Applications of Paxos Algorithm

Gurkan Solmaz
COP 6938 - Cloud Computing - Fall 2012

Department of Electrical Engineering and Computer Science
University of Central Florida - Orlando, FL

Oct 15, 2012

Outline

- Apache ZooKeeper
- Google Chubby
- Other applications

ZooKeeper - Overview

- Apache ZooKeeper: an effort to develop and maintain an open-source server which enables highly reliable distributed coordination

- Exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming

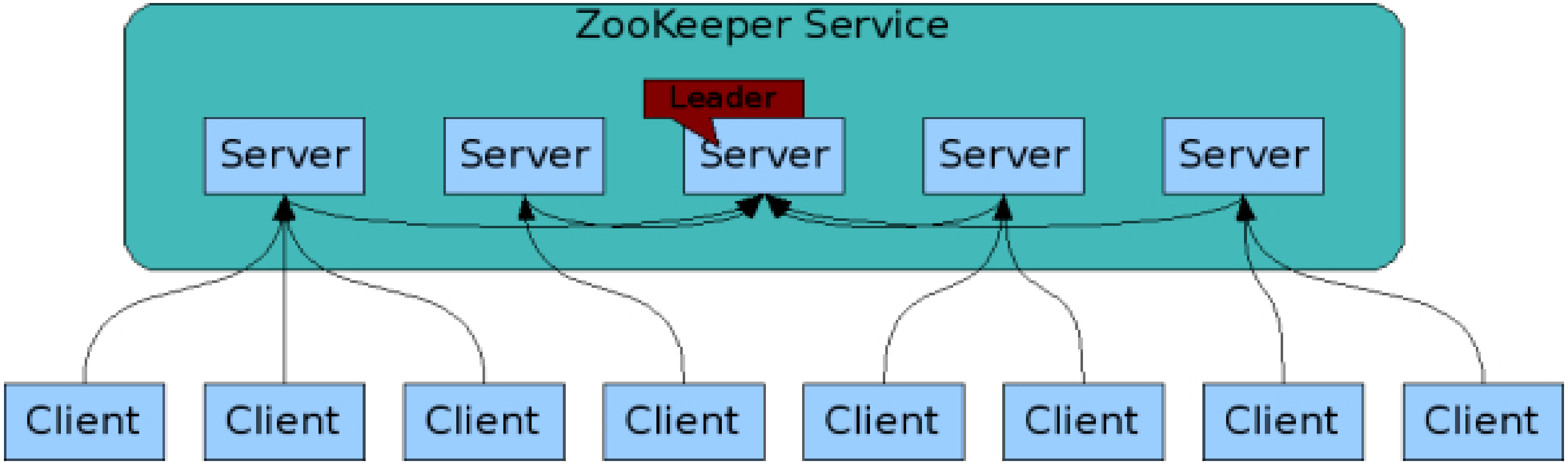
ZooKeeper - Overview

- Designed to be easy to program to
- Runs in Java, has bindings for Java and C
- Uses a data model similar to directory tree structure of file systems
- Coordination services are prone to errors such as race conditions and deadlock
- The motivation is to relieve distributed applications implementing coordination services from scratch

ZooKeeper

- Allows distributed processes to coordinate with each other through a shared hierarchical name space similar to standard file system
- The name space consists of data registers – znodes
- ZooKeeper data is kept in-memory, unlike a file system
- Can achieve high throughput and low latency, can be used in large and distributed systems

ZooKeeper



- Servers that make up the ZooKeeper service must all know about each other

- Servers maintain an in-memory image of state, along with a transaction logs and snapshots in a persistent store

- As long as a majority of the servers are available, the service will be available

