SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

APACHE ZooKeeper™

Department of MCA

Course: 16CA817 - Big Data Analytics

Unit IV : Hadoop Environment

V Semester / III MCA

Motivation

What makes Distributed System coordination difficult?

DEADLOCK

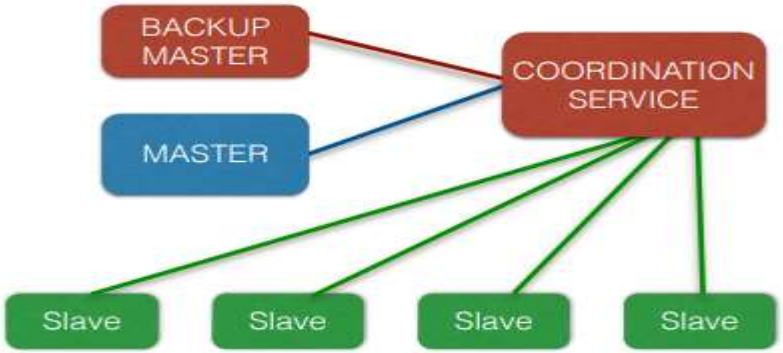
Motivation

Sender does not know:

- whether the message was received
- whether the receiver's process died before/after processing the message

Partial failures make application writing difficult

Motivation



Typical coordination problems

- Static configuration
- Dynamic Configuration
- Group Membership
- Leader selection
- Mutually exclusive access
- Barriers

Zookeeper

- Distributed co-ordination service to manage large set of hosts

Zookeeper

- It allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers
- Allows developers to focus on core application logic rather than distributed nature of the application
- Used by a cluster to coordinate between themselves and maintain shared data with robust synchronization techniques
- Highly scalable
- Handles partial failures in distributed systems

Zookeeper services

- **Naming service** – Identifying the nodes in a cluster by name like DNS
- **Configuration management** - Latest and up-to-date configuration information of the system for a joining node
- **Cluster management** – Joining / leaving of a node in a cluster
- **Leader election** – Electing a node as leader for coordination purpose
- **Locking and synchronization service** – Locking the data while modifying it
- **Highly reliable data registry** – Availability of data even when one / few nodes are down

Zookeeper operations

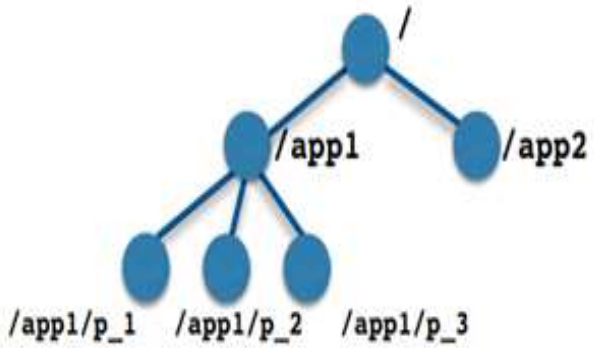
Operation	Description
`create`	Creates a znode (the parent znode must already exist)
`delete`	Deletes a znode (the znode must not have any children)
`exists`	Tests whether a znode exists and retrieves its metadata
`getACL, setACL`	Gets/sets the ACL for a znode
`getChildren`	Gets a list of the children of a znode
`getData, setData`	Gets/sets the data associated with a znode
`sync`	Synchronizes a client's view of a znode with ZooKeeper

Zookeeper characteristics

- **Simple -** stripped-down file system with few operations
- **Expressive** - Primitives are a rich set of building blocks used to build large class of data structure/protocol
- **highly available** –runs on a collection of machines and is designed to be highly available
- **loosely coupled interactions** –participant processes do not need to know about one another
- **Library**- provides an open source, shared repository of implementations and recipes of common coordination patterns

Zookeeper Data model

- Maintains a hierarchical tree of nodes called znodes
- znode stores data and has an associated ACL
- Data access is atomic. A client reading the data stored at a znode entirety, or the read will fail. Similarly, a write will replace all the data associ with a znode
- Znodes are referenced by paths, which in ZooKeeper are represented as slash-delimited character strings



