

Module 16 - Lecture 13

Data Placement

Strategies

Refer Slide Time :(0: 13)

Data Placement Strategies

- **Replication Strategy:**

1. SimpleStrategy ✓
2. NetworkTopologyStrategy

1. **SimpleStrategy:** uses the Partitioner, of which there are two kinds

1. **RandomPartitioner:** Chord-like hash partitioning ✓ *Recommended - uniform across servers*
2. **ByteOrderedPartitioner:** Assigns ranges of keys to servers.

- Easier for **range queries** (e.g., Get me all twitter users starting with [a-b]) *- hot spots*

2. **NetworkTopologyStrategy:** for multi-DC deployments

- Two replicas per DC ✓ *Path Ex-3 DCs*
- Three replicas per DC
- Per DC ✓
- First replica placed according to Partitioner ✓
- Then go clockwise around ring until you hit a different rack ✓ *3 DCs = 6 replicas = 9 replicas - Rack failure tolerance*

Data placement strategies, replication strategy is of two types, simple strategy and network topology strategy. In the simple strategy uses the partitioner, of which there are two kinds of partitioner, one is called, 'Random Partitioner' and the other one is called, 'Byte Order Partitioner'. Random partitioner is caught like has, partitioning and where is the byte order partitioner assigns, the ranges of each to the servers and it is, easier for the range queries for example, let me call, let me call, all the twitter users starting with a to b. But, instead of going for a byte order partitioner, random partitioner, is a recommended partitioner, why because, random partitioner, partitions the keys, uniformly across all the servers and it is, more efficient to use the random petitioner, whereas the byte order petitioner, will have the various issues that after So much, of So many, number of storage and operations, what will happen it will form the hot spots, out of these right operations. So, hot spots will create a problems and again load-balancing has to be done and so on and so, forth. So, it is recommended that, the random partitioner is being used, in most of the cases, as far as the simple strategy is concerned. Now, another strategy for the replication is called a, 'Network Topology Strategy'. So, network topology strategy, for data placement, requires for multiple data multiple datacenter, deployments, are considers, multiple data center deployments. So, for example, network topology strategy will use two replicas per data center. So, if there are three data centers, then it requires, two data, two replicas per datacenter that means, six replicas, it will be there, across different data centers. Similarly when, there will be three replicas per data center and if we have, three different data center. So, nine different replicas will be there across all the data centers. Now, in per datacenter, replication or data placement, strategies will be doing that first replica; will be placed according to the partitioner and then go clockwise, around the ring, until you hit a different racks. So, is storing the replicas on a different rack, ensures, the rack failure tolerant. Right? Fully at all it ensures the rack failure tolerance. So, that means, what we have seen is that, when we go for the network topology strategy, we are, we are placing our replicas, in different data center, if there is a multiple datacenter deployments or geographically distributed, data centers if they are, there then these replicas are separate or stored across these data centers and this particular things can be achieved using the partitioner, what partitioner will do? For example, there are different data centers, let us assume that in, in your multi

data center deployments there are three data centers, which are shown in the figure. So, what it does is, the first replica; will be placed, the partitioner, will place the first replica here, first replicas will be placed according to the partitioner, let us, use the two replicas for data center. So and the other replicas will be, will be placed on the ring and the second replica; will be will be placed. So, that this replica one and two this replica one and two, they resides on a different rack. Similarly this partitioner will also, store on these two replicas on the other data center and also, on other datacenter. Now, you see that different datacenters have a ring and there will be a coordinator, which will coordinate among themselves. So, this particular partitioner has to, contact to these coordinators and they will in turn, not carry out, this network topology strategy-based, the, the data placement, in this manner.

Refer Slide Time : (6: 26)

Snitches

- **Maps:** IPs to racks and DCs. Configured in cassandra.yaml config file
- **Some options:** ✓
 1. • **SimpleSnitch:** Unaware of Topology (Rack-unaware) ✓
 2. • **RackInferring:** Assumes topology of network by octet of server's IP address ✓
 - 101.102.103.104 = x.<DC octet>.<rack octet>.<node octet> ✓
 3. • **PropertyFileSnitch:** uses a config file ✓
 4. • **EC2Snitch:** uses EC2. ✓
 - EC2 Region = DC ✓
 - Availability zone = rack ✓
- Other snitch options available

Handwritten notes:

