failures

asynch

infinite time to process/send request

you can't detect failures

different heartbeat mechanism

time is bounded

N/W response gen

Heartbeats to detect failures
<table>
<thead>
<tr>
<th>Issue</th>
<th>Detection</th>
<th>Resolutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data imbalance</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Hardware Heterogeneity</td>
<td>Progress reports</td>
<td>Load distribution</td>
</tr>
<tr>
<td>Hardware failures</td>
<td>Heartbeats</td>
<td>Restarting nodes</td>
</tr>
<tr>
<td>Persistently flakey nodes</td>
<td>Heartbeat</td>
<td>Quarantine</td>
</tr>
<tr>
<td>Slow node</td>
<td>Progress reports</td>
<td>Load distribution</td>
</tr>
</tbody>
</table>

Hedging $\Rightarrow$ slow nodes

- When does it work?
- Should you avoid hedging?
specific policies better/stronger control over performance

Global LB to direct to an initial cluster

- Pure DNS which gives precise control over which cluster to send a client
- ANYCAST: the network decides which cluster. This is based on the n/w distance

This doesn't imply low latency or better BW
LB within cluster

Key goals: even util & consistent mapping of clients

Can provide consistent mapping but will need to store a lot of state.

Consistent hashing

Virtual notes: used to deal with HW heterogeneity

Even distribution

1. Random
2. Modulo hash
3. Round robin
Distributed Hash Tables

(Tapestry / Dynamo / Cassandra)

Route table creation
- Adding new nodes / deleting nodes
- Statistical properties
- Tapestry optimizes for latency

get/set

- Lookups
- Republish
- Guarantees of the routing table
Distributed Hash Tables
(Tapestry / Dynamo / Cassandra)

- Replication (dirty quorum) replicate to the next N nodes
- Virtual nodes for HW heterogeneity
- Conflict resolution
  - Last writer wins
  - App-specific

get/set
Replication (Active, Lazy, Passive)

Changes across shards

Distributed Transactions

Use a specific replication scheme

Copies of shard 1

Copies of shard 2

Lazy

Passive / Raft / Zookeeper

Active

Linearizability

"real time"

Total / FIFO

Causal based ordering

Based on protocol for reads

* reads have a V checkpoint from other AMs to provide it with causally dependent K/V pair
1. Leader election slows things down
2. Used within a cluster

Active

Used across cluster

Lazy

Improving gossip speed

- Increase size of gossip group
- Send msg more frequently
During a network Partition, your protocol picks \( C \) or \( A \)

- Passive / Active \( \Rightarrow \) CP
- Dynamo / Cassandra / lazy rep-writes \( \Rightarrow \) AP
- 2PC / Chandy Lamport \( \Rightarrow \) CA
Dist Transactions

FE → Coordination

TM

Shard 1

Shard 2

ACID

Durability
* Store to disk

Isolation
* Locks

IPC

Provided by underlying shard maintenance mechanism

Pessimistic

Optimistic

3 PMS for a shard
No concurrency

Pessimistic

Lots of concurrency

No locks
Dist File System

DFS

GFS

Industrial consensus (Zookeepers)

metadata (small)

V: data separation

(linearizable)

accessed infrequently

okay to be slow & unscalable

(eventual)

consistent / something weaker

Very fast & parallelizable
GFS

Meta ⇒ stored @ one centralized (with backups) server

Versus

Total/FIFO

Data ⇒ stored on cluster of chunk servers

append ⇒ not consistent

Not optimized for random writes ⇒ concurrent writes leads to inconsistency

Initially designed for Google search

1. large files
2. sequential reads
3. appends
A quorum Apps is a fine as a state machine. Each is written independently. Doesn’t force the apps into a pattern smaller. Zookeeper Service doesn’t need apps to scale independently of app code. Services isn’t a pattern.
Zookeeper Improve reads

1. Client side caching
   - Improve perf
   - but can allow reads of stale data
2. Read from followers
Distributed is about design choices systems & mapping the right choices to your application requirements