Cloud Computing

lec 1 | L1

Computing as a utility

H2O | gas | electricity

$ $ $$

pay based on usage ↑ meter

Goal: operating system on demand
* illusion of infinite resources
* no up front $
* fine grained billing
ex: hourly

def
style of comp where scalable IT provided as service over internet

1969 Leonard Kleinrock
ARPANET Project
predicted: comp network → utility

1969	1990's	2001	2004
Grid	Utility	SaaS	Cloud
solve BIG prob w/ parallel computing	comp resources as a metered service	network-based subscription 2 app	NEXT GEN Internet Computing
by Global Alliance	failed bc weak internet	2001	NEXT GEN Data Centers

Why now?
① lack of experience w/ large data centers
* unknown pricing model
* risk of transferring
② Tech Factors
* growth of internet
* growth of VM
③ Business Factors
* min capital
* pay-as-u-go

data store
$ on bandwidth
& storage
ex: Amazon-S3, Simple DB

Virtualization

- basis of cloud comp
- partition hardware → flexible & scalable platforms
- techniques:
  - (A) VM ware & Xen
  - (B) OpenNebula
  - (C) Amazon EC2

1 physical machine
VM 16 VM 16 VM 16 VM 16

management
Hybrid/elastic for
Amazon Cloud
player
Amazon EC2 service
# of instances

asking for 10 VM → get 10 VM
not cost effective

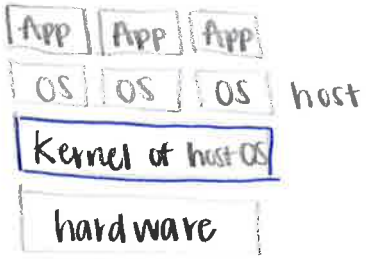
slows
exit computing
solution: containers

App | App | App
OS | OS | OS
Hypervision
Hardware

ex
game
Linux & windows
seperate VM
apple

Containers

by Google



+ light weight (uses only required resources)
+ faster start up than new VM
+ near native performance

ex: Docker, LXC, Linux VServer

Themes

* scalable
* dynamic
* elastic (use what u need)
* accessable via internet
* Transparent
  \ complexity is concealed from user
* service oriented

Services

Software
- access app through internet

ex/ Facebook, Instagram, gmail

Platform
- create app through internet
  stored in Cloud or ether

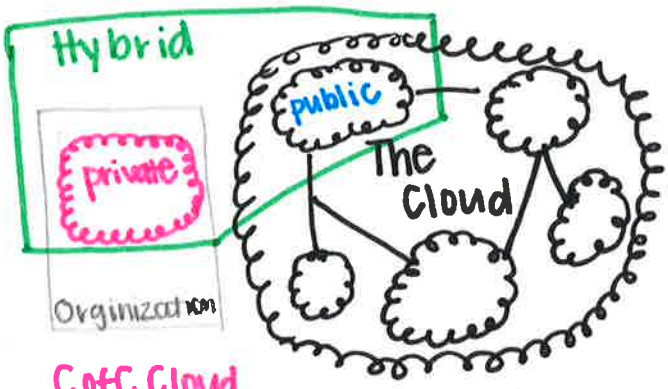
ex/ Google App Engine } Hadoop / Heroku

Billed on AD Revenue } Premium Service
- Software stack
  - Linux
  - Appache
  - MySQL
  - PHP

Infrastructure
- use of computers across internet to
  * parallel processing
  * distribute storage
  * index } data mine
- virtualization

ex/ AmazonEC2 } SciCloud

- OS on demand
- data Store
- runtime
- LB
- Resource Provision



CofC Cloud

Amazon EC2

community
(multiple universities

Short Term Implications

① Startup } prototyping
↓ cost of entry

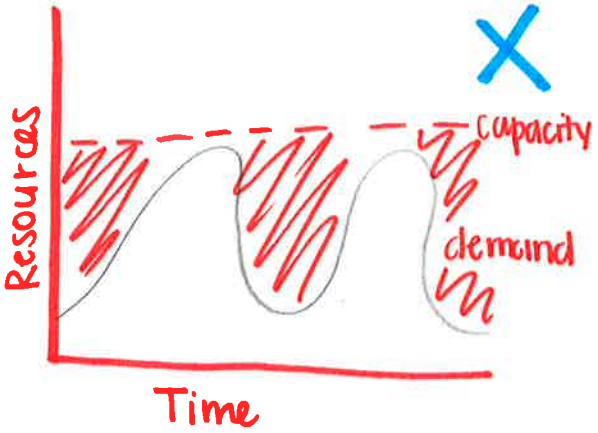
② Batch jobs
ex
banks
└ push all transactions
@ night ↑ recommit

