Imperial College London

LSDS Large-Scale Data & Systems Group

ZooKeeper: Wait-Free Coordination for Internet-Scale Systems

Peter Pietzuch
prp@doc.ic.ac.uk

Department of Computing
Imperial College London
http://lsds.doc.ic.ac.uk

Based on slides by Patrick Hunt and Mahadev Konar

Autumn 2018

What is Coordination?

Group membership

Leader election

Dynamic Configuration

Status monitoring

Queuing

Critical sections

What is ZooKeeper?

Highly available, scalable, distributed, configuration, consensus, group membership, leader election, naming, and coordination service

Difficulty of implementing these kinds of services reliably

- Brittle in the presence of change
- Difficult to manage
- Different implementations lead to management complexity when applications are deployed

Who is using ZooKeeper?

- Hbase
- Solr
- Digg
- LinkedIn
- Many others

ZooKeeper Contributions

Coordination kernel
- Wait-free coordination

Coordination recipes
- Build higher primitives

Experience with coordination
- Some application use ZooKeeper

ZooKeeper Properties

File API without partial reads/writes

No renames

Ordered updates and strong persistence guarantees

Conditional updates (version)

Watches for data changes

Ephemeral nodes

Generated file names

ZooKeeper Guarantees

Linearisable writes

- Writes serialisable + respect precedence

FIFO client order

1. Clients never detect old data
2. Clients get notified of change to watched data within bounded time
3. All requests from client processed in order
4. All results received by client consistent with results received by other clients

Zookeeper Model I

Znode

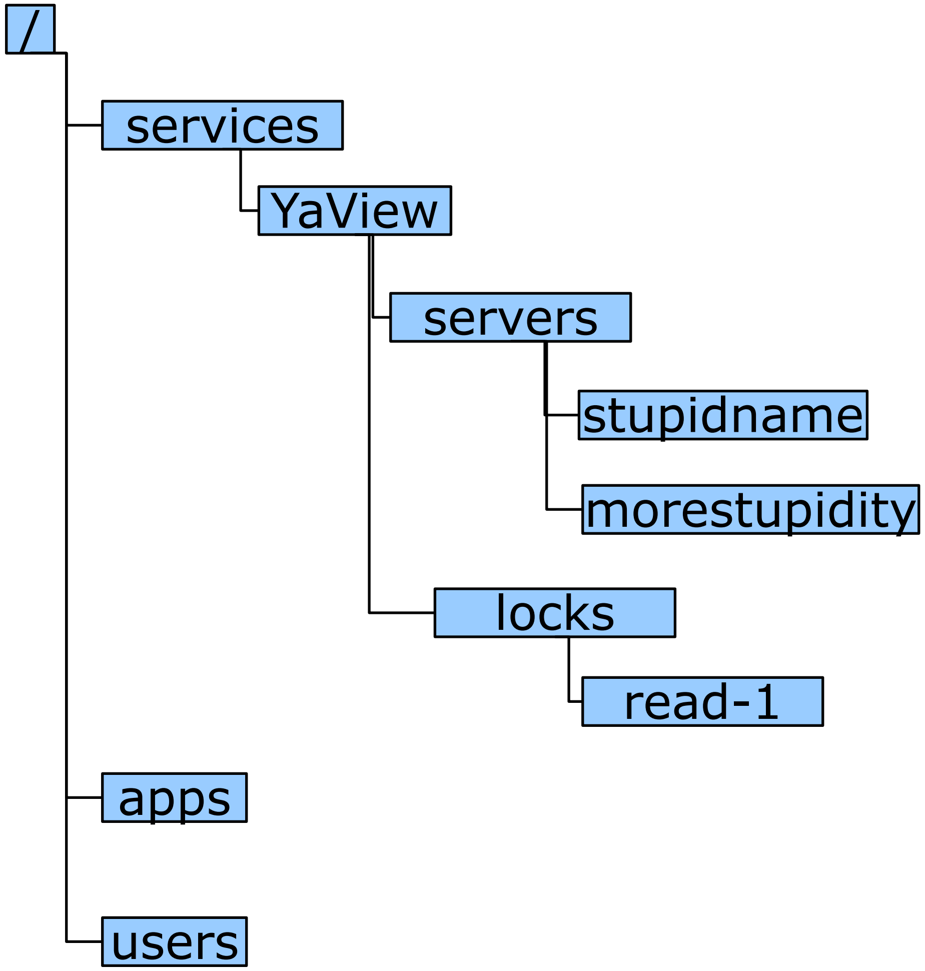
- In-memory data node in ZooKeeper data
- Hierarchical namespace
- UNIX-like notation for path
- Read/written atomically

