ZooKeeper:
Wait-free coordination for Internet-scale systems

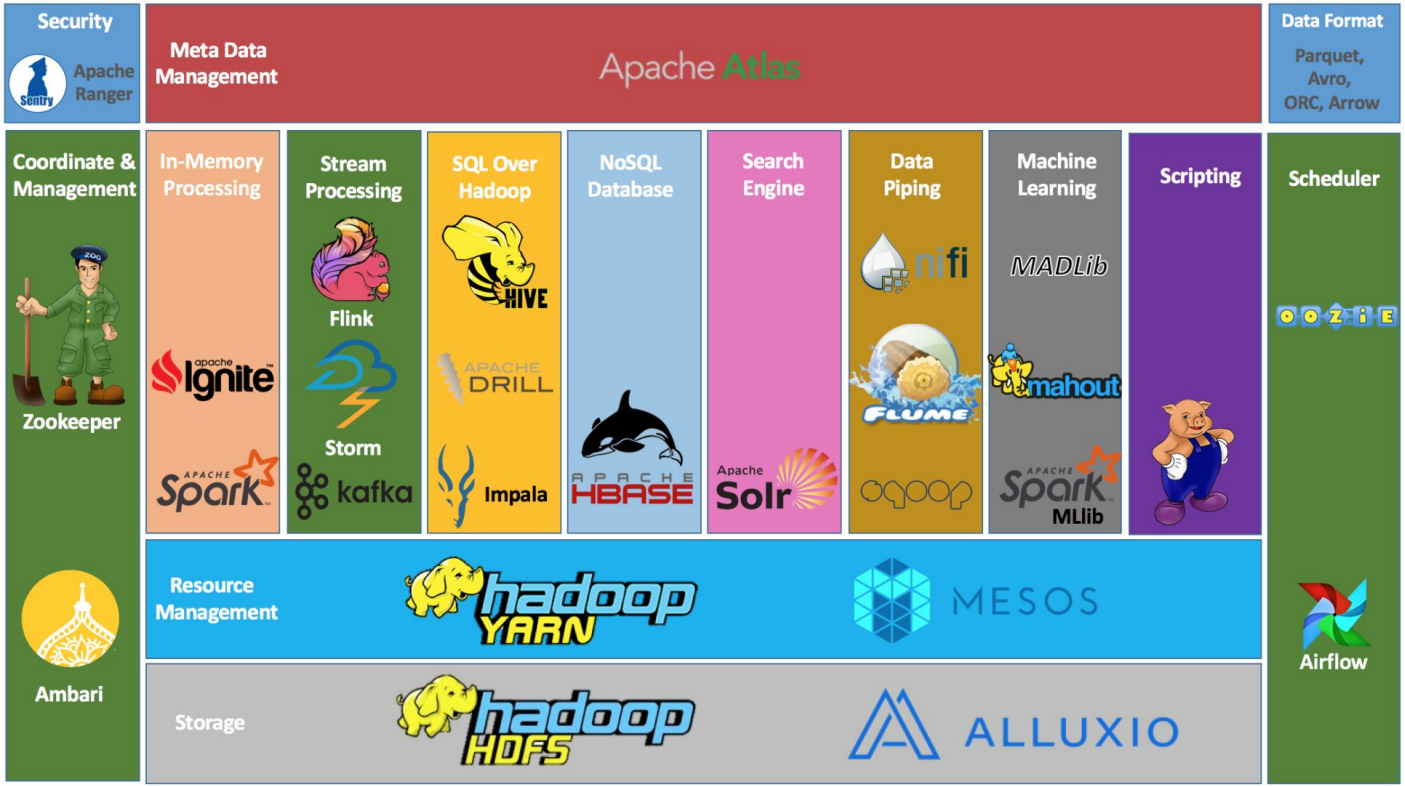
Paper Presentation for EECS 591

Jiongsheng Cai

Nov 6, 2019

“ZooKeeper”?

