AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
DISTRIBUTED SYSTEM VS PARALLEL SYSTEM
• Common/traditional approach: worry only about the disk

• Solution?

• RAID — Redundant Array of Inexpensive Disks

• Not distributed
  — No availability on server failures
  — No scale-out, not elastic
  — Not cheap

• Replication
  — Multiple copies (e.g., 3) of each data item
  — Copies on distinct storage nodes
  — Hopefully, fail independently
FAULT-TOLERANCE 101 – SCALE-OUT DISTRIBUTED STORAGE

- Many storage nodes (100Ks)
- Horizontal scalability, elasticity
- Originally cheap, low reliability, (now less so)
- Failures are the norm rather than the exception
WHERE ARE YOUR FILES?

- Google Drive
- OneDrive
- Dropbox
- iCloud
INSIDE THE CLOUD?
FAULT-TOLERANCE 101 – 
PRIMARY-BACKUP REPLICATION
THE CAP PRINCIPLE
(aka conjecture, theorem, tautology)

Why aren’t distributed stores easy to design and implement?
Brewer [2000]: In Partition tolerant systems, there is a tradeoff between Consistency and Availability.

- Basic intuition:
  - Consistency: no split brain (brain-divided)
  - Availability: to clients everywhere
  - Partitioned tolerance system
PRIMARY-BACKUP LIMITATIONS

- Does not work with network partitions, asynchrony, or “false suspicions”
FAULT-TOLERANCE 101 – SMR ARCHITECTURE

- special algorithms to manage this Type of architecture
  - Eg paxos

Client A

Client B

SMR

Apache Zookeeper

etcd
QUORUM SYSTEMS

- R / W: minimum number of nodes that must participate in a successful read / write
- R + W > N (overlap)

R = 3, W = 3, N = 5

R = 4, W = 2, N = 5
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
What Other Customers Are Looking At Right Now

- Garmin Nuvi 300 Pocket Vehicle GPS
- Sony Walkman A Series
- Haystacks
- The White Stripes Album
- The Dangerous Book for Boys Hardcover by Cornelia Funke

Top-Selling Fragrance at Amazon Beauty

- Get the RIZR in Cosmic Blue and Rose
- Webkinz Are Here
- Hit the Court This Summer with New Tennis Gear

Join() operations are relatively rare in Amazon's environment. Handle Join() well.

Join() operations are relatively rare in Amazon's environment.

Simple key-value store

Availability?

Model: We are here
• Amazon’s e-commerce platform
  ▪ Shopping cart served tens of millions requests, over 3 million checkouts in a single day (2007)
  Until Now relational (SQL) DBs were used
  • Handle Join() well
  • Do we need it?
    • Join() operations are relatively rare in Amazon’s environment
  • Simple key value store

• Availability?
  • Every write must return
  • Unavailability?

• Model:
ASSUMPTIONS AND SYSTEM PROPERTIES

- Multiple nodes
- Network failure, blackouts …
- Simple key-value pairs are enough
- Every key is replicated N (parameter) times
- Security is a non-issue
• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
DESIGN GOALS

• High performance
• Scalability
• Heterogeneity
• High availability
• decentralization
• Partition tolerance
  What are we going to lose?

• Metrics and SLA:
  • No more average of median costumer in mind.
  • Guarantees and optimizations aimed at the 99.9th percentile
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
• We will understand this table
• Design challenge → solution

• Dynamo:
  Sum of distributed system techniques
WHAT ARE WE TRYING TO ACHIEVE?

- Partitioning (scalability)
- High Availability for Writes
- Handling temporary failures
- Recovering from permanent failures
- Membership and failure detection
PROBLEM: PARTITIONING

• Storing data in multiple nodes helps overcoming crash failures
• Splitting data arbitrarily can’t provide strong guarantees
  • Remember the SLA
TECHNIQUE: CONSISTENT HASHING

• A network overlay, like Chord and PASTRY
• Every node gets its place on the ring
• The node’s position on the ring determines the key ranges it is responsible for.
PARTITIONING: CONSISTENT HASHING

- Hash function output range is ring
- Hash of key is a location in the ring
- Walk clockwise to find containing node
- Problems:
  - Non-uniform data/load distribution
  - Cannot leverage more powerful nodes

Key: “236832”
Value: “Best class I’ve ever taken”

Hash(“236832”) = 4
PARTITIONING: VIRTUAL NODES

- Nodes assigned multiple points
- Advantages:
  - Efficient redistribution of load
  - More power = more points on circle
• N represents degree of replication
• Responsible through Nth predecessor
• Each node contains preference list
• Preference list:
  • Contains nodes responsible for given key
PROBLEM: HIGH AVAILABILITY FOR WRITES

• An “add to cart” operation may never be lost, or delayed
• Writes to the same key may happen:
  • Concurrently
  • In different network partitions
• Need to mitigate these somehow
CAUSAL ORDER

Vicki

Hey! How’s your day going?

I don’t think

Are your texts out of order?

Are yours?

I think something is wrong with my phone 😡

Vanessa Wojtusiaz

Hey! How’s your day going?

Hiii 😊😊😊

Are your texts out of order?

I don’t think

I think something is wrong with my phone 😡

Are yours?
TECHNIQUE: VERSIONING

How it Works?

• Vector clocks (Lamport? not really, Honorable mention)
  • A list of <Node, counter> tuples
  • (weakly) order events in the system!
  • Now we know which events could have caused others
D3 & D4 not causally related conflict!
• Usually, when reading different histories of an object it is possible to construct a consistent version

• When it is – coordinator node does just the reconciliation
  • And writes back the reconciled version

• Sometimes it isn’t possible
  • Remember why?

