Lecture 1: Introduction

Course Overview

- Lectures
- Papers
- Exams
- Labs

https://pdos.csail.mit.edu/6.824/

Programing Labs

- Lab 1: MapReduce
- Lab 2: Raft
  - In order to allow any system to be made fault tolerant by replicating it
- Lab 3: Key/value server
  - Use Raft implementation to build fault tolerant key/value server
- Lab 4: Sharded K/V server
  - Shard - refer to splitting-up/partitioning the data among multiple server to get parallel speed
  - Take K/V server and clone it into a number of independent groups and will split the data in K/V storage system —> parallel speed-up by running multiple replicated group in parallel
  - Moving chunk of data between different servers

Infrastructure for Application

Infrastructure

- Storage
- Communication
- Computation

Target: Build **an interface** that when we look to an application , it will be simple file system (**Abstraction**)

Topics

Implementation

- RPC - mark the fact that we're communicating over an unreliable network,
- Thread - allow to harness multi-core computers —> Concurrency control

Scalability

Performance

- Scalability/Scalable speed up

Failure

Fault Tolerance

- Availability
- Recoverability (e.x: save the lastest state and recover)

Solutions:

- **Non-volatile storage (hard drivers, flash, solid state)**
  - Meant was *moving disk arm* and waiting for *disk platter to rotate* (slow)
  - Store check point or a log of the system's state —> Back up /Repairs.
  - Tend to be expensive to update
- **Replication**
  - Have more than 1 copy of data floating around
  - Have lots of different versions

Availability

Consistency

```
Put(K,V)

Get(K) -> V
```

Strong consistency:

- be guaranteed to see the most recent write
- very expensive to implement

Week consistency:

(prefer more in real world)

- Avoid communication as much as possible
  - Data rack usually put in different city - it take *ms* - *s* to communicate

Map Reduce

Google - publish 2004 - when

- have to build index of the web + sort data
- want to have framework for non distributed engineer

PageRank —> run multiple MapReduce job

MapReduce —> Run in multiple workers , which have master to coordinate

- Master: Run this Map func in Input file
- Worker: Read Input file with Map func

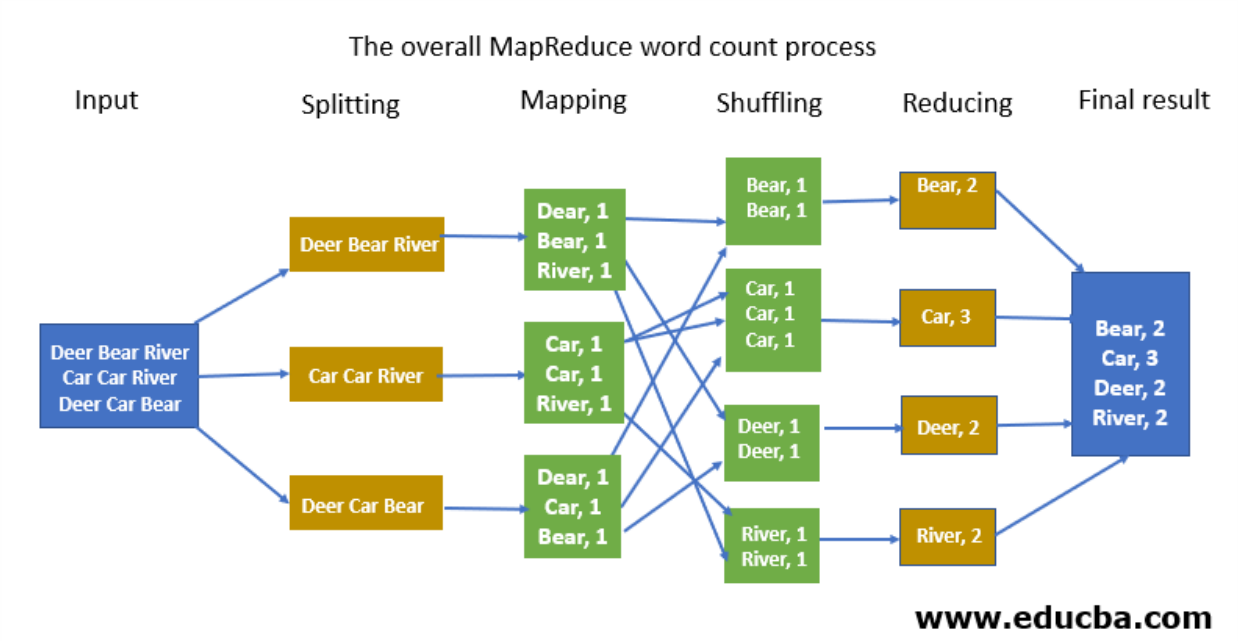
Map(k,v)

```
    split k into words

    for word in words
        //write data in map workders local disk to accumulating all the
keys and values produced by the maps run on that workers
        // E.x: Dear, 1  Bear, 1    River, 1
        emit(word, 1)
```

Reduce(k,v)

```
    // have to talk with all workers "I want run reduce of key 'A', please
see all key 'A' and send them to network to me (Shuffling)
    emit(len(v))
```



(The arrow '—>' in that picture, refer to MapReduce workers process have to go off and talk across the network to correct GFS servers that store the input + fetch over the network to the MapReduce worker machines)

—> The most bottleneck in MapReduce is **Network throughput**

Because we want flexibility to be able to read any piece of input on any worker server

—> we need kind of network file system to store input data

—> Google File System - GFS: cluster file system:

HDFS

- Automatically splits up big file across lots of servers in 64MB chunks
  - VD: 1TB web crawled —> GFS —> split into chunks and distribute evenly over GFS servers
- By that way, Maper will read in parallel

- 2004, run Map func in same machine store data.
- 2020, Modern data center today actually have many root switches, and each rack switch has a connection to each replica root —> split up traffic among the root

—> No longer run Map func in the same machine that store data