Partitioning in Cloud Storage

K V SUBRAMANIAM

CHAPTER 5 : REPLICATION – MARTIN KLEPPMAN – DESIGNING DATA INTENSIVE APPLICATIONS

What is a partition

- <u>Definition</u>

- “Mathematically, a partition refers to dividing a set or a number into disjoint, non-empty subsets that together cover the entire original set or sum to the original number.”

- When your dataset is broken into disjoint non-empty subsets.

- Each subset can be independently managed

Why partition

- Cloud applications scale using elastic resources,
  - backend bottlenecks can occur if the data store remains the same.

- Partitioning
  - divides large data into smaller parts distributed across nodes,
  - improving query and IO performance.

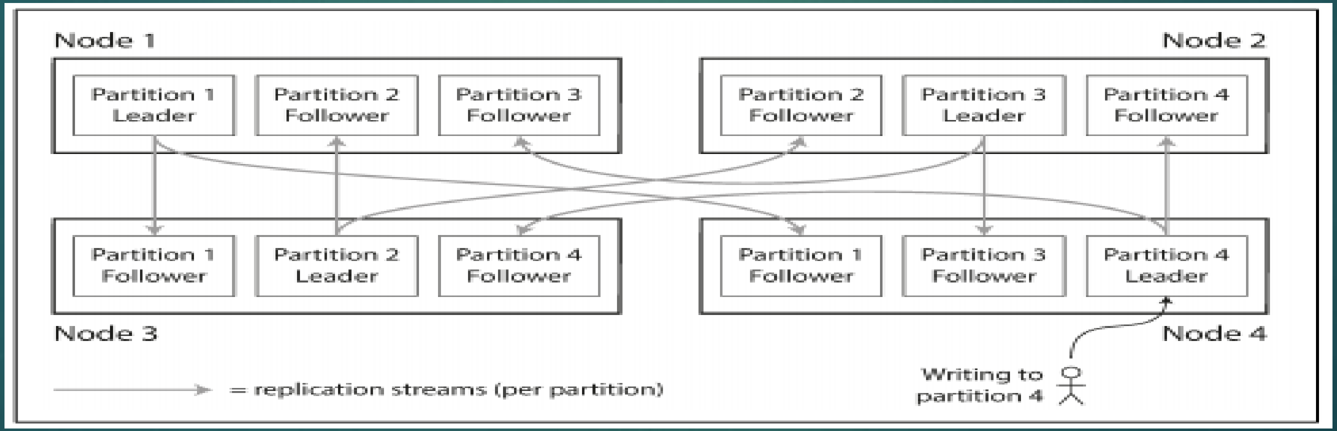
- Each data item belongs to one partition,
  - allowing simultaneous operations on multiple partitions.

- Nodes process their own partitions independently,
  - enabling throughput scaling by adding nodes.

- For example, spreading data across multiple disks enhances overall IO performance.

Leader based replication

- Large, complex queries or heavy IO tasks can be parallelized across multiple nodes.
- Partitioning is often paired with replication, so each partition's copies exist on several nodes for fault tolerance.
- A single node can hold multiple partitions.



Goal of partitioning

- **Objective :**
  - spread data across nodes
  - so that query load is evenly distributed
- Fair share – total load/#partitions
  - Ideal case when data is equally distributed
  - Results in linear scaling with #partitions
- **Skew:** when some partition has more data than others
  - Increased skew reduces effect of partitioning
  - Extreme case – single node has all the data and load
- **Hot Spot**
  - Partition with disproportionately high load

Class exercise

- Consider a scenario where we have 1000 records and 4 nodes. Each node should be responsible for ¼ of the records.
- Design a scheme to partition the data into the 4 nodes

Partitioning schemes

- Key Range Partition
- Hash Partition

Key Range Partition

- **How**
  - Non-overlapping ranges of key to define subsets
  - Partition to node mapping for query handling
- Benefit
  - Easy for clients to process
- Prone to skew
  - Certain ranges can be more frequent
- Used by Hbase
- Can cause hotspots if key not chosen carefully – example: timestamps

