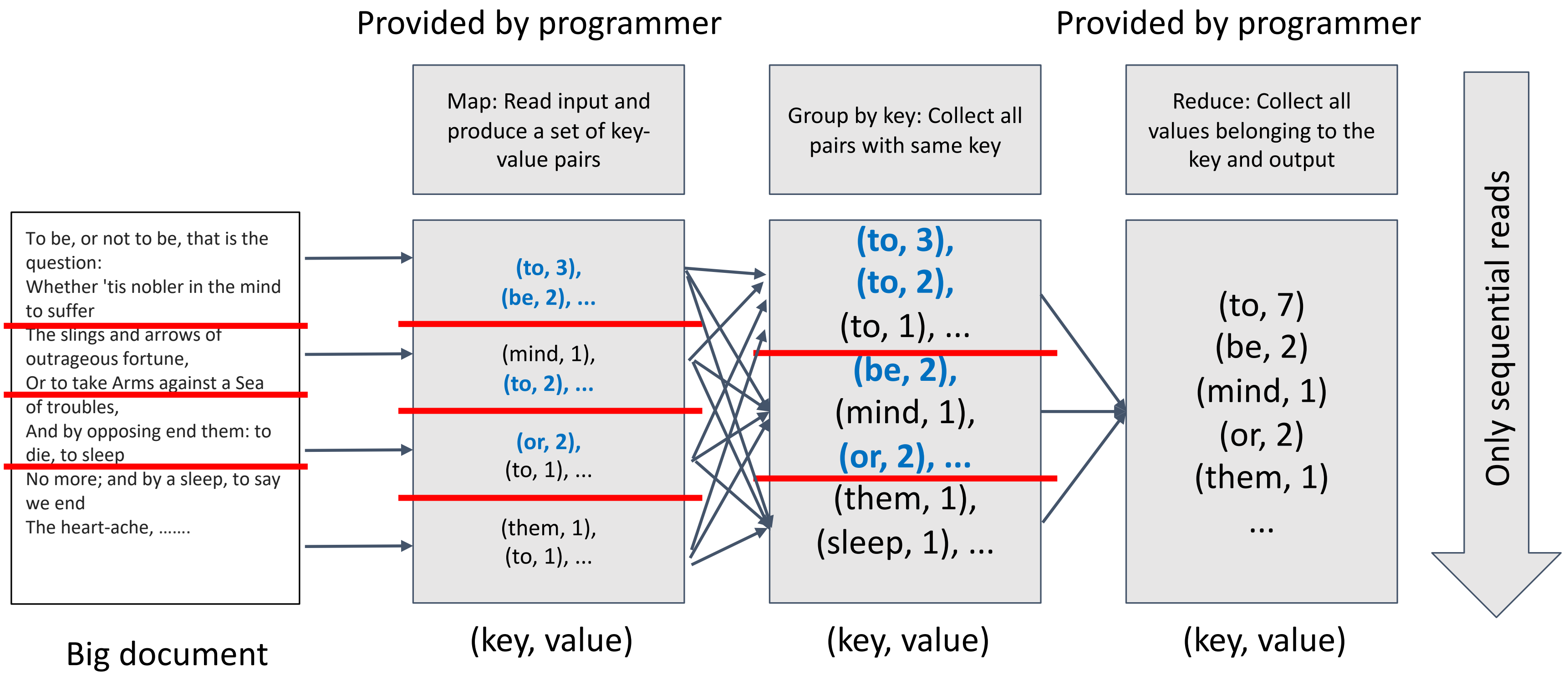
교환 결합

- If the Reduce function is commutative and associative, values can be combined in any order for the same result
  - E.g., word counting, matrix-vector multiplication
  - **Commutative:** $a + b = b + a$
  - **Associative:** $(a + b) + c = a + (b + c)$

바로 연산을 map 단계에서 하면 됨

- In this case, the Reducer's work can be pushed to the Map tasks
  - E.g., instead of producing $(w, 1)$, $(w, 1)$, …, produce (w, m)
- It reduces the amount of data shuffling for the group by key step

Combiner for Word Counting



Pop Quiz

- Let's find common friends of friends on a social network
- What are the map and reduce functions?

