M 120: DISTRIBUTED SYSTEMS

Clouds and the CAP Theorem

*Slides include material provided by Indranil (Indy) Gupta and Ken Birman

The Hype!

- Forrester in 2010 – Cloud computing will go from $40.7 billion in 2010 to $241 billion in 2020.
- Goldman Sachs predicted cloud computing will grow at annual rate of 30% from 2013-2018
- Hadoop market to reach $20.8 B by by 2018: Transparency Market Research
- Companies and even Federal/state governments using cloud computing now: fbo.gov


Google App Engine
amazon web services
RightScale
ORACLE
GridGain 2.0
Elastic Server
VirtualBox
virtualLogix
vmware
force.com platform as a service
CiTRIX
OpenVZ
10gen
KVM
Microsoft
parascale Powering Cloud Storage
hp
Xen
GIGASPACES
Parallels Optimized Computing
RightScale
enomaly
elastra
ORACLE
BUNGEEconnect
PowerVM
Virtualiron

Many Cloud Providers

- AWS: Amazon Web Services
  - EC2: Elastic Compute Cloud
  - S3: Simple Storage Service
  - EBS: Elastic Block Storage
- Microsoft Azure
- Google Cloud/Compute Engine/AppEngine
- Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera
- And many many more!

Two Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- Private clouds are accessible only to company employees
- Public clouds provide service to any paying customer:
  - Amazon S3 (Simple Storage Service): store arbitrary datasets, pay per GB-month stored
    - As of 2019: 0.4c-3 c per GB month
  - Amazon EC2 (Elastic Compute Cloud): upload and run arbitrary OS images, pay per CPU hour used
    - As of 2019: 0.2 c per CPU hr to $7.2 per CPU hr (depending on strength)
  - Google cloud: similar pricing as above
  - Google AppEngine/Compute Engine: develop applications within their appengine framework, upload data that will be imported into their format, and run

Customers Save Time and $$$

- Dave Power, Associate Information Consultant at Eli Lilly and Company: “With AWS, Powers said, a new server can be up and running in **three minutes** (it used to take Eli Lilly **seven and a half weeks** to deploy a server internally) and a **64-node Linux cluster** can be online in five minutes (compared with three months internally). … It's just shy of instantaneous.”

- Ingo Elfering, Vice President of Information Technology Strategy, GlaxoSmithKline: “With Online Services, we are able to reduce our IT **operational costs** by roughly **30%** of what we’re spending”

- Jim Swartz, CIO, Sybase: “At Sybase, a private cloud of virtual servers inside its datacenter has saved nearly **$US2 million annually** since 2006, Swartz says, because the company can share computing power and storage resources across servers.”

- 100s of startups in Silicon Valley can harness large computing resources without buying their own machines.

But what exactly IS a cloud?

Cloud = a fancy word for a distributed system

- A "cloud" is the latest nickname for a distributed system
- Previous nicknames for "distributed systems" have included
  - Peer-to-peer systems
  - Grids
  - Clusters
  - Timeshared computers (from the 60s and 70s)

Distributed Systems Rant

- Nicknames come and go, but the core concepts underlying distributed systems stay the same
  - And they are used decade after decade
    - E.g. Lamport Time stamps were invented in the 1970s and they are used in almost all distributed/cloud systems
    - This course is about these distributed systems concepts
- A few years from now, there may be a new nickname for distributed systems
  - The core concepts will remain the same and they will continue to be used in real systems

What is a Cloud?

- It's a cluster!
- It's a supercomputer!
- It's a datastore!
- It's superman!

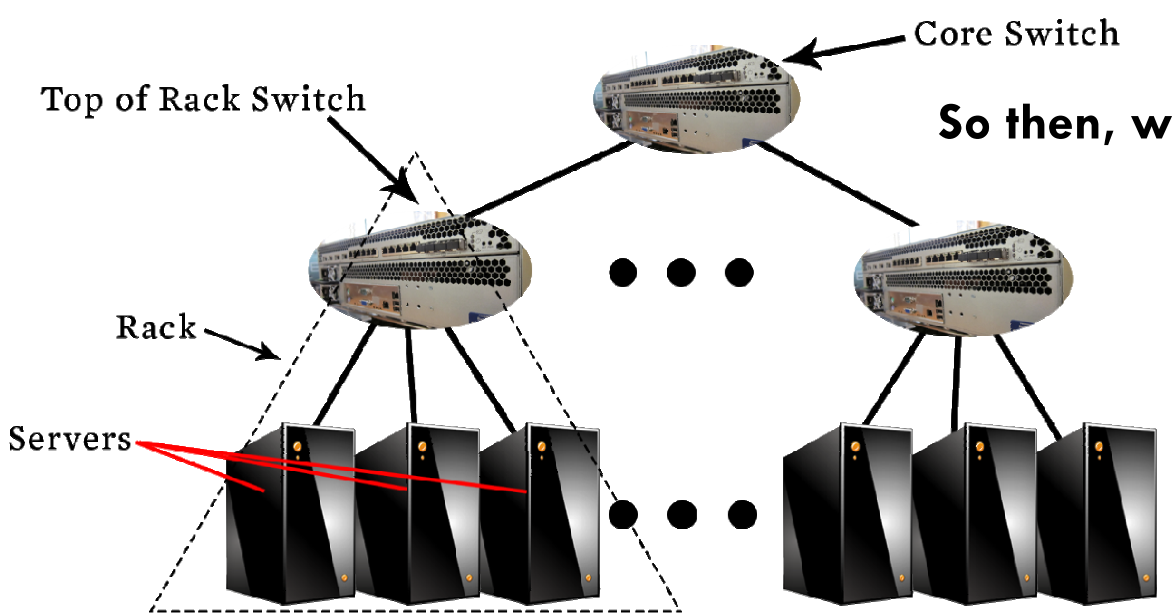
- None of the above
- All of the above

- Cloud working definition = Lots of storage + compute cycles nearby

What is a Cloud?

