Cassandra - A Decentralized Structured Storage System

Avinash Lakshman and Prashant Malik
Facebook

Presented by Gregory Kesden

Agenda

- **Outline**
- **Data Model**
- **System Architecture**
- **Implementation**
- **Experiments**

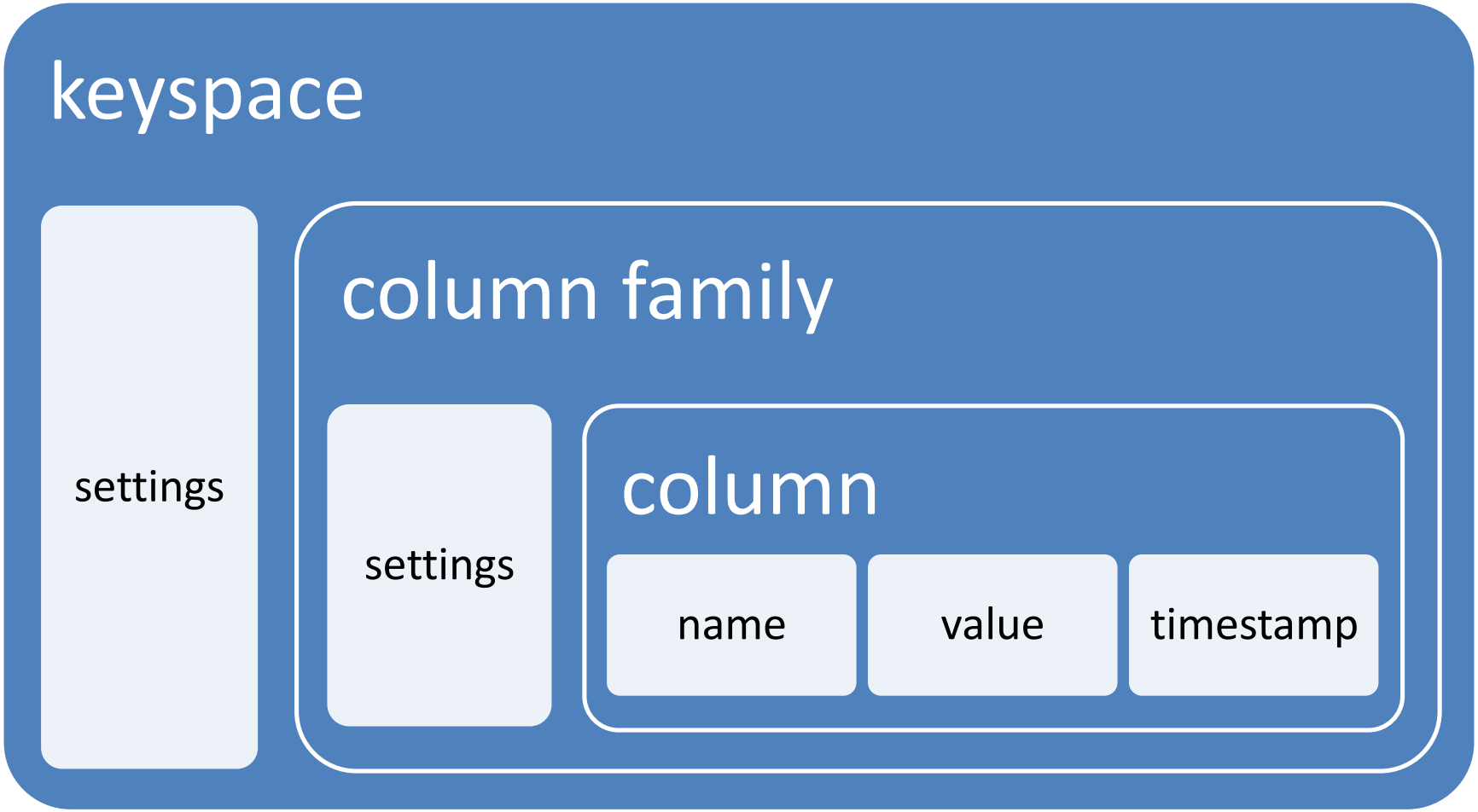
Outline

- **Extension of Bigtable with aspects of Dynamo**
- **Motivations:**
  - **High Availability**
  - **High Write Throughput**
  - **Fail Tolerance**

Data Model

- **Table is a multi dimensional map indexed by key (row key).**
- **Columns are grouped into Column Families.**
- **2 Types of Column Families**
  - **Simple**
  - **Super (nested Column Families)**
- **Each Column has**
  - **Name**
  - **Value**
  - **Timestamp**

Data Model



* Figure taken from Eben Hewitt's (author of Oreilly's Cassandra book) slides.