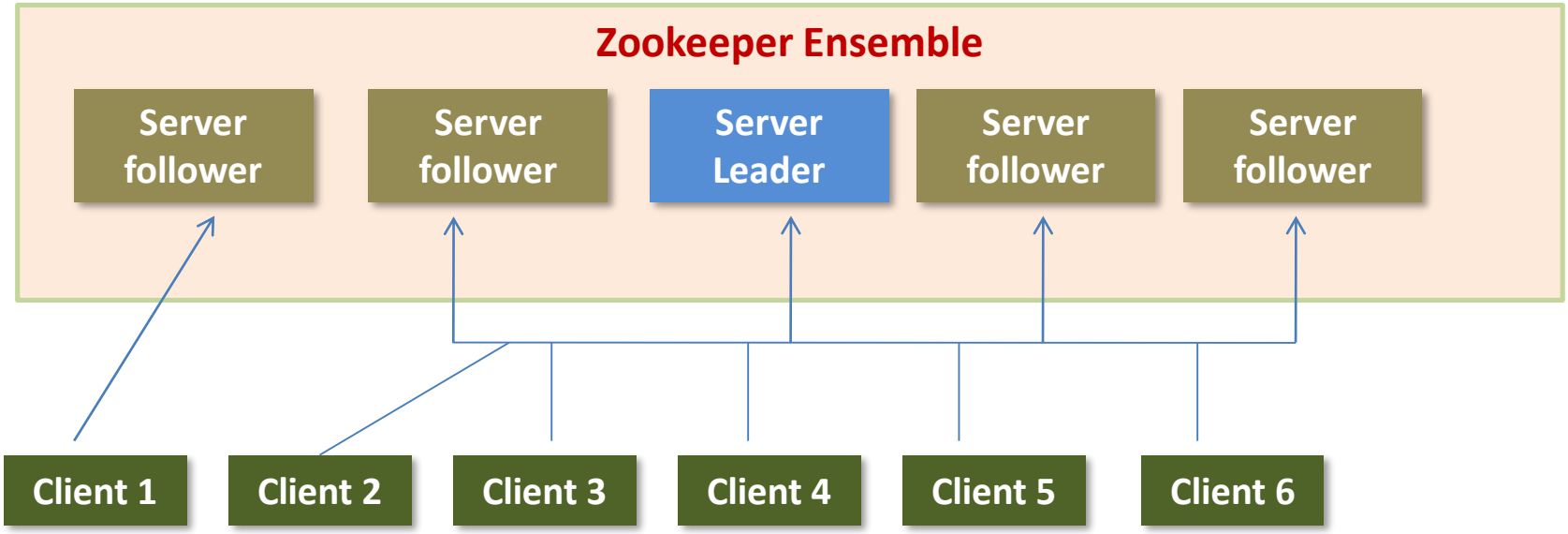
Properties of Znode

- Znodes can be one of two types: ephemeral or persistent
- **Ephemeral znode** is deleted by client's session ends.
- **Persistent znode** is not tied to the client's session and is deleted only when explicitly deleted by a client
- **Sequence numbers:** A *sequential* znode is given a sequence number by ZooKeeper as a part of its name
- **Watches :** allow clients to get notifications when a znode changes in some way
- **Sessions**: client initiates session and has an associated timeout

Architecture

- Follows the Client-Server Architecture
- **Ensemble:** Collection of all the Server nodes in the Zookeeper ecosystem (Min. 3)
- **Server:** one among-st the other servers, provide all sorts of services to its clients
- **Server Leader:** elected at the service startup, performs automatic data recovery for clients
- **Follower:** one of the servers, is to follow the orders passed by the Leader
- **Client:** request service from the server

