Cloud Computing and Distributed Systems

Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 09
Design of Zookeeper

Design of Zookeeper. Preface, content of this lecture, we will discuss the Design of a Zookeeper, which is a coordination service or distributed application. We will discuss its fundamentals, its design goals, the architecture of apache zookeeper, and various uses its applications.

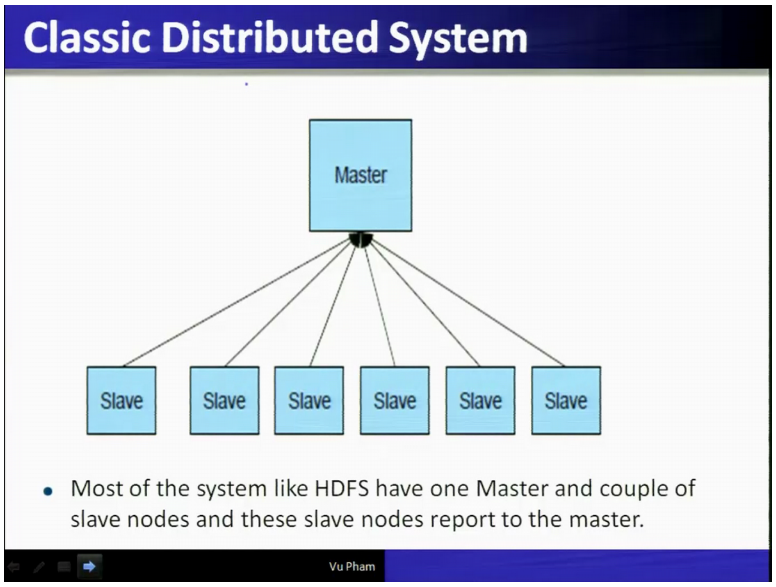
(Refer Slide Time: 01:29)

ZooKeeper, why do we need it?

- Coordination is important

To understand what a zookeeper does, it does a very simple task, which is called coordination, but coordination is important and it is not so simple. In this figure, this shows a situation of a traffic condition in a particular city. This kind of traffic situation requires a coordination, so that the progress at all points of time can be there in the traffic flow. So, the that important, and we will see in our situation in our scenario of distributed systems and a cloud computing systems, how this particular coordination is useful in the applications.

(Refer Slide Time: 01:43)



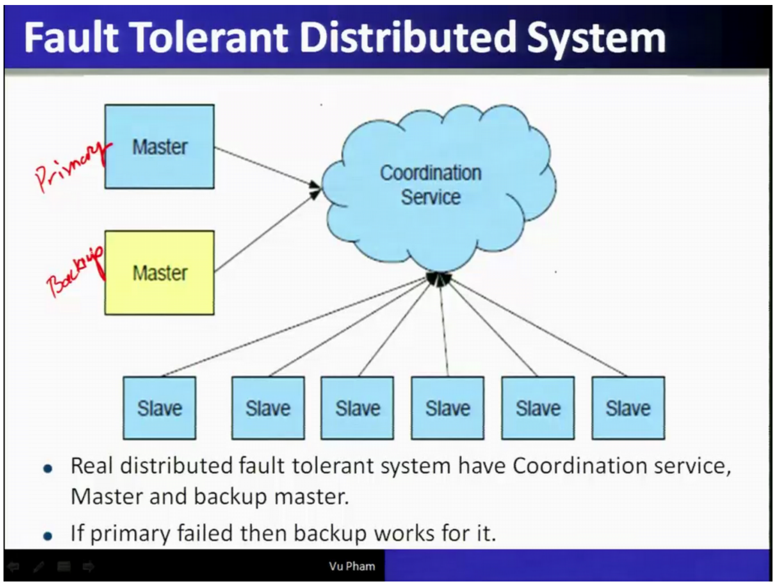
Classic Distributed System

- Most of the system like HDFS have one Master and couple of slave nodes and these slave nodes report to the master.

Vu Pham

This is a simple distributed system scenario, where there is a master and a couple of slaves. These slaves, they send their heart beats or their results to the master, all of them and master will then application issues. This particular kind of scenario, you can you have seen in like HDFS, where there is one master and a couple of slaves, and these slaves reports these slave reports to the master.

(Refer Slide Time: 02:32)



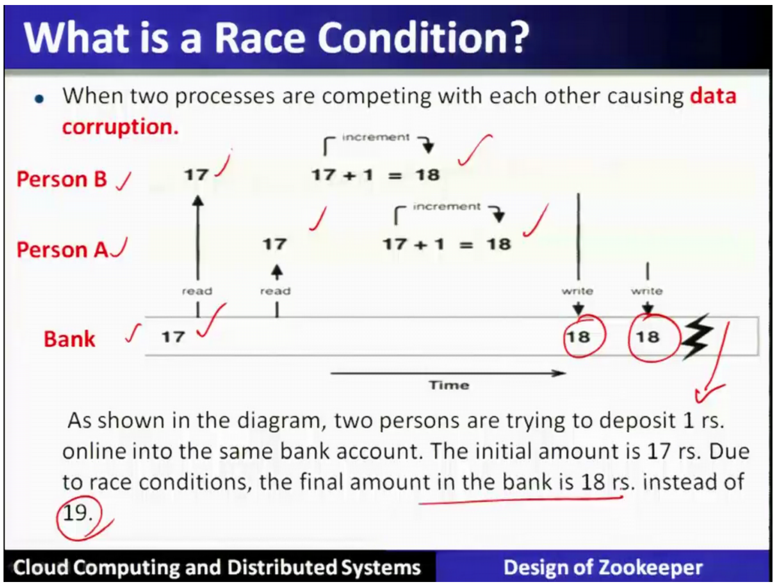
Fault Tolerant Distributed System

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

Vu Pham

Now, if you require a fault tolerant distributed system, then the scenario will be changed from the previous master that is the primary and the backup that means, the masters should be more than one. So if one fails, automatically the other resume its role in there is a requirement of a coordination service, the slaves and the master to ensure the fault tolerant distributed system at all points of time. So, real distributed fault tolerant system have a coordination service, which always which ensures the existence of the master and a few backup masters. So, if a primary master fails, then the backup can he still continue to serve the application without any disruption.

(Refer Slide Time: 03:51)



What is a Race Condition?

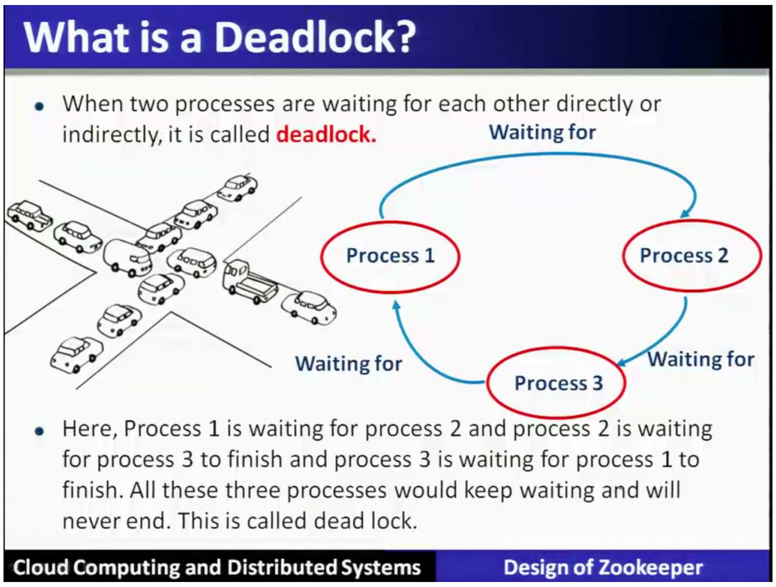
- When two processes are competing with each other causing **data corruption.**

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.

Cloud Computing and Distributed Systems | Design of Zookeeper

Let us see some of the basics, which we are going to use in our discussion. What is a race condition? Here, it is shown that there is a bank account, having an amount 17 in a particular account, and two persons A and B; they want to increment this particular account value. So, they read both at the same point of time, so both of them will get the same value in the account, and then both of them will increment write, so where is the problem. So, if two people are incrementing on 17, the final value should be 19. But, due to the race conditions, the final value is not correct that is it comes out to be 18 that is 1 increment is lost, this is called a race condition.

(Refer Slide Time: 04:55)



What is a Deadlock?

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

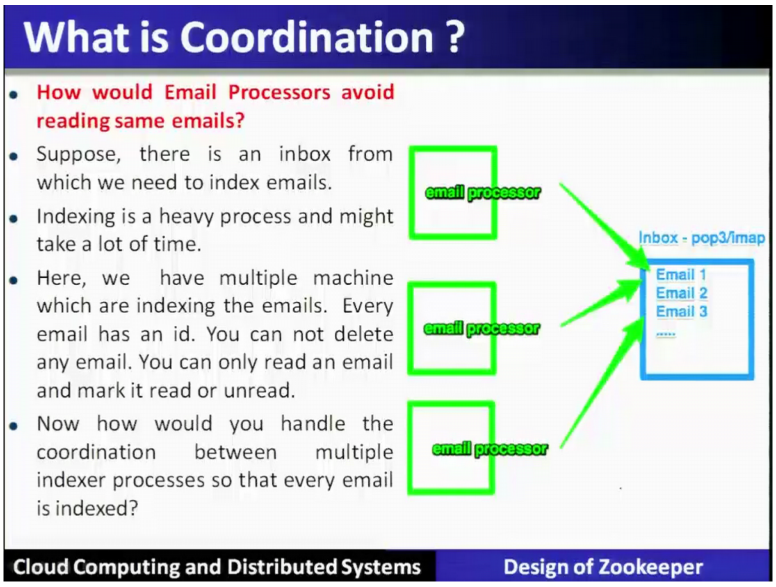
Waiting for
Process 1
Process 2
Waiting for
Waiting for
Process 3

- 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.

Cloud Computing and Distributed Systems — Design of Zookeeper

Another problem, which we have seen in the traffic jams lights. In the system, when two processes are waiting for each other. So, for example, if there are three processes, who are circularly waiting for each other, then they may lead to a dead lock, because they are waiting, they are holding state and waiting for each other. And that will kept on waiting, and this situation is called deadlock. So, race condition and deadlocks are two important things; two important problems, which require to be solved, while the coordination is to be achieved.

(Refer Slide Time: 05:39)



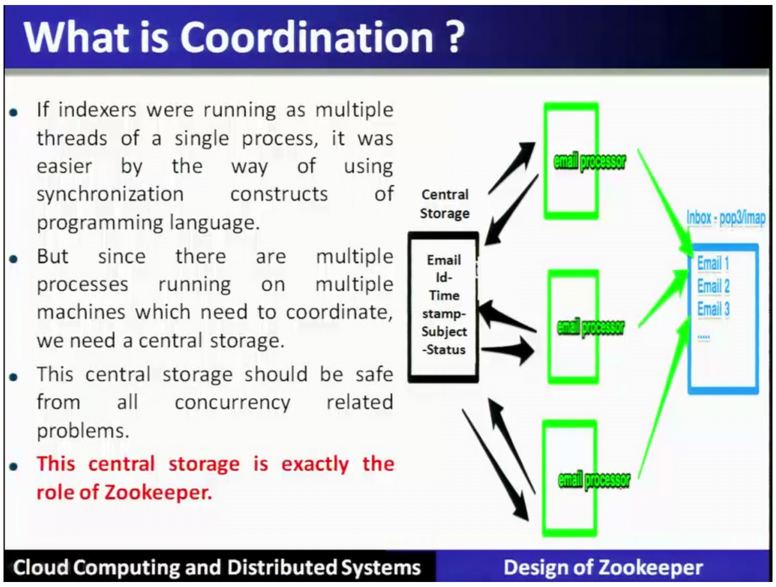
What is Coordination ?

- **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?

Cloud Computing and Distributed Systems — Design of Zookeeper

Now, one application, which requires the coordination is let us say the email processors. Suppose, there is an inbox, from which we need to index mails. Indexing is a heavy process has an id. So, now you would handle the coordination between multiple indexers, process, and the email indexing.

(Refer Slide Time: 06:15)



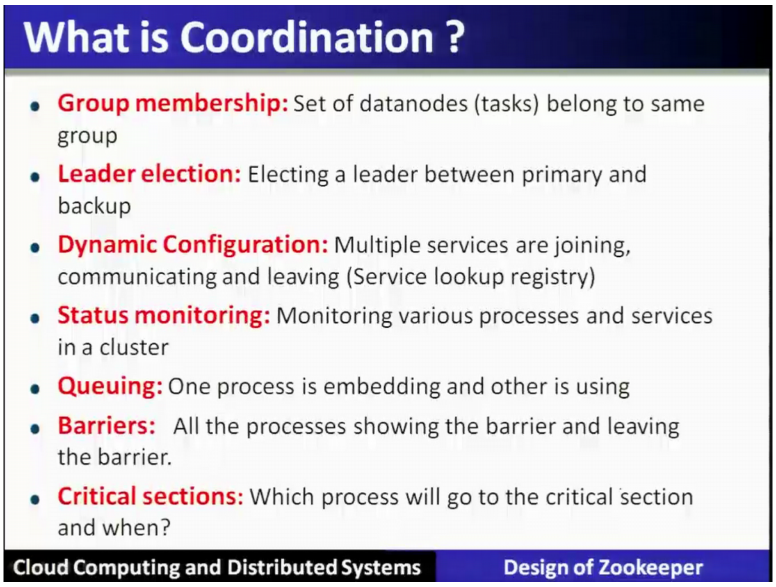
What is Coordination ?

