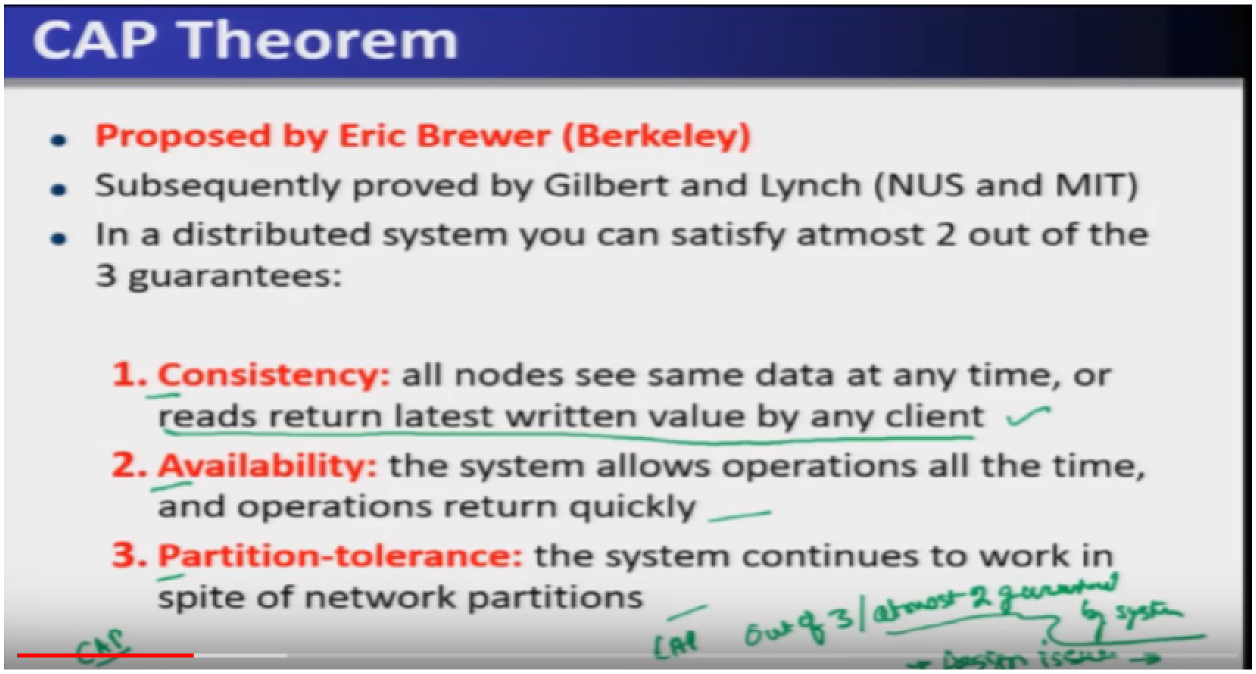
Lecture – 14

Cap Theorem

Cap theorem.

Refer Slide Time :(0: 16)



CAP Theorem

- Proposed by Eric Brewer (Berkeley)
- Subsequently proved by Gilbert and Lynch (NUS and MIT)
- In a distributed system you can satisfy atmost 2 out of the 3 guarantees:

1. Consistency: all nodes see same data at any time, or reads return latest written value by any client ✓
2. Availability: the system allows operations all the time, and operations return quickly
3. Partition-tolerance: the system continues to work in spite of network partitions

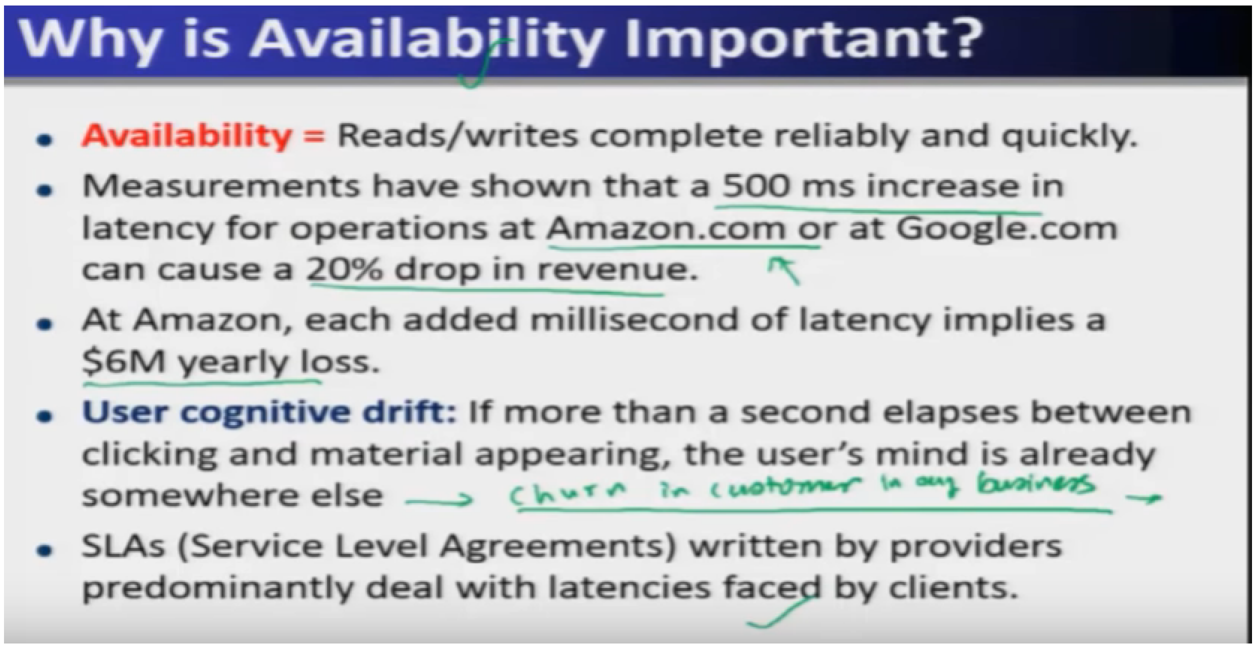
CAP

CAP Out of 3 | atmost 2 guaranteed by system

→ Design issue →

Cap theorem was proposed by Eric Brewer, in Berkeley and which was subsequently proved by Gilbert and Lynch. In a distributed systems you can specify, at most two out of three different guarantees; which is now, specified as, which is known as, the cap theorem. Let us see, what these three different guarantees? Three different things are. So, the first one is called, ‘Consistency’, the second one is called, ‘Availability’, third one is called, ‘Partition tolerance’. Cap theorem says that, out of these three at most two can be the guaranteed, by any systems. So, let us see the, consistency that means all the nodes see the same data, at any time or the read returns the latest return value, by any client. So, that is called, ‘Consistency’, that means at all points of time, the node will, get the most recent data, at any point of time whenever it is being, referred or it is being accessed, by the read operation. So, whenever the read operation is performed, it will, read it will return the latest write, by the client, if that is, guaranteed at all points of time, then it is called, ‘Consistency’. Availability says that the system allows operation, all the time and operations returned quickly. Meaning to say that, the operations, the system always is operating and whenever is being accessed, it will perform, it will return, it will return the operations, very quickly. So, this is called, ‘Availability’. Third criteria, is called, ‘Partition Tolerance’ that is the system continues to work in spite, of network partitions. So, the cap theorem says that, out of three, out of these three. So, cap, c a p that means consistency availability and partition tolerant, in short, it is called, ‘Cap Theorem’. Cap theorem says that, out of three, out of these three different parameters, at most, two can be guaranteed, at any time, by the system. So, this is, the design issue that we are going to see, how what is the implication of this cap theorem or different no sequel systems which are being available, at this point of time.

