

Apache Zookeeper

Hunt, P., Konar, M., Junqueira, F.P. and Reed, B., 2010, June. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX Annual Technical Conference (Vol. 8, p. 9).

And other papers from <http://scholar.google.com>. ZK is a great example of successful open source software with strong underlying CS principles documented in scholarly articles.

Knowns and Unknowns in Distributed Systems

- There are the known knowns
 - “I recognize this problem. There is a design pattern for it.”
- And there are the known unknowns
 - “I bet there is a design pattern for this. Let’s identify it and go from there.”
- And then there are the unknown unknowns...
 - “I thought of a great way to solve this!”
 - “What could possibly go wrong?”

We want to remove some of the unknown unknowns.



<https://engineering.pinterest.com/blog/building-follower-model-scratch>

Pinterest engineering blog

- Using Zookeeper to store Redis shard configuration information.
- It took 1 lead developer and 2-3 part-time team members 8 weeks to go from nothing to production.
- Pinterest has about ~100 million active users.

Classic Problems in Distributed Systems

- Global Naming
- Identity Management, Authentication, Authorization
- Distributed Transactions
- Distributed State Machines
- Messaging, publish-subscribe, notification
- Leader Election, primary copies, failover

Classic Distributed Systems Examples

Git

- Versioning

Distributed File
Systems

- Write, not just read at scale
- Global identities, groups

Replicated Databases

- Primary copies, leaders, transactions

Resource Managers

- Torque, SLURM, etc
- Fault tolerance, process coordination

More Examples

Domain Name Servers

- Naming, caching
- Eventual consistency
- Scaling power of hierarchies and name spaces

REST Systems

- Idempotent state
- Error messages

Queuing Systems

- Message order, replay, guaranteed delivery

Apache Zookeeper as a Case Study

Apache
Zookeeper

- Illustrates several distributed systems concepts
- Some interesting design choices and assumptions

Illustrative Only

- Lectures are not an endorsement

Many Others to
Study

- Apache Storm, Kafka, Spark, Hadoop, etc

Interesting Times

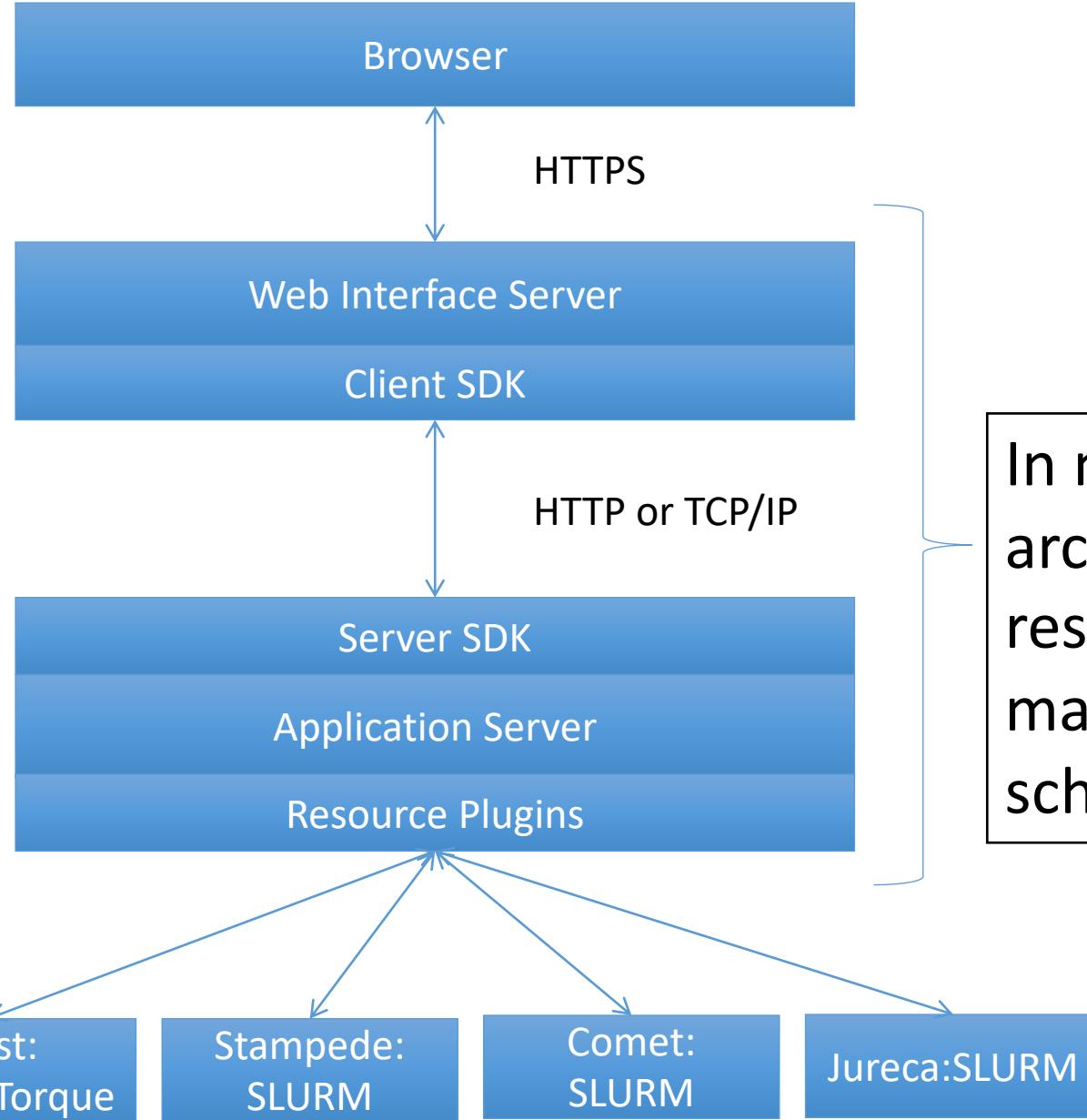
- Open source software that is small enough to understand and experiment with
- But cloud-native, scalable, used to run many companies

