Distributed Systems

CAP and Clouds

Everyone talking about clouds

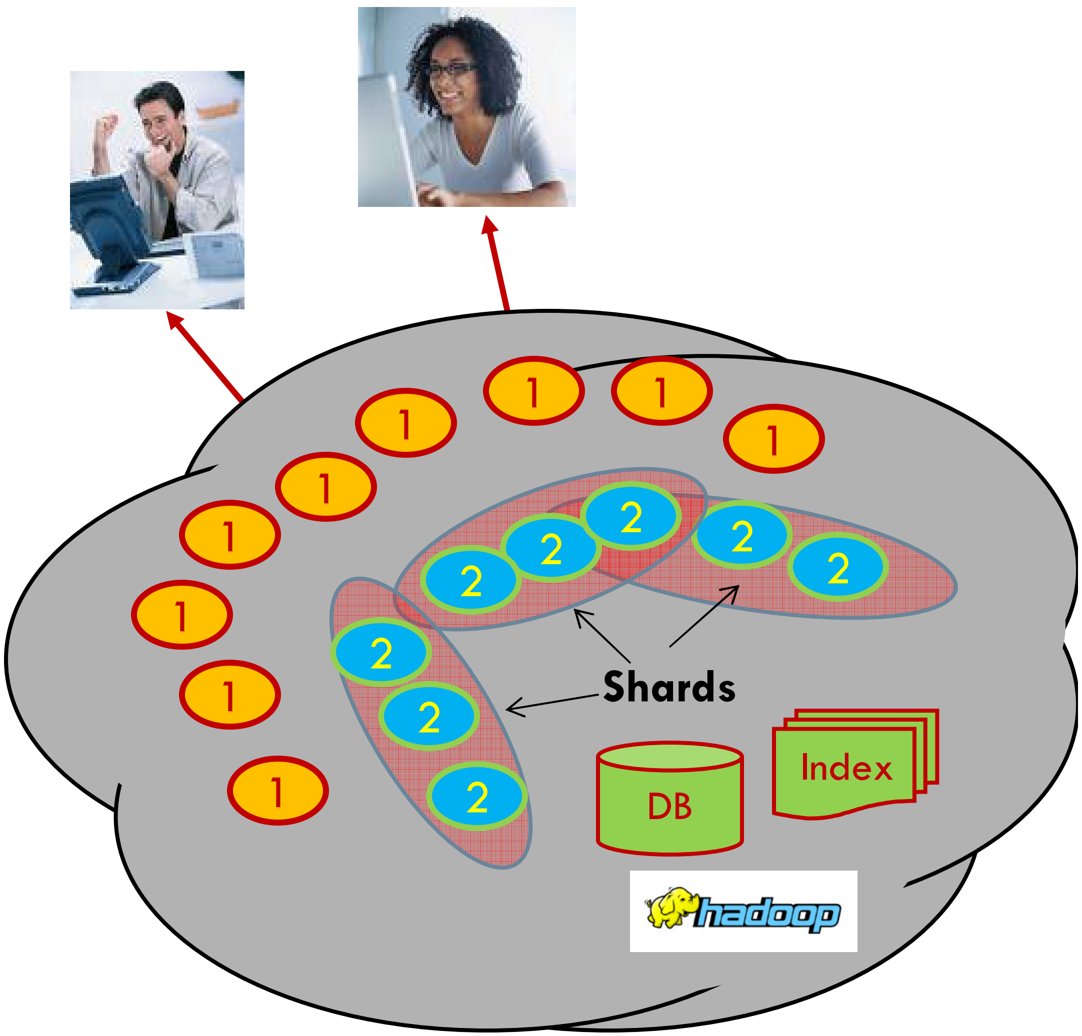
- What are these things?
  - Fancy buzzword for massive data centers with distributed systems technologies?
  - Any form of computing accessible over a net?
  - Any activity involving access to and using massive data sets?
  - Outsourcing technology?
    - i.e., ship data and computation to a remote place where computing and storage are cheap?
  - All of the above?
- Support web systems, social networks, e-commerce, and many others
  - Significant examples owned by big companies (e.g., Amazon, Microsoft. Google)
    - Can enable start-ups to be successful "overnight"
    - Scalable high assurance applications not well-supported yet
      - ATC, banking, mgt of electronic records, military apps in the cloud?

