

Lecture 16

Design of Zookeeper

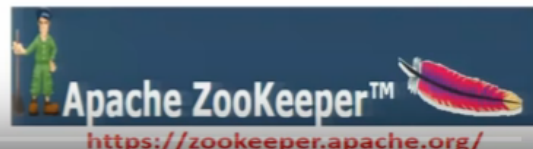
Design of Zookeeper.

Refer slide time: (0:18)

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**design of ZooKeeper**', which is a service for coordinating processes of distributed applications.
- We will discuss its basic fundamentals, design goals, architecture and applications.



Preface; Content of this lecture; in this lecture we will discuss the 'Design of Zookeeper', which is a service, for coordinating processes, in a distributed application. We will discuss; the fundamentals, the design goals, architecture and the applications of Zookeeper. Zookeeper is also known as, 'Apache Zookeeper', which is an open-source foundation. So it is available, a free to be used, for the open communities.

Refer slide time: (1:00)

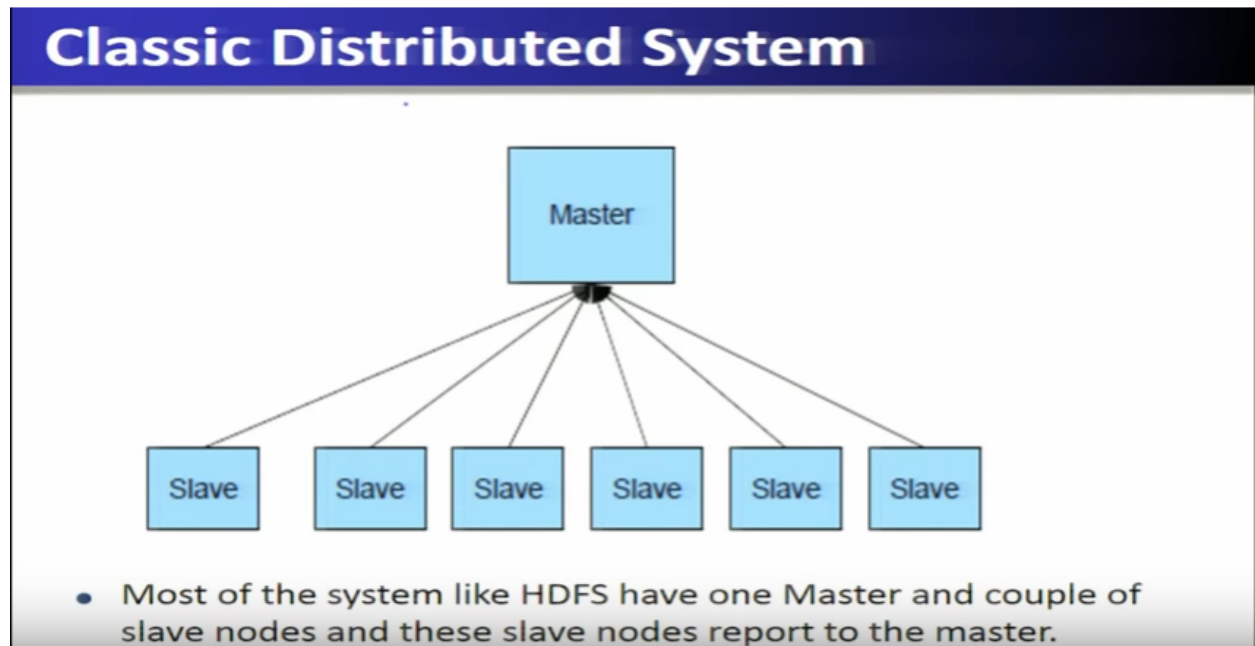
ZooKeeper, why do we need it?

- **Coordination is important**



Zookeeper, why do we need it? So Zookeeper provides the coordination service. What is the coordination? We can understand by looking up, this particular picture, there will be traffic condition, at the intersection, so if there is a coordination then, there will not be any such congestion and there will be a smooth flow of the traffic, similarly here, in our scenario, of the big data computing, we have the hundreds and thousands of cluster nodes, wherein the processes are executing, these cluster nodes are coming up and going down, that means, they are prone to the failure and different applications require to be run-on, these nodes. Therefore, for the smooth way of resource allocation, in a fault alternate or a reliable manner, there is a requirement of coordination service and Zookeeper is, used for that purpose or designing of different, big data applications. And the libraries, we will see in this part of the discussion, having understood what coordination is? And how important the coordination is?

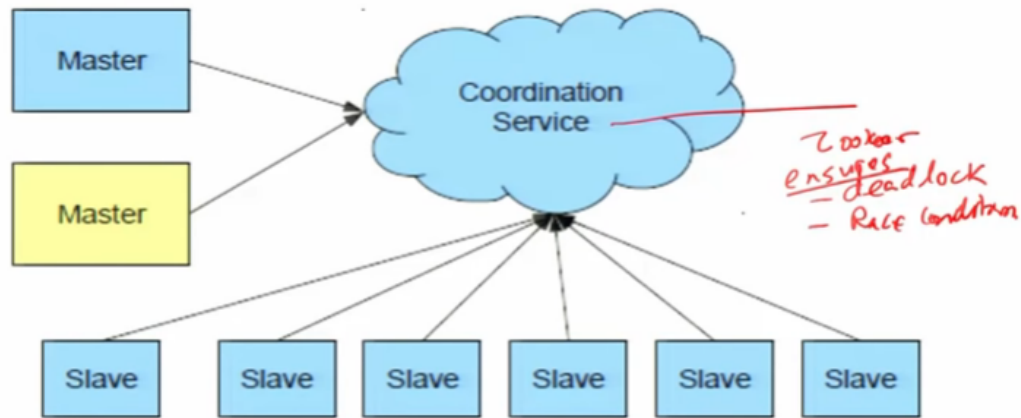
Refer slide time: (2:24)



Now we have to understand, in another scenario, which is of the master of slave relation, of a distributed system, here most of the systems, like HDFS has one master and a couple of slave nodes and these slave nodes, will report to them, to the paid, to the master. What happens if the master fails? The entire system will collapse, to work.

Refer slide time: (2:51)

Fault Tolerant Distributed System



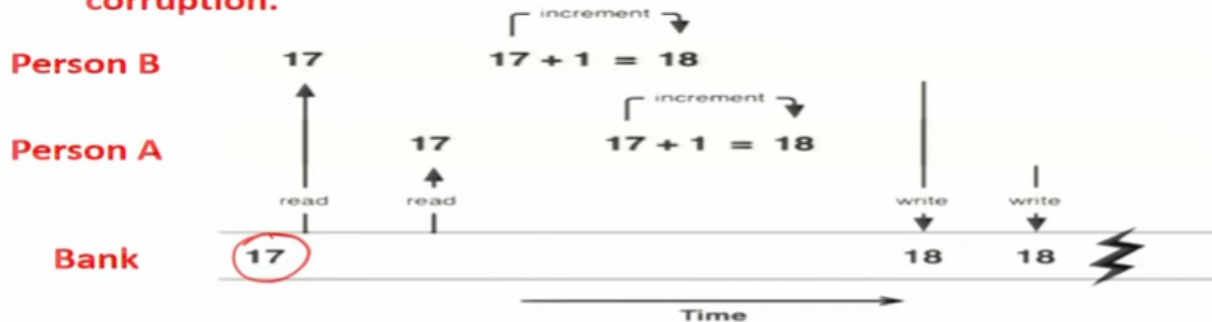
- Real distributed fault tolerant system have Coordination service, Master and backup master.
- If primary failed then backup works for it.

The fault tolerant distributed systems, where in, between master and slave, there is a coordination service, which coordinates between the master and slave processes, running on different nodes. So the irrespective of the node failures, the coordination service will, assign the nodes to these master and therefore, a fault tolerant distributed system can be envisaged, using such a coordination service. So, this coordination service is very much required, to achieve this fault tolerant, coordinated view of any application in a distributed environment, so real distributed fault-tolerant system how, a coordination service, that is the master and a backup master, so if a master fails, then the backup master takes over. and this particular taking over of, backup master, as the master is to be monitored and decided by the coordination service, so therefore, the master and the slave, can be used in an application, irrespective of the failure. So, if the primary, that is the master primary fails, then the backup can be taken up, to ensure the fault tolerant distributed system. Here in this example, the coordination service is the Zookeeper, which provides the coordination services, to achieve the fault tolerant distributed system. This particular coordination service, has we ensure that, it should not lead to a deadlock and also not suffer from, the race conditions. so and, it ensures that deadlock, can be prevented deadlock should not happen, in any of the case and also, it should not face the risk conditions.

Refer slide time: (2:51)

What is a Race Condition?

- When two processes are competing with each other causing **data corruption**.



As shown in the diagram, two persons are trying to deposit 1 rs. online into the same bank account. The initial amount is 17 rs. Due to race conditions, the final amount in the bank is 18 rs. instead of 19.

To understand these two terms, what do you mean by the race condition? we can see through this particular example, so when two processes are competing with each other, causing data corruption. for example, person A and person B together, they want to update, the same variable that is the bank account, which is having the current value 17, at the same point of time, with the value 1, then what will happen is, they may read the variable 17 and update at their end and finally write back, the values. So instead of, two increments, only one increment is now, taken into the effect. Therefore, together the two processes which are competing may cause, the data corruption here in this example, which is shown, this condition is called a, 'Race Condition'. so as shown in the diagram, two persons are trying to deposit 1 rupees online, on to the same bank account, the initial amount is 70 rupees, due to the race condition, the final amount is written as 18 rupees, instead of 19. so therefore, this condition is called a, 'Race Condition', where competing processes, which are writing the or which are accessing the variable may cause, the race conditions and therefore, may corrupt the data. This has to be avoided in the coordination service, why because multiple, multiple servers, multiple systems, they are trying to access or trying to run the same service. Therefore, this race condition may be a possibility and the coordination service has to, be such that this condition should not occur.

Refer slide time: (7:03)

What is a Deadlock?

- When two processes are waiting for each other directly or indirectly, it is called **deadlock**.



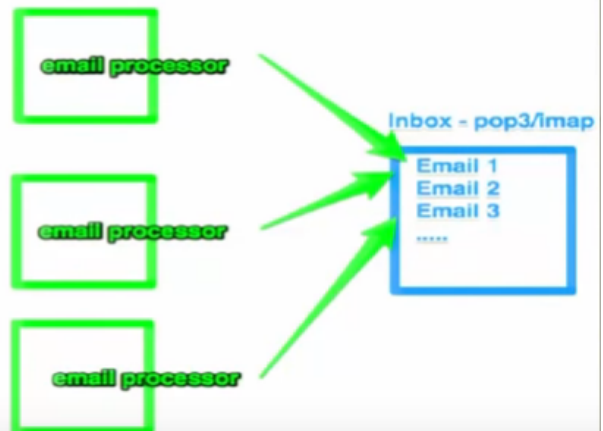
- Here, Process 1 is waiting for process 2 and process 2 is waiting for process 3 to finish and process 3 is waiting for process 1 to finish. All these three processes would keep waiting and will never end. This is called dead lock.

Another problem is about, deadlock. So when two processes are, waiting for each other directly or indirectly. And none of is basically able to progress, then it is called a, 'Deadlock'. so here, in this example, set up waiting processes 1, 2, 3, they are said to be deadlock, if they are waiting for each other, for the resource, allocation and none of them are, finishing for the others to complete. Hence, this circularly waiting process are said to be a deadlock. So in our scenario, in a distributed, computing where different, servers are running different processes and they may be communicating with each other, they may also suffer to this particular problem, which is called a, 'Distributed Dead Lock Problem'. So hence, the situation of coordination service, should be such that, it should not lead to a deadlock.