- If indexers were running as multiple threads of a single process, it was easier by the way of using synchronization constructs of programming language.
- But since there are multiple processes running on multiple machines which need to coordinate, we need a central storage.
- This central storage should be safe from all concurrency related problems.
- **This central storage is exactly the role of Zookeeper.**

Cloud Computing and Distributed Systems | Design of Zookeeper

Now, if the indexers are running as multiple threads on a single process, then the most of the operating system and the programming constructs provides the synchronization in the programming language, and that can be solved. But, if multiple processes are running on multiple machine, which needs to be coordinated, we need a central for the coordination, this particular indexer or indexes are running over multiple machines. So, this particular central storage, which is responsible for the coordination, for all concurrent related issues will serve this purpose in the pure distributed system model. This particular central storage can be modelled or can be modelled as the zookeeper application.

(Refer Slide Time: 07:18)



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?

Cloud Computing and Distributed Systems Design of Zookeeper

So, we have seen enough of examples for the coordination. Let us see, what are the other use cases of the coordination. The first one is called group membership, so that means, are executing different tasks so for a particular application, and its corresponding task, they may form a group. These nodes may join, they may leave, therefore a group membership is required, and for that a coordination service is very much needed to ensure the group membership across several applications, which requires different tasks or different data nodes.

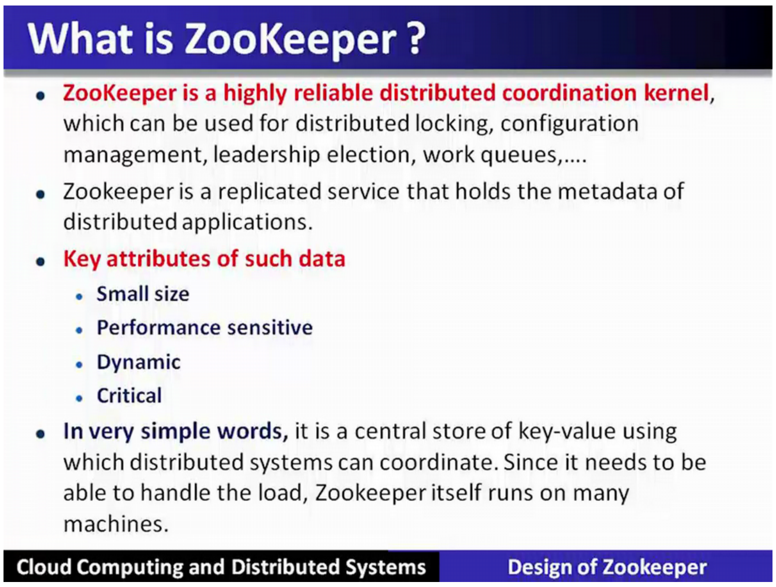
Similarly, another use case is about the leader election. So, we have seen in the fault tolerant distributed system, so there exist more than one master for tolerating any kind of failure that is called single point of failure. So, let us say that primary and backup masters are being provided out of several servers; one of them will be the primary that is called leader, so that particular leader will be elected through a election. So, if there are three different servers maintaining the coordination service, one another use case is about the dynamic configuration.

So, the multiple services if it is running on a particular cloud or a distributed system so for each services, the nodes or the servers which are joining and leaving, therefore a service look up registry is very much required. And that is to run each service a separate configuration that is the number of nodes, which nodes are running those services are required that is called dynamic configuration. And the dynamic configuration is a coordination problem.

Similarly, about the status monitoring, that is the monitoring will of various processes and services in the cluster is also a coordination problem. Queuing is also a coordination problem to some extent that is one process is giving an output, which will be fade as the input to the other piped process. Therefore, forming a queue, this particular way of piping, several processes for the application or a pipeline for a application is nothing but a coordination problem.

Barriers is also an application of a coordination problem, so that means, all the process will join at a barrier, and then they may leave from the barrier after all the processes joins it, that is also a coordination problem at the barrier end. Similarly, the critical sections, that means, to coordinate that across since the particular critical section, it has to be only one process at a time can be executing into the critical section; it is also a coordination problem.

(Refer Slide Time: 10:49)



What is ZooKeeper ?

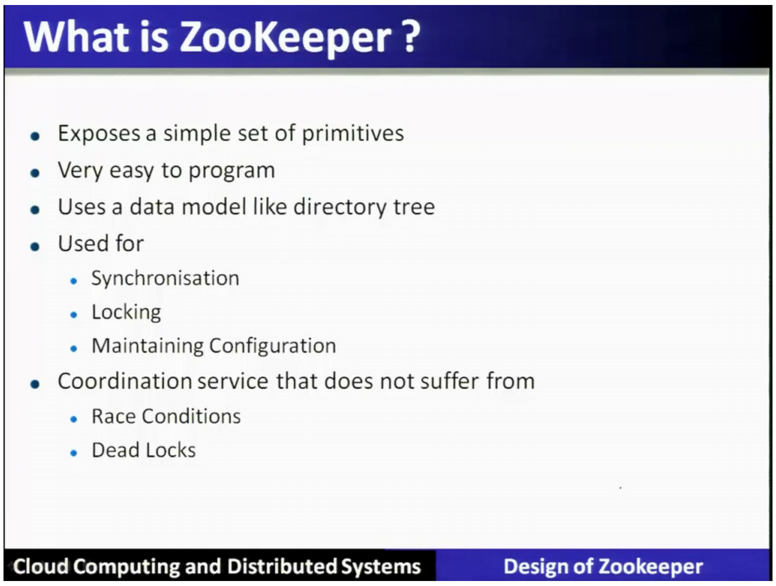
- **ZooKeeper is a highly reliable distributed coordination kernel**, which can be used for distributed locking, configuration management, leadership election, work queues,....
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- **Key attributes of such data**
  - 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.

Cloud Computing and Distributed Systems | Design of Zookeeper

So, how this, so, this is done through the apache zookeeper. Let us see how it achieves this kind of coordination, which is applied over wide variety of applications. So, zookeeper is a highly reliable distributed coordination kernel. Kernel in the sense, it is not providing all these services, such as locking, configuration management, leader election that we have already seen. But this is the core, which can be in sense; it is a highly reliable distributed coordination kernel, so it provides lot of APIs. And using this API, this application can be build or can use the coordination service to build the applications.

Zookeeper is a replicated service that holds the metadata of the distributed applications. So, when we say metadata, that means, it is distributed application. It manages, hence the data is small, which zookeeper manages. Several key attribute of this matter is that it is of small size, and it is of performance sensitive, it is dynamic, it is critical. In very simple words, it is the central store of key value store using the distributed system to coordinate for different distributed applications. So, it need to be able to manage to handle the load, the zookeeper itself runs on many machines.

(Refer Slide Time: 12:34)



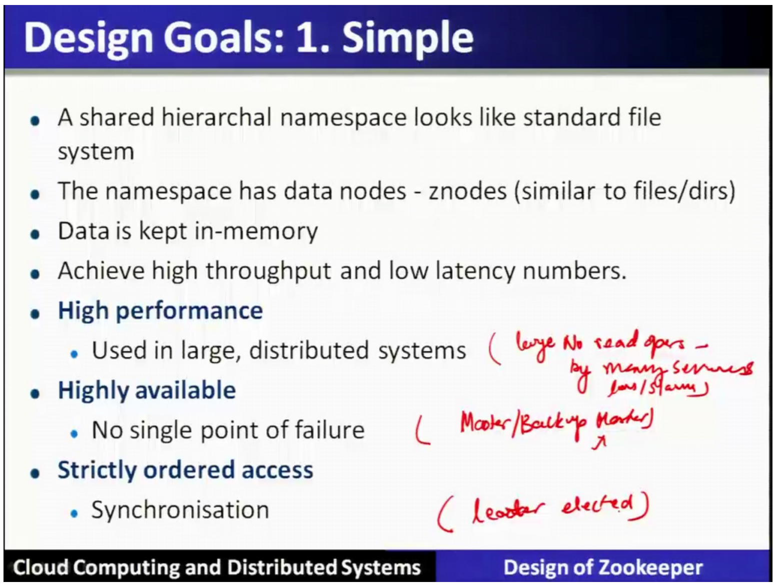
What is ZooKeeper ?

- 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

Cloud Computing and Distributed Systems Design of Zookeeper

So, zookeeper will provide simple primitives, which are easy to program for the clients to use for their applications. It uses a very simple data model like a directory tree that we have seen in a many file system or a directory structure of a file system. Zookeeper is basically used for synchronization in a distributed application. It is used for locking that is called distributed locking the consistency, and also maintain the configuration that is dynamic configuration. Such a coordination service, which is provided by the zookeeper ensures that does not suffer from race conditions and deadlocks.

(Refer Slide Time: 13:24)



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 (large No read opers - by many servers ldrs/slaves)
- **Highly available**
  - No single point of failure (Master/Backup Master)
- **Strictly ordered access**
  - Synchronisation (leader elected)

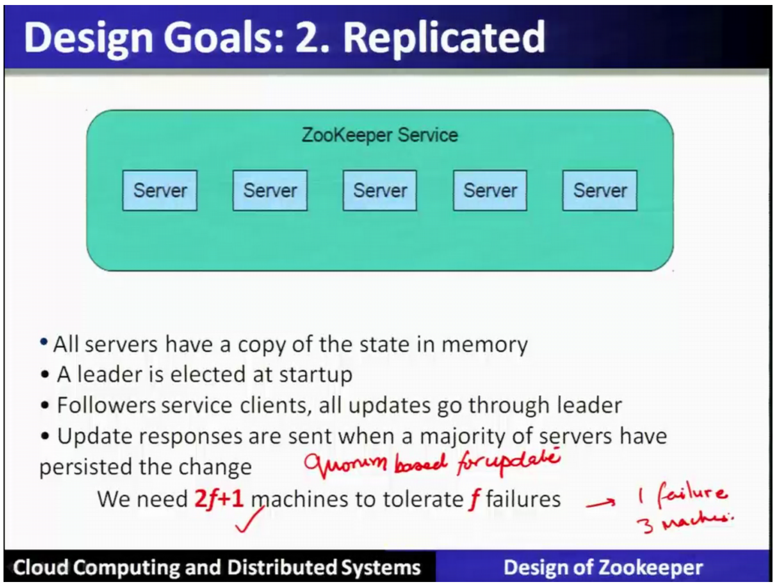
Cloud Computing and Distributed Systems | Design of Zookeeper

Therefore, the design goal is to make to ensure the simplicity that is in the form of the hierarchal namespace, which looks very similar to the standard file system hierarchy. However, the namespace has the data nodes, which are called znodes, similar to the files or directories. These data is kept in memory. And therefore, it achieves the high throughput and low latency numbers.

Since, this particular aspect that is the namespace is a metadata, therefore it can be kept in the memory because of its small size. This will give the high; that is it is used in the large scale distributed system; it is also to be highly available. Why, because there is no single point of failures, because it is replicated, and also it is strictly ordered; why, because it ensures the synchronization. So, in the zookeeper, all three things that is high performance, highly available, and strictly ordered access is supported.

So, high performance is basically used in a large scale, that means, it is supporting large number of reads questions, which are served by many servers that means, leaders and its slaves; therefore it achieves the high performance for its operations. It is highly available that is there is no single point of failure that means, so because there is a master and there is a backup master also, which ensures the failure tolerance. It is strictly ordered access that is synchronisation is guaranteed, since there is a leader elected at all points of time. Therefore, this ensures the strictly ordered access and also ensures the sync rate environment.

(Refer Slide Time: 16:13)



Design Goals: 2. Replicated

ZooKeeper Service
Server
Server
Server
Server
Server

- All servers have a copy of the state in memory
- A leader is elected at startup
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change Quorum based for update

We need **2f+1** machines to tolerate *f* failures ✓ → 1 failure 3 machines.

