Databases and Big Data

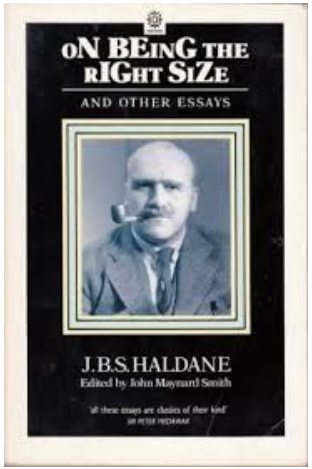
Cloud Computing

Recap

- Big data is everywhere
- Big data carries (potentially) big value
- Big data needs big infrastructure
  - Who owns it?
  - Who pays for it?

Design Principle

- **Principle of incommensurate scaling:**
  - Changing a parameter by a factor of ten usually requires a new design
  - **“On being the right size”**
- That means:
  - Supporting big data workloads requires different design to RDBMS.



Today

- User’s view
- Systems’ view
- Google Cloud Example
- AWS Overview

Cloud Computing

- ~~Informal: computing with large datacenter~~
- More accurately:
  - Computing as a utility
  - Do your work on someone else's infrastructure

Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz,
Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia
(Comments should be addressed to abovetheclouds@cs.berkeley.edu)

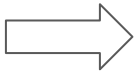
UC Berkeley Reliable Adaptive Distributed Systems Laboratory *
http://radlab.cs.berkeley.edu/

February 10, 2009

Cloud Computing

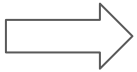
- Implication

Computing as utility



Economics (got to be cheap)
Quality of service (SLA)

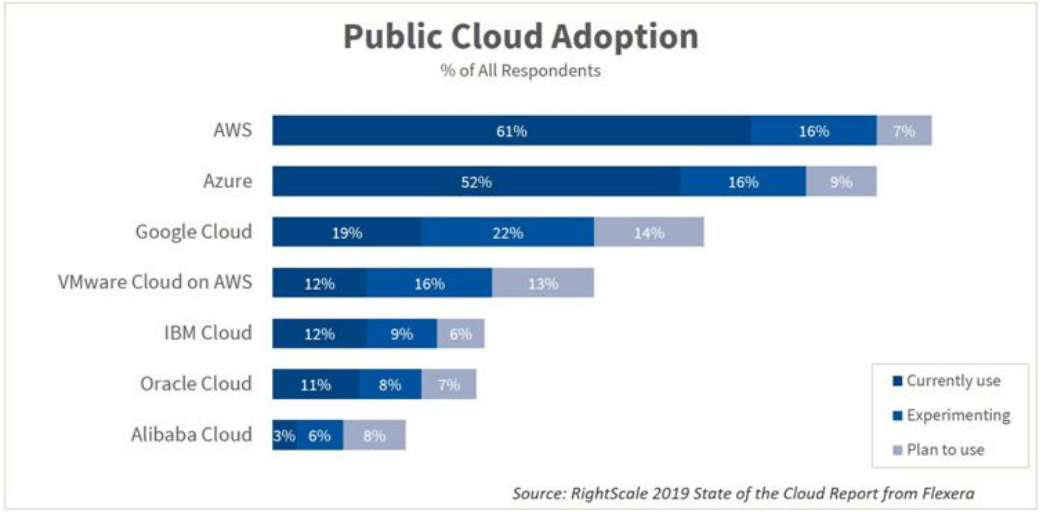
Computing on other people's infrastructure



Security, flexibility (your house, your rule)

Cloud Computing

- Public clouds
  - **Anyone can use**
  - Arrays of options
- Private clouds
  - Similar interface, but
    - Smaller
    - **Privately own**



Azure

amazon web services

Different design choices

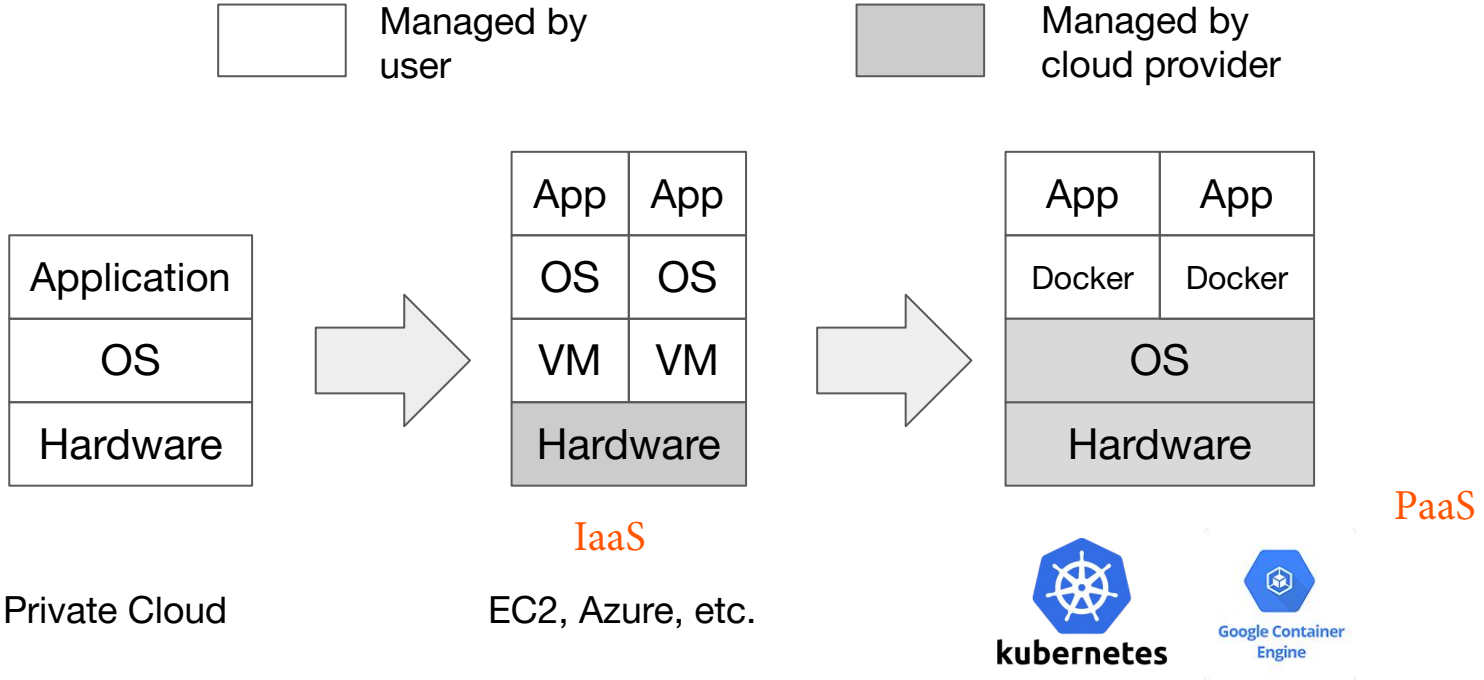
Cloud Computing

- Public cloud: our focus
  - Infrastructure as a Service (IaaS) VM, disks, etc.
  - Platform as a Service (PaaS) AppEngine, ERM
  - Software as a Service (SaaS) Github, GoogleDocs, AutoML