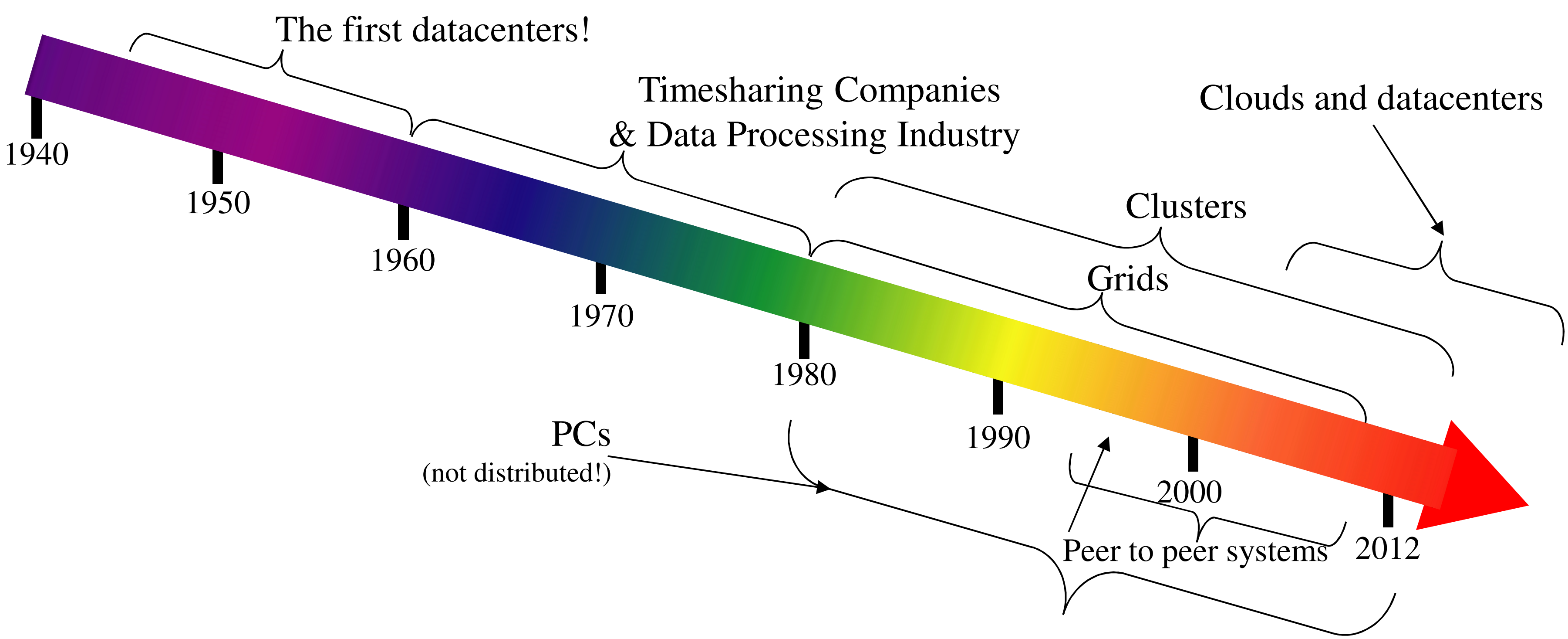
- A single-site cloud (aka "Datacenter") consists of
  - Compute nodes (grouped into racks) (2)
  - Switches, connecting the racks
  - A network topology, e.g., hierarchical
  - Storage (backend) nodes connected to the network (3)
  - Front-end for submitting jobs and receiving client requests (1)
  - (1-3: Often called "three-tier architecture")
  - Software Services
- A geographically distributed cloud consists of
  - Multiple such sites
  - Each site perhaps with a different structure and services

A Sample Cloud Topology

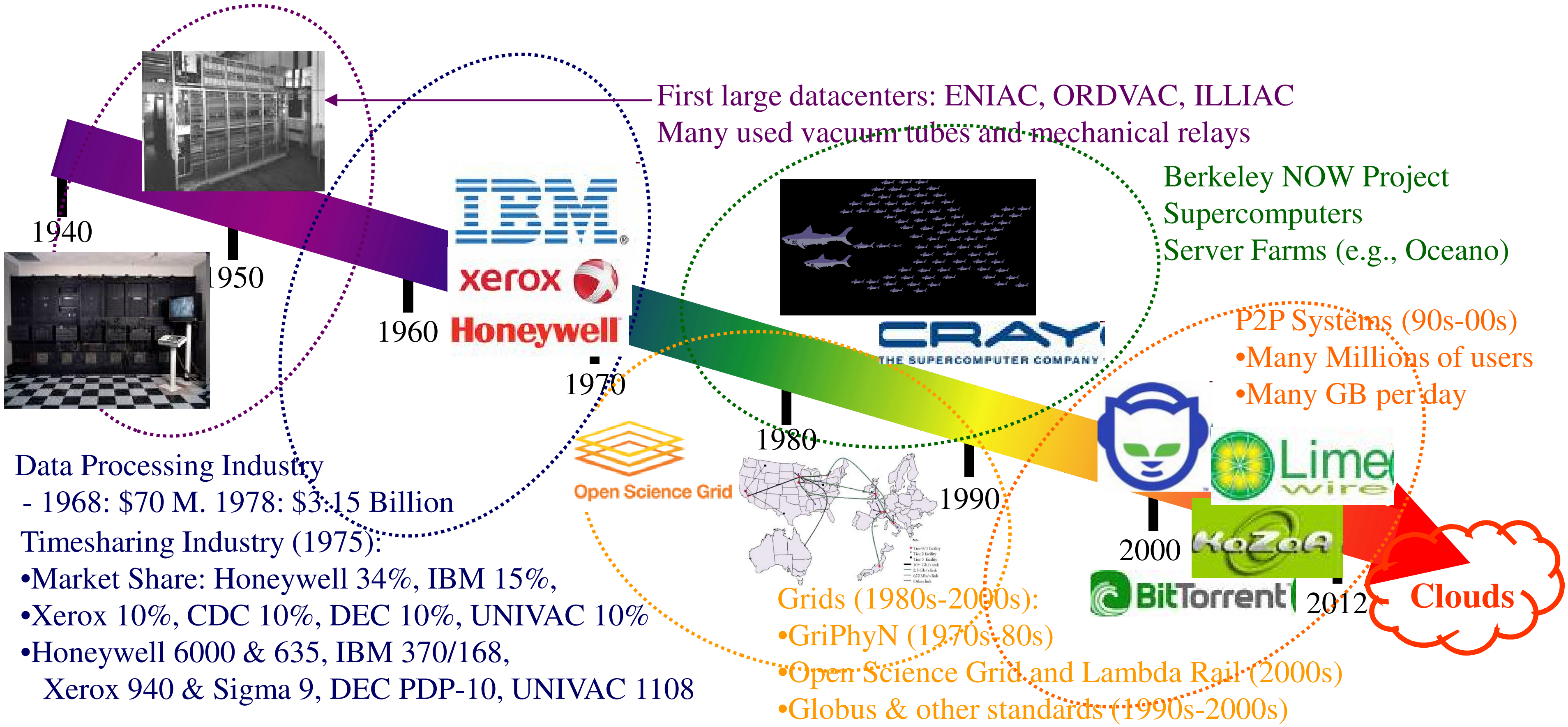


So then, what is a cluster?

"A Cloudy History of Time"



"A Cloudy History of Time"



Trends: Technology

- Doubling Periods – storage: 12 mos, bandwidth: 9 mos, and (what law is this?) cpu compute capacity: 18 mos
- Then and Now
  - Bandwidth
    - 1985: mostly 56Kbps links nationwide
    - 2015: Tbps links widespread
  - Disk capacity
    - Today's PCs have TBs, far more than a 1990 supercomputer

Trends: Users

- Then and Now
  - Biologists in 1990: were running small single-molecule simulations, today run large-scale genome sequencing experiments on hundreds of machines
  - Physicists today: CERN's Large Hadron Collider producing many PB/year

Prophecies

- In 1965, MIT's Fernando Corbató and the other designers of the Multics operating system envisioned a computer facility operating "like a power company or water company".

- Plug your thin client into the computing Utility and Play your favorite Intensive Compute & Communicate Application
  - Have today's clouds brought us closer to this reality? Think about it.