Refer Slide Time :(3: 26)

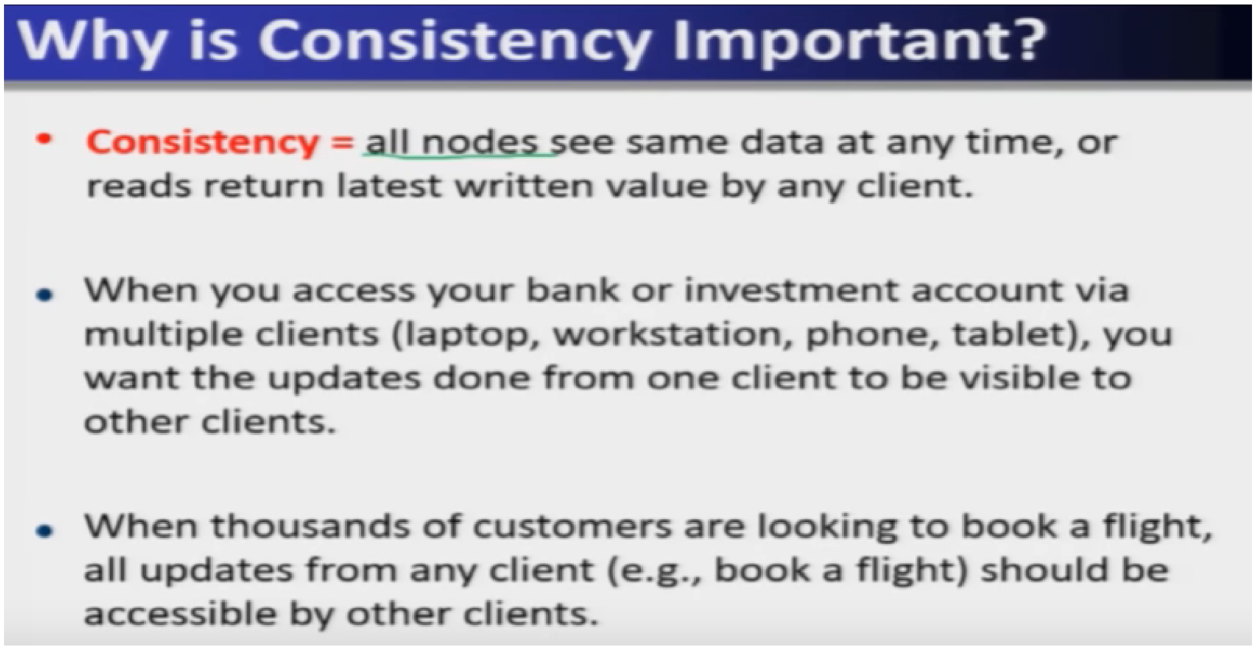


Why is Availability Important?

- **Availability =** Reads/writes complete reliably and quickly.
- Measurements have shown that a 500 ms increase in latency for operations at Amazon.com or at Google.com can cause a 20% drop in revenue.
- At Amazon, each added millisecond of latency implies a $6M yearly loss.
- **User cognitive drift:** If more than a second elapses between clicking and material appearing, the user's mind is already somewhere else → Churn in customer in any business →
- SLAs (Service Level Agreements) written by providers predominantly deal with latencies faced by clients.

Now, let us see, all these three different things, which is specified in the cap theorem that is the first one islet us, understand about the availability and what is the importance of availability? Availability says that read and write will complete reliably and quickly, at all point of time. So, in if we measure, these are read and write operation times and let us see that, this particular measurement will have an increase, of 500 millisecond, latency then let us see, what is the implication of most of the operations in the company. So, this latency, of read and write of 500milliseconds, it is shown that, for the companies like Amazon or a Google, highest cost, drop of 20% in the revenue model, meaning to say that, so Amazon.com, is dealing with the sales of these items. So, if there is latency, of 500milliseconds, in these performance, then obviously customers will churn and therefore they will drop in the revenues. So, at imagine each added millisecond, of latency implies, implies a six million yearly loss. So, a user cognitive drip; is there and if more than a second he lapses, between the clicking and the material appearing, the users mind is already somewhere else and this will lead to a churn, in the customers, in any business. So, this is a, most important parameter in some of the cases. So, the companies, like online which are dealing with the e-commerce online sales, that is imagine company or the Google, which taps the advertisements and other such product which are and services online, has to ensure, high availability and not only reliability but, it has to be very, quickly that is the latency also has to be very minimal. So, that the users can get the services. So, therefore a service level agreements, written by the providers, predominantly deal with the latencies, which are faced by the clients and therefore the availability is also, one of the most important parameter or criteria, in the cap theorem.

Refer Slide Time :(6: 11)

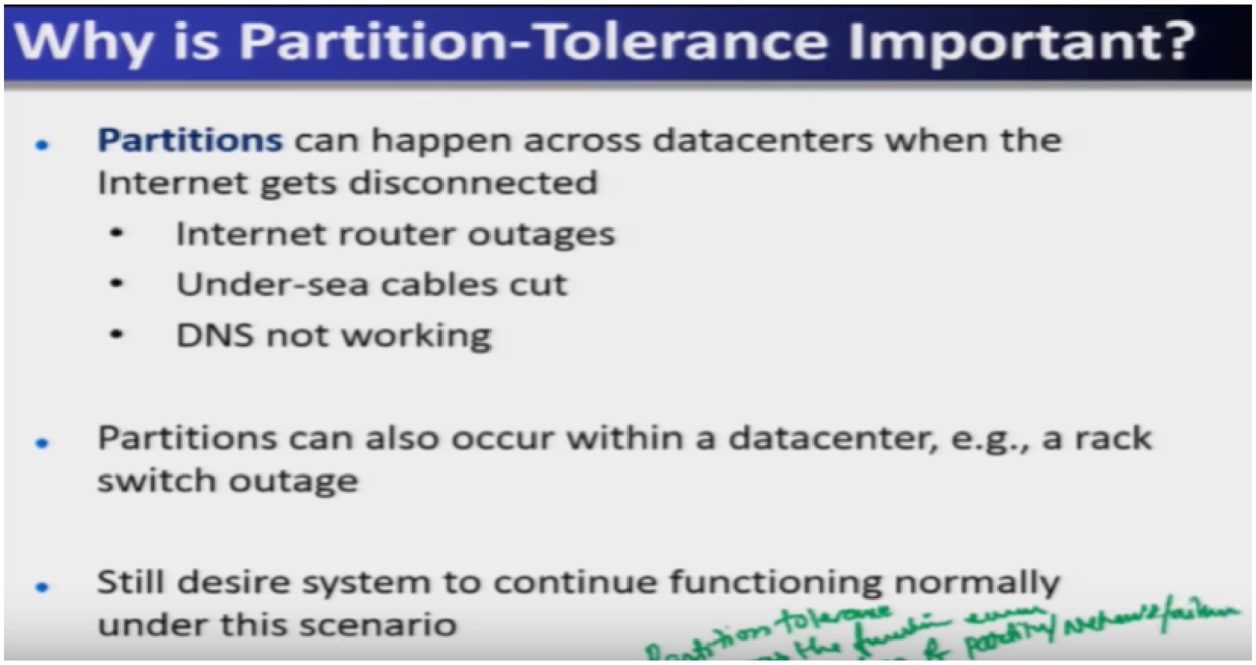


Why is Consistency Important?