(Almost) Everything is a service

Cloud Computing

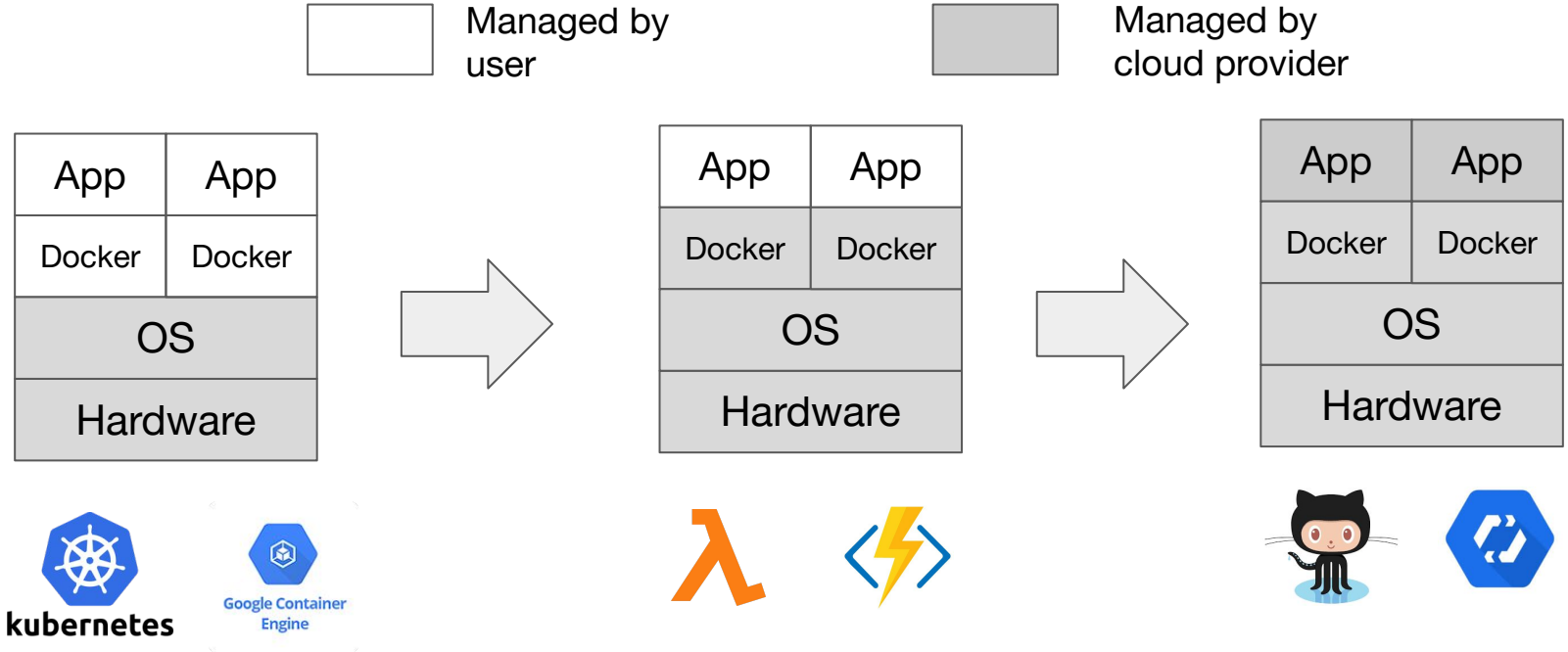
- Who manage what?



Cloud Computing

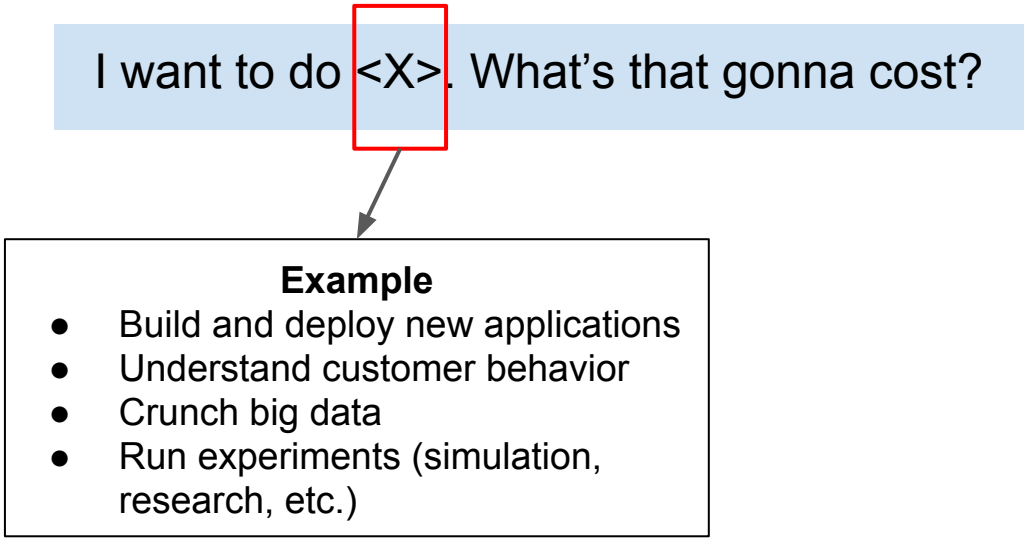
- Who manage what?

Let Cloud manage VM/Docker image as well
e.g. Lambda
Also another PaaS



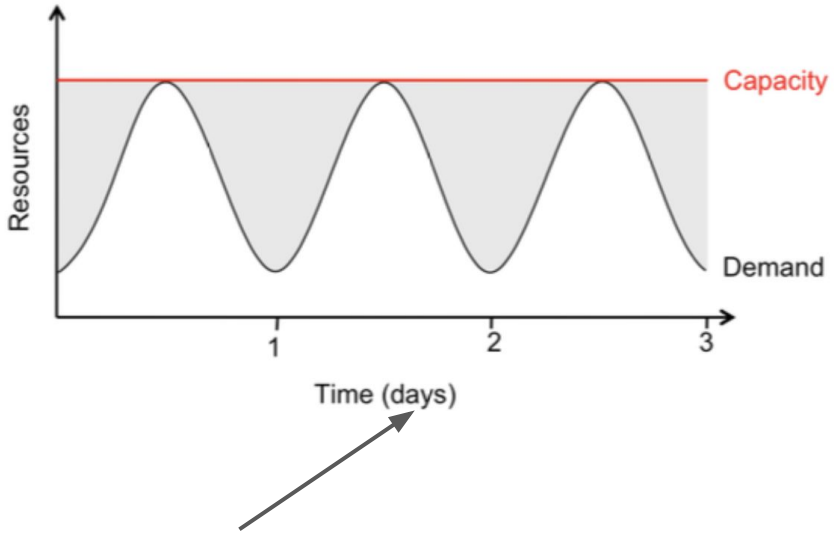
Cloud Computing

- To buy or to rent?
  - That is the question!



