ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, Benjamin Reed
2010

Presented by Chongyang Shi (advocate) on Feb 9, 2018

Existing coordination systems

- Most coordination systems at the time were primitive-specific:
  - Queuing: Amazon Simple Queue Service
  - Leader election: Schiper and Toueg, 2008
  - Configuration management: Sherman et al., 2005
- Or lock-based, or too high level of abstraction to be universal
  - Chubby (Burrows, 2006)
- Need to allow implementing primitives from a lower level, and if possible be lock-free or even wait-free…

And similar ideas…

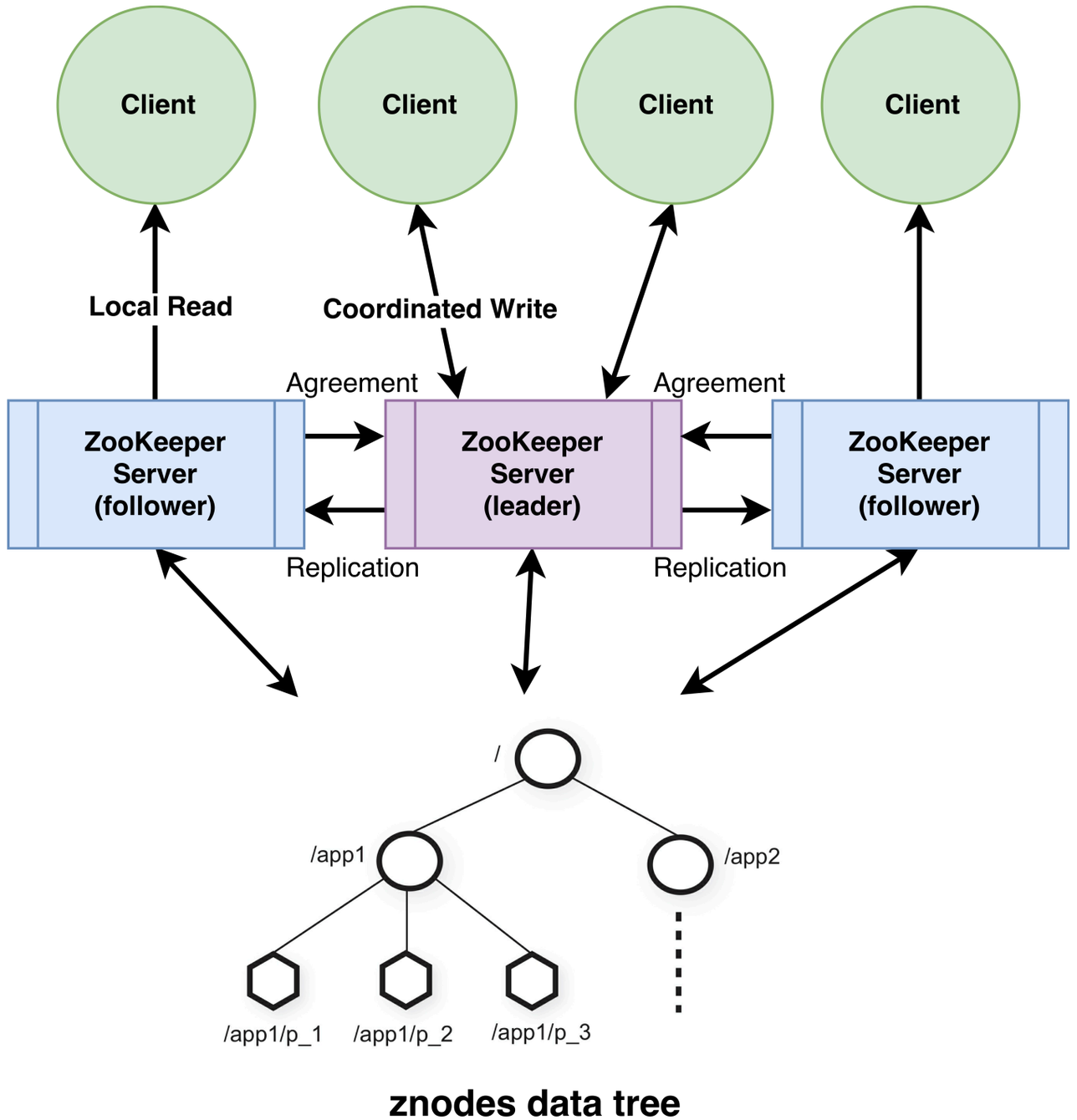
- Fault-tolerant systems
  - ISIS (Birman, 1985) and derivatives - less abstraction
  - Paxos (Lamport, 1998) - efficiently replicated state machines
- Distributed lock servers
  - Boxwood (MacCormick et al., 2004) - performance scaled less well
- Mini-transactions
  - Sinfonia (Aguilera et al., 2007) - optimised for application data storage (rather than meta-data), and without change notifications (watches)
  - Dynamo (DeCandia, et al., 2007) - non-hierarchical storage and no write consistency

Building High Performance Coordination Kernels

- Configuration
    - Static and dynamic parameters
- Group membership
    - Awareness of other members in the group
    - Failure detection and recovery
- Leader election
    - Consensus systems

ZooKeeper

- Client-Server-Store model.
- Hierarchal data store for storing coordination data mostly small in size.
- Asynchronous multi-queuing support for clients.
- API for higher level mechanism implementations.