Ex- 101.102.103.104 → Same DC
 Cassandra Design
 ✓ 1. Key to server mapping
 ✓ 2. IP to Rack/DC mapping
 map IPs to Rack/DCs Snitches

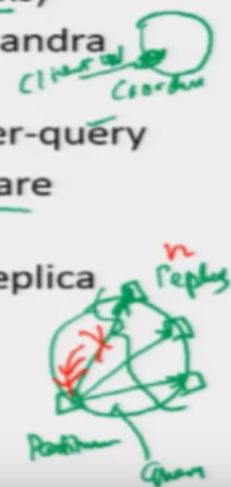
Now, the next issue, in this system is called the, 'Mapping of IP' to the to the wrap, racks and data center. So, the first thing is, about key to the server. So, this we have seen key to the server mapping, in the previous slide. Now we are, going to see, the another design issue, which is called, how the IPS are to be mapped, to the racks and data center. So, the mapping of IPs, to the rack and datacenter, is done through the technique which is called a, 'Snitches'. So, this particular configuration, is can be done, in a file which is called a, 'Cassandra Dot Yaml' configuration file and let us see, these technique, which is called the, 'Snitches' which does this mapping of IP to, the racks and data center. So, there are different type of snitches the, the first type of snitch, is called the, 'Simple Snitch', simple snitch is unaware of the topology and it is rack in aware. So, that means it will, it will be done, in a manner, which will be a random, assignment and doesn't have any inference or doesn't have any information, about from looking from the IP addresses, where it is going to be mapped on, it's called, 'Simple Snitch'. The other type of

snitch is called the, 'Rack Inferring Snitch' which assumes, the topology, of the network by an octet, of server IP addresses. So, for example, if this IP address is there, one zero one point, one zero two point, one zero three point, one zero four, this inverts that the, the first octet, is not going to be used for any inference, the second octet, represents the data center address and third octet, represents the rack octet and the fourth one is the node octet. So, if let us say that for example, if let us say the, the IP addresses, is let us say, 1 0 1 point, 1 0 2 & 1 0 4 point, 1 0 2 that means both these IP addresses, they are mapped to the same data center. Similarly we can see that, 1 0 2 point, 1 0 3 & 1 0 4 that means within that data center, they are referring to the different racks and within the rack, if let us say that, the fourth octet becomes the, the node octet. So, this way, the IP addresses, will be able to infer about the topology, if it is, used in this signature which is called, 'Rack Inferring Snitch'. Another type of snitch is called the, 'Property File Snitch' which uses, the configuration file and fourth type of the snitch is sued, in the ec2 snitch and which is used in easy to type of storage system, that is the cloud storage system. So, easy to has, to kind of inferencing, one is thee c2 region, which will tell about the which data center is used to store the data, the other is called, 'Availability Zone' and which will tell, about which rack uses, which rack is used to store the data. So, if it is, used the easy to based the cloud system. There are many various other snitches options are available, so we have just introduced some of the most commonly used snitches.

Refer Slide Time :(11: 02)

Writes

- Need to be lock-free and fast (no reads or disk seeks) ✓
- Client sends write to one coordinator node in Cassandra cluster
 - Coordinator may be per-key, or per-client, or per-query ✓
 - Per-key Coordinator ensures writes for the key are serialized ✓
- Coordinator uses Partitioner to send query to all replica nodes responsible for key
- When X replicas respond, coordinator returns an acknowledgement to the client
 - X? (circled in red)



In used in the Cassandra. Now, we are going to see the operations, which are supported in the Cassandra in the form of write operations. So, as we have seen that, the writes has to be lightning-fast, writes which is supported by the Cassandra, because it is write a heavy, workloads. So, let us see how the Cassandra supports the write operations. So, the writes need to be log free and fast, hence it not require the right does not require, to be, to read or to perform a disk access, before performing the right operations and the

cursor, the client ends the right, to one of the coordinate or node in the Cassandra cluster, like we have seen, in the previous slide that one of the, one of the server, becomes the coordinator and the client, will send the right operation, to the coordinator and the coordinator, will in perform this right operation. So, coordinator may be a per key basis or a per client, basis or a per query basis, it depends upon different use cases and applications and all these variations, are possible for the coordinator. So, perky coordinator ensures the right for, right for the keys are serialized. So, that means for perky, coordinator will ensure that the rights, all the rights which performed, over that particular coordinator for per key basis, they are all serialized and this ensures, the consistency in some of the cases that we will see, in Some of the applications where it is, wherever it is required to be useful. The next, thing is about the coordinator uses the partitioner, to send the queries to all the replicas nodes, responsible for the key. So, this particular coordinator will now, send using partitioner, to send the queries, to all the replicas responsible for the key, where the keys are restored. Now, when X of these replicas, out of them, if X, if out of them, out of n, let us say n replicas and X replies back, what is this X number is when X replies or responds the coordinator returns an acknowledgment to the client. So, what is this value of x? Which is acceptable, that we will see in the further slides?