- Consistency = all nodes see same data at any time, or reads return latest written value by any client.
- When you access your bank or investment account via multiple clients (laptop, workstation, phone, tablet), you want the updates done from one client to be visible to other clients.
- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients.

Now, second criteria, which is called a, 'Consistency', of a cap theorem C of that is called, 'Consistency'. Consistency says that, all the nodes, see the same data, at any time are or reads returns, the latest return values by the client. Meaning to say that, the nodes whatever updates are happening, on the nodes by different write operations. So, they are updated, they are up-to-date. So, whenever read is requested, it will always return, the latest write operations which are done by the any client. So, reads are very latest, returns read returns the very latest, information which are written by the client, if that is maintained at all points of time that is called, 'Consistency'. Now, when you access your bank account or investment account, by multiple clients wire, laptop, work station, phone, tablets, excess price you want these updates to be done from, one client to be visible to the other clients. It's not that, if you have done through the mobile phone and your mobile phone, updates are and you are, you are now, accessing or referring reading wire, laptop, they may, not get that a decent update, if that is the case, then it is not a consistency. So, similarly when a thousands of customers are looking to book a flight ticket, all the updates from a client, should be accessible by the other client, in the reservations, airline reservation, to booking a tickets. So, consistency also, is going to be, an important parameter.

Refer Slide Time :(7: 49)



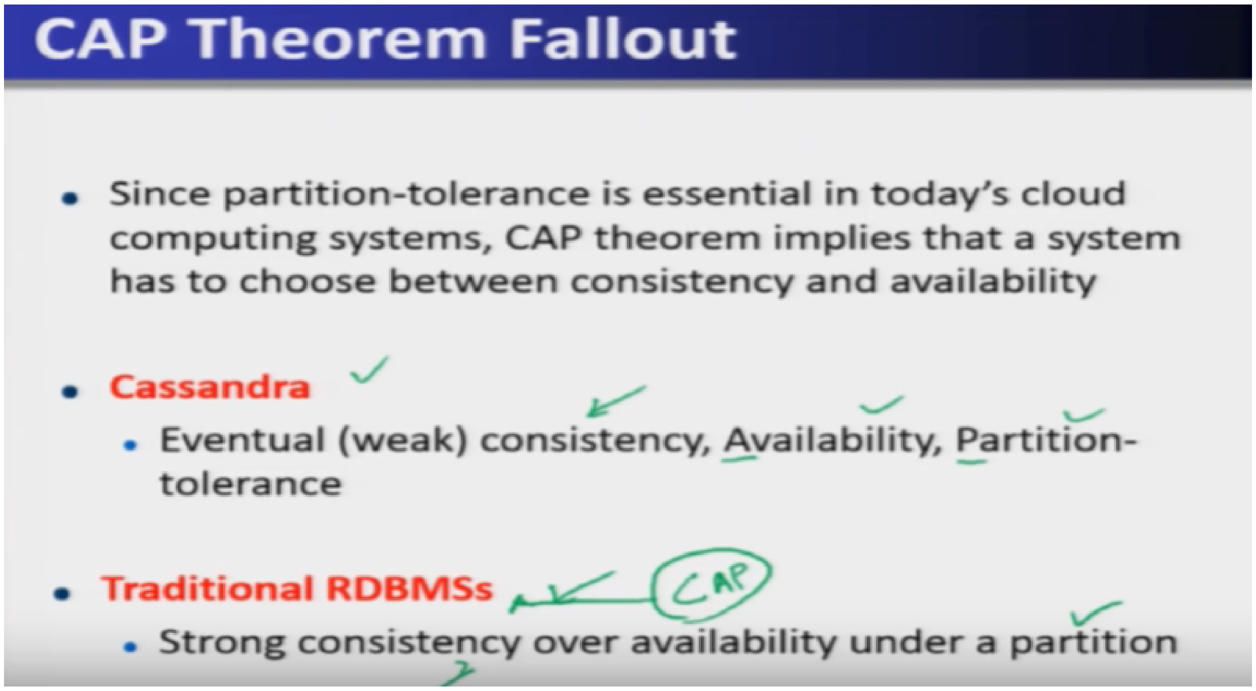
Why is Partition-Tolerance Important?

- **Partitions** can happen across datacenters when the Internet gets disconnected
  - Internet router outages
  - Under-sea cables cut
  - DNS not working
- Partitions can also occur within a datacenter, e.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

Partition tolerance
[illegible] the functioning even
[illegible] & partitions/ Network failures

Now, another parameter, which is there in cap, c a p, P, P, P stands for the partition. So, partitions can happen across datacenters, when the internet gets disconnected. So, this can happen by way of, Internet routages, internet router outages, are undersea cable cuts or DNS not working, in all these scenarios, you will see that, the network gets partitioned and the entire data center, some of the data, centers may be it is connected and partition. So, partition tolerance, partitions can occur, within the data center, within the rack or switch outages and so on. So, rack switches also, can basically ensure or induce these partitions, in tune to the network. So, still the, the desire system, the desire is that system to continue functioning normally under, the partition that is why? It is called, 'Partition Tolerance'. Partition tolerance, ensures the functioning, even in the face, of partitions or it is also called as, 'Network Failures'.

Refer Slide Time :(9: 24)