Four Features New in Today's Clouds

- Massive scale.
- On-demand access: Pay-as-you-go, no upfront commitment.
  - And anyone can access it
- Data-intensive Nature: What was MBs has now become TBs, PBs and XBs.
  - Daily logs, forensics, Web data, etc.
  - Humans have data numbness: Wikipedia (large) compressed is only about 10 GB!
- New Cloud Programming Paradigms: MapReduce/Hadoop, NoSQL/Cassandra/MongoDB and many others.
  - High in accessibility and ease of programmability
  - Lots of open-source

Combination of one or more of these gives rise to novel and unsolved distributed computing problems in cloud computing.

I. Massive Scale

- Facebook [GigaOm, 2012]
  - 30K in 2009 -> 60K in 2010 -> 180K in 2012
- Microsoft [NYTimes, 2008]
  - 150K machines (80K just for running Bing)
  - Growth rate of 10K per month
  - In 2013, Microsoft Cosmos had 110K machines (4 sites)
- Yahoo! [2009]:
  - 100K, split into clusters of 4000
- AWS EC2 [Randy Bias, 2009]
  - 40K machines, 8 cores/machine
- eBay [2012]: 50K machines
- HP [2012]: 380K in 180 DCs
- Google [2011, Data Center Knowledge] : 900K

Quiz: Where is the World's Largest Datacenter?

Quiz: Where is the World's Largest Datacenter?

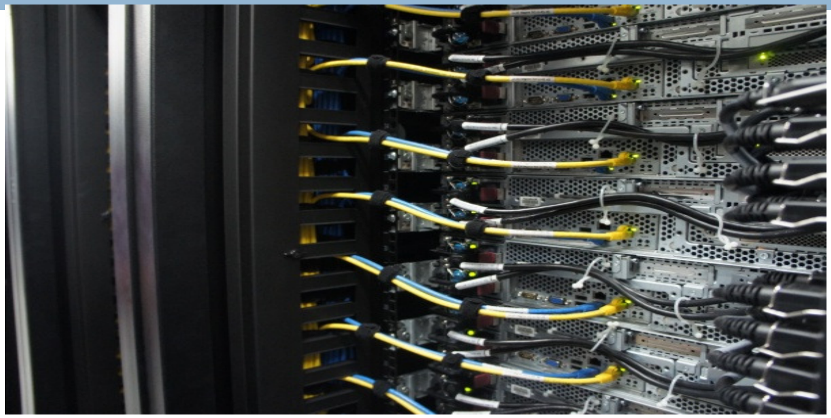
- (2018) China Telecom. 10.7 Million sq. ft.
- (2017) "The Citadel" Nevada. 7.2 Million sq. ft.
- (2015) In Chicago!
  - 350 East Cermak, Chicago, 1.1 MILLION sq. ft.
  - Shared by many different "carriers"
  - Critical to Chicago Mercantile Exchange
- See:
  - https://www.gigabitmagazine.com/top10/top-10-biggest-data-centres-world
  - https://www.racksolutions.com/news/data-center-news/top-10-largest-data-centers-world/

What does a datacenter look like from inside?

- A virtual walk through a datacenter
- Reference: http://gigaom.com/cleantech/a-rare-look-inside-facebooks-oregon-data-center-photos-video/

Servers

Front

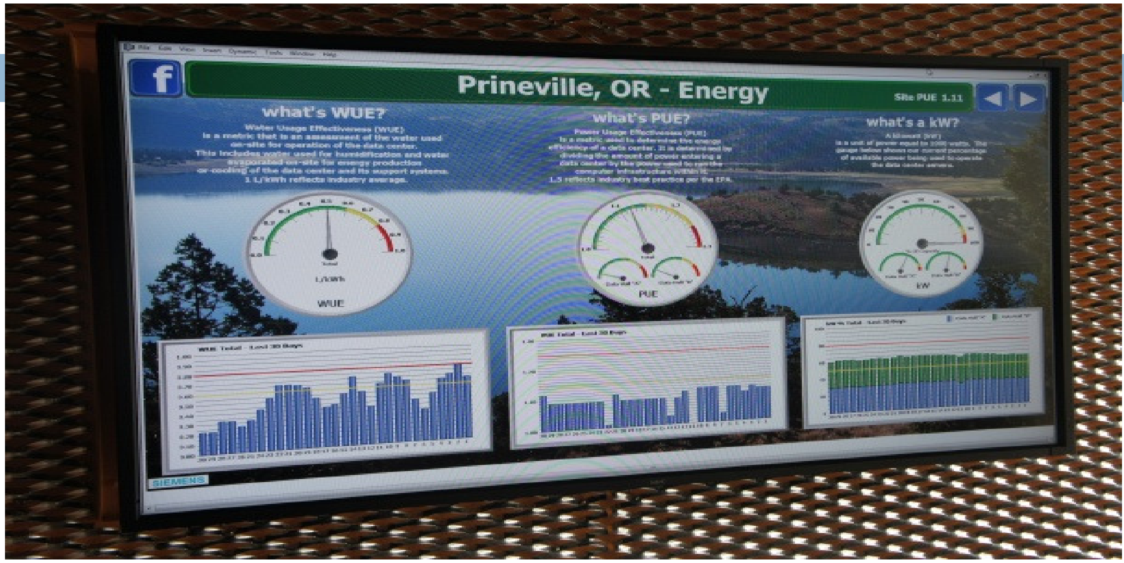


Back

In

Some highly secure (e.g., financial info)

Power



Off-site

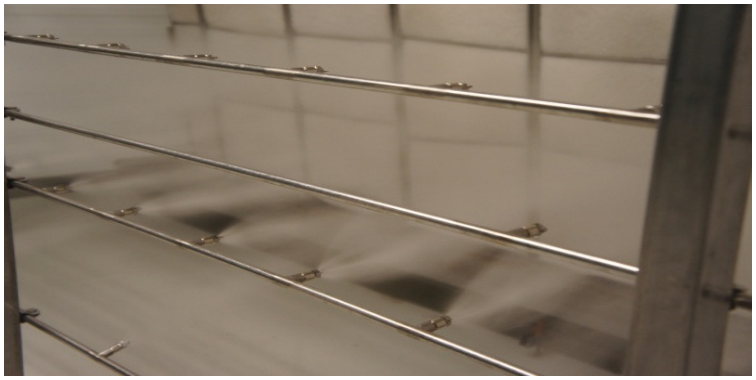
On-site

- WUE = Annual Water Usage / IT Equipment Energy (L/kWh) – low is good
- PUE = Total facility Power / IT Equipment Power – low is good (e.g., Google~1.1)

Cooling

Air sucked in from top (also, Bugzappers)

Water purified



Water sprayed into air

15 motors per server bank

Extra - Fun Videos to Watch

- Microsoft GFS Datacenter Tour (Youtube)
  - https://www.youtube.com/watch?v=KG4cE362ETE

- Timelapse of a Datacenter Construction on the Inside (Fortune 500 company)
  - http://www.youtube.com/watch?v=ujO-xNvXj3g

II. On-demand access: *aaS Classification

- On-demand: renting a cab vs. (previously) renting a car, or buying one. E.g.:
  - AWS Elastic Compute Cloud (EC2): a few cents to a few $ per CPU hour
  - AWS Simple Storage Service (S3): a few cents 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 (cgroups, Kubernetes, Dockers, VMs,…). Often said to subsume HaaS.
  - Ex: Amazon Web Services (AWS: EC2 and S3), OpenStack, Eucalyptus, Rightscale, Microsoft Azure, Google Cloud.