Refer Slide Time : (13: 58)

Writes (2)

- **Always writable: Hinted Handoff mechanism**
 - If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
 - When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).
- **One ring per datacenter**
 - Per-DC coordinator elected to coordinate with other DCs
 - Election done via Zookeeper, which runs a Paxos (consensus) variant

Handwritten notes:

- Always writable - → hinted handoff mechanism
- zookeeper - PAXOS (consensus protocol)

Now, right operation we are seeing that, this way always writable, that means right operation, always is performed and which applies our using the hinted, handoff mechanism. So, what is this hinted handoff mechanism? To basically which uses, the which is used for the right operation that we are going to see now. So, if a replica is down, the coordinator rights to all our replicas, let us understand the same diagram that if this is a partitioner and these are all let us say there are three different replicas and if, this wants to write. So, if any of these replicas, let us say this replica is down, the coordinator rights to all other replicas and keeps the right, locally until the down replica comes up. So, for this, replica which is down, the coordinator in, coordinator himself, coordinator himself keeps, the values, at its end and whereas it will

send it, to all other replicas, these values, if the replica is down, the coordinator rights to all other replicas and keeps the right, locally down, locally until the no replica comes up and backs up. When all the replicas are down, let us say then, then the coordinator, will buffer, all the right operations at this end, and, and wait for up to the few hour still, these replicas which are down, they may come up and they may get the updates. So, this is called, 'Hinted Handoff Mechanism' and which is used, to perform this always writable and irrespective of the replicas are down or not, some are down or all are down, even than the right, is performed and using the hinted handoff mechanism. So, we have explained that, what do you mean by always, right able. So, in this particular scenario using hinted handoff and of mechanism. Now, as far as, we know that, there is one ring per data center, So, per a datacenter coordinator, is elected to coordinate with other data center and this election is done via the zookeeper, which runs, a paxos, protocol. So, again we have, already seen and discussed for example, this is a three data center. So, every data center, will have a coordinator and they may these coordinators will communicate with each other. So, this coordinator if they are, they are elected, using the zookeeper and the zookeeper, zookeeper runs, a Paxos, consensus protocol.

Refer Slide Time :(17: 27)

Writes at a replica node

On receiving a write

1. Log it in disk commit log (for failure recovery)
2. Make changes to appropriate memtables
 - **Memtable** = In-memory representation of multiple key-value pairs
 - Typically append-only datastructure (fast)
 - Cache that can be searched by key
 - Write-back as opposed to write-through

Later, when memtable is full or old, flush to disk

- Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key
- *SSTables are immutable (once created, they don't change)*
- Index file: An SSTable of (key, position in data sstable) pairs
- And a Bloom filter (for efficient search)