ZooKeeper


Security
Apache Ranger
Meta Data Management
Apache Atlas
Data Format
Parquet, Avro, ORC, Arrow
Coordinate & Management
Zookeeper
Ambari
In-Memory Processing
apache Ignite
Spark
Stream Processing
Flink
Storm
kafka
SQL Over Hadoop
HIVE
APACHE DRILL
Impala
NoSQL Database
APACHE HBASE
Search Engine
Apache Solr
Data Piping
nifi
FLUME
sqoop
Machine Learning
MADLib
mahout
Spark MLlib
Scripting
Scheduler
OOZIE
Airflow
Resource Management
hadoop YARN
MESOS
Storage
hadoop HDFS
ALLUXIO

ZooKeeper

- Developed at Yahoo! Research
- Started as sub-project of Hadoop, now a top-level Apache project
- Development is driven by application needs

Motivation

- In the past: a single program running on a single computer with a single CPU
- Today: **distributed applications** consist of independent programs running on a changing set of computers

Developers have to deal with **coordination logic** and **application logic** at the same time!

Question: How do you elect the leader?

应用程序逻辑

application logic

A program that crawls the Web

网络请求

one machine (the leader) should coordinate the effort

主节点应该协调负荷

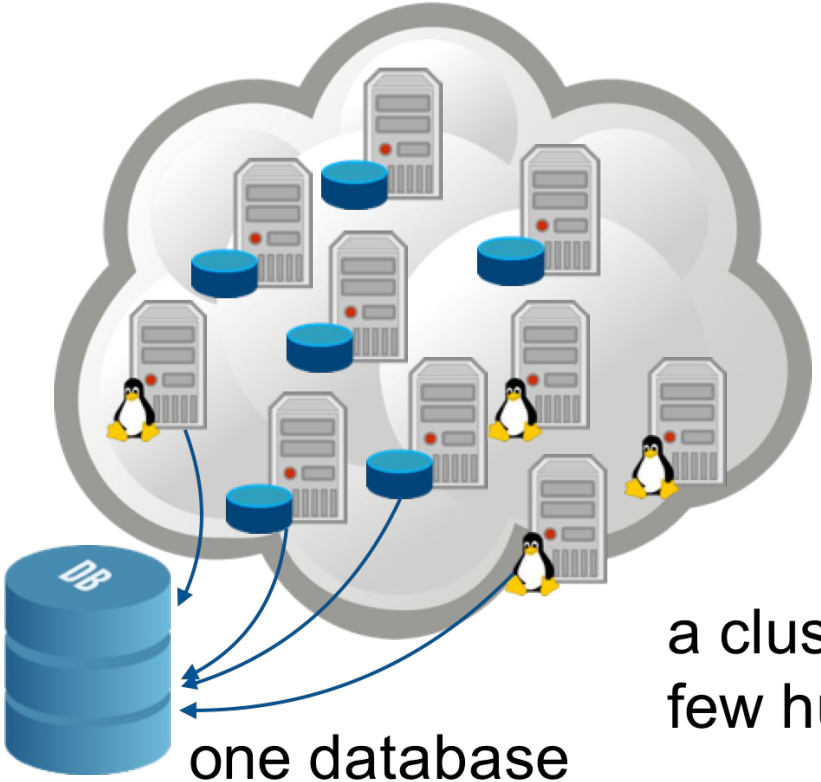
coordination logic

协调逻辑

一个有若干机器的集群

a cluster with a few hundred machines

Question: How do you lock a service?



application logic

A program that crawls the Web

请求处理的数据存储在数据库中：谁访问、何时访问

The progress of the crawl is stored in a DB: who accesses what & when?