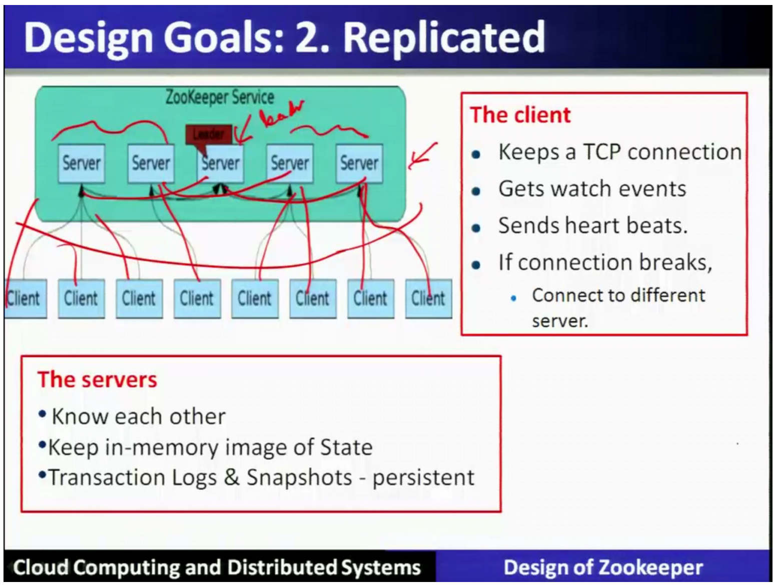
Cloud Computing and Distributed Systems

Design of Zookeeper

The 2nd design goal is about the replication. So, all the servers have a copy of the state in the memory. So, a leader is elected at the startup. Followers are those servers, who are not elected as a leader. So, they also have the updated state at all point of time, and it is the responsibility of the leader to ensure that all the servers. So, the clients for any update, it has to follow through the leader.

So, update responses are sent, when the majority of the servers have persisted the change. So, majority means, it is following the quorum based approach for updates. Therefore, for fault tolerance, we require 2f plus 1 machine to tolerate f failures. So, to tolerate one failure, so it required how many machines, 3 machines. So, we will see that minimum three machines are required to tolerate one failure. If two are failing in three, then basically it is not the majority, hence this will violate this condition.

(Refer Slide Time: 17:57)



Design Goals: 2. Replicated

ZooKeeper Service
Leader
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

The client

- Keeps a TCP connection
- Gets watch events
- Sends heart beats.
- If connection breaks,
  - Connect to different server.

The servers

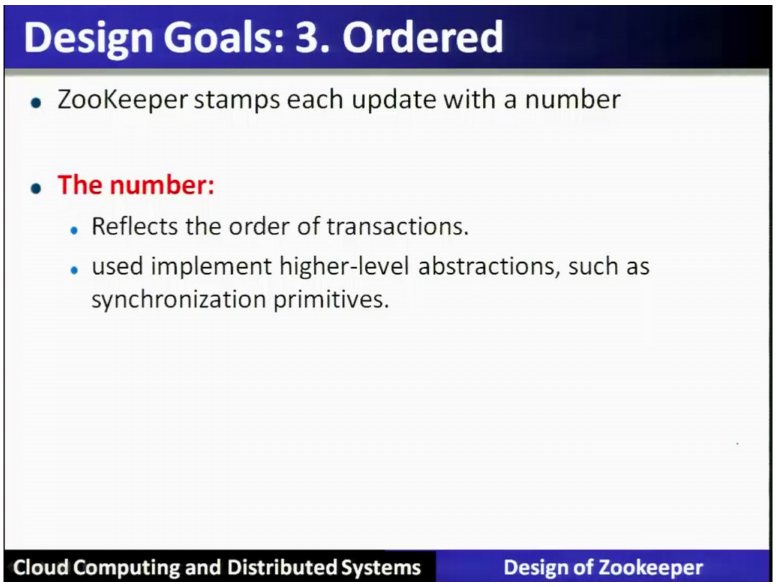
- Know each other
- Keep in-memory image of State
- Transaction Logs & Snapshots - persistent

Cloud Computing and Distributed Systems | Design of Zookeeper

So, the replicated structure, we have seen we will divide into two different classes; one is called client, we will keep the TCP connections and also gets a watch event, it will always through that it will be recognize that it is having a connection to the zookeeper. If the connection breaks, then basically it is no longer the client. Similarly, the severs will know each other; and server will keep in memory image of the state; it will also use the transaction logs and takes the snapshot in a persistent storage.

Here in this particular figure, we have seen all the servers, which are knowing each other. And through the election, one of them will be the leader, and others are the followers. Clients can connect to any of this particular leader or the follower, and keep on sending the heartbeat, so that they will know that they are basically the clients of these services.

(Refer Slide Time: 19:31)



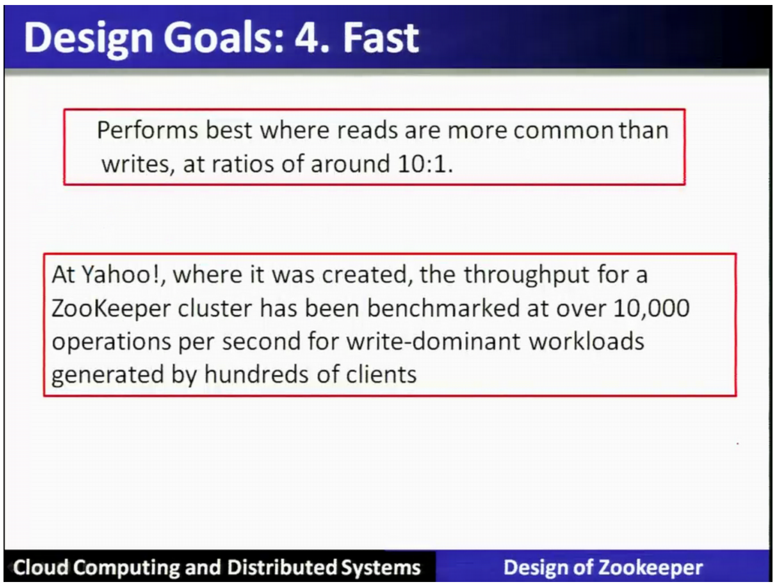
Design Goals: 3. Ordered

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

Cloud Computing and Distributed Systems | Design of Zookeeper

The 3rd design goal is the ordering. So, zookeeper stamped each update with a number. So, the number reflects the order of the transaction; and used to implement the higher level abstraction, such as synchronization primitives.

(Refer Slide Time: 19:55)



Design Goals: 4. Fast

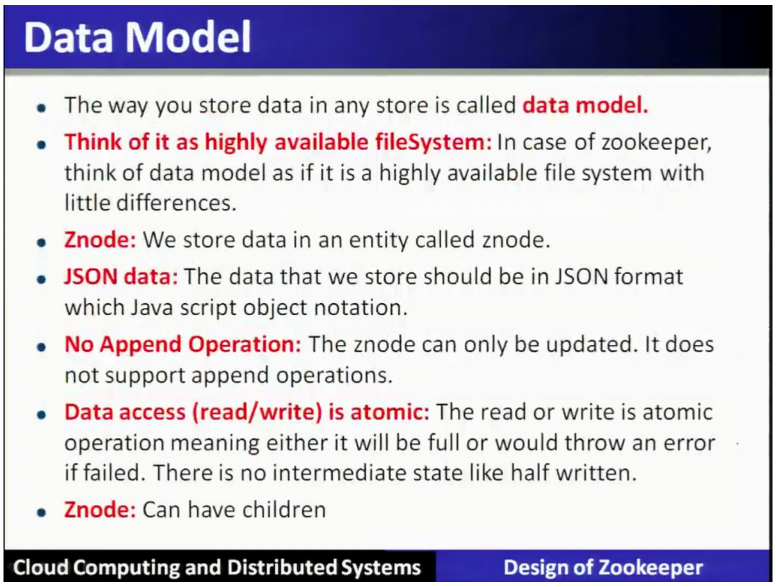
Performs best where reads are more common than writes, at ratios of around 10:1.

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

Cloud Computing and Distributed Systems | Design of Zookeeper

4th goal is about the speed. So, this achieves, when it performs the best. Performance, when the reads are more than the writes, at the ratio let us say 10 is to 1. Then in that scenario, it is benchmark that it basically has achieved 10,000 operations per second, generated by hundreds of clients.

(Refer Slide Time: 20:27)



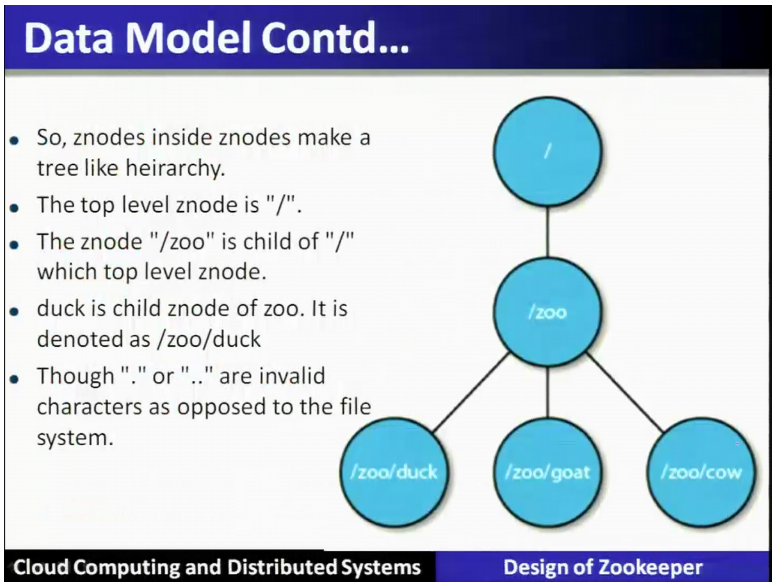
Data Model

- 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

Cloud Computing and Distributed Systems Design of Zookeeper

Let us see the data model used in the zookeeper. The way you store the data in any data store is called a data model. Now, here in the it uses a file system like interface, so zookeeper will think of data model, as if it is highly available file system where the nodes are called znodes, which will store the data in an entity. The data is in JSON format. And it does not supports the append operation, it can only be updated. Now, data access is done through the read and write operation, which is to be done in an atomic manner; either it will be full or it will give (Refer Time: 21:29) Znodes can have the childrens.

(Refer Slide Time: 21:32)



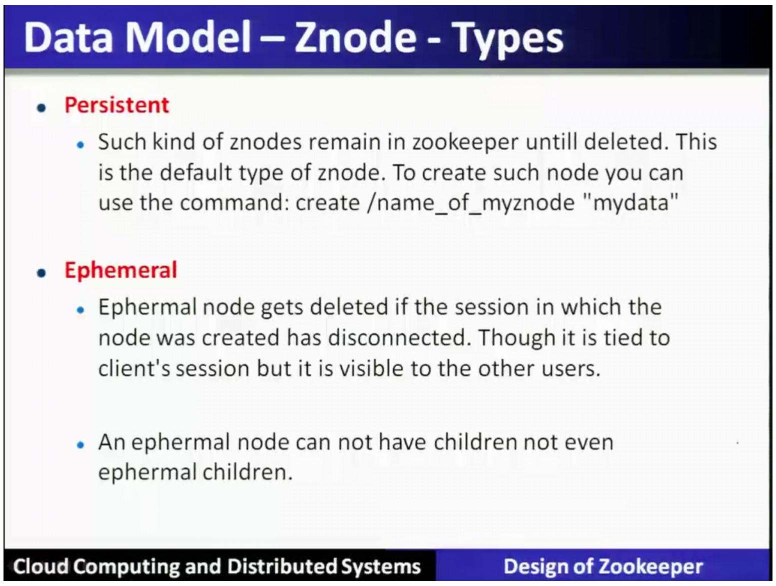
Data Model Contd...

- So, znodes inside znodes make a tree like heirarchy.
- 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.

Cloud Computing and Distributed Systems Design of Zookeeper

So, this is a typical data model, which is called here the nodes are called z. So, this particular hierarchy of znodes will stop, and then followed by a znode slash zoo; goat and cow, they are the child of slash zoo znode.

(Refer Slide Time: 22:01)



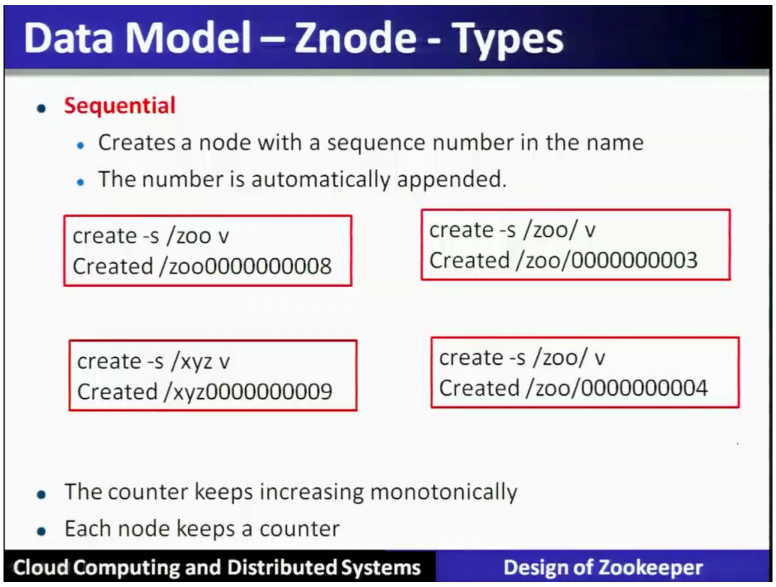
Data Model – Znode - Types

- **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**
  - Ephermal 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 ephermal node can not have children not even ephermal children.