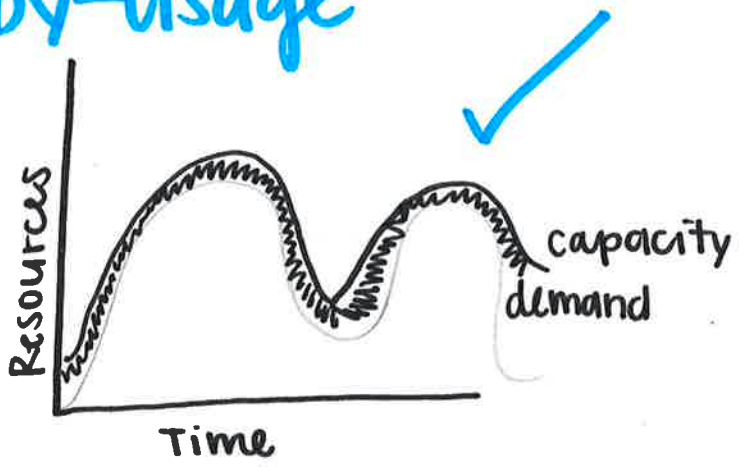
③ One-off Task
NY Times
└ digitize all earlier copies

④ Cost efficient for scientific comp

⑤ research @ scale
* test which model is best

Pay-by-usage





risk of overprocessing
└ under utilization

Sunk Cost

Long Term Implication

App Software (startups)
Cloud ⅋ Client parts
Infrastructure Software
VM awareness
Hardware Systems
Containers

Econ of Cloud Providers

* build large datacenter
  └ $$$ $100 mili
* large companies already own
  ex/ google, Amazon

Cost of | Medium DC > Large DC
Network
Storage
Admin

* Power
  - ✓ 3.6 ¢ | Idaho | hydro electric
  - x 10 ¢ | Cali | transmit ele over long distance
  - x 18 ¢ | Hawii | ship fuel $$
* Cooling
  └ dc near rivers

* world's 5⅋6th most energy consumption*

users
startups
↓
research
↓
enterprise
↓
edu
↓
HPC

generating revenue

Microsoft : windows is already downloaded to VM ⅋ they send you access
→ pay for liscense

Google : reuse infrastructure

Policy ⅋ Business

Reputation Fate Sharing	customers creating SPAM content (cloud providers click 4 this)
Software Liscensing	Pay 4 liscensing ex: windows

Econ of Cloud Providers - Failures

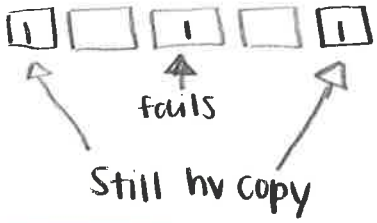
high availability servers → commodity servers

@least 1 failure per day....

low per server $

SOLUTION: replicate data & computation

ex: Mapreduce



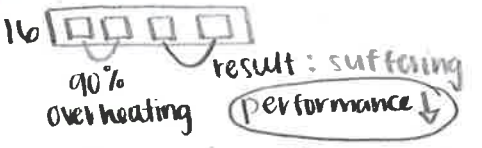
Adoption

Challenges	Solutions
Availability	what if their servers go down?? ✓ Multiple Providers instead of 10 google (8 amazon, 2 google) — Microsoft & Azure ✓ auto scaling to prevent DDos attacks 
Data Lock-in	Standardization Locked into 1 company ex/ Amazon S3 └ can't migrate from google to amazon
Data Confidentiality	✓ Encryption ✓ Firewall ✓ Geographical data store

Growth

Data transfer bottleneck	✓ Fedex disk ✓ Amazon Snowmobile
Performance Unpredictibility	✓ improve VM support ✓ Schedule VM ✓ Redistribute VM
Scalable Solution	Scalable Storage
in Large Distributed Systems	map reduce invent debugger 4 dist VM
Scaling FAST	✓ auto Scaler for VM ✓ snapshots

Scale storage & VM



Cloud Providers

Cloud Services that other apps can utilize
- $ FREE
- ex/ Google / Yahoo Maps
  - " " Web Services

VM Image
- 2 instances
  - 1 instance load some → saved as Snapshot or VMI
  - 1 instance software → VMI
- volume – virtual hard disk
  - ↑ disk space

Amazon Web Services — Public Cloud Prvider

- **Elastic Cloud Compute (EC2)**
  - * early pioneer of cloud computing
  - * OS on demand ..... Linux, Windows, ...
  - * complete virtual comp (based on XEN VIP)
    - flavors
      - CPU
      - Mem
      - Disk Space] Large Min (160gb)
    - bit form image

$ Model = CPU Hours + (Machine) size + bandwidth in → out
ex: small VM

- * FAST startup (2-3 min)
- * instance snapshotting
  - state of VM instance → you can create the instance again from snap
  - # of hours instance = ↑↓
- * flexible API to control everything
  - test (browser)
- * 300 - 400 instance types ...
  - general
    - micro
    - Sm → XL
  - advanced
    - ↑ mem — ex/ image processing
    - ↑ CPU — ex/ scientific computing
    - ↑ ....