II. On-demand access: *aaS Classification

- PaaS: Platform as a Service
  - You get access to flexible computing and storage infrastructure, coupled with a software platform (often tightly coupled)
  - 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 365 Online

III. Data-intensive Computing

- Computation-Intensive Computing
  - Example areas: MPI-based, High-performance computing, Grids
  - Typically run on supercomputers (e.g., NCSA Blue Waters)
  - Often working on relatively small amount of data but running intense computations on it
- 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 several MapReduce jobs
    - 300 TB of data, 10K cores, many tens of hours (~2008)

IV. New Cloud Programming Paradigms (cont'd)

- Easy to write and run highly parallel programs in new cloud programming paradigms:
  - 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!

Two Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- Private clouds are accessible only to company employees
- Public clouds provide service to any paying customer

- You’re starting a new service/company: should you use a public cloud or purchase your own private cloud?

Single site Cloud: to Outsource or Own?

- Medium-sized organization: wishes to run a service for *M* months
  - Service requires 128 servers (1024 cores) and 524 TB
- Outsource (e.g., via AWS): *monthly* cost
  - S3 costs: $0.12 per GB month. EC2 costs: $0.10 per CPU hour (costs from 2009)
  - Storage = $ 0.12 X 524 X 1000 ~ $62 K
  - Total = Storage + CPUs = $62 K + $0.10 X 1024 X 24 X 30 ~ $136 K
- Own: monthly cost
  - Storage ~ $349 K / *M*
  - Total ~ $ 1555 K / *M* + 7.5 K (includes 1 sysadmin / 100 nodes)
    - using 0.45:0.4:0.15 split for hardware:power:network and 3 year lifetime of hardware

Single site Cloud: to Outsource or Own?

- Breakeven analysis: more preferable to own if:
  - $349 K / *M* < $62 K (storage)
  - $ 1555 K / *M* + 7.5 K < $136 K (overall)

  Breakeven points
  - *M* > 5.55 months (storage)
  - *M* > 12 months (overall)

- As a result
  - Startups use clouds a lot
  - Cloud providers benefit monetarily most from storage

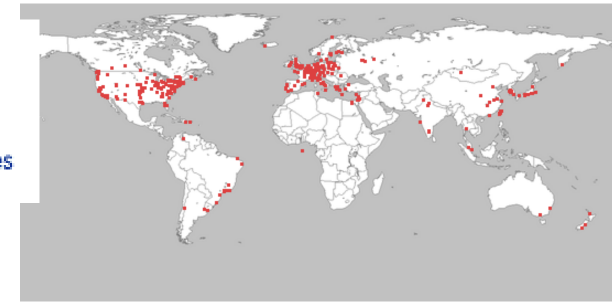
Academic Clouds: Emulab

- A community resource open to researchers in academia and industry. Very widely used by researchers everywhere today.
- https://www.emulab.net/
- A cluster, with currently ~500 servers
- Founded and owned by University of Utah (led by Late Prof. Jay Lepreau)

- As a user, you can:
  - Grab a set of machines for your experiment
  - You get root-level (sudo) access to these machines
  - You can specify a network topology for your cluster
  - You can emulate any topology

PLANETLAB

An open platform for developing, deploying, and accessing planetary-scale services



- A community resource open to researchers

- http://www.planet-lab.org/
- Currently, ~ 1077 nodes at ~500 sites across the world
- Founded at Princeton University (led by Prof. Larry Peterson), but owned in a federated manner by the sites
- Node: Dedicated server that runs components of PlanetLab services.
- Site: A location, e.g., UoA that hosts a number of nodes.
- Sliver: Virtual division of each node. Currently, uses VMs, but it could also other technology. Needed for timesharing across users.
- Slice: A spatial cut-up of the PL nodes. Per user. A slice is a way of giving each user (Unix-shell like) access to a subset of PL machines, selected by the user. A slice consists of multiple slivers, one at each component node.
- Thus, PlanetLab allows you to run real world-wide experiments.
- Many services have been deployed atop it, used by millions (not just researchers): Application-level DNS services, Monitoring services, CoralCDN, etc.
- PlanetLab is basis for NSF GENI https://www.geni.net/

Public Research Clouds

- Accessible to researchers with a qualifying grant
- Chameleon Cloud: https://www.chameleoncloud.org/
  - HaaS
  - OpenStack (~AWS)
- CloudLab: https://www.cloudlab.us/
  - Build your own cloud on their hardware

Summary

- Clouds build on many previous generations of distributed systems
- Especially the timesharing and data processing industry of the 1960-70s.
- Need to identify unique aspects of a problem to classify it as a new cloud computing problem
  - Scale, On-demand access, data-intensive, new programming
- Otherwise, the solutions to your problem may already exist!