Refer slide time: (8:03)

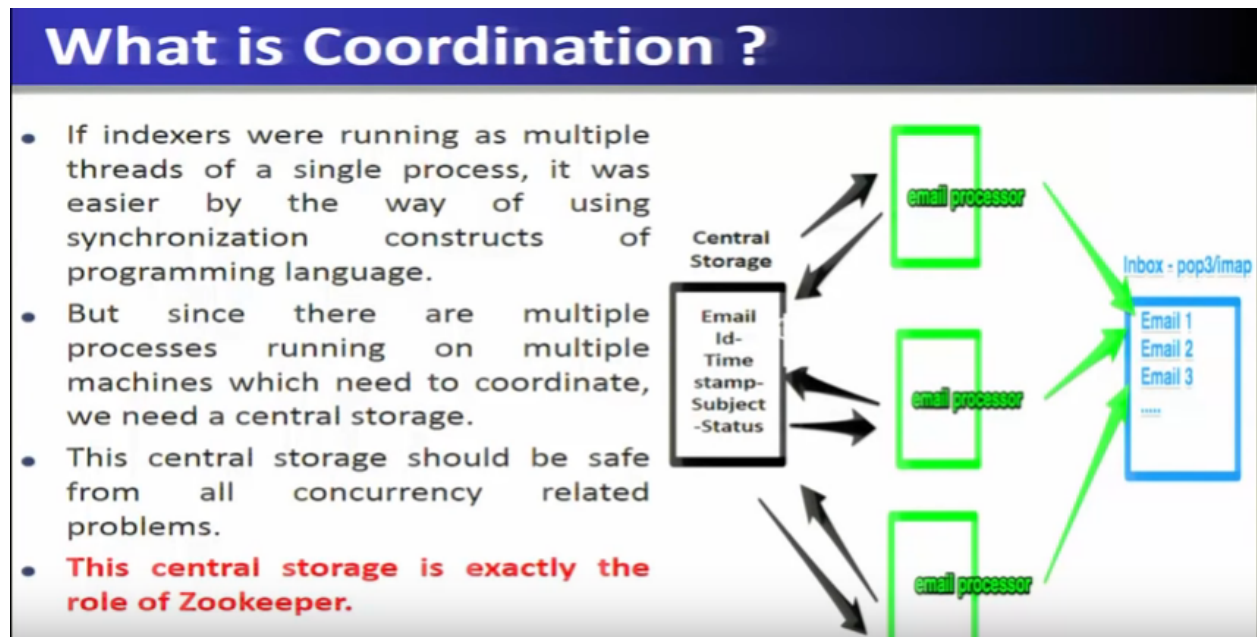
What is Coordination ?

- How would Email Processors avoid reading same emails?**
- Suppose, there is an inbox from which we need to index emails.
- Indexing is a heavy process and might take a lot of time.
- Here, we have multiple machine which are indexing the emails. Every email has an id. You can not delete any email. You can only read an email and mark it read or unread.
- Now how would you handle the coordination between multiple indexer processes so that every email is indexed?



So let us see, some of the example of the coordination service in a distributed system, for big data computations. Let us, understand the email processors. And how they these email process are avoid reading the same email? Using the coordination service, suppose there is an inbox, from which we need to index emails, indexing is a heavy process and might take, a lot of time. Hence, we have multiple machines, which are indexing the emails, every email has an ID and you cannot delete any email, you can only read an email and mark it as, read or unread. Now, how would you handle the coordination, between multiple indexer processes, so that every mail is indexed, that is shown here, in this particular picture.

Refer slide time: (9:10)



So if the indexers were running, as multiple threads on a single machine or a single processor, then it would be, easily handled with the, help of synchronization constructs of a programming language. But, since they area multiple processes running on multiple machines ,we need to coordinate; that is we require, central storage kind of scenario, implemented on a distributed system, which is nothing but a coordination service. So this central storage would be safe from all concurrency related problems ,this central storage is exactly, the role of the coordination, in the Zookeeper.

Refer slide time: (9:49)

What is Coordination ?

- **Group membership:** Set of datanodes (tasks) belong to same group
- **Leader election:** Electing a leader between primary and backup
- **Dynamic Configuration:** Multiple services are joining, communicating and leaving (Service lookup registry)
- **Status monitoring:** Monitoring various processes and services in a cluster
- **Queuing:** One process is embedding and other is using
- **Barriers:** All the processes showing the barrier and leaving the barrier.
- **Critical sections:** Which process will go to the critical section and when?

So again, what is the coordination? That is about, group membership that is the setoff tasks, under the data node; belong to the same group: that is also a coordination service. To elect a leader between the primary and the replicas, that also is a coordination service. than dynamic configuration that is multiple services are joining and communicating and there, that is why common, configuration is required, to be maintained, where in the active servers are to be registered is, also called, 'Dynamic Configuration'. And is a part of coordination service. Status monitoring where monitoring, we are various processes and services in a cluster is maintained, dynamically is also a coordination service, queuing that is one process is, embedding and the other is using that is called, 'Coordination Service'. Barriers all the processes showing the barriers and leaving the barrier is, also a coordination service. Critical sections where processes go into the critical section and then, they may, leave the critical section, so that others can enter into the critical section ,so this kind of critical section is also kind of coordination service.

Refer slide time: (11:22)

What is ZooKeeper ?

- **ZooKeeper is a highly reliable distributed coordination kernel**, which can be used for distributed locking, configuration management, leadership election, work queues,....
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- **Key attributed of such data**
 - Small size
 - Performance sensitive
 - Dynamic
 - Critical
- **In very simple words**, it is a central store of key-value using which distributed systems can coordinate. Since it needs to be able to handle the load, Zookeeper itself runs on many machines.

So what is the Zookeeper? So having understood the coordination service, Zookeeper is a highly reliable distributed coordination kernel, which can be used for distributed locking, configuration, management, leader election, work queues and so on. So, a Zookeeper is a replicated service that holds the metadata of distributed application, by meaning, to say they are a replicated service means, that particular service, runs on the clusters, which is consist of multiple nodes or multiple server machines, that is why it is called the, 'Replicated Service'. So, how that is all done, in the Zookeeper, which is the replicated service, on one hand, while providing the centralized configuration service, that is the coordination service, that is the design goal of the Zookeeper, which we are going to cover. so the key attributes, of such a data which the Zookeeper, requires to maintain in the replicated services are, the size of the data is small and the performance is very sensitive and it is dynamically changing and it is also very critical to manage, in the coordination replicated service, by the Zookeeper. so in a very simple word, the Zookeeper is a central store of key value pairs, using which, the distributed systems can coordinate, since it needs to be able to handle the load, the Zookeeper itself runs on many machines.

Refer slide time: (13:05)

What is ZooKeeper ?

- Exposes a simple set of primitives
- Very easy to program
- Uses a data model like directory tree
- Used for
 - Synchronisation
 - Locking
 - Maintaining Configuration
- Coordination service that does not suffer from
 - Race Conditions
 - Dead Locks

So, the Zookeeper exposes a simple set of primitives, it is very easy to program, uses the data model like, the directory tree and it is used for synchronization, locking, maintaining configuration. So, the coordination service, should also not suffer from, race condition and deadlock, though it has to be providing the synchronization locking and maintaining the configuration, as a service, that is the Zookeeper, does this and it uses, to do this, it uses a data model, which is like a tree, directory tree and it also has a very easy program interface, for so that the programmers can use it, for designing their own distributed application, for big data computation .and it also exposes a simple set of primitives that, we will see.

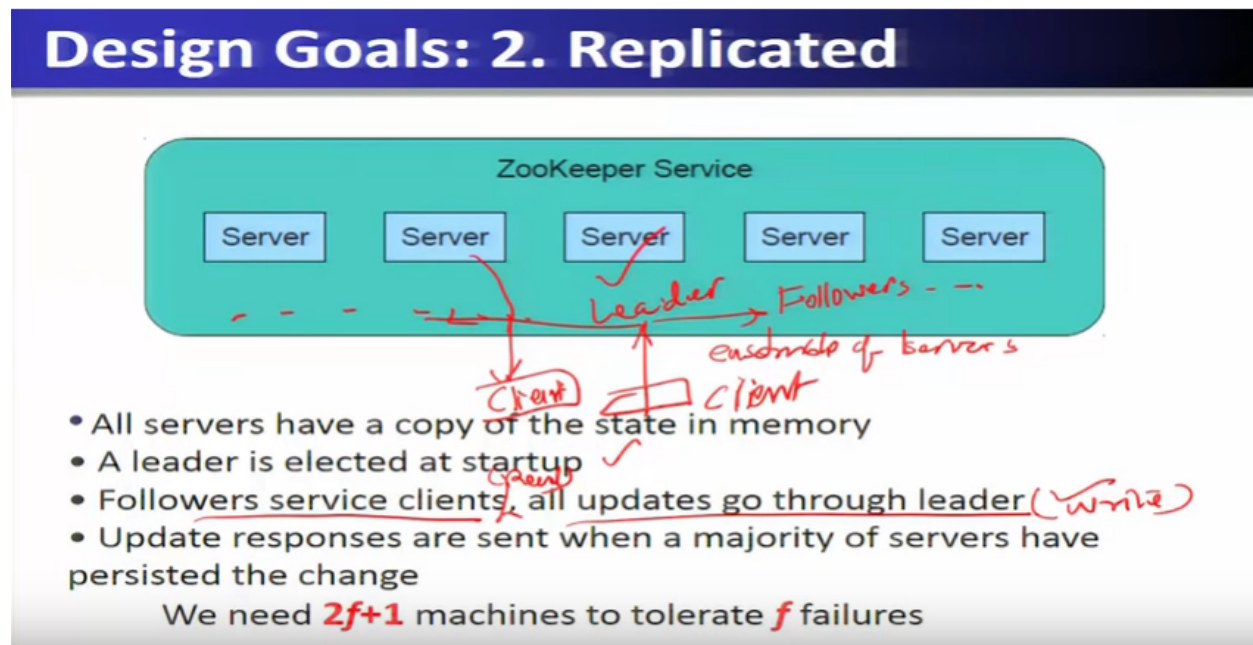
Refer slide time: (14:01)

Design Goals: 1. Simple

- A shared hierarchal namespace looks like standard file system
- The namespace has data nodes - znodes (similar to files/dirs)
- Data is kept in-memory
- Achieve high throughput and low latency numbers.
- **High performance**
 - Used in large, distributed systems
- **Highly available**
 - No single point of failure
- **Strictly ordered access**
 - Synchronisation