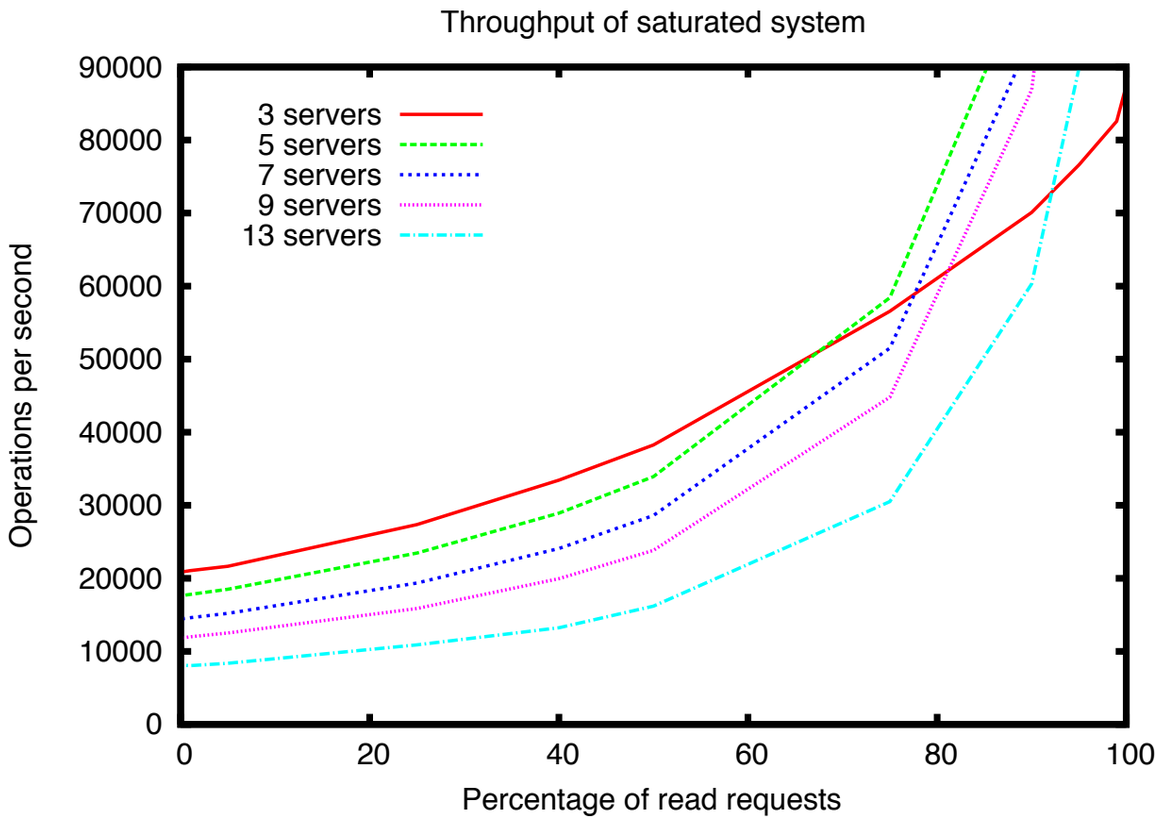
Reliability

- Wait-free coordination
  - In addition to lock-free — no deadlocking
- Guaranteed FIFO client ordering
- Linearisable writes
  - Snapshots for recovery are therefore rolling rather than stop-the-world
- Server replication and automated recovery of consensus
- Sessions and ephemeral nodes to prevent stalling by dead peers

Performance

- Support multiple, asynchronous outstanding requests per client
- Polling-free waiting
    - Server notifies clients of znode changes through client *watches*
        - Implicit guarantee of notification-before-read ordering
        - Explicit memory synchronisation available with low performance impact
- Fast, up-to-date group information retrieval through client-maintained ephemeral child nodes under a group node.
- In-order messaging processed directly through TCP

Performance, as seen in benchmarks



Servers	100% Reads	0% Reads
13	460k	8k
9	296k	12k
7	257k	14k
5	165k	18k
3	87k	21k

Each client:
> 100 requests outstanding
1K per read/write

Server:
Max of 2,000 concurrent requests
Can be separated into ensembles

Under extreme saturation:
Very fast reads due to read locality;
Slower writes due to atomic broadcast and file system logging, but can be improved with faster network hardware and SSDs.