Cloud Computing

- Pay-as-you-go
  - Charged by service usage
    - VMs, disk, network
    - By minutes, hours, or bytes
  - No upfront cost
  - No deposit

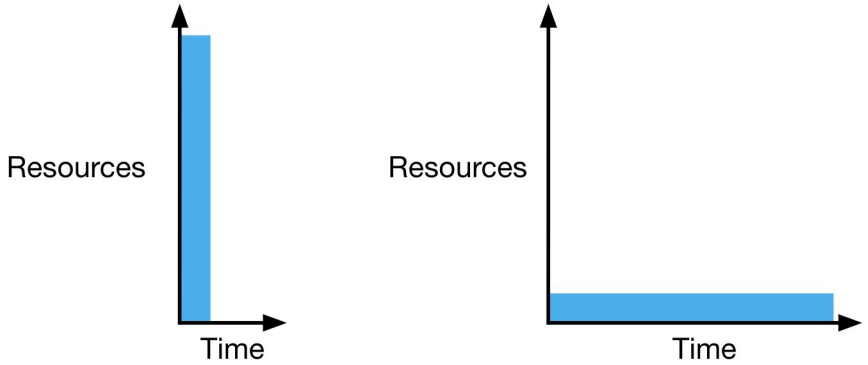


Ideal workloads

Cloud Computing

- Elasticity
  - 1000 VMs for 1 hour
  - vs. 1 VM for 1000 hours
  - Same price, 1000x faster results

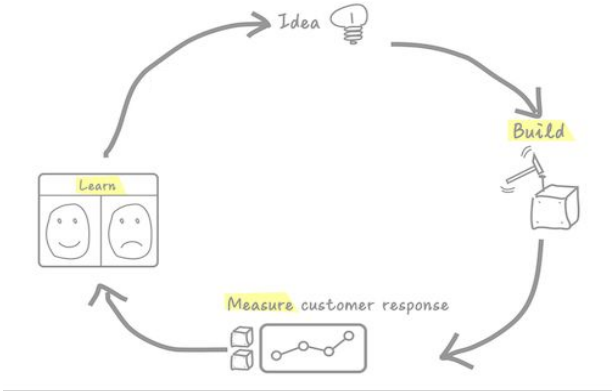
Get access to 1000s of machines in seconds



Cloud Computing

- Software as a Service
  - Faster time-to-market
  - Faster response to user
  - Faster access to state-of-the-art technologies

Faster innovation



Cloud Computing

	Buy	Rent
Utilization (using it all the time?)	100%	Vary
Flexibility (same hardware 5 years later?)	Yes	No
People cost	Yes	No
Opportunity cost (time-to-market)	High	Low
High availability	Costly	Cheap
Security	-	-

People: If you dont mind paying for engineers
to build/maintain infrastructure
If you want to ensure High Availability -> Costly

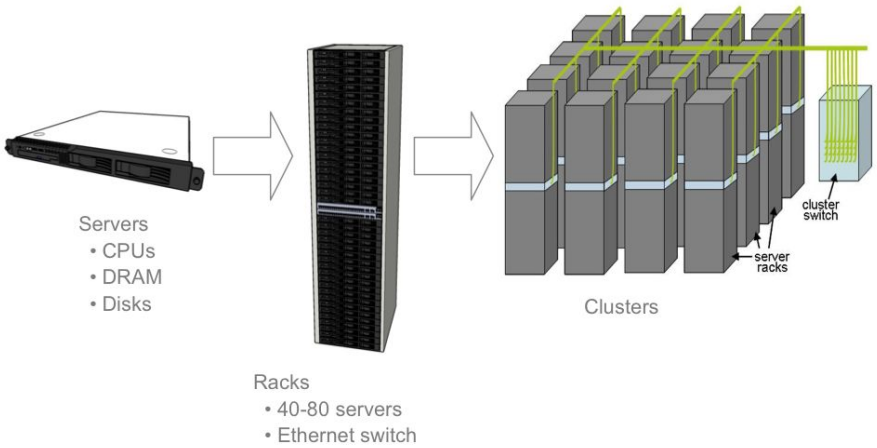
Cloud Computing

- Possible, because
  - Economies of scale
    - Cloud providers buy, manage hardware in bulk
  - Innovation in software

Cloud Computing - Systems’ View

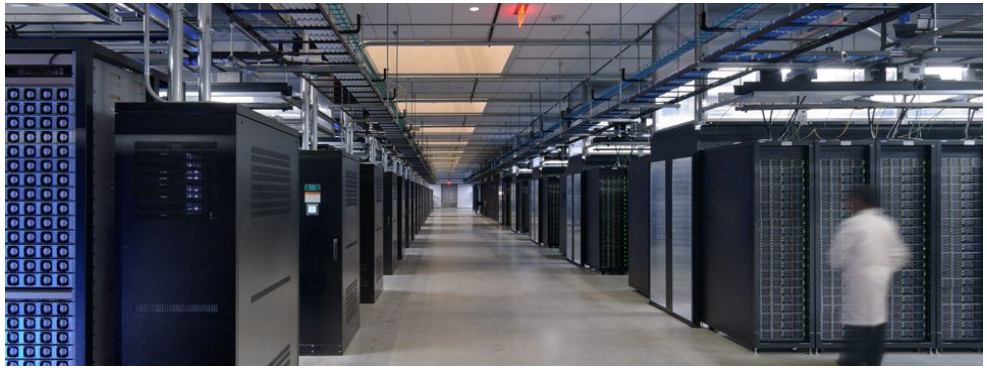
Cloud Computing

- The hardware
  - Rows and rows of rack-mounted servers
  - 1 data center ~ 100K of servers
    - 100MW (big part is cooling)
    - Few clusters per data center



Cloud Computing

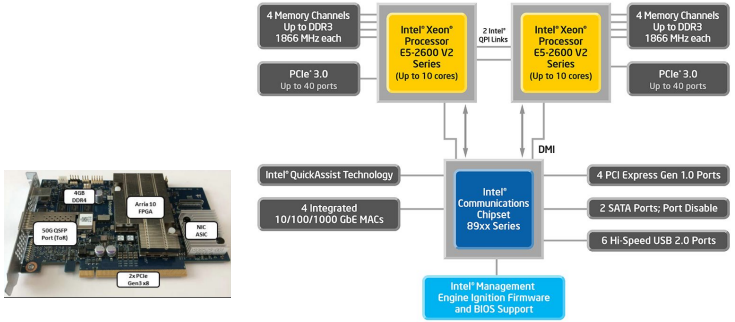
- Where your cloud is



Cloud Computing

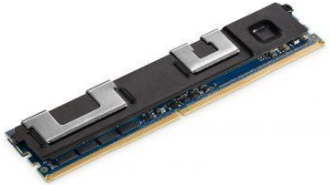
- Compute resources
  - Initially: commodity CPUs
  - Now: GPUs, ASICs

2-socket server



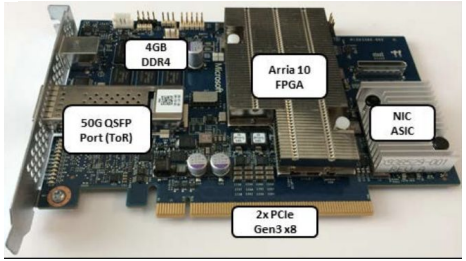
Cloud Computing

- Storage:
  - Basic: HDD, SDD
  - New: NVM, archival storage
  - Often separated from Compute:
    - NAS



Cloud Computing

- Network:
  - Basic: 10-100GB, dumb NICs
  - New: smart network
    - Programmable switches
    - Network co-processor
    - Smart NICs
    - FPGA



Cloud Computing

- Wide range of choices

Standard Systems	I Web	III Database	IV Hadoop	V Haystack	VI Feed
CPU	High 2 x E5-2670	High 2 x E5-2660	High 2 x E5-2660	Low 1 x E5-2660	High 2 x E5-2660
Memory	Low 16GB	High 144GB	Medium 64GB	Med-Hi 96GB	High 144GB
Disk	Low 250GB	High IOPS 3.2 TB Flash	High 15 x 4TB SATA	High 30 x 4TB SATA	Medium 2TB SATA + 1.6TB Flash
Services	Web, Chat	Database	Hadoop	Photos, Video	Multifeed, Search, Ads

[Facebook servers]

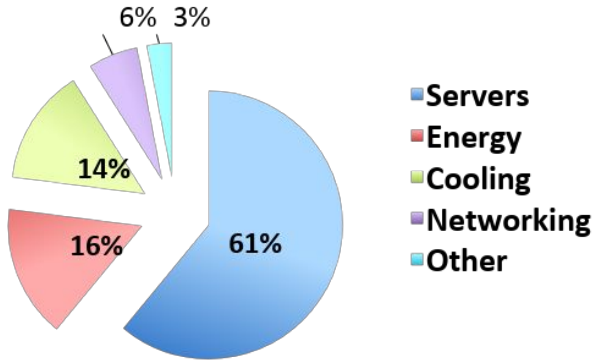
Cloud Computing

- Useful numbers, again.