Types of Znode

- Regular
- Ephemeral

Flags of Znode

- Sequential flag



ZooKeeper Model II

Watch mechanism

- Get notification
- One time triggers

Other properties of Znode

- Not designed for data storage, only meta-data or configuration
- Can store information such as timestamp version

Session

- Connection to server from client is a session
- Timeout mechanism

ZooKeeper API

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)

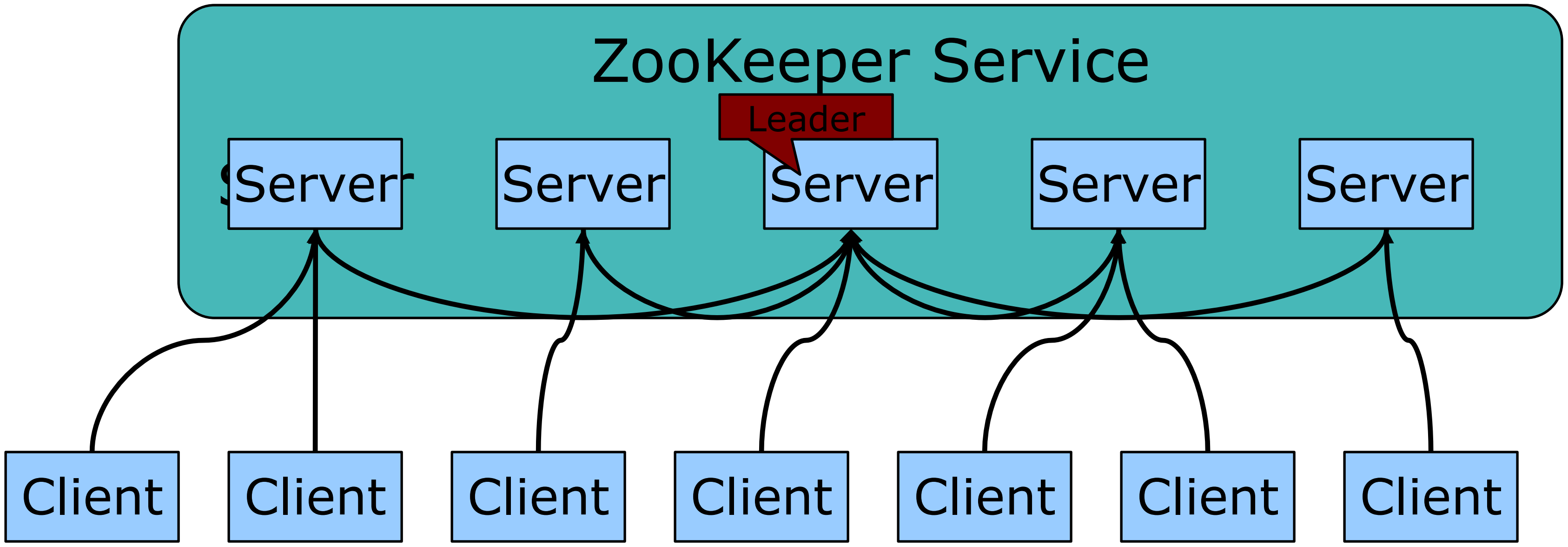
ZooKeeper Design

All servers store copy of data in memory

Leader elected at start-up

Followers service clients, all updates go through leader

Update responses sent when majority of servers persisted change



Zookeeper Use Cases

Leader Election

Group Membership

Work Queues

Configuration Management

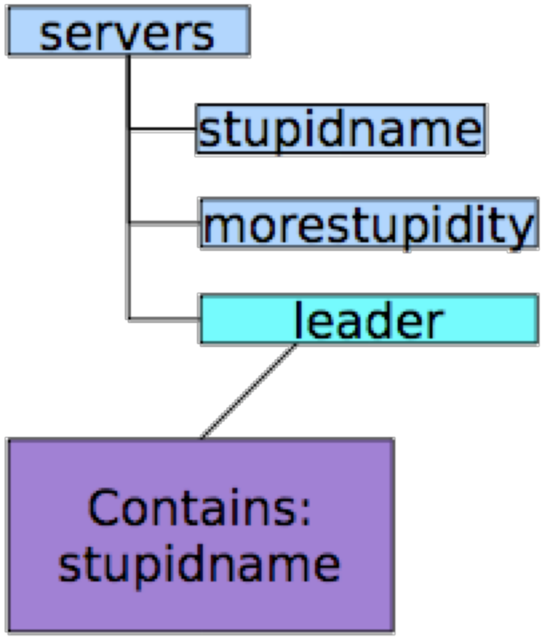
Cluster Management

Load Balancing

Sharding

Leader Election

1. getdata("/servers/leader", true)
2. if successful follow leader and exit
