The Chubby lock service for loosely-coupled distributed systems

ifox小组
苏飞　赵东生　樊锴　单栋栋

Outline

- Paxos Made Simple
- The Chubby Lock Service
- YAHOO-ZooKeeper

Paxos Made Simple

苏飞
理论实验室

Contents

What
Why
How
Where

What is Paxos

- Help to implementing a fault-tolerant distributed system, provide *distributed consensus in a network of several processors*
- a family of algorithms
  - cheap Paxos, fast Paxos, generalized Paxos, Byzantine Paxos...

Why Paxos

- *Considered the most effective in the field*
- all working protocols for asynchronous consensus we have so far encountered have Paxos at their core.
- Raised from intuitive and reasonable reasoning

Where & How…

- Context
- Three roles in the Paxos protocol
  - *Proposer*
  - *Acceptor*
  - *Learner*
- Goal: consensus

How…

- The Safety properties
  - A value that has been proposed may be chosen
  - Only a single value is chosen
  - A leaner never learns that a value has been chosen unless it actually has been chosen

Reasoning

- P1: An acceptor must accept the first proposal that it receives.
- P1 + "majority" requirement →
  - An acceptor must be allowed to accept more than one proposal
- Keeping track of proposals:
  - unique numbers
  - a proposal is pairs: (counter, value)
- A value has been chosen only if:
  - A single proposal has been accepted by a majority.

Reasoning

- We can allow multiple proposals to be chosen
- But must guarantee that all chosen proposals have the same value:
- P2: If a proposal with value *v* is chosen, then every higher-numbered proposal that is chosen has value *v*.
- P2 guarantees the safety property of consensus.

Reasoning

- P2a: If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v.
- P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

Reasoning

- P2c: If a proposal (n, v) is issued, then there exists a majority S of acceptors, such that:
  - **Either** no acceptor in *S* has accepted any proposal numbered less than *n*
  - **Or** *v* is the value of the highest-numbered proposal among all proposals numbered less than *n* accepted by the acceptors in *S*.
- *P2c* → *P2b* → P2a → P2

Algorithm

- **Phase 1a: *Prepare***
  - Send Msg: Prepare n
- **Phase 1b: *Promise***
  - **For *Acceptor,* if n > previous proposal** number, send back the value.
- **Phase 2a: *Accept!***
  - **proposer fix the value**
- **Phase 2b: *Accepted***
  - ***Acceptor* fix the value, tell the learner**

Review…

- ***What***
- ***Why***
- ***How***
- ***Where***

The Chubby Lock Service（1）

10748236 赵东生
网络实验室

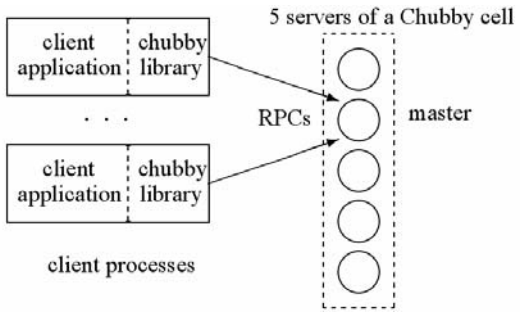
Chubby's Design

- Introduction
- System structure
- File, dirs, handles
- Locks and sequencers
- Events
- API
- Caching

Introduction

- Google Inc.
- distributed lock service
- loosely-coupled distributed system
- to synchronize clients activities
- agree on basic environment information
- reliability and availability
- used in GFS and Bigtable

System structure



Files, dirs, handles

- File interface similar to, but simpler than UNIX
  - /ls/foo/wombat/pouch
- the name space contains files and directories called nodes
- each node has various meta-data

Files, dirs, handles (cont.)

- clients open node to obtain handles
- handles ~ UNIX file descriptors
- handle include
  - Check digits – prevent client guess handle
  - Sequence number – which master generated the lock
  - Mode information – use to recreate the lock state when the master changes
- other operation needs handle

Locks and sequencers

- Chubby uses a file or directory to act as a reader-writer lock
- One client may hold the lock in exclusive (writer) mode.
- Any number of clients hold the lock in shared (reader) mode.
- Locks are advisory - only conflict with other attempts to acquire the same lock

Locks and sequencers

- Locking in Distributed is complex
- Virtual synchrony or virtual time – costly
- Chubby - Only interactions which use locks are numbered
- Sequencer - a byte string describing the state of the lock after acquisition
- A lock requests a sequencer
- A client passes the sequencer to the server for protected progress

Events

- Client can subscribe to some events when a handle is created.
- Delivered after the corresponding action has taken place
- Events include
  - file contents modified
  - child node added, removed, or modified
  - chubby master failed over
  - handle has become invalida
  - lock acquired by others
  - conflicting lock requested from another client