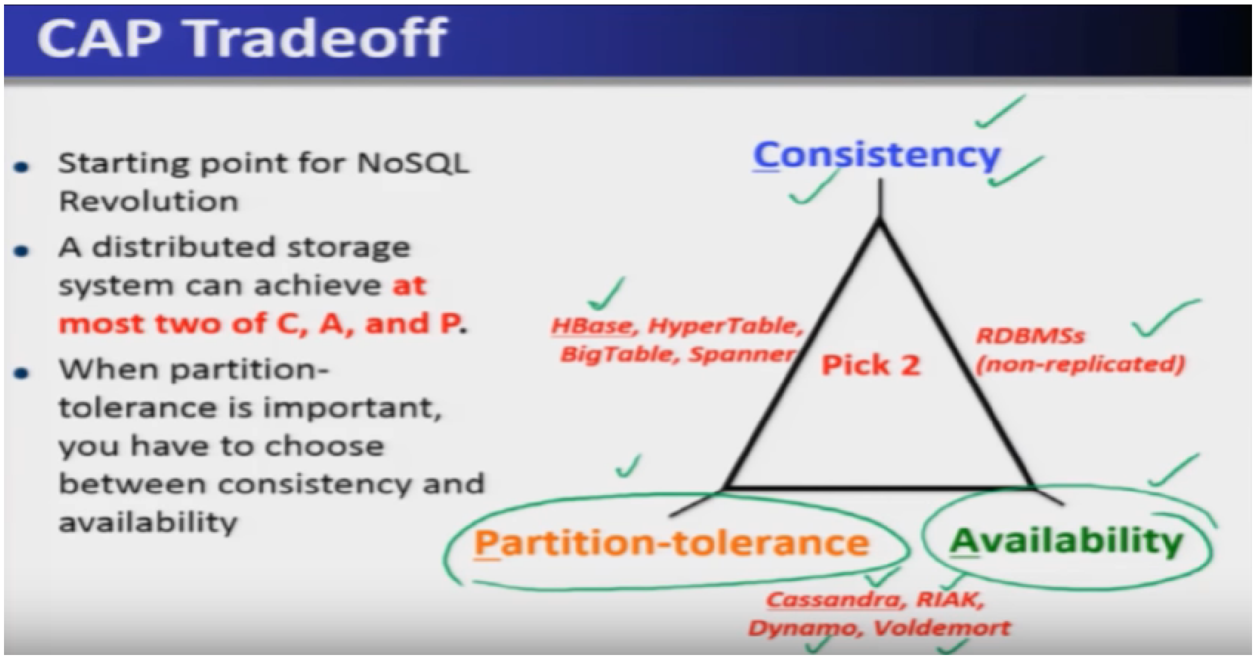


CAP Theorem Fallout

- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability
- Cassandra
  - Eventual (weak) consistency, Availability, Partition-tolerance
- Traditional RDBMSs
  - Strong consistency over availability under a partition

Now, the cap theorem fallout, since the partition tolerant is essential in today's cloud computing systems, the cap theorem implies that a system has to choose between the consistency and the availability. Now, we will see the Cassandra. So, Cassandra uses, eventual that is called a, 'Eventual Consistency' which is also, a weak form of consistency and then it ensures, it also uses the availability and partition tolerance. So, it Cassandra uses, a and P, of a cap, where C is being compromised or that is called, 'Eventual Consistency' or a 'Weak Form of Consistency'. Similarly the traditional databases management system, we see that, provides the strong consistency that, is it provides C and or availability, under the partition since, the since the database, is on the same system hence, the partition is already, partition will not happen hence the cap is, always satisfied or guaranteed, in the traditional RDBMS systems.

Refer Slide Time :(10: 40)

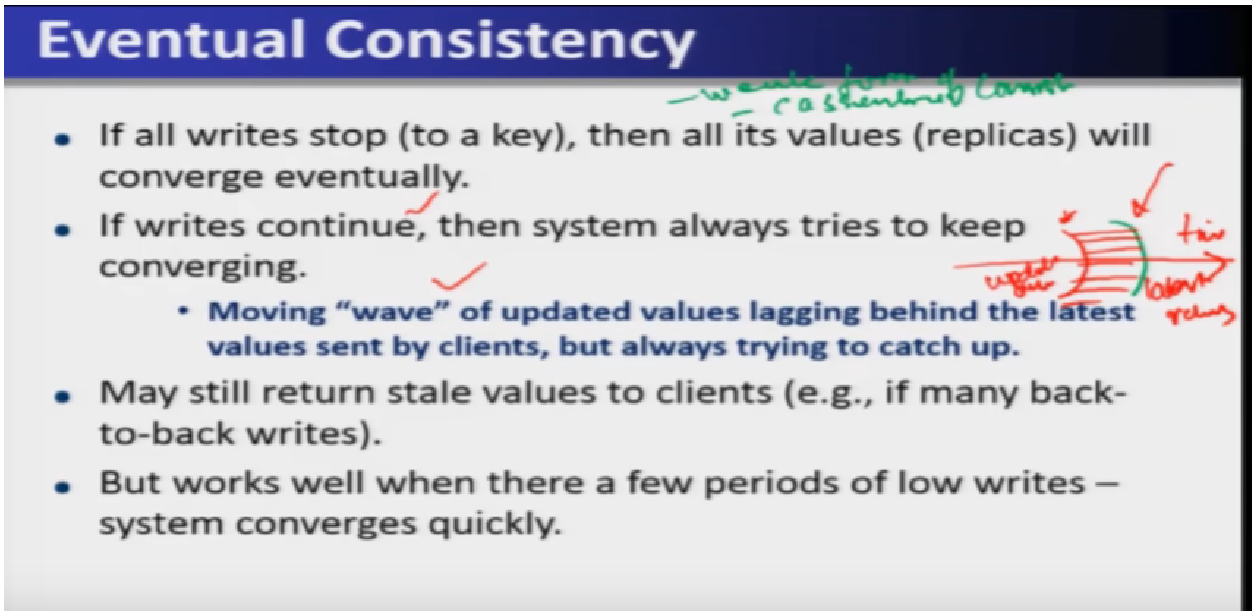


CAP Tradeoff

- Starting point for NoSQL Revolution
- A distributed storage system can achieve **at most two of C, A, and P.**
- When partition-tolerance is important, you have to choose between consistency and availability

So, cap trade-off let us see that, starting point for this no sequel system, the cap trade-off has to be, there so, in a distributed storage system can achieve, at most two out of three, at most two out of this consistency availability and partitioning. So, when the partition tolerance is important, then you have to choose between consistency and availability. So. Same thing is happening in the Cassandra system. So, we have to choose, between consistency and availability, so Cassandra chooses availability and a weaker form of consistency, in Cassandra. Similarly in react and dynamo and Waddell Mott, also uses the partition tolerance and availability with a, with a weaker form of consistency. Now, as far as R DBMS is concerned. So R DBMS, which is not replicated, which ensures the consistency and availability and there is no partition hence, consistency and availability are ensured in R DBMS is now, H Base, prefer reds partition tolerance and consistency or availability. So, we will see that, H Base and hyper table, Big Table and span and uses the, the consistency and partition tolerance or availability.

Refer Slide Time :(12:04)