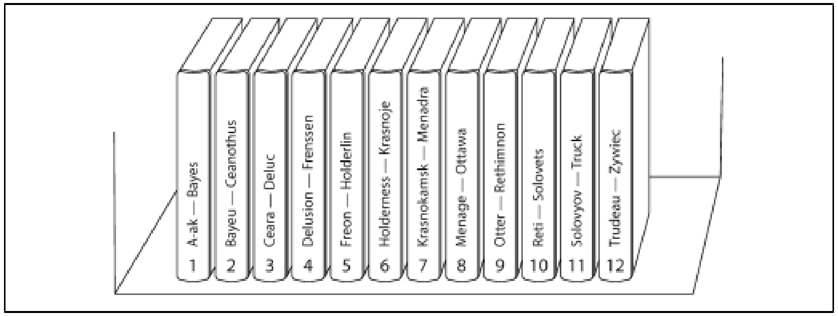


Figure 6-2. A print encyclopedia is partitioned by key range.

Hash based partition

- **How**
  - HashFn(Key)
  - Assign range of hashes to a partition.
  - Good hash function distributes keys evenly
- Benefit
  - Solves the skew problem
- Disadvantages
  - Range queries are more difficult to implement

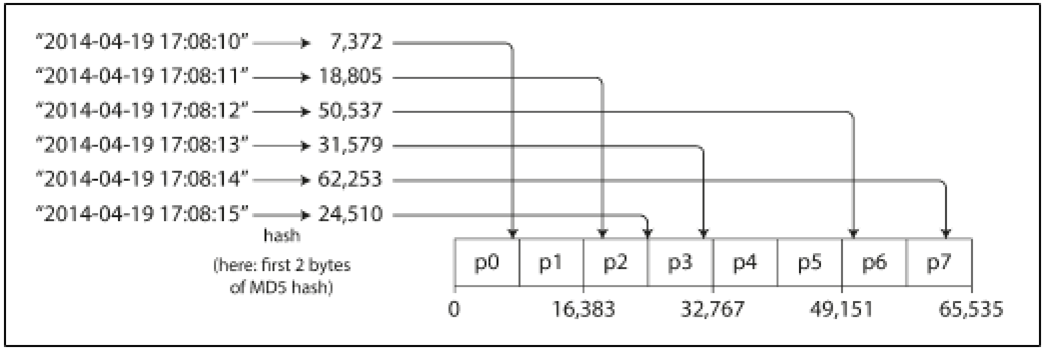


Figure 6-3. Partitioning by hash of key.

Rebalancing Partitions

- Due to changes over time
  - Query load increases
  - Data set size increases
  - Failures to nodes
- Requires data to be moved from one node to another
  - Requirements
    - Minimal data transfer
    - Read/Writes should continue while rebalancing is in progress
    - After rebalancing, skew is minimized

Rebalancing strategies

- What to avoid?
  - Hash Mod n
    - Rather assign ranges
    - Otherwise there were will be movements across all nodes
- What will work?
  - Fixed number of partitions
    - Create more partitions than nodes
    - Map multiple partitions to a node.
    - Entire partition is moved

