빅데이터 분석 및 응용

L01: MapReduce & DFS

Summer 2020

Kookmin University

In this lecture

- Motivation of MapReduce
- Distributed File System

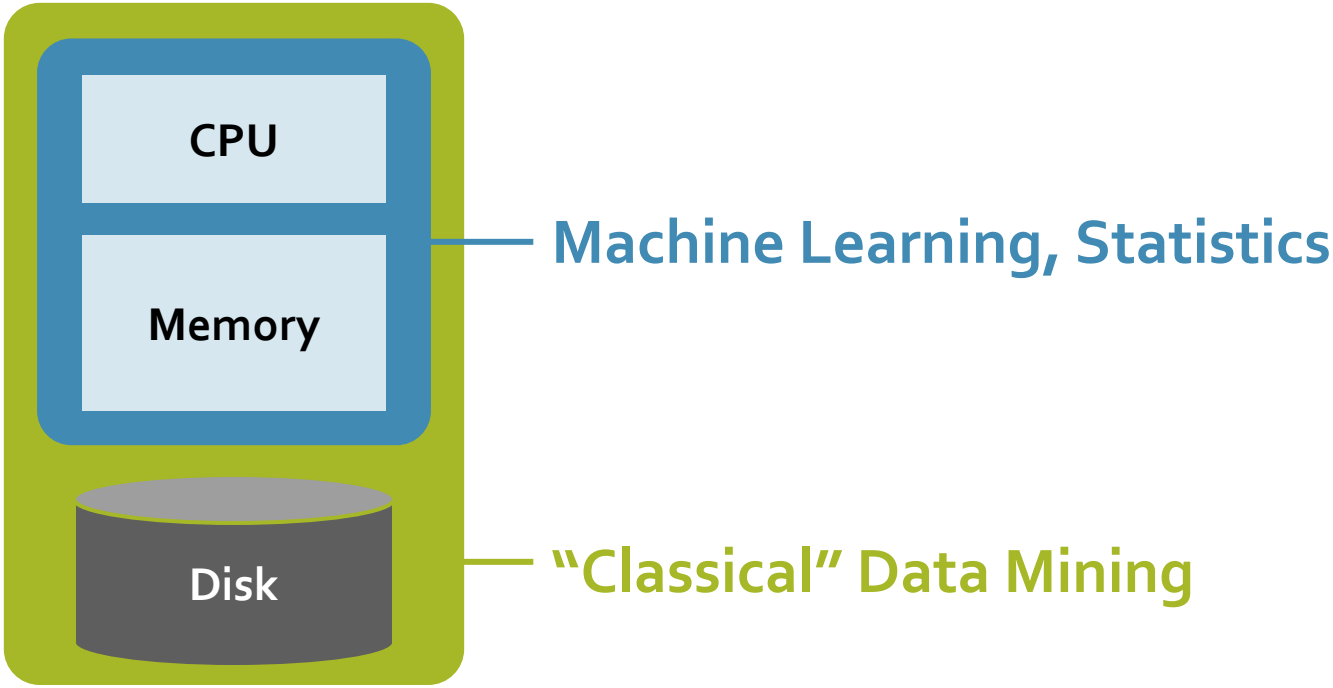
MapReduce

- **Large scale computing** for **data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard

- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

조작모델

Single Node Architecture

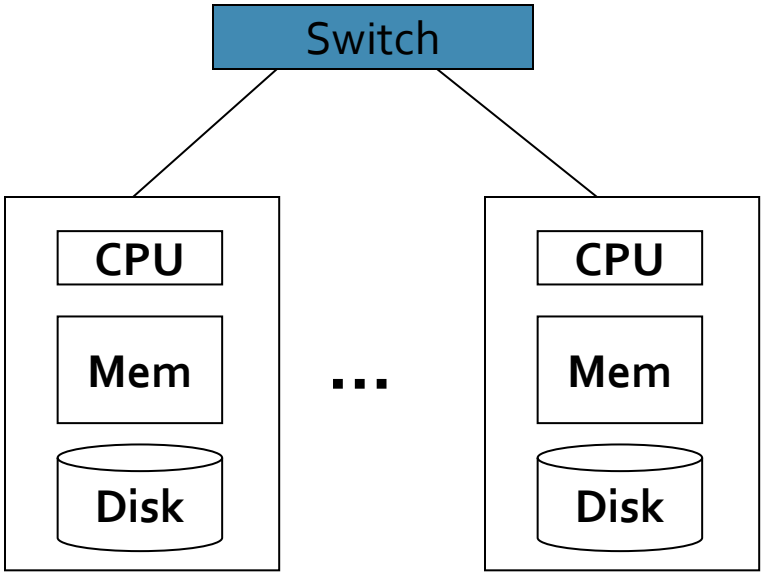


Motivation: Google Example

- 20+ billion web pages
- Average size of webpage = 20KB
- 20 billion * 20KB = 400TB
- Disk read bandwidth = 50MB/sec
- Time to read = 8 million seconds = 92+ days
- Even longer to do something useful with the data
- Today, a standard architecture for such problems is emerging:
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture

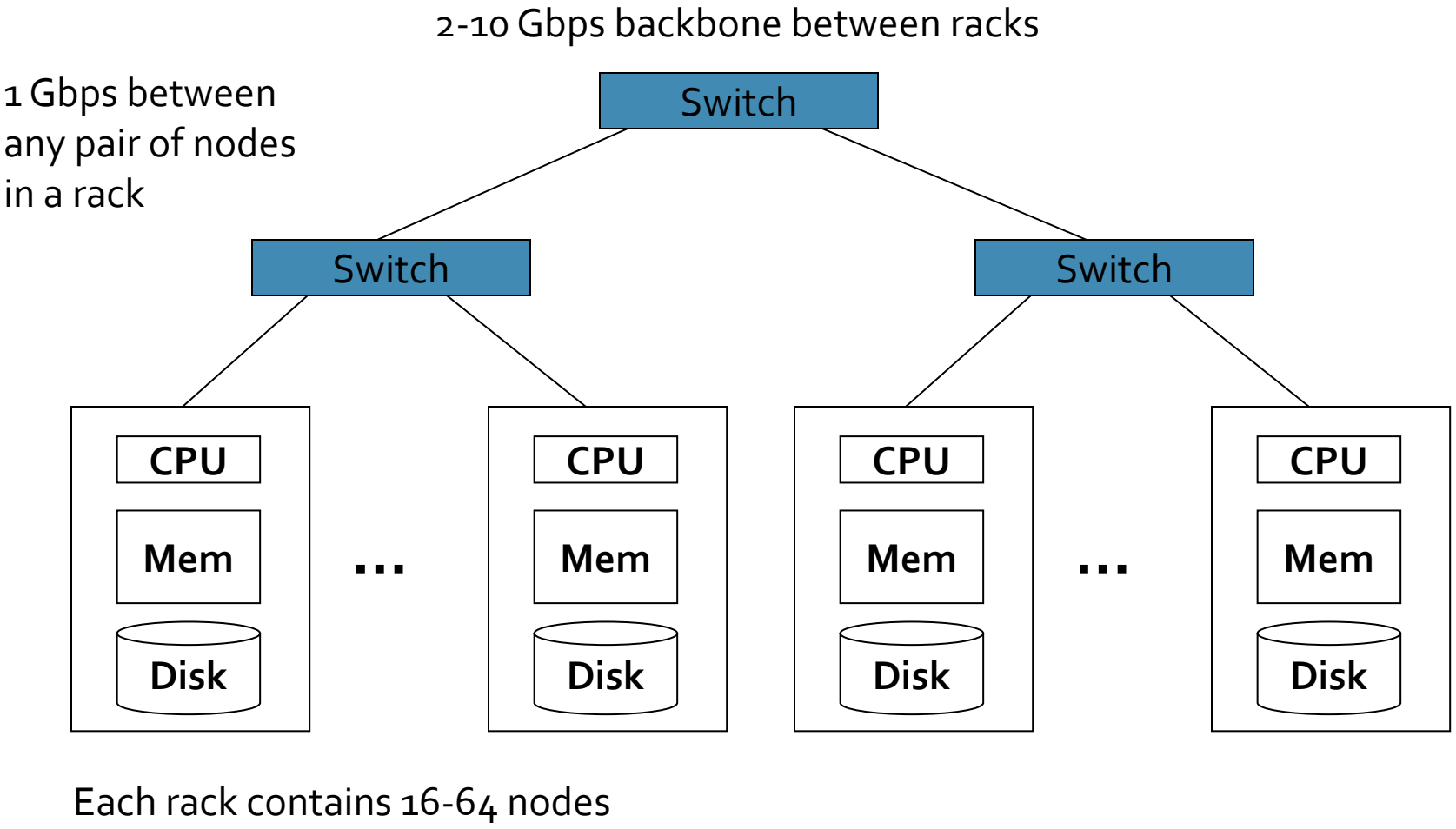
1 Gbps between
any pair of nodes
in a rack