System Architecture

- **Partitioning**

  How data is partitioned across nodes

- **Replication**

  How data is duplicated across nodes

- **Cluster Membership**

  How nodes are added, deleted to the cluster

Partitioning

- **Nodes are *logically* structured in Ring Topology.**
- **Hashed value of key associated with data partition is used to assign it to a node in the ring.**
- **Hashing rounds off after certain value to support ring structure.**

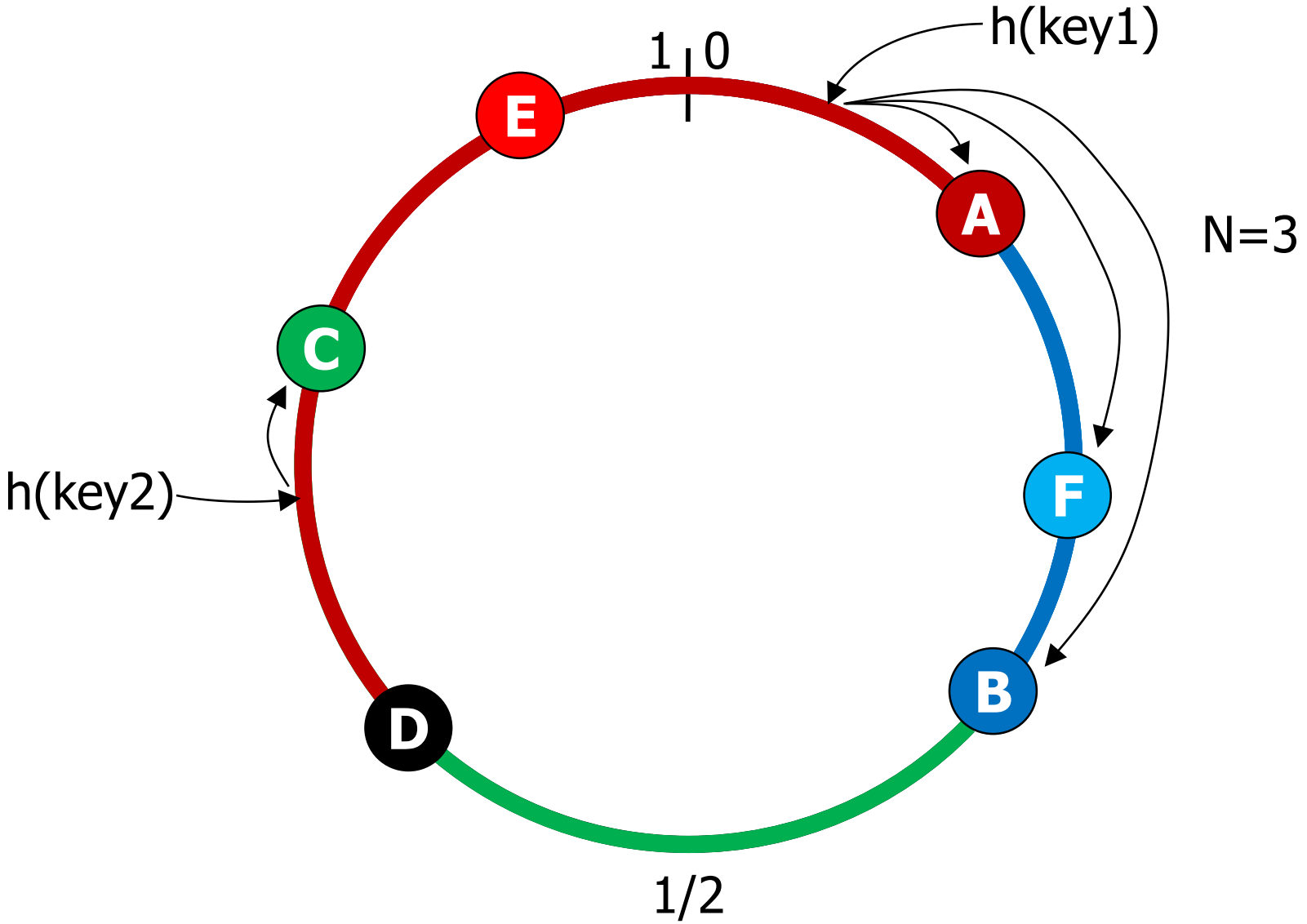
- **Lightly loaded nodes moves position to alleviate highly loaded nodes.**

Replication

- **Each data item is replicated at N (replication factor) nodes.**

- **Different Replication Policies**
  - **Rack Unaware –** replicate data at N-1 successive nodes after its coordinator
  - **Rack Aware –** uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for
  - **Datacenter Aware –** similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

Partitioning and Replication



* Figure taken from Avinash Lakshman and Prashant Malik (authors of the paper) slides.

