ZooKeeper

—— by zyh

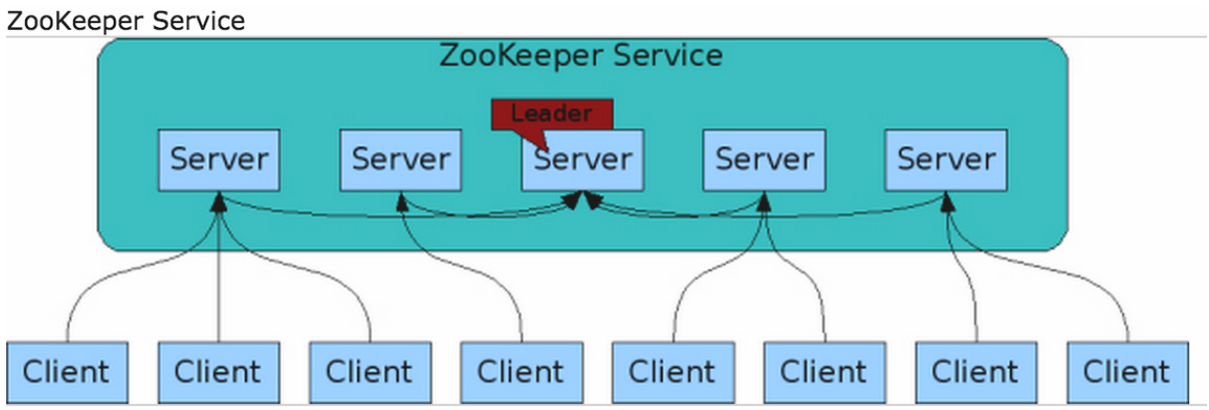
Strategy

Distributed Partial Error Coordination

Configuration Mutex Log

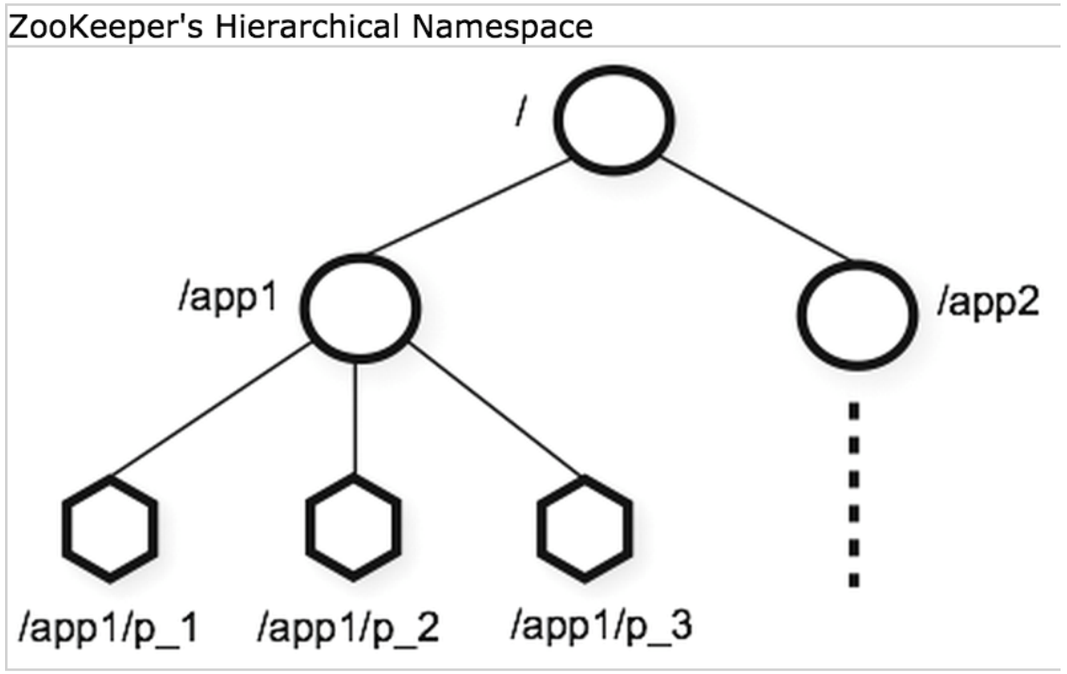
A Distributed Coordination Service

- Design Goals
  - Simple
  - replicated
  - ordered
  - fast



Model

- Similar File System
- Hierarchical namespace
- Nodes



Nodes and ephemeral nodes

