Zookeeper

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

ZooKeeper is a distributed co-ordination service to manage large set of hosts. Co-ordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API. ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

The ZooKeeper framework was originally built at "Yahoo!" for accessing their applications in an easy and robust manner. Later, Apache ZooKeeper became a standard for organized service used by Hadoop, HBase, and other distributed frameworks. For example, Apache HBase uses ZooKeeper to track the status of distributed data. This tutorial explains the basics of ZooKeeper, how to install and deploy a ZooKeeper cluster in a distributed environment, and finally concludes with a few examples using Java programming and sample applications.

Audience

This tutorial has been prepared for professionals aspiring to make a career in Big Data Analytics using ZooKeeper framework. It will give you enough understanding on how to use ZooKeeper to create distributed clusters.

Prerequisites

Before proceeding with this tutorial, you must have a good understanding of Java because the ZooKeeper server runs on JVM, distributed process, and Linux environment.

Copyright & Disclaimer

Table of Contents

1. ZOOKEEPER – OVERVIEW

ZooKeeper is a distributed co-ordination service to manage large set of hosts. Co-ordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API. ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

The ZooKeeper framework was originally built at "Yahoo!" for accessing their applications in an easy and robust manner. Later, Apache ZooKeeper became a standard for organized service used by Hadoop, HBase, and other distributed frameworks. For example, Apache HBase uses ZooKeeper to track the status of distributed data.

Before moving further, it is important that we know a thing or two about distributed applications. So, let us start the discussion with a quick overview of distributed applications.

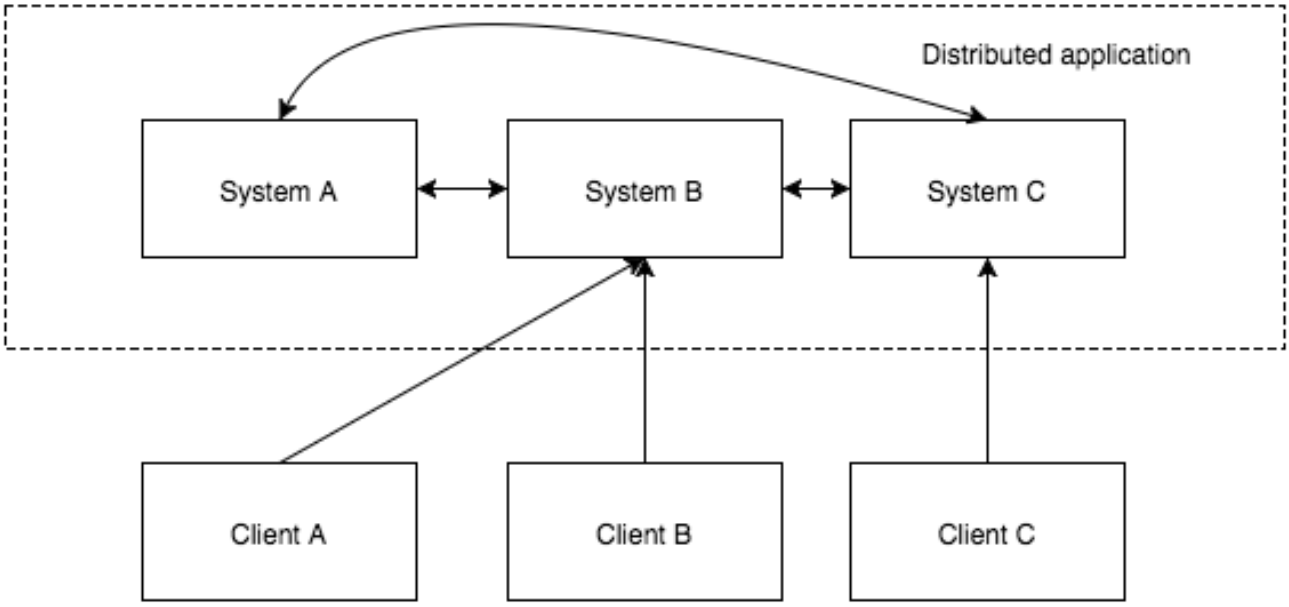
Distributed Application

A distributed application can run on multiple systems in a network at a given time (simultaneously) by coordinating among themselves to complete a particular task in a fast and efficient manner. Normally, complex and time-consuming tasks, which will take hours to complete by a non-distributed application (running in a single system) can be done in minutes by a distributed application by using computing capabilities of all the system involved.