Gossip Protocols

- **Network Communication protocols inspired for real life rumour spreading.**
- **Periodic, Pairwise, inter-node communication.**
- **Low frequency communication ensures low cost.**
- **Random selection of peers.**
- **Example – Node A wish to search for pattern in data**
  - **Round 1 – Node A searches locally and then gossips with node B.**
  - **Round 2 – Node A,B gossips with C and D.**
  - **Round 3 – Nodes A,B,C and D gossips with 4 other nodes ……**
- **Round by round doubling makes protocol very robust.**

Gossip Protocols

- **Variety of Gossip Protocols exists**
  - **Dissemination protocol**
    - **Event Dissemination: multicasts events via gossip. high latency might cause network strain.**
    - **Background data dissemination: continuous gossip about information regarding participating nodes**
  - **Anti Entropy protocol**
    - **Used to repair replicated data by comparing and reconciling differences. This type of protocol is used in Cassandra to repair data in replications.**

Cluster Management

- **Uses Scuttleback (a Gossip protocol) to manage nodes.**
- **Uses gossip for node membership and to transmit system control state.**
- **Node Fail state is given by variable ‘phi’ which tells how likely a node might fail (suspicion level) instead of simple binary value (up/down).**
- **This type of system is known as Accrual Failure Detector.**

Accrual Failure Detector

- **If a node is faulty, the suspicion level monotonically increases with time.**

  $\Phi(t) \rightarrow k$ as $t \rightarrow k$

  Where k is a threshold variable (depends on system load) which tells a node is dead.

- **If node is correct, phi will be constant set by application. Generally**

  $\Phi(t) = 0$

Bootstrapping and Scaling

- **Two ways to add new node**
  - **New node gets assigned a random token which gives its position in the ring. It gossips its location to rest of the ring**
  - **New node reads its config file to contact it initial contact points.**
- **New nodes are added manually by administrator via CLI or Web interface provided by Cassandra.**

- **Scaling in Cassandra is designed to be easy.**
- **Lightly loaded nodes can move in the ring to alleviate heavily loaded nodes.**

Local Persistence

- **Relies on local file system for data persistency.**
- **Write operations happens in 2 steps**
  - **Write to commit log in local disk of the node**
  - **Update in-memory data structure.**
  - **Why 2 steps or any preference to order or execution?**
- **Read operation**
  - **Looks up in-memory ds first before looking up files on disk.**
  - **Uses Bloom Filter (summarization of keys in file store in memory) to avoid looking up files that do not contain the key.**