ZooKeeper

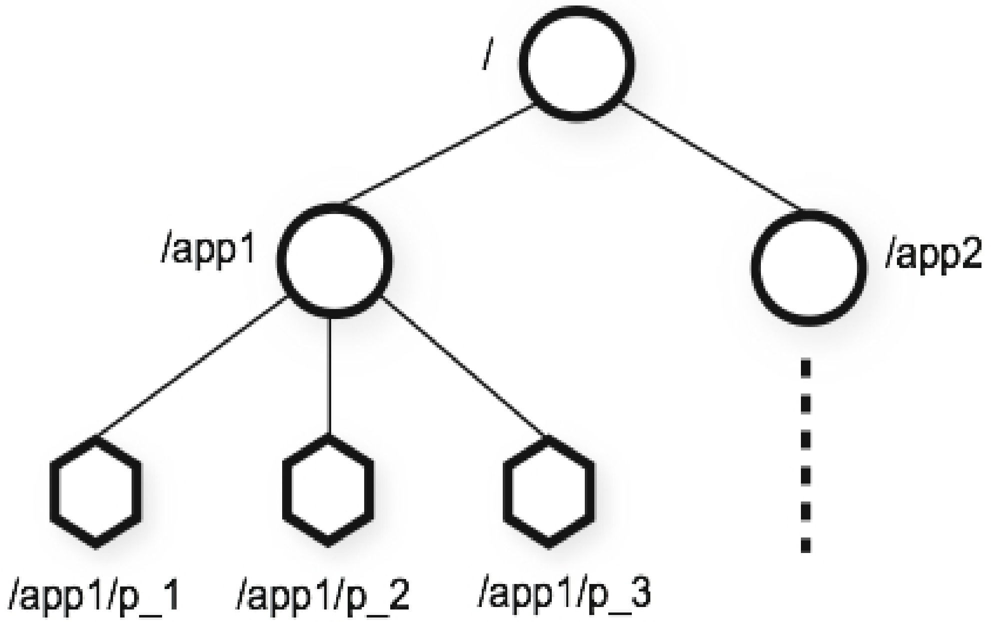
- Clients connect to a single ZooKeeper server

- The client maintains a TCP connection through which it sends requests, gets responses, gets watch events, and sends heart beats

- If the TCP connection to the server breaks, the client will connect to a different server

- ZooKeeper stamps each update with a number that reflects the order of all transactions

ZooKeeper – Name space



- The hierarchical name space provided by ZooKeeper is similar to the standard file system
- A name is a sequence of path elements separated by a slash (/)
- Every node in ZooKeeper's name space is identified by a path

ZooKeeper – Nodes

- Each node in a ZooKeeper namespace can have data associated with it as well as children
- Unlike file system, similar to having a file-system that allows a file to also be a directory
- The term “znode” is used to specify ZooKeeper data nodes
- The data in the nodes: version numbers, ACL changes and time stamps to allow cache validations and coordinated updates

ZooKeeper – Nodes

- The data stored at each znode in a name space is read and written atomically
- Reads get all the data bytes associated with a znode and a write replaces all the data
- Each node has an Access Control List (ACL) that restricts who can do what
- Ephemeral nodes: znodes which exist as long as the session that created the znode is active
- When the session ends the ephemeral node is deleted