Cloud Computing and Distributed Systems | Design of Zookeeper

There are three different type persistent, ephemeral, and sequential (Refer Time: 22:08) until deleted. And this is a default type of znode. Ephemeral node get to the client users also. Ephemeral nodes cannot have in their names.

(Refer Slide Time: 22:20)



Data Model – Znode - Types

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

```
create -s /zoo v
Created /zoo0000000008
```

```
create -s /zoo/ v
Created /zoo/0000000003
```

```
create -s /xyz v
Created /xyz0000000009
```

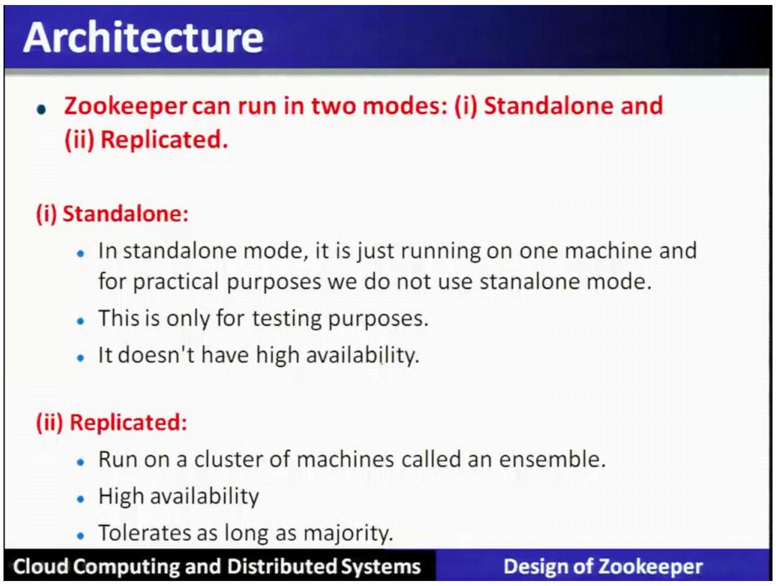
```
create -s /zoo/ v
Created /zoo/0000000004
```

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

Cloud Computing and Distributed Systems | Design of Zookeeper

These numbers are automatically appended. The counters keep increasing monotonically, and each node keeps a counter. Example here, first time when a node v is created znode v is created; its number is 3. Then second time, they are sequential.

(Refer Slide Time: 22:47)



Architecture

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

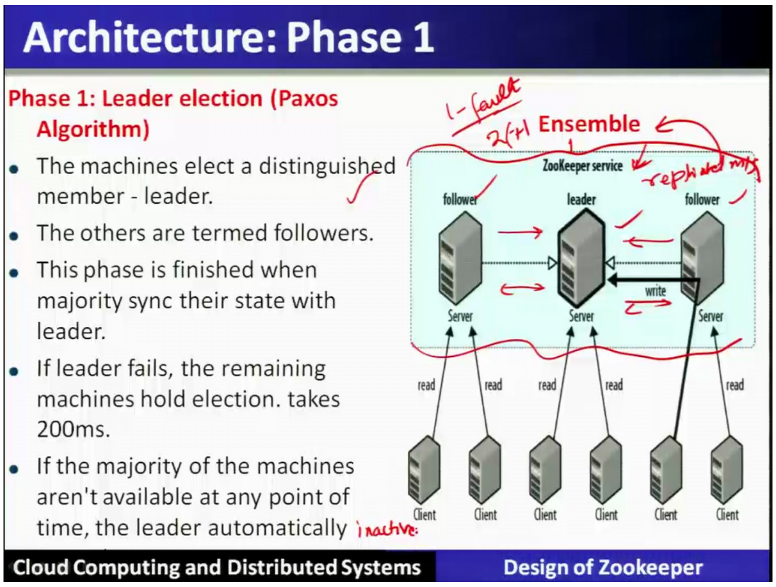
(i) Standalone:
- In standalone mode, it is just running on one machine and for practical purposes we do not use stanalone mode.
- This is only for testing purposes.
- It doesn't have high availability.

(ii) Replicated:
- Run on a cluster of machines called an ensemble.
- High availability
- Tolerates as long as majority.

Cloud Computing and Distributed Systems | Design of Zookeeper

Let us see the architecture of zookeeper. So, zookeeper can run in two different modes for the testing purpose in into the lab. Therefore, it does not supports many important features like high availability, and so on and also it is not used in the in the practice. So, the other mode which is called a replicated mode, which will be launched it runs on a cluster of machines, which is called ensemble. It ensures high availability; and it tolerates as long as the majority of machine are responding.

(Refer Slide Time: 23:37)



Architecture: Phase 1

Phase 1: Leader election (Paxos Algorithm)

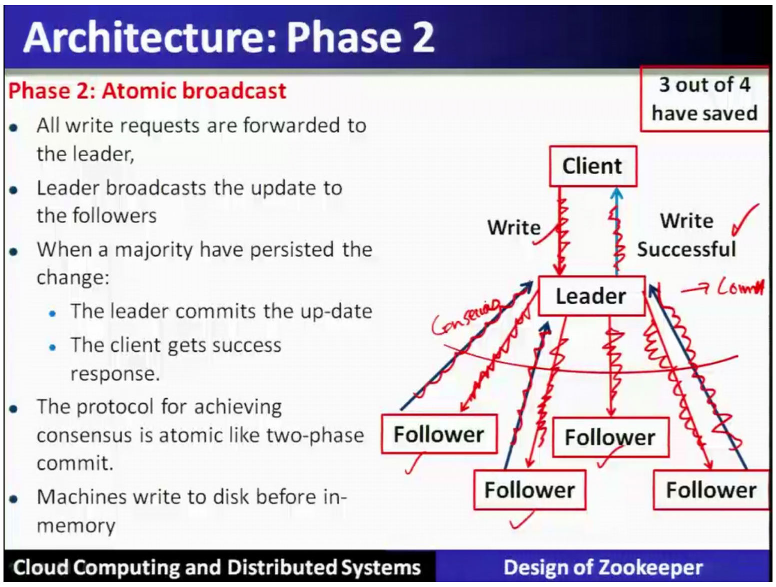
- The machines elect a distinguished member - leader.
- The others are termed followers.
- This phase is finished when majority sync their state with leader.
- If leader fails, the remaining machines hold election. takes 200ms.
- If the majority of the machines aren't available at any point of time,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

Cloud Computing and Distributed Systems | Design of Zookeeper

See that this particular architecture is supported by two different phases; the phase one is called leader election, and it is done through the Paxos algorithm. Here, we can see that the set of machines in the zookeeper, we call it as ensemble. And here, there are number of clients are there, who are accessing the zookeeper service. So, among that servers or the machines, more than one machines are there, so among them, they will that is called ensemble. If we go to the previous slide, (Refer Time: 24:25) replicated service will run on a cluster of machines, which is called ensemble. So, this zookeeper service will run on which is called ensemble.

And among them, one of a leader, this leader election is done through the exchange of ping messages based on their id's, their heartbeats (Refer Time: 24:59) elect one of them to be the leader. And all other machines will become for the leader and followers. So, as you know, that three machines that means, it is able to tolerate 1 machine failure that is here, it is following 2f plus 1. This phase that is the leader election phase is finished, when the majority sync their state with the leader, that means, they will sync their states with the leader. And if the leader fails, the remaining machine can hold the election within 200 milliseconds. So, if the majority of the machines are not available at any point of time, the leader automatically leader automatically will cease to work.

(Refer Slide Time: 26:07)



Architecture: Phase 2

Phase 2: Atomic broadcast

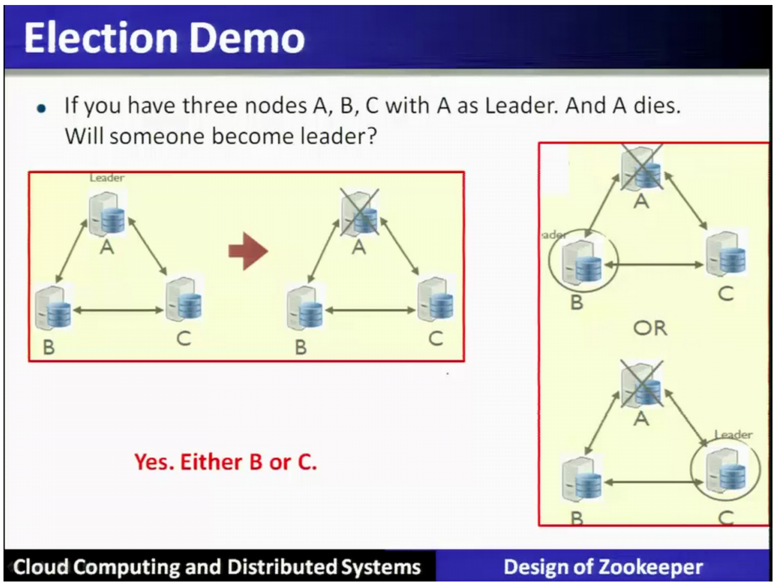
- All write requests are forwarded to the leader,
- Leader broadcasts the update to the followers
- When a majority have persisted the change:
  - The leader commits the up-date
  - The client gets success response.
- The protocol for achieving consensus is atomic like two-phase commit.
- Machines write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

3 out of 4 have saved
Client
Write
Write Successful
Leader
Consensus
Commit
Follower
Follower
Follower
Follower

Cloud Computing and Distributed Systems Design of Zookeeper

Write requests are forwarded to the leader, followers in this manner. Now, when the majority have persisted the change, let us say out of 4, 3 is the majority. When they have persisted the change, and they will inform it to the leader about it, then the leader will do the commit of this particular write operation. And for this the client will be informed about the success. The protocol for achieving the consensus is atomic like two-phase like two-phase commit.

(Refer Slide Time: 27:14)



Election Demo

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

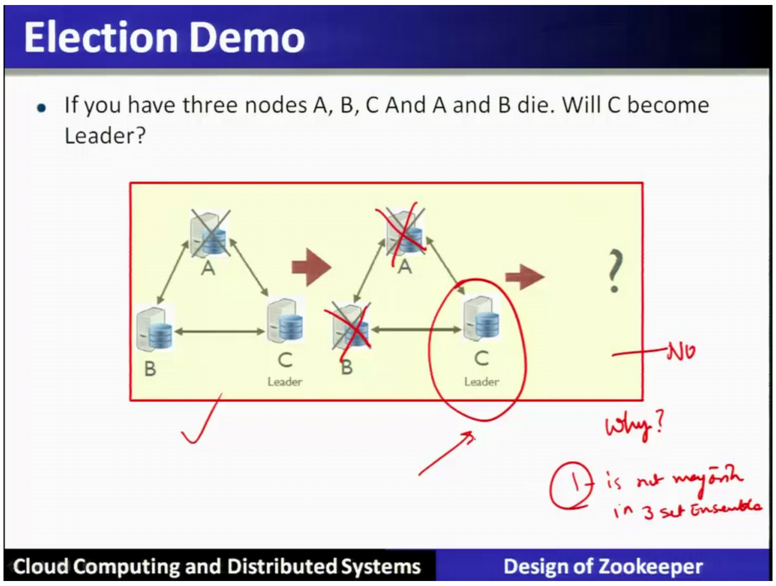
Leader
A
B
C
A
B
C
A
Leader
B
C
OR
A
Leader
B
C

Yes. Either B or C.

Cloud Computing and Distributed Systems Design of Zookeeper

Let us see, so if you have three different nodes, when A dies, then among B and C, because out of three, two are the majority. So, if A fails, either B or C can become (Refer Time: 27:39).

(Refer Slide Time: 27:40)



Election Demo

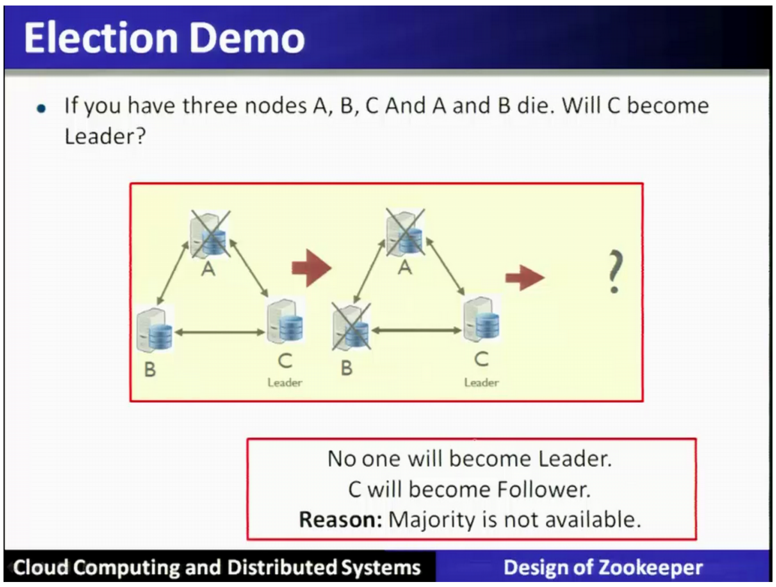
- If you have three nodes A, B, C And A and B die. Will C become Leader?