Some General Advice

- Take a class in distributed systems.
- Get a book on distributed systems and read it.
 - Smart people have thought about these problems for a long time.
 - Jim Gray invented two-phase commit c. 1978, for example
 - CS problems often map to non-CS analogies
- There are already many algorithms and design patterns for most problems that you will face.
- The challenge: knowing the right prior solution for the problem at hand
- The anti-pattern: reinventing something in a text book out of ignorance

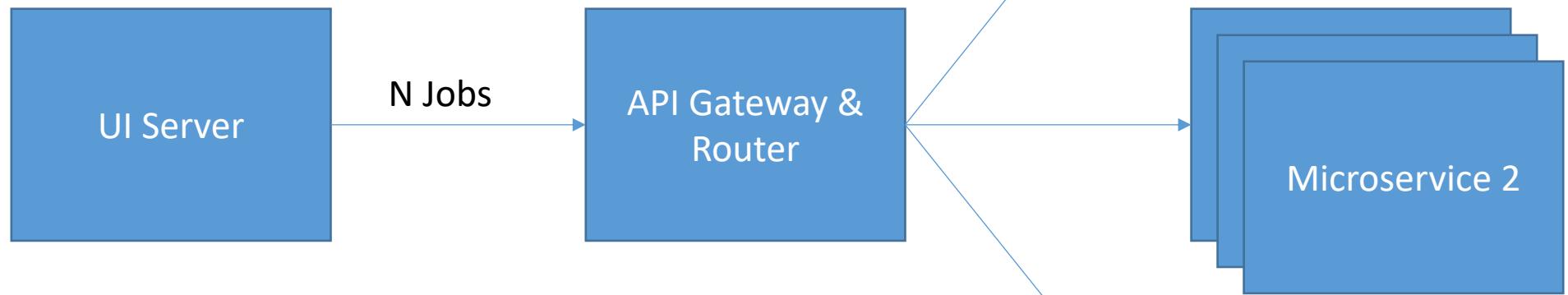
Recall the Gateway Octopus Diagram

“Super” Scheduling
and Resource
Management



In micro-service arch, these need resource management and scheduling

A Simplified Microservice System



Some questions:

- Is Replica 2 of microservice 3 up and running?
- Do I have at least one service running?
- Microservice 3 uses Master-Worker, and the Master just failed. What do I do?
- Replica 2 just came up and needs to find configuration information. How can it do that?

Apache Zookeeper and Microservices

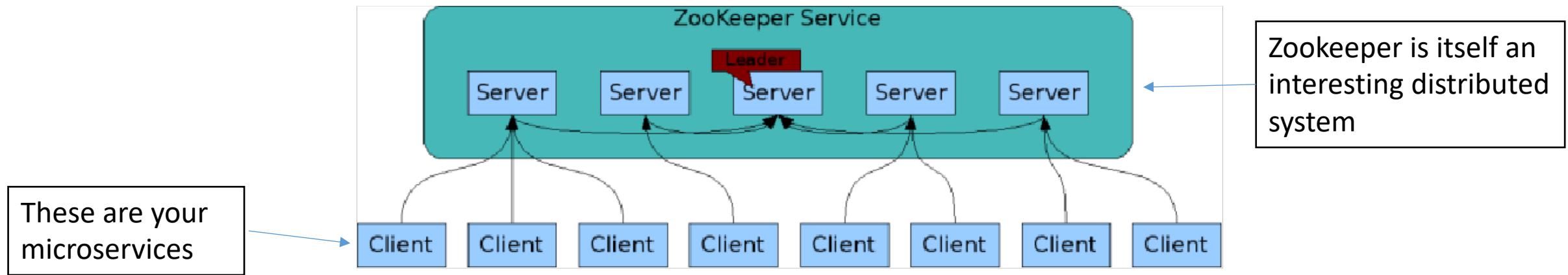
- ZK can manage information in your system
 - IP addresses, version numbers, and other configuration information of your microservices.
 - This is stuff that is dynamic or otherwise doesn't fit into a static configuration file.
- The health of the microservices.
- The state of a particular calculation.
- Group membership



Apache Zookeeper Is...

- A system for solving **distributed coordination problems** for multiple cooperating clients.
- A lot like a distributed file system...
 - As long as the files are tiny.
 - And you could get notified when the file changes
 - And the full file pathname is meaningful to applications
- A way to solve microservice management problems.
- **An interesting implementation of a distributed system itself.**
 - Look under the hood AND read the papers

The ZooKeeper Service



- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data in memory (!)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server.
- Writes go through the leader & need majority consensus.

Zookeeper Summary

- ZooKeeper provides a simple and high performance kernel for building more complex coordination primitives.
 - Helps distributed components share information
- Clients (your applications) contact Zookeeper services to read and write metadata.
 - Read from cache but writes are more complicated
 - Sometimes, just the existence and name of a node are enough
- Tree model for organizing information into nodes.
 - Node names may be all you need
 - Lightly structured metadata stored in the nodes.
- **Wait-free** aspects of shared registers with an **event-driven** mechanism similar to cache invalidations of distributed file systems
- Targets simple metadata systems that read more than they write.
 - Small total storage