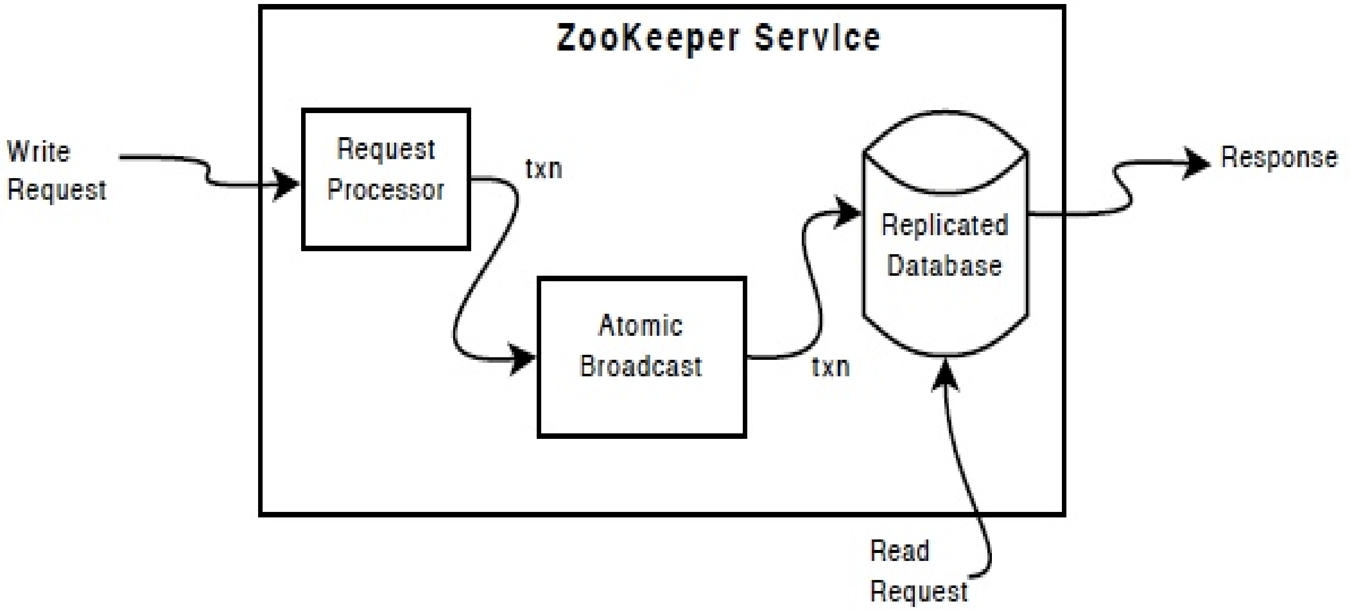
ZooKeeper – Watches

- Clients can set a watch on a znode
- A watch will be triggered and removed when the znode changes
- When a watch is triggered, the client receives a packet saying that the znode has changed
- If the connection between the client and one of the servers is broken, the client will receive a local notification

ZooKeeper – Guarantees

- Sequential Consistency: Updates from a client will be applied in the order that they were sent
- Atomicity: Updates either succeed or fail, no partial results
- Single System Image: A client will see the same view of the service regardless of the server that it connects to
- Reliability: Once an update has been applied, it will persist from that time forward until a client overwrites the update
- Timeliness: The clients view of the system is guaranteed to be up-to-date within a certain time bound

ZooKeeper – Implementation



- The replicated database is an in-memory database containing the entire data tree

- Updates are logged to disk for recoverability, and writes are serialized to disk before they are applied to the in-memory database

ZooKeeper – Implementation

- Every server services clients. Clients connect to exactly one server to submit requests
- Read requests are serviced from the local replica of each server database
- Requests that change the state of the service, write requests, are processed by an agreement protocol
- As part of the agreement protocol all write requests from clients are forwarded to a single server, called the leader

ZooKeeper – Implementation

- The rest of the ZooKeeper servers are called followers
- Followers receive message proposals from the leader and agree upon message delivery
- The messaging layer takes care of replacing leaders on failures and syncing followers with leaders
- ZooKeeper uses a custom atomic messaging protocol

ZooKeeper – Implementation

- Since the messaging layer is atomic, ZooKeeper can guarantee that the local replicas never diverge

- When the leader receives a write request:

→ calculates what the state of the system is, when the write is to be applied

→ transforms the state into a transaction that captures this new state

ZooKeeper – Internals

- At the heart of ZooKeeper is an atomic messaging system that keeps all of the servers in sync
- The guarantees by the messaging system:

→ Reliable delivery: If a message, ***m***, is delivered by one server, it will be eventually delivered by all servers

→ Total order: If ***a*** is delivered before ***b*** by one server, ***a*** will be delivered before ***b*** by all servers

→ Causal order: If ***b*** is sent after ***a*** has been delivered by the sender of ***b***, ***a*** must be ordered before ***b***

ZooKeeper – Internals

- Messaging protocol:

→ Packet: a sequence of bytes sent through a FIFO channel

→ Proposal: a unit of agreement. Proposals are agreed upon by exchanging packets with a quorum of servers (e.g. the NEW_LEADER proposal)

→ Message: a sequence of bytes to be atomically broadcast to all servers. A message put into a proposal and agreed upon before it is delivered

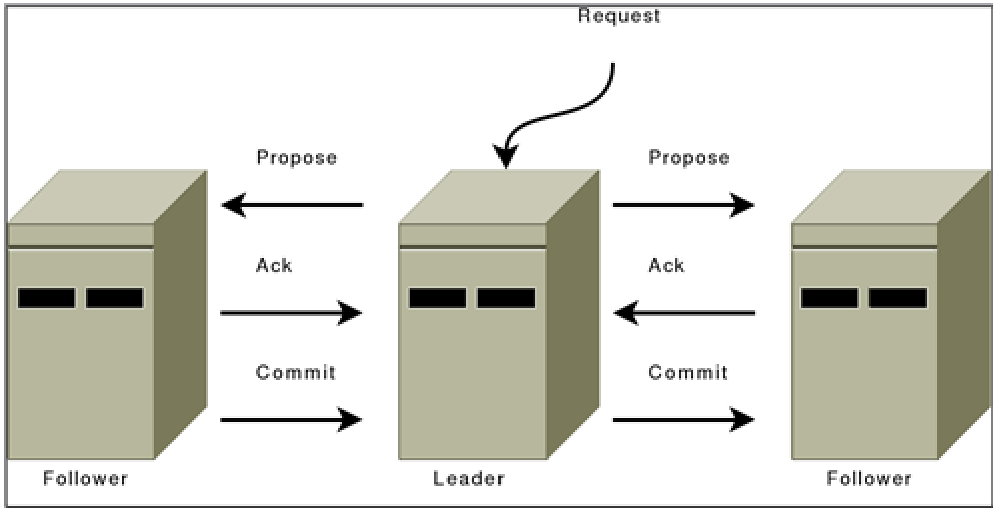
ZooKeeper – Internals

- Two phases of messaging:

→ Leader activation: a leader establishes the correct state of the system and gets ready to start making proposals

→ Active message: a leader accepts messages to propose and coordinates message delivery

ZooKeeper – Internals



- Once the leader is elected, it starts sending proposals
- All communication channels are FIFO, so everything is done in order
- *commit* to all followers as soon as a quorum of followers have *acknowledged* a message

Chubby - Overview

- Google`s distributed lock service
- A fault-tolerant system that provides a distributed locking mechanism and stores small files
- Typically there is one Chubby instance, or “cell”, per data center
- Several Google systems such as the Google File System (GFS) and Bigtable use Chubby for distributed coordination and to store a small amount of metadata

Chubby

- Chubby achieves fault-tolerance through replication

- A typical Chubby cell consists of five replicas, running the same code, each running on a dedicated machine

- Every Chubby object (e.g., a Chubby lock or file) is stored as an entry in a database, it is this database that is replicated

- At any one time, one of the replicas is considered to be the “master”

Chubby

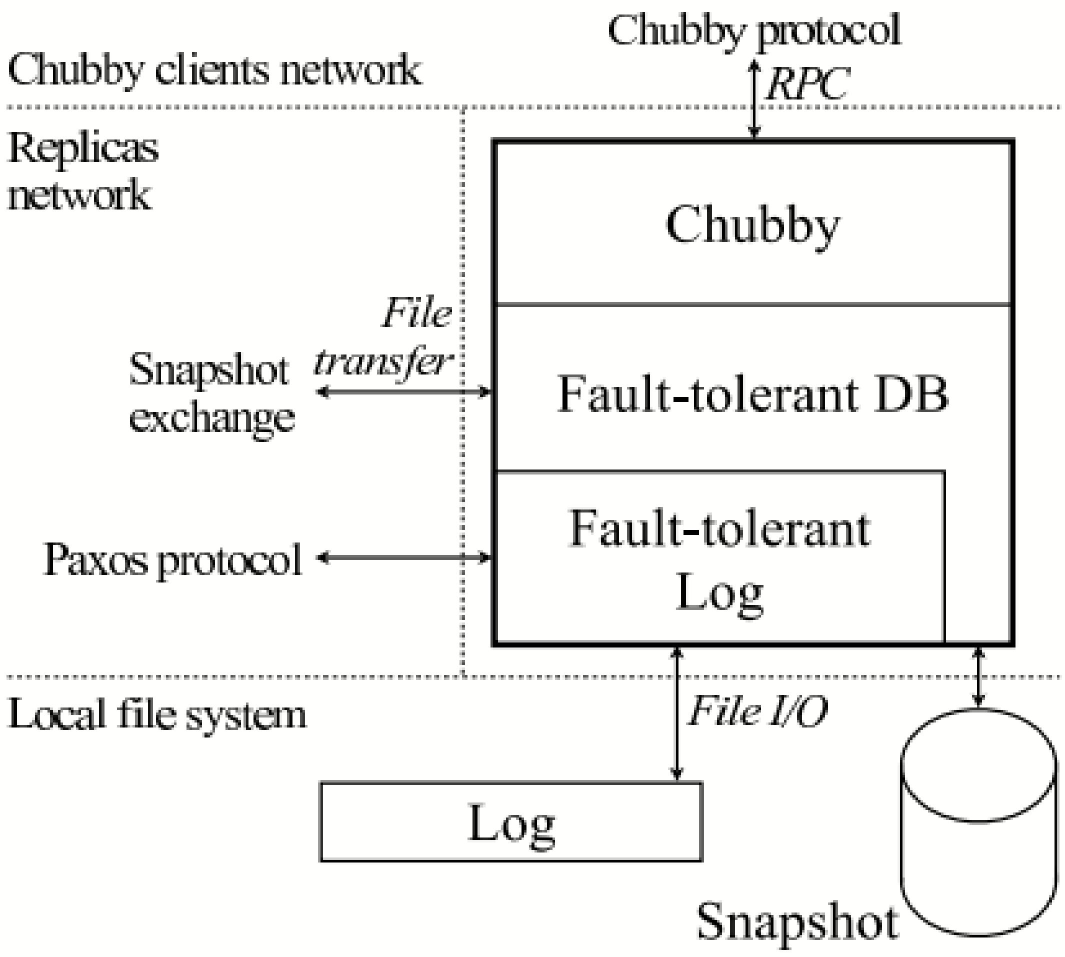
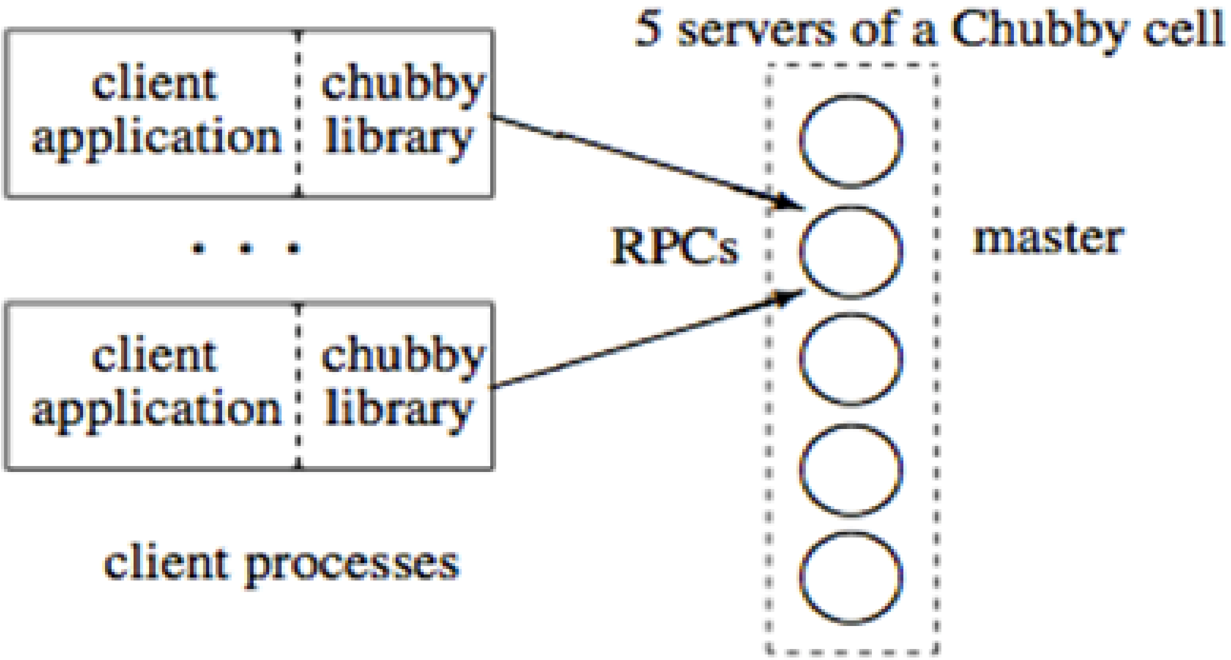


Figure 1: A single Chubby replica.

Chubby

- 2 components: A server and a client library
- Master is selected in the cell using Paxos algorithm



Other Applications

- The OpenReplica replication service:

  → uses Paxos to maintain replicas for an open access system that enables users to create fault-tolerant objects

  → provides high performance through concurrent rounds and flexibility through dynamic membership changes

- Autopilot cluster management service:

  → A service used by Microsoft for Bing

Other Applications

- Google Spanner:

  → A massively scalable distributed database NewSQL platform designed by Google

  → Used internally within their infrastructure as part of the Google platform

  → Uses the Paxos algorithm, makes heavy use of hardware-assisted time synchronization using GPS clocks and atomic clocks to ensure global consistency

References

- Dan C. Marinescu "Cloud Computing: Theory and Practice"
- "Apache ZooKeeper", http://zookeeper.apache.org
- T. Chandra, R. Griesemer and J. Redstone (2007). "Paxos Made Live - An Engineering Perspective"
- Michael Isard (2007), "Autopilot: Automatic Data Center Management"
- Philipp Lenssen, "Google Chubby and the Paxos Algorithm".
- Zhang Yang, "Chubby and Paxos" presentation.
- "Wikipedia," http://en.wikipedia.org/wiki/Paxos_(computer_science).

Thank you !