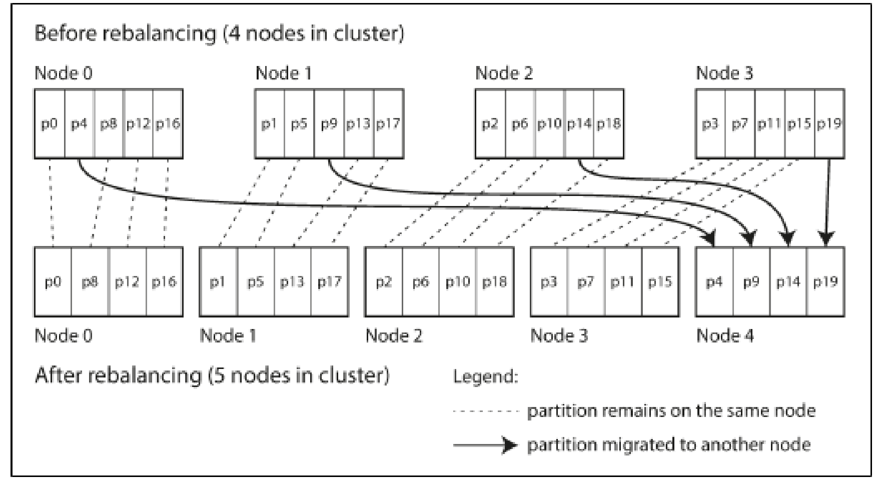
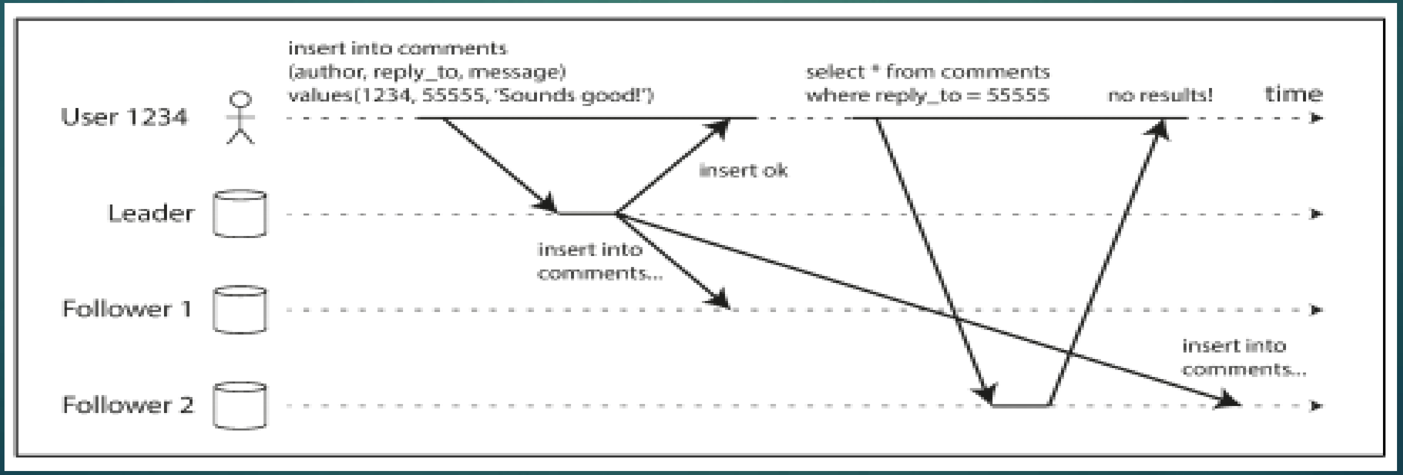


Figure 6-6. Adding a new node to a database cluster with multiple partitions per node.

Reading your own writes consistency

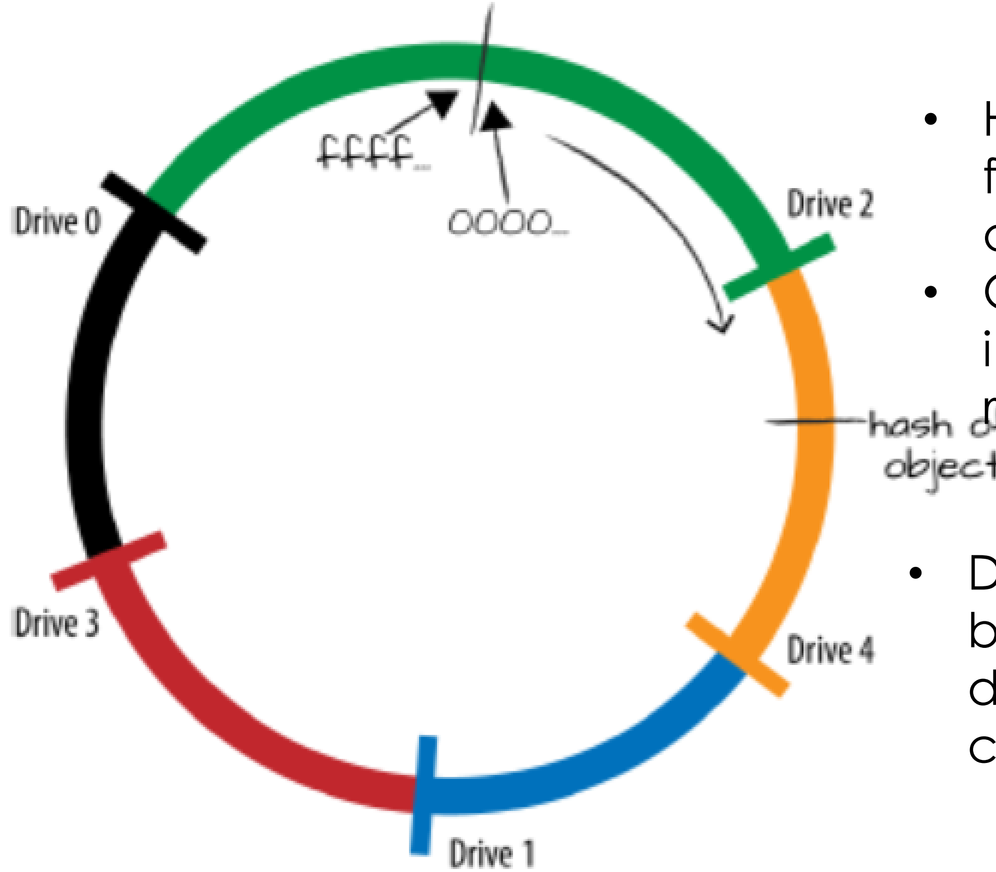
- Read-after-write consistency (read-your-writes consistency)
  - ensures that users always see their own updates when they read data again.
- It does not guarantee
  - updates from other users will be immediately visible.