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L3 cache reference	20 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Snappy	3,000 ns
Send 2K bytes over 10Ge	2,000 ns
Read 1 MB sequentially from memory	100,000 ns
Read 4KB from NVMe Flash	50,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA → Europe → CA	150,000,000 ns

Cloud Computing

- Total Cost of Ownership (TCO):
  - Capital Expense (CapEx): buildings, generators, cooling, hardware, etc.
  - Operational Expense (OpEx): electricity, maintenance, people, etc.



[Source: James Hamilton]

Major cost comes from hardware
→ better utilize them well

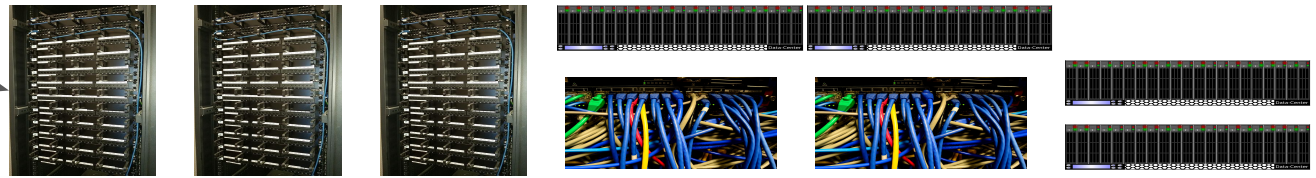
Cloud Computing

Storage | Compute | RAM | Network

Resources that users want

?? → ?

Resources that providers have



Cloud Computing

- Design goal:
  - Manage data center resources
  - Provide services to users
  - While guaranteeing:
    - High utilization
    - Competitive SLA

Many of them

Different types of services

To justify TCO

To attract more users

Cloud Computing

- Most difficult problem: failures

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hrs to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hrs)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packet loss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vIPs for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures (2-4% failure rate, machines crash at least twice)
~thousands of hard drive failures (1-5% of all disks will die)

Not to mention human errors, software bugs

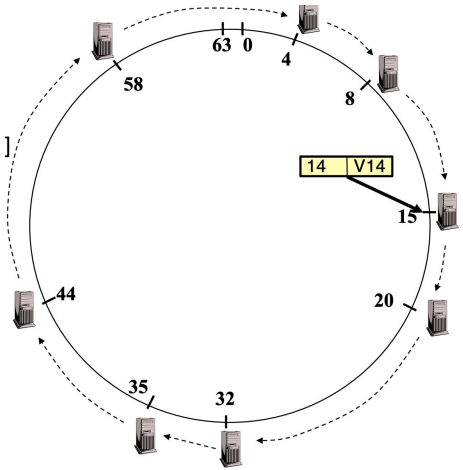
Availability/Reliability must come from software

Question: How to keep functioning under failures?

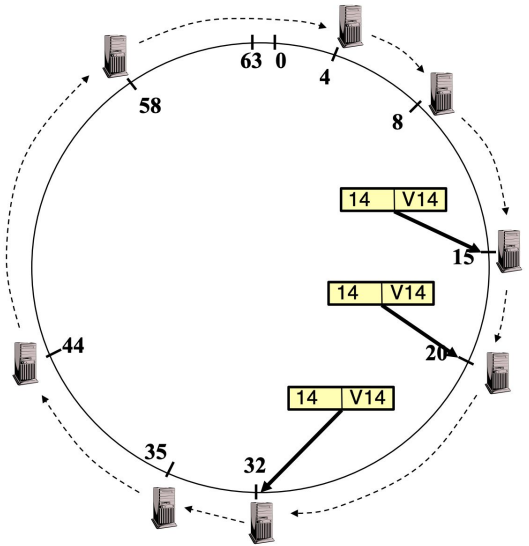
Cloud Computing

even if node 15 fails, user can go to node 20 and 32 to get it back

- How to deal with failures:
  - Redundancy redundancy redundancy
  - Example

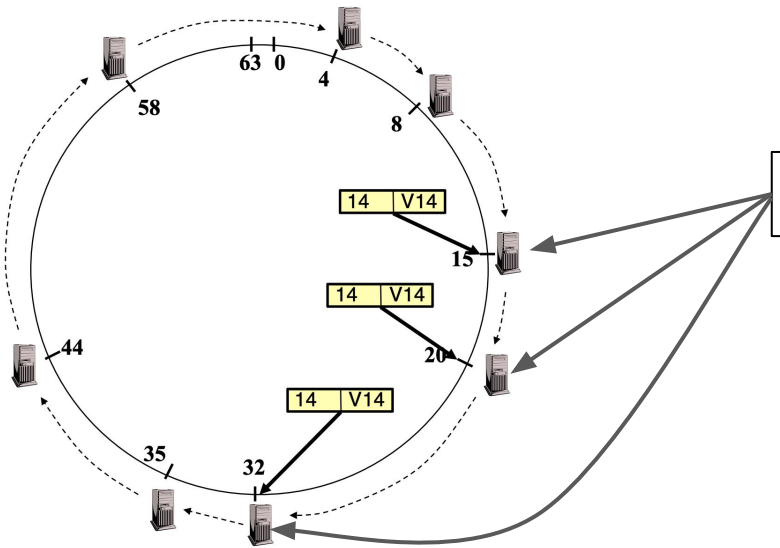


3-way replications



Cloud Computing

- Replication



insert(14, v14)

Example

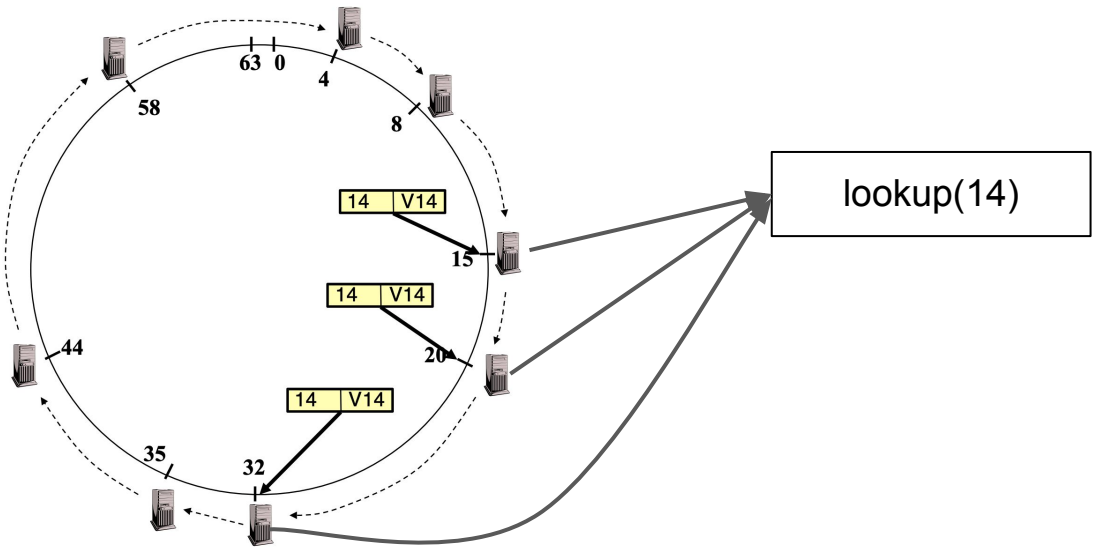
R(k): set of nodes storing k (replicas)

