Two Categories:
Private Cloud & Public Cloud
private cloud: accessible only to company employees
public cloud: provide service to any paying customer
- Amazon s3(simple storage service): Store arbitrary datasets, pay GBs/Month;
- Amazon EC2(Elastic computing cloud): Upload & run arbitrary OS images, pay CPUs/h
- Google App Engine/Comput Engine: Develop applications within their app engine framework, upload data that will be imported into their format and run

Cloud = Lots of storage + compute cycles nearby

Simple Cloud Topology:

```
                 Core Switch
                 /  …… | ……  \
     /      Rack Switch  RS      RS
Rack  -      / …… | ……\
     \     Servers Servers..
```

Today's Cloud

Four Features New in Today's Cloud
- I - Massive Scale
- II - On-demand Access: Pay-as-you-go, no upfront commitment
  Anyone can access it
- III - Data Intensive Nature: MBs become TBs, PBs and XBs
  Daily logs, forensics, web data ...
  Human has data numbness: Wiki compress is Only 10 GB
- IV - New Cloud Programming Paradigms: Mapreduce/Hadoop, NoSQL/Cassandra/MangoDB and many others
  High in accessibility and ease of programmability
  Lots of open-source

II - On-demand Access: *aaS Classification
Renting a cab vs. (previously) renting a car, or buying one. Ex.:
• AWS Elastic Compute Cloud (EC2) (renting a cab):
  a few cents to a few $ per CPU hour
• AWS Simple Storage Service (S3) (renting a car or having one already):
  a few cents to a few $ per GB-month

HaaS: Hardware as a Service
- You get access to barebones hardware machines, do whatever you want with them, ex: your own cluster
- Not always a good idea because of security risks

IaaS: Infrastructure as a Service
- You get access to flexible computing and storage infrastructure. **Virtualization** is one way of achieving this (what's another way, e.g., using Linux). ***Often said to subsume HaaS.***
- Ex: Amazon Web Services (AWS: EC2 and S3), Eucalyptus, Rightscale, Microsoft Azure

PaaS: Platform as a Service
- You get access to flexible computing and storage infrastructure, coupled with a software platform (often tightly)
- Ex: Google's AppEngine (Python, Java, Go)

SaaS: Software as a Service
- You get access to software services, when you need them. Often said to subsume SOA (Service Oriented Architectures).
- Ex: Google docs, MS Office on demand

III - Data-intensive Computing
Computation-Intensive Computing
- Example areas: MPI-based, high- performance computing, grids
- Typically run on supercomputers (e.g., NCSA Blue Waters)

Data-Intensive
- Typically store data at datacenters
- Use compute nodes nearby
- Compute nodes run computation services

In data-intensive computing, the focus shifts from computation to the data
- CPU utilization no longer the most important resource metric, instead I/O is (disk and/or network)

IV - New Cloud Programming Paradigms
Easy to write and run highly parallel programs in new cloud programming paradigms:
Google: MapReduce and Sawzall
Amazon: Elastic MapReduce service (pay-as-you-go)
Google (MapReduce)
- Indexing: a chain of 24 MapReduce jobs
- ~200K jobs processing 50PB/month (in 2006)

Yahoo! (Hadoop + Pig)
- WebMap: a chain of 100 MapReduce jobs
- 280 TB of data, 2500 nodes, 73 hours

Facebook (Hadoop + Hive)
- ~300TB total, adding 2TB/day (in 2008)
- 3K jobs processing 55TB/day