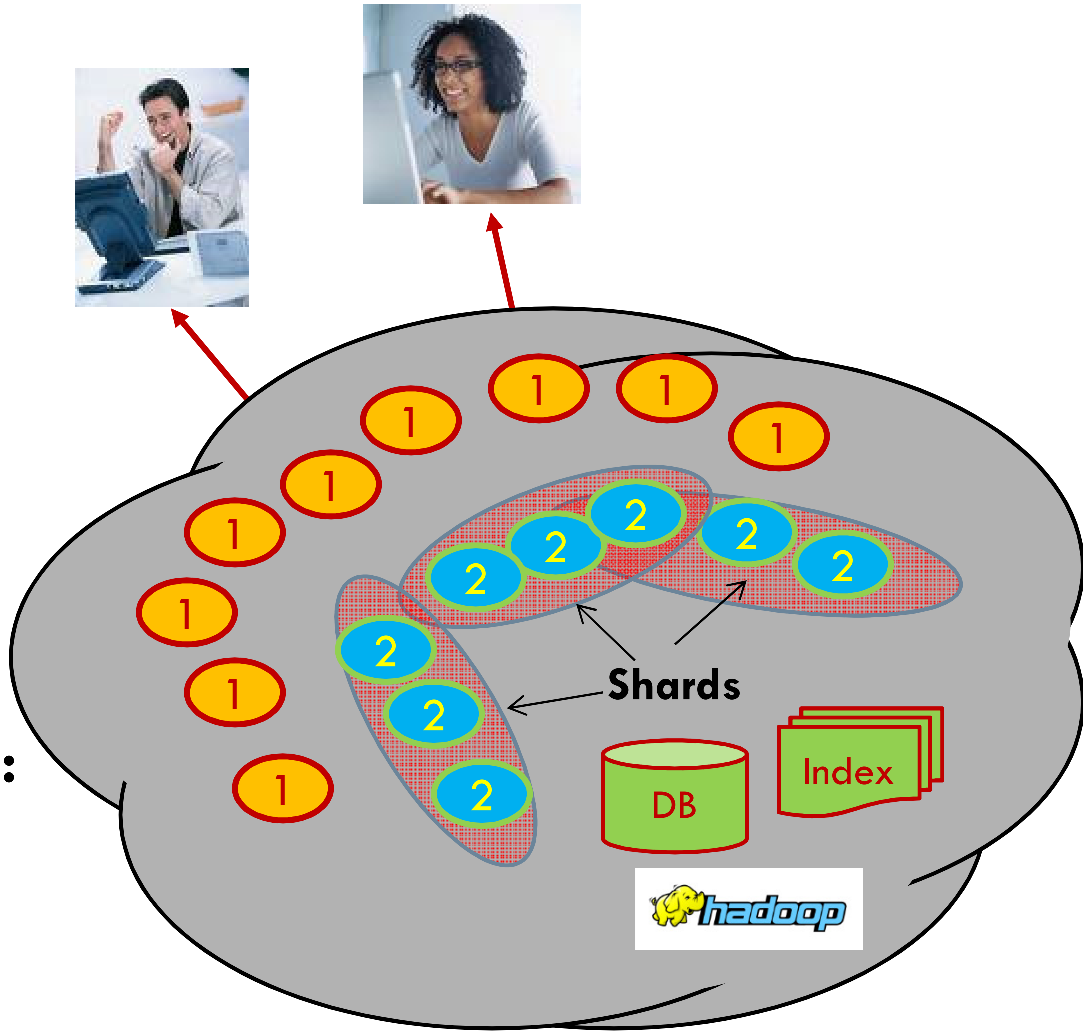
Break

How are clouds structured?

- Clients talk to clouds using web browsers or the web services standards
  - But this only gets us to the outer "skin" of the cloud data center, not the interior
  - Consider Amazon: it can host entire company web sites (like Target.com or Netflix.com), data, servers (EC2) and even user-provided virtual machines
    - Brings up performance, security, privacy issues

Big picture overview

- Client requests are handled in the "first tier" by
  - PHP or ASP pages
  - Associated logic
- These lightweight services are fast and very nimble
- Much use of caching: the second tier



Clouds have multiple tiers

- Tier 1: Very lightweight, responsive "web page builders" that can also route (or handle) "web services" method invocations. Limited to "soft state".
- Tier 2: (key,value) stores and services that support tier 1. Basically, various forms of caches.
- Inner tiers: Online services that handle requests not handled in the first two tiers. These can store persistent files, run transactional services. But we shield them from load.
- Back end: Runs offline services that do things like indexing the web overnight for use by tomorrow morning's tier-1 services.

Replication

- A central feature of the cloud
- To handle more work, make more copies
  - In the first tier, which is highly elastic, data center management layer pre-positions inactive copies of virtual machines for the services we might run
    - Exactly like installing a program on some machine
  - If load surges, creating more instances just entails
    - Running more copies on more nodes
    - Adjusting the load-balancer to spray requests to new nodes
- If load drops… just kill the unwanted copies!
  - Little or no warning. Discard any "state" they created locally.

Replication is about keeping copies

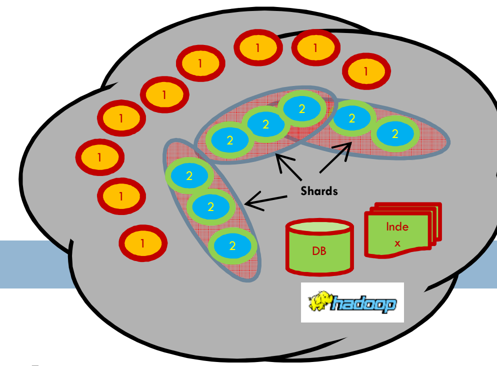
- The term may sound fancier but the meaning isn't

- Whenever we have many copies of something we say that we've replicated that thing
  - Usually "replica" implies "identical"
  - Instead of *replication* we use the term *redundancy* for things like alternative communication paths (e.g. if we have two distinct TCP connections from some client system to the cloud)
  - Redundant things might not be identical. Replicated things usually play identical roles and have equivalent data.

Things we can replicate in a cloud

- Files or other forms of data used to handle requests
  - If all our first tier systems replicate the data needed for end-user requests, then they can handle all the work!
  - Two cases:
    1. data is "write once" like a photo
    2. data evolves over time, like the current inventory count for the latest iPad in the Apple store
- Computation
  - Here we replicate some *request* and then spread work of computing the answer over multiple programs in the cloud
  - We benefit from parallelism by getting a faster answer
  - Can also provide fault-tolerance

Shards



- The caching components running in tier two are central to the responsiveness of tier-one services
  - We need to replicate data within our cache to spread loads and provide fault-tolerance
  - But not everything needs to be "fully" replicated. Hence we often use "shards"
    - Partition the data
    - Store the partition (shard) on a few replicas

Sharding used in many ways

- The second tier could be any of a number of caching services:
  - Memcached: a sharable in-memory key-value store
  - Dynamo: A replicated key-value service created by Amazon as a scalable way to represent the shopping cart and similar data
  - BigTable: A very elaborate key-value store created by Google and used not just in tier-two but throughout their “GooglePlex” for sharing information
  - Other DHTs that use key-value APIs
- Notion of sharding is cross-cutting
  - Most of these systems replicate data to some degree

Do we *always* need to shard data?

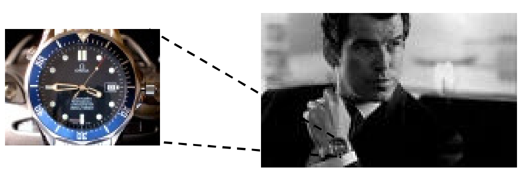
- Imagine a tier-one service running on 100k nodes
  - Can it ever make sense to replicate data on the entire set?
- *Yes*, if some kinds of information might be so valuable that almost every external request touches it.
  - Must think hard about patterns of data access and use
  - Some information needs to be heavily replicated to offer super fast access on vast numbers of nodes
  - We want the level of replication to match level of load and the degree to which the data is needed on the critical path

Concept of "consistency"

- A replicated entity behaves in a consistent manner if it mimics the behavior of a non-replicated entity
  - E.g. if I ask it some question, and it answers, and then you ask it that question, your answer is either the same or reflects some update to the underlying state
  - Many copies but acts like just one
- An inconsistent service is one that seems "broken"

Consistency lets us ignore implementation

A <u>consistent</u> distributed system will often have many components, but users observe behavior indistinguishable from that of a single-component reference system



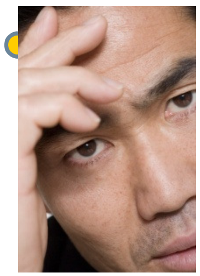
Reference Model

Implementation

Dangers of Inconsistency

My rent check bounced?
That can't be right!