Insert(k,v):
Write to any node in R(k)
Return when any replica returns

Lookup(k):
Read from any node in R(k)
Return the first result

Cloud Computing

- Replication



Example

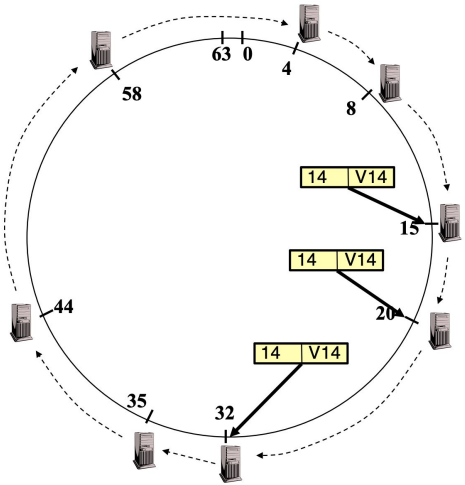
R(k): set of nodes storing k (replicas)

Insert(k,v):
Write to any node in R(k)
Return when any replica returns

Lookup(k):
Read from any node in R(k)
Return the first result

Cloud Computing

- Replication



Eventual Consistency
-> insert v14 to node 15
-> then node 15 try to replicate node 20 and 32
-> but before this replication finish, user read from node 32/20 which is the stale value, not the latest value.

-> read **does not return the latest** write

Example

R(k): set of nodes storing k (replicas)

Insert(k,v):
Write to any node in R(k)
Return when any replica returns

Lookup(k):
Read from any node in R(k)
Return the first result

Remember what problems here?

Cloud Computing

- CAP theorem: in any distributed system, choose 2
  - **C**onsistency:
    - Read returns the latest write
  - **A**vailability:
    - Every request receives a response
  - **P**artition Tolerance:
    - Continue to work when network fails

Example: Amazon DynamoDB chooses **AP**

give up on consistency

Question: How to scale?

Question: How to maximize utilization?

Question: How to isolate users?

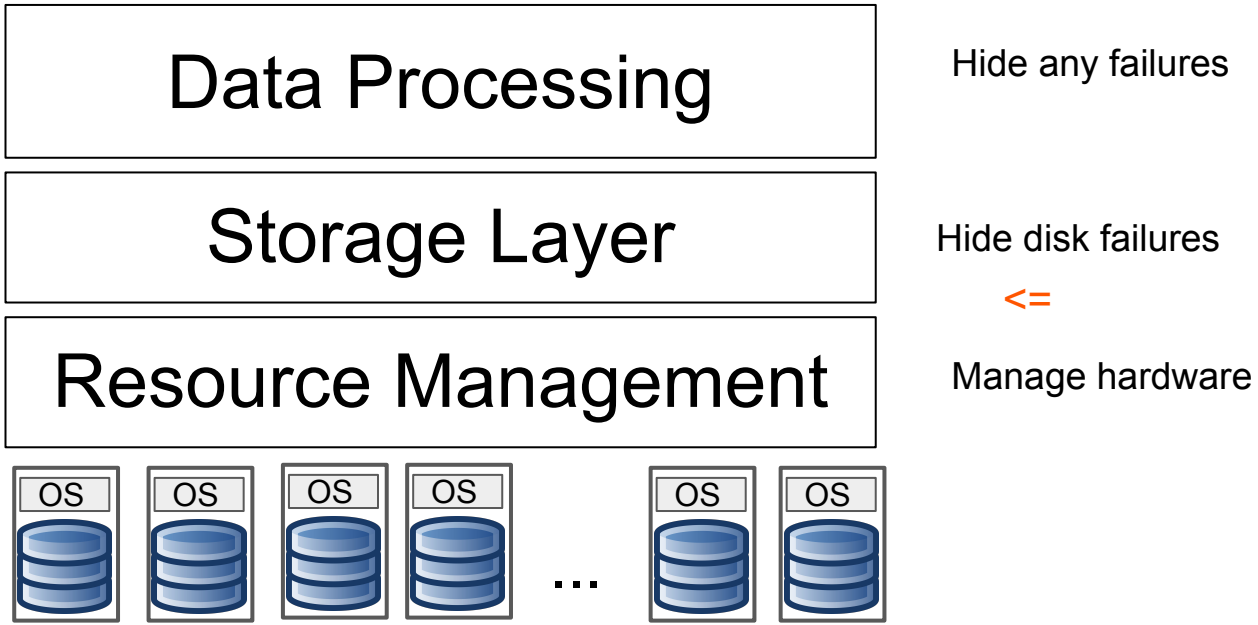
Question: How to monitor?

….

Cloud Computing

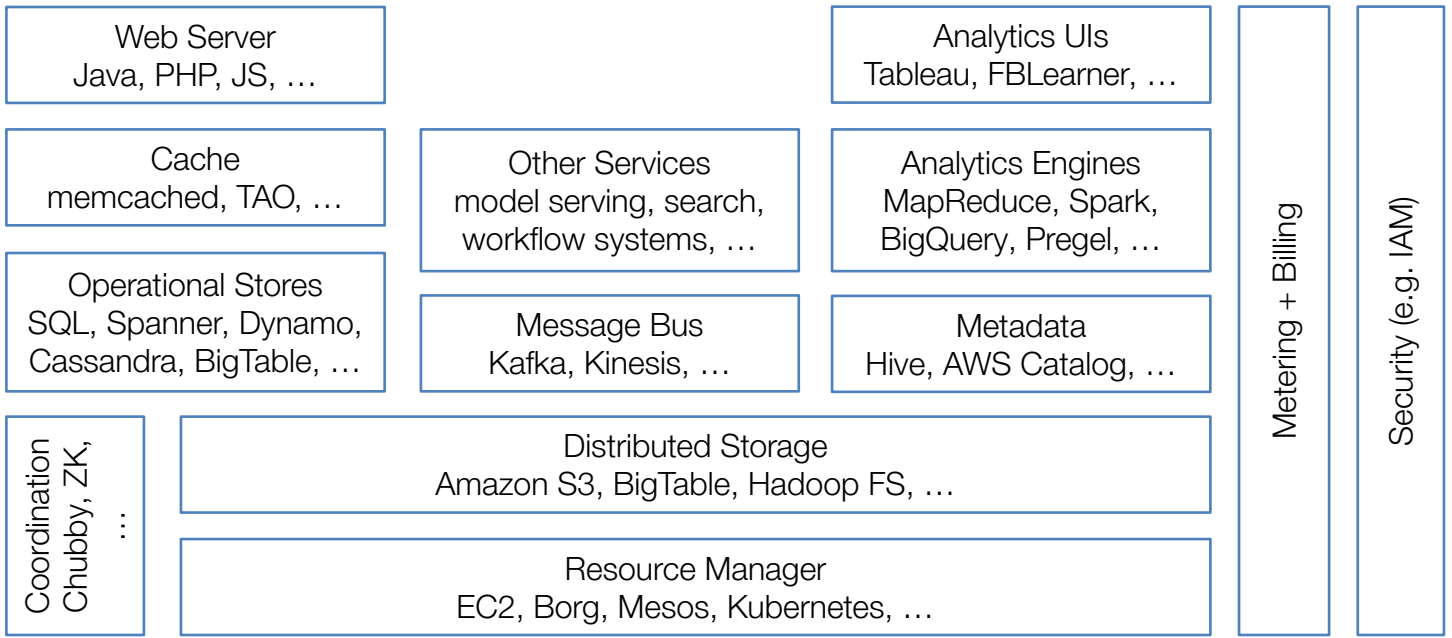
- High-level stack

Storage Layer:
hide complexity of disk
(replicate data even if one disk fails, still can retrieve data required)