API

- Handled are created by Open()
- destroyed by Close()
- GetContentsAndStat(), GetStat(), ReadDir()
- SetContents(), SetACL()
- Delete()
- Acquire(), TryAcquire(), Release()
- GetSequencer(), SetSequencer(), CheckSequencer()

Caching

- Client caches file data and meta-data reduce read traffic
- Handle and lock can also be cached
- Master keeps a list of which clients might be caching
- Invalidations keep consistent
- Clients see consistent data or error

The Chubby Lock Service（2）

00448161 樊锴
网络实验室

Client Sessions

- **Sessions** maintained between client and server
  - Keep-alive messages required to maintain session every few seconds
- If session is lost, server releases any client-held handles.
- What if master is late with next keep-alive?
  - Client has its own (longer) timeout to detect server failure

Master Failure

- If client does not hear back about keep-alive in *local lease timeout*, session is **in jeopardy**
  - Clear local cache
  - Wait for “grace period” (about 45 seconds)
  - Continue attempt to contact master
- Successful attempt => ok; jeopardy over
- Failed attempt => session assumed lost

Master Failure

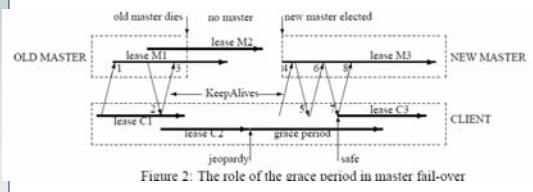


Figure 2: The role of the grace period in master fail-over

Master Failure (2)

- If replicas lose contact with master, they wait for grace period (shorter: 4—6 secs)
- On timeout, hold new election

Reliability

- Started out using replicated Berkeley DB
- Now uses custom write-thru logging DB
- Entire database periodically sent to GFS
  - In a different data center
- Chubby replicas span multiple racks

Scalability

- Arbitrary number of Chubby cells.
- System increases lease times from 12 sec up to 60 sec under heavy load
- Chubby clients cache file data, meta-data, the absence of files, and open handles.
- Data is small – all held in RAM (as well as disk)
- * Use proxies to save KeepAlive and reads traffic.
- * Use partitioning

time since last fail-over	18 days
fail-over duration	14s
active clients (direct)	22k
additional proxied clients	32k
files open	12k
naming-related	60%
client-is-caching-file entries	230k
distinct files cached	24k
names negatively cached	32k
exclusive locks	1k
shared locks	0
stored directories	8k
ephemeral	0.1%
stored files	22k
0-1k bytes	90%
1k-10k bytes	10%
> 10k bytes	0.2%
naming-related	46%
mirrored ACLs & config info	27%
GFS and Bigtable meta-data	11%
ephemeral	3%
RPC rate	1-2k/s
KeepAlive	93%
GetStat	2%
Open	1%
CreateSession	1%
GetContentsAndStat	0.4%
SetContents	680ppm
Acquire	31ppm

Use as a name service

- Example: 3000 clients, TTL = 60s
- Use DNS:
  3000 * 3000 / 60 = 150 000 lookups per sec
- Use Chubby and sessions:
  3000 * 3000 in startup
  3000 * 5 / 60 = 250 KeepAlives per sec (*)
  Use events when change happens
- 90K+ clients communicate with a single Chubby master (2 CPUs)

YAHOO-ZooKeeper

10748200 单栋栋
网络实验室

ZooKeeper

- A highly available, scalable, distributed, configuration, consensus, group membership, leader election, naming, and coordination service

Motivations

- Most distributed applications needed a master, coordinator, controller to manage the sub processes of the applications
- Generally these control programs are specific to applications and thus represent a recurring development cost for each distributed application
- each control program is rewritten it doesn't get the investment of development time to become truly robust, making it an unreliable single point of failure.

Observation

- Distributed systems need coordination
- Programmers can't use locks correctly
  - –distributed deadlocks are the worst!
- Message based coordination can be hard to use in some applications

Wishes

- Simple, robust, good performance
- Tuned for read dominant workloads
- Familiar models and interface
- Wait-free
- Need to be able to wait efficiently

What "works"

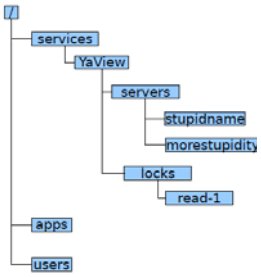
- 1)Programmers use shared file systems
  - Programmers are comfortable with file API
  - file servers are generic infrastructure components
  - It mostly works
- 2)File API and servers lack some needed semantics
  - Reasonable handling of concurrent writes
  - Change notifications
- 3)Design point: start with a file system API model and strip out what is not needed
  - Don't need:
    - Partial reads & writes
    - Rename

