

CS5412: ANATOMY OF A CLOUD

Lecture VII

Ken Birman

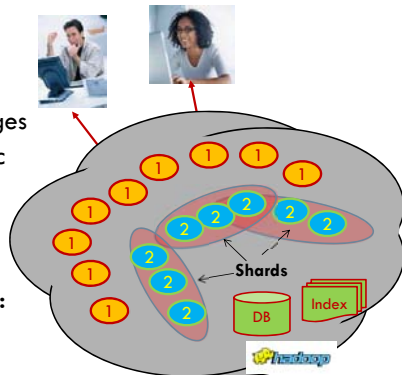
How are cloud structured?

- Clients talk to clouds using web browsers (increasingly more) or the web services standards (increasingly less)
 - ▣ But this only gets as far as 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 (AC3), servers (EC2) and even user-provided virtual machines!

CS5412 Spring 2012 (Cloud Computing: Birman)

Big picture overview

- Client requests are handled in the “**first tier**” by
 - ▣ E.g. PHP or ASP pages
 - ▣ And associated logic
- These lightweight services are fast and very nimble
- Much use of caching: the **second tier**



CS5412 Spring 2012 (Cloud Computing: Birman)

Many styles of system

- **Near the edge** of the cloud, focus is on
 - ▣ vast numbers of clients
 - ▣ rapid response
- **Inside**, we find high-volume services that operate in a pipelined manner, asynchronously
- **Deep inside the cloud**, we see a world of virtual computer clusters that are scheduled to share computing resources and on which massively-parallel applications like MapReduce (Apache Hadoop) are very popular

CS5412 Spring 2012 (Cloud Computing: Birman)

In the outer tiers replication is key

- We need to replicate
 - ▣ **Processing**: each client has what seems to be a private, dedicated server (for a little while)
 - ▣ **Data**: as much as possible, that server has copies of the data it needs to respond to client requests without any delay at all
 - ▣ **Control information**: the entire structure is managed in an agreed-upon way by a decentralized cloud management infrastructure

CS5412 Spring 2012 (Cloud Computing: Birman)

What about the “shards”?

- The caching components running in tier two are central to the responsiveness of tier-one services
 - ▣ Basic idea: to always use cached data if at all possible, so the inner services (here, a DB and a search index stored in a set of files) 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

CS5412 Spring 2012 (Cloud Computing: Birman)

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 Distributed Hash Tables that use key-value APIs
 - ▣ Dynamo: A 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
- The notion of sharding is cross-cutting
 - ▣ Most of these systems replicate data to some degree

CS5412 Spring 2012 (Cloud Computing: Birman)

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 blindingly fast access on vast numbers of nodes
 - ▣ The principle is similar to the way Beehive operates (simultaneity, swarming, perception)
 - Even if we don’t make a dynamic decision about the level of replication required, the principle is similar
 - We want the level of replication to match the level of load and the degree to which the data is needed on the critical path

CS5412 Spring 2012 (Cloud Computing: Birman)

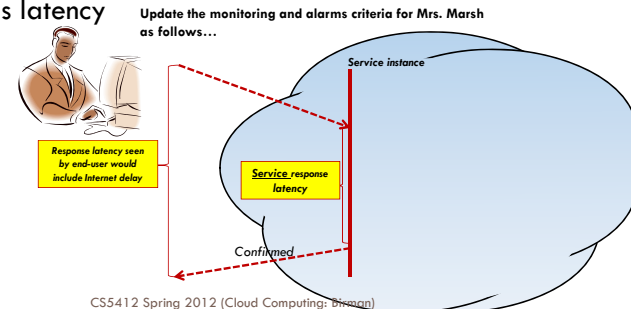
And it isn't just about updates

- Should also be thinking about patterns that arise when doing reads (“queries”)
 - ▣ Some can just be performed by a **single** representative of a service
 - ▣ Others might need that several (or even a huge number of) machines undertake parts of the work in **parallel**
- The term sharding is used for data, but here we might talk about “parallel computation on a shard”

CS5412 Spring 2012 (Cloud Computing: Birman)

What does “critical path” mean? 1/2

- Focus on the latency of the reply to the client
- Critical path is formed by actions that contribute to this latency



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

CS5412 Spring 2012 (Cloud Computing: Birman)

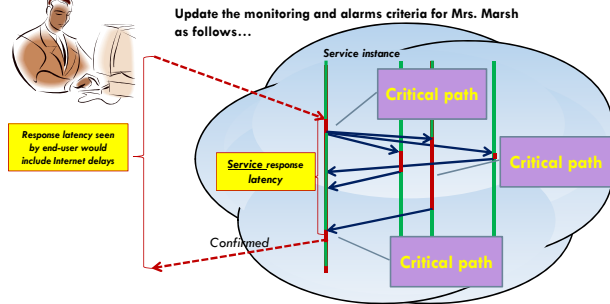
First-tier parallelism

- 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, noted in a talk that many Amazon pages have content from 50 or more **parallel subservices** that run, in real-time, on your request!