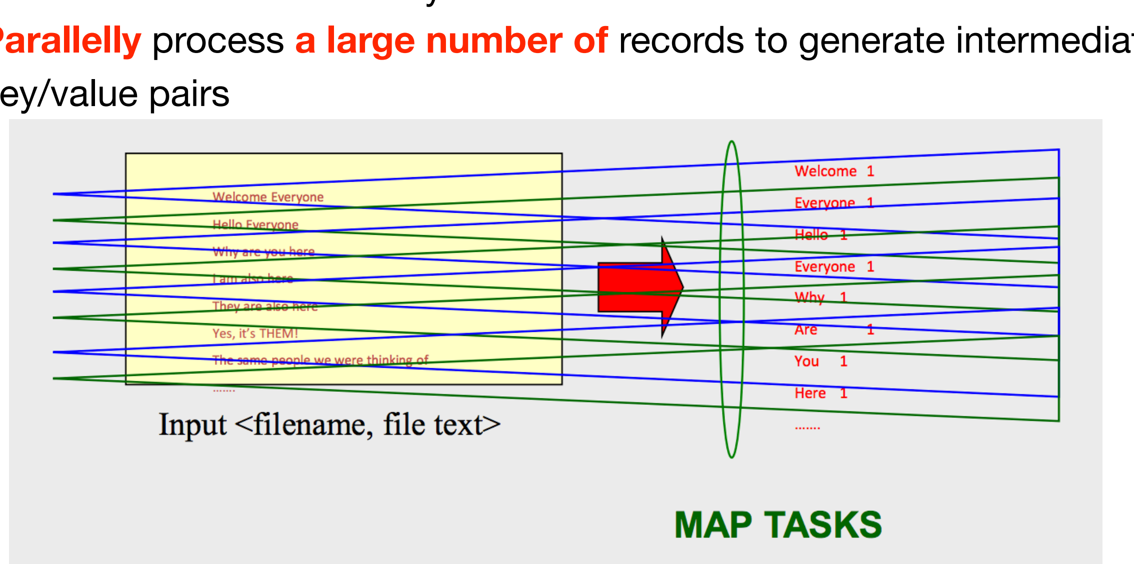
Similar numbers from other companies, e.g., Yieldex, eharmony.com, etc.
NoSQL
- MySQL is an industry standard, but Cassandra is 2400 times faster!

MapReduce Topic

Video: https://class.coursera.org/cloudcomputing-001/wiki/Week1Overview
(1) MapReduce Paradigm
Map
- Process individual records to generate intermediate key/value pairs
- **Parallelly** process individual records to generate intermediate key/value pairs

```
Text                 Key        Value
Welcome Everyone     Welcome      1
                     Everyone     1    MAP task 1
--------------------------------------------------------
Hello Everyone       Hello        1
                     Everyone     1    MAP task 2
```

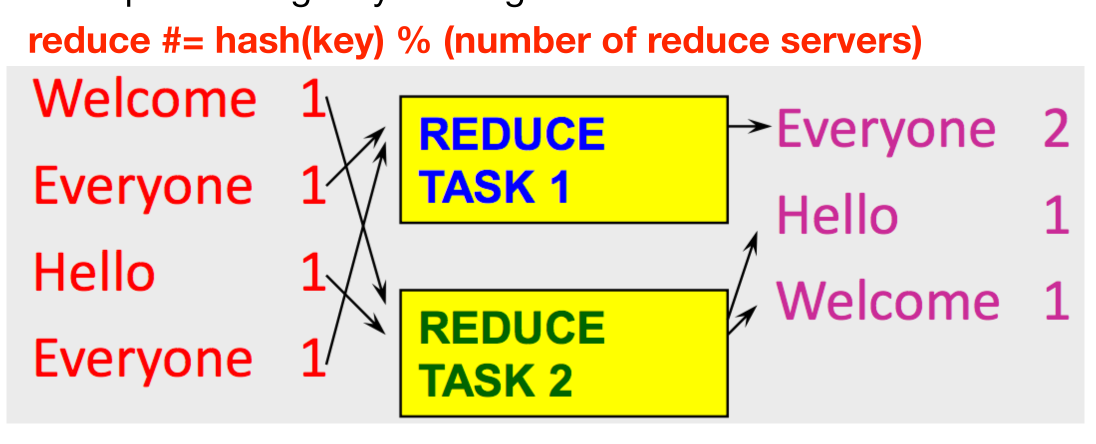
- **Parallelly** process **a large number of** records to generate intermediate key/value pairs



Reduce
- Reduce process and merges all intermediate value associated per key

```
                         Key        Value
Welcome     1            Everyone     2
Everyone    1     =>     Welcome      1
Hello       1            Hello        1
Everyone    1
```

- Each key assigned to one Reduce
- Parallelly process and merges all intermediate value by partitioning keys
  Hash partitioning: key is assigned to
  reduce #= hash(key) % (number of reduce servers)



----

(2) MapReduce Examples
Distributed Grep
Reverse Web-Link Graph
Count URL frequency
Sort

----
(3) MapReduce Scheduling
Externally: For user
- Write a map program and a reduce program
- Submit job and wait for result
- Need to know nothing about parallel/distributed programming

