Cassandra

Structured Storage System over a P2P Network

Avinash Lakshman, Prashant Malik

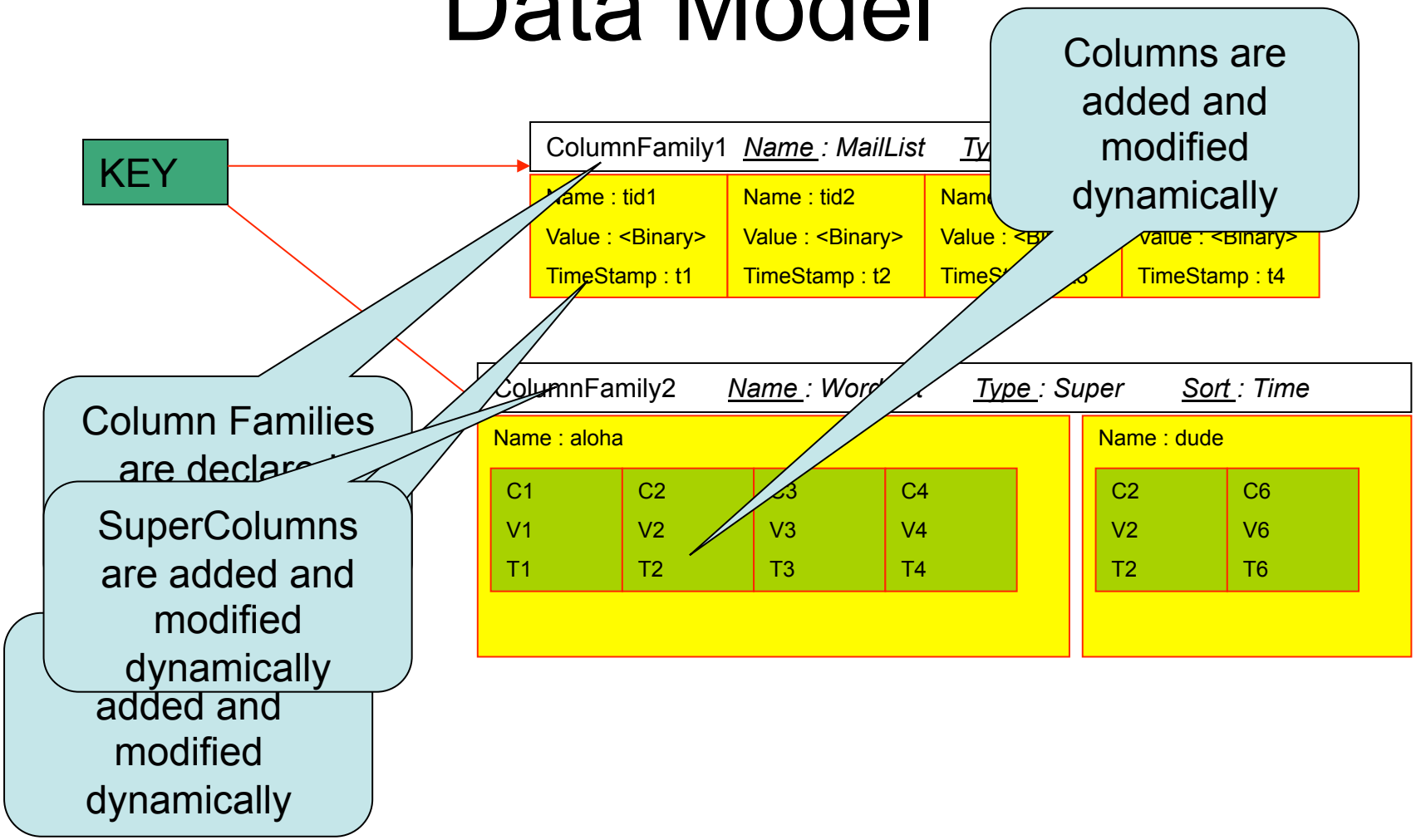
Why Cassandra?

- Lots of data
  - Copies of messages, reverse indices of messages, per user data.
- Many incoming requests resulting in a lot of random reads and random writes.
- No existing production ready solutions in the market meet these requirements.

Design Goals

- High availability
- Eventual consistency
  - trade-off strong consistency in favor of high availability
- Incremental scalability
- Optimistic Replication
- “Knobs” to tune tradeoffs between consistency, durability and latency
- Low total cost of ownership
- Minimal administration

Data Model


KEY
ColumnFamily1 Name : MailList
Name : tid1
Value : <Binary>
TimeStamp : t1
Name : tid2
Value : <Binary>
TimeStamp : t2
TimeStamp : t4
Columns are added and modified dynamically
ColumnFamily2 Name : Word Type : Super Sort : Time
Name : aloha
C1 C2 C4
V1 V2 V3 V4
T1 T2 T3 T4
Name : dude
C2 C6
V2 V6
T2 T6
Column Families are declare
SuperColumns are added and modified dynamically
added and modified dynamically