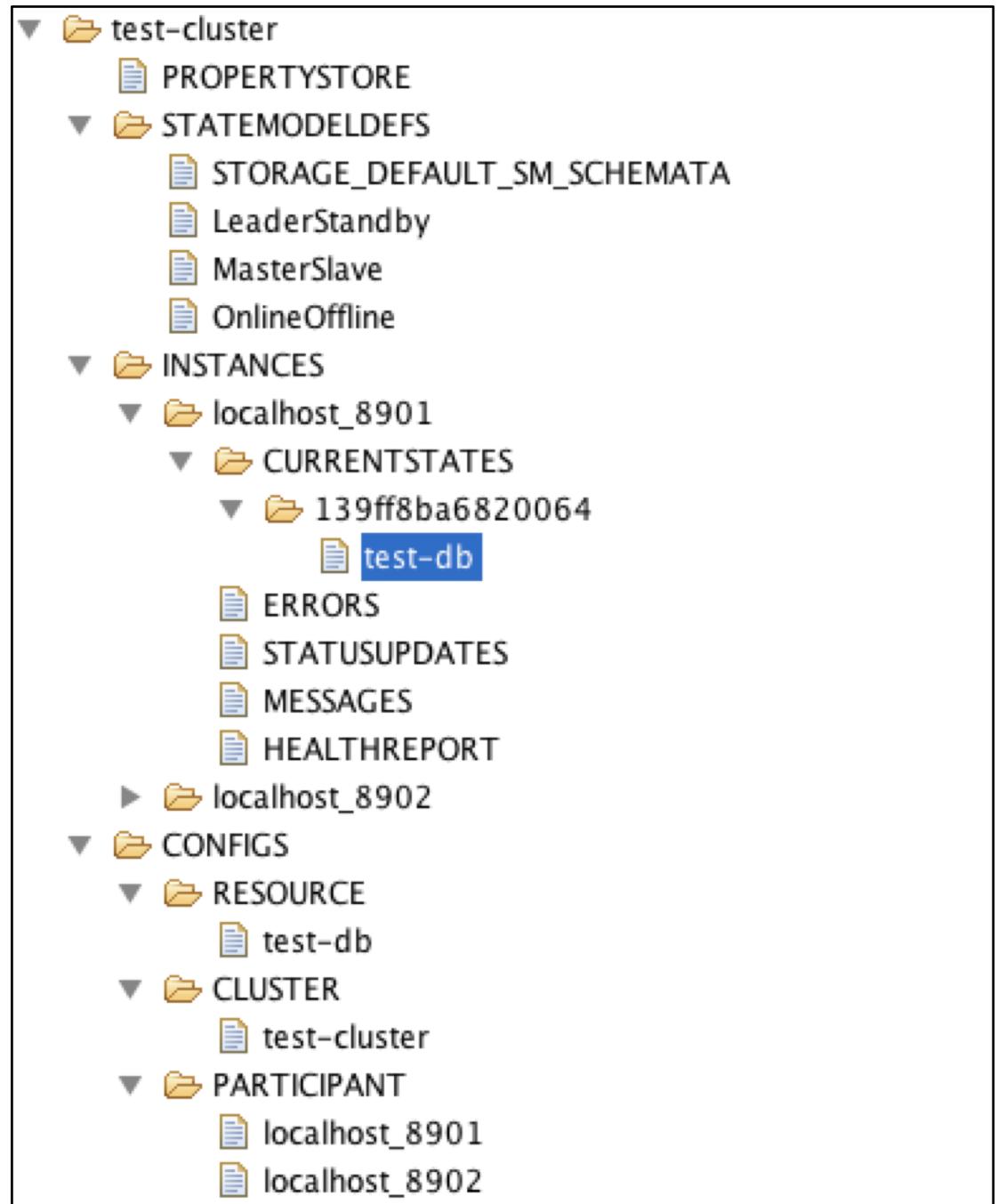
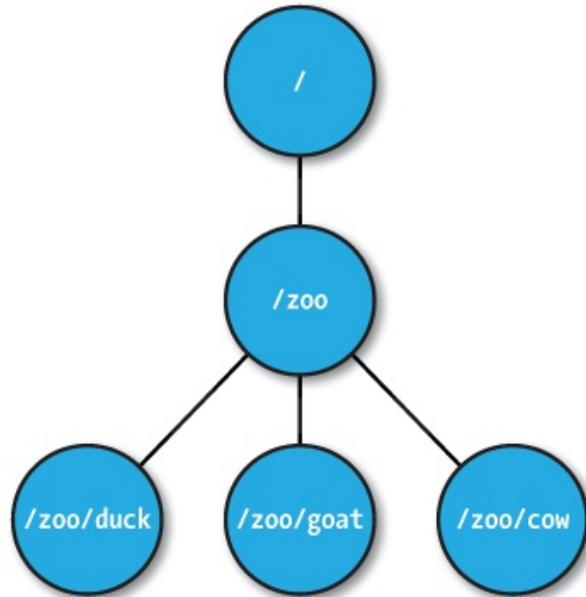
Zookeeper, More Briefly

- Zookeeper Clients (that is, your applications) can create and discover nodes on ZK trees
- Clients can put small pieces of data into the nodes and get small pieces out.
 - 1 MB max for all data per server by default
 - Each node also has built-in metadata like its version number.
- You could build a small DNS or an LDAP server with Zookeeper.

Can you think of ways that this would be useful?

ZNodes

- Maintain a stat structure with version numbers for data changes, ACL changes and timestamps.
- Version numbers increases with changes
- Data is read and written in its entirety



ZNode types

Regular

- Clients create and delete explicitly

Ephemeral

- Like regular znodes associated with sessions
- Deleted when session expires

Both Regular and Ephemeral Nodes can be **Sequential**: the name includes a universal, monotonically increasing counter

Zookeeper API (1/2)

- **create(path, data, flags):** Creates a znode with path name path, stores data[] in it, and returns the name of the new znode.
 - *flags* enables a client to select the type of znode: regular, ephemeral, and set the sequential flag;
- **delete(path, version):** Deletes the znode path if that znode is at the expected version
- **exists(path, watch):** Returns true if the znode with path name path exists, and returns false otherwise.
 - Note the *watch* flag

Zookeeper API (2/2)

- **getData(path, watch)**: Returns the data and meta-data, such as version information, associated with the znode.
- **setData(path, data, version)**: Writes data[] to znode path if the version number is the current version of the znode
- **getChildren(path, watch)**: Returns the set of names of the children of a znode
- **sync(path)**: Waits for all updates pending at the start of the operation to propagate to the server that the client is connected to.

What Can You Do with this Simple API?

Recall earlier our classic distributed systems: DNS, Git, NFS, etc.

Reality Check

- Zookeeper's API is low level and can be hard to use.
- Apache Curator (<http://curator.apache.org/>) provides a higher level API and recipes for using Zookeeper.
 - Easier to use and less error prone
 - But Java only
- HashiCorp's Consul is also a popular choice
 - “Next generation” after Zookeeper
 - Enterprise considerations like security built in

What Can You Do with this Simple API?