Input

A -> B,C,D
B -> A,C,D
C -> A,B
D -> A,B

→ Map tasks → Group by keys → Reduce tasks →

Output

A,B -> C,D
A,C -> B
A,D -> B
B,C -> A
B,D -> A

(A,B) -> C

(A,D) -> C

(A,B) -> D

(A,C) -> B

(A,C) -> D

(A,D) -> B

(B,A) -> C

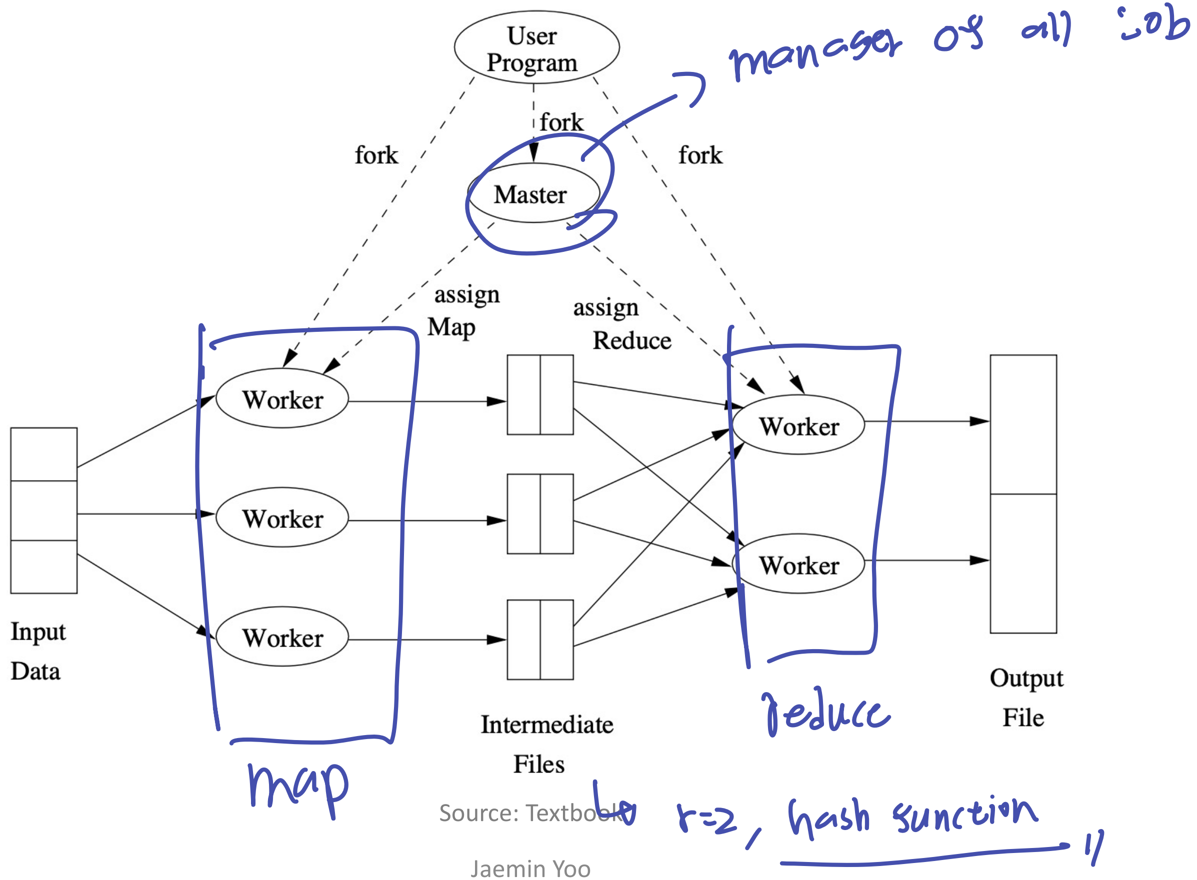
(B,A) → D

(C,A) → B

Outline

1. Distributed Computing for Data Mining
2. MapReduce: Basics
3. **MapReduce: Details**

MapReduce Program Execution



Source: Textbook

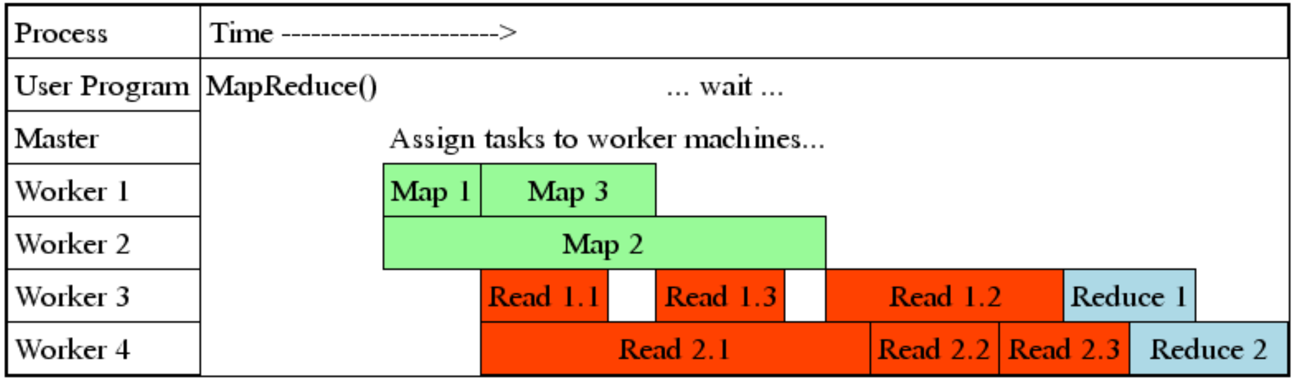
MapReduce Program Execution

- User program forks a Master controller and Worker processes
  - A worker handles either Map or Reduce tasks, but not both
- **The Master controller**
  - Schedules Map and Reduce tasks and assigns them to Worker processes
    - One Map task per chunk is reasonable
    - Reduce tasks should be fewer than Map tasks
      - Otherwise, too many intermediate files per Map task
  - Keeps track of task status (idle, executing at a Worker, or completed)
  - Worker process reports to Master when it finishes a task

MapReduce Program Execution

- **Map task**
  - Creates a file for each Reduce task on the Map Worker's local disk
  - The Master is informed of the files and the destination Reduce tasks
- **Reduce task**
  - When assigned to a Worker, it is given all the files that form its input
  - Executes code written by the user and writes its output to a file

MapReduce Pipelining



Source: Textbook Slides

Coping with Node Failures

need to rerun

to algorithm way

- **Master failure**
  - Entire MapReduce job must be restarted
- **Map worker failure**
  - Detected by Master because it periodically pings Worker processes
  - ?!. All Map tasks assigned this Worker have to be redone, even if completed
    - Output for Reduce tasks reside in compute node and are unavailable
  - Master informs each Reduce task that the location of its input has changed
- **Reduce worker failure**
  - ?. Only in-progress tasks are reset to idle and the Reduce task is restarted

Number of Reduce Tasks

- Parallelism can be maximized by using one Reduce task per key
- However, this plan is not usually the best
  - Overhead associated with each task created
  - Often there are far more keys than there are compute nodes