coordination logic

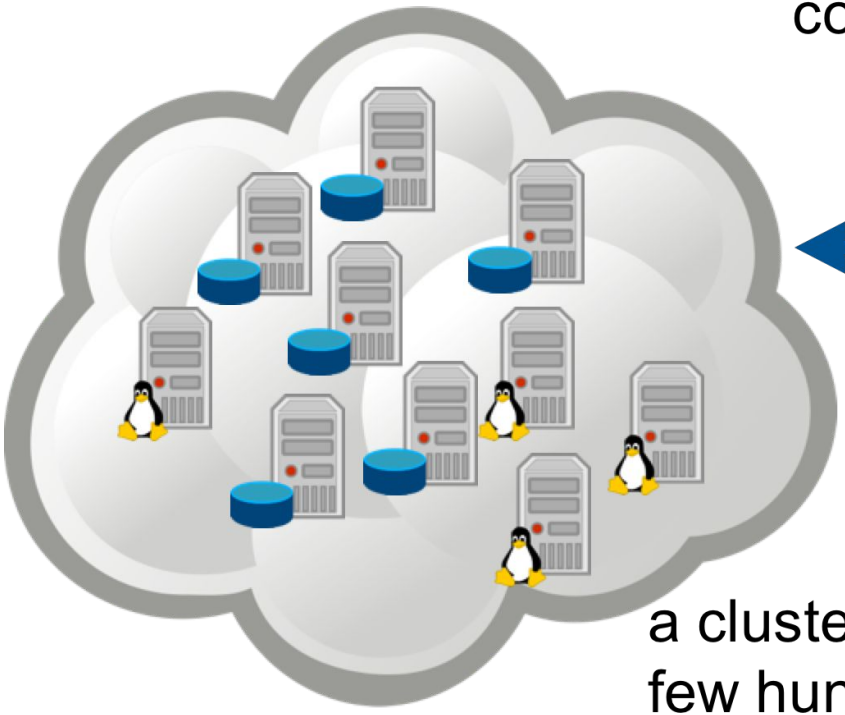
a cluster with a few hundred machines

one database

Question: How can the configuration be distributed?

配置文件

configuration file



application logic

A program that crawls the Web

每个请求都以同一个配置文件开始

Every worker should start with the same configuration

coordination logic

a cluster with a few hundred machines

Solution approaches

- Be specific: develop a particular service for each coordination task
  - Leading election
  - Locking service
  - Configuration
- Be general: provide an API to make many services possible

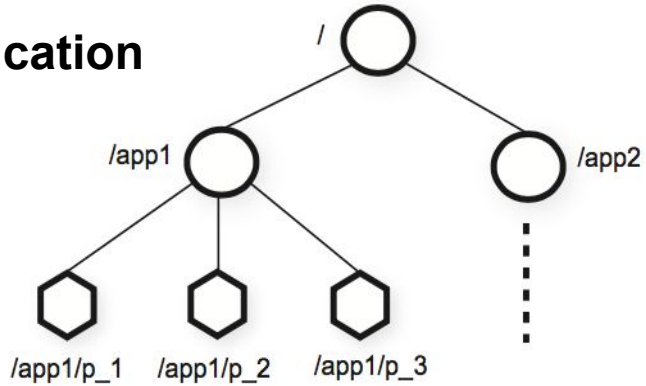
ZooKeeper: exposing an API that enables application developers to **implement their own primitives**

ZooKeeper Terminology

- **Client**: user of the ZooKeeper service
- **Server**: process providing the ZooKeeper service
- **znode**: **in-memory data** node in ZooKeeper, organised in a hierarchical name space (the data tree)
- **Update/write**: any operation which modifies the state of the data tree
- Clients establish a **session** when connecting to ZooKeeper

ZooKeeper's data model

- znodes are organized in a hierarchical name space
- use the standard UNIX notation for file system paths
- znodes are not designed for general data storage. Instead, they map to **abstractions of the client application**



Znode

- **Regular**: Create and delete by client explicitly

- **Ephemeral**: Either delete them explicitly, or let the system remove them automatically when the session that creates them terminates (deliberately or due to a **failure**).