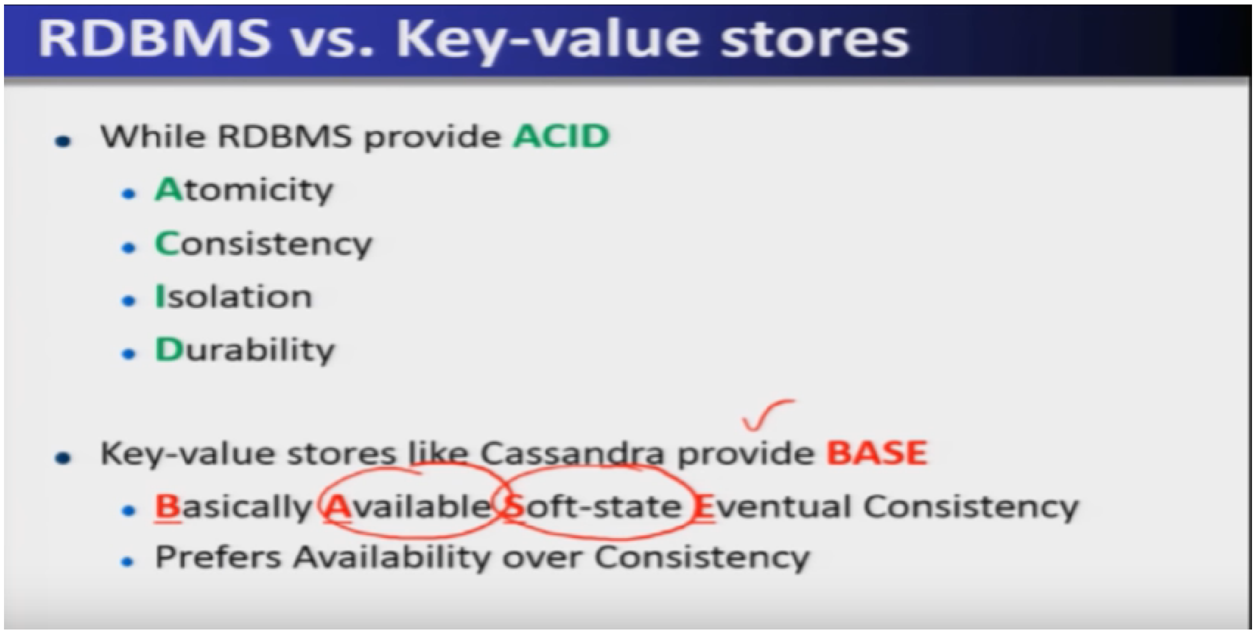


Eventual Consistency

- If all writes stop (to a key), then all its values (replicas) will converge eventually.
- If writes continue, then system always tries to keep converging.
    - Moving "wave" of updated values lagging behind the latest values sent by clients, but always trying to catch up.
- May still return stale values to clients (e.g., if many back-to-back writes).
- But works well when there a few periods of low writes – system converges quickly.

Let us see the, eventual consistency, which is a weak form of consistency. Which is supported in the Cassandra? Let us go and detail see, how the Cassandra is trading off with the consistency, to ensure, the partition, tolerance and the availability. Now, if all the writes stopped, to a particular key, then all its values in the replicas will eventually converge. meaning to say that, if Some of the rights, do not takes place or do not get updated at Some of its replicas, then those replicas, will be updated or will become consistent after Some point of time. Hence, they will be eventually converge. So, if the write continues then system always tries to keep converging. So this way, the moving wave, of the updated values, moving wave of updated values, lagging behind the latest values, sent by the, by the client. So, these waves will move over the time. So, that says that, moving wave, of an updated values, will be lagging behind the latest values sent by the client, but always tries to catch up. So this difference will always be there. Now if, the number of new values are not coming, then obviously, eventually it will become the latest one, after eventually it will, be converging it. So if, the right continues then, the system tries to, tries to keep converging and this may still, return some of the stale values to the client, if many back to back right operations are there, but it will work fine when, there are a few, periods of low right's and So, the system can converge quickly.

Refer Slide Time :(14:34)

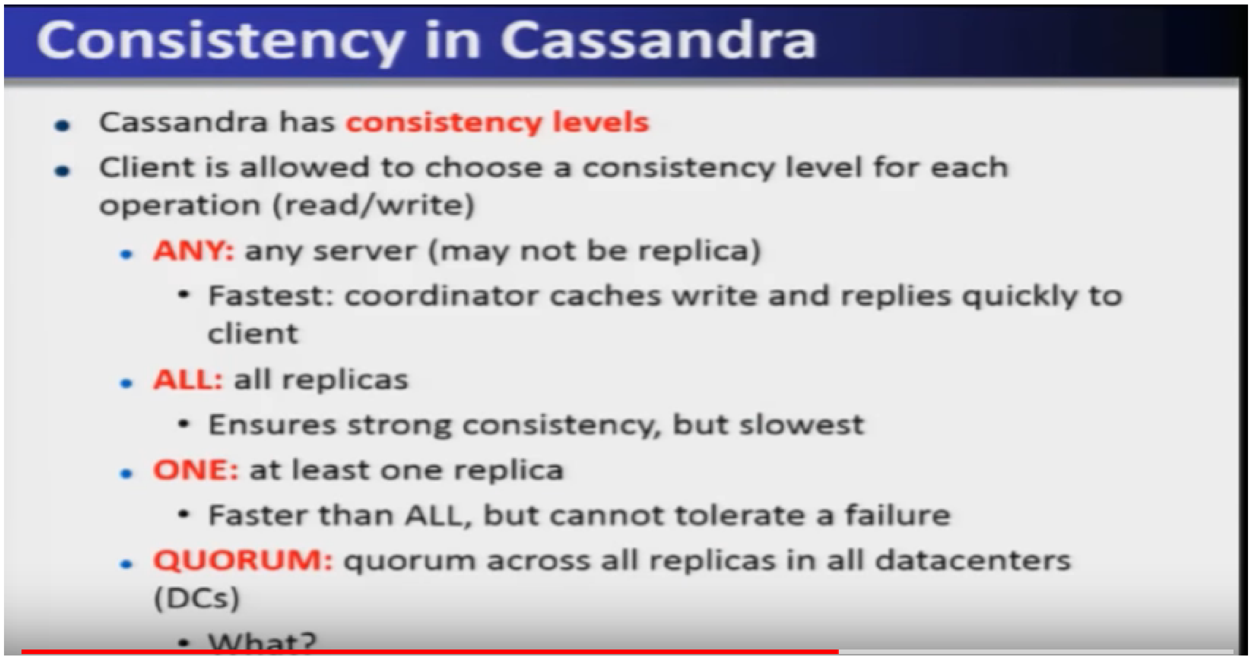


RDBMS vs. Key-value stores

- While RDBMS provide ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Key-value stores like Cassandra provide BASE
  - Basically Available Soft-state Eventual Consistency
  - Prefers Availability over Consistency

Now, let us, compare the RDBMS with the key value stores. So, So RDBMS provides the acid properties, atomicity, consistency, isolation and durability, whereas key value store like Cassandra provides, the base property. Base is basically available, soft state, eventual consistency. So basically available means, there, they are it provides the availability insurance and Soft state eventual consistency, Soft state that we have seen, in the form of cash cashing, in in-memory operations, most of that the table properties are stored in cash and that is in the form of Memtable and eventual consistency means, finally everything, will be updated, that is called, 'Eventual Consistency', not immediately but, eventually it will be updated.