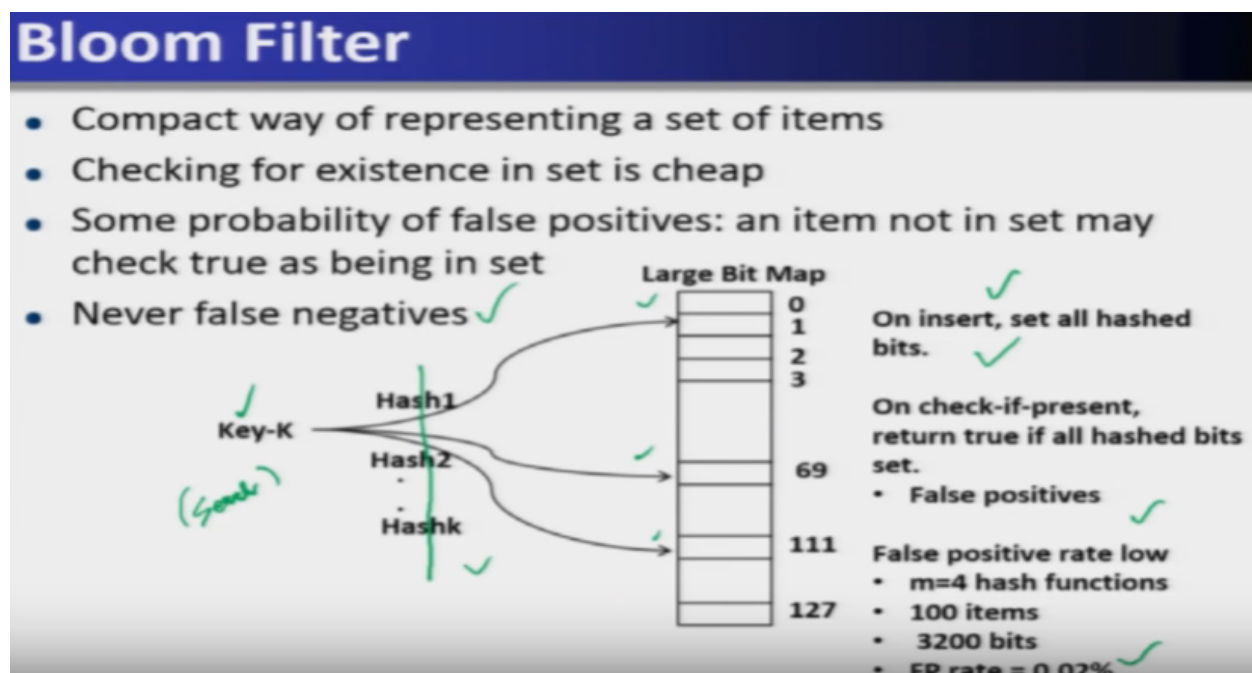
Cache memtable	Disk SSTable
----------------	--------------

↑ Bloom filter for efficient search

Now right said the replica nodes, we are going to see the details of it. Now, when a right is performed, at the replica. So, it will be done in this following manner, on receiving a right, it will be first logged, in to the disk commit log, to ensure the reliability and in case, if it is a failure, of any kind then it can be recovered, from the failure, that is why, the it has to be first log into the disk commit log. Now then, it make changes to the appropriate Memtable. So, Memtable is an in-memory representation, of multiple key value pairs and typically, this particular Memtable, has append only data, structure therefore it is fast, to write, in the Memtable and there is a cache that can be searched, by the key. So, this is Okay? This is maintained in a cache and therefore it can be searched, by the key therefore these operations, writing and

searching, is become fast, if it is, done through the Memtable. Now, in Memtable, the operation is, the right back, that means it will be stored, in this in memory Memtable and then later on, it will be, written on the on the disk, whereas another, approach which is called a, 'Right Through' that means if it is, written to be on right, is to be done on the disk or to make a persistent, then basically it is going to be a time taken and that process is called, 'Right Through'. But, Cassandra by default supports the right back. So, it writes into the map, into the Memtable and that's it. So, hence the write operations are lightning-fast, by this method, of using cash to, do the to, to write the multiple key value pairs. Now, when the Memtable becomes full or becomes old then automatically it will be flushed, to the disk at a later point of time. So, the data, file will be, stored in the in the form of, the SS table. So, when it is done, then these data, files are stored in the SS table which is nothing but, a list of key value pairs, which are sorted by the key. So, in the cache, the same thing is maintained that is called a, 'Memtable' and within the disk, this particular same task is maintained in the form of SS table. So, the data, file which is stored in a in an SS table, that is full form is called, 'Sorted String Table' is a list of, key value pairs and which are Sorted by the key. So, SSTables are immutable, that is once created they don't change and they are also, maintained maintaining an index file, so that search becomes fast. So, index file is associated with a, with the SSTable of which will indicate the key and the positions in the SS table pairs. So, that is maintained in the index, for make it the, the, the retrieval of faster one. Now, to make it, more efficient retrieval using SSTable, a bloom filter is, introduced here in Cassandra, bloom filter provides an efficient search, through this SS table.