Other partitioning strategies

- **Dynamic partitioning**
  - When partition grow beyond a threshold
  - Split it in two.
  - Can also be merged in case of deletes
  - Each node can handle multiple partitions.

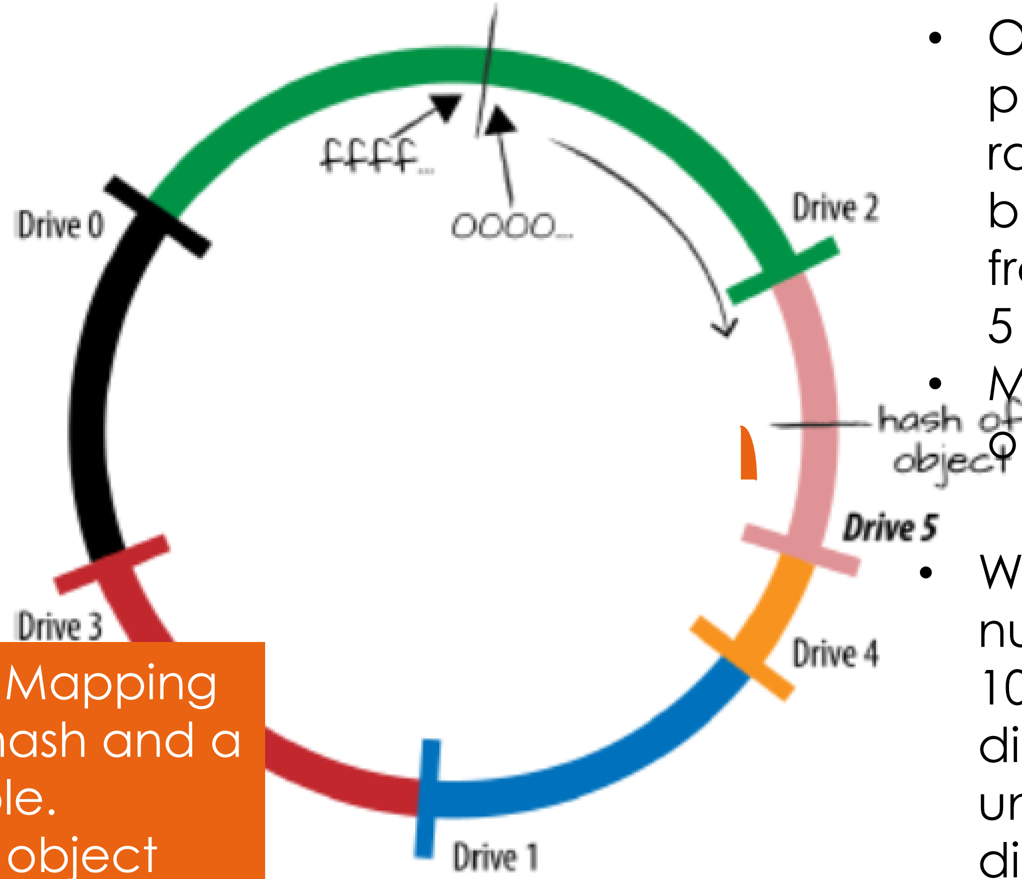
Ring Consistent Hashing Basics



- Hash disk id to find position of disk in ring
- Object stored in next disk in ring

- Disk id could be IP address, drive name, or combination

Adding a New Drive



- Objects in purple hash range have to be moved from disk 4 to 5
- Move from only 1 disk
- With large number (say 100) of disks, disks will be uniformly distributed