Recall earlier our classic distributed systems: DNS, Git, NFS, etc.

Configuration Management

- All clients get their configuration information from a named znode
 - /root/config-me
- Example: you can build a public key store with Zookeeper
- Clients set watches to see if configurations change
- Zookeeper doesn't explicitly decide which clients are allowed to update the configuration.
 - That would be an implementation choice
 - Zookeeper uses leader-follower model internally, so you could model your own implementation after this.

The Rendezvous Problem

- Classic distributed computing algorithm
- Consider master-worker
 - Specific configurations may not be known until runtime
 - EX: IP addresses, port numbers
 - Workers and master may start in any order
- Zookeeper implementation:
 - Create a rendezvous node: /root/rendezvous
 - Workers read /root/rendezvous and set a watch
 - If empty, use watch to detect when master posts its configuration information
 - Master fills in its configuration information (host, port)
 - Workers are notified of content change and get the configuration information

Locks

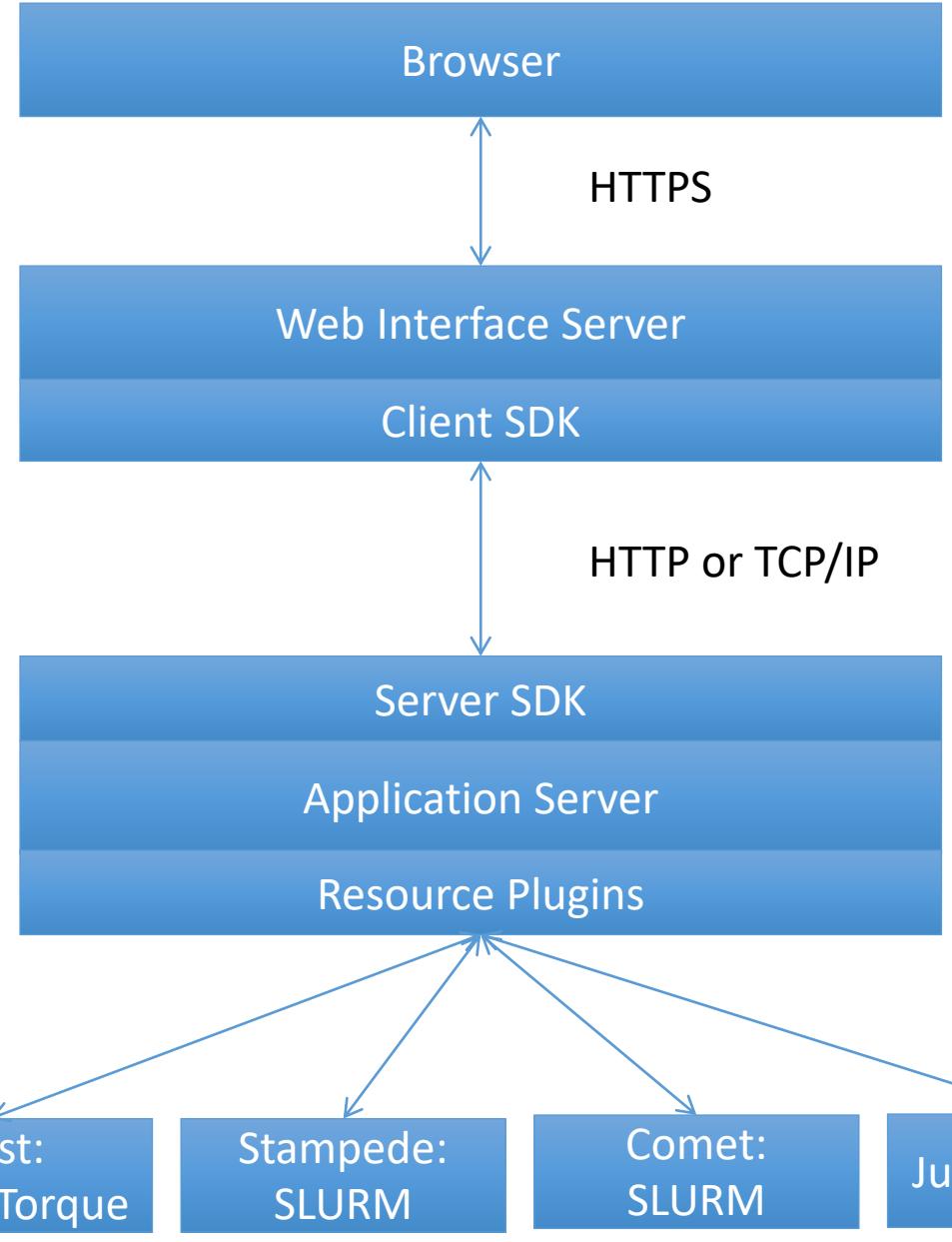
- Familiar analogy: lock files used by Apache HTTPD and MySQL processes
- Zookeeper example: who is the leader with primary copy of data?
- Implementation:
 - Leader creates an ephemeral file: /root/leader/lockfile
 - Other would-be leaders place watches on the lock file
 - If the leader client dies or doesn't renew the lease, clients can attempt to create a replacement lock file
- Use SEQUENTIAL to solve the herd effect problem.
 - Create a sequence of ephemeral child nodes
 - Clients only watch the node immediately ahead of them in the sequence