Znode

- **SEQUENTIAL** flag:
  - monotonically increasing counter appended to a znode's path
  - counter value of a new znode under a parent is always larger than value of existing children

- **WATCH** flag:
  - allow clients to receive timely notifications of changes without requiring polling
  - Watches indicate that a change has happened, but do not provide the change

ZooKeeper API

- `String create(path, data, flags)`
  - creates a znode with path name path, stores data in it and sets flags (ephemeral, sequential)
- `void delete(path, version)`
  - deletes the anode if it is at the expected version
- `Stat exists(path, watch)`
  - watch flag enables the client to set a watch on the znode
- `(data, Stat) getData(path, watch)`
  - returns the data and meta-data of the znode
- `Stat setData(path, data, version)`
  - writes data if the version number is the current version of the znode
- `String[] getChildren(path, watch)`

ZooKeeper API

- ZooKeeper does not use handles to access znodes
  (no `open()` or `close()` methods)

- All methods have both a synchronous and an asynchronous version

ZooKeeper Guarantees

- **Linearizable writes**: all requests that **update** the state of ZooKeeper are serializable and respect precedence
- **FIFO client order**: all requests from a given client are executed in the order that they were sent by the client

A-linearizability (asynchronous linearizability)

Think!

When electing a new leader:

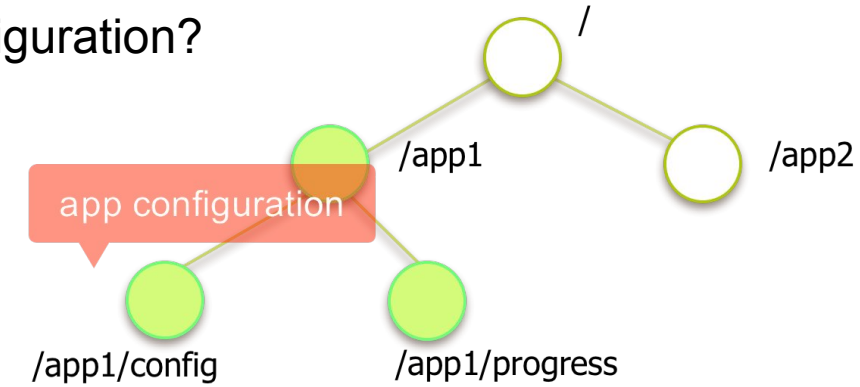
- As the new leader starts making changes, we do not want other processes to start using the configuration that is being changed
- If the new leader dies before the configuration has been fully updated, we do not want the processes to use this partial configuration

deleting ***ready*** znode -> updating the various configuration znodes -> creating ***ready*** znode

Example of primitives: Configuration

Questions:

1. How does a **new** worker query for a configuration?
2. How does an administrator **change** the configuration **on the fly**?
3. How do the workers read the **new** configuration?



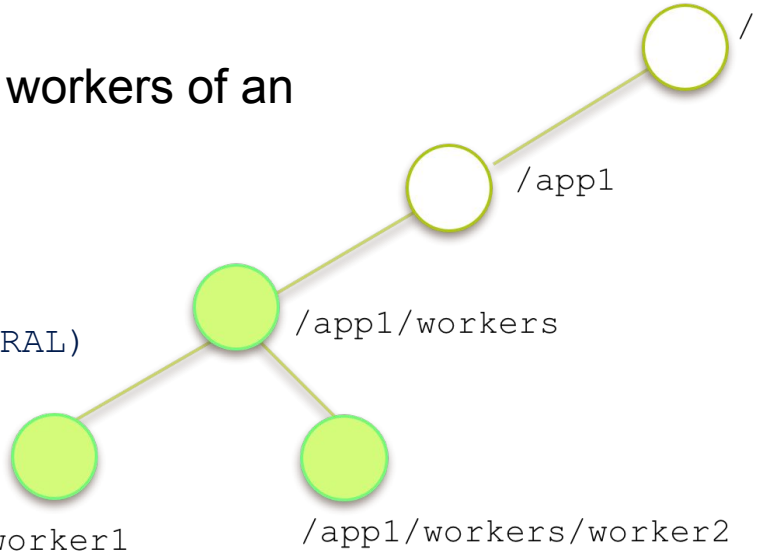

```
[configuration stored in/app1/config]
1.  getData(/app1/config,true)
2.  setData(/app1/config/config_data,-1)
    [notify watching clients]
3.  getData(/app1/config,true)
```