I/O Architecture

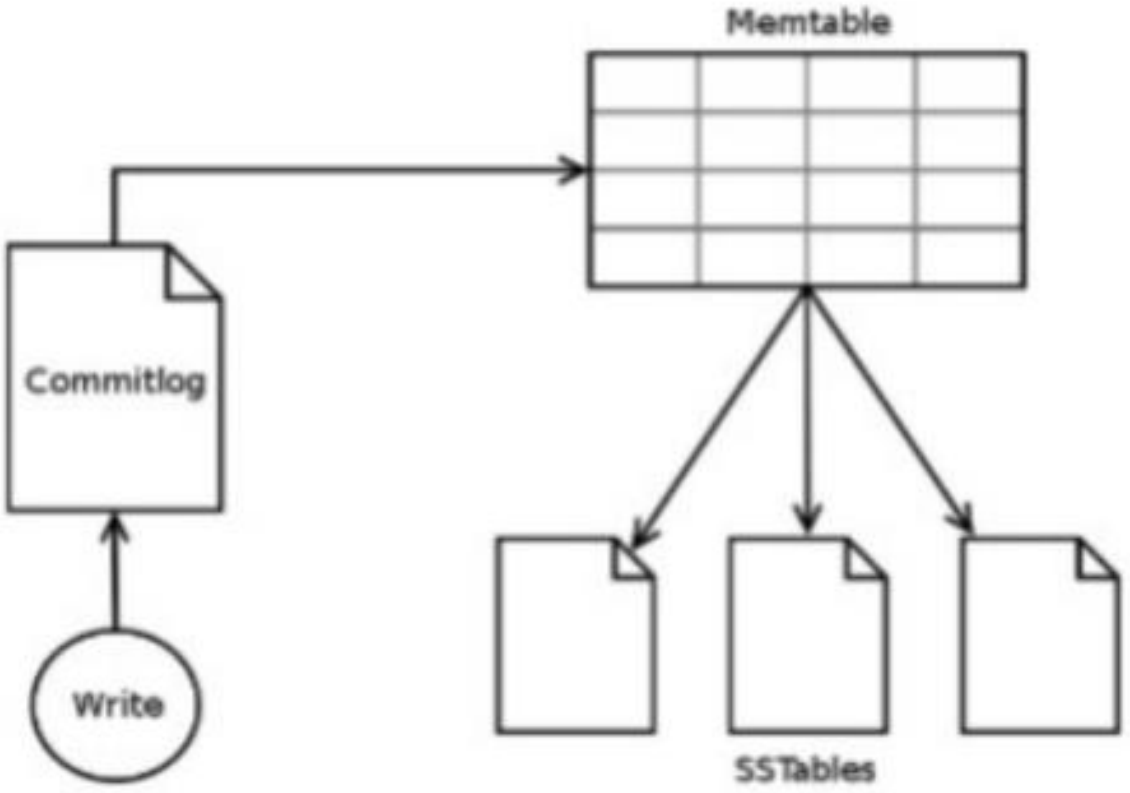
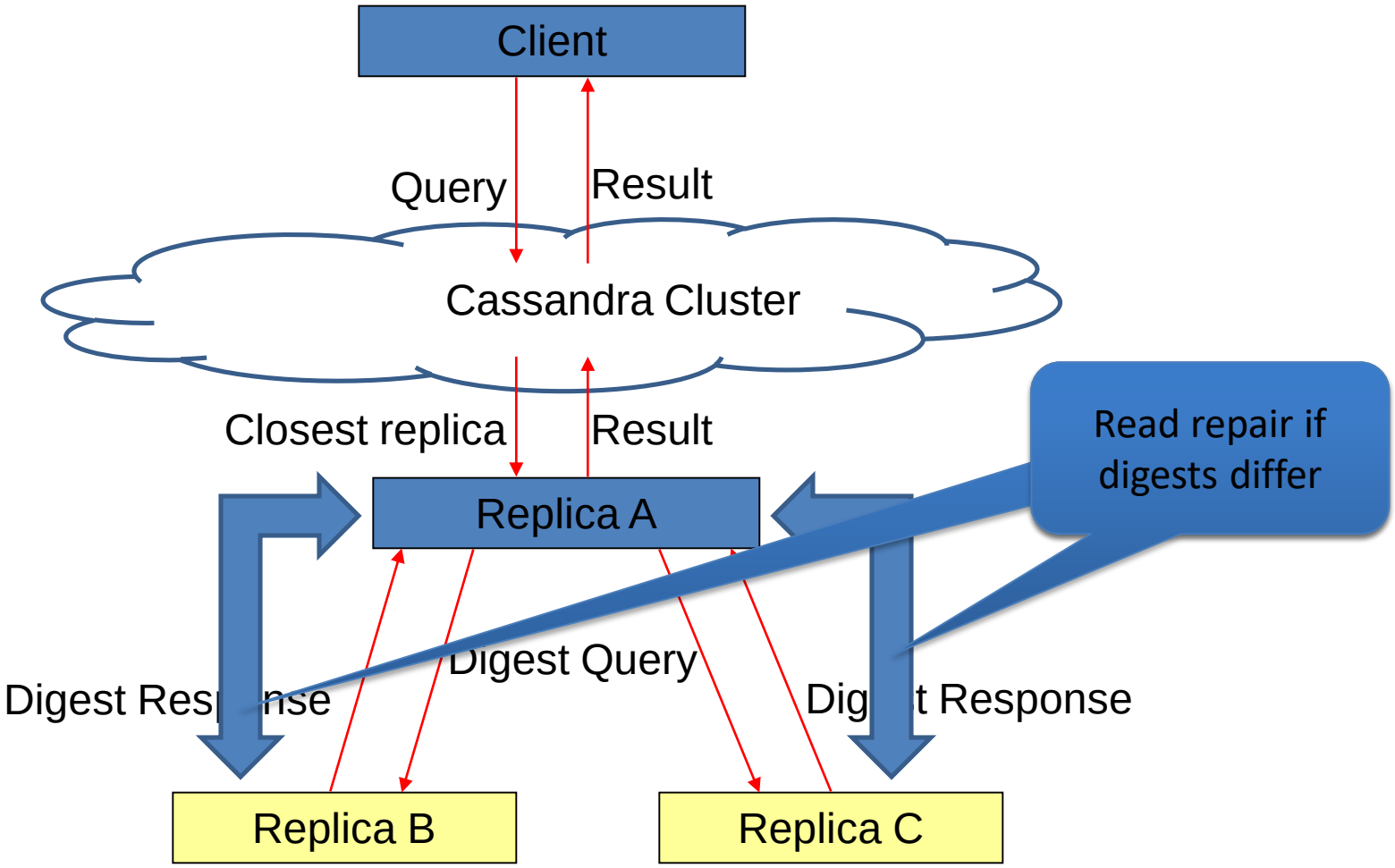


Fig 2: Cassandra I/O architecture

Read Operation



* Figure taken from Avinash Lakshman and Prashant Malik (authors of the paper) slides.