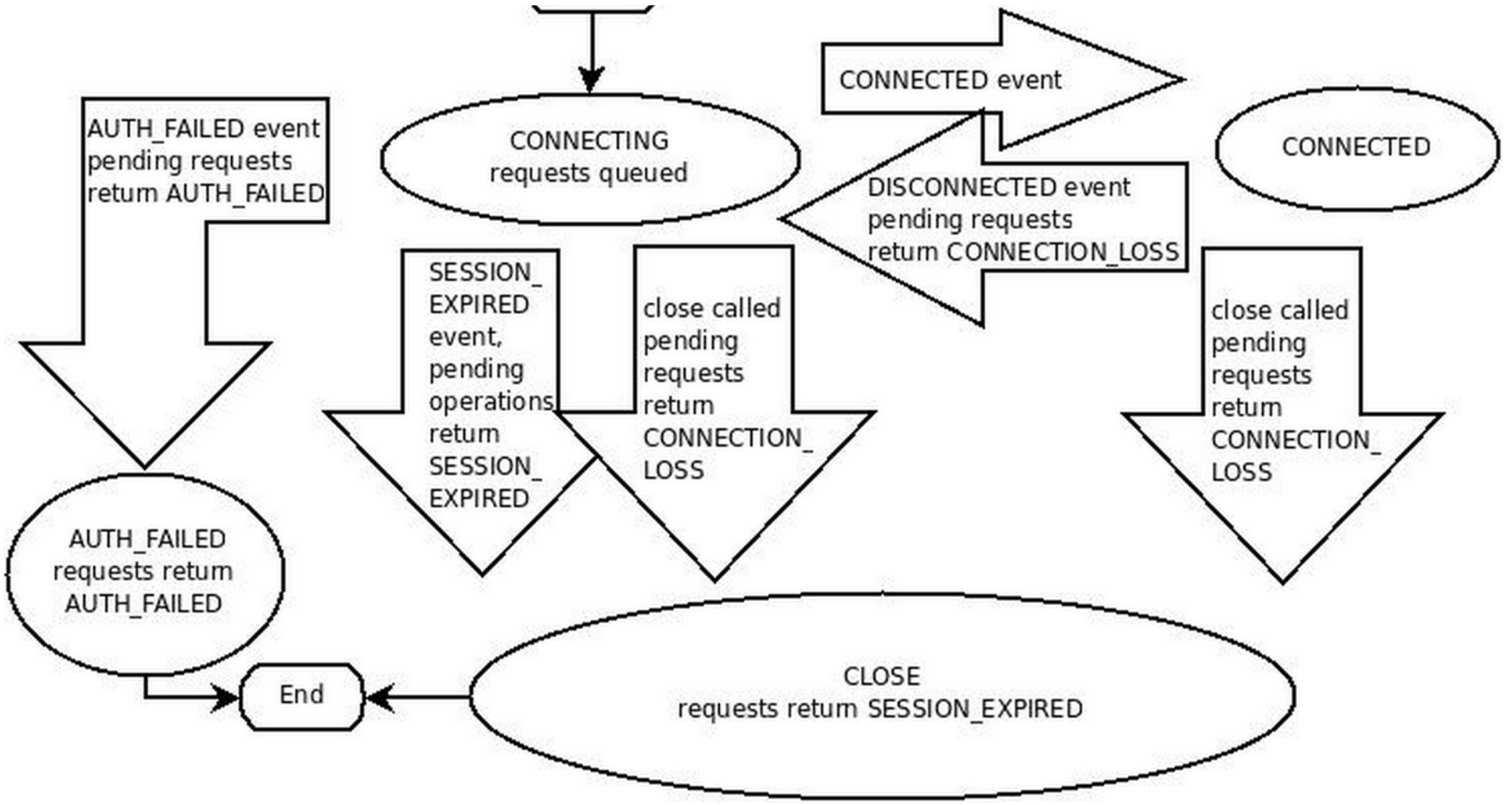
- Data Storage
- Access Control List (ACL)
- Version
- Timestamps

```
[zk: 127.0.0.1:2181(CONNECTED) 6] get /zoo
root_zoo
cZxid = 0x4
ctime = Wed Dec 24 11:03:50 CST 2014
mZxid = 0x4
mtime = Wed Dec 24 11:03:50 CST 2014
pZxid = 0x7
cversion = 1
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 1
```

