Contents

[1.Introduction 2](#_Toc59882495)

[1.1 Distributed Systems 3](#_Toc59882496)

[2. MapReduce 3](#_Toc59882497)

[2.1 Scheduler 4](#_Toc59882498)

[2.2 Fault Tolerance: 6](#_Toc59882499)

[2.2.1 Server Failure: 6](#_Toc59882500)

[2.2.2 RM Failure: 7](#_Toc59882501)

# 1.Introduction

Amazon web services provide:

* Ec2 : Elastic compute cloud (upload and run arbitrary OS images, pay per CPU hour used )
* S3 : Simple Storage Service (Pay per GB-month used)
* EVS : Elastic Block Storage

Cloud = lot of storage + compute cycles nearby

A single-site clouod consists of

* Computer nodes (grouped into racks)
* Switches, connecting the racks
* A network topology
* Storage (backend nodes) connecting to the network
* Front-end for submitting jobs and receiving client-requests
* Software services

Features of Clouds:

* Massive Scale
* On demand access
* Data intensive nature
* New Cloud Programming Paradigms: MapReduce/ Hadoop, NoSQL/ Cassandra/ MongoDB and many others.

**Clouds are Distributed Systems**

## 

## 1.1 Distributed Systems

A distributed system is a collection of entities, each of which is **autonomous**, **programmable**, **asynchronous** **and failure prone**, and which communicate through an unreliable communication medium.

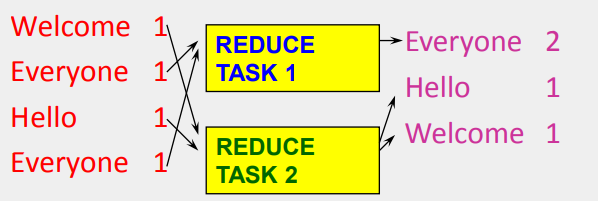
**Challenges :**

* Failures: no longer the exception, but rather a norm
* Scalability: 1000s of machines, Terabytes of data
* Asynchrony: clock skew and clock drift
* Concurrency: 1000s of machines interacting with each other accessing the same data

# 2. MapReduce

**Maps** : Process individual records to generate intermediate key/ Value pairs. Processes sequentially and independently and hence the process can be highly parallelized.

**Reduce :** Process and merge all intermediate values associate per key. Parallely processes and merges all intermediate values by partitioning keys



Each key is assigned to Reduce no = Hash(key) % number of reduce servers

**Example:** Distributed Grep:

* Input : Text files
* Output : Lines that match pattern
* Map : Emits a line if it matches the supplied pattern
* Reduce : copies the intermediate data to output

## 2.1 Scheduler

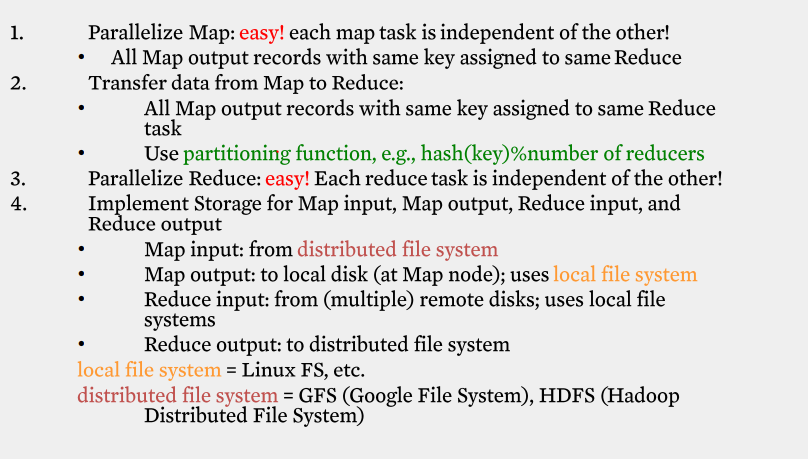
Externally for the user:

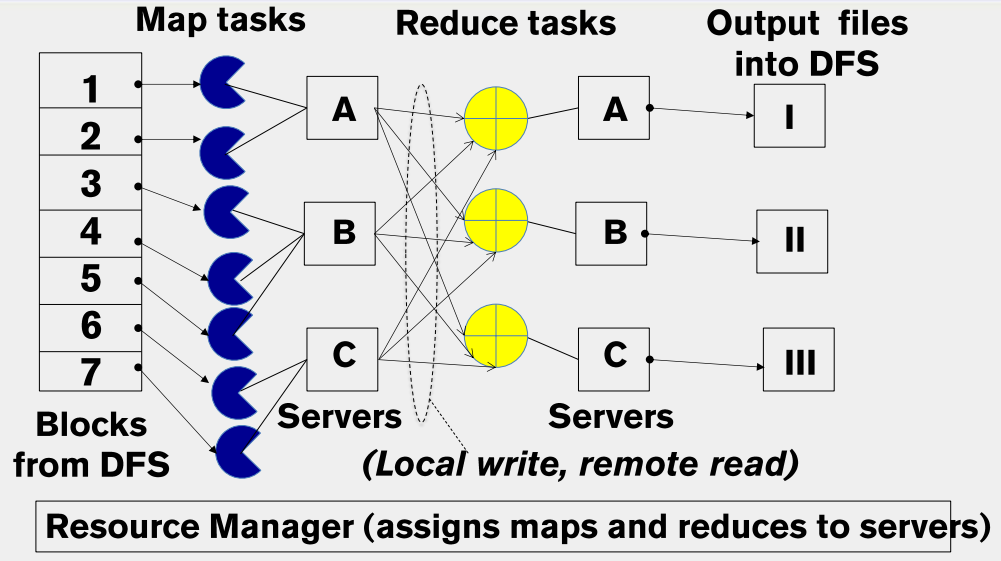
* Write a Map and a Reduce program (short)
* Submit Job; wait for the result
* Need to know nothing about parallel and Distributed programming

Internally for the Paradigm and the Scheduler:

* Parallelize Map
* Transfer data from map to Reduce
* Parallelize Reduce
* Implement storage for Map input, Map output, Reduce input and Reduce output

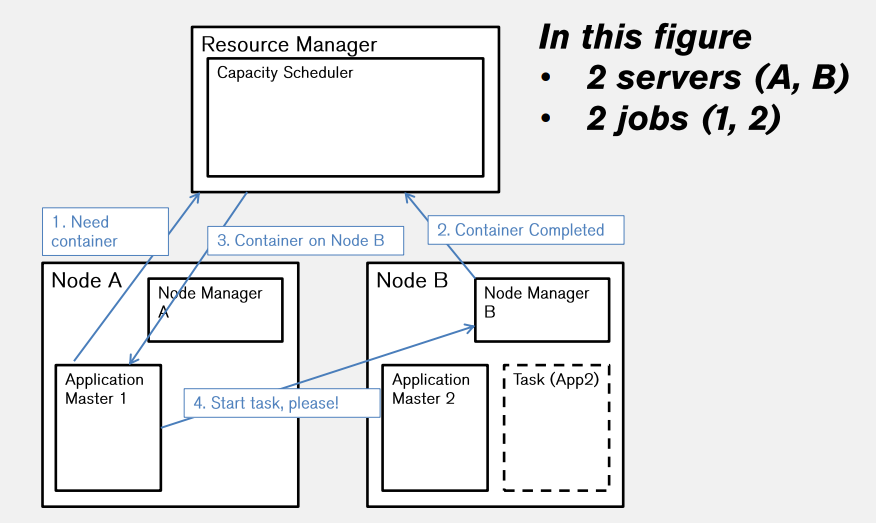
For the cloud:





**The YARN Scheduler :**

* YARN = Yet Another Resource Manager
* Treats each server as collection of containers
  + Container = some CPU + some Memory
* Has 3 Main components
  + Global Resource Manager (RM)
    - Scheduling
  + Per-Server Node Manager (NM)
  + Per-Application (job) Application Master (AM)



## 2.2 Fault Tolerance:

### 2.2.1 Server Failure:

* NM heartbeats to RM
  + If server fails, RM lets all affected AMs know, and AMs take action
* NM keeps track of each task running at its server
  + If task fails while in-progress, mark the task as idle and restart it
* AN heartbeats to RM
  + On failure, RM restarts AM, which then syncs up with its running tasks

### 2.2.2 RM Failure:

* Use old checkpoints and bring up secondary RM

### 2.2.3 Slow Servers

* The slowest machine also slows down the entire job
* Keep track of progress of each task (% done )
* Perform backup (replicated) execution of straggler task : task considered done when first replica complete. Called **speculative execution**