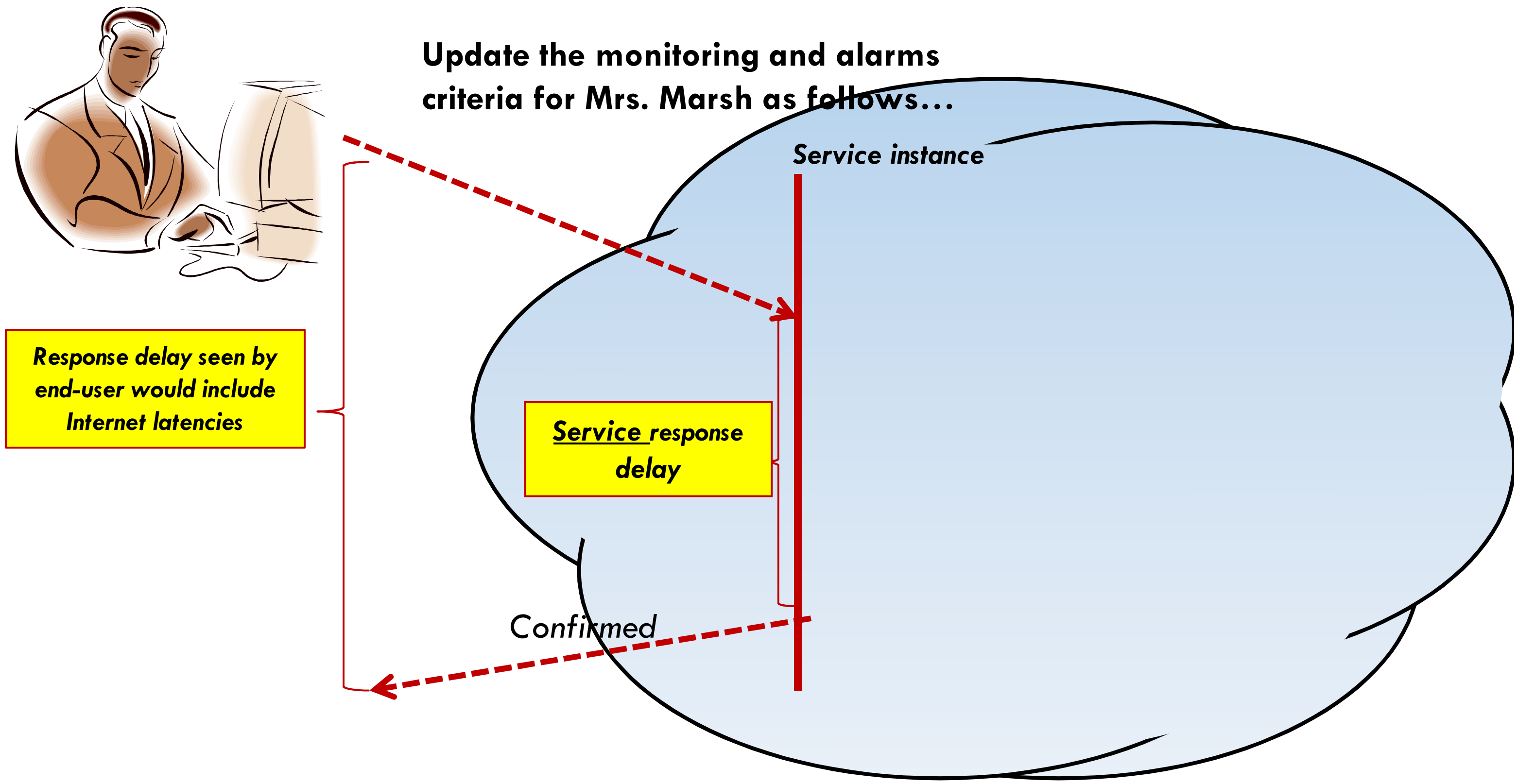
- Inconsistency causes bugs
  - Clients would never be able to trust servers… a free-for-all



- Weak or "best effort" consistency?
  - Common in today's cloud replication schemes
    - To avoid delaying the "critical path"
  - But strong security guarantees demand consistency
  - Would you trust a medical electronic-health records system or a bank that used "weak consistency" for better scalability?

Concept of "critical path"

- Focus on delay until a client receives a reply
- Critical path are actions that contribute to this delay



What if a request triggers updates?

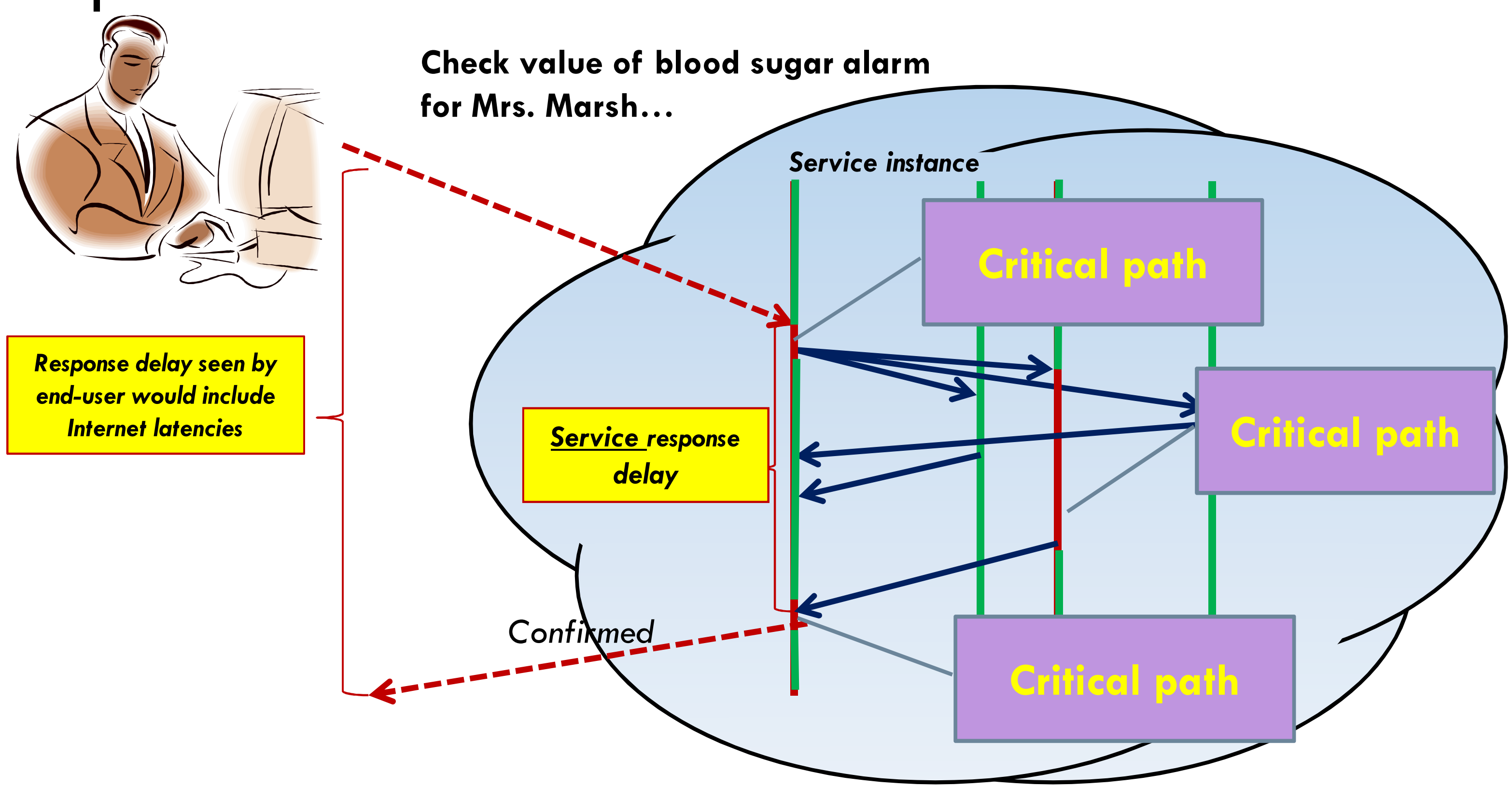
- If the updates are done "asynchronously" we might not experience much delay on the critical path
  - Cloud systems often work this way
  - Avoids waiting for slow services to process the updates but may force the tier-one service to "guess" the outcome
  - For example, could optimistically apply update to value from a cache and just hope this was the right answer
- Many cloud systems use these sorts of "tricks" to speed up response time