What we do need

- Ordered updates with strong persistence guarantees
- Conditional updates
- Watches for data changes
- Ephemeral nodes
- Generated file names

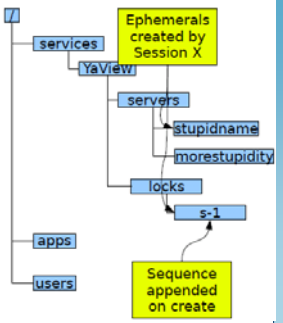
Data model

- Hiearchical name space
- Each znode has data and children
- Nodes maintain a stat structure
- Clients can set *watches* on nodes
- Data is read and written in its entirety
- has the notion of ephemeral nodes



Create Flags

- 1)Ephemeral: the znode will be deleted when the session that created it times out or it is explicitly deleted
- 2)Sequence: the the path name will have a monotonically increasing counter relative to the parent appended



ZooKeeper Guarantees

- Clients will never detect old data.
- Clients will get notified of a change to data they are watching within a bounded period of time.
- All requests from a client will be processed in order.
- All results received by a client will be consistent with results received by all other clients.

ZooKeeper API

- All the operate on path
  - String create(path, data, acl, flags)
  - void delete(path, expectedVersion)
  - Stat setData(path, data, expectedVersion)
  - (data, Stat) getData(path, watch)
  - Stat exists(path, watch)
  - String[] getChildren(path, watch)
  - void sync(path)
  - Stat setACL(path, acl, expectedVersion)
  - (acl, Stat) getACL(path)

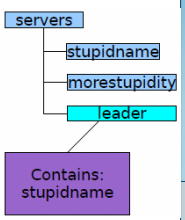
Add watch in there operations

Chubby API

- Only open() takes a node name,all others operate on handles
- Handle only created by open(),closed by close()
  - handle Open(path,...)
  - Close(handle,...)
  - GetContentsAndStat(handle,...), GetStat(handle), ReadDir(handle,...)
  - SetContents(handle), SetACL(handle,...)
  - Delete(path)
  - Acquire(), TryAcquire(), Release()
  - GetSequencer(), SetSequencer(), CheckSequencer()

leader Election

- 1)getData("/servers/leader", true)
- 2)if successful follow the leader described in the data and exit
- 3)create("…/servers/leader", hostname, EPHEMERAL)
- 4)if successful lead and exit
- 5)goto step 1

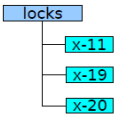


If a watch is triggered for "…/servers/leader", followers will restart the leader election process

Locks

1)id = create("…/locks/x-", SEQUENCE|EPHEMERAL)
2)getChildren("…/locks"/, false)
3)if id is the 1st child, exit
4)exists(name of last child before id, true)
5)if does not exist, goto 2)
6)wait for event
7)goto 2)

unlock,just delete it



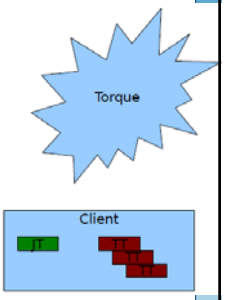
Each znode watches one other. No herd effect.

Shared Locks

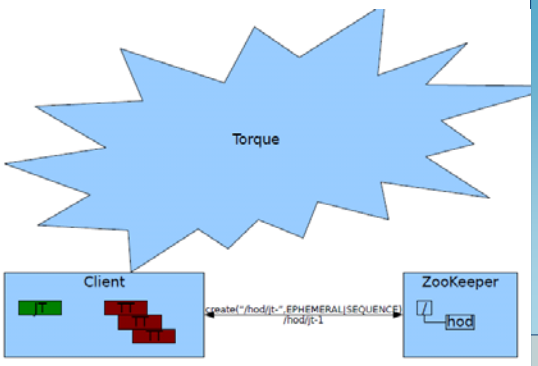
1)id = create("…/locks/s-", SEQUENCE|EPHEMERAL)
2)getChildren("…/locks"/, false)
3)if no children that start with x- before id, exit
4)exists(name of the last x- before id, true)
5)if does not exist, goto 2)
6)wait for event
7)goto 2)

HOD

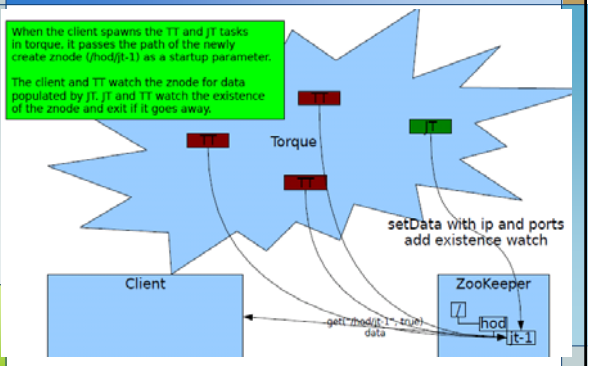
- 1)A client submits a request to start jobtracker and a set of tasktrackers to torque
- 2)The ip address and the ports that the jobtracker will bind to is not known apriori
- 3)The tasktrackers need to find the jobtracker
- 4)The client needs to find the jobtracker