Example of primitives: Group Membership

Questions: 一个应用程序中的所有 workers 如何注册他们自己？

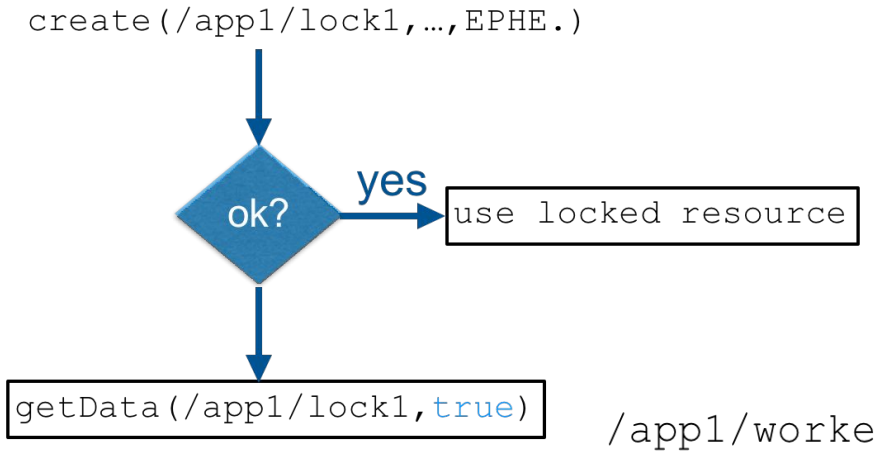
1. How can all workers (slaves) of an application register themselves?
2. How can a process find out about all active workers of an application? 进程如何找到一个应用程序中的所有活跃的workers？

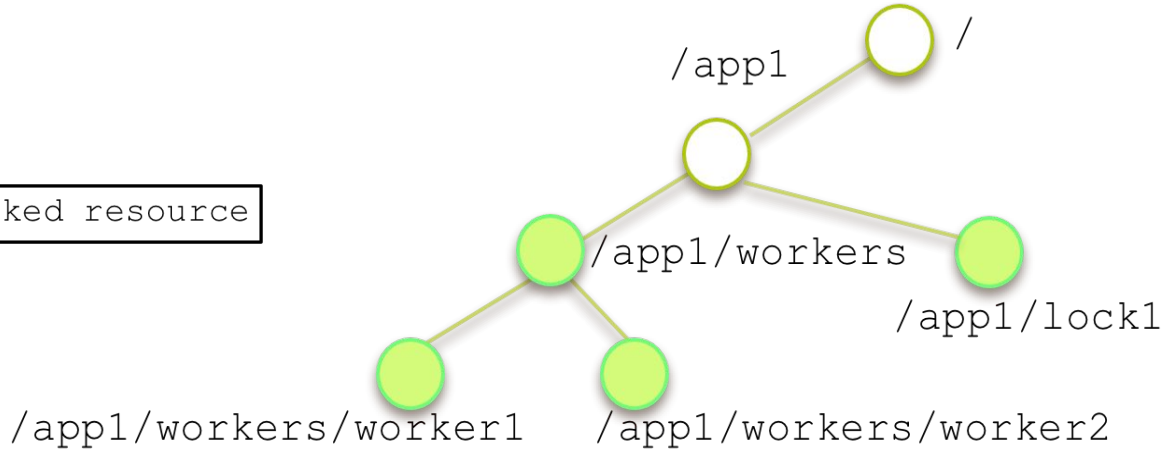
```
[a znode is designated to store workers]
1.  create(/app1/workers/worker, data, EPHEMERAL)
2.  getChildren(/app1/workers, true)
```



Example of primitives: Simple Locks

Questions: How can all workers of an application use a single resource through a lock?