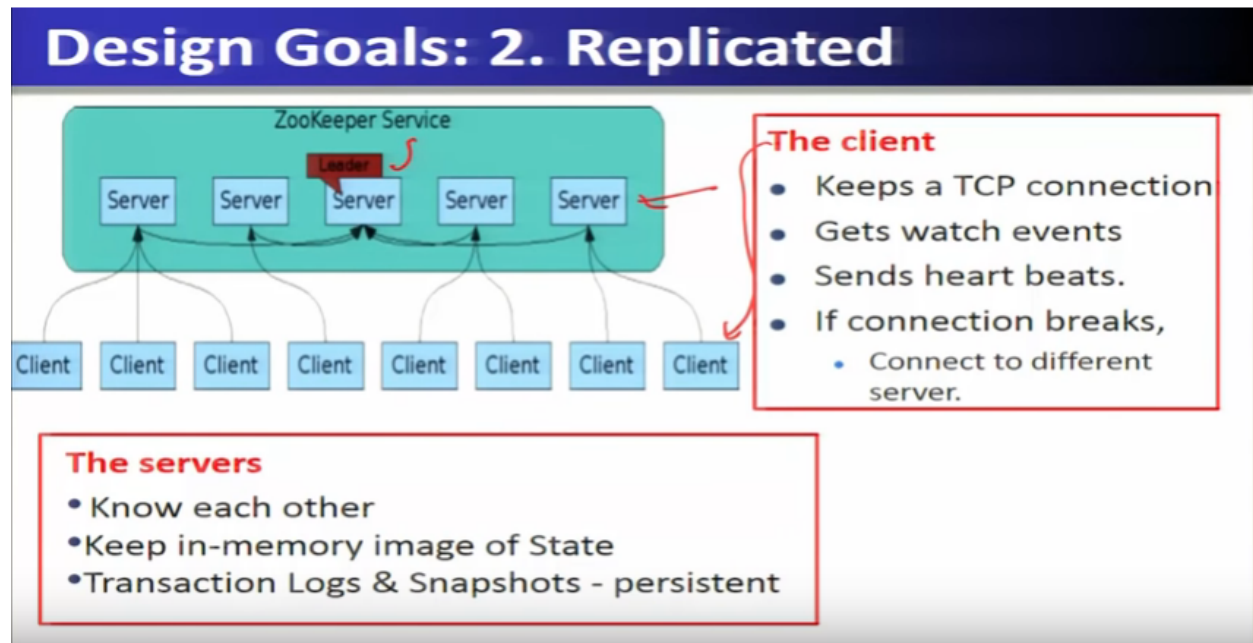
So, the first design goal of Zookeeper is the simple, simplicity. So, it has the shared, hierarchical namespace which looks like, a standard file system of a UNIX. So, the namespace highest the data nodes and these data nodes are, known as Z nodes, which disturb this, which differentiates from the other type of the data node usage. So, the data is kept in memory and will achieve high throughput and low latency numbers. So high performance is, one of the important features of providing, the simple service that is used in the large and distributed system and it is also highly available, that is not a single point of failure, it suffers and it also strictly ordered, the axis that is by providing the synchronization.

Refer slide time: (14:59)



So, the second goal of the Zookeeper is the replication or a replicated service. So here, the Zookeeper runs on, the example of servers. So here we have seen, there are five different servers, it can be many more also, which runs the Zookeeper service, so it is an ensemble of servers. so it is not one server, on which the Zookeeper will run, but it is an ensemble observers on the Zookeeper will run, so all the servers have the copy of the state, in the memory and now, since all servers have the same state .therefore, leader will be elected, at the I startup and at all the time let us say, this is the real leader and all others are the followers. So, the leader is elected as I start up and at all point of time, a leader is always elected, to run this Zookeeper service. Now, the followers service, the clients, all the updates go through the leader, meaning to say that, if there is a write operation by the client, it has to go through the leader, whereas the read operation by the client, can be done through any of these available servers, in the end sample. so therefore ,it is written that, the followers service the clients, I but all the updates has to go through the leader. Updates in the sense, the write operations has to be channeled through, the leader whereas all the read operations, can be performed or can be serviced, to the client from any of the followers therefore. Update responses are sent, when a majority of the servers have persisted the change. So here we need, the majority we need $2F + 1$ machines, to tolerate F failures.

Refer slide time: (17:20)



So this, concept can be understood here, in this diagram that, you can see that the client, can be connected to any of the servers, whether it is follower or it is the leader. But, as far as read, read operations are concerned, the client can be serviced, by any of the follower servers as far as, the write or update operation is concerned, it has to go through the leader elected and out of these servers, at this point of time, this particular server is denoted as, the leader. So here, we have the set of servers, they are they will know each other and they will keep in memory image of the state and also maintains the transaction log and snapshots, persistently. Whereas the other, this is one such entity, in the Zookeeper, the other entity is called a, 'Client'. So, the client keeps a TCP connection, gets a watch events and sends the heartbeats, ID of the connection breaks, they will connect to a different servers. So these are, the client's activity in this replicated service, of Zookeeper.

Refer slide time: (18:36)

Design Goals: 3. Ordered

- ZooKeeper stamps each update with a number
- **The number:**
 - Reflects the order of transactions.
 - used implement higher-level abstractions, such as synchronization primitives.

Third important goal of Zookeeper is, about the ordering. So, so the Zookeeper will time stamp each updates, with a number, so the number is nothing but, it reflects the order of the transaction and it is used, to implement the high level abstraction, such as synchronization primitives. We will see this aspect in more detail in the further slides.

Refer slide time: (19:04)

Design Goals: 4. Fast

Performs best where reads are more common than writes, at ratios of around 10:1.

At Yahoo!, where it was created, the throughput for a ZooKeeper cluster has been benchmarked at over 10,000 operations per second for write-dominant workloads generated by hundreds of clients

Now fourth design goal is, about the speed, which is fast. So, it performs best when the reads are more common, than writes, at the ratio of around 10 is to 1. So, at Yahoo! where it was created the Zookeeper, the throughput of the Zookeeper cluster, has been benchmarked at over 10,000 operations per second for the write, dominant workloads generated by hundreds and thousands of clients.

Refer slide time: (19:33)

Data Model

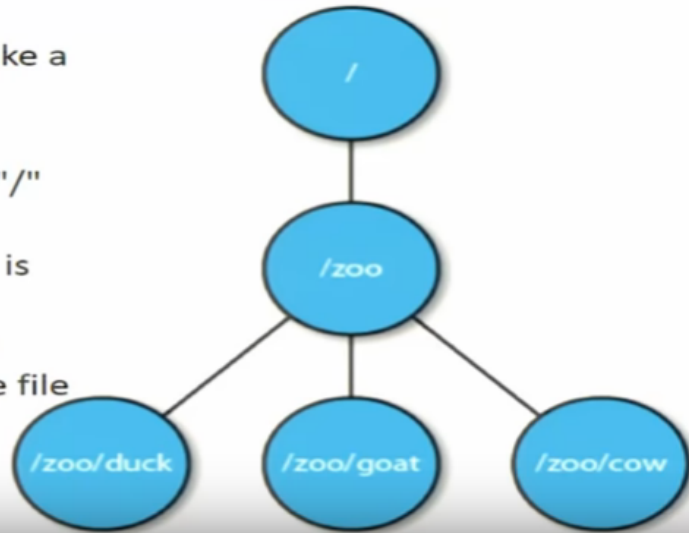
- The way you store data in any store is called **data model**.
- **Think of it as highly available fileSystem:** In case of zookeeper, think of data model as if it is a highly available file system with little differences.
- **Znode:** We store data in an entity called znode.
- **JSON data:** The data that we store should be in JSON format which Java script object notation.
- **No Append Operation:** The znode can only be updated. It does not support append operations.
- **Data access (read/write) is atomic:** The read or write is atomic operation meaning either it will be full or would throw an error if failed. There is no intermediate state like half written.
- **Znode:** Can have children

Let us see the data model. The way you store data in any store is called, 'Data Model'. Think of it as highly available file system: in case of Zookeeper, think of a data model, as if it is highly available file system, with the little differences. Here, in this file system of Zookeeper, the nodes are called, 'Z Nodes', where we store, the data in an entity, which is also called as a 'Znode'. The JSON data: the data that means store, should be in the JSON format, which Java Script objection, object notation follows. And it has the append operation, so Z node can only be updated, it does not support, the append operation. So, so the data axis here is, that is the read and write operations, they are atomic, so read and write operation is atomic operations, meaning either it will be full or would throw an error a field. Therefore, in no intermediate state, like half written is possible in the atomic node. and Znode can have the children's, so again we are repeating that no append operation, so Z node can only be updated, it does not support the append operations, in this particular scenario.

Refer slide time: (21:05)

Data Model Contd...

- So, znodes inside znodes make a tree like hierarchy.
- The top level znode is "/".
- The znode "/zoo" is child of "/" which top level znode.
- duck is child znode of zoo. It is denoted as /zoo/duck
- Though "." or ".." are invalid characters as opposed to the file system.



So, here we can see that the Z nodes, inside the Z nodes will make like a tree hierarchy. where the top-level Z node is the root node, which is the top level Z node and let us say that, then child is let us say, a Z node of this top node, zoo and this is called now the, 'Duck' and so on. So there will be a hierarchical way, these Z nodes will be organized in the, the namespace. So it provides the data model, of the hierarchical tree structure, to support the namespace.

Refer slide time: (21:52)

Data Model – Znode - Types

- **Persistent**
 - Such kind of znodes remain in zookeeper untill deleted. This is the default type of znode. To create such node you can use the command: `create /name_of_myznode "mydata"`
- **Ephemeral**
 - Ephemeral node gets deleted if the session in which the node was created has disconnected. Though it is tied to client's session but it is visible to the other users.
 - An ephemeral node can not have children not even ephemeral children.

So, Z node has various types, there are two different type of Z nodes, one is called, 'Persistent', such kind of Z nodes, will remain in Zookeeper until it is deleted. This is the default type of Z node. To create

such a node you can use the command: create slash name of my Z node, my data. Ephemeral type of Z node. Ephemeral nodes gets deleted, if the session in which, the node was created has disconnected. Though it is tied to the client session, but it is visible to the other users. So, an ephemeral node, cannot have the children not even the ephemeral children's. So there are two different type of Z node, persistent and ephemeral, we have just discussed. So, ephemeral is related to the session, whereas persistent it will remain in the Zookeeper, until it is deleted.

Refer slide time: (22:50)

Data Model – Znode - Types

- **Sequential**
 - Creates a node with a sequence number in the name
 - The number is automatically appended.

<pre>create -s /zoo v Created /zoo0000000008</pre>	<pre>create -s /zoo/ v Created /zoo/0000000003</pre>
<pre>create -s /xyz v Created /xyz0000000009</pre>	<pre>create -s /zoo/ v Created /zoo/0000000004</pre>

- The counter keeps increasing monotonically
- Each node keeps a counter

Now, another type is called, 'Sequential', will create a nodes, with the sequence numbers, in the namespace, the number is automatically appended. And the counters will keep on, monotonically increasing. So each node has a counter. so here we can, see that the, the Z node which is created has, this kind of sequence numbers and whenever the new nodes are increasing, added, so these numbers are keep on, now increasing, whenever the nodes are, whenever a new node is created with a sequence number.

Refer slide time: (23:33)

Architecture

- **Zookeeper can run in two modes: (i) Standalone and (ii) Replicated.**

(i) Standalone:

- In standalone mode, it is just running on one machine and for practical purposes we do not use standalone mode.
- This is only for testing purposes.
- It doesn't have high availability.

(ii) Replicated:

- Run on a cluster of machines called an ensemble.
- High availability
- ~~Tolerates as long as majority.~~

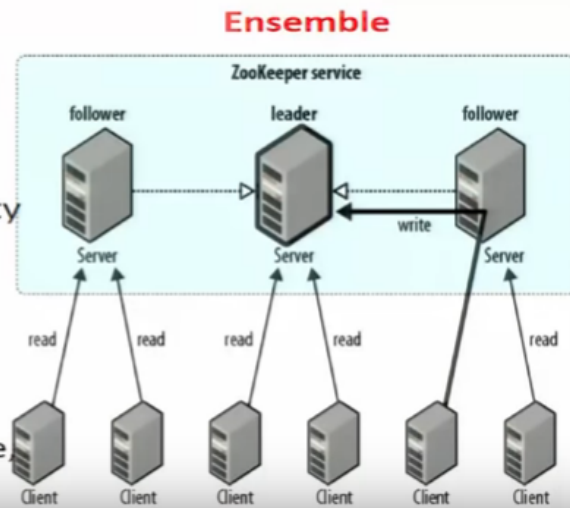
So, the architecture of Zookeeper, we can see here that Zookeeper can run in two modes, one is standalone, the other is called replicated mode. In a standalone mode, it is just running one, one machine and for practical purpose, we do not use this standalone mode. This is only for testing purposes and does not have, the high availability because, it is running on only one machine and if it fails, the entire operation will stop functioning. Therefore, it is only meant for testing purposes. The other mode of operation in the Zookeeper is called, 'Replicated Mode', it is used for the production clusters in, in this mode. So, it runs on the cluster of machines called, 'Ensemble'. And it ensures high availability and also tolerates, as long as, it ensures the majority of the machines are, in active service.