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 tier. 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.

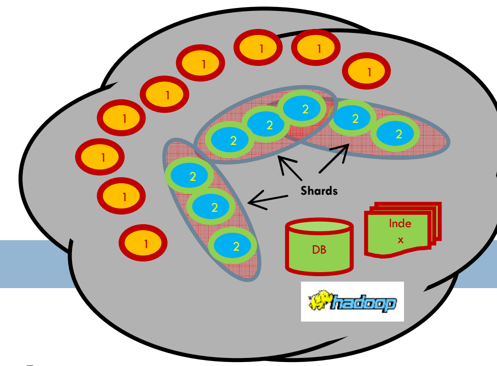
Replication is about keeping copies

- 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
  - Use cached data at first-tier whenever possible so the inner services are shielded from "online" load
  - 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" with just 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
  - Other kinds of DHTs that use key-value APIs
  - 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
- 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

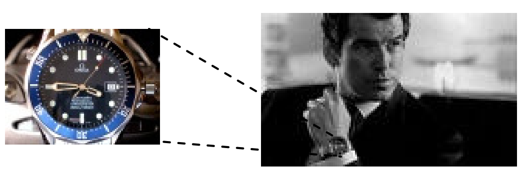
Concept of "consistency"

- 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

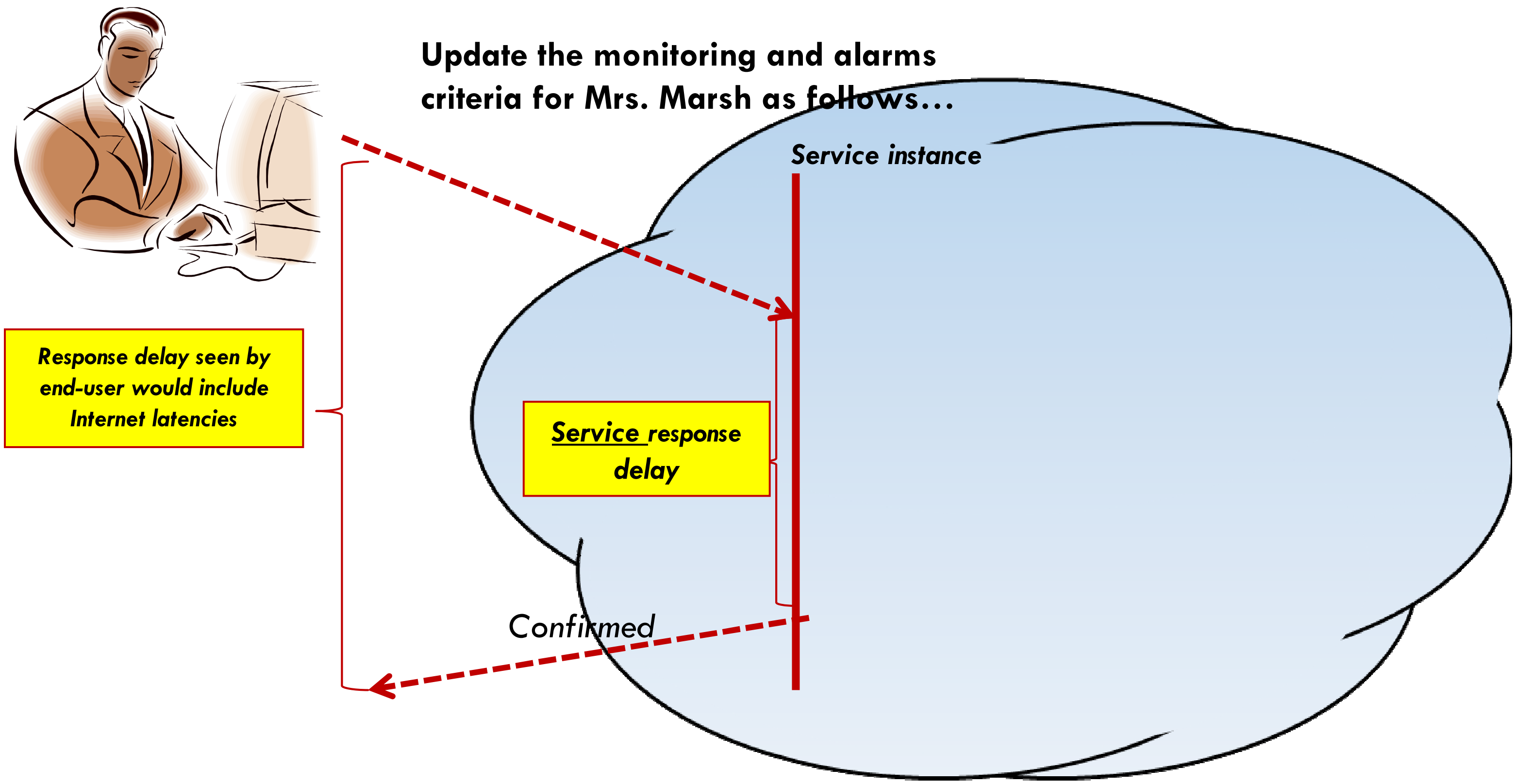
My rent check bounced? That can't be right!