First-tier parallelism

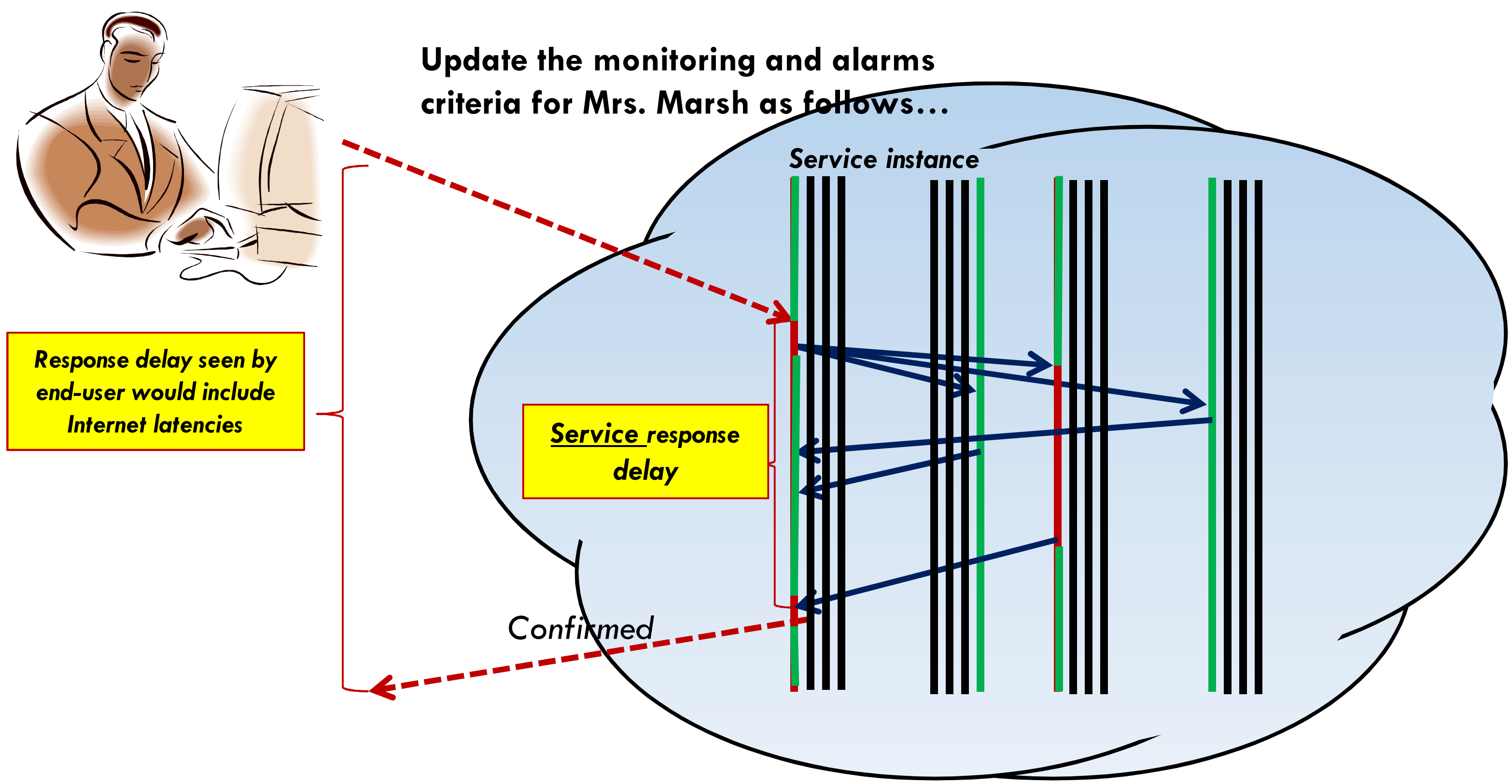
- Parallelism is vital to reducing critical delay
- Key question:
  - Request has reached some service instance X
  - Will it be faster…
    - … For X to just compute the response
    - … Or for X to subdivide the work by asking subservices to do parts of the job?
- Glimpse of an answer
  - Werner Vogels, CTO at Amazon, commented in one talk that many Amazon pages have content from 50 or more parallel subservices that ran, in real-time, on your request!

Concept of "critical path"

- In this example of a parallel read-only request, the critical path centers on the middle "subservice"