3. create("/servers/leader", hostname, EPHEMERAL)
4. if successful lead and exit
5. goto step 1



Leader Election in Python

```
handle = zookeeper.init("localhost:2181", my_connection_watcher, 10000, 0)

(data, stat) = zookeeper.get(handle, "/app/leader", True);

if (stat == None)
  path = zookeeper.create(handle, "/app/leader",  hostname:info, [ZOO_OPEN_ACL_UNSAFE],
                            zookeeper.EPHEMERAL)

                  if (path == None)
                     (data, stat) = zookeeper.get(handle, "/app/leader", True)
                             # someone  else is the leader
                             # parse the string path that contains the leader address

                  else

                            # we are the leader continue leading

else

        #someone else is the leader
        #parse the string path that contains the leader address
```

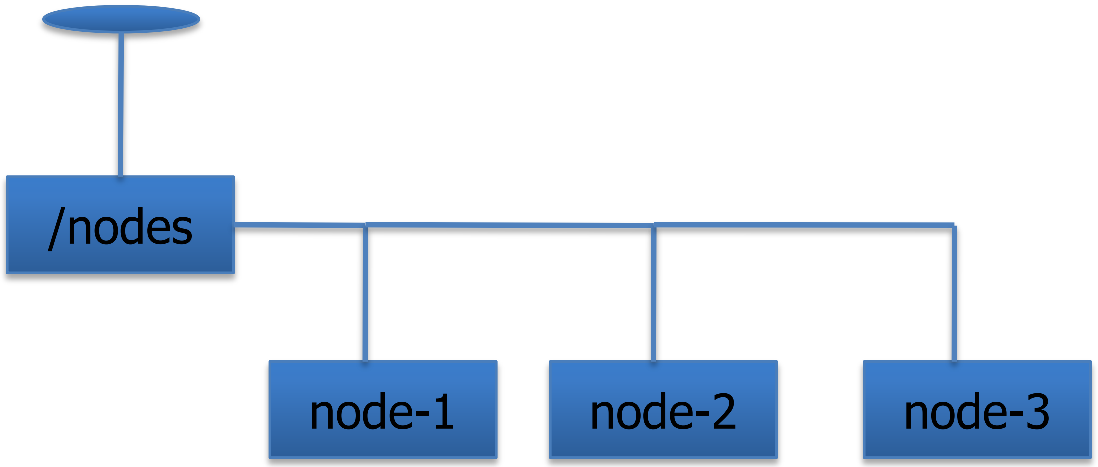
Cluster Management

Monitoring process:

1. Watch on /nodes

2. On watch trigger do
   getChildren(/nodes, true)

3. Track which nodes have gone away

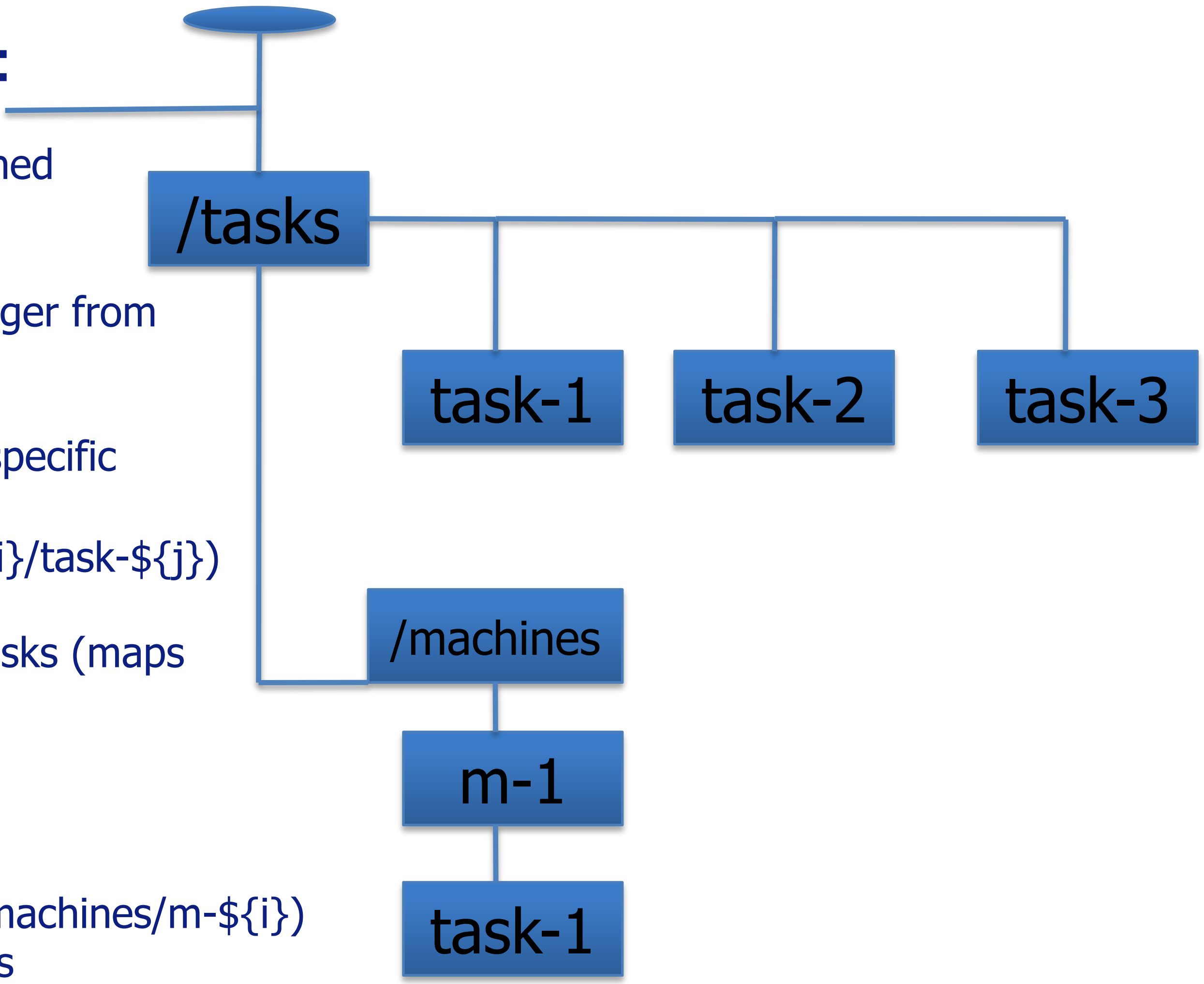


Each node:

1. Create /nodes/node-${i} as ephemeral nodes

2. Keep updating /nodes/node-${i} periodically for node status changes (status updates could be load/iostat/cpu/others)

Work Queues



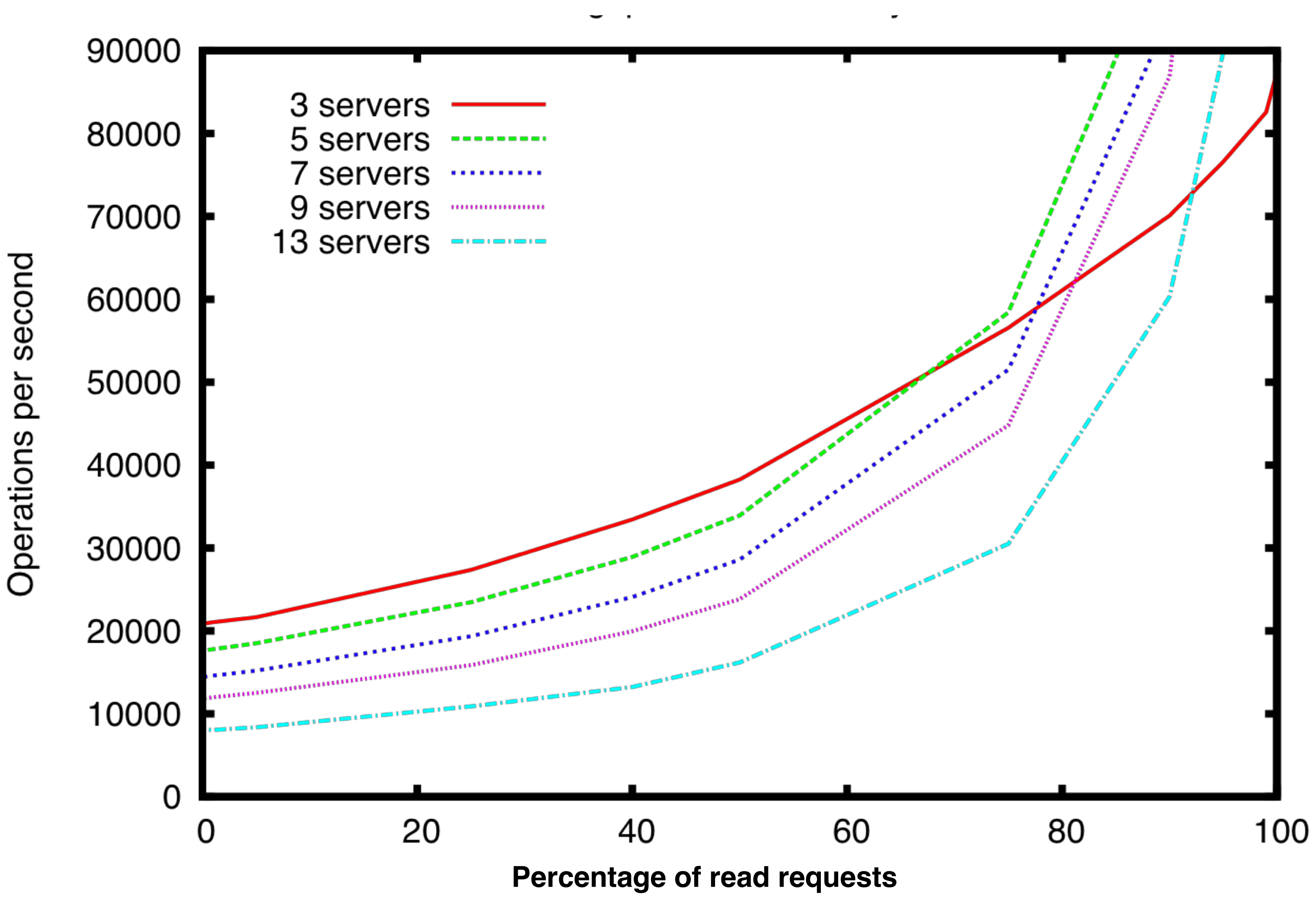
Monitoring process:

1. Watch /tasks for published tasks
2. Pick tasks on watch trigger from /tasks
3. Assign it to a machine specific queue by creating create(/machines/m-${i}/task-${j})
4. Watch for deletion of tasks (maps to task completion)

Machine process:

1. Machines watch for /(/machines/m-${i}) for any creation of tasks
2. After executing task-${i} delete task-${i} from /tasks and /m-${i}

Performance Evaluation: Throughput


3 servers
5 servers
7 servers
9 servers
13 servers
Operations per second
Percentage of read requests