A
B
C
Leader
A
B
C
Leader
?
No
Why?
1 is not majority in 3 set Ensemble

Cloud Computing and Distributed Systems | Design of Zookeeper

And if two are failed, A and B let us say they are failed, remaining only one. What will happen, whether c will become a leader or not? No. Why? Because, because 1 is not the majority in 3 set ensemble, so that means, to work in this particular scenario, a majority should always be present, otherwise it will not have a leader in its operations.

(Refer Slide Time: 28:30)



Election Demo

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

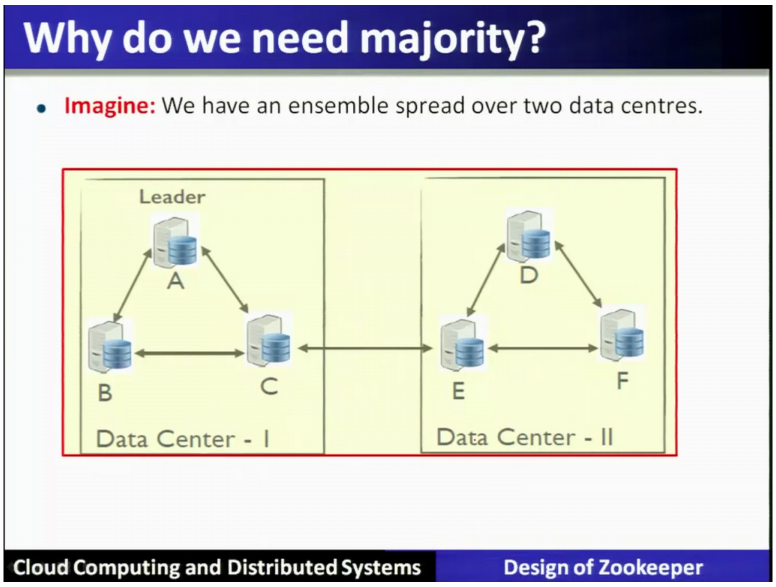
A
B
C
Leader
A
B
C
Leader
?

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

Cloud Computing and Distributed Systems | Design of Zookeeper

So, this will become a follower, there will not be any leader present in that case.

(Refer Slide Time: 28:35)



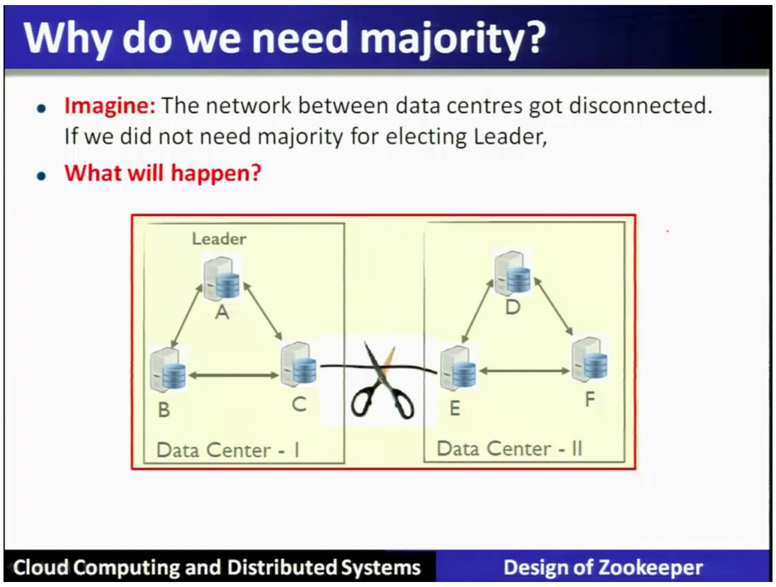
Why do we need majority?

- **Imagine:** We have an ensemble spread over two data centres.

Cloud Computing and Distributed Systems | Design of Zookeeper

Let us say if you do not agree to it regarding the majority, let us see what problem will happen. Let us say that you have an ensemble two different data centers; data center one has three; another data center also has three. So, there are total how many, six are there.

(Refer Slide Time: 29:05)



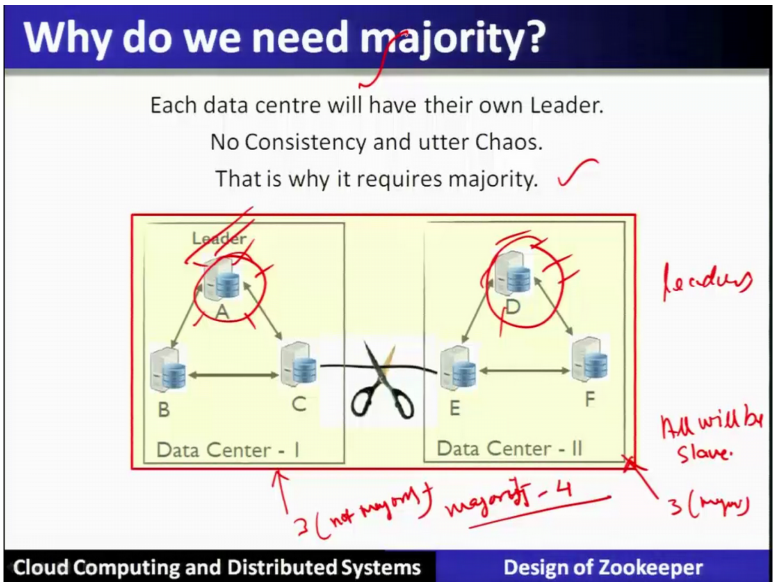
Why do we need majority?

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

Cloud Computing and Distributed Systems | Design of Zookeeper

Now, if this particular link is broken, both these data will elect the leader.

(Refer Slide Time: 29:14)



Why do we need majority?

Each data centre will have their own Leader.

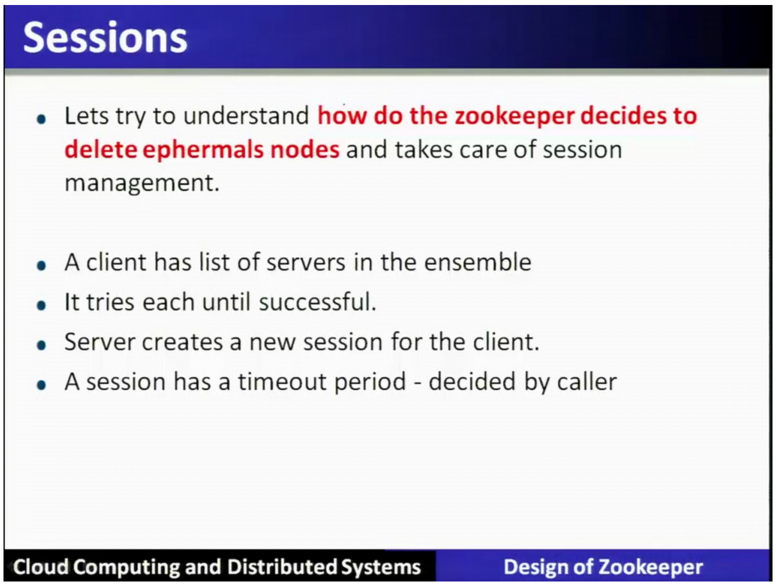
No Consistency and utter Chaos.

That is why it requires majority. ✓

Cloud Computing and Distributed Systems | Design of Zookeeper

They will elect their own leader in this particular scenario, when there is no link. This will non-ensure any consistency, and it will be utter chaos situation. That is why, a majorities. 4 is required to be in the majority number, (Refer Time: 29: 43) they basically are disconnected. Then they will be only 3 that means, not in the majority. Therefore, this particular leader cannot be there be that means, both of them should not have any leader the slave. So, the majority is very much important to ensure coordination at all points of time.

(Refer Slide Time: 30:19)



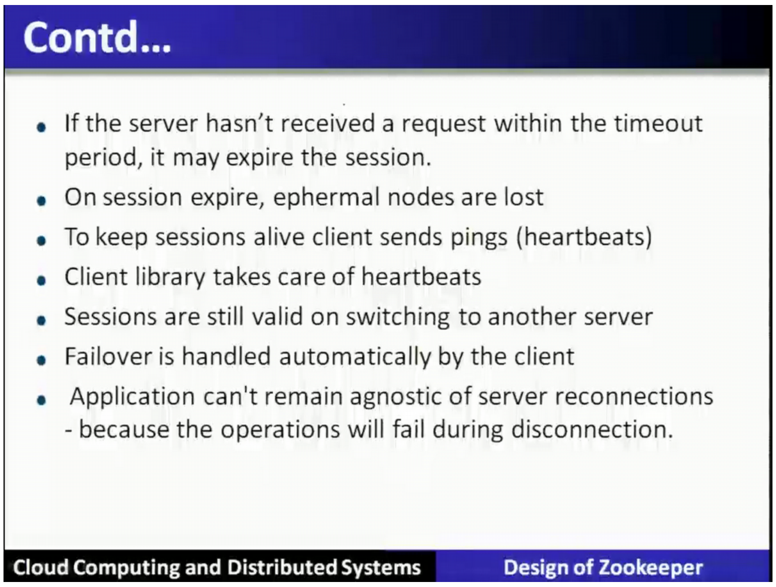
Sessions

- Lets try to understand **how do the zookeeper decides to delete ephermals 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

Cloud Computing and Distributed Systems | Design of Zookeeper

Now, then there is a concept of a session, let us try to understand how the zookeeper delete the ephermal nodes and take care of the session management. So, a client has a list of servers in the ensemble, till it is successful. So, the server create a new session for a client; a session has a time period that is decided by the caller.

(Refer Slide Time: 30:51)



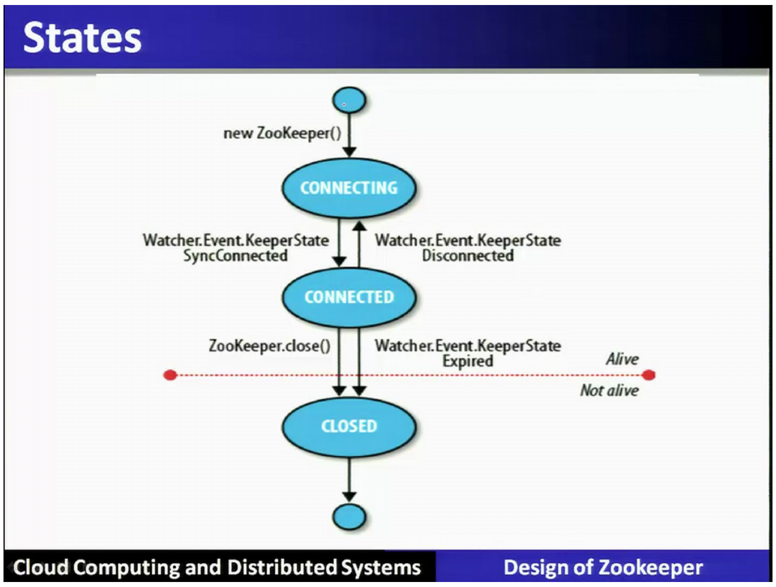
Contd...

- If the server hasn't received a request within the timeout period, it may expire the session.
- On session expire, ephermal 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.

Cloud Computing and Distributed Systems Design of Zookeeper

Server has not received the request within a particular timeout, it may expire the session. On session expiry, ephermal nodes are lost. The client sends the server; and a failover is main agnostic of the server reconnections, because of the operations will fail during the disconnections.

(Refer Slide Time: 31:08)



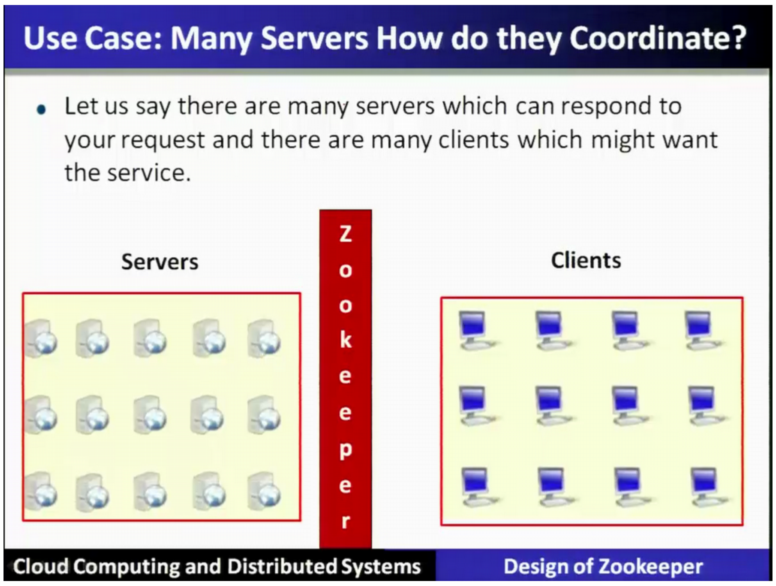
States

new ZooKeeper()
CONNECTING
Watcher.Event.KeeperState SyncConnected
Watcher.Event.KeeperState Disconnected
CONNECTED
ZooKeeper.close()
Watcher.Event.KeeperState Expired
Alive
Not alive
CLOSED

Cloud Computing and Distributed Systems Design of Zookeeper

So, these are the transitions of these states, which the watcher will keep track and keep on informing about the coordination.