Zookeeper and Science Gateways

Recall the Gateway Octopus Diagram

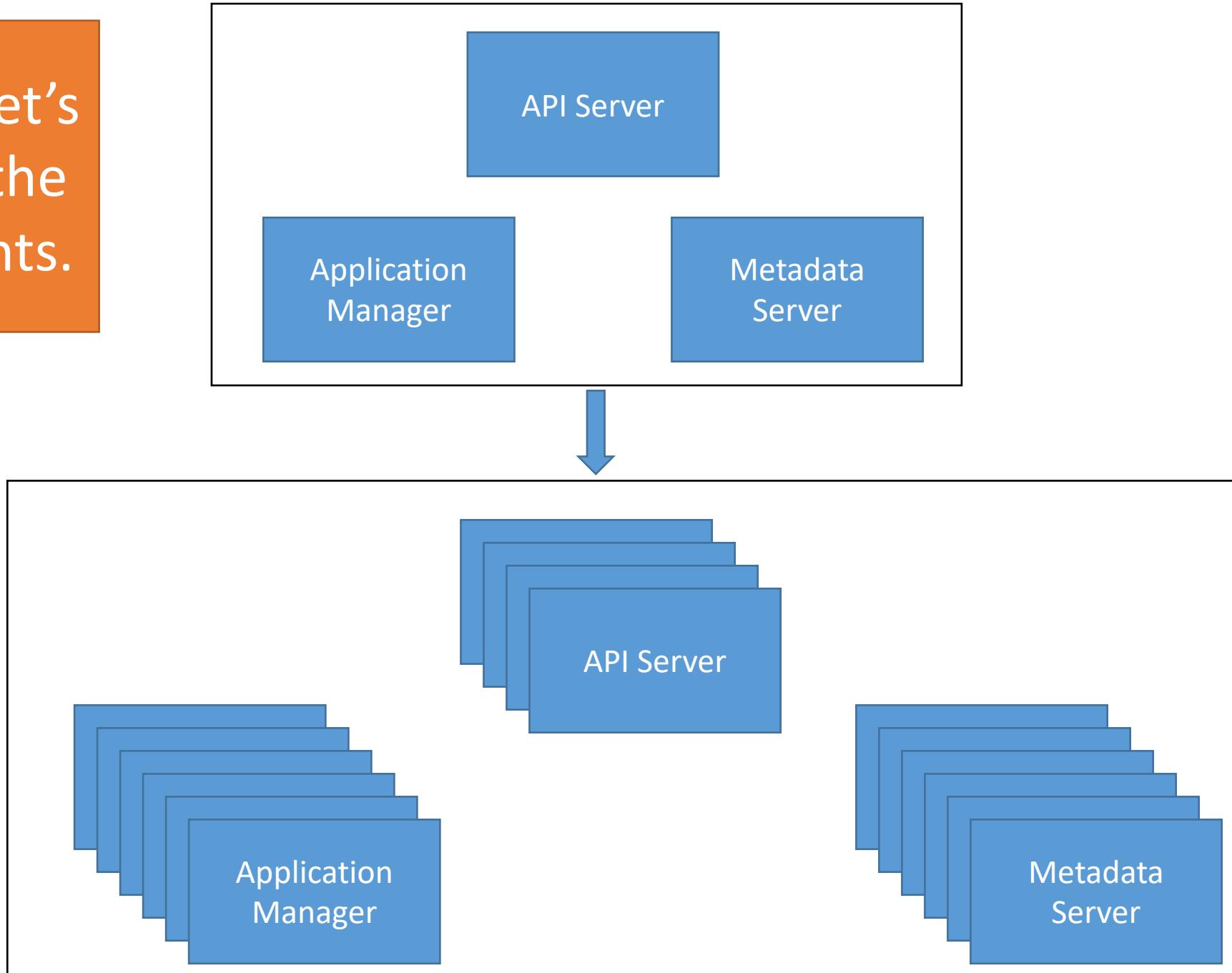
“Super” Scheduling
and Resource
Management

Different archs,
Schedulers,
admin domains,
...



In micro-service arch, these also need scheduling

And now let's
replicate the
components.



Why Do This?

- Fault tolerance
- Increased throughput, load balancing
- Component versions
 - Not all components of the same type need to be on the same version
 - Backward compatibility checking
- Component flavors
 - Application managers can serve different types of resources
 - Useful to separate them into separate processes if libraries conflict.

Configuration Management

- Problem: gateway components in a distributed system need to get the correct configuration file.
- Solution: Components contact Zookeeper to get configuration metadata.
- Comments: this includes both the component's own configuration file as well as configurations for other components
 - Rendezvous problem

Service Discovery

- Problem: Component A needs to find instances of Component B
- Solution: Use Zookeeper to find available group members instances of Component B
 - More: get useful metadata about Component B instances like version, domain name, port #, flavor
- Comments
 - Useful for components that need to directly communicate but not for asynchronous communication (message queues)

Group Membership

- Problem: a job needs to go to a specific flavor of application manager. How can this be located?
- Solution: have application managers join the appropriate Zookeeper managed group when they come up.
- Comments: This is useful to support scheduling