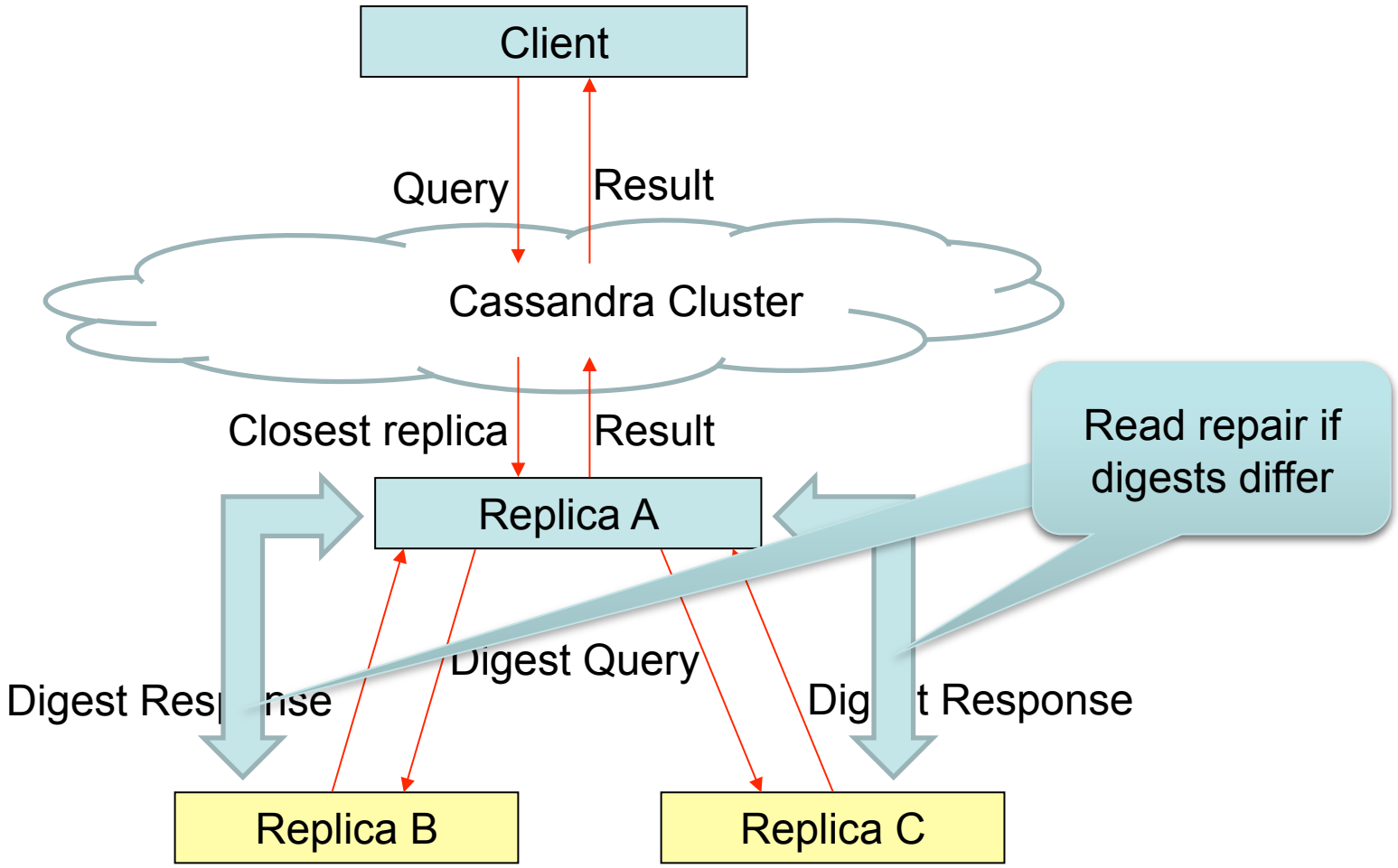
Write Operations

- A client issues a write request to a random node in the Cassandra cluster.
- The “Partitioner” determines the nodes responsible for the data.
- Locally, write operations are logged and then applied to an in-memory version.
- Commit log is stored on a dedicated disk local to the machine.

Write Properties

- No locks in the critical path
- Sequential disk access
- Behaves like a write back Cache
- Append support without read ahead
- Atomicity guarantee for a key per replica
- “Always Writable”
  - accept writes during failure scenarios

Read


Client
Query
Result
Cassandra Cluster
Closest replica
Result
Replica A
Read repair if digests differ
Digest Query
Digest Response
Digest Response
Replica B
Replica C

Cluster Membership and Failure Detection

- Gossip protocol is used for cluster membership.
- Super lightweight with mathematically provable properties.
- State disseminated in O(logN) rounds where N is the number of nodes in the cluster.
- Every T seconds each member increments its heartbeat counter and selects one other member to send its list to.
- A member merges the list with its own list .

Accrual Failure Detector

- Valuable for system management, replication, load balancing etc.
- Defined as a failure detector that outputs a value, PHI, associated with each process.
- Also known as Adaptive Failure detectors - designed to adapt to changing network conditions.
- The value output, PHI, represents a suspicion level.
- Applications set an appropriate threshold, trigger suspicions and perform appropriate actions.
- In Cassandra the average time taken to detect a failure is 10-15 seconds with the PHI threshold set at 5.

Properties of the Failure Detector

- If a process p is faulty, the suspicion level

  $\Phi(t) \rightarrow \infty$ as $t \rightarrow \infty$.
- If a process p is faulty, there is a time after which $\Phi(t)$ is monotonic increasing.
- A process p is correct $\Leftrightarrow$ $\Phi(t)$ has an ub over an infinite execution.
- If process p is correct, then for any time T,

  $\Phi(t) = 0$ for $t >= T$.

Performance Benchmark

- Loading of data - limited by network bandwidth.
- Read performance for Inbox Search in production:

	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Average	26.13 ms	44.41 ms

Lessons Learnt

- Add fancy features only when absolutely required.
- Many types of failures are possible.
- Big systems need proper systems-level monitoring.
- Value simple designs

Questions?