Refer Slide Time :(15:37)

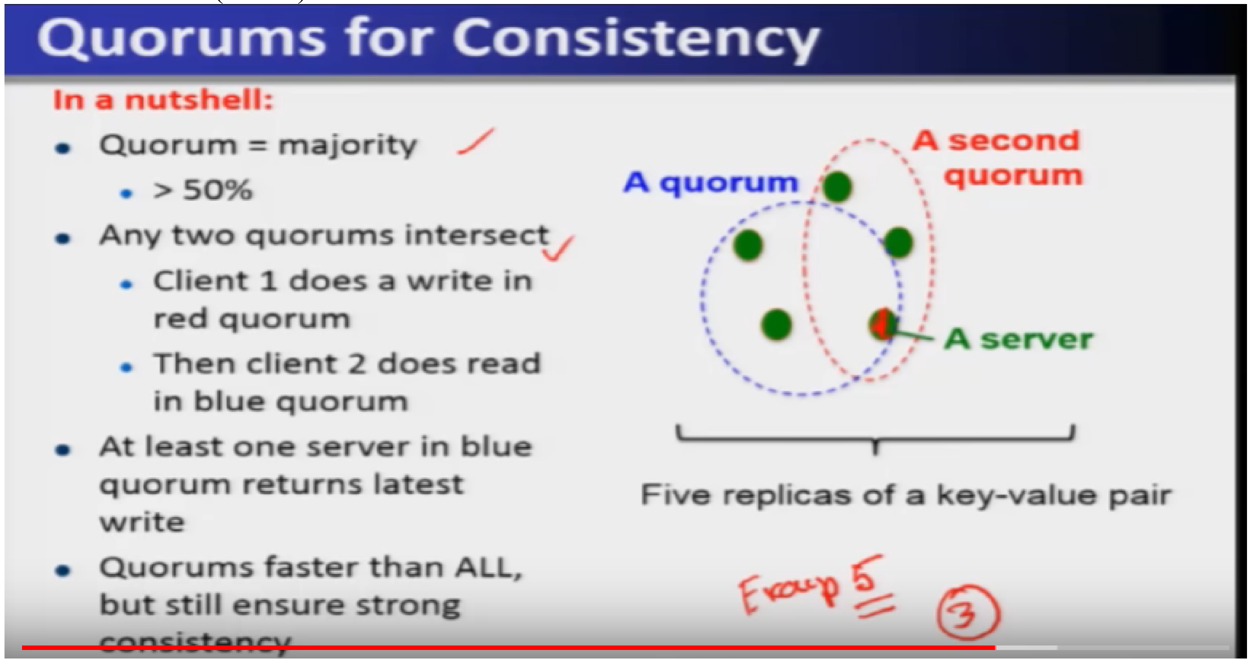


Consistency in Cassandra

- Cassandra has consistency levels
- Client is allowed to choose a consistency level for each operation (read/write)
  - ANY: any server (may not be replica)
    - Fastest: coordinator caches write and replies quickly to client
  - ALL: all replicas
    - Ensures strong consistency, but slowest
  - ONE: at least one replica
    - Faster than ALL, but cannot tolerate a failure
  - QUORUM: quorum across all replicas in all datacenters (DCs)
    - What?

So, it prefers the availability or the consistency in the base model. Now, let us see the consistency, levels which are supported in the Cassandra, Cassandra has different consistency levels. So, the client is allowed to choose the consistency level, for each operation, that read and write, among these different consistency levels. So they are, one that is any, second type of consistency level is all, third is one and fourth is quorum. Let us, understand one by one all these consistency level. So any, consistency level, by mean, any consistency level, that means, any server may not be the replica is can, can allow, this operation to be complete. So this becomes a fastest, because the coordinator caches, the right operation and returns quickly to the, to the client, to perform this read and write operations hence, this is the fastest one. The second one is called, 'All Replicas'. That means, it ensures that the read and write operations, write operations requires, to ensure that, it has to be updated at all the replicas. So this is a kind of strong consistency, so it will provide, if the consistency level is all then it will be as the slowest one. Third one is called' 'One', that is at least one of these replicas gets updated, so it is faster than all, but it is slower there any, why because it cannot tolerate the failure of all, of the replicas. And fourth one is called, 'Quorum', quorum says that, quorum that means, a quorum across all the replicas in the data center is to be updated, what do you mean by the quorum? That we are going to see? So quorum is between the all and one, so quorum is some number K. So, we are going to see how many replicas are required to be updated, under the quorum system.

Refer Slide Time :(17:41)



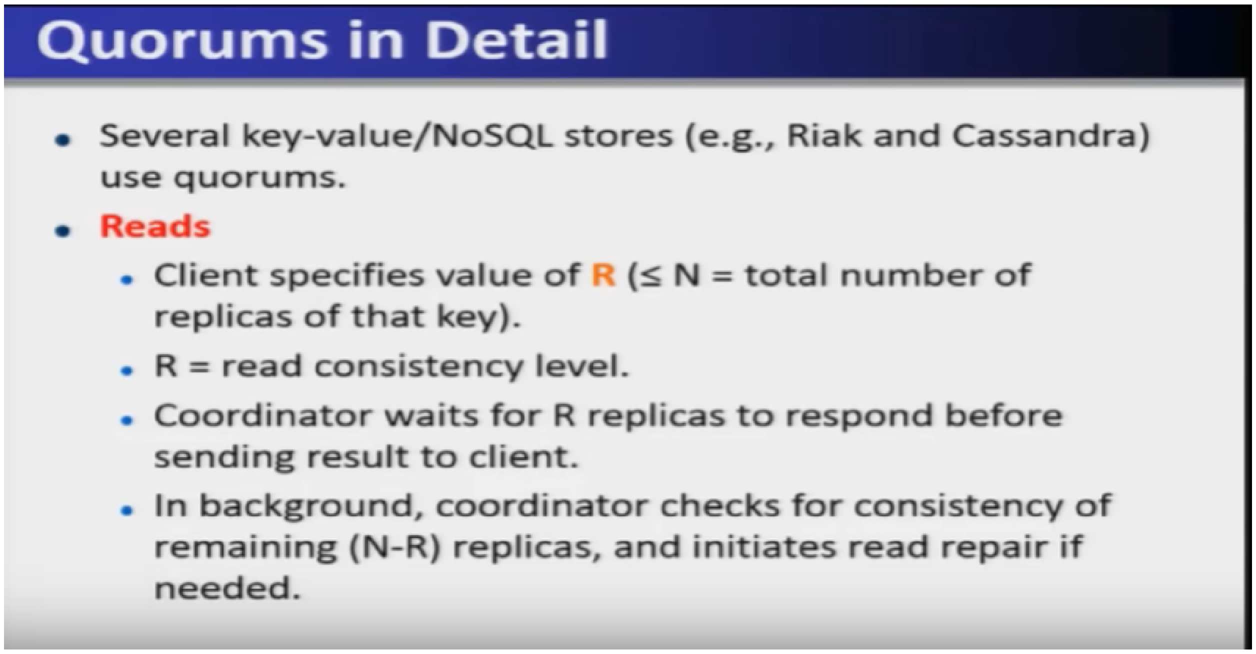
Quorums for Consistency

In a nutshell:

- Quorum = majority
  - > 50%
- Any two quorums intersect
  - Client 1 does a write in red quorum
  - Then client 2 does read in blue quorum
- At least one server in blue quorum returns latest write
- Quorums faster than ALL, but still ensure strong consistency

So quorum's for consistency, quorum says that majority, so if there are in this example, there are five different replicas, so the quorum says, the majority means, more than fifty percent, so that, becomes the three. So minimum at least three different quorums, different replicas are to be updated. So, for any to the proper these upper quorum, which has to be satisfied is that, if any to quorum's, if we take the intersection then there must be a common, replica common servers between any two system. So, any two quorum vary intersect, client one does the right operation, in the right quorum and the Cline two, reads from the blue quorum, then it will get the update from, because there is a common server in both the quorum's, So at least one server in the blue quorum, gets the latest Right? So quorums are faster than all, but ensure the strong consistency.