Refer slide time: (24:28)

Architecture: Phase 1

Phase 1: Leader election (Paxos Algorithm)

- The machines elect a distinguished member - leader.
- The others are termed followers.
- This phase is finished when majority sync their state with leader.
- If leader fails, the remaining machines hold election. takes 200ms.
- If the majority of the machines aren't available at any point of time, the leader automatically steps down.



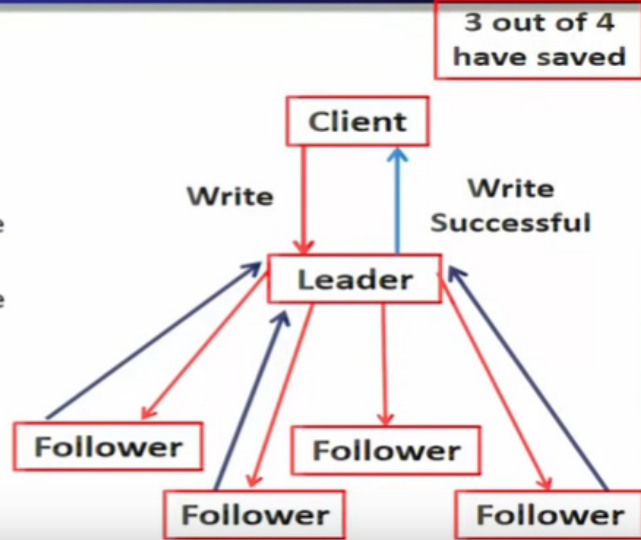
So, let us see the architecture: and in this architecture, this Zookeeper runs in two phases. so the phase one is about the leader election, which uses the Paxos like algorithm, which is called, 'Z service'. So, which is called, 'Job, Z operations'? So the machines will elect the distinguished members, which is called the, 'Leader over Here'. Which is shown here, as the Zookeeper service in the phase 1 and the other, servers are termed as the followers, in the ensemble, servers for running the Zookeeper service. Now, this phase is finished when the majority, will sync their state with the leaders and if the leader fails, the remaining machines will hold the election and elect another, follower as the leader and this process of leader election takes only 200. Now, if the majority of the machines aren't available at any point of time, the leader automatically steps down. So leader election has to be ensured if the majority rule follows and therefore, the majority rule is important, to functioning of the Zookeeper.

Refer slide time: (25:56)

Architecture: Phase 2

Phase 2: Atomic broadcast

- All write requests are forwarded to the leader,
- Leader broadcasts the update to the followers
- When a majority have persisted the change:
 - The leader commits the up-date
 - The client gets success response.
- The protocol for achieving consensus is atomic like two-phase commit.
- Machines write to disk before in-memory



Now in phase 2, there will be an atomic broadcast, that is all right requests are forwarded to the, to the leader. And so, the client you see here is forwarding the right request to the leader. and the leader broadcast the updates to all the follower, when a majority have persisted the change, that is the leader commits, the update and the client gets the success response, then the protocol for achieving the consensus is atomic and this is like two-phase commit protocol. So the machines will write, to the disk before in memory.

Refer slide time: (26:48)

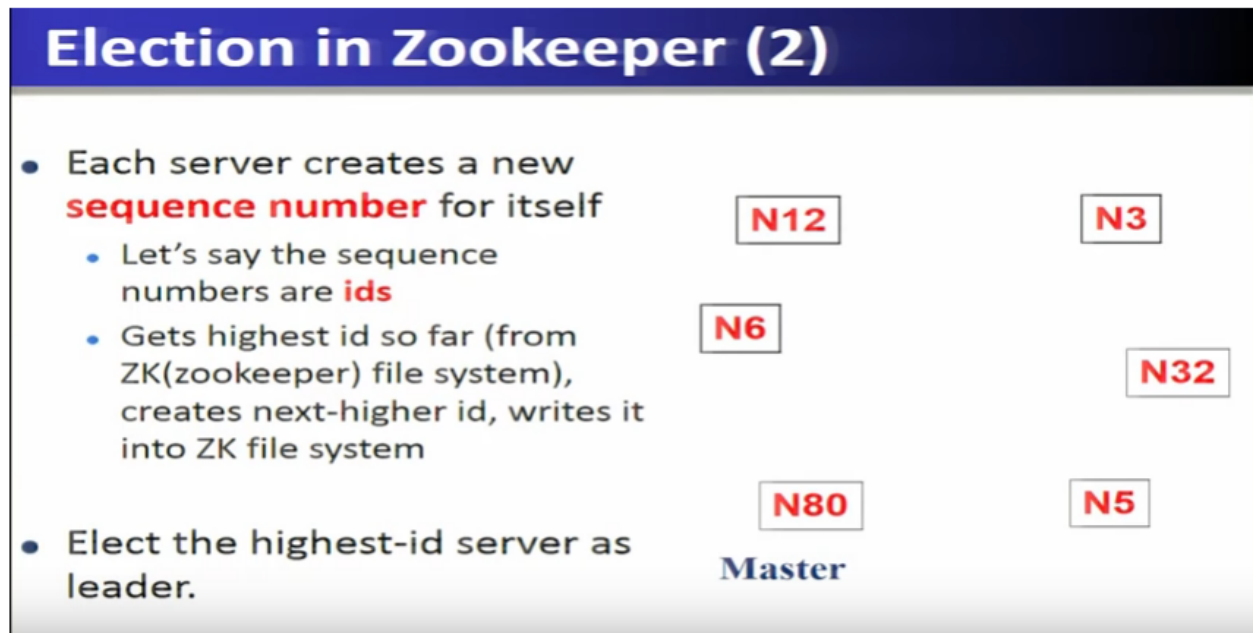
Election in Zookeeper

- Centralized service for maintaining configuration information
- Uses a variant of Paxos called Zab (Zookeeper Atomic Broadcast)
- Needs to keep a leader elected at all times

Reference: <http://zookeeper.apache.org/>

So, now let us see in more detail about, the phase one, which is the election process in Zookeeper. So, Zookeeper is the centralized service, for maintaining the configuration information. Now, Zookeeper uses, a variant of Paxos which is called, 'Zab', that is Zookeeper atomic broadcast. Now, Zookeeper needs to keep, the lead leader elected at all point of time, how that is leader is elected, we will understand the complete process in the Zookeeper.

Refer slide time: (27:20)



So, each server will create at the new sequence number for itself. and this particular sequence number is stored in a file, so let's say, that the sequence numbers are the node IDs and the highest node ID seen so far, from the Zookeeper file system, will create the next higher ID, I had write sit into the Zookeeper file. Now, elect the highest ID server, as the leader. So, here in this example, the different servers will have their, IDs and a t +, 6 + ,12 +, 3+, 32, + n5, among these six different servers, the one with the highest ID is an 80 and it is now, after this election it will be elected as, the leader or the master.

Refer slide time: (28:20)

Election in Zookeeper (3)

- **Failures:**

- One option: everyone monitors current master (directly or via a failure detector)
 - On failure, initiate election
 - Leads to a flood of elections
 - Too many messages



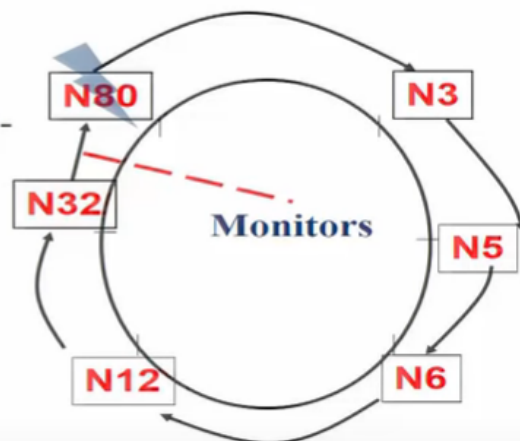
Now, what about the failures? Now, during the election or after the election that the master may crash. So one option is that, everyone monitors, the current master, directly why are the failure detectors. So that means, in normal cases, leader election is not happening, again and again even if the master fails, so every other node monitors the current master. and whenever it detects, that the master is failed, it will initiate the election, that means, if they initiate the election it will lead to a flood of lot of messages and too many messages, therefore this option, of whenever there is a failure, everyone start electing the leader, this will require a flood of messages.

Refer slide time: (29:15)

Election in Zookeeper (4)

- **Second option:** (implemented in Zookeeper)

- Each process monitors its next-higher id process
- **if** that successor was the leader and it has failed
 - Become the new leader
- **else**
 - wait for a timeout, and check your successor again.



Therefore, we will see a second option, which is implemented in Zookeeper. Where in the each process monitors its next higher ID process. For example, process number 32 will monitor process number 80 and 80 will monitor 3 and 3 will monitor 5 and 5 will monitor 6 and 6 will monitor 12 and 12 will monitor 32. So, each process will monitor its next higher ID process. Now, if that successor, to which a process is monitoring, is detected as the failed and if that, failed process, happens to be the leader, then the, that particular process, who has detected, this leader to be failed becomes the new leader. So, there is no leader election required, here in this case of Zookeeper, otherwise then it will wait for a timeout. And check the, the successor again.

Refer slide time :(30:11)

Election in Zookeeper (5)

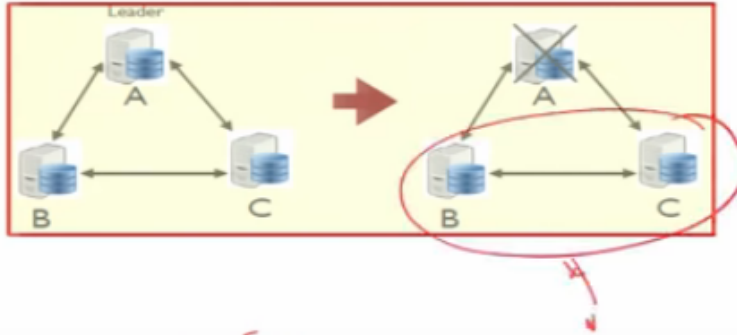
- What about id conflicts? What if leader fails during election ?
- To address this, Zookeeper uses a **two-phase commit** (run after the sequence/id) protocol to commit the leader
 - Leader sends NEW_LEADER message to all
 - Each process responds with ACK to at most one leader, i.e., one with highest process id
 - Leader waits for a majority of ACKs, and then sends COMMIT to all
 - On receiving COMMIT, process updates its leader variable
- Ensures that safety is still maintained

So, therefore, what about the IDs conflict, what about if the leader fails during the election in both these cases, to handle these cases the Zookeeper, does this election, using two-phase commit protocol. So, Zookeeper runs after the sequence ID and this particular protocol, used to commit the leader. So, leader will send the new leader message, to all of the other processes and each process, will respond with the acknowledge, to at most one leader, that is one with the highest process ID seen. So, far the other processes will respond, as an acknowledgement, allowing that particular process, to become the leader. Now leader waits for the majority of acknowledgments and then sends the commit, to all on receiving the commit protocol, on receiving the commit message, the process updates its leader variable and it ensures that the safety is always maintained in this two-phase leader election protocol.