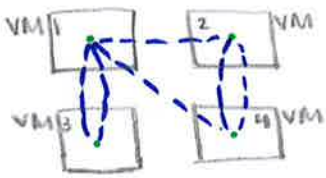
- * **Terms:**
  - **access key** private PEM key to access VM thru Network
  - **Secret key** contains + access key Id
  - **Security group** allow instance access to ___ server
  - **availability zone** ≈ ex/ us East (OHIO)
  - **Amazon Machine Image**
    - ex/ start Virtual Box from VMI
    - VM File
    - ami - XXXX
    - stored in AS3
  - hardware-file
  - creates instance ← Amazon Machine Image
    - virtual server
      - *CPU
      - *Mem
      - *Storage

everything
- hardware
- VM
- host
- copy

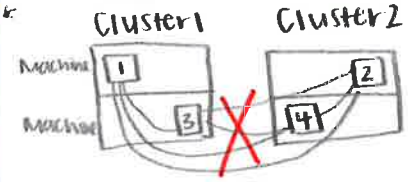
→ Start VMI2

bet snapshot
lighter than VMI

Message Processing Interface
do calculation & exchange info



↑ looks like ↓



problem??
every step you are sending message
solution?? 1 cluster

✓ $$

public / private instances

only S3 storage $

running Machine
amazon instance i-xxx

elastic IP addresses

public IP amazon provides

no cost = VM + IP address

cost = ~~VM~~ killed + IP address (not released)

Pricing Models

On Demand – Short term
unpredictable workload

Spot Instances – EC2 instance 90% of on demand $$
ex/ use 50% other 50% cheaper
urgent needs for Large
Amazon will terminate when they need w/out notificatio

Reserved Instances – 1 to 3 yr deal

Dedicated Host – Physical Server for me

Per Sec Billing – manage instances running for irregular time
ex/ dev/ testing
data processing
batch processing
Serverless computing

Troubles

- if power on & off = all hard disk data is LOST
  use snapshot
- ip addy is random
- can NOT turn off public IP addy
- don't forget to terminate instance

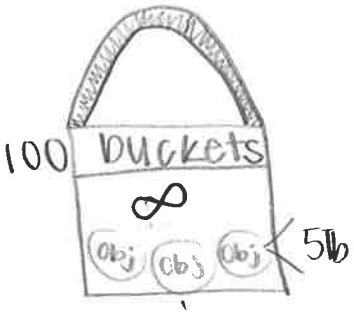
cheaper

• Simple Storage Service (S3)

web

* ↑load, ↓load, ∃ store data
* Bucket store data
  ↓
  container

Publically accessable



100 buckets per account
no limit of # of objects
- data
- meta data

5TB >

Stored in location NEVER Leave
Can NOT modify / append data
SOL: copy object
make changes
store as new obj
delete old obj
get unique key

copy
modify
Store as new
delete old

$

.023 per GB/Month for 1st 50TB
.022 per GB / Month for 450TB

* download data anywhere
  enable others ∃ charge via amazon dev Pay

faster FS

only through EC2

• Elastic Block Storage (EBS) ≈ external hard disk

have VM w/ data → kill VM → download data → create new VM! → upload data

Solution: VM + Volumes
- 1 GB to 16TB
- 20 per account
- stores snapshot
- file storage

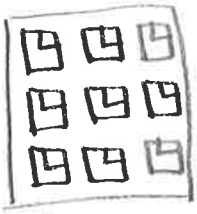
* automatically replicates to prevent data loss

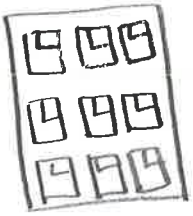
write lots of Scripts

20 volumes < 16TB
∃
snapshots

• Management Console

instance /hr
volume / Month
Network / Month





Management Provider

Elastic Fox

- Firefox for managing Amazon EC2
- Hybrid Fox (fork of elastic fox)

Similar

Amazon Management Console

ISV : independent Software Vendors

My own DB

S3 Firefox Organizer

EC2 ? EBS?

S3

Private Cloud Provider

for circulizing resources

Eucalyptus

Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems

Web based version of EC2

function as a software overlay

purpose

- ✓ tech preview platform for Public Clouds
- ✓ basic dev platform for open source community

everyone uses openstack

OpenStack

founded by Nasa & Rackspace

EC2 **NOVA** start VM → computational cores

S3 **SWIFT** store objects

Glance store image

EBS **Cinder** block storage

Compute

Volumes

Network

glance - img

Nova - compute

Swift - obj

SciCloud

1st private cloud @ UT

uses Eucalyptus & OpenStack

Scaling Applications

Scaling Information System

MANDATORY (entipise app deployment)