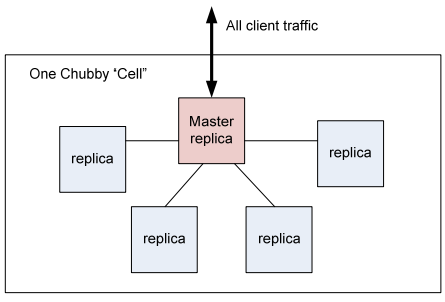
HOD with ZooKeeper



HOD with ZooKeeper



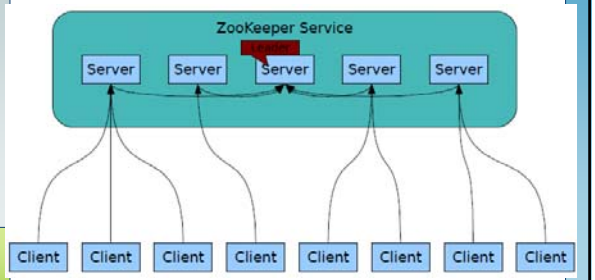
Chubby Topology



ZooKeeper Servers

- All servers store a copy of the data
- A leader is elected at startup
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change

ZooKeeper Servers



- Clients only connect to a single server.
  - The client maintains a TCP connection
  - Use this to send requests, gets responses, gets watch events, and sends heart beats.
- break TCP connection to the server
  - the client will connect to a different server.
- client first connects to the service
  - the first server setup a session for the client.
- client needs to connect to another server
  - session will be reestablished by new server.

ZooKeeper Sessions

- client first connect service, session create
  - Session id 64-bit number,timeout(2~60s) .
- server creates a password for the session
  - Use this any server can validate.
- client sends password with the session id whenever it reestablishes the session with a new server.
- Client sent requests to keep session alive
- Time out the session
  - client send a PING request to keep the session alive
  - PING request allow client and server to verify each other still alive.

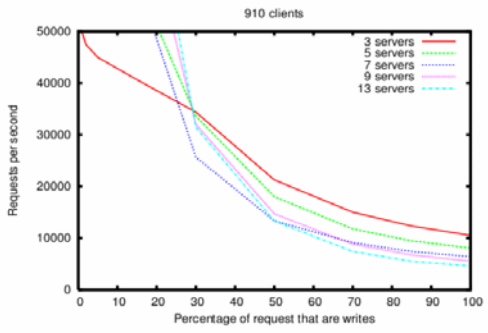
ZooKeeper Watches

- Watches are one time triggers
  - if you get a watch event and you want to get notified of future changes, you must set another watch.
- latency between getting the event and sending a new request to get a watch
  - cannot reliably see every change that happens to a node in ZooKeeper. Be prepared to handle the case where the znode changes multiple times between getting the event and setting the watch again.
- disconnect from a server
  - (for example, when the server fails), all of the watches you have registered are lost, so treat this case as if all your watches were triggered.

Performance at Extremes

Servers	1% Writes	100% Writes
13	265115	4592
9	195178	5550
7	147810	6371
5	75308	8048
3	49827	10519

Performance



difference

- All API take a path (no file handles and no open and close)
- Quorum based updates with reads from any servers (you may get old data – if you call sync first, the next read will be current as of the point of time when the sync was run at the oldest. All updates flow through an elected leader (re-elected on failure).
- Written in Java
  - have Java and C interfaces

Status

- Started oct/2006. Prototyped fall 2006. Initial implementation March 2007. Open sourced in Nov 2007.
- A Paxos variant (modified multi-paxos)
- Zookeeper is a software offering in Yahoo whereas Hadoop

参考文献

- Paxos
  - [Chandra, et al.,2007] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. Portland, Oregon, USA: ACM, 2007, pp. 398-407. http://labs.google.com/papers/paxos_made_live.html
  - [Lamport,2001] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, pp. 18-25, 2001.
  - http://en.wikipedia.org/wiki/Paxos_algorithm
- [Burrows,2006] M. Burrows, "The Chubby Lock Service for Loosely-Coupled Distributed Systems," presented at OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, 2006.
- Zookeeper
  - http://zookeeper.wiki.sourceforge.net
  - http://developer.yahoo.com/blogs/hadoop/2008/03/intro-to-zookeeper-video.html

Thank You !