Consistent Hashing: Mapping between range of hash and a Drive can be variable. Locating drive for a object requires hashing and then looking for the next drive clockwise

Modified Consistent Hashing

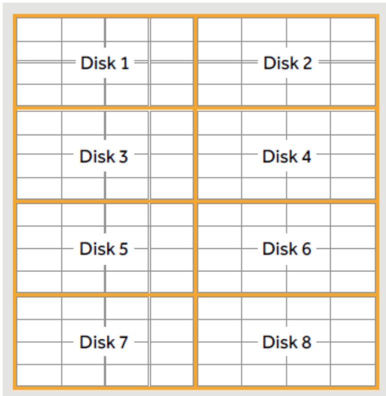
- <u>Problem</u>: May take some time to move a hash range
  - Objects may need to be marked unavailable when hash range is in transit
  - Locating an object also takes time.
- Solution
  - Divide hash range into *partitions (fixed size)*
    - <u>These are not disk partitions</u>
  - Move partitions from one disk to another

MCH: Partition Power

- Total partitions in cluster = $2^{\text{partition power}}$
- If partition power = 15 ,
  - Total partitions $2^{15} = 32{,}768$.
- Those 32,768 partitions mapped to the drives.
- Number of drives might change
- Number of partitions is fixed
  - Hash(object) → partition
  - Partition number does not change with increasing #drives
  - Only some partitions are moved to the new drive

Swift Partitions - 1 Node

Node 1



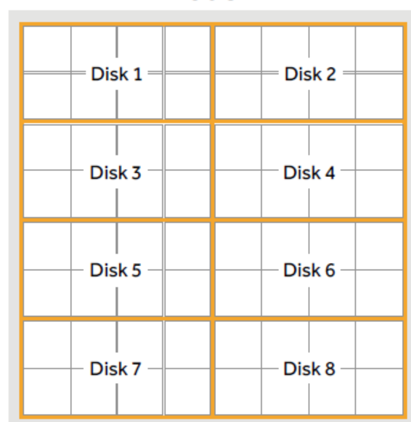
8 Disks - 16 Partitions/Disk

Example:
Assuming equally weighted disks.

8 * 16 = 128 partitions

Swift Partitions - Adding A Node: Partitions Are Reassigned

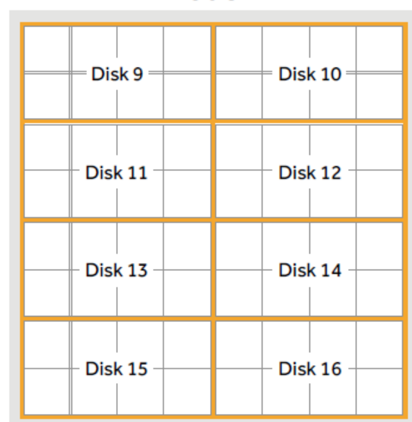
Node 1



8 Disks - 8 Partitions/Disk

+

Node 2

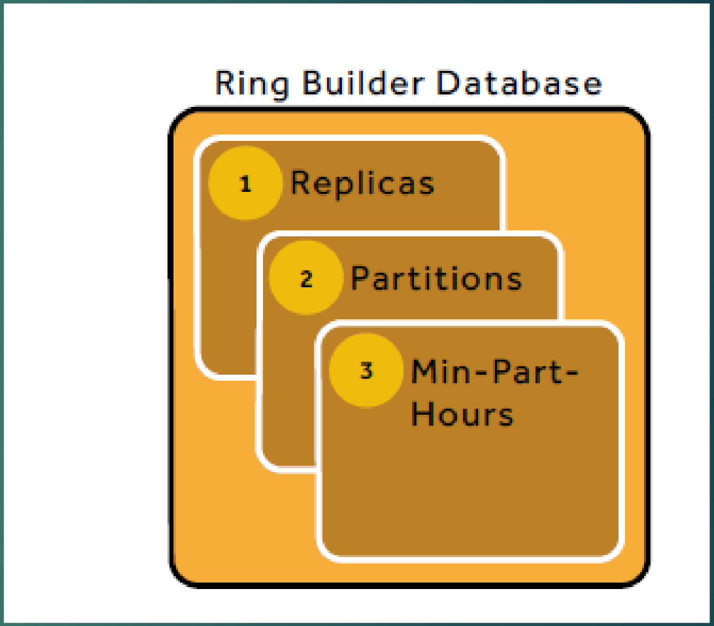


8 Disks - 8 Partitions/Disk

16 * 8 = 128 partitions

MCH: Replication

- Partitions are replicated, not files
- Replica count determines number of replicas
- *Unique-as-possible* algorithm used
  - Replicas are placed as far from each other as possible
  - For example, if two regions are tied for least-used region, replicas are placed in two regions
- This process is called *ring-building*