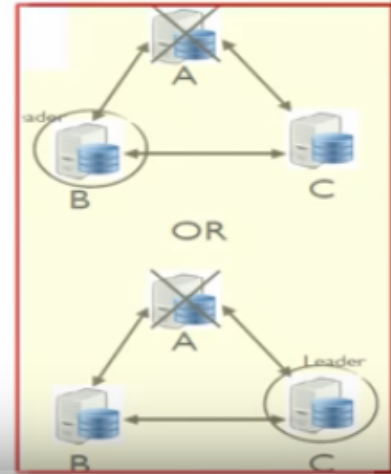
Refer slide time :(31:20)

Election Demo

- If you have three nodes A, B, C with A as Leader. And A dies. Will someone become leader?



Yes. Either B or C.

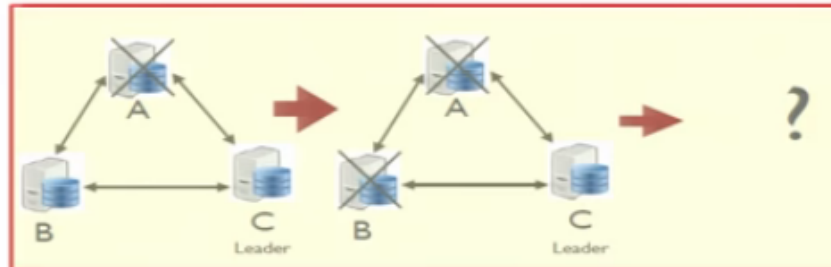


Let us see the election process, demonstration in Zookeeper, now if you have three nodes ABC with leader elected as a node shown on the left side of the diagram. And if a dies will someone become the leader, someone that is the remaining B and C will become the leader. So, if a dies then B and C which becomes, the majority, of three that is two out of three is the majority. So, majority survives hence, out of B and C one will be elected as the leader. So, the answer is yes that means either B or C can become, the elect can become the leader why because, even after the failure of a the remaining node set, remains in the majority.

Refer slide time :(32:16)

Election Demo

- If you have three nodes A, B, C And A and B die. Will C become Leader?



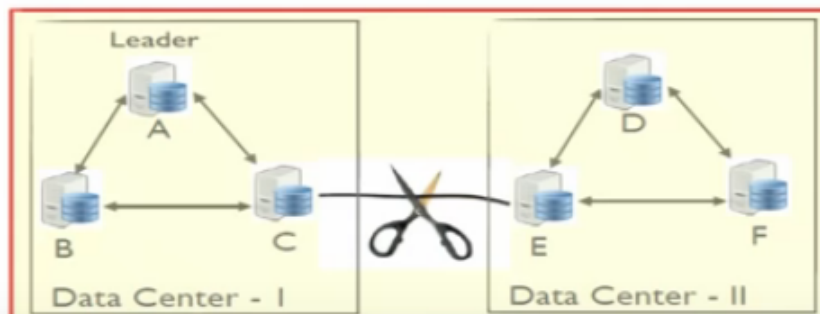
No one will become Leader.
C will become Follower.
Reason: Majority is not available.

Now what about if two nodes are filled, it is not only one but two nodes are filled. So, the remaining one is node number, one out of three, one out of three is not in a majority, therefore if you have three nodes. And two of them dies, then can the C will the C becomes the leader, the answer is no, no one will become the leader, why because C become the follower and the reason is that the majority, is not available if two nodes a dies, therefore the leader election, will happen as long as, the after the failure the majority, serve is majority of the server it is surviving.

Refer slide time :(33:06)

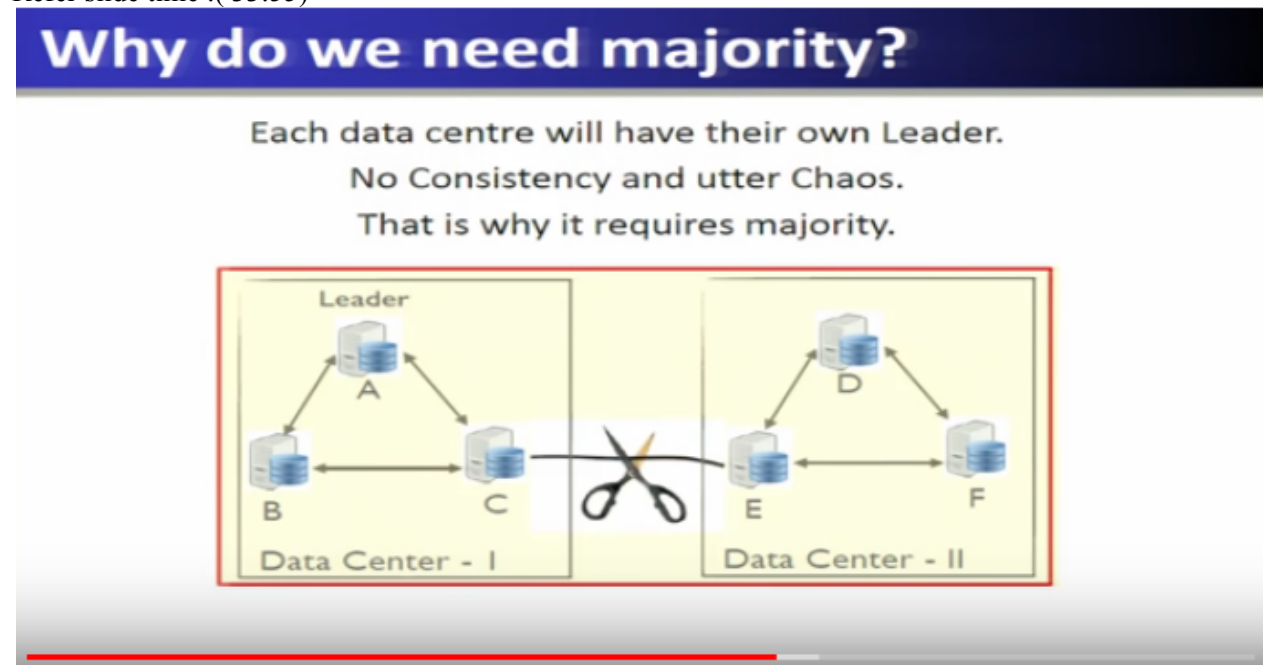
Why do we need majority?

- **Imagine:** The network between data centres got disconnected. If we did not need majority for electing Leader,
- **What will happen?**



Now let us see why the majority, is important feature or important parameter in the leader election. So, imagine that you have an ensemble, of servers which are spreaded across, two different data centers. So, this example shows that, you have six servers which are spreaded across two data centers that is three on one data center and remaining three out of six, will be on the other data center. Now if let us say there will be a disconnection, which is the network disconnection, which connects these two data center. So, imagine the network between the data center got disconnected, now if we did not need the majority, of electing the leader then what will happen that every data center will be electing one leader.

Refer slide time :(33:55)



And therefore in this particular scenario, there will be two leaders, appointed and in this case there will be violation, of leader election, there cannot be two leaders at the same point of time, hence therefore the majority is required,

Refer slide time :(34:17)

Sessions

- Lets try to understand **how do the zookeeper decides to delete ephemerals nodes** and takes care of session management.
- A client has list of servers in the ensemble
- It tries each until successful.
- Server creates a new session for the client.
- A session has a timeout period - decided by caller

To avoid this kind of scenarios, now we will see about the sessions which are maintained in the Zookeeper. Now let us try to understand, how the Zookeeper decides to delete the ephemeral nodes and take care of session management. So, a client has the list of servers, in the example and it tries each until it becomes successful. And the server creates a, new session for the client; a session has the timeout period, which is decided by the caller.

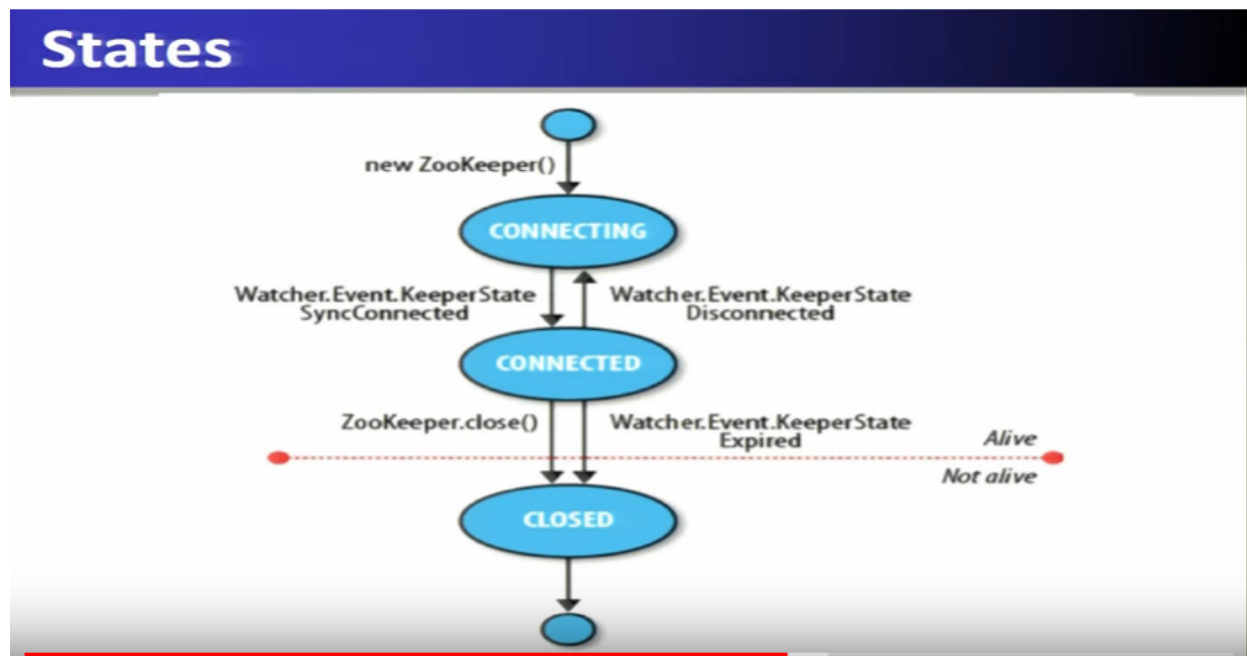
Refer slide time :(34:51)

Contd...

- If the server hasn't received a request within the timeout period, it may expire the session.
- On session expire, ephemeral nodes are lost
- To keep sessions alive client sends pings (heartbeats)
- Client library takes care of heartbeats
- Sessions are still valid on switching to another server
- Failover is handled automatically by the client
- Application can't remain agnostic of server reconnections
 - because the operations will fail during disconnection.

So, if the server hasn't received the request, within a particular timeout period, it may expire the session, on session expiry ephemeral nodes are lost, to keep the session alive, the client sends the ping that is the heartbeats and the client library takes care of heartbeats, sessions are still valid on switching to another server. So, failover is handled automatically, by the client and applications cannot remain agnostic of the server reconnections, because the operations, will feel during the disconnection.