- 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?

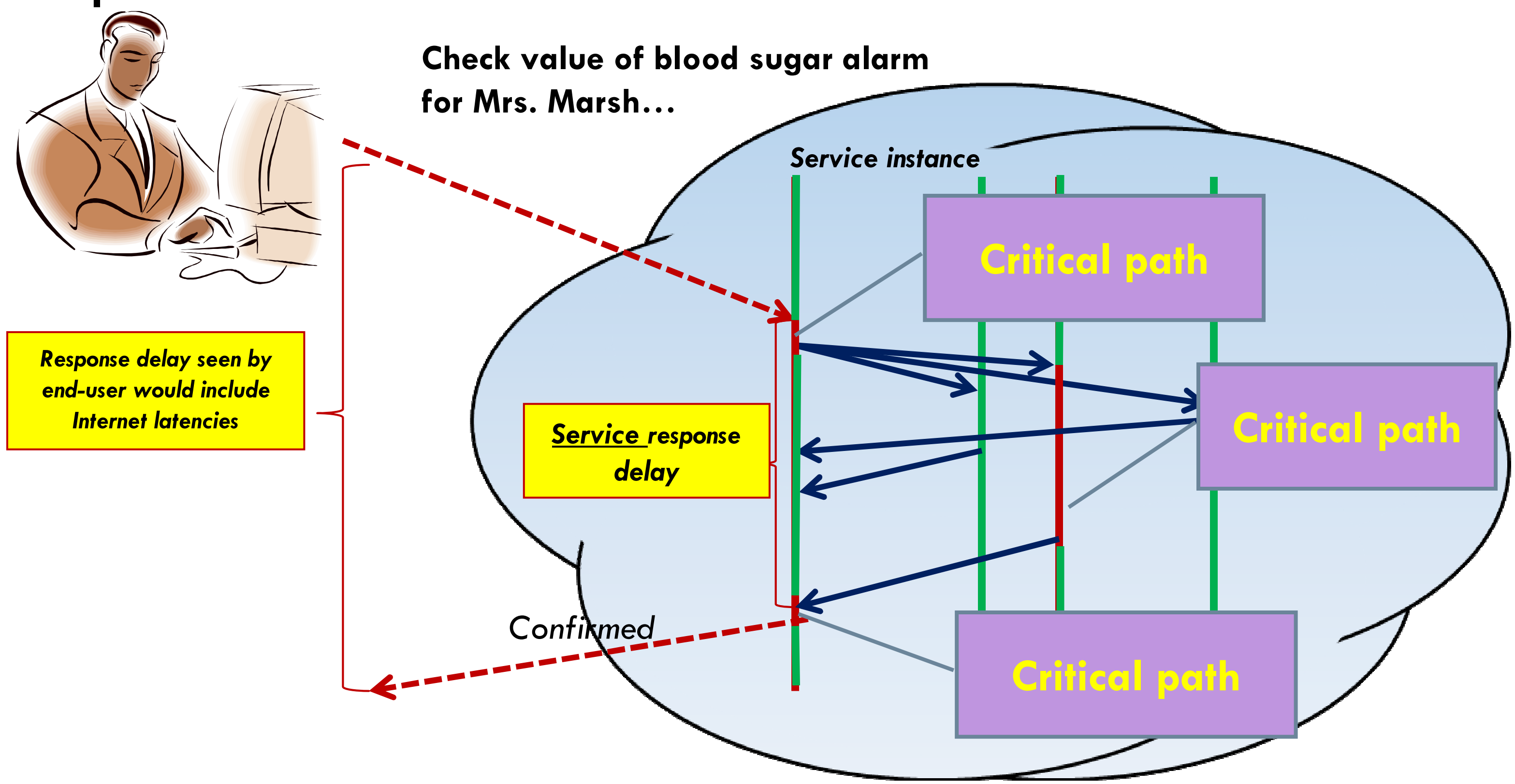
- If the updates are done "asynchronously" we might not experience much delay on the critical path
  - Cloud systems often work this way
  - Avoids waiting for slow services to process the updates but may force the tier-one service to "guess" the outcome
  - For example, could optimistically apply update 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