https://www.swiftstack.com/blog/2013/02/25/data-placement-

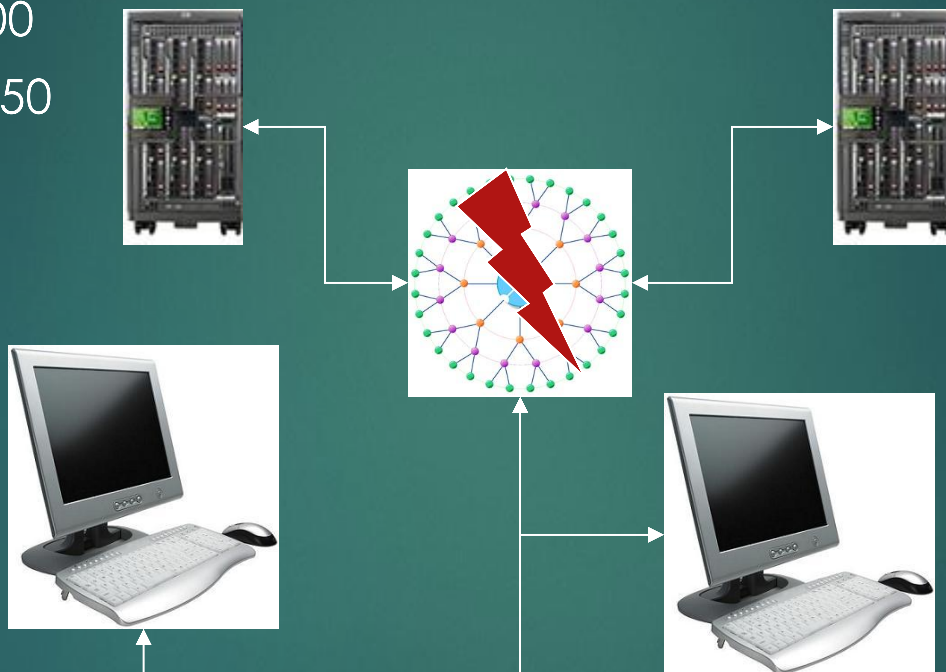
Consistency and Network Partitioning
2-node example