Refer slide time :(35:23)

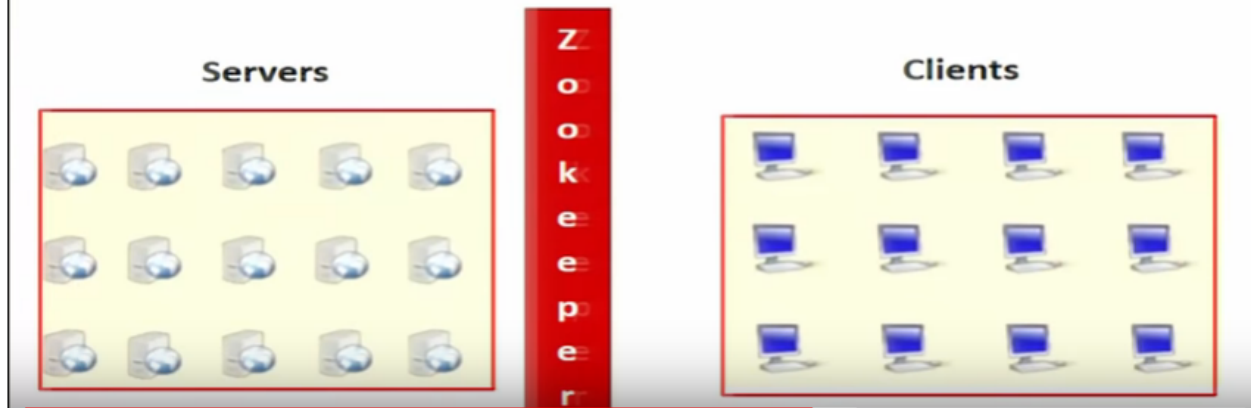


So, this is automatically handled by, the Zookeeper let us understand that when a new ZooKeeper process is starting, about connecting to it then it will watch, the events and keep the state synchronized, with the connected, with the clients that is called, 'Connected'. So, the Zookeeper, when it is closing the connection, then it will be closed. And in between it will watch the events, Keeper State and this particular, when it is expired, then it will send that the watch event state is disconnected.

Refer slide time :(36:10)

Use Case: Many Servers How do they Coordinate?

- Let us say there are many servers which can respond to your request and there are many clients which might want the service.

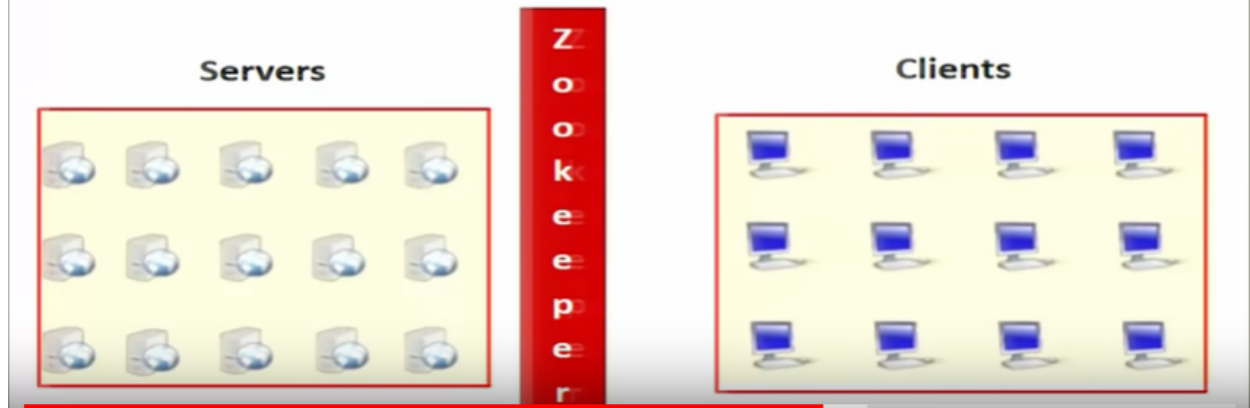


So, these state transitions are handled, now let us see the use case where many servers are available and how the coordination service, will ensure the availability of server, to any applications, which is running on the server. So, now let us see that there are many servers which can respond, to your request and there may be manic line, which one this particular service to none. So, the client want to know which of these servers are alive at a, particular point of time and directly, every client instead of maintaining the list of servers, which are active and keep on updating, at their level instead of that let us keep a coordination service, which's called a, 'Zookeeper'. Now Zookeeper will coordinate between, all the set of servers and between the set of all different clients.

Refer slide time :(36:58)

Use Case: Many Servers How do they Coordinate?

- Let us say there are many servers which can respond to your request and there are many clients which might want the service.

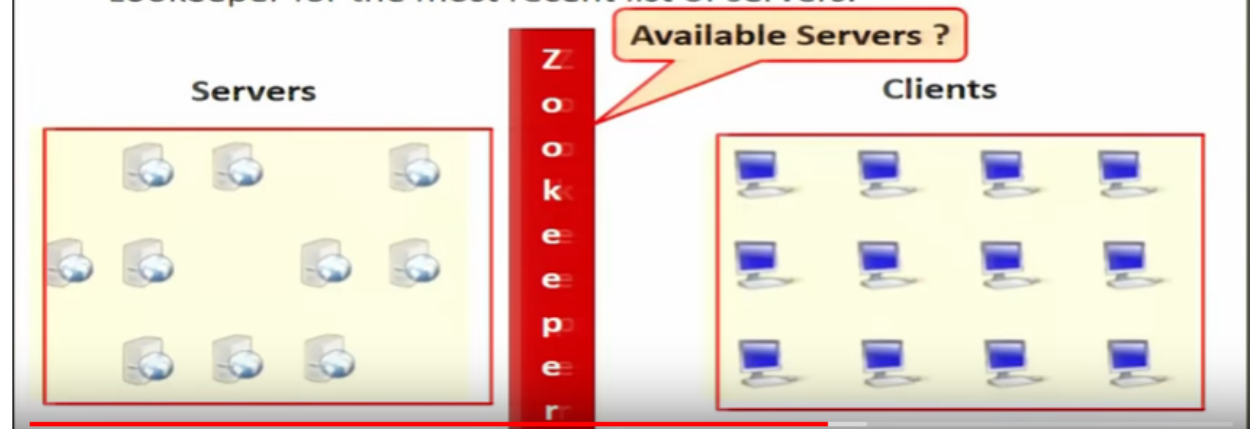


So, from time to time, some of the servers will keep going down. How can all client can keep track of these different servers?

Refer slide time :(37:11)

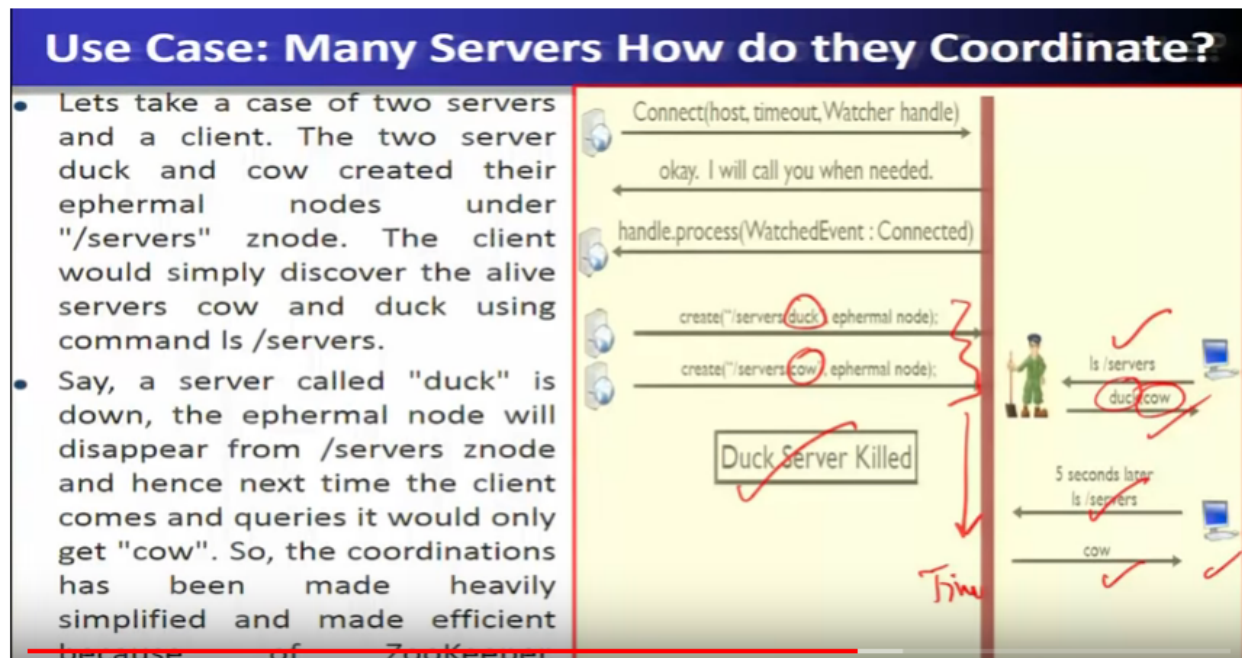
Use Case: Many Servers How do they Coordinate?

- It is very easy using zookeeper as a central agency. Each server will create their own ephemeral znode under a particular znode say `"/servers"`. The clients would simply query zookeeper for the most recent list of servers.



So, the available servers will be now maintained, not by the client, but by the service which is called a, 'Zookeeper'.

Refer slide time :(37:18)



Now let us see let us take the case, of two servers and a client. So, the two servers duck and cow are created, by the created, they are ephemeral nodes/ server, which is shown over here, the client would simply discover, the alive servers, cow and duck using the commands / servers. Now after some time the server called, ' Duck, ' is now down. So, both are now ephemeral nodes which are now created with the name duck and the cow and after, some time, this time flows down, after some time this particular server dog, got killed. So, then this particular client, when it again accesses slash servers, it will see only that cow is visible. So, earlier it was duck and cow two servers, were alive because they have created an ephemeral nodes. So, a server called, ' Duck, ' is down the ephemeral nodes, will disappear from the / server Z node and hence, the next time the client comes up and queries, it will only get the cow. So, the core Nations has to be made heavily simplified, in this Zookeeper and made efficient also, because of the Zookeeper.

Refer slide time :(37:18)

Guarantees

- **Sequential consistency**
 - Updates from any particular client are applied in the order
- **Atomicity**
 - Updates either succeed or fail.
- **Single system image**
 - A client will see the same view of the system, The new server will not accept the connection until it has caught up.
- **Durability**
 - Once an update has succeeded, it will persist and will not be undone.
- **Timeliness**
 - Rather than allow a client to see very stale data, a server will shut down,

Now Zookeeper, guarantees the sequential, consistency that is the updates from any particular clients are applied, the order in which these updates are done, that is called, 'Sequential Consistency'. Now second thing which second property, which the Zookeeper guarantees is about atomicity, we see that the updates are either completely, successful or null that is either it is fully done or it is not done, that is called, 'Atomicity Property', to ensure the consistency, of the dataset. So, single system image that is a client will see a single, view of the system. So, when a new server will not accept the connection, until it has been cut up. So, therefore every client so, the clients will see the same view, of the set of systems or the state variable here in case of Zookeeper. Now another thing which guaranteed, by the Zookeeper is called, 'Durability', that is once the update has been successful it will persist and it will not be undone, another part is called, 'Timeliness'. So, rather than allowing, the client to see the very steel data, the server will in this case will shut down.

Refer slide time :(40:07)

Operations

OPERATION	DESCRIPTION
create	Creates a znode (parent znode must exist)
delete	Deletes a znode (mustn't have children)
exists/ls	Tests whether a znode exists & gets metadata
getACL, setACL	Gets/sets the ACL for a znode getChildren/ls Gets a list of the children of a znode
getData/get, setData	Gets/sets the data associated with a znode
sync	Synchronizes a client's view of a znode with ZooKeeper

So, these are some of the operations, of the Zookeeper create delete, exist LS get a CL that is access control list, set a CL get data set data sync, will not go in more detail.