* **fault tolerance** – if 1 node fails ALL do NOT
* **high availability** – servers are available 99.999% of the time
* **scalability** – nodes support a specific load

when Load ↑↓ System should Scale up / Scale down

1 server can support 8-10 req in 100 ms

Large - - - - - - - - 12 req

Med - - - - - - - - - 8 req

Small - - - - - - - - - 6 req

2 Scaling Models

① Vertical Scaling ↑

as load ↑ replace current node with more powerful machine

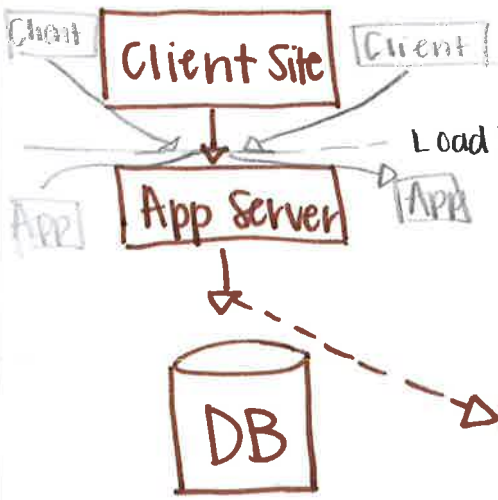
CON: risk of losing current running job during switch

ex/ small to medium

② Horizontal Scaling ↔

add servers w/ existing servers
add more nodes to system

ex/ MapReduce & Hadoop



Load Balancer – distributes incoming traffic 2 multiple servers

ex : sites to servers
✓ response time improved
✓ web server farms
✓ hides complexity from site

when pages are dynamically created and clicked multiple times then it gets stored so it doesn't need to be regenerated multiple times

Load Balancer Types

(A) Network Based

google.com → DNS server routes IP
- US IP – server
- EU IP – diff server

(B) Network Layer

not flexible

all requests coming from 1 IP get routed to 1 server

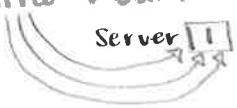
(C) Transport Layer

all request coming from 1 location → 1 server (room: 235)

useful if connections are short & established frequently

(D) Application Layer

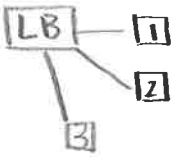
buy something online → cart



Classes

Non adaptive

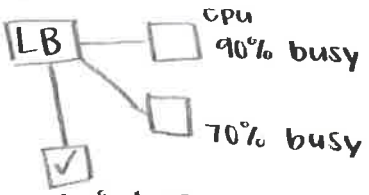
doesn't check current load



ex: Round Robin

Load Distributer

Adaptive



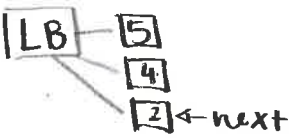
where next request goes

ALG

Random

Round Robin

Join-Shortest-Queue



EX

- Ngix in lab
- HA prox
- Pen

Testing w/ Simulation Load concurrently

- Benchmark Tools
  - Tsung, Jmeter
- Multiple Protocalls (supported by benchmark tools)
  - HTTP, XMPP
  - SSL support

Bottleneck: DB can only support 25 requests (SQL read) *1 r 3w*

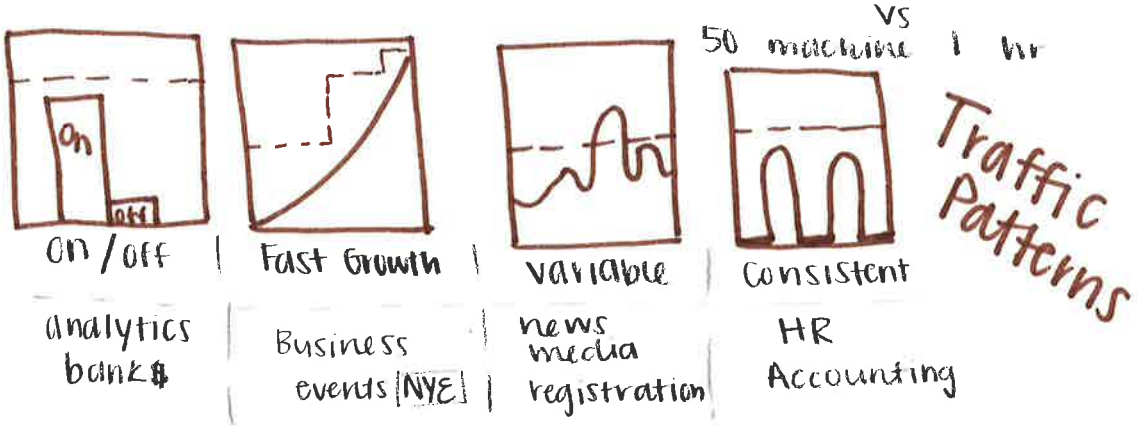
SOL: NOSQL or MongoDB
Scalable & elastic

Auto Scaling

dynamic reaction to defined metrics by Scaling

- ✓ ↑ availability
- ✓ Cost Saving
- ✓ Energy Saving

ex
- website w/ unknown visitor #
- viral marketing campaigns
- apps that need elasticity
- Scientific app
  - 1 machine 50 hr
  - vs
  - 50 machine 1 hr



Scaling Policy when to Scale (typically threshold-based) configure optimally is hard

Resource Provision Policy How to Scale

ex / Amazon AutoScale

Vendor Neutral

static LB + estimate Load on each server

static LB + optimal resource provision policy

Amazon Auto Scaling (comercial)

lecture 3 | 4

Scale computing resources dynamically & predictibily
- dynamically: >75% add 1, <35% kill 1
- predictibily: every friday add 1

A EC2 instances → Auto Scaling Groups

define min/max # of instances] prevent DDos attack

Amazon Watch

Monitor **EC2 instances** → all metrics & set alarms
CPU / mem / ...