Operations

- create
- creates a node at a location in the tree
- delete
- deletes a node
- exists
- tests if a node exists at a location
- get data
- reads the data from a node
- set data
- writes data to a node
- get children
- retrieves a list of children of a node

Sessions


CONNECTED event
AUTH_FAILED event
pending requests
return AUTH_FAILED
CONNECTING
requests queued
CONNECTED
DISCONNECTED event
pending requests
return CONNECTION_LOSS
SESSION_
EXPIRED
event,
pending
operations
return
SESSION_
EXPIRED
close called
pending
requests
return
CONNECTION_
LOSS
close called
pending
requests
return
CONNECTION_
LOSS
AUTH_FAILED
requests return
AUTH_FAILED
End
CLOSE
requests return SESSION_EXPIRED

Watches

ZooKeeper's definition of a watch: a watch event is one-time trigger, sent to the client that set the watch, which occurs when the data for which the watch was set changes.

- One-time trigger
- Sent to the client
- The data for which the watch was set

触发观察触发器的操作

设置观察的操作	create		delete		setData
	znode	该 znode 的子节点	znode	该 znode 的子节点	
Exists	NodeCreated		NodeDeleted		NodeDataChanged
getData			NodeDeleted		NodeDataChanged
getChildren		NodeChildrenChanged	NodeDeleted	NodeChildrenChanged	

Guarantees about Watches

- disconnect and reconnect
- order

ACL

- Similar to UNIX file access permissions
- ZooKeeper does not have a notion of an owner of a anode
- Not recursive
- Pluggable authentication schemes

Builtin ACL Schemes: world, auth, digest, ip

For example, the pair (ip:19.22.0.0/16, READ) gives the READ permission to any clients with an IP address that starts with 19.22.

ACL Permission

- CREATE: you can create a child node
- READ: you can get data from a node and list its children.
- WRITE: you can set data for a node
- DELETE: you can delete a child node
- ADMIN: you can set permissions

Recipes and Solutions

- Barriers
- Queues
- Locks
- Two-phased Commit
- Leader Election

Enter	Leave
1. Create a name *n* = *b*+"/"+*p* 2. Set watch: **exists(*b* + "/ready", true)** 3. Create child: **create(*n*, EPHEMERAL)** 4. **L = getChildren(b, false)** 5. if fewer children in L than *x*, wait for watch event 6. else **create(b + "/ready", REGULAR)**	1. **L = getChildren(b, false)** 2. if no children, exit 3. if *p* is only process node in L, delete(n) and exit 4. if *p* is the lowest process node in L, wait on highest process node in P 5. else **delete(*n*)** if still exists and wait on lowest process node in L 6. goto 1

Obtaining a read lock:	Obtaining a write lock:
1. Call **create( )** to create a node with pathname "`_locknode_/read-`". This is the lock node use later in the protocol. Make sure to set both the *sequence* and *ephemeral* flags. 2. Call **getChildren( )** on the lock node *without* setting the *watch* flag - this is important, as it avoids the herd effect. 3. If there are no children with a pathname starting with "`write-`" and having a lower sequence number than the node created in step **1**, the client has the lock and can exit the protocol. 4. Otherwise, call **exists( )**, with *watch* flag, set on the node in lock directory with pathname staring with "`write-`" having the next lowest sequence number. 5. If **exists( )** returns *false*, goto step **2**. 6. Otherwise, wait for a notification for the pathname from the previous step before going to step **2**	1. Call **create( )** to create a node with pathname "`_locknode_/write-`". This is the lock node spoken of later in the protocol. Make sure to set both *sequence* and *ephemeral* flags. 2. Call **getChildren( )** on the lock node *without* setting the *watch* flag - this is important, as it avoids the herd effect. 3. If there are no children with a lower sequence number than the node created in step **1**, the client has the lock and the client exits the protocol. 4. Call **exists( ),** with *watch* flag set, on the node with the pathname that has the next lowest sequence number. 5. If **exists( )** returns *false*, goto step **2**. Otherwise, wait for a notification for the pathname from the previous step before going to step **2**.

Thanks.

Random Next…