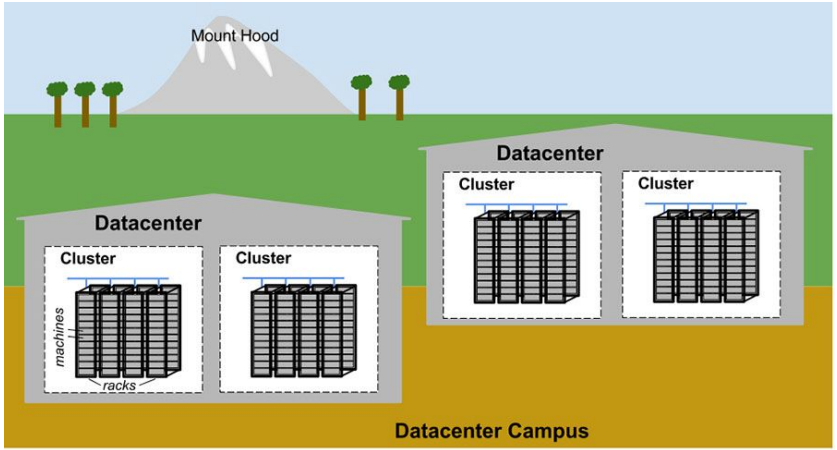
Cloud Computing

- Cloud software stack in practice



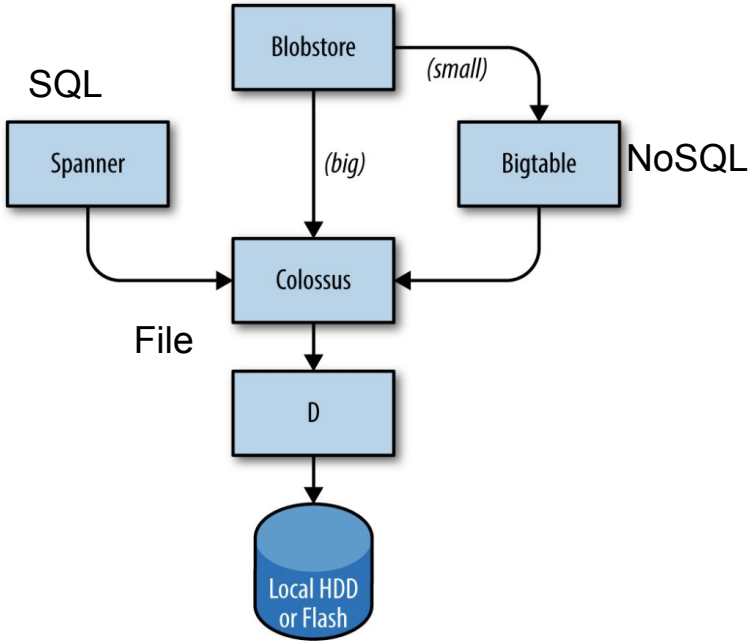
Google Private Cloud

Google Cloud

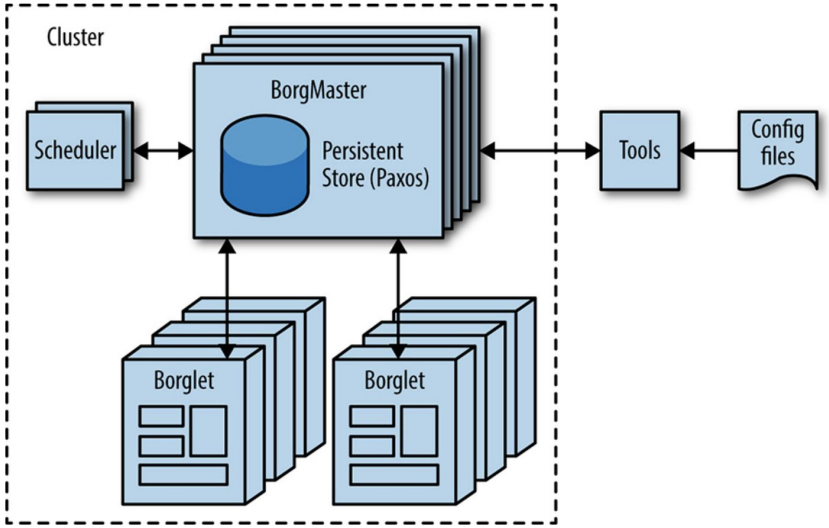


Google Cloud

Storage layer

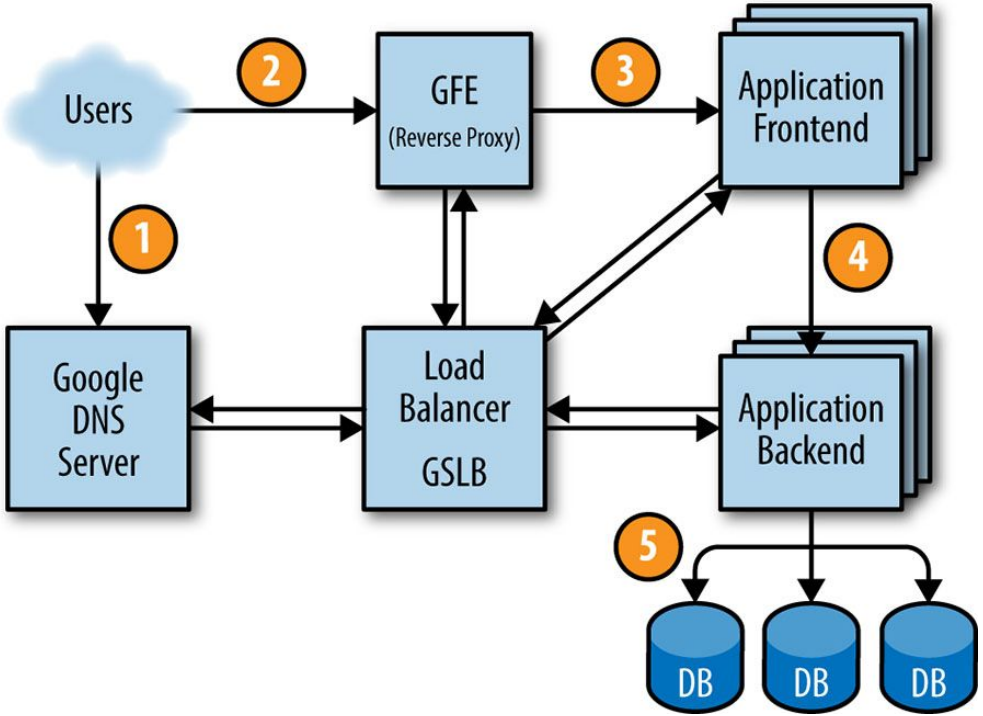


Resource
management layer



Google Cloud

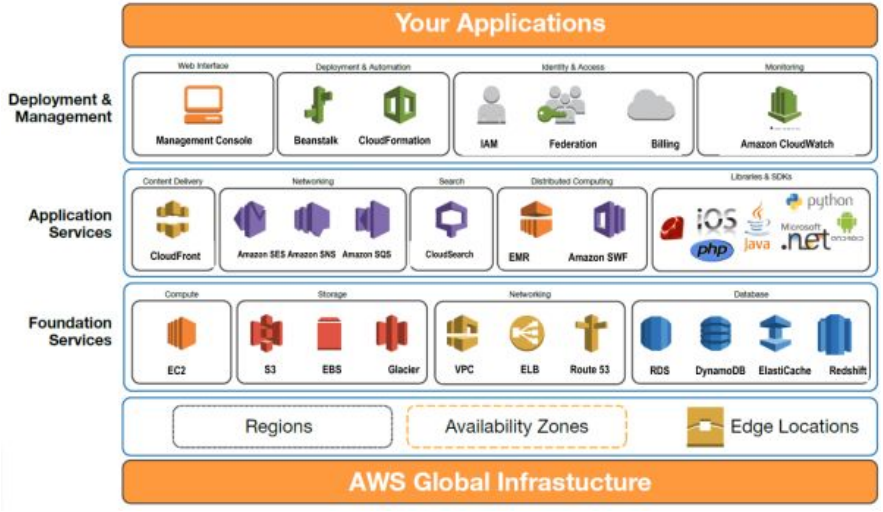
- Life of a request

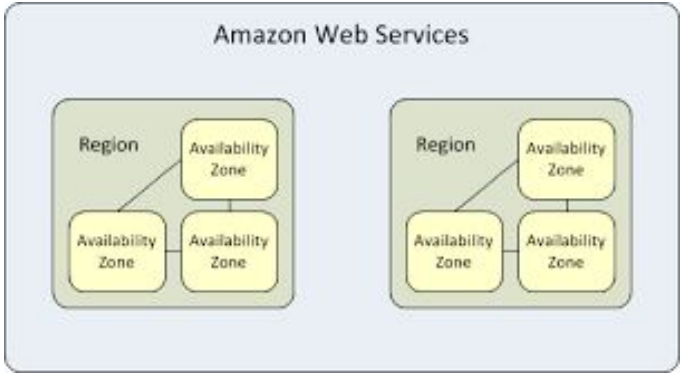


Amazon Web Service

AWS

- Most popular
  - The earliest
- Only cover key concepts here
  - You have $100 to play with
- **Region**
  - Where servers are
  - Geographical region, e.g. ap-southeast-1
  - Multiple **availability zones** per regions, e.g. ap-southeast-1a



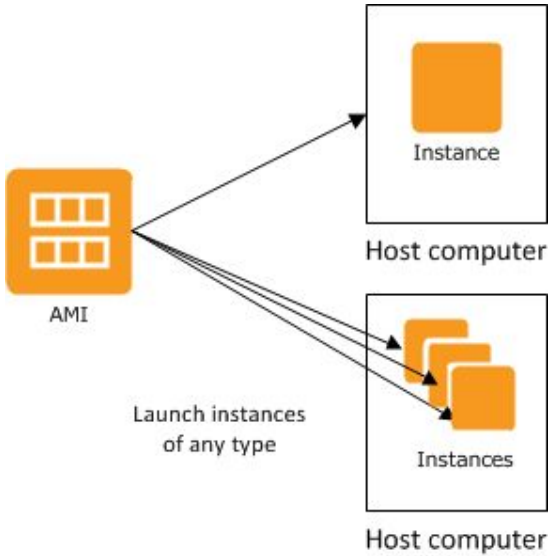


AWS

- Many services, but this course:
  - Elastic Cloud Computing (EC2)
  - Simple Storage Service (S3)