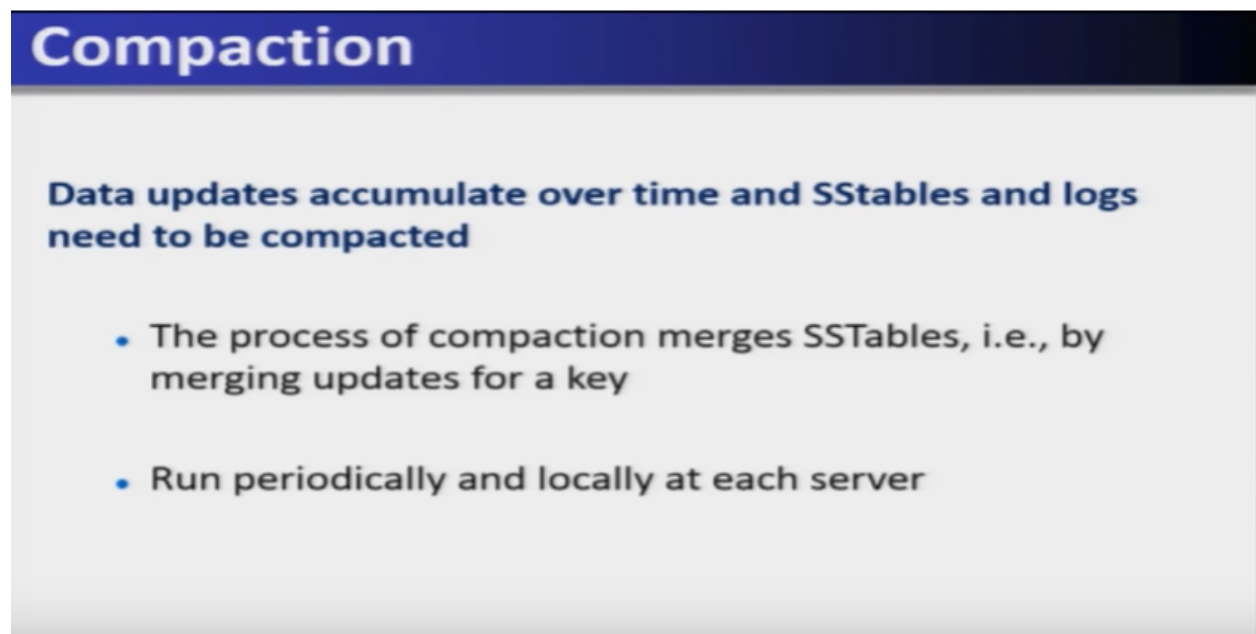
Refer Slide Time :(21: 28)



Let us see, what do you mean by the bloom filter how, it enhances, the efficiency for searching. So, bloom filter, is a compact way of representing a set of items and in this data, structure checking for the existence, in the set whether the element is present, in the set is cheap. So, the sum probability, of false positives are there that means an item, which is not, in there in the set may, may, may it turns out to be too sometimes.

But, there will be an hour false a negatives. So, that means if, the item is not there, always it will say, no with 100% confidence, in some of the cases, even if the item, is not present, even then it will say sometimes, present that is called, 'False Positives' The working of, the bloom filter is like this. So, whenever a key, which required to be means searched, we want to search a key, through the bloom filter, then it has to, go through the K different hash, functions and every hash, function will generate a bit and this bit will be, then inserted onto the large bit map. So, that means, when it is written, that these bits are set and when it is searched, then using hash function it will check, whether all the bits are one, if it is, all the bits are one, then it will say the item is present, although if it is not, then it will be a false positive, if it is, zero sum in one of these bits then obviously, it will be false negative that means it is not inserted. So, when we insert, a key into this, then all these index, are the large bit map, will be set, at all the bits, all the key bits and if you want to check, if it is present or not, these hash bits are being checked, if it is 0 or 1 well if all are, zeros if some are zeros then, we'll be basically it is not present and sometimes, these false positives, will happen and it will say, it is present, although it is not and if we see the performance, of this bloom filter, the false positive, rates are very, very less that is point zero, two percentage is very, rare. So, that means it will give, that means it will check or search, with a high efficiency and happy the high probability,

Refer Slide Time :(24: 09)



Compaction

Data updates accumulate over time and SSTables and logs need to be compacted

- The process of compaction merges SSTables, i.e., by merging updates for a key
- Run periodically and locally at each server

it gives a correct method. Now then, coming back to the Cassandra' again, we have discussed that, that means data, updates will accumulate, over the time and SS tables and the log need to be compacted, compaction, will is a process, which will merge, different SSTables and locks also, together. So, the process of compaction, merges SSTable by merging updates for a key, it says that, the different SS tables, which are immutable, now required to be merged and all the data, has to be made persistent. So, that other operations becomes efficient. So, this particular compaction runs periodically and locally at each server.