Architecture



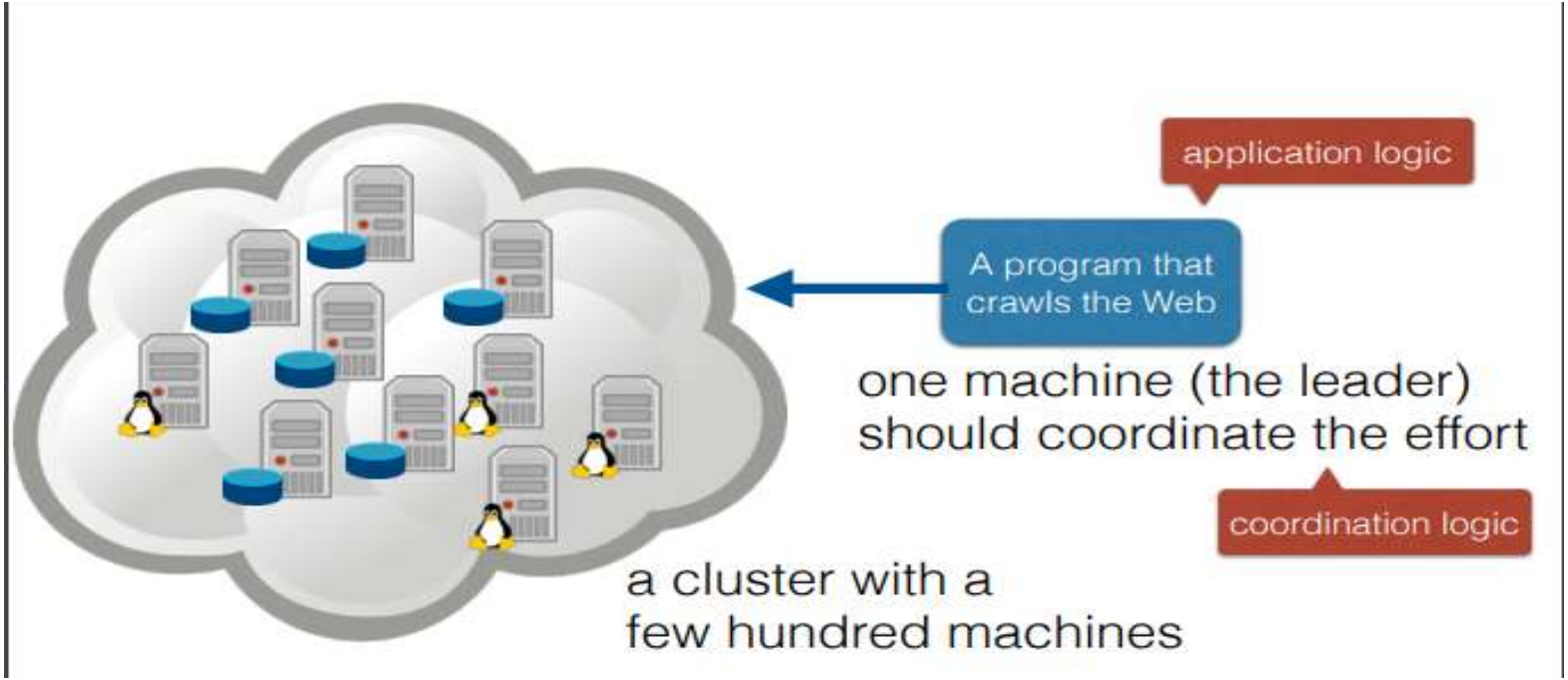
Zookeeper Implementation

- ZooKeeper service can run in two modes.
- In *standalone mode*, there is a single ZooKeeper server
- In production, ZooKeeper runs in *replicated mode*, on a cluster of machines called an *ensemble*
- high-availability is achieved through replication
- It uses a protocol called Zab that runs in two phases which may be repeated indefinitely
  - ***Phase 1: Leader election***
  - ***Phase 2: Atomic broadcast***

Leader selection

- Machines in an ensemble go through a process of electing a distinguished member, called the *leader*
- The other machines are termed *followers*
- Process is finished once a majority of followers have synchronized their state with the leader

Leader Selection



Atomic broadcast

- All write requests are forwarded to the leader, which broadcasts the update to the followers
- When a majority have persisted the change, the leader commits the update, and the client gets a response saying the update succeeded

References

- Tom White, “ Hadoop: The Definitive Guide” Third Edition, O’reilly Media, 4th Edition, 2012
- https://www.edureka.co/blog/zookeeper-tutorial/
- https://www.tutorialspoint.com/zookeeper/zookeeper_installation.htm