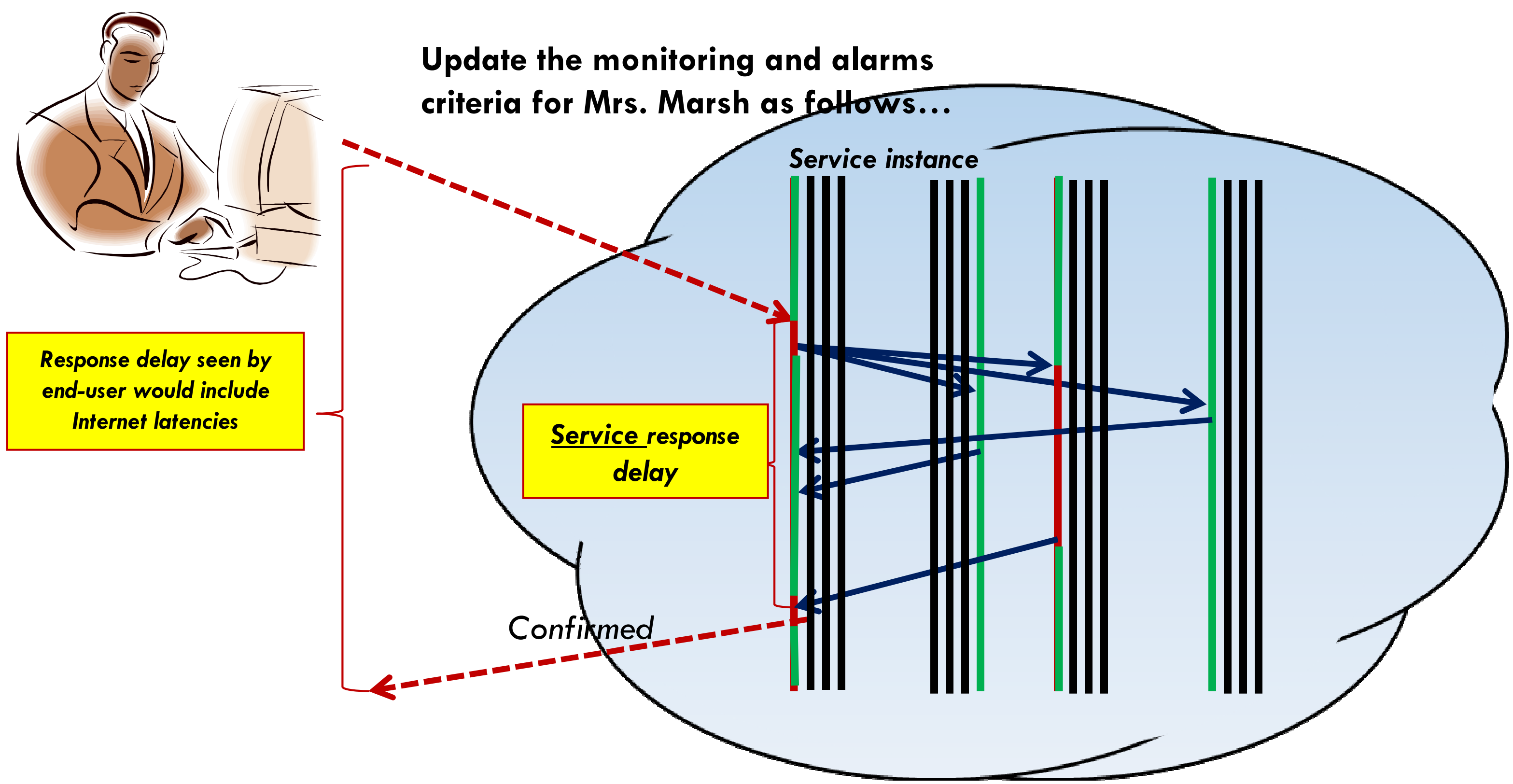
- Parallelism is vital to speeding up first-tier services
- Key question:
  - Request has reached some service instance X
  - Will it be faster…
    - … For X to just compute the response
    - … Or for X to subdivide the work by asking subservices to do parts of the job?
- Glimpse of an answer
  - Werner 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"