Each rack contains 16-64 nodes

In 2011 it was guesstimated that Google had 1M machines, http://bit.ly/Shh0RO

Cluster Architecture



In 2011 it was guesstimated that Google had 1M machines, http://bit.ly/Shh0RO

Cluster Computing Challenges (1)

- Node failures
  - A single server may stay up for 3 years (1000 days)
  - If you have 1000 servers, expect to loose 1/day
  - People estimated Google has ~1M machines in 2011
    - 1000 machine fail every day!
- How to store data persistently and keep it available if nodes can fail?
- How to deal with node failures during a long-running computation?

Cluster Computing Challenges (2)

- Network bottleneck
  - Network bandwidth = 1 Gbps
  - Moving 10TB takes approximately 1 day
- Distributed programming is hard!
  - Need a simple model that hides most of the complexity

MapReduce

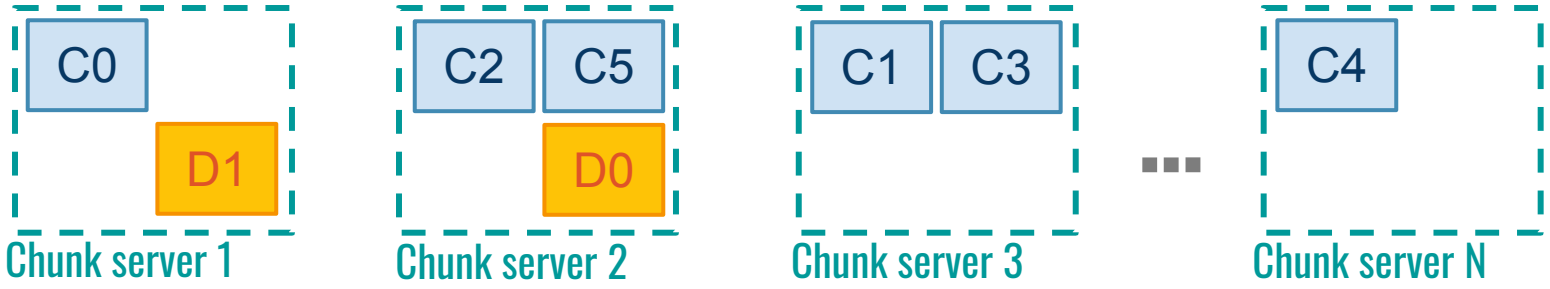
- MapReduce addresses the challenges of cluster computing
  - Store data redundantly on multiple nodes for persistence and availability
  - Move computation close to data to minimize data movement
  - Simple programming model to hide complexity of all this magic

Redundant Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

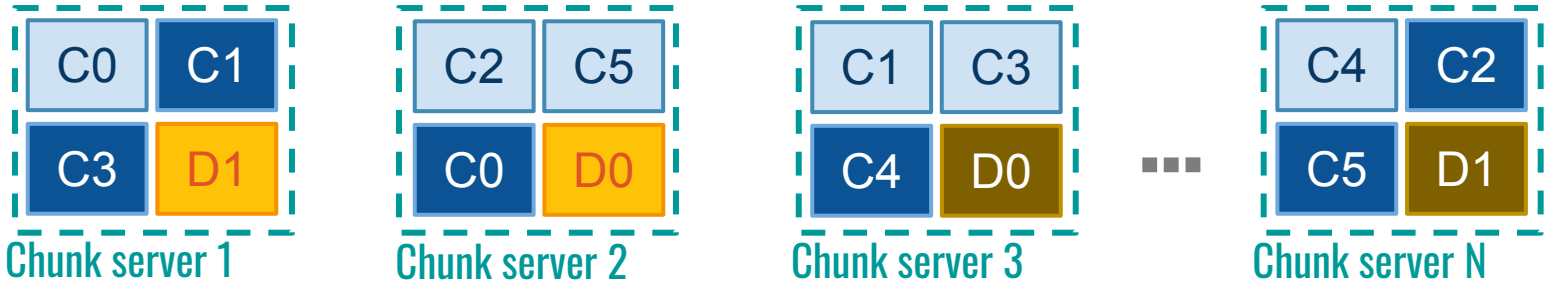
Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines



Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  - Ensures persistence and availability



Chunk servers also serve as compute servers

Bring computation directly to the data!

Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Questions?