Akash 100

Akash 100

Agni takes 50

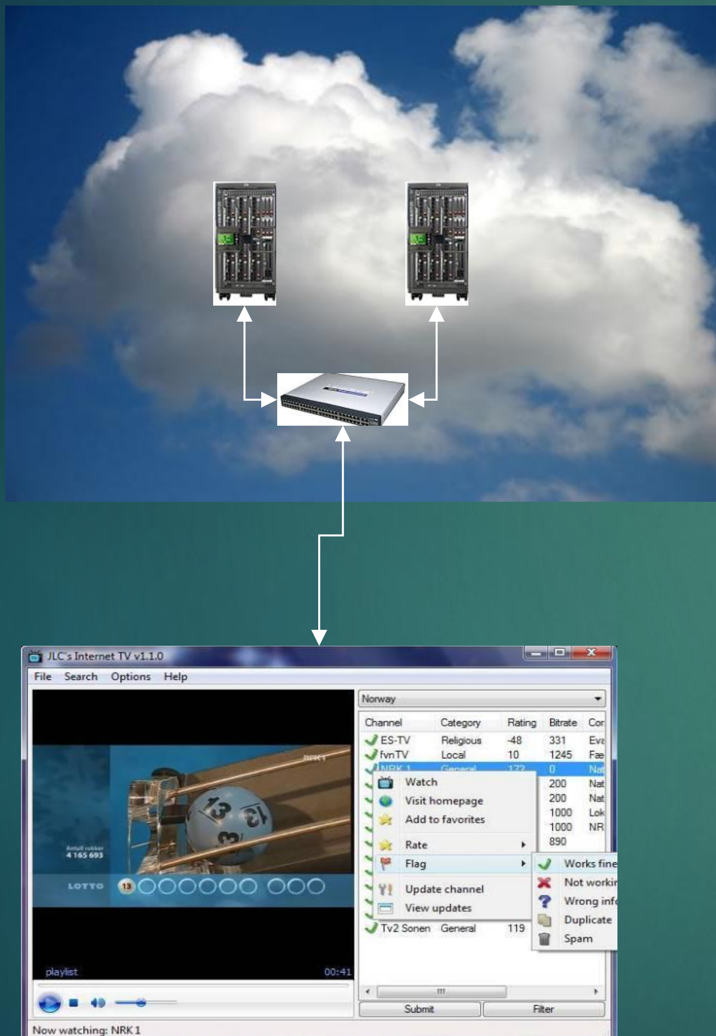
Akash 50



4. Bring servers down (no availability)

tly (inconsistent) OR

Practical example: Netflix

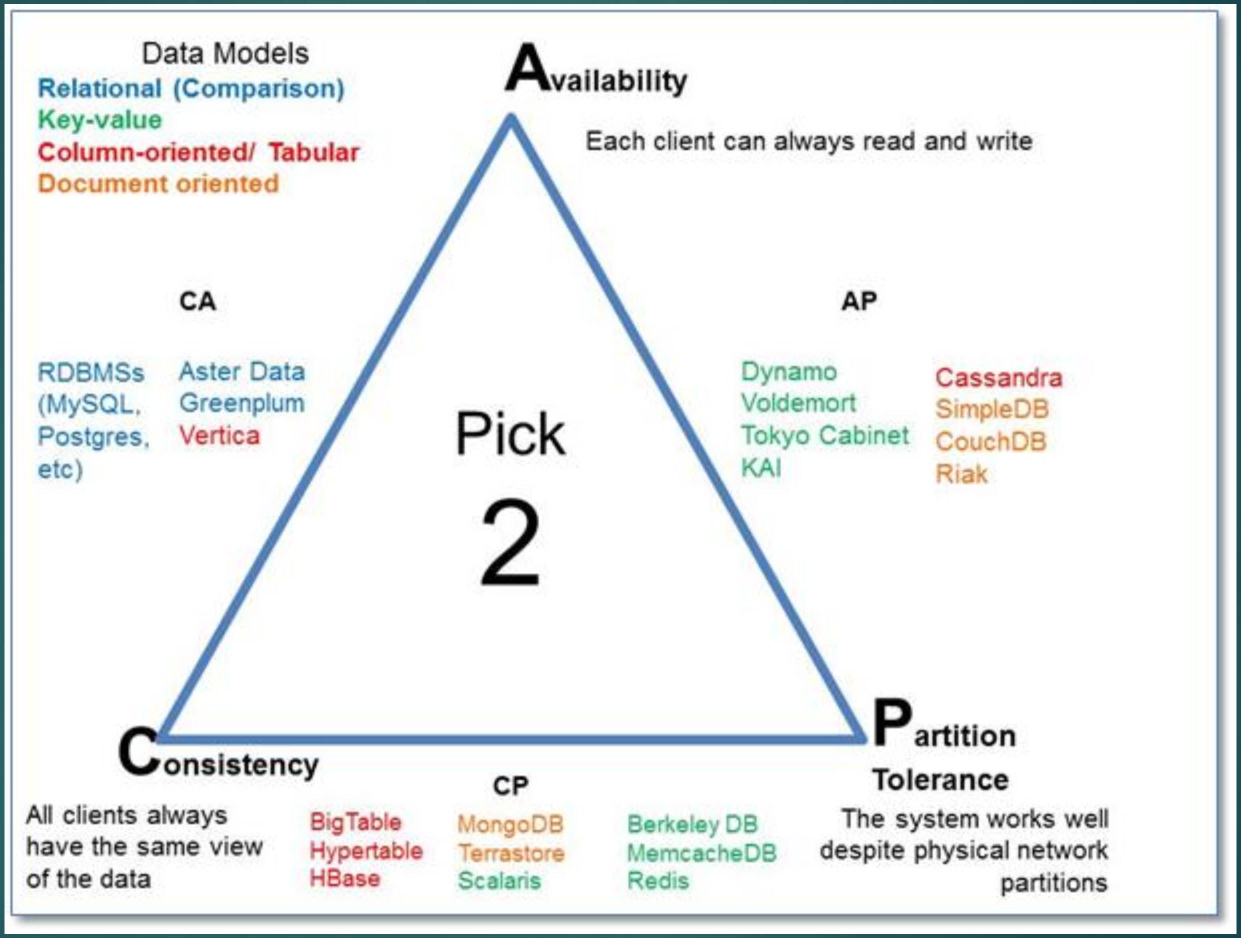


- Netflix: video on demand over the Internet
- Runs on Amazon cloud
- Consider the following scenario
  - User at TV updates list of favorites
  - Load balancer sends update to server 1
  - Set top box requests favorites list
  - Load balancer sends update to server 2
  - Is the returned result consistent? Depends!
- Comparing NoSQL Availability Models by Adrian Cockcroft, http://perfcap.blogspot.com/2010/10/comparing-nosql-availability-models.html

Implications of CAP Theorem

- Conventional view
  - Network partitioning will happen
    - Includes high latency
  - Cloud applications have to deal with either non-availability or inconsistency
- Contra view
  - CAP theorem focuses only on one cause of storage down time – human error, application error are equally important
  - Trade-offs should depend on system – mainframe or Windows
  - Partitions rare – not good to make general design decision based upon a rare case