- We've seen S3
  - Simple bucket-based key-value store (Week 3)
  - Dynamo design (this week)

AWS

- EC2
  - Amazon Machine Image (**AIM**): like a virtual machine image
  - **Instance**: running image on specific hardware
    - Like a virtual machine
  - Instance **type**: what hardware

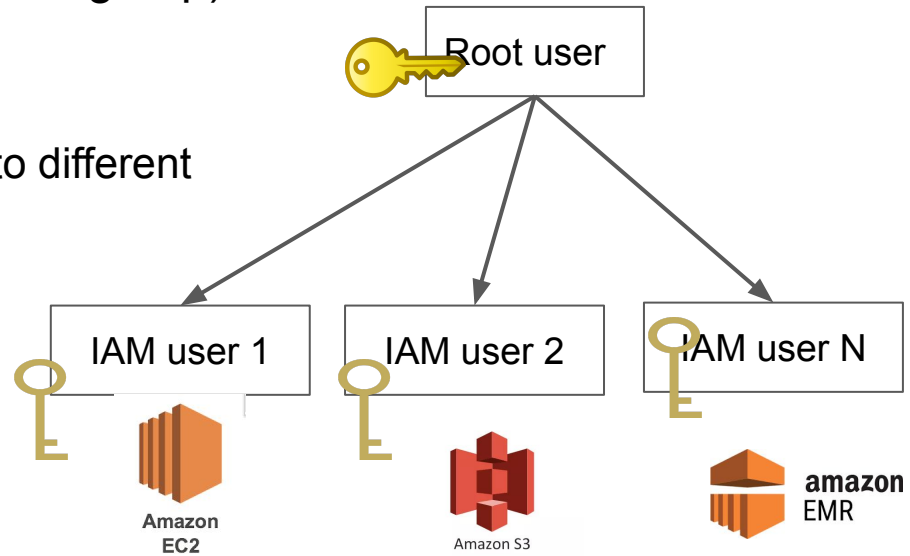


Instance	vCPU*	CPU Credits / hour	Mem (GiB)	Storage	Network Performance
t2.nano	1	3	0.5	EBS-Only	Low
t2.micro	1	6	1	EBS-Only	Low to Moderate
t2.small	1	12	2	EBS-Only	Low to Moderate
t2.medium	2	24	4	EBS-Only	Low to Moderate

Free*

AWS

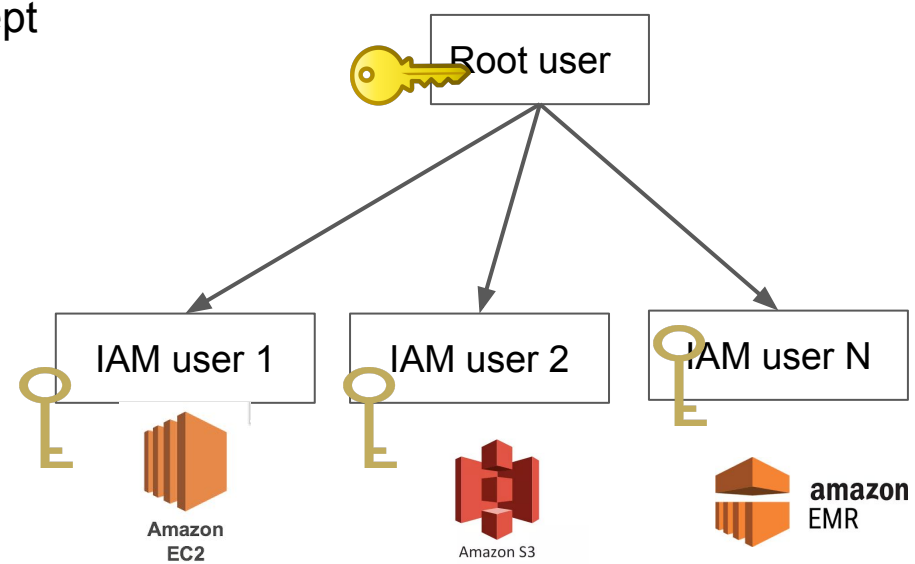
- Identity and Access Management (IAM):
  - Your account has 1 root user (when sign up)
  - Root user can:
    - Create **IAM users**
    - Grant different IAM user access to different services



Why all these troubles?