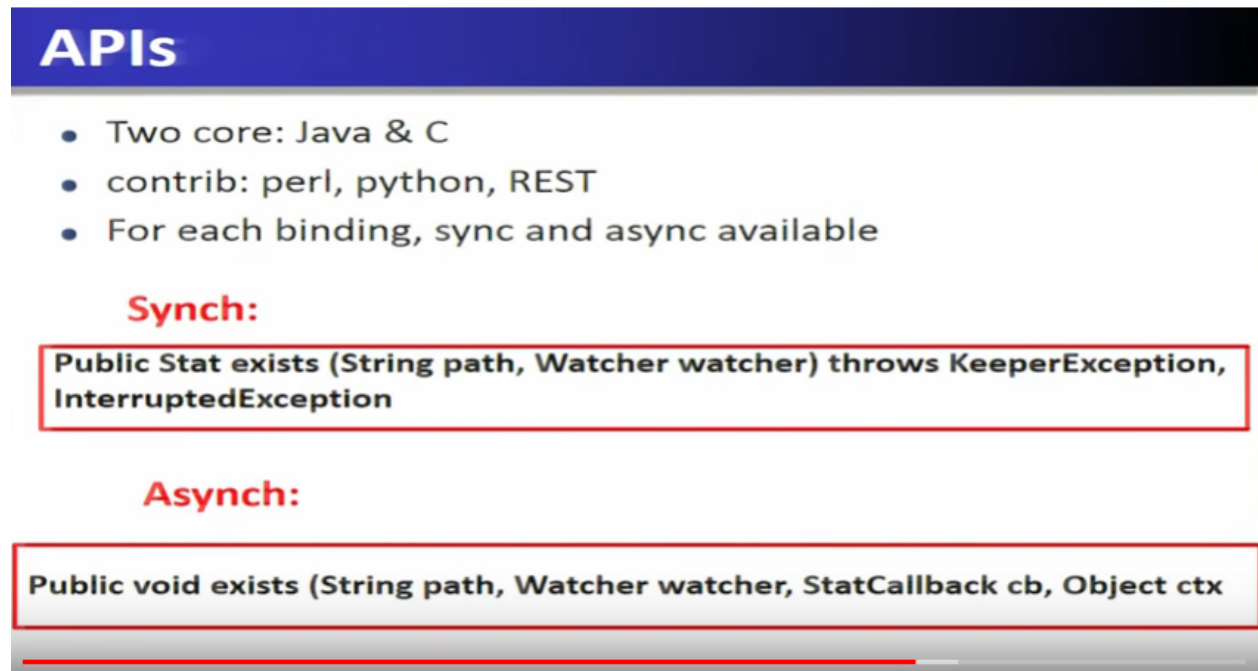
Refer slide time :(40:22)

Multi Update

- Batches together multiple operations together
- Either all fail or succeed in entirety
- Possible to implement transactions
- Others never observe any inconsistent state

And you will see we have just shows you the, the type of operations, commands which are supported by the, the Zookeeper which are very simple. Now multi updates here the batches, together supports the multiple operations, either all or either all are none is done in entirety and possible, to implement the transition in this scenario and others will never observe any inconsistent state.

Refer slide time :(40:22)



The slide is titled "APIs" in a blue header. It lists three bullet points: "Two core: Java & C", "contrib: perl, python, REST", and "For each binding, sync and async available". Below the list, there are two sections. The first is labeled "Synch:" in red, followed by a red-bordered box containing the text "Public Stat exists (String path, Watcher watcher) throws KeeperException, InterruptedException". The second is labeled "Asynch:" in red, followed by a red-bordered box containing the text "Public void exists (String path, Watcher watcher, StatCallback cb, Object ctx".

APIs

- Two core: Java & C
- contrib: perl, python, REST
- For each binding, sync and async available

Synch:

Public Stat exists (String path, Watcher watcher) throws KeeperException, InterruptedException

Asynch:

Public void exists (String path, Watcher watcher, StatCallback cb, Object ctx

Different APIs available and out of them two coreopsis' are Java and C and other contributions are Perl Python rest and for each building, the synchronization and asynchronous, things are available, which are therein.

Refer slide time :(41:10)

Watches

- Clients to get notifications when a znode changes in some way
- Watchers are triggered only once
- For multiple notifications, re-register

Now another part which's available in the Zookeeper is called, 'Watches', so, clients to get notifications when a Z node changes in some way, is done through the watches. So, Watchers are triggered only once for multiple notations, they have to reread Easter. So, what triggers are available in Zookeeper and this is used, to get the notations, notifications for the clients. So, whenever there are a Z no changes, in some way the clients will get the notification.

Refer slide time :(41:58)

Watch Triggers

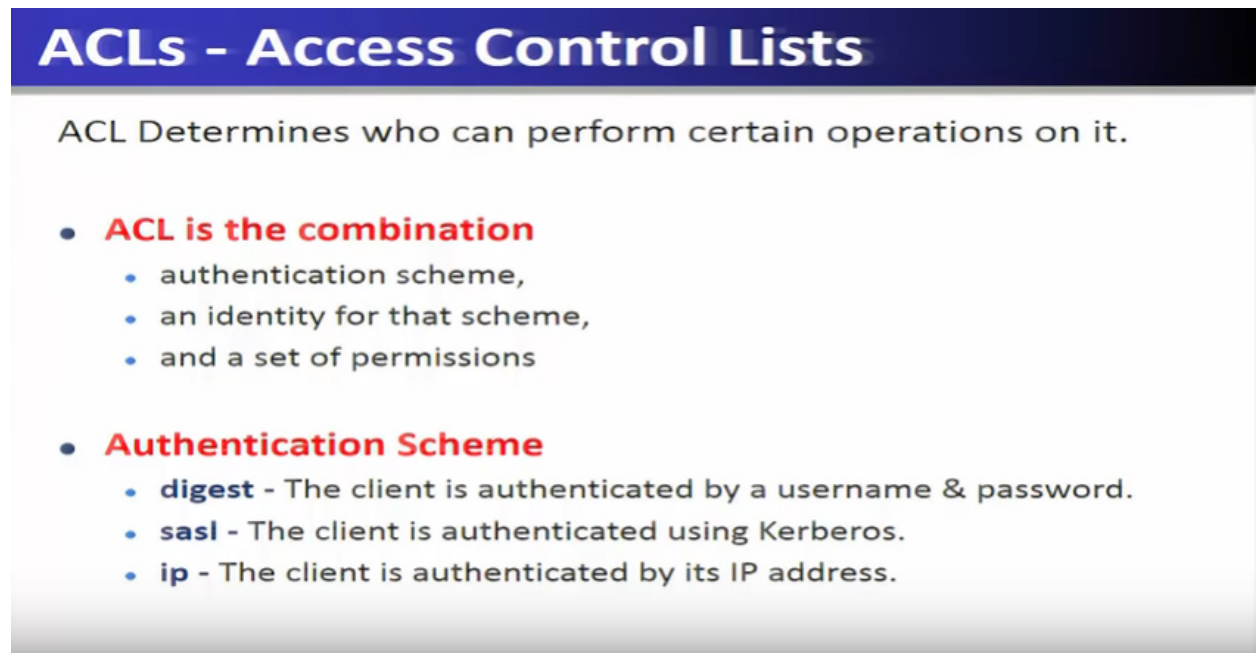
- The read operations exists, getChildren, getData may have watches
- Watches are triggered by write ops: create, delete, setData
- **ACL (Access Control List)** operations do not participate in watches

WATCH OF ...ARE TRIGGERED	WHEN ZNODE IS...
exists	created, deleted, or its data updated.
getData	deleted or has its data updated.
getChildren	deleted, or its any of the child is created or deleted

So, what triggers so, the read operations which exists here is to get children, get data may have watches, which are triggered operations. So, watches are triggered by, the right operation, such as create delete set

data, then there is an access control list, of operation ,which do not participate in the watches. So, watch of the following are triggered, that is exists create delete or its data is updated, in the Z node then it gets triggered, using exists command. Now in the get data that means when in the Z node the operation is deleted or it has that data updated, then using this particular gate data, it will get the trigger similarly, when the Z node gets deleted or it's any of a children is created or deleted then using a get, children the watch triggered is triggered, using get children.

Refer slide time :(43:22)



ACLs - Access Control Lists

ACL Determines who can perform certain operations on it.

- **ACL is the combination**
 - authentication scheme,
 - an identity for that scheme,
 - and a set of permissions
- **Authentication Scheme**
 - **digest** - The client is authenticated by a username & password.
 - **sasl** - The client is authenticated using Kerberos.
 - **ip** - The client is authenticated by its IP address.

So, in access control list, access control list determines, who can perform certain operations on it success control list is a combination, of authentication scheme and identity for that scheme and a set of permissions. So, authentication is scheme, contains the digest, that is the client is authenticated by the username and the password, which is nothing but it digest, generates a choices then SASL is that the client is authenticated using curved arrows and it will give a session, ACL list that is called SASL IP is the client is authenticated, by its IP address then it is an IP.

Refer slide time :(44:15)

Use Cases

Building a reliable configuration service

- A Distributed lock service

Only single process may hold the lock

Now use cases, is that building a reliable configuration service, using curvy nose, now another use case is the distributed lock service, that is only a single process, may hold the lock at any point of time, this distributed lock service is provided by the Zookeeper. Now let us see when not to use this

Refer slide time :(44:40)

When Not to Use?

1. To store big data because:

- The number of copies == number of nodes
- All data is loaded in RAM too
- Network load of transferring all data to all Nodes

2. Extremely strong consistency

Zookeeper is when you want to store, very big data, then the Zookeeper cannot be stored and all also when extremely strong consistency, applications are required then also, it is it should not be used.

Refer slide time :(44:54)

ZooKeeper Applications: The Fetching Service

- **The Fetching Service:** Crawling is an important part of a search engine, and Yahoo! crawls billions of Web documents. The Fetching Service (FS) is part of the Yahoo! crawler and it is currently in production. Essentially, it has master processes that command page-fetching processes.
- The master provides the fetchers with configuration, and the fetchers write back informing of their status and health. The main advantages of using ZooKeeper for FS are recovering from failures of masters, guaranteeing availability despite failures, and decoupling the clients from the servers, allowing them to direct their request to healthy servers by just reading their status from ZooKeeper.
- Thus, FS uses ZooKeeper mainly to manage configuration metadata, although it also uses Zoo- Keeper to elect masters (leader election).

Now let us see some of the Zookeeper applications and here, we are going to discuss, the fetching service, as the Zookeeper application. So, the fetching service is nothing but here in this service the crawling is an important part, of this fetching service, which is nothing but a search engine and in this particular service the Yahoo crawls billions, of web documents. So, this fetching service is a part, of Yahoo's crawler and it is currently in the production, coessentially it has the master processes that commands, page fetching processes. So, the master provides, the fetchers with the configuration and the features right back, informing of their status and the health, the main advantage of using Zookeeper, for fetching service are recovering from the failure of the masters, guaranteeing availability despite failures and decoupling, the client from the servers allowing them to direct their request to the Hydra servers by just reading their status from the Zookeeper thus fetching service, uses Zookeeper mainly to, to manage the configuration metadata although it uses, the Zookeeper to elect the masters that is done through the leader election.

Refer slide time :(46:54)

ZooKeeper Applications: Katta

- **Katta:** It is a distributed indexer that uses Zoo- Keeper for coordination, and it is an example of a non- Yahoo! application. Katta divides the work of indexing using shards.
- A master server assigns shards to slaves and tracks progress. Slaves can fail, so the master must redistribute load as slaves come and go.
- The master can also fail, so other servers must be ready to take over in case of failure. Katta uses ZooKeeper to track the status of slave servers and the master (**group membership**), and to handle master failover (**leader election**).
- Katta also uses ZooKeeper to track and propagate the assignments of shards to slaves (**configuration management**).