The time to complete the task can be further reduced by configuring the distributed application to run on more systems. A group of systems in which a distributed application is running is called a **Cluster** and each machine running in a cluster is called a **Node**.

A distributed application has two parts, **Server** and **Client** application. Server applications are actually distributed and have a common interface so that clients can connect to any server in

the cluster and get the same result. Client applications are the tools to interact with a distributed application.



Benefits of Distributed Applications

- **Reliability** – Failure of a single or a few systems does not make the whole system to fail.
- **Scalability** – Performance can be increased as and when needed by adding more machines with minor change in the configuration of the application with no downtime.
- **Transparency** – Hides the complexity of the system and shows itself as a single entity / application.

Challenges of Distributed Applications

- **Race condition** - Two or more machines trying to perform a particular task, which actually needs to be done only by a single machine at any given time. For example, shared resources should only be modified by a single machine at any given time.
- **Deadlock** – Two or more operations waiting for each other to complete indefinitely.
- **Inconsistency** – Partial failure of data.

What is Apache ZooKeeper Meant For?

Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques. ZooKeeper is itself a distributed application providing services for writing a distributed application.

The common services provided by ZooKeeper are as follows:

- **Naming service** – Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.

- **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 and node status at real time.

- **Leader election** – Electing a node as leader for coordination purpose.

- **Locking and synchronization service** – Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.

- **Highly reliable data registry** – Availability of data even when one or a few nodes are down.

Distributed applications offer a lot of benefits, but they throw a few complex and hard-to-crack challenges as well. ZooKeeper framework provides a complete mechanism to overcome all the challenges. Race condition and deadlock are handled using **fail-safe synchronization approach**. Another main drawback is inconsistency of data, which ZooKeeper resolves with **atomicity**.

Benefits of ZooKeeper

Here are the benefits of using ZooKeeper:

- **Simple distributed coordination process**

- **Synchronization** – Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.

- **Ordered Messages**

- **Serialization** – Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queue to execute running threads.

- **Reliability**

- **Atomicity** – Data transfer either succeed or fail completely, but no transaction is partial.