With replicas we just load balance

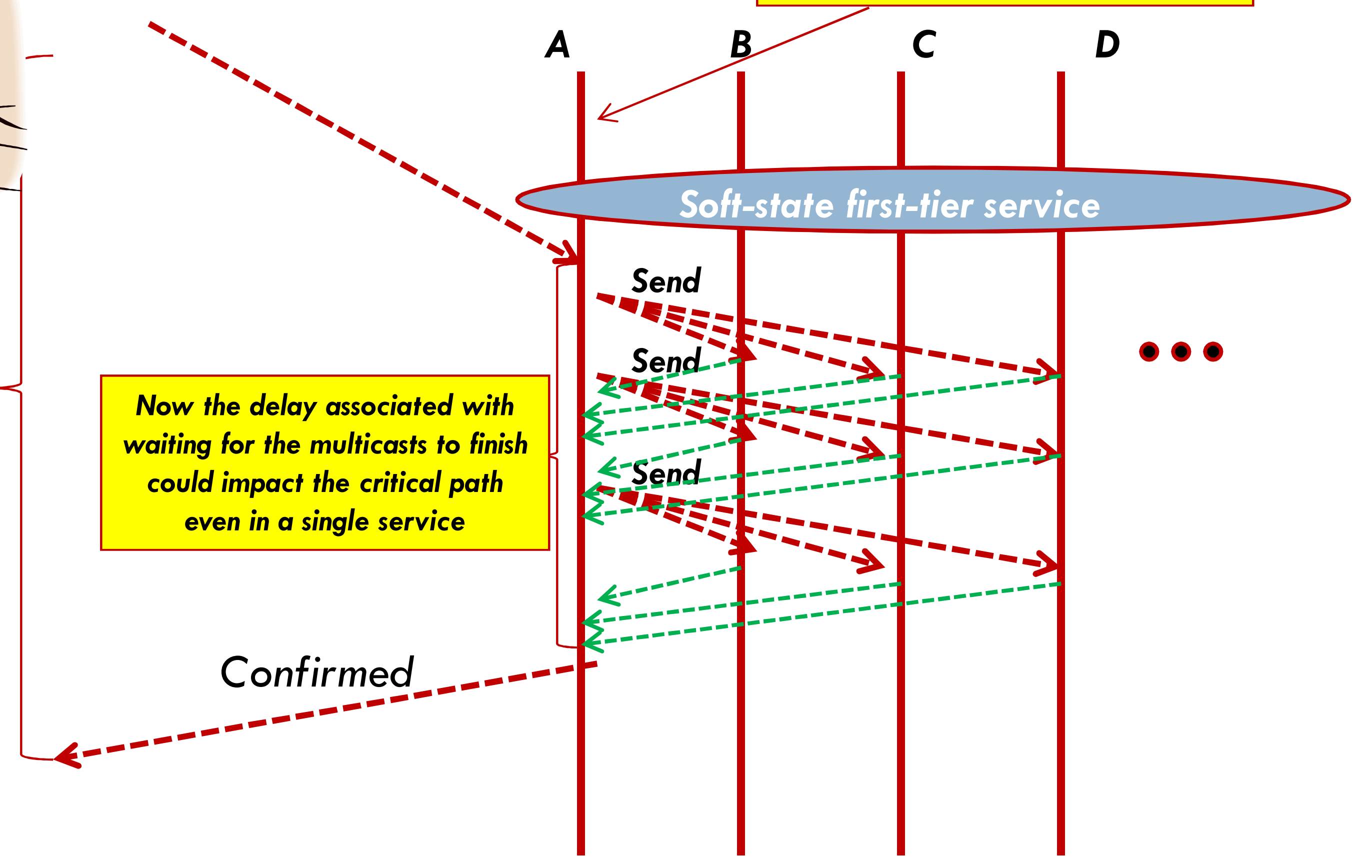


But when we add updates….

Update the monitoring and alarms criteria for Mrs. Marsh as follows…

Execution timeline for an individual first-tier replica

Response delay seen by end-user would also include Internet latencies not measured in our work



What if we send updates without waiting?

- Several issues now arise
  - Are all the replicas applying updates in the same order?
    - Might not matter unless the same data item is being changed
    - But then we clearly need some “agreement” on order
  - What if the leader replies to the end user but then crashes and it turns out that the updates were lost in the network?
    - Data center networks are surprisingly lossy at times
    - Also, bursts of updates can queue up
- Such issues result in *inconsistency*

Eric Brewer’s CAP theorem

- In a famous 2000 keynote talk at ACM PODC, Eric Brewer proposed that “you can have just two from Consistency, Availability and Partition Tolerance”

Eric Brewer's CAP theorem

- Consistency
  - any data item has a value reached by applying all prior updates in some agreed upon order
  - Must never forget an update once it has been accepted and client has been sent a reply (durability)
- Availability
  - Service should keep running and offer rapid responses even if a few replicas have crashed/are unresponsive
  - No client ever left waiting (even if can't get needed data now)
- Partition tolerance
  - System should continue to run even if net fails, cutting off some nodes from the others

Eric Brewer’s CAP theorem

- Brewer argues that data centers need very snappy response, hence availability is paramount
  - And they should be responsive even if a transient fault makes it hard to reach some service (hence, partition tolerance)
  - Thus, should use cached data to respond faster even if the cached entry can’t be validated and might be stale, wrong, or partially missing
- Conclusion: weaken consistency for faster response

CAP theorem

- A proof of CAP was later introduced by MIT's Seth Gilbert and Nancy Lynch
  - Suppose a data center service is active in two parts of the country with a wide-area Internet link between them
  - We temporarily cut the link ("partitioning" the network)
  - And present the service with conflicting requests
- The replicas can't talk to each other so can't sense the conflict
- If they respond at this point, inconsistency arises

Is inconsistency a bad thing?

- How much consistency is really needed in the first tier of the cloud?
  - Think about YouTube videos. Would consistency be an issue here?
  - What about the Amazon "number of units available" counters. Will people notice if those are a bit off?
- Puzzle: can you come up with a general policy for knowing how much consistency a given thing needs?

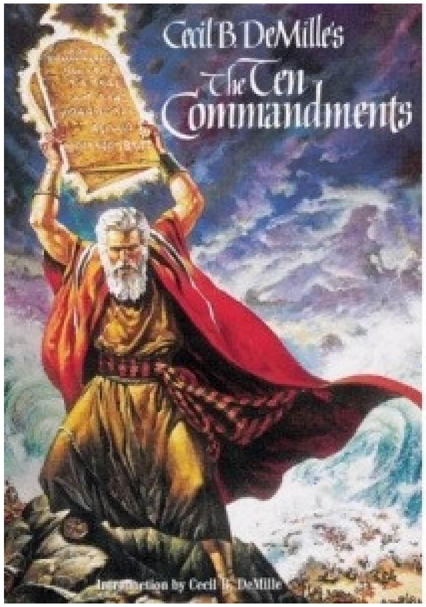
THE WISDOM OF THE SAGES

eBay's Five Commandments

- As described by Randy Shoup at LADIS 2008

Thou shalt…

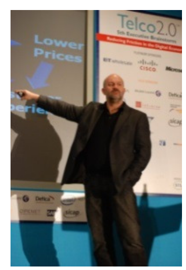
1. **Partition Everything**
2. **Use Asynchrony Everywhere**
3. **Automate Everything**
4. **Remember: Everything Fails**
5. **Embrace Inconsistency**



Vogels at the Helm

- Werner Vogels, CTO at Amazon
- He was involved in building a new shopping cart service
  - The old one used strong consistency for replicated data
  - New version was build over a DHT, like Chord, and has weak consistency with eventual convergence

- This weakens guarantees… but
  - ***Speed matters more than correctness***



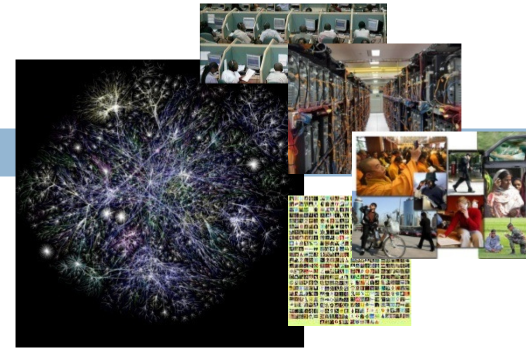
James Hamilton's advice

VP & Engineer, Amazon

- Key to scalability is decoupling, loosest possible synchronization
- *Any* synchronized mechanism is a risk
  - His approach: create a committee
  - Anyone who wants to deploy a highly consistent mechanism needs committee approval

.... They don't meet very often

Consistency



Consistency technologies just don't scale!

But inconsistency brings risks too!

My rent check bounced? That can't be right!

- Inconsistency causes bugs
  - Clients would never be able to trust servers… a free-for-all

- Weak or "best effort" consistency?
  - Strong security guarantees demand consistency
  - Would you trust a medical electronic-health records system or a bank that used "weak consistency" for better scalability?

Puzzle: Is CAP valid in the cloud?

- Facts: data center networks don't normally experience partitioning failures
  - Wide-area links do fail
  - But most services are designed to do updates in a single place and mirror read-only data at others
  - So the CAP scenario used in the proof can't arise
- But Brewer's argument is also about performance
  - Argues against waiting for a slow service to respond and instead, using any single replica you can find

Example – new X-Box released

- New X-Box released weeks before X-mas
  - 100,000s of parents visit web page on Amazon
  - Amazon does not want to miss a single sale
- Options
  - Perfect accuracy: delay response by forcing user to wait while web-page builder (first-tier) asks inventory service (inner tier) to reserve X-Box
    - not all reservations pan out, may lose real sales this way
  - Optimistic mode: book sale without checking inventory
    - Highly responsive service with some risk of overselling
- Amazon: Each 100ms delay reduces sales by 1%!