Elastic LB

distributes traffic across multiple **EC2 instances**
* detects health
  └ diverts traffic from bad ones

my own autoscaling system

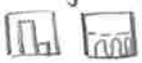
LB

Measure Performance of Current Setup

* Linux shell commands
* Collectd – install on every server & to master
  - no performance impact
* Cacti – gives graph too
  - 20% ↓ performance

Scaling Policy

Time – every Friday @ 1 pm



Reactive – threshold: if >75% in last 5 min

Predictive – next = mean of last 5 min

Resource Provisioning

cloud servers are heterogenous

Simple – estimation based on Heuristic (if ... then ...)

Optimal – various instance types (diff power & price)
- make policy about current configiration

add servers?

Consider you pay per hour so you don't want hour = START → kill

what is the current machine?

what am I paying for?

ex/ 2 - S → 12
1 M → 8
1 L → 36
56
↓ add S or add M kill S
58

ex

S	6	$0.25/h
M	12	$0.4/h

load 6 → use 1 small ✓

load 12 → sol 1

add 1 small = total 50¢

cost = cost of 2S − 10 min but NO Time Lost
= 50 − .04 10 min profit } 46¢

sol 2

Kill small ∄ add med = total

cost = cost of m + 10 min of s } 44¢ ✓

= 0.4 + 0.04

Definitions:

Region — 1 task w/ independent characteristics
own capacity of instances

Instance Type — each region = MULT instance Types

Time bag - time ⟷ interval where an instance is

Killing Cost - $ LOST when instance is killed before full pay period

Retain Cost - $ of lived duration pay period

Apply This

(1) identify scalable components ex: not db → instances

(2) parameters of ↓ ex: # of small support? 6

(3) combine w/ current load

Amazon AutoScale

Dynamic Deployment Template

providing the same snapshot for all servers even new ones added during scaling

ex/ CloudML ∄ TOSCA

MapReduce

- data processing from 1950's handled @ scale
  - 1 map & 1 Red
  - complex mult map & red
- functional programming + distributed file system
- Google 2007

Map:
- high order function (output of another method as input)
- apply function to every ele in [] → new []

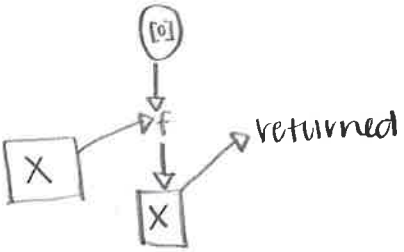
```
map f lst
```

Fold:
- high order

```
fold (method(), [])  → single val
fold f x Lst
```

x: accumulator val

inital motiv



ex
```
square x = x * x
map square [1,2,3] → [1,4,9]
```

ex
```
fold (+) 0 [1,2,3,4,5] → 15
fold (*) 1 [1,2,3,4,5] → 120
```

ex
```
fold (+) 0 (map square [1,2,3])
```

Implicit Parallelism

Map & Reduce tasks concurrently
elements are independent

- partition list between computers
- apply map to each partition
- combine map results w/ fold

associative: order does NOT matter
(a+b)+c = a+(b+c)
// parallelisation

LARGE Data Probs

Map
- * iterate over Large # of records
- * Extract something from each ex: want ID
- * shuffle & sort results] group ID's w/ "3"

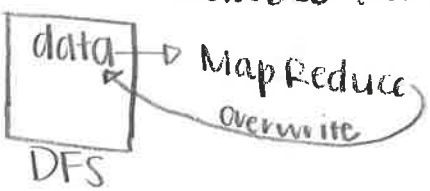
Reduce
- * aggregate results] sum of all ID's w/ "3"
- * generate Final Output

MapReduce

input : [(k, v)] list — (offset, Line content)

- designed for processing Petabytes
- deployed on Large cluster of servers

↔ Scalable (add servers)



map: $(K, V) \rightarrow (K', V')^*$ → mult (K', V') → 0

reduce: $(K', [V'])^* \rightarrow (K'', V'')^*$ → 0

any function

1 $(K', [V'])$ all vals
Sum [v']

1 val per key

User defined
Function for map
" " reduce

framework takes of handeling map reduce (failed nodes

dd ① data distribution
partition & replication

S ② Scheduling
assign workers to map & r

Sync ③ Synchronization
Sorting int data

error hand ④ ERRORS
detects & restarts

Slow work ⑤ Slow workers & re executes

Partitioner

how map output keys are distributed across reduce

Partition (key, partition) = HashFun(key) mod partitions

= 0...99

Partition Group

A - 1
B - 1
E - 3
D - 2
C - 1

partition (key, partition)
- Hash (key) % partitions

Combiner

optimize network traffic
mini reducer stage in mem

Combine (k', [v']) → (k'', v'')*

--- single val
---- (key, single val) passed into reduce

① Reduce waits for Map to finish → stored in mem
intermediate data

② data is orderd by key be4 reduce

Who implemented MapReduce?