Now let us see another service which is called, 'Katta', so, Katta is the distributed, indexer that uses Zookeeper for the coordination service and it is an example of non Yahoo application. So, Katta divides the work of indexing using shots. So, a master server assigns the shard, to the slaves and tracks the progress. So, slips can fail so, the master must redistribute the loads as the slaves come and go. So, master canal so, fail so, the other servers must be ready to take over in case of the failures. So, Katta uses Zookeeper, to keep the status of slave servers and the master, therefore it is called the group membership, which is done through the, the Zookeeper and to handle the master failure by the leader election. So, the two aspects of leader election of Zookeeper, which is used here in Katta service is called the, 'Group Membership', and the leader election. So, Katta also uses Zookeeper to, to crack and propagate the assignments of shots to the client, which is maintained as the configuration management. So, three different, uses of Zookeeper is used in Katta service, first is called the group membership, where the it has to keep track of slave servers and the master servers, which is called a group membership, the second one is to handle the master failover, is done by the leader election, third is that whenever, the to keep track and propagate the assignments, of charge to the to the slaves this configuration management, is used of as a service from the Zookeeper.

Refer slide time :(48:19)

ZooKeeper Applications: Yahoo! Message Broker

- **Yahoo! Message Broker: (YMB)** is a distributed publish-subscribe system. The system manages thousands of topics that clients can publish messages to and receive messages from. The topics are distributed among a set of servers to provide scalability.
- Each topic is replicated using a primary-backup scheme that ensures messages are replicated to two machines to ensure reliable message delivery. The servers that makeup YMB use a shared-nothing distributed architecture which makes coordination essential for correct operation.
- YMB uses ZooKeeper to manage the distribution of topics (configuration metadata), deal with failures of machines in the system (failure detection and group membership), and control system operation.

Now another Zookeeper application is Yahoo message broker so, that is called, 'YMB'. So, this is an young product. So, Yahoo's message broker, YMB is a distributed publish subscribe system this particular system manages, thousands of topics that clients can publish my Edge's to receive the messages from, the topics are distributed among the set of servers, to provide the scalability. Now each topic is replicated, using the primary backup scheme that ensures messages are replicated to two machines to ensure reliable message delivery, the servers that make up the broker, that is YMB uses, the sheer nothing distributed architecture, which makes the coordination essential for correct operations. So, YMB uses Zookeeper to manage the distribution, of topics that is done in the configuration metadata, also YMB uses Zookeeper to deal with the failures of the machine, in the system that is the failure detectors and the group membership, is managed using the Zookeeper. And then the control system operation and it also controls the system operation. So, therefore we can summarize that the publish subscribe system, which is made by the Yahoo that is called, 'Yahoo Message Broker System', uses the help or uses the Zookeeper, as the coordination service, for to manage the distribution, of the topics using configuration metadata, then it will deal with the failures of the machine, in the system using the failure detect detection and the group membership, of the Zookeeper. And also it performs the control operation.

Refer slide time :(50:22)

ZooKeeper Applications: Yahoo! Message Broker

- The topics directory has a child znode for each topic managed by YMB.
- These topic znodes have child znodes that indicate the primary and backup server for each topic along with the subscribers of that topic.
- The primary and backup server znodes not only allow servers to discover the servers in charge of a topic, but they also manage leader election and server crashes.

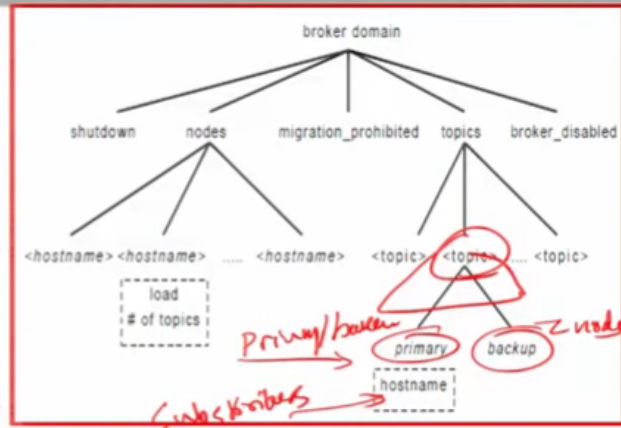


Figure: The layout of Yahoo! Message Broker (YMB) structures in ZooKeeper

So, this particular figure shows the part of Z node layout, for Yahoo message broker system. And so, each broker, domain has the Z node called the, 'Nodes', that has an ephemeral Z nodes for each of the active servers that compose of Yahoo message broker service. So, each YMB server creates an ephemeral node, under the nodes with the load and status information, providing both the group membership and the status information, using the Zookeeper of the yahoo message broker. So, this is shown here as per the layout of a yahoo message broker structure, in the Zookeeper. So, here you can see that these are the different, ephemeral nodes, which maintains the, the topics and using the ephemeral Z nodes in this particular scenario. So, the topics directory, has topics directory as, the child Z node for each topics, which is managed by the YMB these topics that is Z notes have the children's you note that indicate the primary and the backup server, of each of each topic, for his topic along with the subscribers of that particular topic. So, these are the subscribers and these are the primary end and the backup, xenon. So, every topic is maintained as the so, each topic IO the child Z node that indicates the primary and the backup servers for each topic along with the subscriber of that topic. So, the primary and backup servers, that is Z node not only allows the server's to discover the servers in the charge of the topic. But also manages the leader election and server crashes.

Refer slide time :(52:46)

More Details

ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt and Mahadev Konar
Yahoo! Grid
{phunt,mahadev}@yahoo-inc.com

Flavio P. Junqueira and Benjamin Reed
Yahoo! Research
{fpj,breed}@yahoo-inc.com

See: <https://zookeeper.apache.org/>



So, more detail of Zookeeper, we can see through this particular paper, given written by Zookeeper weight free coordination for internet scale systems, by Yahoo peoples.

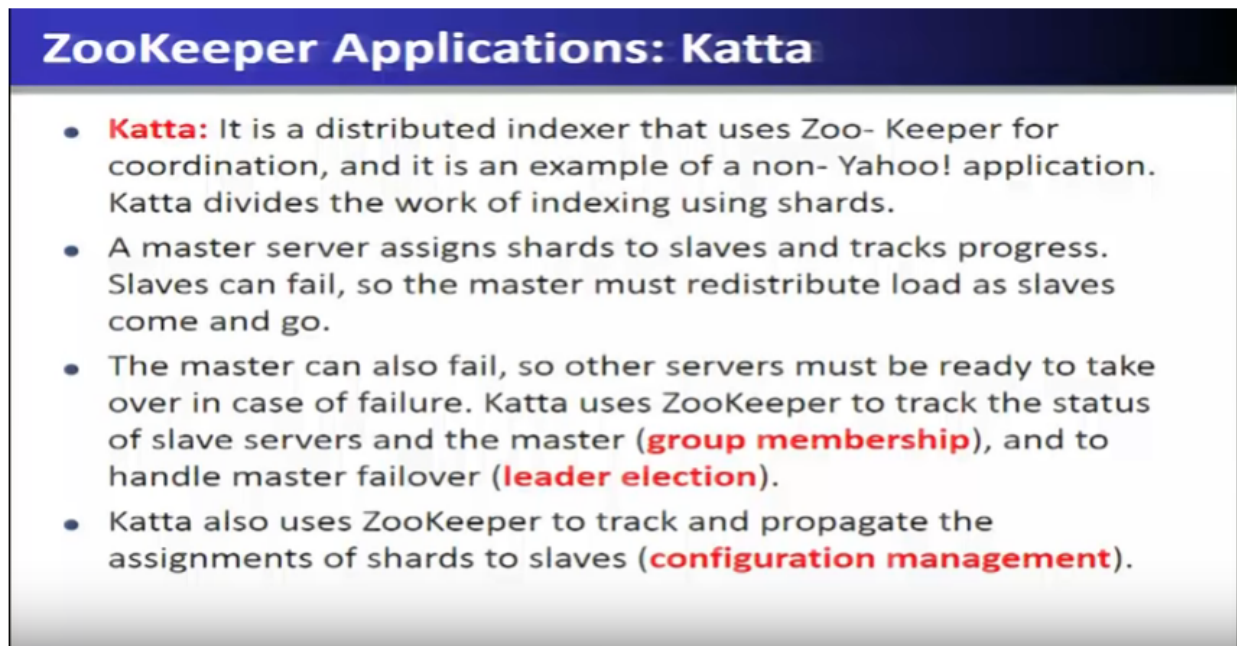
Refer slide time :(53:02)

Conclusion

- **ZooKeeper** takes a wait-free approach to the problem of coordinating processes in distributed systems, by exposing wait-free objects to clients.
- **ZooKeeper** achieves throughput values of hundreds of thousands of operations per second for read-dominant workloads by using fast reads with watches, both of which served by local replicas.
- In this lecture, we have discussed the basic fundamentals, design goals, architecture and applications of ZooKeeper.

Conclusion the Zookeeper takes a weight free approach, to the problem of coordinating processes in the distributed system. By exposing the weight free objects to the client so, Zookeeper achieves throughput values of hundreds and thousands of operation, per second for read dominant workloads, by using fast reads with the watches, both of which are served by the local replicas. So, in this lecture we have discussed the basic fundamentals design goal principles architectures and applications, of the Zookeeper which is a centralized coordination, service and which has these particular properties, such as,

Refer slide time :(53:49)



ZooKeeper Applications: Katta

- **Katta:** It is a distributed indexer that uses Zoo- Keeper for coordination, and it is an example of a non- Yahoo! application. Katta divides the work of indexing using shards.
- A master server assigns shards to slaves and tracks progress. Slaves can fail, so the master must redistribute load as slaves come and go.
- The master can also fail, so other servers must be ready to take over in case of failure. Katta uses ZooKeeper to track the status of slave servers and the master (**group membership**), and to handle master failover (**leader election**).
- Katta also uses ZooKeeper to track and propagate the assignments of shards to slaves (**configuration management**).

The use cases which we have seen about.

Refer slide time :(53:55)

What is Coordination ?

- **Group membership:** Set of datanodes (tasks) belong to same group
- **Leader election:** Electing a leader between primary and backup
- **Dynamic Configuration:** Multiple services are joining, communicating and leaving (Service lookup registry)
- **Status monitoring:** Monitoring various processes and services in a cluster
- **Queuing:** One process is embedding and other is using
- **Barriers:** All the processes showing the barrier and leaving the barrier.
- **Critical sections:** Which process will go to the critical section and when?

We have seen the use cases of the Zookeeper, as the group membership, where the set of nodes where set of data nodes, where belong to the same group, indifferent applications. We have also seen the use cases where the leader election, is done using the Zookeeper. I said as an service, we have also seen the dynamic configuration, aspects of Zookeeper and where in multiple services are joining and some are leaving, we have also seen the status monitoring, aspects our as a service of Zookeeper, where it has monitored various processes and services in the cluster. We have also seen about the queuing, aspect of this coordination service, we have seen barriers we have also seen the critical section. So, in nutshell where our while writing, the distributed application for Big Data computations, these different services are mashed together are collected to the other end is being provided by the Zookeeper. So, writing application, building application, with Zookeeper, is going to be very, very, important.

Refer slide time :(55:19)

What is ZooKeeper ?

- **ZooKeeper is a highly reliable distributed coordination kernel**, which can be used for distributed locking, configuration management, leadership election, work queues,....
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- **Key attributed of such data**
 - Small size
 - Performance sensitive
 - Dynamic
 - Critical
- **In very simple words**, it is a central store of key-value using which distributed systems can coordinate. Since it needs to be able to handle the load, Zookeeper itself runs on many machines.

And we will see in the further slides, how the Zookeeper is used in different applications, for realizing these services. Thank you.