(Refer Slide Time: 31:29)



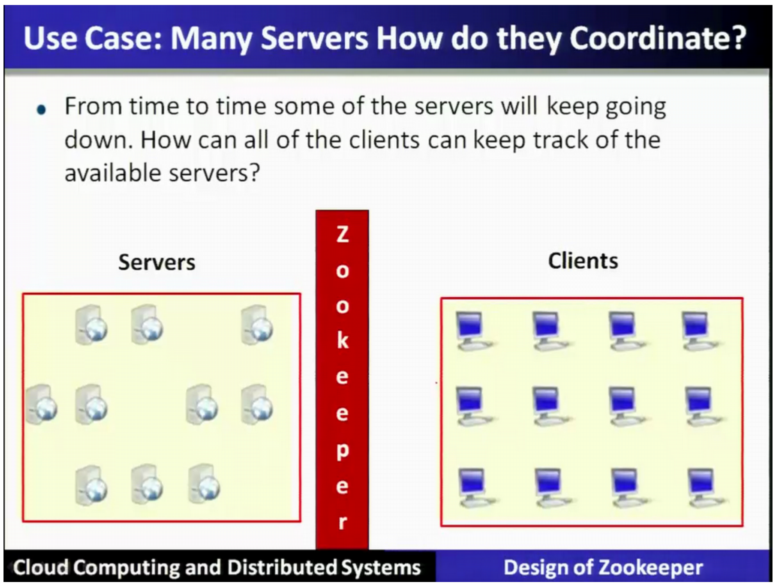
Use Case: Many Servers How do they Coordinate?

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

Cloud Computing and Distributed Systems Design of Zookeeper

Now, here we consider we have seen or we are looking in the slide that there are many servers available; similarly on the other side, we have many clients. Not all servers are responding to the client for a particular service. So, for a particular service, the client will contact to the zookeeper, and zookeeper will find out who are the servers. It will try to find out the ephemeral servers, who are active in by the zookeeper. So, zookeeper comes in between that is on one side, lot of servers are there; on the other side, lot of clients are there, and zookeeper is in between to coordinate between them for an applications.

(Refer Slide Time: 32:20)



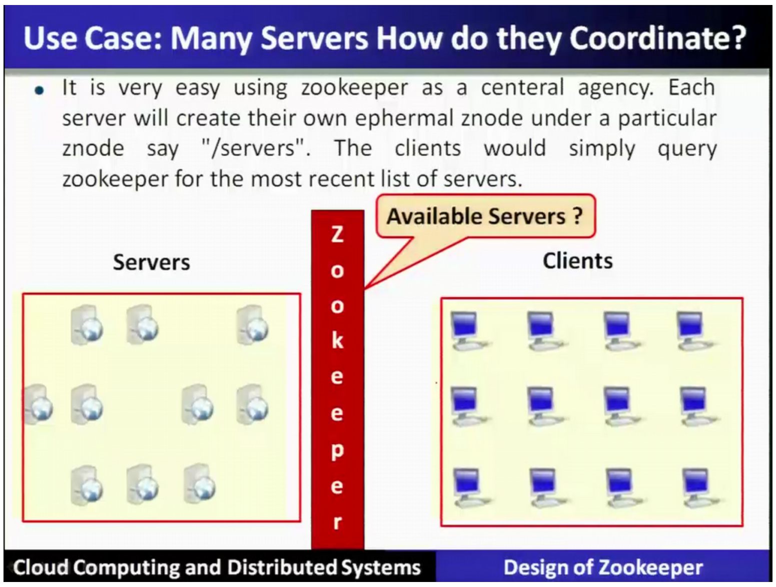
Use Case: Many Servers How do they Coordinate?

- From time to time some of the servers will keep going down. How can all of the clients can keep track of the available servers?

Cloud Computing and Distributed Systems   Design of Zookeeper

So, (Refer Time: 32:22) require, because the server are keep coming up and going down at different point of and the client, who are the available server (Refer Time: 32:33) who will basically do this coordination. To do this coordination, clients are provided with the very simple interface that is a file system kind of interface. So, client will think that it is writing on a particular file node or a znode. So, the operations to access large number of servers becomes quite simple.

(Refer Slide Time: 32:58)



Use Case: Many Servers How do they Coordinate?

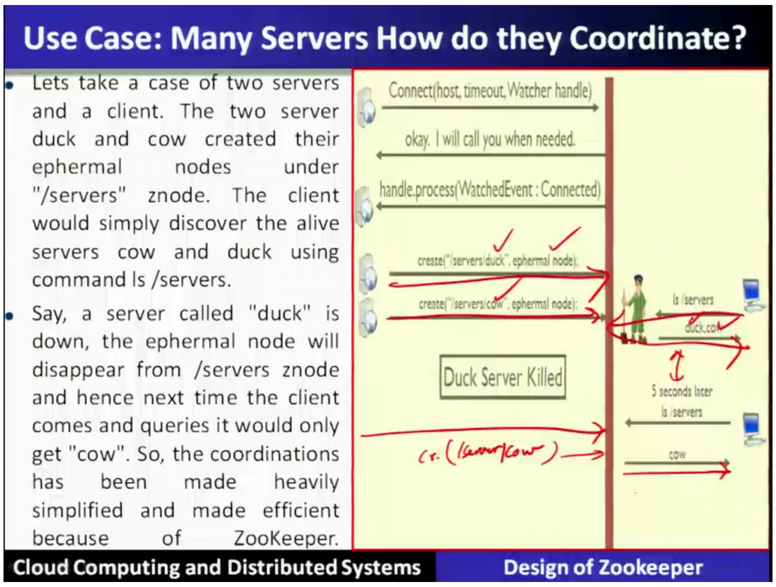
- It is very easy using zookeeper as a centeral agency. Each server will create their own ephermal znode under a particular znode say "/servers". The clients would simply query zookeeper for the most recent list of servers.

Available Servers ?
Servers
Zookeeper
Clients

Cloud Computing and Distributed Systems | Design of Zookeeper

So, zookeeper is becoming a central agency here. So, each server will create its own ephemeral node that is znode under a particular directory that is slash servers. So, the client would simply query for the most recent list of servers in this manner.

(Refer Slide Time: 33:21)



Use Case: Many Servers How do they Coordinate?

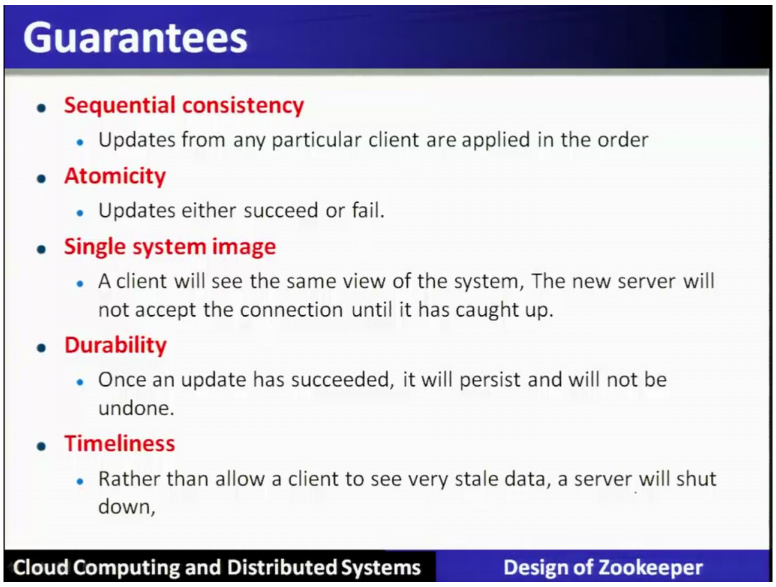
- Lets take a case of two servers and a client. The two server duck and cow created their ephermal nodes under "/servers" znode. The client would simply discover the alive servers cow and duck using command ls /servers.
- Say, a server called "duck" is down, the ephermal node will disappear from /servers znode and hence next time the client comes and queries it would only get "cow". So, the coordinations has been made heavily simplified and made efficient because of ZooKeeper.

Connect(host, timeout, Watcher handle)
okay. I will call you when needed.
handle.process(WatchedEvent : Connected)
create("/servers/duck", ephermal node);
create("/servers/cow", ephermal node);
ls /servers
duck,cow
Duck Server Killed
5 seconds later
ls /servers
cow
cr. (/server/cow)

Cloud Computing and Distributed Systems | Design of Zookeeper

So, client will query about the severs. So, all the ephemeral servers who are basically active at that point of time, will get respond by the zookeeper in the form of a list. For example now, so this list now after some elapse after some point of particular servers are not active or let us say that (Refer Time: 33:56) the server that is called cow remains visible at that point of time as an ephemeral node only the cow. So, the zookeeper will coordinate between the set of servers and the client to ensure the availability of the servers. And this is the coordination activity of zookeeper without more detail botheration at the client sent.

(Refer Slide Time: 34:32)



Guarantees

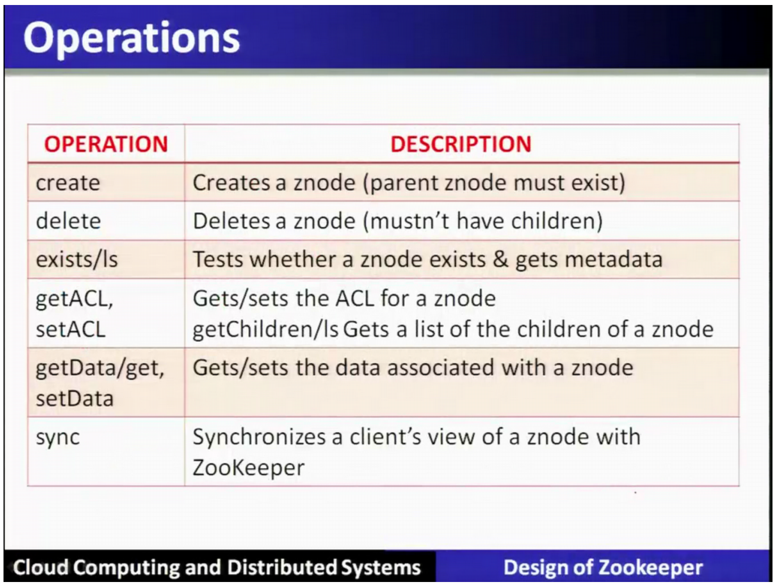
- Sequential consistency
  - Updates from any particular client are applied in the order
- Atomicity
  - Updates either succeed or fail.
- Single system image
  - A client will see the same view of the system, The new server will not accept the connection until it has caught up.
- Durability
  - Once 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,

Cloud Computing and Distributed Systems | Design of Zookeeper

Therefore, there is guarantees in their coordination service. The first one is called it ensure the sequential consistency are applied through the leader, and which ensures the sequential consistency, which is the strongest form of consistency. The second guarantees is done, which is called atomicity that means, all this operation read and write, they are all atomic that is updates or eithers fail or successful the consistency and avoids the race condition.

Single system image the client will see only a view of the system as a file system; therefore it will provide a very simple interface to program in such a large scale distributed system. Durability that once an, that is whenever in memory update is done, it has to be made persistent and also it will be stored in the log, so that it can be recovered even inspite of the failures. Timeliness also is guaranteed that a client to see the stale data, the server will shut down here in this particular case.

(Refer Slide Time: 35:58)



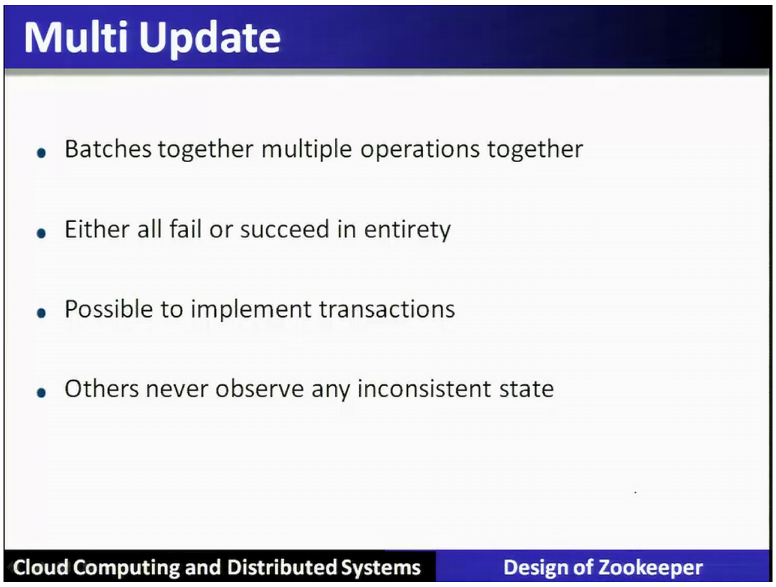
Operations

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

Cloud Computing and Distributed Systems   Design of Zookeeper

There are different set of operations, which are supported in the form of APIs. Create that we have already seen, delete also, we will delete the znode, exists or ls, getACL, getData.