- Google (C++)
- Hadoop
  - opensource by Yahoo, now Apache
- Custom... ex: DB = CouchDB

how to move data 2 worker?

DONT move worker 2 data

limited
* bandwidth
* not enough RAM to hold data in Mem
* disk access is slow

local disks on the node in cluster

DFS
ex: GFS google
HDFS hadoop

DFS

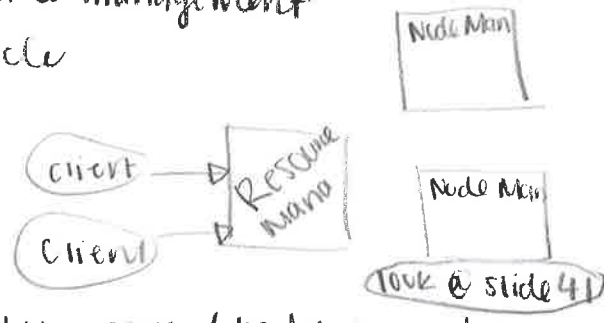
- pick cheaper commodity hardware (new) — scale ⟷
- #'s of Huge files (multi gb files)
- once a file is written... must append to it — generate new file or overwrite it
- Large reads over RAM

* store files as chunks VS GB
  - easier to run Map tasks on (64 mb)
* replicate chunk across 3 nodes
* 1 MASTER Node handels access to RAM (contains Namenode manager)
  - garbage collection
  - re replication / balancing
* no caching
* Simplified API

Hadoop Model

... YARN — resource management
allocate
input - HDFS
run
output - HDFS
Job life cycle

1. create cluster
2. put data → into blocks
   replicate/store in cluster
3. Job Run .... copy map 2 nodes — task - run map / reduce on block
4. store results in HDFS



task attempt - if it fails mult
node will be abandoned

more nodes = better preformance
Scales to petabytes of data

scalability probs
master node has too many jobs
resource utilization
if fails abandons ↑ node

MapReduce Alg's

How to get output from input?

Map

Input (k, v) → Output (k', v') *

Combiner Reduce

Input (k', [v']) → Output (k'', v'') *

partition (k, # of part → hashfun(k) % partition → part ID

Counting URL Access Frequency

sim 2 wordcount

Map

Input (Line #, Line) → Output (URL', 1) if appears in Log

Reduce

Input (URL', [1', 1, ...]) → Output (URL'', Sum of [1'])

Distributed Grep

find all rows in .txt files that contain Regular Expression (\.\d+)?

Map

Input (Line #, Line) → (Line #'', Line'') if has regular expres

w/out Shuffle Reduce

.... Identity Function....

Inverted Index

word is in [file1, file2]

Map

Input (Line #, Line) —get each word from line→ (word, File name)

```
map (line#, Line, context){
    pageName = context.getInputSplit+Filename()
    for each word in Line:
        emit (word, in)
```

Reduce

Input (word, [Filename]) → (word, "[file Name]")

```
reduce (word, values){
    page List = []
    for each pageName in values:
        page List.add (pagename)
    emit (word, str (set (page list)))
```

Distributed Global Sort

22
59
64
30
50
20
.txt's

partition 2
20
22

partition 5
59

Start ordering from Map 2 Partition 2 Reduce

if k1<k2 then Part(k1)<= P(k2)

Map
(Line#, Line) → move VAL to key (Line, "")

Reduce
partition → Reduce 0
..... Identity function ....
Reduce
0 ← Reduce 1

Term Co-occurance Matrix

i } j in context

terms²

keys are large so combiner doesnt help

Keys

	Term 1	T2	T3
T1	0	1	5
T2	1	0	1
T3	4	count of Val T2 } T3 in input data	0

Val: depends on input data

"You shall know a word by the company it keeps"

Why? Semantic distance

is the sentence real? compute frequency of words together

MapReduce is inefficient bc syncronization

Seperate tasks = need synchronization (hold map data in mem)
(partitioner)

X **Pairs** ("T1, T2", 1)
("T1, T2", Σ1)

too many pairs
x lots of shuffeling

√ **Stripes** Map ("T1, T2", 1)
("T1, T3", 1)
> T1 → {T2:1, T3:1}

x storage allocation
x code=hard

Combine all sentence's T1 → {T2:1, T3:5, T4:2}

Conditional Probability : Syncronization Approaches

What is the pob that TermB if A is already in sentence?

$$P(B|A) = \frac{count(A,B)}{Count(A)}$$

#1 Pairs

(2nd) gives count(A, B) = ("A, B", 1)

(1st) get count(A) = ("A", 1)

get val 1st

how? Partition all "A" & "A, _" to 1 reduce

countA = Store in mem ~ might crash

count "A, _"

countA

emit countA

emit count "A, _"

Cheating

#2 Stripes

compute "A, *"

compute P(B|A)