  - Next-generation dbs are much faster – CAP theorem irrelevant

CAP theorem: diagram



http://blog.flux7.com/blogs/nosql/cap-theorem-why-does-it-matter

Other replication architectures

- **Multi Leader Replication**
  - Use cases
    - Multi Data Center operation
    - Clients with offline operation – calendar apps on mobile phone
    - Collaborative Editing – Google Docs

- Leaderless Replication
  - No master node. Any node can potentially be master – Cassandra, Dynamo DB

Request Routing

- How does a client know which partition to get data from?
- Strategies
  - Client aware of partitioning strategy – so sends request directly
  - Client sends to routing tier – which routes to correct node
  - Client talks to any node which forwards the request.

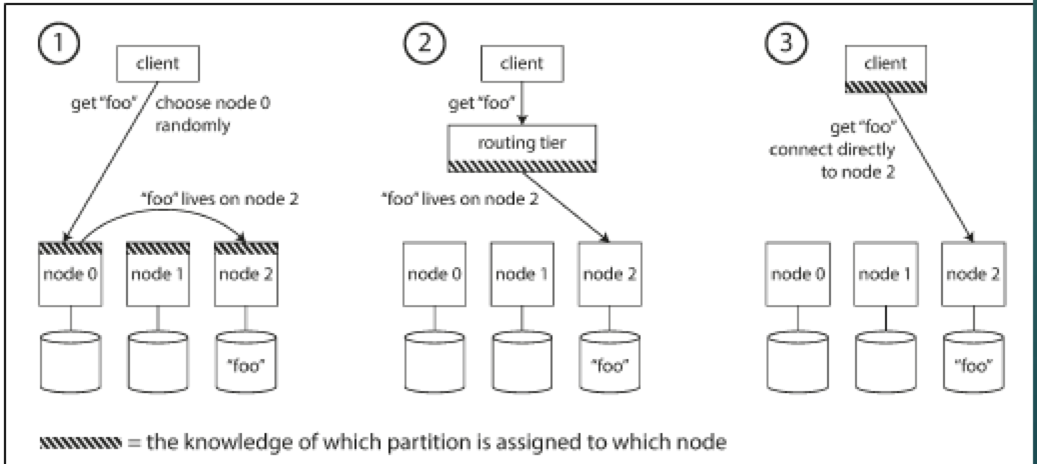
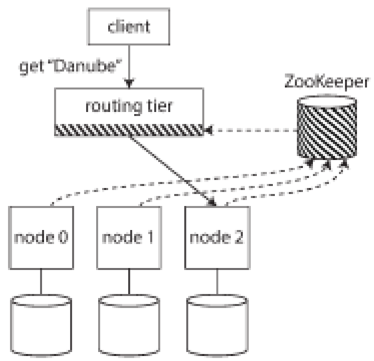


Figure 6-7. Three different ways of routing a request to the right node.

Request Routing

- Use Zookeeper to coordinate knowledge of partition to node mapping



Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

= the knowledge of which partition is assigned to which node

Figure 6-8. Using ZooKeeper to keep track of assignment of partitions to nodes.