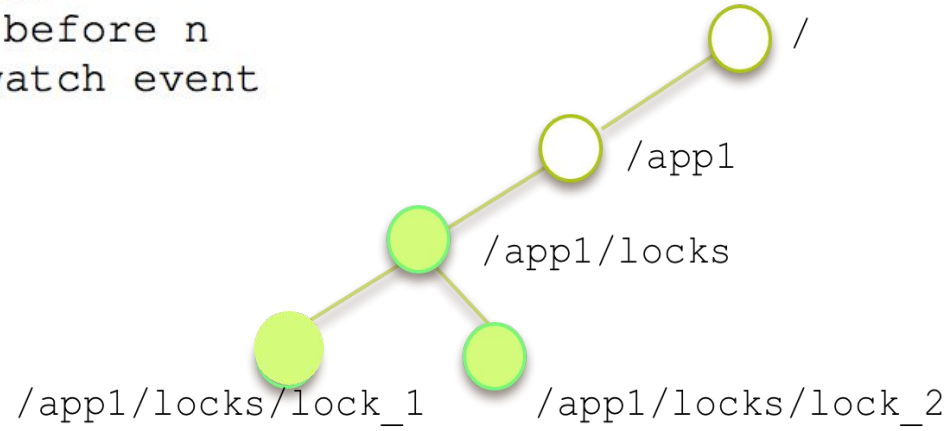
Example of primitives: Simple Locks without Herd Effect

Lock

```
1 n = create(l + "/lock-", EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event
6 goto 2
```

Unlock

```
1 delete(n)
```



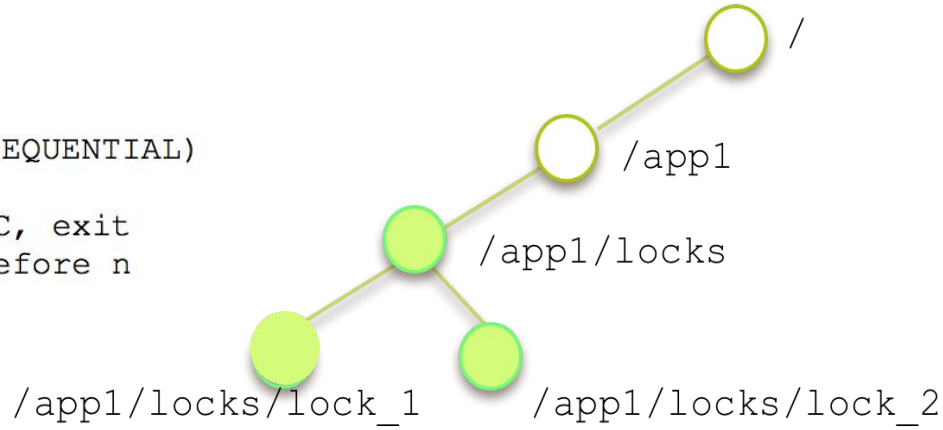
Example of primitives: Read/Write Locks

Write Lock

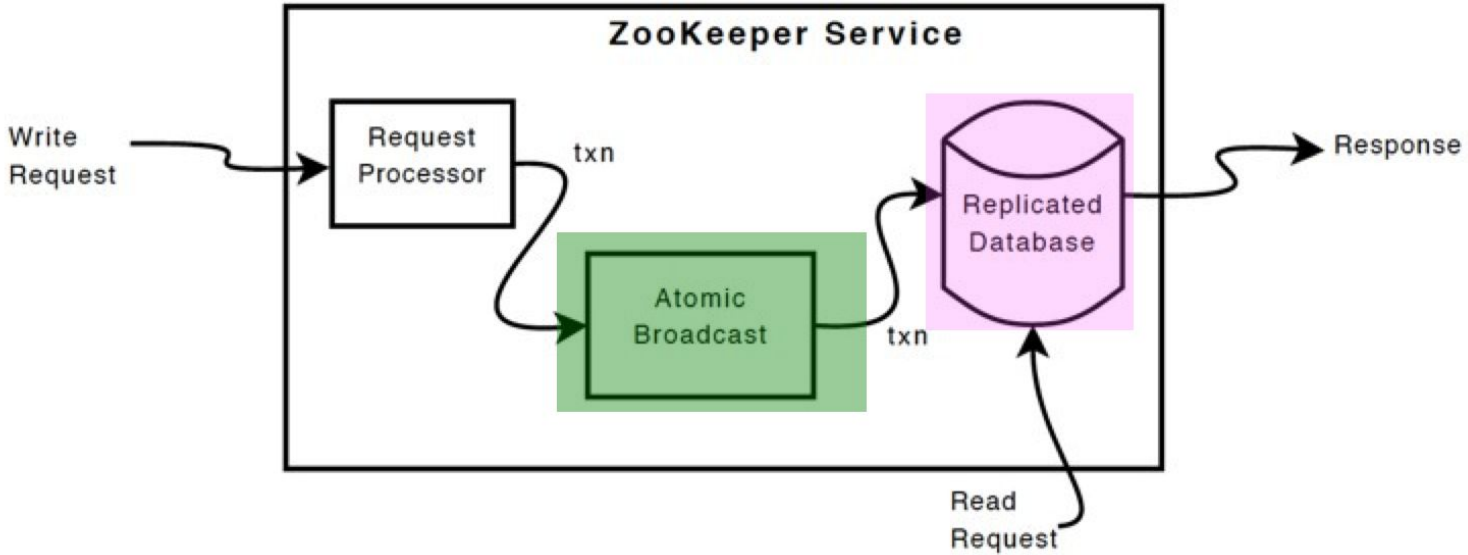
```
1  n = create(l + "/write-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if n is lowest znode in C, exit
4  p = znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 2
```

