CS 5412 - Cloud Computing

Recitation 04/11: Zookeeper

Sagar Jha

April 11, 2018

Different Forms of Coordination

- Configuration - List of operational parameters for the nodes
- Group membership - membership of components of the system
- Leader election - In general, roles for each node in the system

Goal - We want a common framework for building different coordination services

Locking Approach

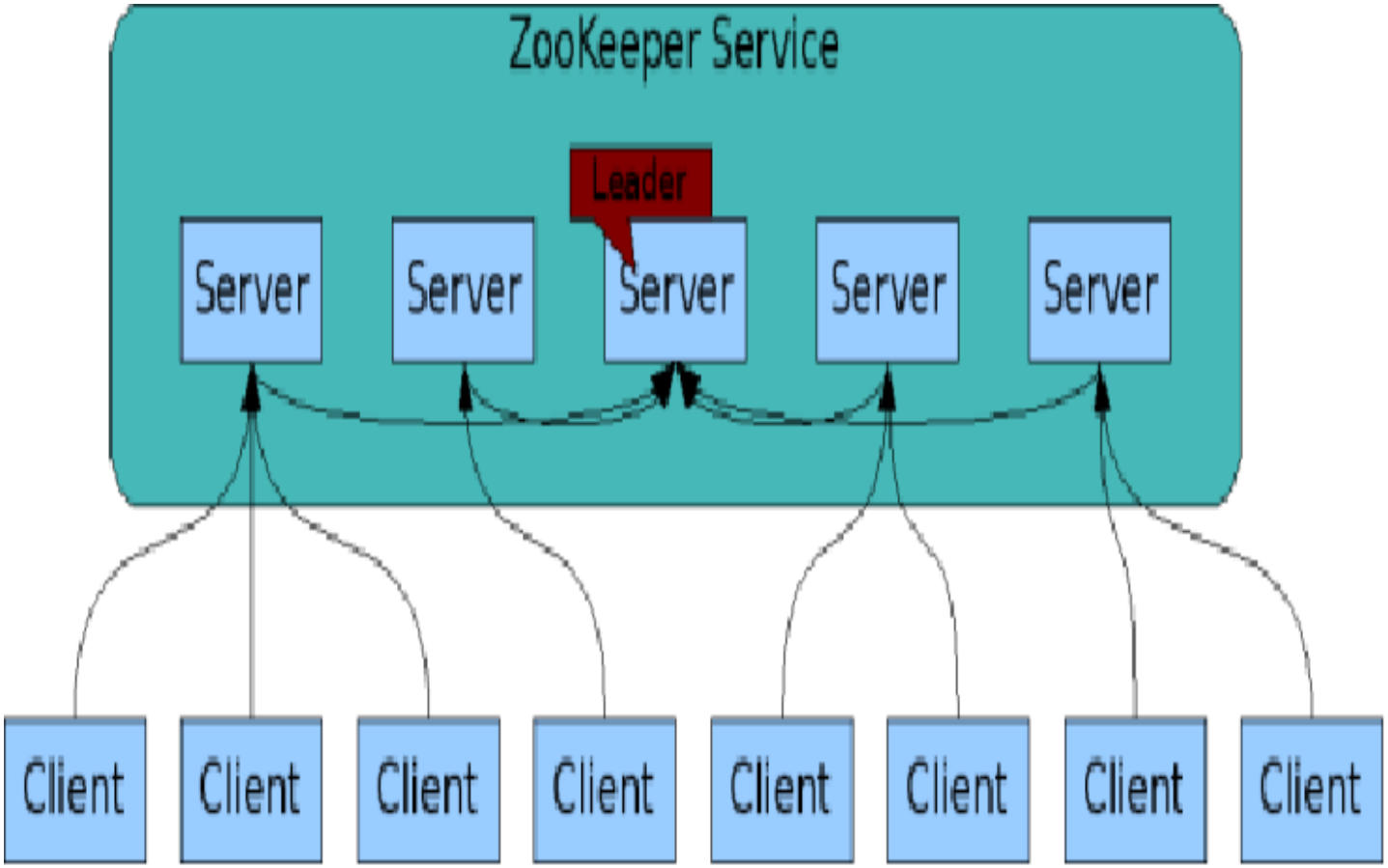
- Obtain locks on shared resources (e.g. files) to read/update the state
- E.g. Write configuration in a config file and require a lock (read/write) to read or update configuration
- Blocking - slow or faulty clients can slow down everyone
- Complicated failure semantics

Zookeeper

- Coordination kernel - Implement coordination primitives on top of Zookeeper
- Wait-free client operations
- Strong consistency (linearizability) for updates

Zookeeper Design : Znodes

- Abstraction of a set of **data nodes**, organized in a hierarchical namespace
- Example - One directory for each component of the application (client, edge, backend), storing configuration meta-data needed for startup
- Regular vs Ephemeral znodes - Regular nodes are explicitly created and destoyed, ephemeral nodes exist for a given client-server session


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

ZOOKEEPER API (1/2)

create(path, data, flags): Creates a znode with path name path, stores data[] in it, and returns the name of the new znode.

- *flags* enables a client to select the type of znode: regular, ephemeral, and set the sequential flag;

delete(path, version): Deletes the znode path if that znode is at the expected version

exists(path, watch): Returns true if the znode with path name path exists, and returns false otherwise.