• In these cases, the divergent versions are returned to the client (an Amazon service)
  • Client service reconciled according to business logic, timestamp, etc.
PROBLEM: HANDLING TEMPORARY FAILURES

- Nodes may be under maintenance
- Datacenters and clusters may experience network shortages
- High availability requires rebalancing the load
  - Upon failures
  - Upon recovery
SLOPPY QUORUMS

• First: Regular Quorums
  • \( R + W > N \)

• Why couldn’t Dynamo just use them?
  • Oh, right, partitions!

• Sloppy quorums are relaxed quorums
  • It is not guaranteed that all read and write quorums intersect
  • Relaxed consistency
Let’s assume that each value is stored in $N = 3$ nodes

Node A in the preferred list is unavailable

Store in D instead, with a hint:
  - Value should be stored in A

When A is available again, D hands the values back to A

What did we achieve?
PROBLEM: MEMBERSHIP AND FAILURE DETECTION

• How useful are responses to failures if we can’t detect failures?
• Membership: How do I know that a node joined or was removed?
  o Should such events be announced explicitly or inferred?
MEMBERSHIP

• Explicit, rather than inferred
  o Saves unnecessary rebalancing
  o Propagates using gossip, Epidemic protocols? what about pandemic protocols
• This may still create temporary partitions
• Super nodes “seeds”
  o Known to all
  o split views become very unlikely
FAILURE DETECTION

• Purely local
  o If A didn’t receive a response from B within timeout, A considers B as temporary unavailable

• Why not Global?
  o Permanent additions and removals are already covered, no need to detect them
PROBLEM: HANDLING PERMANENT FAILURES

• Nodes may be unavailable for long times
• Replicas (including hinted ones) may remain unreachable

Solution:
• Background replica synchronization (have more up-to-date copies)
• Verify using Merkle Trees
TECHNIQUE: MERKLE TREES

• Hash trees, patented in 1979
• Every node maintains a tree per range
• Leaves are hashes of single items
• Parents nodes are hashes of children
• Easy to verify that a range is up to date between nodes

• $O(\log N)$ data transfer worst case, instead of $O(N)$
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
• Remember, we're in 2007:
EVALUATION

• Load is very well balanced
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
BACK TO THE FUTURE

• Dynamo wasn’t a big success as an independent service
  • Too difficult for programmers to use
• Paper had a lot of impact!
  • Cited 5K times
  • Influenced many future systems
• NoSQL databases (e.g., Cassandra) – next week
• DynamoDB – a publicly available service by AWS
AGENDA

• Introduction
• Background
• Design goals
• Challenges, observations and solutions
  • (with some additional background)
• Evaluation
• Back to the future
• Conclusion
CONCLUSION

• Crash course in distributed storage
• We saw some challenges, tradeoffs and techniques
• We covered (fingers crossed):
  • Fault tolerance 101
  • Partitioning
  • High Availability for Writes
  • Handling temporary failures
  • Membership and failure detection
  • Recovering from permanent failures
QUESTIONS?

strong consistency when a node crashes

broadcasting

ok everything responded except one

it's coming anytime now

may as well time out
FAULT-TOLERANCE 101 – PRIMARY-BACKUP REPLICATION

• High throughput primary-backup variant
SMR - AGREING ON A LOG

<table>
<thead>
<tr>
<th>x ← 3</th>
<th>x ← 5</th>
<th>y ← x+1</th>
<th>z ← x+y</th>
<th>y ← 14</th>
<th>w ← 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x ← 3</td>
<td>x ← 5</td>
<td>y ← x+1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x ← 3</td>
<td>x ← 5</td>
<td>y ← x+1</td>
<td>z ← x+y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PRIMARY-BACKUP LIMITATIONS

- Does not work with network partitions, asynchrony, or “false suspicions”
- Works with benign (correct) servers only
  - No resilience to bugs, intrusions
ALSO BLOCKCHAINS

• Bitcoin - Nakamoto consensus
  – Byzantine SMR
  – Unknown set of participants

• Works with benign (correct) servers only
  – No resilience to bugs, intrusions
• Is this perfect?
  • What happens in case of a network partition?
• Reconciliation
  • The app developers know better, let them decide what to do
  • THIS is where consistency is relaxed
  • Spoiler alert: it’s not a bad idea
THE COORDINATOR NODE

• Assigned to each key
• In charge of replicating data N times
• Upon write, sends info to N nodes
  • Returns to client after W (parameter) responses, including self
• Upon read, asks N nodes for data
  • Waits for R (parameter) responses, including self and returns
PUT (KEY, VALUE, CONTEXT)

• Coordinator generates new vector clock and writes the new version locally
• Send to N nodes
• Wait for response from W-1 nodes

• Using W=1
  • High availability for writes
  • Low durability
(VALUE, CONTEXT) ← GET (KEY)

• Coordinator requests existing versions from N
• Wait for response from R nodes
• If multiple versions, return all versions that are causally unrelated
• Divergent versions are then reconciled
• Reconciled version written back

• Using R=1
  • High performance read engine
EXECUTION OF OPERATIONS

• put and get operations

• Client can send the request
  • to the node responsible for the data
    • Save on latency, code on client
  • to a generic load balancer
    • Extra hop
For some services, Dynamo provides the ability to trade-off durability guarantees for performance.

In the optimization each storage node maintains an object buffer in its main memory.

Each write operation is stored in the buffer and gets periodically written to storage by a writer thread.

In this scheme, read operations first check if the requested key is present in the buffer. If so, the object is read from the buffer instead of the storage engine.
EVALUATION