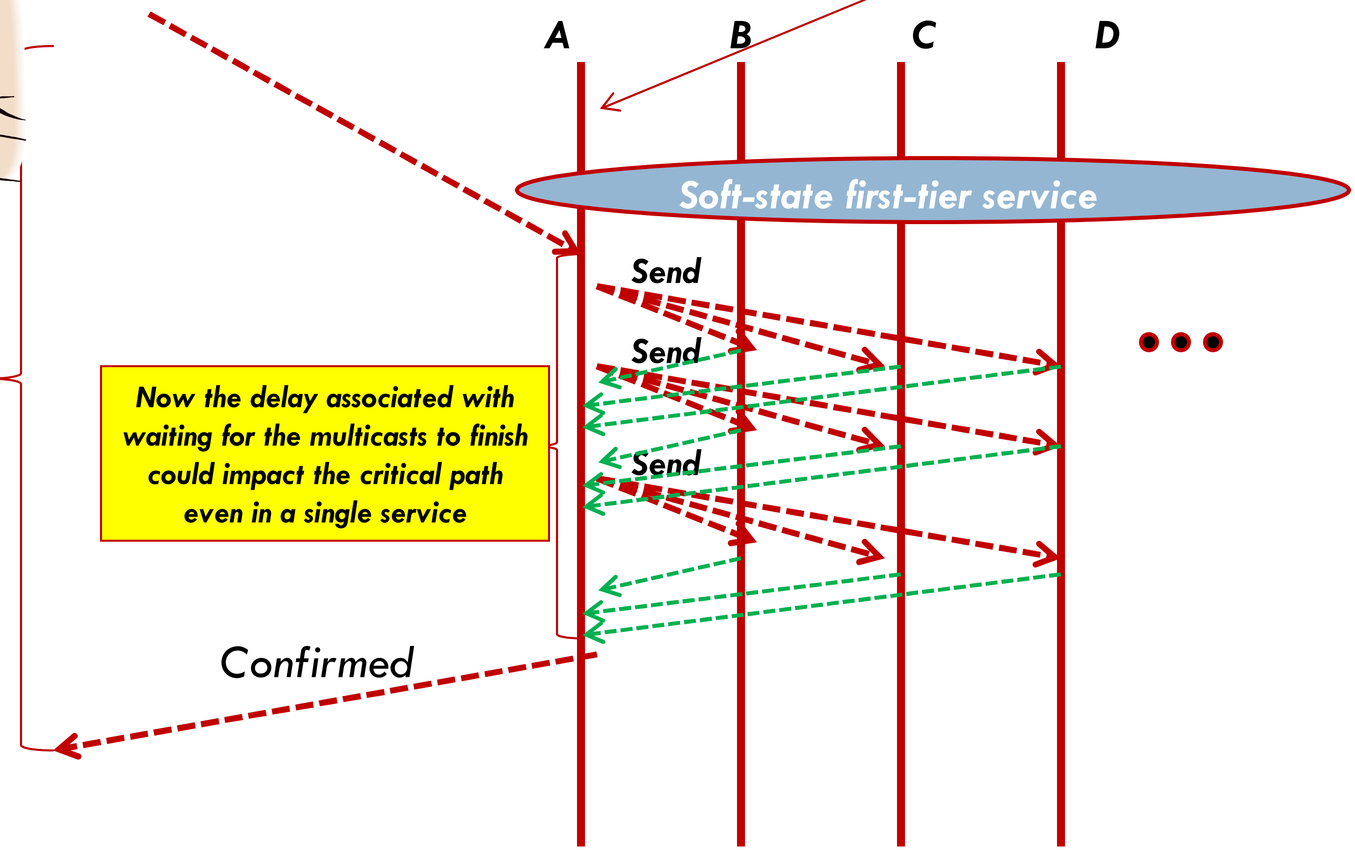
With replicas we just load balance



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

Now the delay associated with waiting for the multicasts to finish could impact the critical path even in a single service

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?

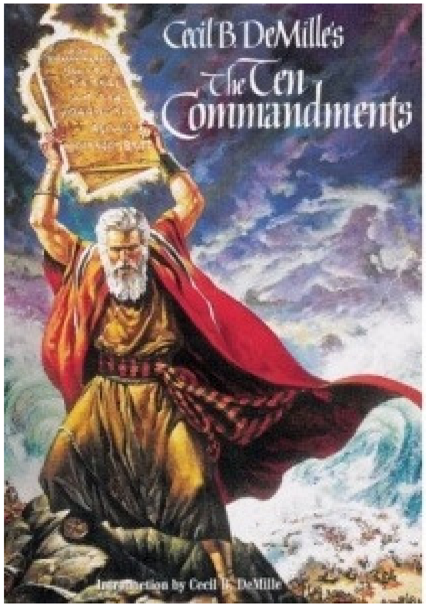
THE WISDOM OF THE SAGES

eBay's Five Commandments

- As described by Randy Shoup at LADIS 2008

Thou shalt…

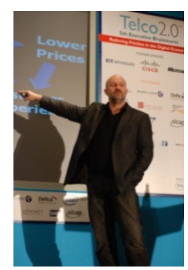
1. **Partition Everything**
2. **Use Asynchrony Everywhere**
3. **Automate Everything**
4. **Remember: Everything Fails**
5. **Embrace Inconsistency**



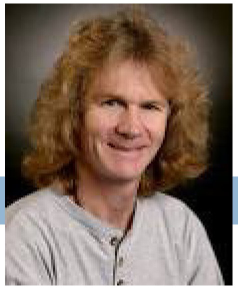
Vogels at the Helm

- Werner Vogels is CTO at Amazon.com…
- 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*

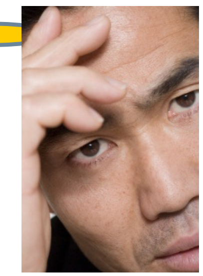
Consistency

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
- Brewer's argument about not waiting for a slow service to respond does make sense
  - Argues for 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%!