Coordination Services

DIDA: Class 05

Coordination services

- Assume you need to implement a fault-tolerant service with *N* replicas
- A general strategy would be to run State Machine Replication (SMR)
- For that you could use a Paxos library to enforce a total order on commands

Coordination services

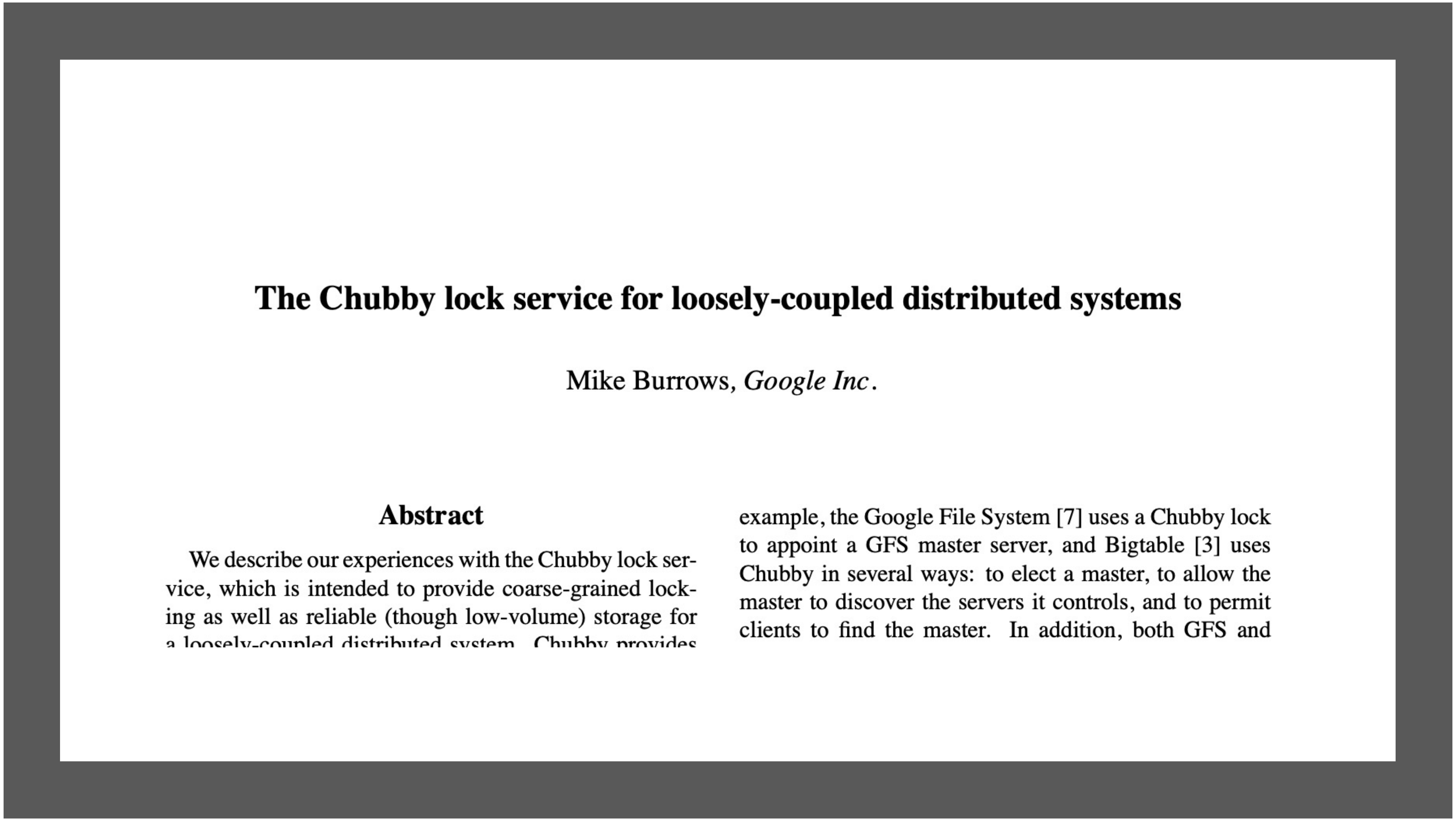
- But in some cases, you may think that using SMR is an overkill. For instance:
  - Maybe many of your commands do not need to be totally ordered
  - You just need SMR to keep a small amount of the application state, for instance some configuration parameters
  - *N* can be very large, making the use of Paxos unpractical
  - Some other reason

Coordination services

- A coordination service is a service designed to help applications that want to avoid implementing Paxos
- The coordination service must be fault-tolerant
- Thus, the coordination service is implemented using Paxos

Warning!