(Refer Slide Time: 36:16)



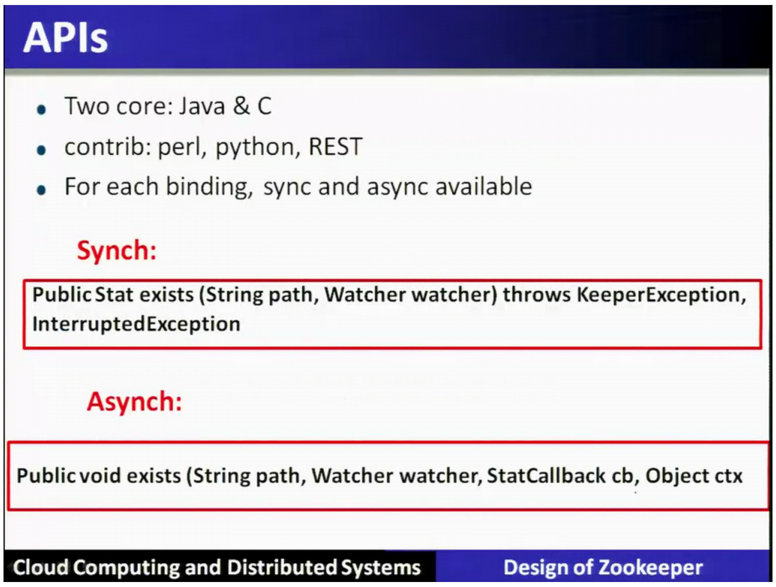
Multi Update

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

Cloud Computing and Distributed Systems   Design of Zookeeper

There are (Refer Time: 36:16) for multiple updates, so that are batch together, and they are supported in the form of transactions.

(Refer Slide Time: 36:25)