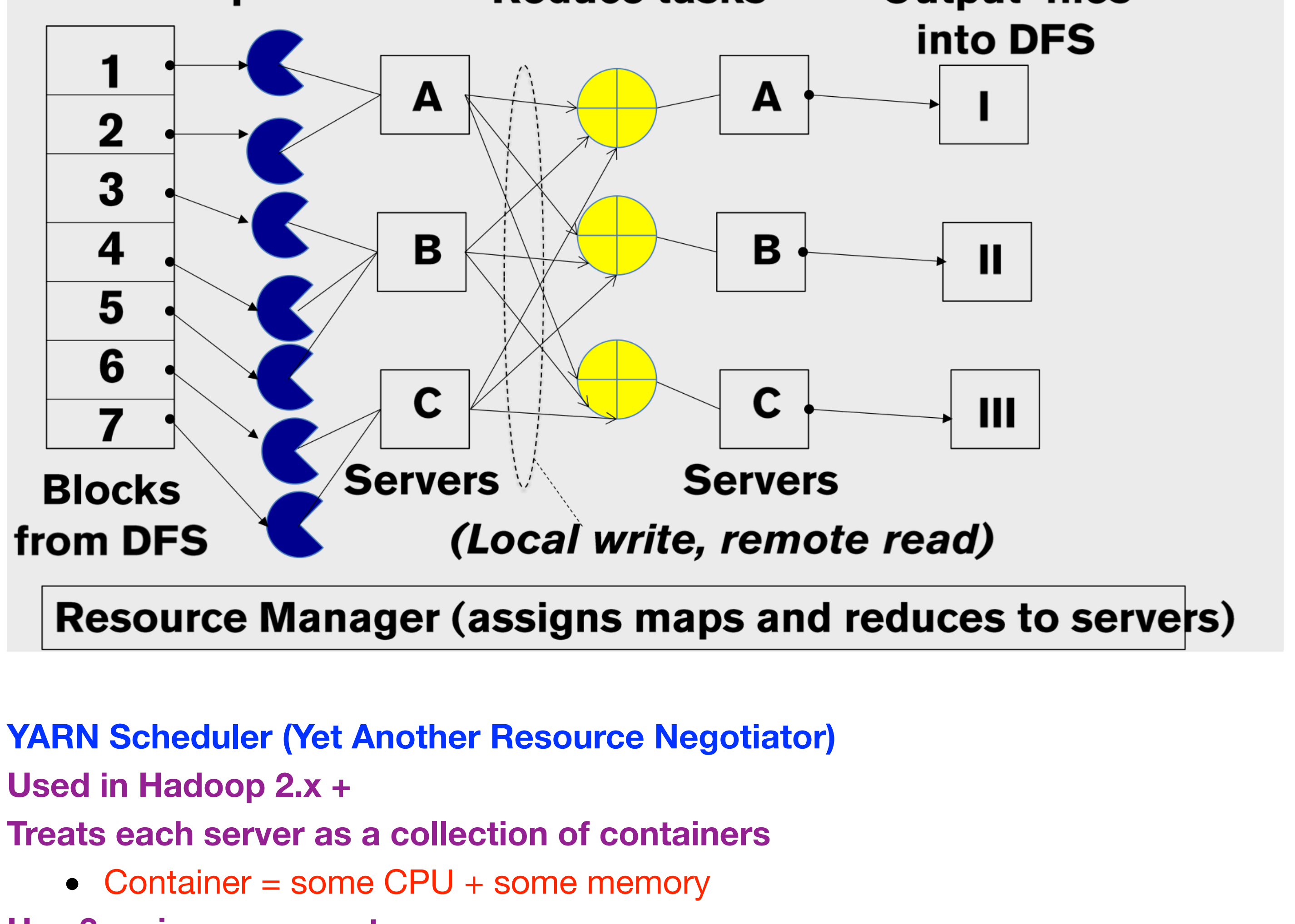
Internally: For the paradigm and scheduler
(1) Parallelize Map: easy! Each map task is independent of the other!
- All Map output records with same key assigned to same Reduce

(2) Transfer data from Map to Reduce:
- All Map output records with same key assigned to same Reduce task
- Use partitioning function: hash(key) % (number of reducers)

(3) Parallelize Reduce: easy! Each reduce task is independent of the other!
(4) Implement Storage for Map inout, Map output, Reduce input and Reduce output:
- Map input: from ***Distributed File System***
- Map output: to local disk (at Map node); uses ***Local File System***
- Reduce input: from (multiple) remote disks; uses ***Local File System***
- Reduce output: to ***Distributed File System***

Local File System = Linux FS, etc
Distributed File System = GFS (Google File System), HDFS (Hadoop Distributed File System)