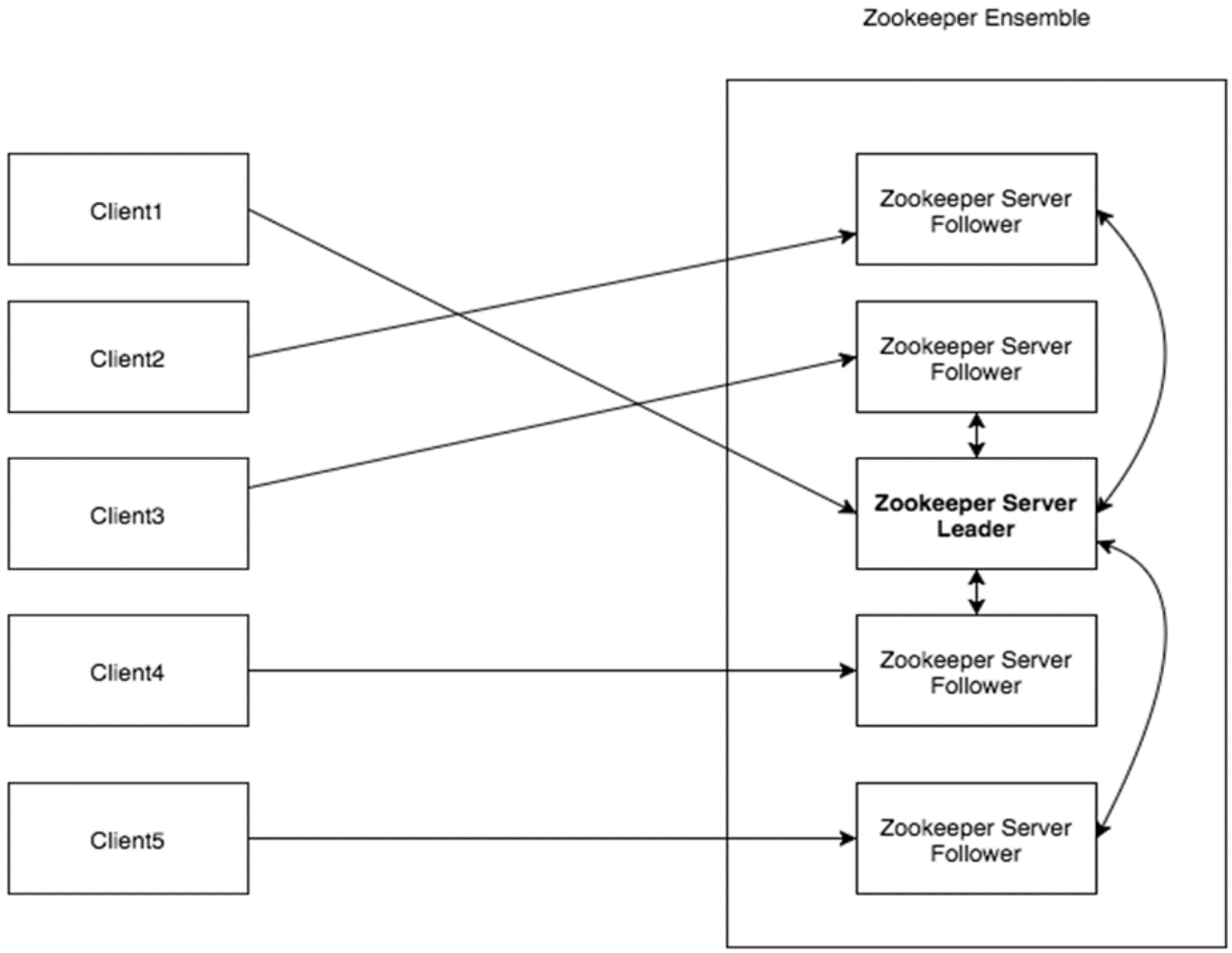
2. ZOOKEEPER – FUNDAMENTALS

Before going deep into the working of ZooKeeper, let us take a look at the fundamental concepts of ZooKeeper. We will discuss the following topics in this chapter:

- Architecture
- Hierarchical namespace
- Session
- Watches

Architecture of ZooKeeper

Take a look at the following diagram. It depicts the "Client-Server Architecture" of ZooKeeper.



Each one of the components that is a part of the ZooKeeper architecture has been explained in the following table.

Part	Description
Client	Clients, one of the nodes in our distributed application cluster, access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive. Similarly, the server sends an acknowledgement when a client connects. If there is no response from the connected server, the client automatically redirects the message to another server.
Server	Server, one of the nodes in our ZooKeeper ensemble, provides all the services to clients. Gives acknowledgement to client to inform that the server is alive.
Ensemble	Group of ZooKeeper servers. The minimum number of nodes that is required to form an ensemble is 3.
Leader	Server node which performs automatic recovery if any of the connected node failed. Leaders are elected on service startup.
Follower	Server node which follows leader instruction.