Trade offs between pairs & Stripes

(A) Creating (K, V) pairs = lots of time

(B) Size

(C) Combining ✓ always better to combine values in Map

Complex data types in Hadoop

easy — string
\ regular expressions

hard — create write comparable

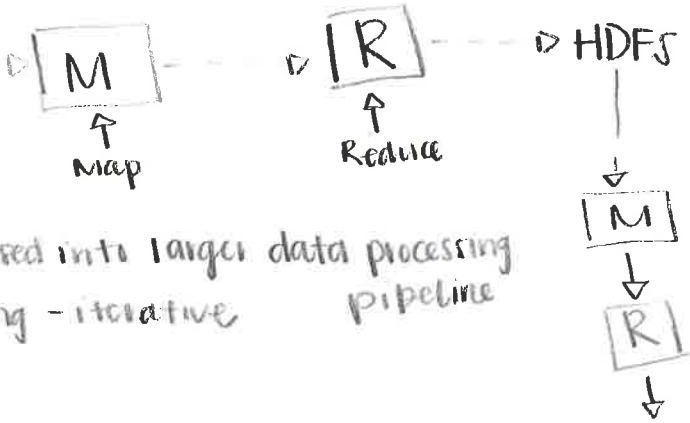
MapReduce Ex: Lab 5

Why would you use Hadoop?

* Extract, Transform, Load pipelines
  ex: yelp review dataset → analyze who has the best positive reviews
* Reporting & Analysis
  ex: websites when data is too large for sql database
* Machine Learning
  ex: recomending products to users
  ex: apply java library to image to abstract face for ppl regoghition as key & val
* graph algorithm
  ex: trends in social media
  page rank

Map Reduce Jobs

* typically shat, code wise
  ex: identity reducer
* represents a data flow, rather than tasks



* can be composed into larger data processing pipeline
  ex: gaming - iterative
* data MUST be partitioned across many reducare tasks that write out to diff HDFS files] IF Large
* diff input formats

Designing MapR Alg

Goal: what is the desired output from the input data?

① Map Function
- what is Map Input (key, val) } function
- what is Map Output (key, val) }

② Reduce Fun
- what is Input (key, [value]) } function
- what is Output (key, value) }

Term co occurrence matrix

of times word is in sentence

key	Term1	Term2	3
Term1	0	2	3
Term2	1	0	1
key Term3	4	count of T2, T3 = 7	val = 0 ... depends on input data

Terms^2 = Matrix size

keys are large → combiners don't help

why? how often do terms occur near each other

# of times _ occurs in _

:

figure out if this is real eng sentence or not

(#1) Pairs approach [("T1, T2", 1) key, val

✓ easy implementation
✓ easy understand
X too many pairs

(#2) Stripes [combine (k, v) pairs emit
a → { b → 1 ; c → 3 ; ... }
array or dictionary

✓ less step, map & r between short & shuffel
✓ better combiners
X diff to implement

Reducer: count based on 2nd key

* Map & reduce are independent so you process each k, v
* tools for sync
  - Keep int values in memory in hashmap (combine in map)
  - Sorting functions for keys
  - broadcasting ... assign value to all values (loop)

conditional Prob(B|A) ...
a a

Reduce → (k, [V]) / (v1, [V])

("A, B", 1) (A, 1) ← terms — complex key

Sort & partition
:
only use 1st part

how can we force all A's A1 — comes to certain reduce
compute (a, *) = # of a
then (a, b1) ... (a, b...)
(a, *) ... ...

Stripes ... prevent all these prob

Platform as a Service

Manages... infrastructure
+
software enviornment

* multi tenant architecture
* built in scalability
* integrated w/ cloud services & DB
* simplify prototyping
* $ for ONLY cloud services used — upfront $ ↓
* Vendor Lock in

Type 1: web.app

* built on Iaas Cloud
  ex: google AppEngine — exception bc before Iaas
* open-source computing platforms (not tied to Iaas provider)
  ex: Red Hat OpenShift deployed on diff clouds
* social app platforms (upload my own game 2 facebook)
  L add ons for Facebook / Google +

Type 2: Function as a Service

platform for hosting microservices / Serverless

"App" callable function

Trigger: upload new image to S3

Precondition: file size > 10 MB

Execute: resize_image (file)

ex: AWS Lambda λ
&
Apache OpenWhisk

Type 3: Data Processing as a Service

deploying data processing applications

* provider manages enviornment } cluster

user ↑load: app + resource limit

ex: AWS Elastic MapReduce

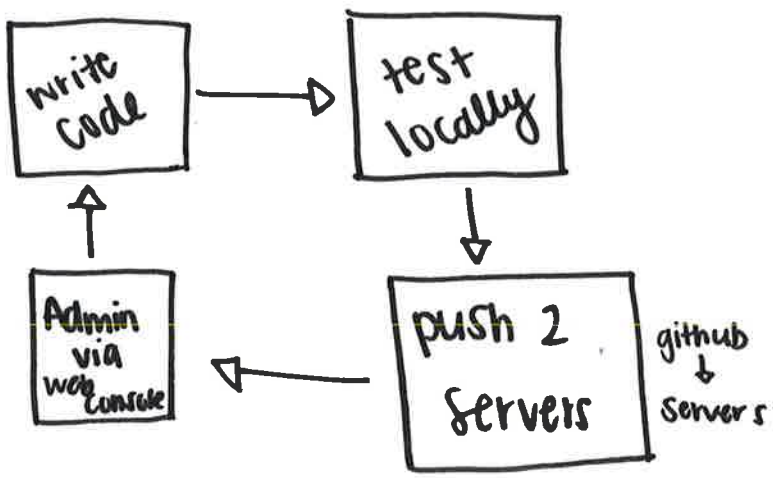
}

Google Cloud DataPro

Google AppEngine

* PaaS for dev } hosting apps
* easy to build, maintain, } scale
* nothing 2 manage unless you want to optomise