Read Lock

```
1  n = create(l + "/read-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if no write znodes lower than n in C, exit
4  p = write znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 3
```

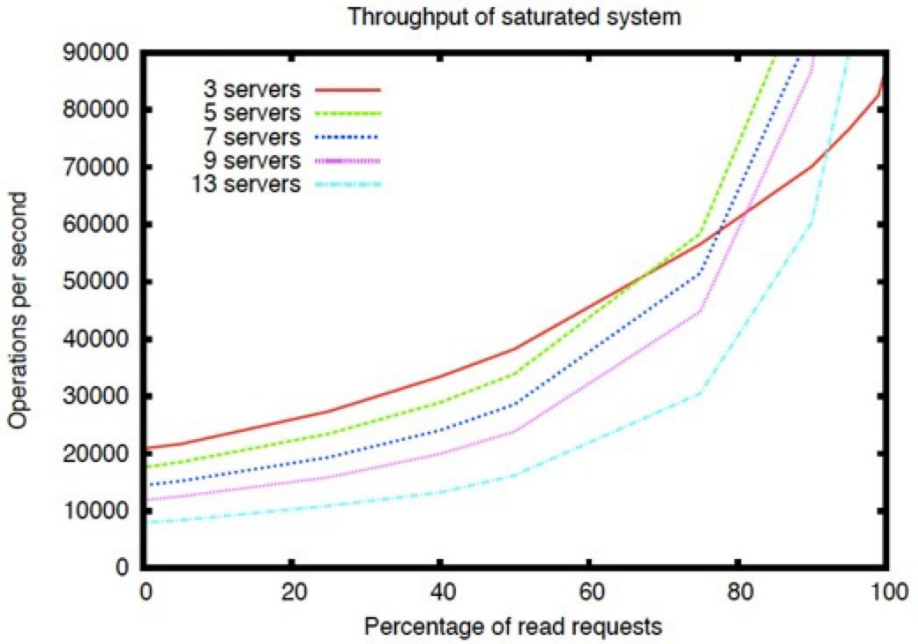


A little about implementation



Throughput

Setup: 250 clients, each client has at least 100 outstanding requests (read/write of 1K data)



Summary

- Wait-free
- Event ordering
- Watch mechanism

Thoughts

- a minimalist and flexible coordination system
- scales well with increase in read operations, but does not with increase in write operations
- punts the ball to the clients

Questions?

Reference

ZooKeeper: Wait-free coordination for Internet scale system, Hunt al., 2010

https://people.eecs.berkeley.edu/~istoica/classes/cs294/15/notes/18-zookeeper.pdf

http://deptinfo.unice.fr/twiki/pub/Minfo/DistributedAlgo/TUDelft-coordination_zookeeper.pdf