CS5412 Spring 2012 (Cloud Computing: Birman)

What does “critical path” mean? 2/2

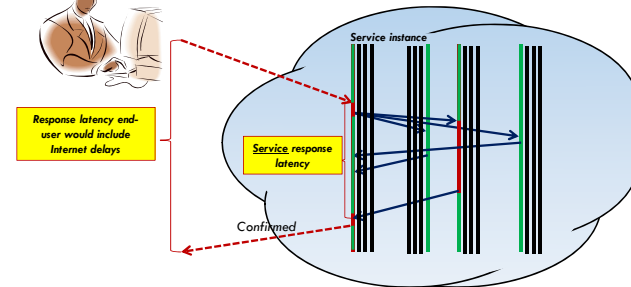
- In this example of a parallel read-only request, the critical path centers on the middle “subservice”



CS5412 Spring 2012 (Cloud Computing: Birman)

With replicas we just load balance

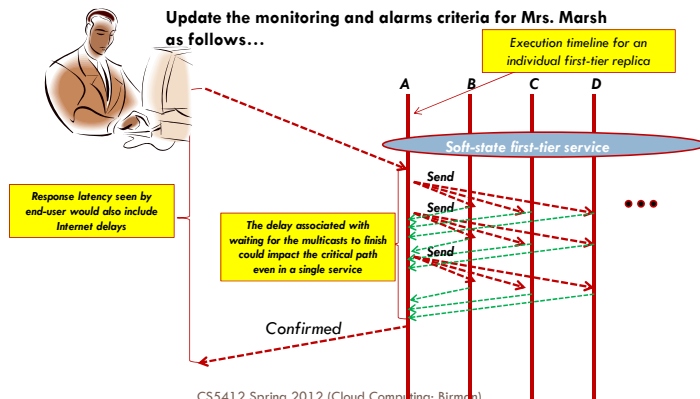
Update the monitoring and alarms criteria for Mrs. Marsh as follows...



CS5412 Spring 2012 (Cloud Computing: Birman)

But when we add updates....

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



CS5412 Spring 2012 (Cloud Computing: Birman)

What about updating w/o 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 clearly we do 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*

CS5412 Spring 2012 (Cloud Computing: Birman)

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”
 - ▣ He 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. So they should use cached data to respond faster even if the cached entry can't be validated and might be stale!
- Conclusion: **weaken consistency for faster response**

CS5412 Spring 2012 (Cloud Computing: Birman)

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 in between
 - ▣ 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

CS5412 Spring 2012 (Cloud Computing: Birman)

(Clarification) Intermission ... /1

- **Consistency (C)**
 - ▣ There must be a total order on all operations
 - Logical equivalent to centralization with run-to-completion operation semantics
 - ▣ Each operation looks as if it were completed at a single instant
 - ▣ Each update is applied to all relevant replicas at the same logical time
 - ▣ Consistency that is both instantaneous and global is simply impossible

SCD course @ UNIPD (Tullio Vardanega)



(Clarification) Intermission ... /2

- **Availability (A)**
 - ▣ Every request received by a non-failing node must yield a response
 - ▣ Every request processing must terminate even under severe network failures
- **Partition Tolerance (P)**
 - ▣ Under PT, the network may lose arbitrarily many messages sent from one node to another
 - ▣ When a network is partitioned, all messages sent from nodes in one component of the partition to nodes in another component are lost
 - ▣ Any node failure can be seen as a network partition

SCD course @ UNIPD (Tullio Vardanega)



(Clarification) Intermission ... /3



- The probability that any one node fails (giving rise to a network partition) rises exponentially with the number of nodes
 - $P(\text{failure}) = 1 - P(\text{individual_node_not_failing})^{\text{number_of_nodes}}$
- **Choosing C over A in the presence of partitions**
 - ▣ The system will preserve the guarantees of atomic reads and writes, and reject some requests
- **Choosing A over C**
 - ▣ The system will respond to all requests, potentially returning stale reads and accepting conflicting writes
 - ▣ Some of the conflicts may be resolved by Lamport's like algorithms

SCD course @ UNIPD (Tullio Vardanega)

(Clarification) Intermission ... /4



- Brewer's Yield and Harvest notions
- **Yield**
 - ▣ The probability of completing a request
 - ▣ More useful metric than *uptime*, as being down at peak or off-peak times generates the same uptime but vastly different yields
- **Harvest**
 - ▣ The fraction of data reflected in the response
 - The completeness of the answer to the query

$$\frac{\text{data_available}}{\text{total_data}}$$
- System design and implementation can influence whether faults impact Yield, Harvest or both
 - ▣ Replicated systems map faults to *reduced Yield* at peak times
 - ▣ Partitioned systems map faults to *reduced Harvest* for the same Yield

SCD course @ UNIPD (Tullio Vardanega)

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?