- Note the *watch* flag

ZOOKEEPER API (2/2)

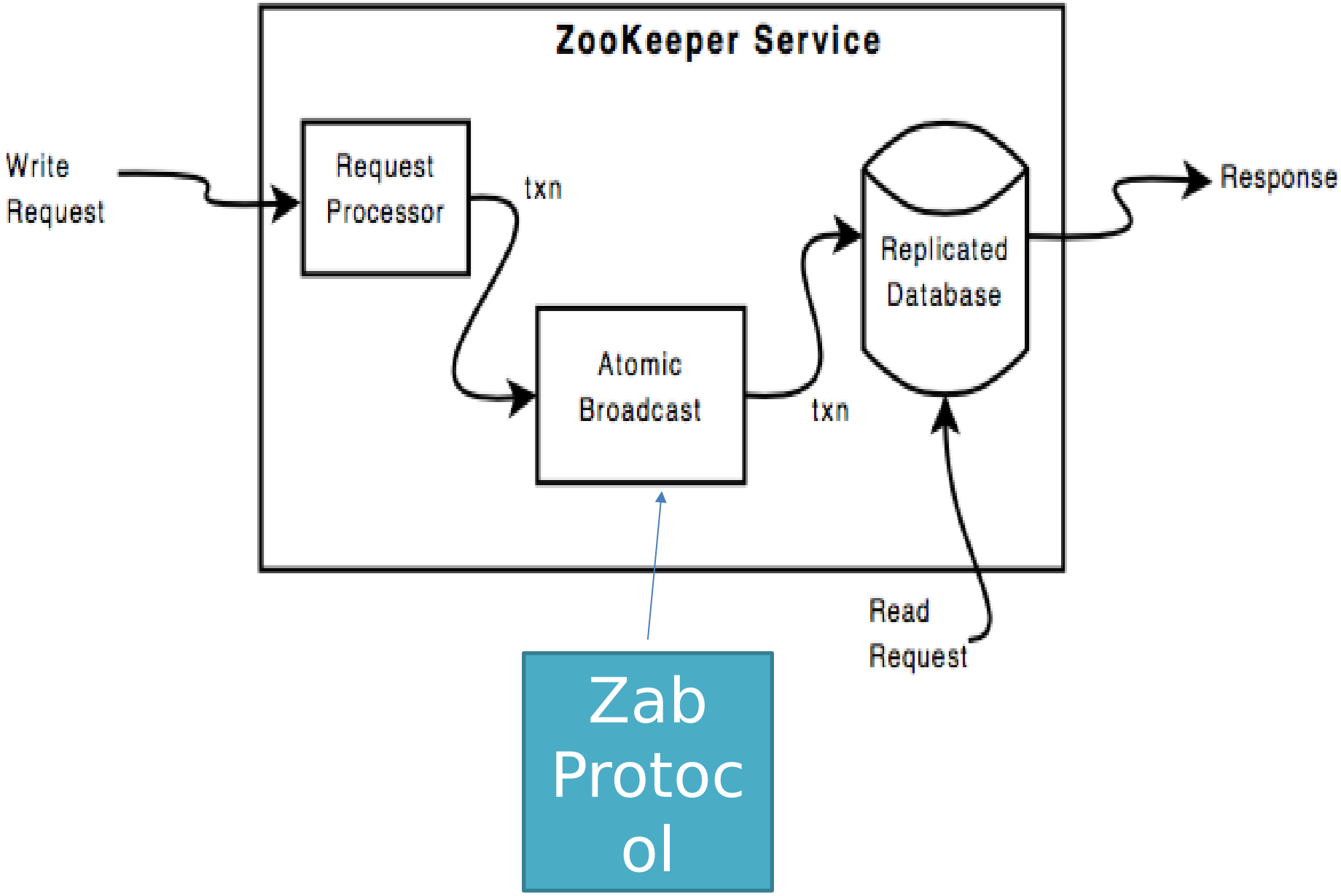
getData(path, watch): Returns the data and meta-data, such as version information, associated with the znode.
setData(path, data, version): Writes data[] to znode path if the version number is the current version of the znode
getChildren(path, watch): Returns the set of names of the children of a znode
sync(path): Waits for all updates pending at the start of the operation to propagate to the server that the client is connected to.

Extended Example 1: Membership management

- Each node joins by creating an **ephemeral** znode under *members* (with ip_addr, port etc. as meta-data)
- List of members is obtained by reading *members*
- A membership change is detected by setting a watch for the children of *members*

Extended Example 2: Large configuration changes by leader

- Leader deletes a special znode *ready*, changes configuration files, creates *ready*
- Client keeps a watch on this file. Reads the configuration only when it exists
- Possible race condition: client sees *ready* exists, proceeds to read config files while a new leader deletes it and proceeds to change the config files
- Not possible because of Zookeeper ordering guarantees


ZooKeeper Service
Write Request
Request Processor
txn
Atomic Broadcast
txn
Replicated Database
Response
Read Request
Zab Protocol

ZOOKEEPER: A SOLUTION FOR THIS ISSUE

The need in many systems is for a place to store configuration, parameters, lists of which machines are running, which nodes are "primary" or "backup", etc.

We desire a file system interface, but "strong, fault-tolerant semantics"

Zookeeper is widely used in this role. Stronger guarantees than GFS.

- Data lives in (small) files. ***Zookeeper is quite slow and not very scalable.***

SHOULD I USE ZOOKEEPER FOR EVERYTHING?

Zookeeper isn't for long-term storage, or for large objects. Put those in the GFS. Then share the URL, which is small.

Use Zookeeper for small files used to do distributed coordination, synchronization or configuration data (small objects).

Mostly, try to have Zookeeper handle "active" configuration, so it won't need to persist data to a disk, at all.

ZOOKEEPER DURABILITY LIMITATION

Zookeeper is mostly used with no real "persistency" guarantee, meaning that if we shut it down completely, it normally loses any "state"

There is a checkpointing mechanism, but not fully synchronized with file updates. Recent updates might not yet have been checkpointed.

- The developers view this as a tradeoff for high performance.
- Normally, it is configured to run once every 5s.