Refer Slide Time :(18:46)

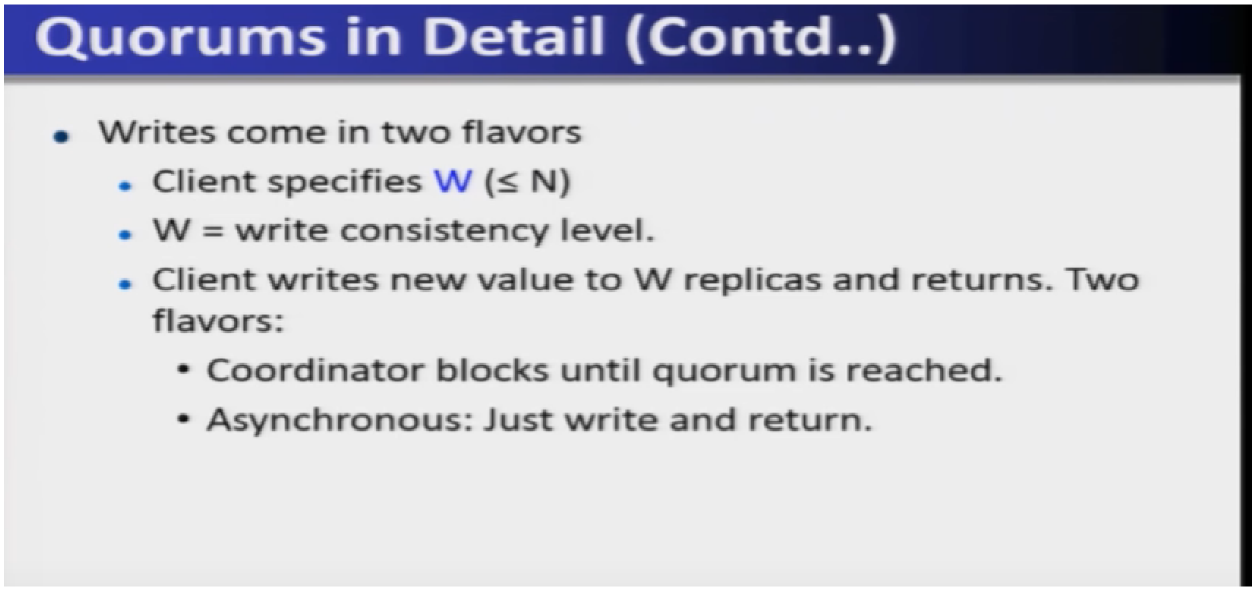


Quorums in Detail

- Several key-value/NoSQL stores (e.g., Riak and Cassandra) use quorums.
- Reads
  - Client specifies value of R (≤ N = total number of replicas of that key).
  - R = read consistency level.
  - Coordinator waits for R replicas to respond before sending result to client.
  - In background, coordinator checks for consistency of remaining (N-R) replicas, and initiates read repair if needed.

So quorum’s, let us see in more detail, So several key value pairs, that is no sequence to react and Cassandra uses, the quorum. So reads; that is the client specifies the value of R, which is the value of the quorum, which is less than the number of replicas, So R is the read consistency level, So coordinator waits for R replicas to respond before sending, the result to the client. In the background, the coordinator, checks for the consistency of remaining and - R replicas and initiate, the read repair if any. So meaning to say that, not only it the, the read is being satisfied by, specifying the R number of replicas, but what about the, the remaining replicas, that is n minus R, whether are they consistent or not that also required to be checked, for the consistency, of the remaining replicas. So that has to be done in the background and if they are inconsistent then, the read repair is to be initiated, So that eventually they may be consistent at all the levels.

Refer Slide Time :(19:55)

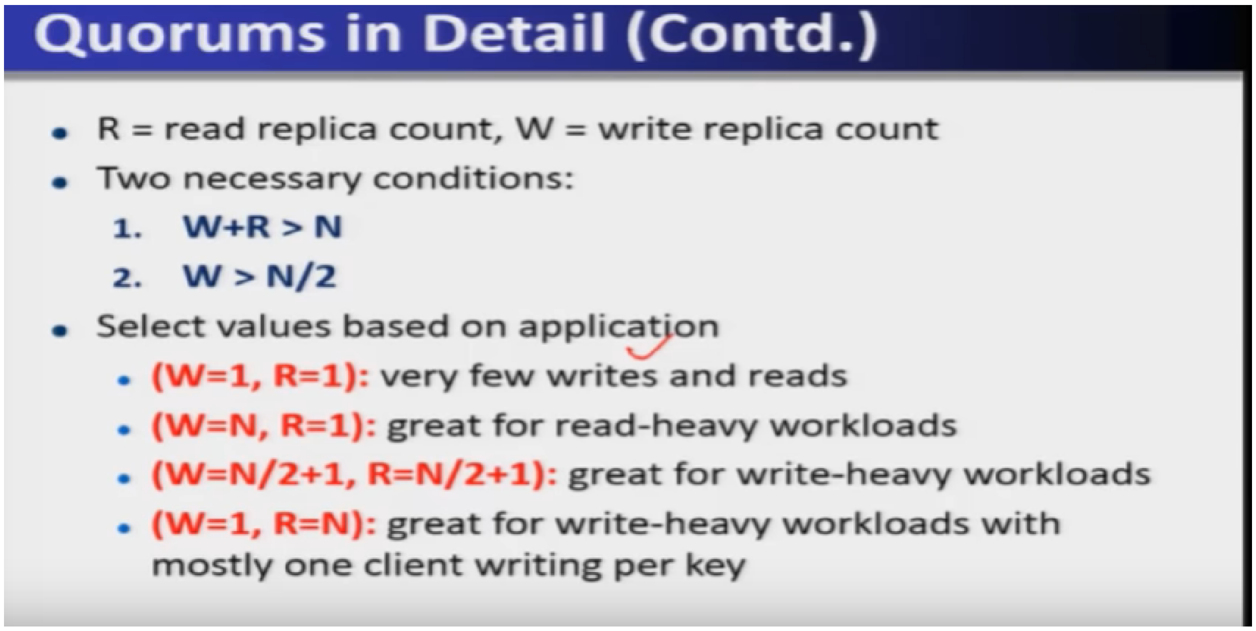


Quorums in Detail (Contd..)

- Writes come in two flavors
  - Client specifies W (≤ N)
  - W = write consistency level.
  - Client writes new value to W replicas and returns. Two flavors:
    - Coordinator blocks until quorum is reached.
    - Asynchronous: Just write and return.

So the reads sometimes, does the read repair, to ensure the eventual consistency. now, quorum's the write come in to flavor, when a client writes w, into the number of replicas, less than total number n, then write replication, then write consistency level, we have to specify. So the client writes, a new value to W replicas turn, there are two flavors. So the, coordinator blocks, until the quorum is reached or the it will be in the I synchronous, that means, it will just write and return, returns back.