Internal Workings of MapReduce

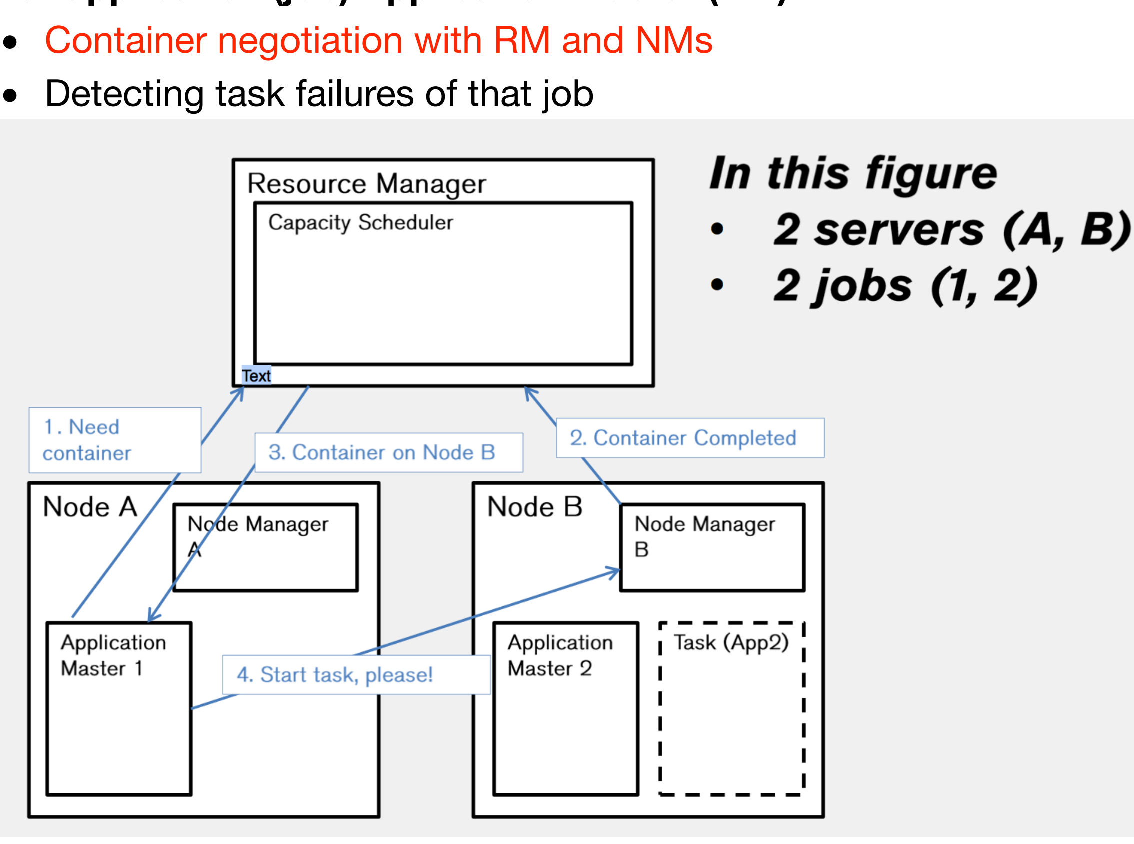


YARN Scheduler (Yet Another Resource Negotiator)
Used in Hadoop 2.x +
Treats each server as a collection of containers
- Container = some CPU + some memory

Has 3 main components
(1) Global Resource Manager (RM)
- Scheduling

(2) Per-server Node Manager (NM)
- Daemon and server-specific functions

(3) Per-application (job) Application Master (AM)
- Container negotiation with RM and NMs
- Detecting task failures of that job



----
(4) MapReduce Fault Tolerance
Server Failure
NM (per-server) heartbeats to RM
- If server fails, RM then let all effected 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

AM heartbeats to RM
- On failure, RM restart AM, which then syncs up with its running tasks

RM Failure
- **Use old checkpoints and bring up secondary RM**

Heartbeats also used to piggyback container request
- Avoid extra message

Slow servers (Stragglers)
- **The slowest machine slow down the entire job down**
- **Due to bad disk, network bandwidth, CPU or memory**
- **Keep track of "progress" of each task (% done)**
- **Perform backup (replicated) execution of straggler task: task consider done when first replica complete. Called Speculative Execution**

Locality
Since cloud has hierarchical topology (e.g. racks)
GFS/HDFS stores 3 replicas of each of chunks (e.g., 64 MB in size)
- maybe on different racks

MapReduce attempts to schedule a map task on
- A machine that contains a replica of corresponding input data, or failing that
- The same rack as a machine containing the input, or falling that
- Anywhere