- To simplify things, in the course I do not discuss much the problem of having nodes crashing *and recovering*.
- In order to be able to recover in a safe way, processes need to store their state in stable storage at critical points in the execution:
  - For instance, an acceptor that accepts a value from a proposer, must store this fact in stable storage before sending “accepted”, to ensure that it will not contradict itself if it later crashes and recovers.
- Still, read system have to take this in to account and some of the bibliography for coordination services address this issue.



The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, *Google Inc*.

Abstract

We describe our experiences with the Chubby lock service, which is intended to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system. Chubby provides

example, the Google File System [7] uses a Chubby lock to appoint a GFS master server, and Bigtable [3] uses Chubby in several ways: to elect a master, to allow the master to discover the servers it controls, and to permit clients to find the master. In addition, both GFS and

Paxos Made Live - An Engineering Perspective

Tushar Chandra
Robert Griesemer
Joshua Redstone

June 20, 2007

Chubby

- Small file system and lock service built on top of Paxos
- Chubby state is kept by a set of replicas to ensure fault-tolerance
- State-updates are performed using Paxos, to ensure consistency
- Distributed applications can use Chubby to coordinate
  - For instance, to elect a primary

Chubby

- Note:
  - Chubby was developped at Google first without using Paxos
  - Replicas of Chubby maintained shared state using a third-party replicated database
  - Eventually, Google decided to build their custom, Chubby-specific, replicated database using Paxos

Main goal of Chubby

- To act as a “lock service”
- The lock service can be used by an application to elect a master
- Replicas and clients of that application can be notified when the master changes

- Based on the notion of “lease”: a period of time after which a primary must “give up” unless it is able to renew the lease

Paxos implementation

- Chubby implements Paxos with some twists:
  - When a process becomes a leader, it remains a leader for some pre-defined period of time (which is also a lease)
  - Other nodes do not attempt to become leader while the lease is valid
  - This avoids contention, i.e., scenarios where multiple Chubby replicas attempt to become leaders and prevent the progress of each other
  - Clients send requests to the leader
  - If the leader fails, after the lease expires, other replicas will try to become the leader
  - The new leader must get a majority of votes

Chubby

- Many applications can use the same Chubby service (also called a Chubby *cell* in the paper)
- Chubby internally maintains a hierarchical file system, such that it can keep the state of different files/locks (these are named *nodes*)
  - `Example: /ls/foo/wombat/pouch`
- Each Chubby node has a version number, that is incremented every time the content is updated and a lock number, that is incremented every time the associated lock moves from *free* to *held*.

Chubby

- Nodes can be *ephemeral*
  - *Ephemeral* nodes are deleted automatically if the client that has created the nodes disconnects

Locks

- Chubby supports shared and exclusive locks
- Clients that obtain an exclusive lock are guaranteed to hold the lock for a lease time known as the lock-delay
- If client holding the lock fails, the lock can be re-acquired by another client after the lease expires
- If a client holding the lock releases the lock, the other clients are notified and can immediately attempt to acquire the lock