CS5412 Spring 2012 (Cloud Computing: Birman)

CS5412 Spring 2012 (Cloud Computing: Birman)



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



CS5412 Spring 2012 (Cloud Computing: Birman)

Vogels at the Helm


- Werner Vogels, CTO at Amazon.com ...
- Involved in building a new shopping cart service ...
 - ▣ The old one used strong consistency for replicated data
 - ▣ New version was built over a DHT, like **Chord**, and has weak consistency with eventual convergence
 - Chord: a scalable P2P lookup service
- This weakens guarantees ... but
 - ▣ *Speed matters more than correctness*



CS5412 Spring 2012 (Cloud Computing: Birman)

James Hamilton's advice

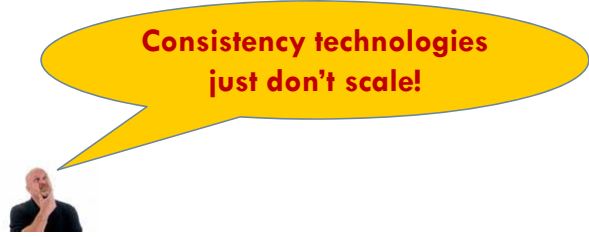
- Key to scalability is decoupling, loosest possible synchronization
- Any synchronized mechanism is a risk
 - ▣ His approach: create a committee
 - ▣ Anyone who wants to deploy a highly consistent mechanism needs committee approval (cf. Lamport's Paxos)



.... They don't meet very often

CS5412 Spring 2012 (Cloud Computing: Birman)

Consistency



CS5412 Spring 2012 (Cloud Computing: Birman)

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?



CS5412 Spring 2012 (Cloud Computing: Birman)

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
 - ▣ But does this preclude that replica being consistent?

CS5412 Spring 2012 (Cloud Computing: Birman)

What does “consistency” mean?

- We need to pin this basic issue down!
- As used in CAP, consistency is about two things
 - ▣ First, that updates to the same data item are applied in some agreed-upon order
 - ▣ Second, that once an update is acknowledged to an external user, it won't be forgotten
- **Not all systems need both properties**

CS5412 Spring 2012 (Cloud Computing: Birman)

This illustrates a challenge!

- Cloud systems just can't be approached in a one-size fits all manner
- For performance-intensive scalability scenarios we need to look closely at tradeoffs
 - ▣ Cost of stronger guarantee, versus
 - ▣ Cost of being faster but offering weaker guarantee
- If systems builders blindly opt for strong properties when not needed, we just incur other costs!
 - ▣ Amazon: Each 100ms delay reduces sales by 1%!

CS5412 Spring 2012 (Cloud Computing: Birman)

Properties we might want

- **Consistency:** Updates in an agreed order
- **Durability:** Once accepted, won't be forgotten
- **Responsiveness:** Replies with bounded delay
- **Security:** Only permit authorized actions by authenticated parties
- **Privacy:** Won't disclose personal data
- **Resilience:** Failures can't prevent the system from providing desired services
- **Coordination:** actions won't interfere with one another

CS5412 Spring 2012 (Cloud Computing: Birman)

Does CAP apply deeper in the cloud?

- The principle of wanting speed and scalability certainly is universal
- But many cloud services have strong consistency guarantees that we take for granted but depend on
- Marvin Theimer at Amazon explains
 - ▣ Avoid costly guarantees that aren't even needed
 - ▣ But sometimes you just need to guarantee something
 - ▣ Then, be clever and **engineer it to scale**
 - ▣ And expect to revisit it each time you scale out 10x

CS5412 Spring 2012 (Cloud Computing: Birman)

Cloud services and their properties

| Service | Properties it guarantees |
|--------------|---|
| Memcached | No special guarantees |
| Google's GFS | File is current if locking is used |
| BigTable | Shared key-value store with many consistency properties |
| Dynamo | Amazon's shopping cart: eventual consistency |
| Databases | Snapshot isolation with log-based mirroring (a fancy form of the ACID guarantees) |
| MapReduce | Uses a "functional" computing model within which offers very strong guarantees |
| Zookeeper | Yahoo! file system with sophisticated properties |
| PNUTS | Yahoo! database system, sharded data, spectrum of consistency options |
| Chubby | Locking service... very strong guarantees |

CS5412 Spring 2012 (Cloud Computing: Birman)

Is there a conclusion to draw?

- One thing to notice about those services ...
 - ▣ Most of them cost 10's or 100's of millions to create!
 - ▣ Huge investment required to build strongly consistent and scalable and high performance solutions
 - ▣ Oracle's current parallel database: billions invested
- CAP isn't about telling Oracle how to build a database product ...
 - ▣ CAP is a warning to you that strong properties can easily lead to slow services
 - ▣ But thinking in terms of weak properties is often a successful strategy that yields a good solution and requires less effort

CS5412 Spring 2012 (Cloud Computing: Birman)

Core problem?

- When can we safely sweep consistency under the rug?
 - ▣ If we weaken a property in a safety-critical context, something bad can happen!
 - ▣ Amazon and eBay do well with weak guarantees because many applications just didn't need strong guarantees to start with!
 - ▣ By embracing their weaker nature, we reduce synchronization and so get better response behavior
- But what happens when a wave of high assurance applications starts to transition to cloud-based models?

CS5412 Spring 2012 (Cloud Computing: Birman)