Cloud computing elasticity requires the ability to distribute computations and data across multiple

systems. Coordination among these systems is one of the critical functions to be exercised in a distributed

environment. The coordinationmodel depends on the specific task, such as coordination of data storage,

orchestration of multiple activities, blocking an activity until an event occurs, reaching consensus for

the next action, or recovery after an error.

The entities to be coordinated could be processes running on a set of cloud servers or even running

on multiple clouds. Servers running critical tasks are often replicated, so when one primary server fails,

a backup automatically continues the execution. This is only possible if the backup is in a hot standby

mode – in other words, the standby server shares the same state at all times with the primary.

For example, in the distributed data store model discussed in Section 3.5, the access to data is

mitigated by a proxy. This proxy is a single point of failure; thus, an architecture with multiple proxies

is desirable. These proxies should be in the same state so that, whenever one of them fails, the client

could seamlessly continue to access the data using another proxy.

Consider nowan advertising service that involves a large number of servers in a cloud. The advertising

service runs on a number of servers specialized for tasks such as database access, monitoring, accounting,

event logging, installers, customer dashboards,5 advertising campaign planners, scenario testing, and

so on. A solution to coordinate these activities is through configuration files shared by all systems.

When the service starts or after a system failure, all servers use the configuration file to coordinate their

actions. This solution is static. Any change requires an update and redistribution of the configuration

file. Moreover, in case of a system failure the configuration file does not allow recovery from the state

of each server prior to the system crash, which is a more desirable alternative.

A solution for the proxy coordination problem is to consider a proxy as a deterministic finite state

machine that performs the commands sent by clients in some sequence. The proxy has thus a definite

state and, when a command is received, it transitions to another state. When P proxies are involved,

all of them must be synchronized and must execute the same sequence of state machine commands;

this can be ensured if all proxies implement a version of the Paxos consensus algorithm described in

Section 2.11.

ZooKeeper is a distributed coordination service based on this model. The high-throughput and

low-latency service is used for coordination in large-scale distributed systems. The open-source software

is written in Java and has bindings for Java and C. Information about the project is available at

http://zookeeper.apache.org/.

The ZooKeeper software must first be downloaded and installed on several servers; then clients can

connect to any one of these servers and access the coordination service. The service is available as long

as the majority of servers in the pack are available.

The organization of the service is shown in Figure 4.4. The servers in the pack communicate with

one another and elect a leader. A database is replicated on each one of them and the consistency of the replicas is maintained. Figure 4.4(a) shows that the service provides a single system image. A client

can connect to any server of the pack.

A client uses TCP to connect to a single server. Through the TCP connection a client sends requests

and receives responses and watches events. A client synchronizes its clock with the server. If the server

fails, the TCP connections of all clients connected to it time out and the clients detect the failure of the

server and connect to other servers.

Figures 4.4(b) and (c) show that a read operation directed to any server in the pack returns the same

result, whereas the processing of a write operation is more involved; the servers elect a leader, and

any follower receiving a request from one of the clients connected to it forwards it to the leader. The

leader uses atomic broadcast to reach consensus. When the leader fails, the servers elect a new leader.

The system is organized as a shared hierarchical namespace similar to the organization of a file

system. A name is a sequence of path elements separated by a backslash. Every name in Zookeper’s

namespace is identified by a unique path (see Figure 4.5).

In ZooKeeper the znodes, the equivalent of the inodes of a file system, can have data associated with

them. Indeed, the system is designed to store state information. The data in each node includes version numbers for the data, changes of ACLs,6 and time stamps. A client can set a watch on a znode and

receive a notification when the znode changes. This organization allows coordinated updates. The data

retrieved by a client also contains a version number. Each update is stamped with a number that reflects

the order of the transition.

The data stored in each node is read and written atomically. A read returns all the data stored in

a znode, whereas a write replaces all the data in the znode. Unlike in a file system, Zookeeper data,

the image of the state, is stored in the server memory. Updates are logged to disk for recoverability, and

writes are serialized to disk before they are applied to the in-memory database that contains the entire

tree. The ZooKeeper service guarantees:

1. Atomicity. A transaction either completes or fails.

2. Sequential consistency of updates.Updates are applied strictly in the order in which they are received.

3. Single system image for the clients. A client receives the same response regardless of the server it

connects to.

4. Persistence of updates. Once applied, an update persists until it is overwritten by a client.

5. Reliability. The system is guaranteed to function correctly as long as the majority of servers function

correctly.

To reduce the response time, read requests are serviced from the local replica of the server that is

connected to the client.When the leader receives a write request, it determines the state of the system

where the write will be applied and then it transforms the state into a transaction that captures this

new state.

The messaging layer is responsible for the election of a new leader when the current leader fails. The

messaging protocol uses packets (sequences of bytes sent through a FIFO channel), proposals (units

of agreement), and messages (sequences of bytes atomically broadcast to all servers). A message is

included in a proposal and it is agreed on before it is delivered. Proposals are agreed on by exchanging

packets with a quorum of servers, as required by the Paxos algorithm.

An atomic messaging system keeps all the servers in a pack in synch. This system guarantees (a)

reliable delivery: if message m is delivered to one server, it will be eventually delivered to all servers;

(b) total order: if message m is delivered before message n to one server, m will be delivered before n

to all servers; and (c) causal order: if message n is sent after m has been delivered by the sender of n,

then m must be ordered before n.

The application programming interface (API) to the ZooKeeper service is very simple and consists

of seven operations:

• create – add a node at a given location on the tree.

• delete – delete a node.

• get data – read data from a node.

• set data – write data to a node.

• get children – retrieve a list of the children of the node.

• synch – wait for the data to propagate.

The system also supports the creation of ephemeral nodes, which are nodes that are created when a

session starts and deleted when the session ends.

This brief description shows that the ZooKeeper service supports the finite state machine model of

coordination. In this case a znode stores the state. The ZooKeeper service can be used to implement

higher-level operations such as group membership, synchronization, and so on. The system is used by

Yahoo!’s Message Broker and by several other applications.