AWS

- 2 types of credentials
  - **Access key**:
    - Short, 40-character string
    - To invoke AWS APIs, must be kept secret.
  - **SSH key**:
    - Standard SSH key
    - To log-in to the EC2 instances



AWS

- Security group
  - Firewall: rules for network access
  - Each instance belongs to one security group
  - You can create many groups
  - Default option: **publicly available**

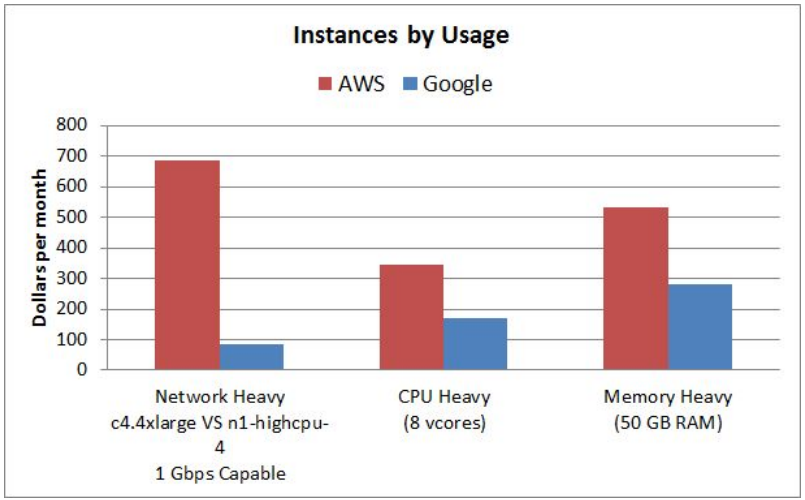
AWS

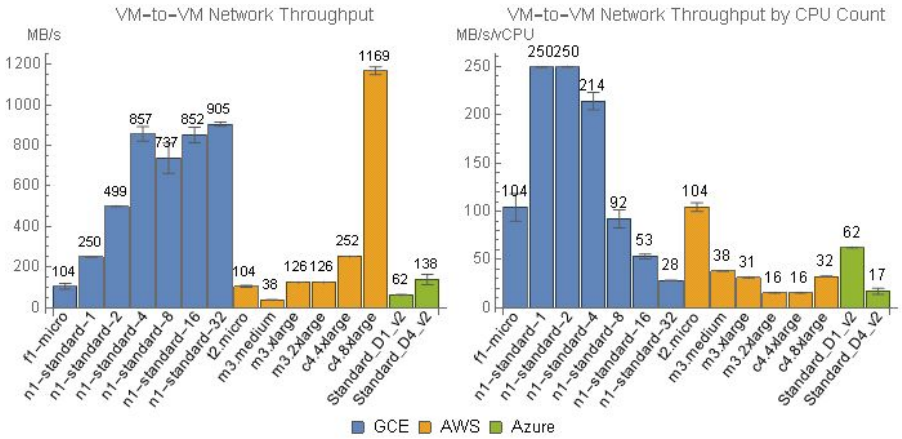
- How to interact with AWS:
  - AWS GUI console
    - Doesn’t scale beyond a handful of instances
  - This course:
    - Use command lines
    - Or boto3 (Python)

Comparison

Microsoft AZURE	Amazon EC2	Google Cloud Engine
Basic tier: A0, A1, A2, A3, A4 Optimized Compute : D1, D2, D3, D4, D11, D12, D13 D1v2, D2v2, D3v2, D11v2,… Latest CPUs: G1, G2, G3, … Network Optimized: A8, A9 Compute Intensive: A10, A11,…	t2.nano, t2.micro, t2.small m4.large, m4.xlarge, m4.2xlarge, m4.4xlarge, m3.medium, c4.large, c4.xlarge, c4.2xlarge, c3.large, c3.xlarge, c3.4xlarge, r3.large, r3.xlarge, r3.4xlarge, i2.2xlarge, i2.4xlarge, d2.xlarge d2.2xlarge, d2.4xlarge,…	n1-standard-1, ns1-standard-2, ns1-standard-4, ns1-standard-8, ns1-standard-16, ns1highmem-2, ns1-highmem-4, ns1-highmem-8, n1-highcpu-2, n1-highcpu-4, n1-highcpu-8, n1-highcpu-16, n1-highcpu-32, f1-micro, g1-small…

Comparison


Instances by Usage
AWS
Google
Dollars per month
800
700
600
500
400
300
200
100
0
Network Heavy
c4.4xlarge VS n1-highcpu-4
1 Gbps Capable
CPU Heavy
(8 vcores)
Memory Heavy
(50 GB RAM)


VM-to-VM Network Throughput
MB/s
1200
1000
800
600
400
200
0
104
250
499
857
737
852
905
104
38
126
126
252
1169
62
138
f1-micro
n1-standard-1
n1-standard-2
n1-standard-4
n1-standard-8
n1-standard-16
n1-standard-32
t2.micro
m3.medium
m3.xlarge
m3.2xlarge
c4.4xlarge
c4.8xlarge
Standard_D1_v2
Standard_D4_v2
VM-to-VM Network Throughput by CPU Count
MB/s/vCPU
250
200
150
100
50
0
104
250250
214
92
53
28
104
38
31
16
16
32
62
17
GCE
AWS
Azure

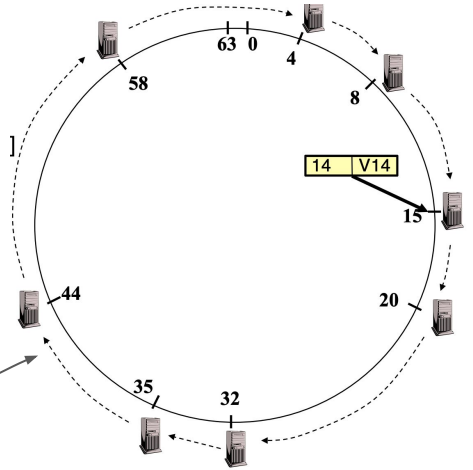
Summary

- Cloud computing levels the playing field
  - At least for Big Data
- Cloud computing not merely buying hardware in bulk
  - Many technical challenges
- From now on, whenever we speak about big data
  - Assume that it’s on the cloud

Question 1

If you add a new machine,
Need to disrupt the distribution a little bit. And accept the fact that it will not be **evenly** distributed

- Consider Dynamo
  - Suppose the hashed keys are uniformly distributed in the ring space
  - How do you guarantee that servers have equal shares of the key space?
    - In this figure, they do not



Given a **fixed** number of machines,
- **Master** server can assign machine ids to be **evenly** distributed across the nodes
- Since keys are evenly distributed, all nodes will have **equal** share of key space

Question 2

- Following up from Q1:
  - Suppose the hashed keys are uniformly distributed in the ring space
  - Suppose servers have equal shares of the key space
  - Does the system guarantee that every server handles the same amount of work (load balanced)?

No,
- Input could have a certain distribution and the hash output follows a certain distribution.
- The read operations may be **heavily skewed**, some keys may be very popular and queried a lot -> all go to one server which handles a **disproportionately** amount of work.

- If a few keys are very popular
- **replicate** them -> send to a few servers
- or redistribute them to some other nodes