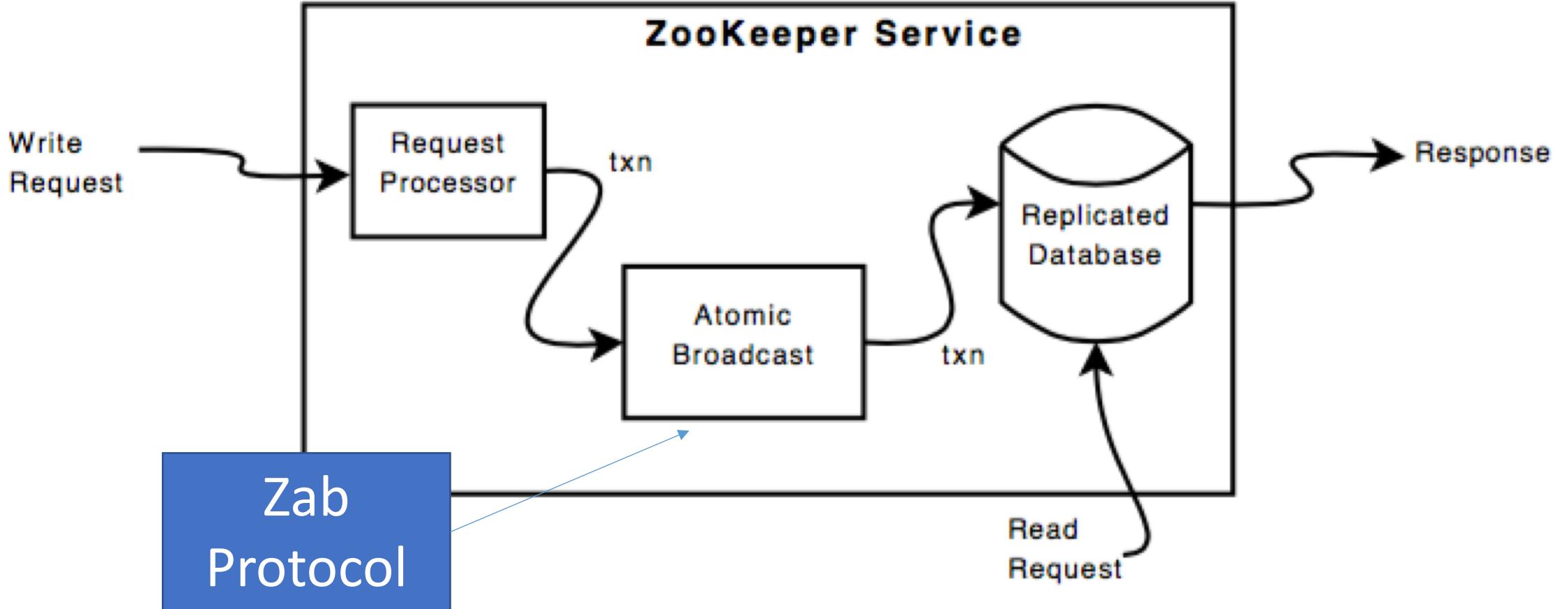
System State for Distributed Systems

- Which servers are up and running? What versions?
- Services that run for long periods could use ZK to indicate if they are busy (or under heavy load) or not.
- Note overlap with our Registry
 - What state does the Registry manage? What state would be more appropriate for ZK?

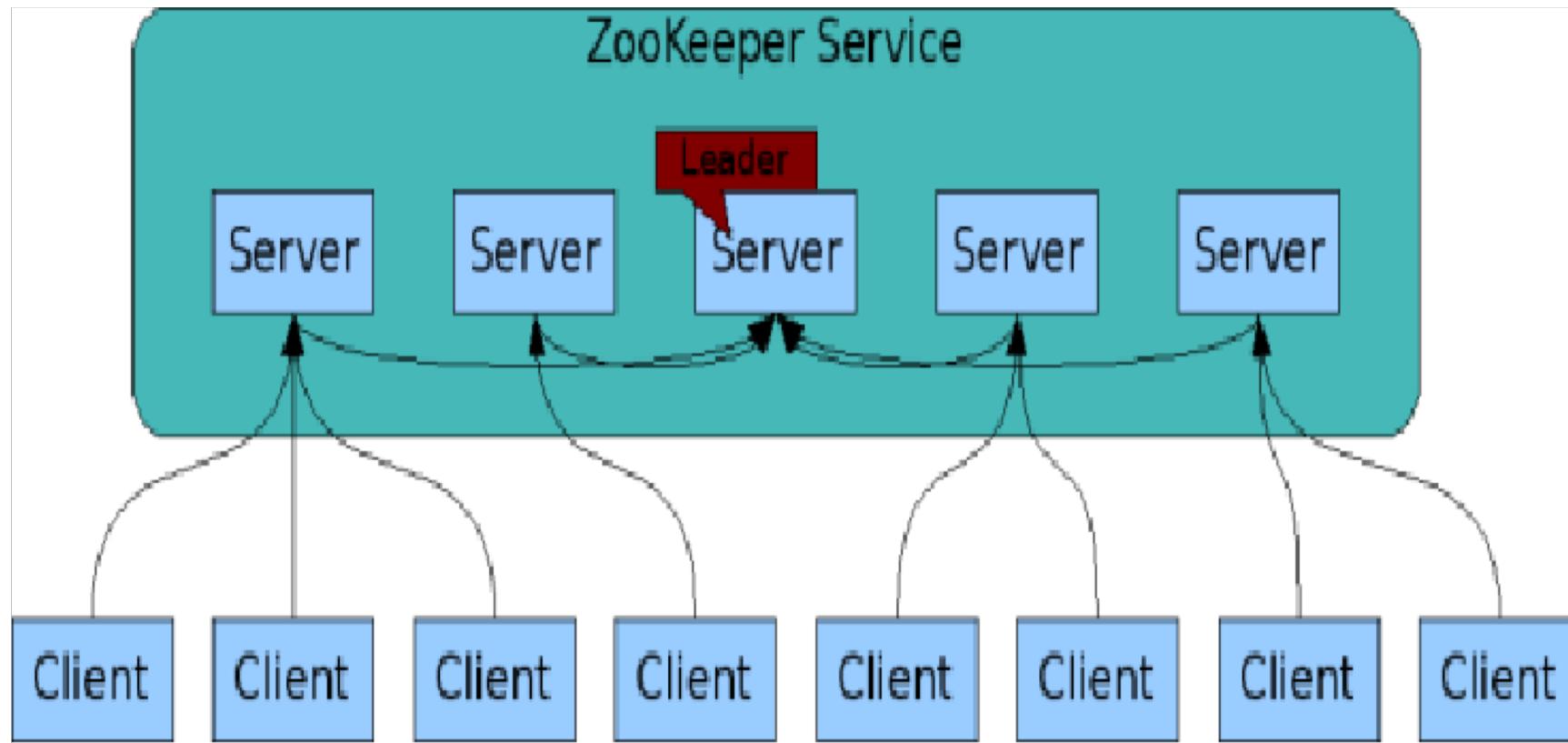
Leader Election

- Problem: metadata servers are replicated for read access but only the master has write privileges. The master crashes.
- Solution: Use Zookeeper to elect a new metadata server leader.
- Comment: this is not necessarily the best way to do this

Under the Zookeeper Hood



Instead of Zab, we'll examine the simpler RAFT protocol. Both are described in academic papers