Refer Slide Time :(25: 01)

Deletes

Delete: don't delete item right away

- Add a **tombstone** to the log
- Eventually, when compaction encounters tombstone it will delete item

Now, let us see how, the delete is supported. So, the delete operation don't delete the items right away rather it adds a tombstone, to the log and when the eventually, when compaction encounters a tombstone it will delete these items.

Refer Slide Time :(25: 17)

Reads

Read: Similar to writes, except

- **Coordinator can contact X replicas (e.g., in same rack)**
 - Coordinator sends read to replicas that have responded quickest in past
 - When X replicas respond, coordinator returns the latest-timestamped value from among those X
 - (X? We will check it later.)
- **Coordinator also fetches value from other replicas**
 - Checks consistency in the background, initiating a **read repair** if any two values are different
 - This mechanism seeks to eventually bring all replicas up to date
- **At a replica**
 - A row may be split across multiple SSTables => reads need to touch multiple SSTables => reads slower than writes (but still fast)

Now, let us see the read operation, which is supported in the Cassandra, read is similar to the write except, the coordinator can contact x replicas, may be in the same rack. So, the coordinator sends the read to the replicas that have responded quickest, in the past and when X replicas respond the, the coordinator returns the latest timestamp value from among those X replicas. So, we will see, what is the value of X,

how we can, exploit this in the efficiency and the design of different application systems. Now, the coordinator also, fetches the values from, other replicas and it will check, for the consistency in the background, initiating read repair if the two values are different, this mechanism seeks, to eventually, bring up, all the replicas up to date. So, at the replicas, we have to see that a row main, may be split across multiple, SStable that means the, the reads need to touch multiple SStable, sometimes and therefore, the reads become slower, than the writes but, still they are faster, compared to the other traditional RDBMS is. So, therefore that means, multiple SStables, sometimes required, to be, to be accessed, for the, the read operation to be made successful that is why the competition Sometimes, is useful.

Refer Slide Time :(26: 54)

Membership

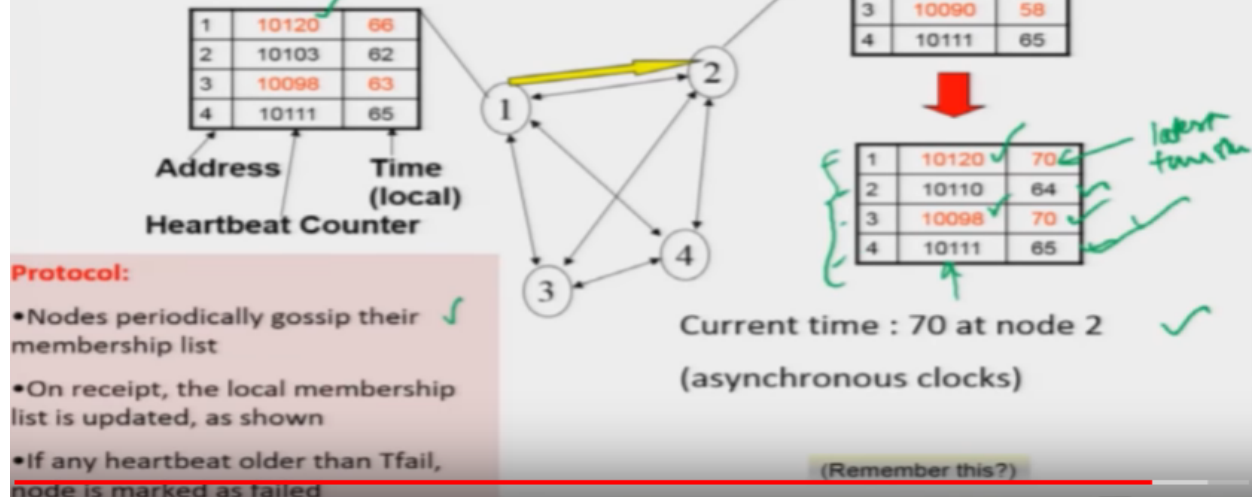
- Any server in cluster could be the coordinator
- So every server needs to maintain a list of all the other servers that are currently in the server
- List needs to be updated automatically as servers join, leave, and fail

Now, we will see the membership, membership in the sense that, the nodes, which are there in the form of a ring, they come and they go down. So, any server in the cluster could be a coordinator, so every, server needs to maintain a list of other servers that are current in the server. And this list need to be updated automatically, as the server join, leaves and fails.