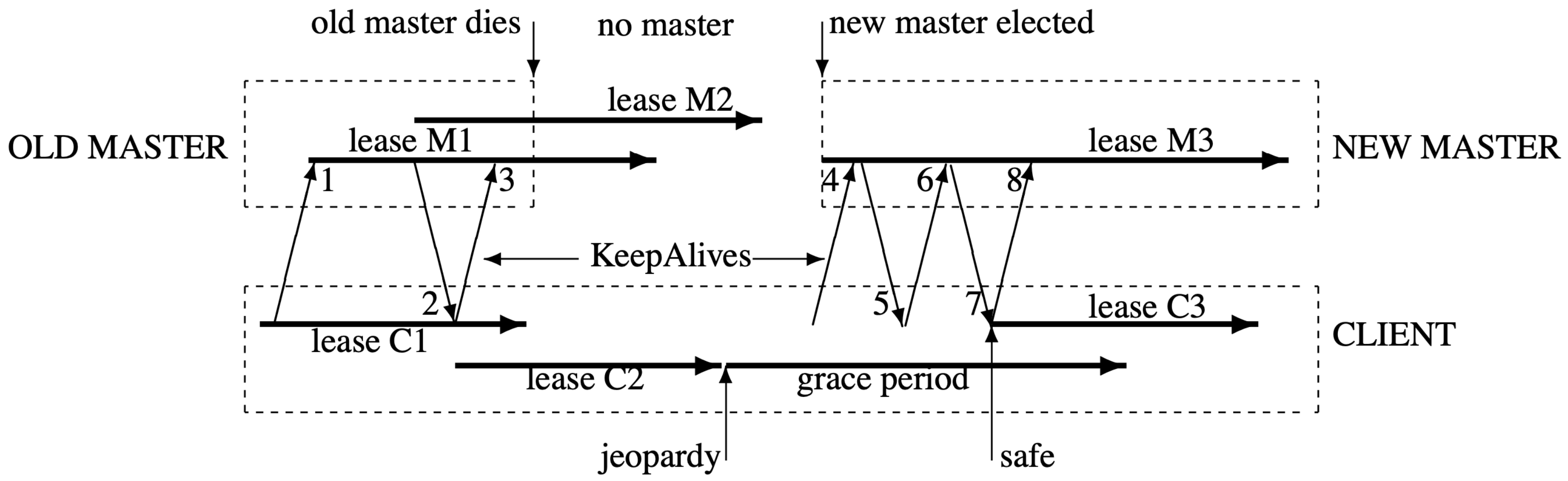
Client cache

- Clients cache the content of a node and information regarding current lock owners. This cache is also valid for some lease time
- If the content of a node is updated, the Chubby service invalidates all caches
- If a client is slow, invalidation will stall until the cache expires
- Clients mut notify the Chubby servers that they are still alive using a keepalive mechanism
  - This mechanism is used to update the lease and to receive updates

Client cache

- If a client (library) does not manage to renew its lease, it cleans the cache and enters a “grace period” where it does not accept request from the application but where it attempts to reconnect to the old master or to a new master.
- If the problem is transient, this allows the client to maintain its session.

Master fail-over



ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt and Mahadev Konar
Yahoo! Grid
{phunt,mahadev}@yahoo-inc.com

Flavio P. Junqueira and Benjamin Reed
Yahoo! Research
{fpj,breed}@yahoo-inc.com

Abstract

In this paper, we describe ZooKeeper, a service for coordinating processes of distributed applications. Since ZooKeeper is part of critical infrastructure, ZooKeeper aims to provide a simple and high performance kernel for building more complex coordination primitives at the client. It incorporates elements from group messaging,

that implement mutually exclusive access to critical resources.

One approach to coordination is to develop services for each of the different coordination needs. For example, Amazon Simple Queue Service [3] focuses specifically on queuing. Other services have been developed specifically for leader election [25] and configura-

ZooKeeper

- Can be seen as an open-source implementation of Chubby
- But with a very important difference:
  - Chubby offers consistent caching while Zookeeper does not

ZooKeeper

- Like Chubby, Zookeeper keeps (small) files replicated using a Paxos-like consensus protocol
- Clients may request to be notified when the file changes (these are called “*watches*”)
- However, this notification is perfomed asynchronously
- Thus, at a given moment, two clients may observe different version of a file
  - Due to this problem, there are open-source libraries that augment Zookeeper with a consistent caching layer (for instance, *curator* from Netflix)

ZooKeeper

- The authors of ZooKeeper “pitch” the lack of consistency and non-blocking as a “plus”:

“ZooKeeper seems to be Chubby without the lock methods, open, and close. Implementing wait-free data objects, however, differentiates ZooKeeper significantly from systems based on blocking primitives such as locks. ”

ZooKeeper

- But the introduction is somehow contradictory because they also say that they implement consensus:

“*In particular, we have found that guaranteeing both FIFO client ordering of all operations and linearizable writes enables an efficient implementation of the service and it is sufficient to implement coordination primitives of interest to our applications. In fact, we can implement consensus for any number of processes with our API, and according to the hierarchy of Herlihy, ZooKeeper implements a universal object*.”

ZooKeeper

- Because it uses consensus to establish a total order on all upadtes

“To guarantee that update operations satisfy linearizability, we implement a leader-based atomic broadcast protocol, called Zab.”

ZooKeeper

- Only updates are linerizable!
- Updates include
  - normal “writes”
  - sequential creates (where an “order-number”is automatically added by the system to the object attributes)
  - a form of “compare-and-swap”
- Reads are performed locally at each replica
  - One replica can read the new value and subsequently another replica can read the old value
  - Clients may attempy to circumvent this by doing an “empty” write before reading, called a “sync” request

ZooKeeper

- Although it is not a lock server, the authors claim that ZooKeeper can be used be used to implement locks.
- For instance, a node can create an emphereal node associated with a lock. The first client to do so becomes the lock owner. The other nodes will put a watch on the node. If the clients fails or unlocks, the node is deleted and other clients may attempt to create the node to get the lock.
- The paper describes another alternative that avoids a rush for the lock.

ZooKeeper

- Another way to maintain locks is as follows:
  - Clients that compete for the lock create an empheral file using the “sequential” feature of ZooKeeper.
  - Each of these files is assigned a sequence number
  - The client with the file with sequence number “1” becomes the lock owner
  - If that client fails, the client with file with sequence number “2” becomes the new lock owner