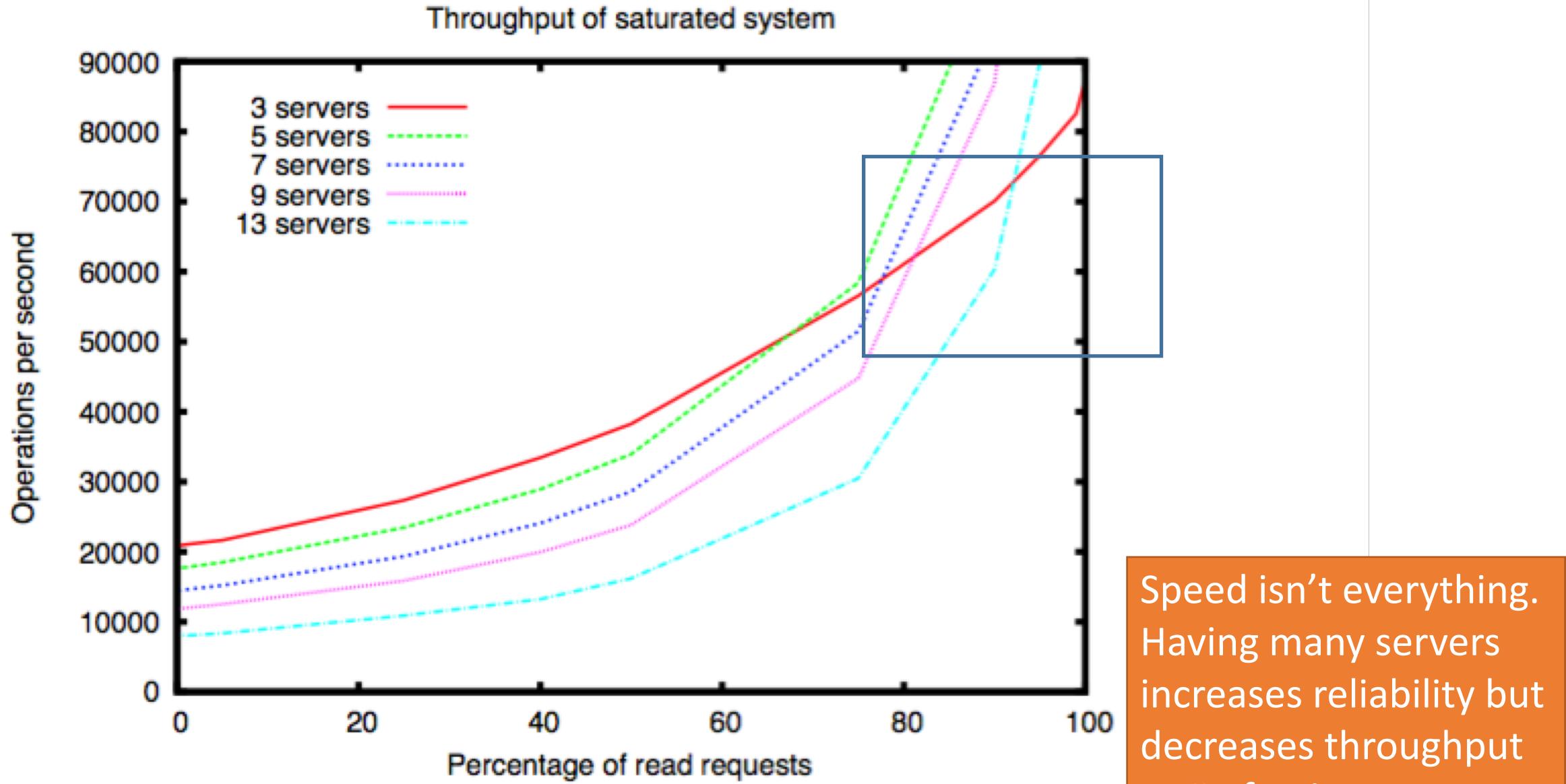


Zookeeper Handling of Writes

- READ requests are served by any Zookeeper server
 - Scales linearly, although information can be stale
- WRITE requests change state so are handled differently
- One Zookeeper server acts as the leader
- The leader executes all write requests forwarded by followers
- The leader then broadcasts the changes
- The update is successful if a majority of Zookeeper servers have correct state at the end of the process
- Zab protocol

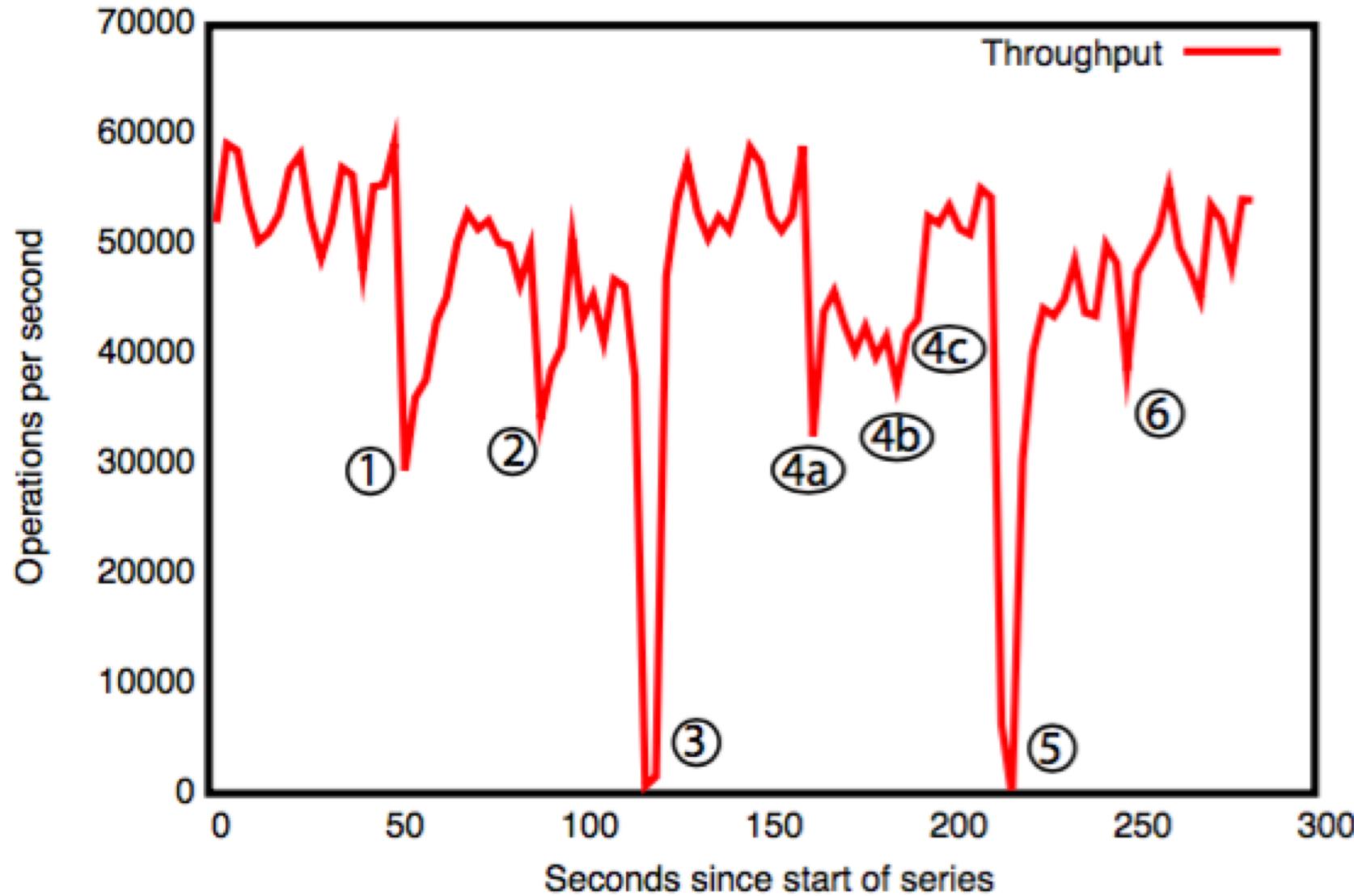
Some Zookeeper Implementation Simplifications