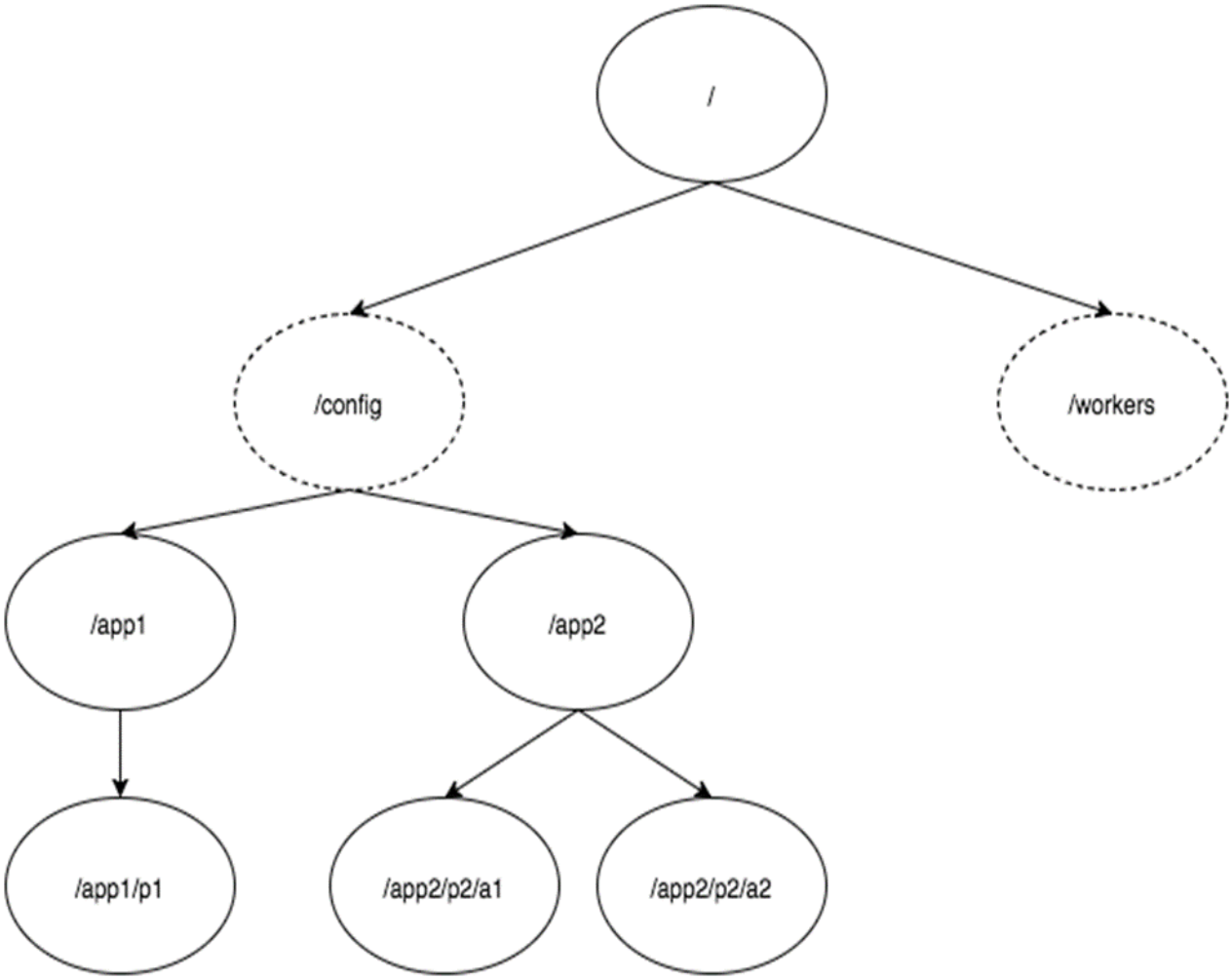
Hierarchical Namespace

The following diagram depicts the tree structure of ZooKeeper file system used for memory representation. ZooKeeper node is referred as **znode**. Every znode is identified by a name and separated by a sequence of path (/).

- In the diagram, first you have a root **znode** separated by "/". Under root, you have two logical namespaces **config** and **workers**.

- The **config** namespace is used for centralized configuration management and the **workers** namespace is used for naming.

- Under **config** namespace, each znode can store upto 1MB of data. This is similar to UNIX file system except that the parent znode can store data as well. The main purpose of this structure is to store synchronized data and describe the metadata of the znode. This structure is called as **ZooKeeper Data Model**.



Every znode in the ZooKeeper data model maintains a **stat** structure. A **stat** simply provides the **metadata** of a znode. It consists of *Version number*, Access Control List (ACL), Timestamp, and Data length.

- **Version number**: Every znode has a version number, which means every time the data associated with the znode changes, its corresponding version number would also increased. The use of version number is important when multiple zookeeper clients are trying to perform operations over the same znode.

- **Access Control List (ACL)**: ACL is basically an authentication mechanism for accessing the znode. It governs all the znode read and write operations.

- **Timestamp**: Timestamp represents time elapsed from znode creation and modification. It is usually represented in milliseconds. ZooKeeper identifies every change to the znodes from "Transaction ID" (zxid). **Zxid** is unique and maintains time for each transaction so that you can easily identify the time elapsed from one request to another request.

- **Data length**: Total amount of the data stored in a znode is the data length. You can store a maximum of 1MB of data.

Types of Znodes

Znodes are categorized as persistence, sequential, and ephemeral.

- **Persistence znode**: Persistence znode is alive even after the client, which created that particular znode, is disconnected. By default, all znodes are persistent unless otherwise specified.

- **Ephemeral znode**: Ephemeral znodes are active until the client is alive. When a client gets disconnected from the ZooKeeper ensemble, then the ephemeral znodes get deleted automatically. For this reason, only ephemeral znodes are not allowed to have a children further. If an ephemeral znode is deleted, then the next suitable node will fill its position. Ephemeral znodes play an important role in Leader election.

- **Sequential znode**: Sequential znodes can be either persistent or ephemeral. When a new znode is created as a sequential znode, then ZooKeeper sets the path of the znode by attaching a 10 digit sequence number to the original name. For example, if a znode with path **/myapp** is created as a sequential znode, ZooKeeper will change the path to **/myapp0000000001** and set the next sequence number as 0000000002. If two sequential znodes are created concurrently, then ZooKeeper never uses the same number for each znode. Sequential znodes play an important role in Locking and Synchronization.

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**