Refer Slide Time :(20:34)

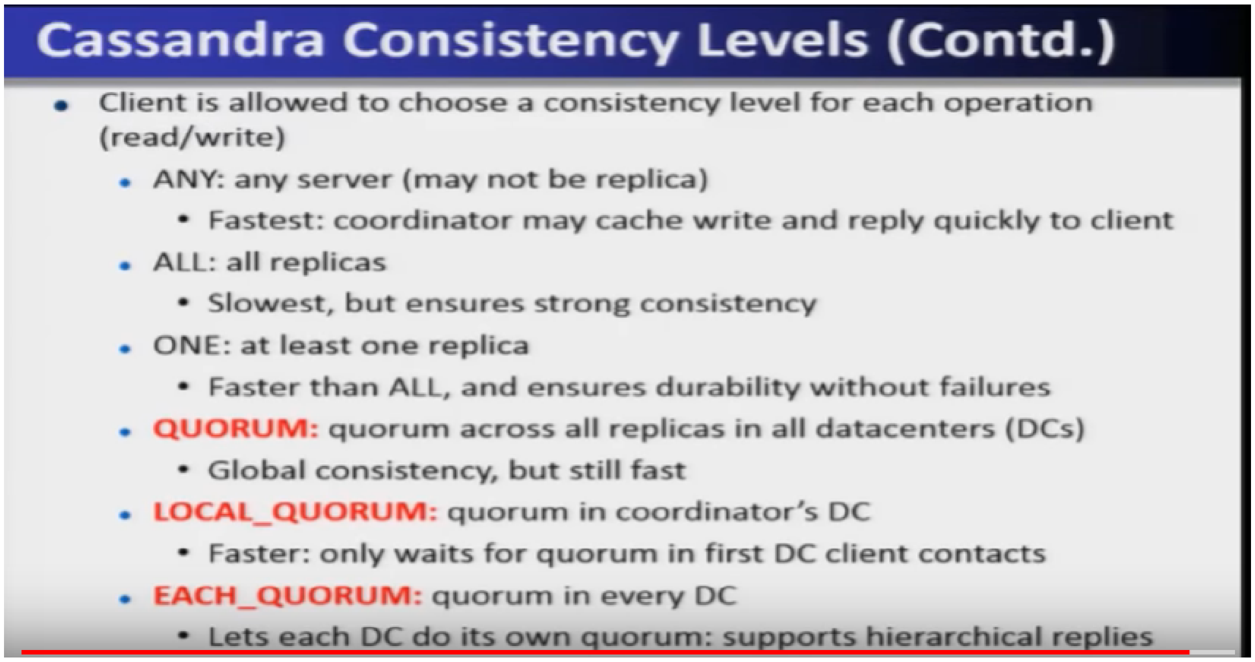


Quorums in Detail (Contd.)

- R = read replica count, W = write replica count
- Two necessary conditions:
  1. W+R > N
  2. W > N/2
- Select values based on application
  - **(W=1, R=1):** very few writes and reads
  - **(W=N, R=1):** great for read-heavy workloads
  - **(W=N/2+1, R=N/2+1):** great for write-heavy workloads
  - **(W=1, R=N):** great for write-heavy workloads with mostly one client writing per key

Now, if let us say, R is the replica, read replica count and W is the Right replica count, then there are two necessary conditions in the quorum, to be satisfied. one is the Right replica count plus, read replica count, must be greater than n and W, plus W is equal to are also greater than n. So that becomes, W is greater than n by two. now, we have to select these values, based on different applications, when there are very few right's and, and very few right's and reads then W is equal to one, when there is a great, for read heavy workloads, then write is n and, and read is one. and when there is a great for the right heavy workloads, then W is N divided by two plus one, varies and R is equal to n divided by two plus one. and it is great for, the right heavy workload, with more, with mostly one client writing perky, then W is equal to 1 and R is equal to n. So these are different replicas count, for read and write can beset, according to the applications, which are there.

Refer Slide Time :(21:54)

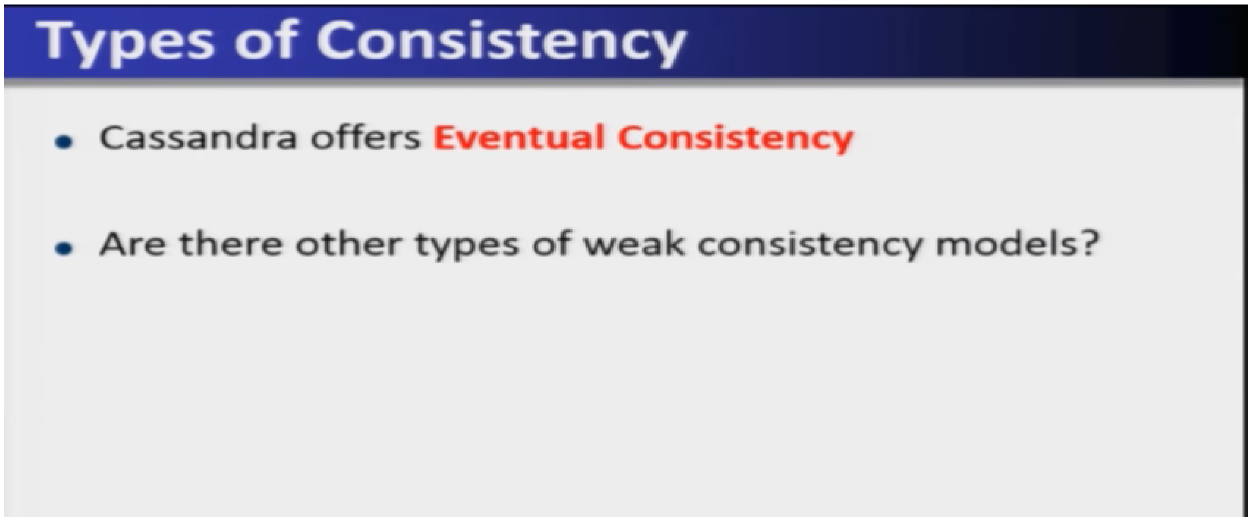


Cassandra Consistency Levels (Contd.)

- Client is allowed to choose a consistency level for each operation (read/write)
  - ANY: any server (may not be replica)
    - Fastest: coordinator may cache write and reply quickly to client
  - ALL: all replicas
    - Slowest, but ensures strong consistency
  - ONE: at least one replica
    - Faster than ALL, and ensures durability without failures
  - **QUORUM:** quorum across all replicas in all datacenters (DCs)
    - Global consistency, but still fast
  - **LOCAL_QUORUM:** quorum in coordinator's DC
    - Faster: only waits for quorum in first DC client contacts
  - **EACH_QUORUM:** quorum in every DC
    - Lets each DC do its own quorum: supports hierarchical replies

So, we have seen the quorum's, across all the replicas, in all the datacenters and it ensures the global consistency, which is still a fast one. So local quorum, So Coulomb's in a in, in the coordinated data center are faster, only waits for the quorum in the first datacenter client contacts, each quorum's, the quorum in every data center, let each data center do its own quorum and support the heretical replies.

Refer Slide Time :(22:23)



Types of Consistency

- Cassandra offers **Eventual Consistency**
- Are there other types of weak consistency models?

So the type of consistency, which Cassandra provides is called, 'Eventual Consistency'. What are the other weak, forms of consistency models, which are available in practice? That we will see, in the, in the next slide.