- Uses TCP for its transport layer.
 - Message order is maintained by the network
 - The network is reliable?
- Assumes reliable file system
 - Logging and DB checkpointing
- Does write-ahead logging
 - Requests are first written to the log
 - The ZK DB is updated from the log
- ZK servers can acquire correct state by reading the logs from the file system
 - Checkpoint reading means you don't have to reread the entire history
- Assumes a single administrator so no deep security



Speed isn't everything.
Having many servers
increases reliability but
decreases throughput
as # of writes
increases.

Time series with failures



1. Failure and recovery of follower.
2. Failure and recovery of follower.
3. Failure of leader (200 ms to recover).
4. Failure of two followers (4a and 4b), recovery at 4c.
5. Failure of leader
6. Recovery of leader (?)

A cluster of 5 zookeeper instances responds to manually injected failures.

Final Thoughts and Cautions

- Zookeeper is powerful but it is only one possible solution.
- Messaging is also very powerful distributed computing concept
 - You could build a queuing system with Zookeeper, but you shouldn't
 - <https://cwiki.apache.org/confluence/display/CURATOR/TN>
 - There are high quality queuing systems already
- Highly available versus elastic, recoverable components
 - Zookeeper is better in the latter case
- Where is the state of your system? Make one choice. Don't have shadow states.

Introduction to Distributed Systems

Or, What I Have Learned Since 1998

Some Definitions

Distributed Systems

- Multiple software components that work together as a single composite entity.

Distributed Computing

- Distributed components work together to perform large computations by breaking them into smaller parts.
- Examples: HTCondor, Apache Hadoop, Apache Spark, MPI, etc

Parallel Computing

- A special case of distributed computing
- Typically on a specialized resource (Big Red II, Blue Waters, etc)
- Can assume tighter coupling

Science Gateways

- At scale are distributed systems.
- Theme for this lecture

Classic Problems in Distributed Systems

- Global Naming
- Identity Management, Authentication, Authorization
- Distributed Transactions
- Distributed State Machines
- Leader Election
- Messaging, publish-subscribe, notification

Classic Distributed Systems Examples

Git

- Versioning

Distributed File
Systems

- Write, not just read at scale
- Global identities, groups,

Replicated Databases

- Primary copies, leaders, transactions

Resource Managers

- Torque, SLURM, etc
- Fault tolerance, process coordination

More Examples

Domain Name Servers

- Naming, caching
- Eventual consistency

REST Systems

- Idempotent state
- Error messages

Queuing Systems

- Message order, replay, guaranteed delivery

Domain Name Servers

- Scaling power of hierarchies and name spaces

The Fallacies of Distributed Computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Zookeeper Design Principles

- For read-dominated systems
- Doesn't prescribe specific coordination primitives.
 - Lower level than that.
- Allow **clients** to create their own coordination applications using the Zookeeper API
 - Configuration Management
 - Locking
 - Rendezvous
 - Group Membership



Zookeeper: Wait-Free Data Objects

Key design choice: wait-free data objects

- Locks are not a Zookeeper primitive
- You can use Zookeeper to build lock-based systems

Resembles distributed file systems

- Smaller data

FIFO client ordering of messages

- Asynchronous messaging
- Assumes you can order messages globally

Basic idea: idempotent state changes

- Components can figure out the state by looking at the change log
- Operations incompatible with state throw exceptions

A good approach when systems can tolerate inconsistent data

- DNS for example
- But not E-Commerce, which needs stronger guarantees

Coordination Examples in Distributed Systems

Configuration

Basic systems just need lists of operational parameters for the system processes: IP addresses of other members
Sophisticated systems have dynamic configuration parameters.

Group Membership, Leader Election

- Processes need to know which other processes are alive
- Which processes are definitive sources for data (leaders)?
- What happens if the leader is faulty?

Locks

- Implement mutually exclusive access to critical resources.

Zookeeper Caches and Watches

Zookeeper clients cache data

- Reads go to the cache

Watches: notify listening clients
when cache has changed

- Watches don't convey the content of the change.
- Only work once; clients decide on the action

Why?

- Networks aren't uniform or reliable (fallacy)
- Centralized, top-down management doesn't scale

Suitable for read-dominated
systems

- If you can tolerate inconsistencies