Facebook Inbox Search

- **Cassandra developed to address this problem.**
- **50+TB of user messages data in 150 node cluster on which Cassandra is tested.**
- **Search user index of all messages in 2 ways.**
  - **Term search : search by a key word**
  - **Interactions search : search by a user id**

Latency Stat	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Max	26.13 ms	44.41 ms

Comparison with MySQL

- **MySQL > 50 GB Data**
  Writes Average : ~300 ms
  Reads Average : ~350 ms

- **Cassandra > 50 GB Data**
  Writes Average : 0.12 ms
  Reads Average : 15 ms

- **Stats provided by Authors using facebook data.**

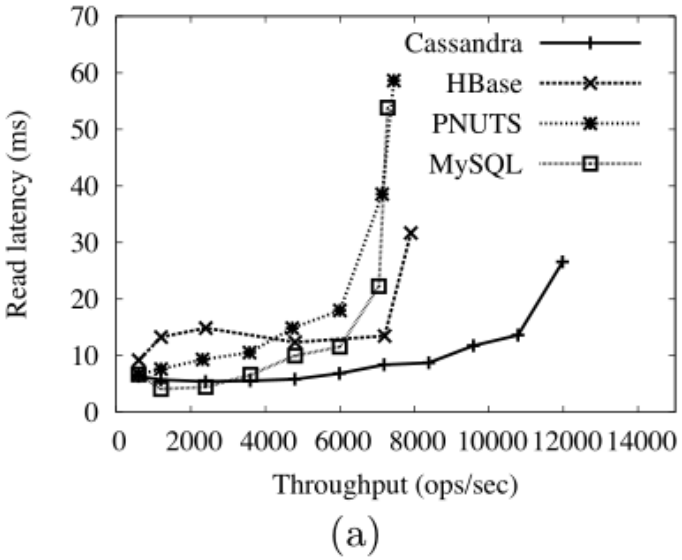
Comparison using YCSB

- **Following results taken from 'Benchmarking Cloud Serving Systems with YCSB' by Brain F Cooper et all.**
- **YCSB is Yahoo Cloud Server Benchmarking framework.**
- **Comparison between Cassandra, HBase, PNUTS, and MySQL.**

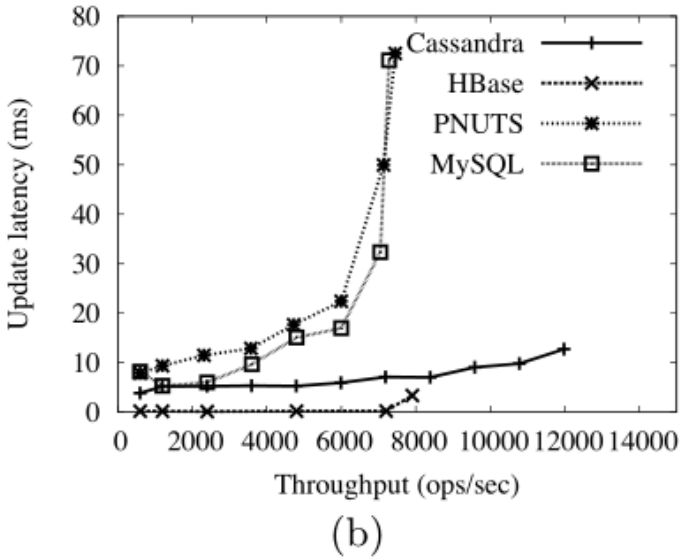
- **Cassandra and Hbase have higher read latencies on a read heavy workload than PNUTS and MySQL, and lower update latencies on a write heavy workload.**
- **PNUTS and Cassandra scaled well as the number of servers and workload increased proportionally.**

Comparison using YCSB

- **Cassandra, HBase and PNUTS were able to grow elastically while the workload was executing.**
- **PNUTS and Cassandra scaled well as the number of**
- **servers and workload increased proportionally. HBase's**
- **performance was more erratic as the system scaled.**



(a)



(b)

Thank You