✓ persistent storage ↑load files
✓ scales servers autoscale
✓ dynamic traffic
✓ asynchronous task queues
✓ schedule tasks "delete file everynight"
✓ Integration w/ Google Cloud Services hv them login w/ google
✓ Secure, sandboxed, enviornment run instances in same server
isolated enviornment



Available Cloud Services

Google Cloud SQL fully managed relational db

DataStore schemaless objectstore.... GQL

Blobstore too big 4 datastore ↑load file > get key... generate links

Available Data Services

Search through structured data ... html, .txt

Memcache improve performance

Logs monitor performance

Remote lets external apps access Engine services

AWS Elastic Beanstalk

Microsoft Windows Azure

SQL db
no SQL

Hybrid IaaS + PaaS
designed for enterprise

ex: xbox 360
Toyota

- **Big Compute** HPC
- **HDInsight** hadoop clusters for data

Open Source:

- **AppScale** move apps from google App engine $$$ → Free
- **AppFog**
- **Cloud Foundry**
- **Heroku** 1st Paas on market

AMAZON

Management Console

90+ services

IAM: Identity & Access Management

how can I group use 1 credit card?

* Manage users create groups
* Manage credentials assign security status
* Manage permissions create policy per user
  ↓ generality
  resource "*"
  ↓
  in what instance

CloudFormation .json

template for hardware & software

free (only pay for services)

DB

Relational

Specify scaling & SQL

↓

done by YOU not Amazon

easy configuration

Aurora

specify type of SQL My... & PostgreSQL

+

they manage

5-10x faster

NO SQL

- Document
- Dynamo – NO SQL
- Neptune – graph

they manage

Pipeline

└ data movement BPM

Moving Data

* **Snowball** - 50 to 80 TB per Month

=

Edge - pre processing

+ launch instances (26 - 52 V CPU)
+ AWS Lambda
+ EC2 AMI
* GPU — real time video analysis
  | Machine Learning

100 TB

Cluster 20 Nodes

1 cloud server

1 **Snowmobile** - Petabytes to Exabytes

=

1250 Snowballs

Serverless?

Servers are abstracted from developers

event driven + triggers vs constant running
if there is a need

$ per second
sub

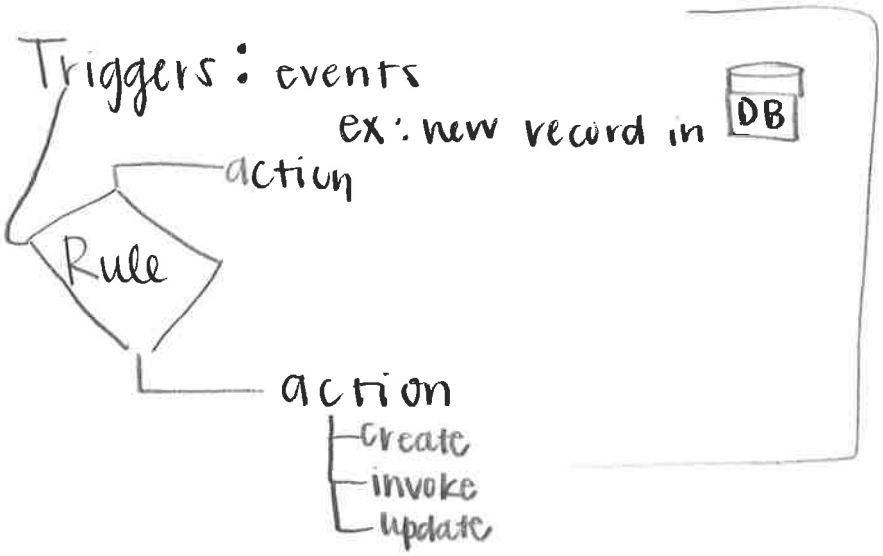
Best 4 newer workloads ..... Amazon Alexa hears u

linked w/ Cloud not standalone

ex: Amazon Lambda — use multi lang
Open .....

based on OpenWhisk — IBM — function is scope — scale

IBM blaowhisk
any lang



- ✓ easy 2 scale
- ✓ rapid Prototyping
- ✓ easy 2 modify functions
- ✓ pay 4 execution
- ✓ applications w/ multi languages
- X hard 2 avoid vendor lock in $$$ 2 migrate
- X tools 4 debugging (hard 2 verify error)
- X architecture is complex
- X slow cold-start
  - hot kept in mem
- X harder to predict $
- X need stateful computations