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
```

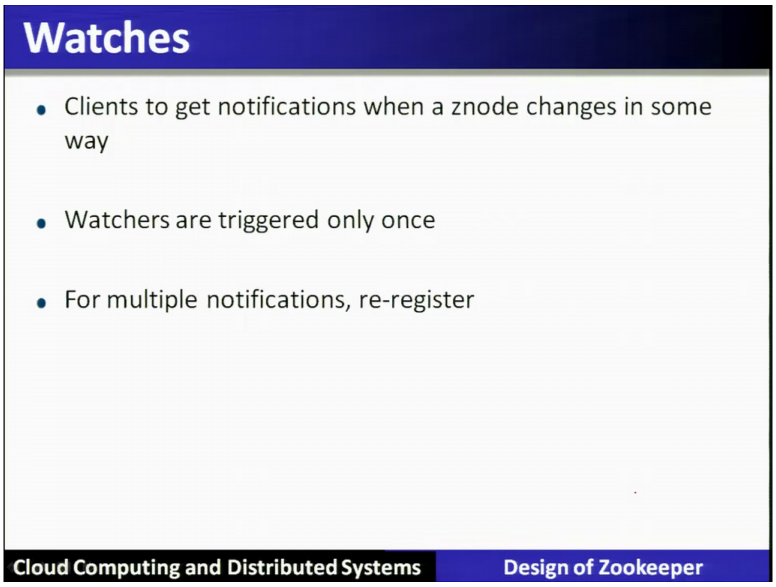
Asynch:

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

Cloud Computing and Distributed Systems | Design of Zookeeper

Different APIs are there.

(Refer Slide Time: 36:29)



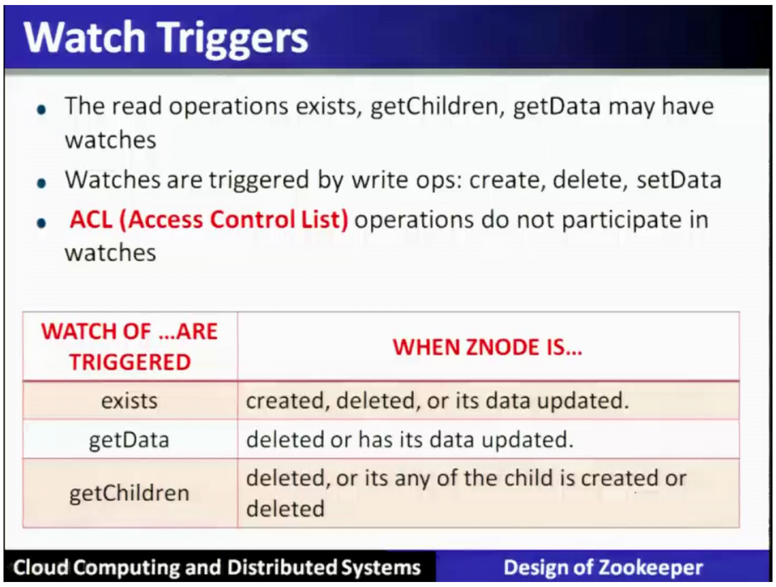
Watches

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

Cloud Computing and Distributed Systems | Design of Zookeeper

Now, there is also a construct, called watches the clients to get notification, whenever there is a change in znode. So, watches are triggered only once.

(Refer Slide Time: 36:41)



Watch Triggers

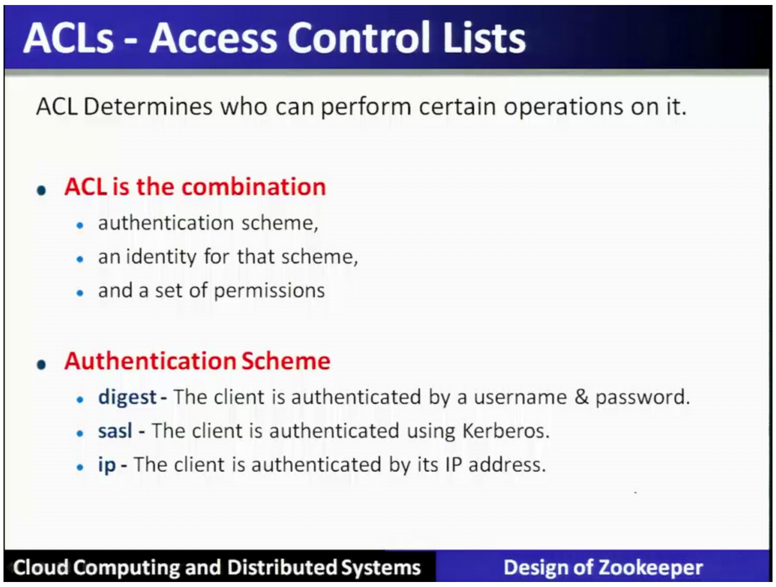
- The read operations exists, getChildren, getData may have watches
- Watches are triggered by write ops: create, delete, setData
- ACL (Access Control List) operations do not participate in watches

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

Cloud Computing and Distributed Systems | Design of Zookeeper

So, watch triggers are being supported through the APIs, using access control list.

(Refer Slide Time: 36:46)



ACLs - Access Control Lists

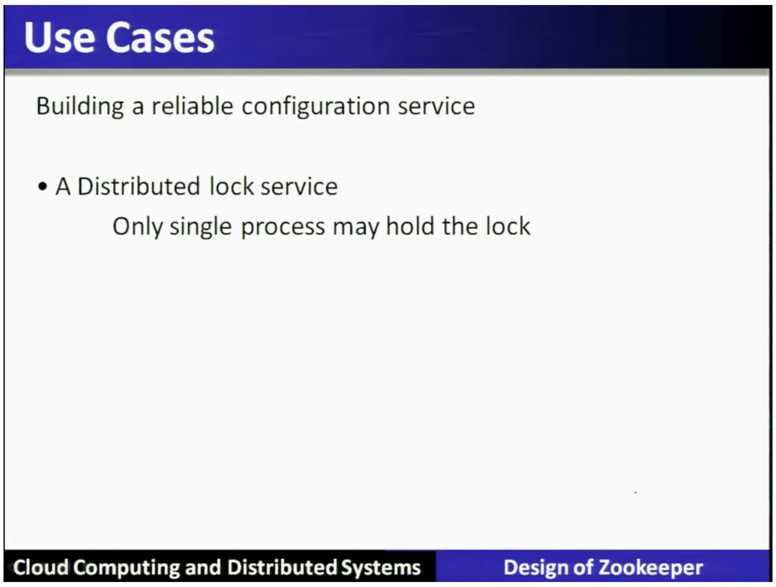
ACL Determines who can perform certain operations on it.

- 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.

Cloud Computing and Distributed Systems | Design of Zookeeper

So, access control list will determine who can perform certain operations on that particular data that is znode. So, ACL is a combination of authentication scheme, an identity for that scheme, and a set of permissions. For example, the authentication schemes, which are used in the access control of the operations are digest that is the simplest form of authentication, where the client is authenticated only by the user and password. sasl is a authentication scheme, where the client is having the session token that is it is being authenticated using Kerberos. And hence, that particular token is available, and he is authenticated already by the Kerberos to used in the system. ip based authenticcation the machine from which the client is accessing, that also is one of the authentication scheme used in ACL that is Access Control List.

(Refer Slide Time: 38:07)



Use Cases

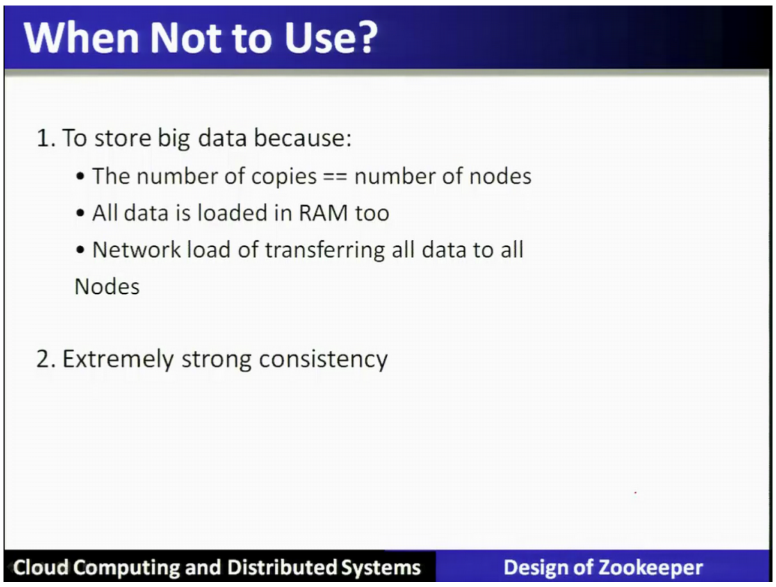
Building a reliable configuration service

- A Distributed lock service
  - Only single process may hold the lock

Cloud Computing and Distributed Systems | Design of Zookeeper

The reliable configuration service. Another one is called distributed lock service.

(Refer Slide Time: 38:13)



When Not to Use?

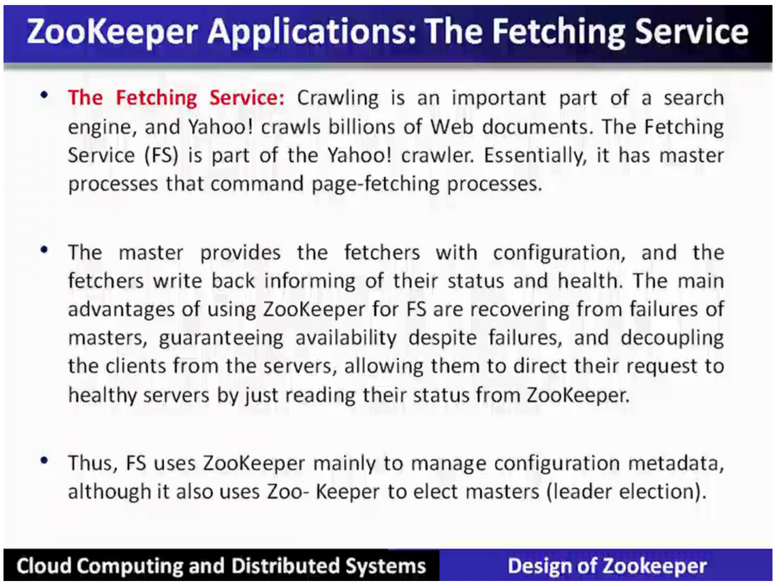
1. To store big data because:
   - The number of copies == number of nodes
   - All data is loaded in RAM too
   - Network load of transferring all data to all Nodes

2. Extremely strong consistency

Cloud Computing and Distributed Systems | Design of Zookeeper

And when not to use them is for storing the big data, because number of copies all are stored in memory. Therefore, and also it is supporting the strong consistency, which is not to be used in a big data scenario.

(Refer Slide Time: 38:30)



ZooKeeper Applications: The Fetching Service

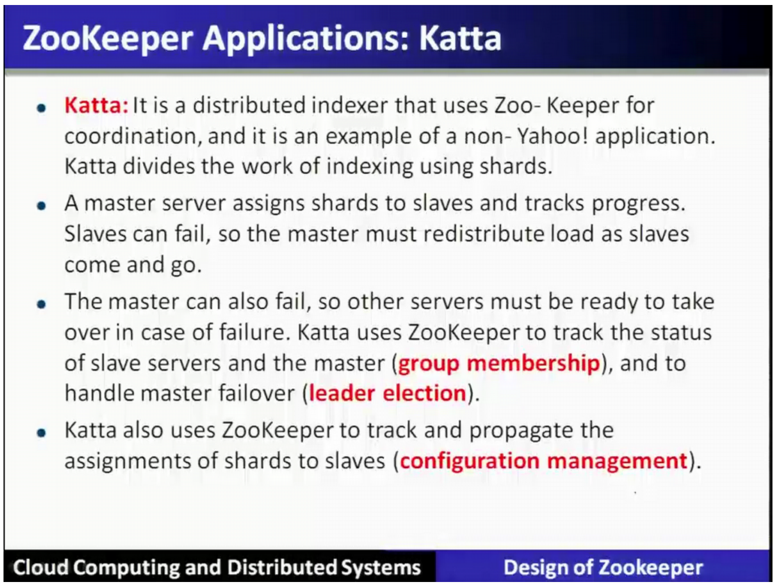
- **The Fetching Service:** Crawling is an important part of a search engine, and Yahoo! crawls billions of Web documents. The Fetching Service (FS) is part of the Yahoo! crawler. Essentially, it has master processes that command page-fetching processes.
- The master provides the fetchers with configuration, 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).

Cloud Computing and Distributed Systems | Design of Zookeeper

Zookeeper applications. There are many zookeeper, let us scan through one by one. The first type of application, which we are discussing is about fetching service. Now, we have seen that the crawling is an important part of the search engine, so the different search engines let us say Yahoo crawls billions of web documents. And the fetching service is a part of the crawler. Essentially, it has a master processes that commands page fetching processes.

So, the master provides that forming their status and the health. The main advantage are the recovering from the failures of the masters, guaranteeing availability, despite failures, decoupling the clients from the servers, allowing them to directly to direct their request to the healthy servers by just reading their status from zookeeper. Thus, fetching service uses the zookeeper mainly to manage the configuration metadata, although it also and the leader election.

(Refer Slide Time: 39:55)



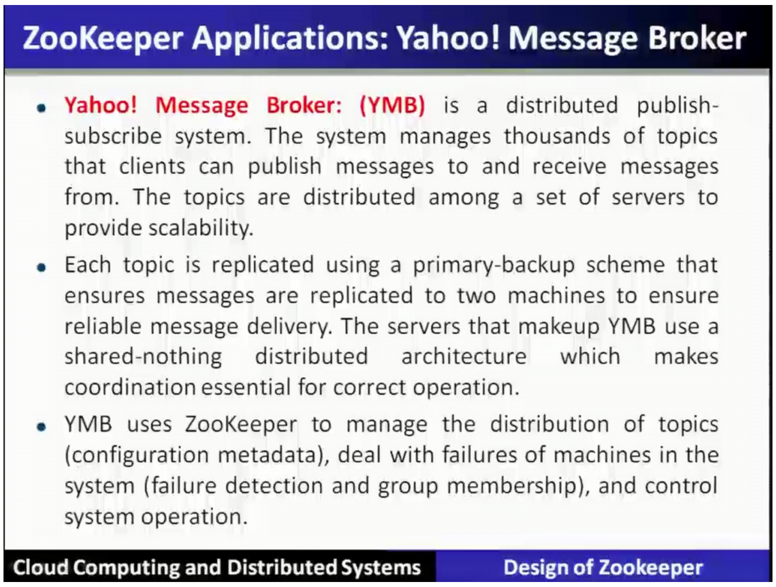
ZooKeeper Applications: Katta

- **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**).

Cloud Computing and Distributed Systems | Design of Zookeeper

Another application of zookeeper is katta. So, in katta, it is a distributed indexer that uses zoo-keeper for the coordination. So, katta device the work of indexing into the shards, the master server assigns the shards, to the slaves and track the progress. The slaves can fail, so the master can redistribute also fail. Servers must be ready to take over katta uses zookeeper to track the status of the slave and the master that is it mention it ensures the group membership, and also handles the master failover that is using the leader election. So, zookeeper is used in two different forms that is as the group membership, and as the leader election. So, katta also uses zookeeper to track and propagate the assignment of the shards to the slaves that is configuration management. So, all three different use cases we can see.

(Refer Slide Time: 40:53)



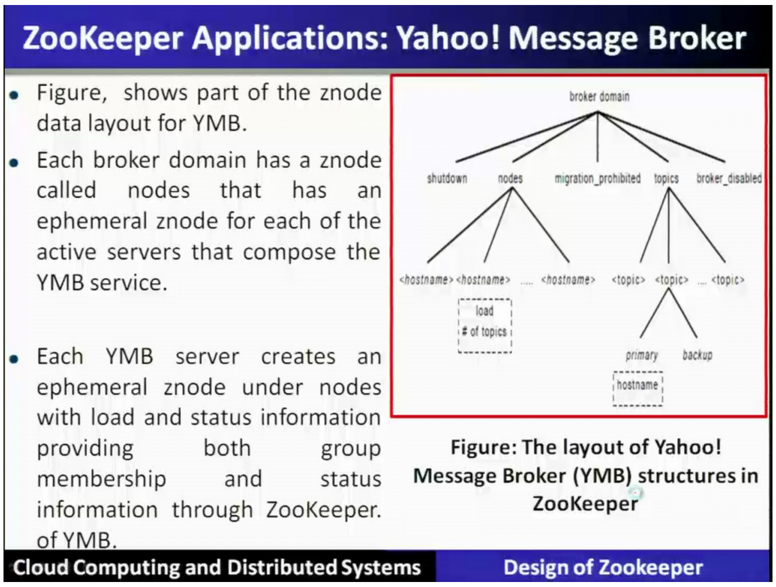
ZooKeeper Applications: Yahoo! Message Broker

- **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.

Cloud Computing and Distributed Systems Design of Zookeeper

The next application yahoo message is a distributed publish subscribe system. The system manages thousands of topic that the client can publish messages and receive the messages from the topics are distributed among the set of servers to provide scalability. Each that ensures messages are replicated to two machines to ensure the reliable message delivery. The servers that message broker use a shared in a coordination essential for correct operation;, zookeeper to manage the distribution of topics that is called configuration management, the failures of the machine that is the failure detection through the zookeeper and also the group membership, and control system operations in detail, the example shown over here about this message broker.

(Refer Slide Time: 41:46)



ZooKeeper Applications: Yahoo! Message Broker

- Figure, shows part of the znode data layout for YMB.
- Each broker domain has a znode called nodes that has an ephemeral znode for each of the active servers that compose the YMB service.
- Each YMB server creates an ephemeral znode under nodes with load and status information providing both group membership and status information through ZooKeeper. of YMB.

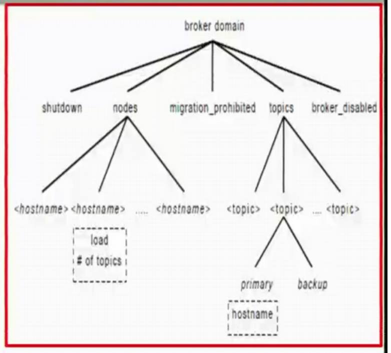
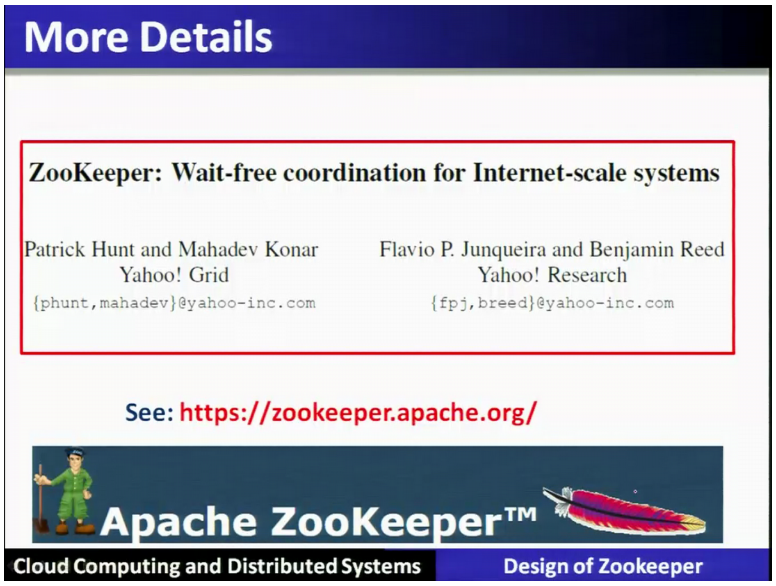


Figure: The layout of Yahoo! Message Broker (YMB) structures in ZooKeeper

Cloud Computing and Distributed Systems | Design of Zookeeper

(Refer Slide Time: 41:51)



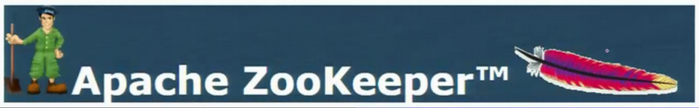
More Details

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

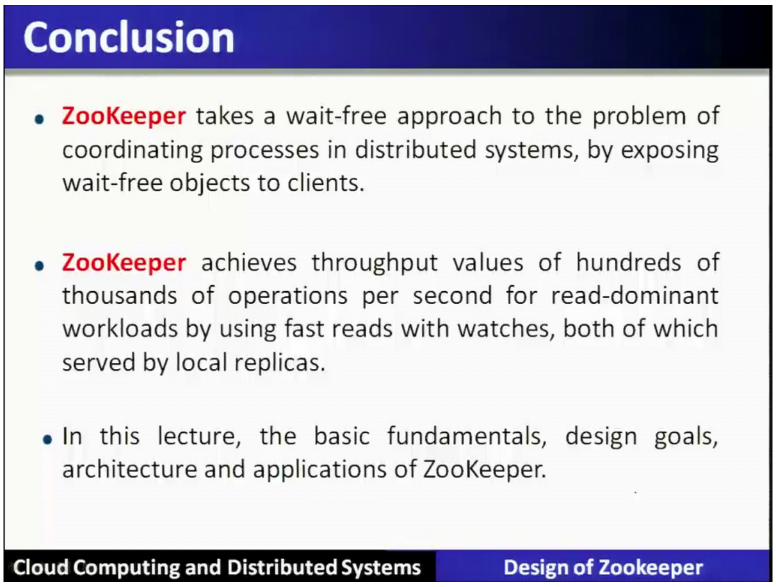
See: https://zookeeper.apache.org/



Cloud Computing and Distributed Systems | Design of Zookeeper

Now, for more details about the zookeeper, wait-free coordination from the people at Yahoo. So, for more details about the zookeeper, you are welcome to see this particular paper.

(Refer Slide Time: 42:09)



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, the basic fundamentals, design goals, architecture and applications of ZooKeeper.

Cloud Computing and Distributed Systems | Design of Zookeeper

Conclusion. Zookeeper takes a wait-free approach processes in the distributed systems, by exposing wait-free objects to the clients. Zookeeper achieves the throughput of operation per second for read-dominant workloads by using fast reads with watches, both of which are the importance of the coordination in different distributed applications. We have also seen many applications, which require zookeeper for either the leader election; or the group membership; or for the failure detection particular lecture. Zookeeper is now being used at different services, which are running either by the blue chip companies uses this zookeeper as a service.

So, in this lecture, we have covered the basic fundamentals that is how the failure toil the zookeeper. This Particular (Refer Time: 43:36), if it is going to be tolerant with one fault, then at least 2 f plus 1 number of servers are required. So, we have seen that to tolerate the failure of one master, at least three different servers are required. To tolerate two failures, it requires seven different servers.

We have also seen the role of majority in the zookeeper; majority is also a kind of quorums. So, if majority is not available, then there is no leader and all of the servers will become the followers. Hence, to ensure that the zookeeper is doing its coordination service for the application, it is very much required that the availability of majority ensemble is very much needed for the correct behavior or the working of a zookeeper.

The zookeeper is not done in one server; at least three servers are required because of the fault tolerance. So, the scaling of zookeeper is very much needed as the size of the application grows. So, also the tolerances many servers are required to run the zookeeper service. Design goals; and also seen the architecture that is there are two different phases; one is called the leader election, where in it will elect the leader out of the ensemble, and the second phase atomic broadcast, we have seen that for doing the updates or the writes the clients have to contact through the server through the leader.

So, the leader will now updates the slaves. And if the majority has agreed, then only the update is finalized. Why the majority or the quorum that we have already seen that it will ensure the consistency that means, if we take the intersection of any two majority or the quorums, there must be one at least one common member, who has already updated. So, for the writes, if it is followed the majority or the quorum that means, they will support the strong consistency without asking from call.

So, all the concepts, which we have seen is being applied here in the zookeeper and we have also seen different applications of the zookeeper, and this gives a good fundamentals, how the cloud system and distributed system in a fault tolerant way are used in the production scenarios.

Thank you.