Refer Slide Time :(27: 19)

Cluster Membership – Gossip-Style

Cassandra uses gossip-based cluster membership



So, the membership, cluster membership is maintained in the form of a gossip and this example, shows that, these heartbeat or these details, are exchanged and, and then based on that, they are being updated. So, here we can see that, this particular protocol here the nodes periodically gossip, their membership list and on receive the local membership list is updated, as shown over here. So, if any heartbeat is older than, T fail known, is mark as failed. So, here we can see that, once no 2 receives, this heartbeat, from node1, then it will try to compare, for example, it has, not seen this one zero, to zero. So, it will be now, added with the latest timestamp. Similarly 1, 1, 8 and 1, 1, 0 will be stored and this 1, 0, 1, 1 is there. So, all these are different heartbeat counters, this is heartbeat counter and these are the notes addresses. So, the note address, is 1, 2, 3, 4 they are being updated, with their heartbeat counts, since they have lost being seen over here. So, the node number1 has more recent, heartbeats over here. So, that value, of the heart weight is updated and similarly for the node number 2, the value is not updated. So, because it has, the information about the, the recent information. So, it is not updated, similarly for node number 3, it is, 1, 0, 98 and this information is more recent so, 1, 0, 9 8 is updated, with a, with a recent timestamp and the node number 4, is not going to be changed here in this case. So, this requires the, the current clock, time and the current clock time, will be used to update the, the entries. So, this way, the membership so, the node 2 knows that, in this topology, there are 4 different nodes, which are working and they are exchanging their heartbeats and this is called the, 'Cluster Membership and the Cassandra uses, the gossip based cluster membership. Now, suspicion mechanism, which is there in Cassandra.

Refer Slide Time :(30: 11)

Suspicion Mechanisms in Cassandra

- Suspicion mechanisms to adaptively set the timeout based on underlying network and failure behavior
- **Accrual detector:** Failure Detector outputs a value (PHI) representing suspicion
- Applications set an appropriate threshold
- **PHI calculation for a member**
 - Inter-arrival times for gossip messages
 - $PHI(t) = -\log(CDF \text{ or } Probability(t_{now} - t_{last}))/\log 10$
 - PHI basically determines the detection timeout, but takes into account historical inter-arrival time variations for gossiped heartbeats
- In practice, $PHI = 5 \Rightarrow 10\text{-}15$ sec detection time

So, suspicions, suspicion mechanism to adaptively, set the time out based on underline network and the failure behavior. So, accrual detector is the failure detector which outputs, the value of PHI, which is representing the suspicion and application set and appropriate threshold Phi, will indicate the calculation for a member. So, it will be, internal times for the gossip, messages and in practice the, this threshold, is of PHI that is, it's ten to fifteen seconds off in detection time.

Refer Slide Time :(30: 50)

Cassandra Vs. RDBMS

- MySQL is one of the most popular (and has been for a while)
- On > 50 GB data
- **MySQL**
 - Writes 300 ms avg
 - Reads 350 ms avg
- **Cassandra**
 - Writes 0.12 ms avg
 - Reads 15 ms avg
- Orders of magnitude faster
- What's the catch? What did we lose?

Now, let us compare the Cassandra with RDBMS. So, my sequel is one of the most popular, RDBMS as on date and if, it is more than, 50 GB of data, let us see the, the performance, in my sequel, the writes are taking 300milliseconds on an average heat, is taking 350 milliseconds on an average, if we compare it with, Cassandra where writes is, lightning fast, that is, it will be 0.12 milliseconds on an average, similarly reads are also very, fast that is 15 milliseconds on an average. So, orders of magnitude this Cassandra, provides these operation writes and reads much faster. Now, let us compare, what is the catch? What is the difference? How it is going to achieve, such a fast writes and better reads and what is the catch and where, where we where the Cassandra is losing, compared to the RDBMS that we are going to see using, the Cap Theorem.