- Significant variation in the # of values for different keys
  - Reduce tasks will exhibit skew in processing time
  - **Rule of thumb:** # of nodes < # of Reduce tasks < # of keys !

진짜 정해지는건 어떻게 하냐로 정해짐

Algorithms using MapReduce

update를 위한 조작은 아니고

종종히 부적절

- MapReduce is not a solution to every problem
  - DFS only makes sense when files are very large and rarely updated in place
  - MapReduce is not suitable for frequent database changes
    - E.g., managing online retails sales (say by Amazon)
- **Original purpose of Google:**

  목적

  - To execute very large matrix-vector multiplications for PageRank
  - Matrix-vector and matrix-matrix calculations fit nicely into MapReduce
  - Relational algebra (i.e., SQL) can also use MapReduce effectively

Matrix-Vector Multiplication

- Suppose we have an $n \times n$ matrix M and a vector $\mathbf{v}$ of length n
- Matrix-vector product is vector $\mathbf{x}$ of length n whose i-th element is:

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

- If $n = 100$, we do not need DFS or MapReduce
- But in large search engines, e.g., Google, n can be tens of billions

Matrix-Vector Multiplication

- Assume v fits in main memory:

Map function
Input: m_{ij}
Output: $(i, m_{ij}v_j)$

Reduce function
Input: $(i, [mi_1v_1, mi_2v_2, \dots\ m_{in}v_n])$
Output: $(i, \sum_j m_{ij}v_j)$

M $\quad$ v

1	2
1	3

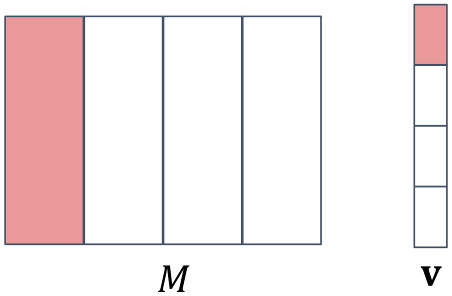
1
2

(1, 1*1 = 1)
(1, 2*2 = 4)
(2, 1*1 = 1)
(2, 3*2 = 6)

(1, [1, 4])
(2, [1, 6])

(1, 5)
(2, 7)

Matrix-Vector Multiplication



- What if the vector $\mathbf{v}$ cannot fit in main memory?
  - Divide M into vertical stripes and divide $\mathbf{v}$ into horizontal stripes
    - The width of vertical strips = the height of horizontal stripes
    - Use enough stripes so that one stripe of $\mathbf{v}$ fits in main memory
  - Each Map task is assigned a chunk from one of M's stripes and the corresponding stripe in $\mathbf{v}$
  - The Map and Reduce tasks then run exactly the same as before

Summary

1. Distributed Computing for Data Mining
   - Distributed File System
2. MapReduce: Basics
   - Map, group by key, reduce, and combiners
   - Example: Word counting
3. MapReduce: Details
   - Coping with node failure
   - Algorithms using MapReduce