Performance, as seen in benchmarks

	Number of servers			
Workers	3	5	7	9
1	776	748	758	711
10	2074	1832	1572	1540
20	2740	2336	1934	1890

Table 2: Create requests processed per second.

	# of clients		
# of barriers	**50**	**100**	**200**
200	9.4	19.8	41.0
400	16.4	34.1	62.0
800	28.9	55.9	112.1
1600	54.0	102.7	234.4

Table 3: Barrier experiment with time in seconds. Each point is the average of the time for each client to finish over five runs.

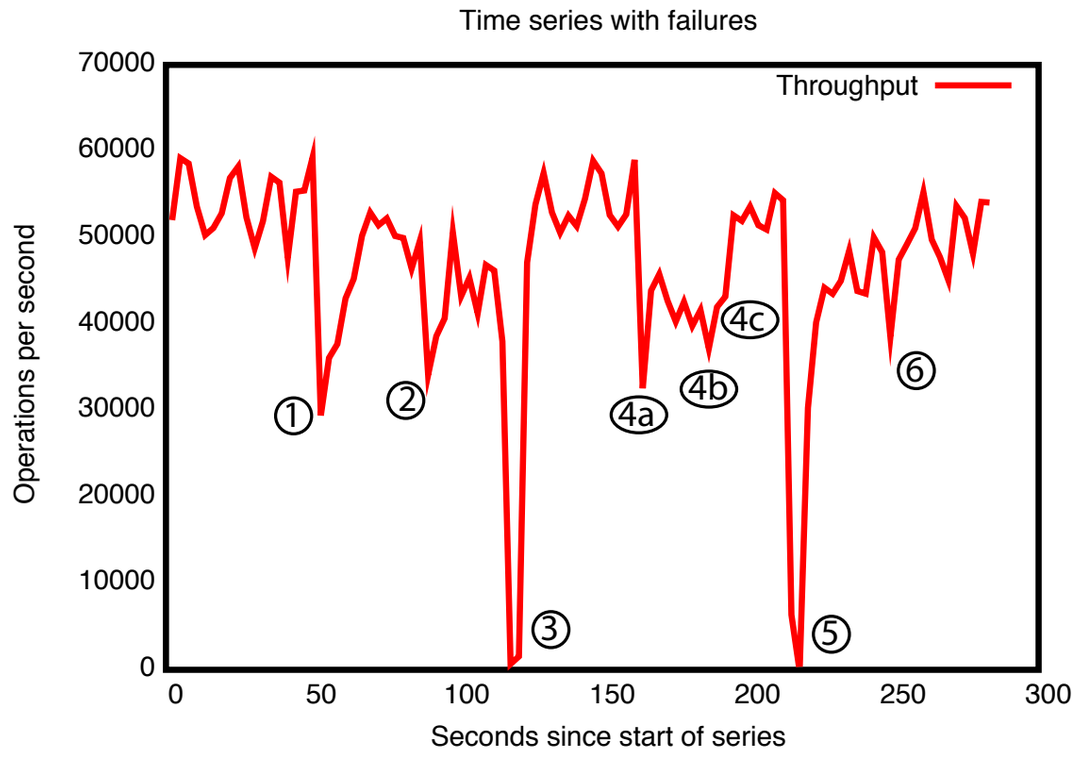
Create, wait, and async delete 50,000 nodes.
With 1K data rather than 5 bytes in Chubby, carrying more coordination information.

Latency scales well:
1.2ms for 3 servers and 1.4ms for 9 servers.

Barrier test: 4:1 Read/Write
non-optimal for the pro-read design
10,700 - 17,000 ops/sec
(full read: 40,000 ops/sec)

Reliability, as tested in simulation

- The events marked in the figure are the following:
- 1. Failure and recovery of a follower;
- 2. Failure and recovery of a different follower;
- 3. Failure of the leader;
- 4. Failure of two followers (a, b) in the first two marks, and recovery at the third mark (c);
- 5. Failure of the leader.
- 6. Recovery of the leader.



Failures not taking down a quorum can be recovered rapidly. Death of leader handled through a brief interruption in service.

Implementing higher level primitives

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)
```

Herd effect-free lock

Automatic SQL-like sequencing of ephemeral nodes provides FIFO lock acquisition.

Polling-free wake up by watching for notification on deletion of the prior locked node.

Ephemeral nodes are deleted even if session ends abnormally, providing automatic error recovery.

Similar to an MCS lock without peer servicing, cache line contention still possible…

Drawbacks (i.e. “different design goals”) (i.e.“future work”)

- Designed specifically for Yahoo!’s needs, and adapted for high read-to-write ratio and trading performance for reliability:
    - Heavy I/O demand on the disk due to full transaction logging
    - Optimised for small meta-data exchange
        - Writers in R/W Locks over ZooKeeper can be blocked by readers?
        - Full path overhead in complex hierarchical znodes
- Watch notification stampeding on consecutive shared resources?
- Workstation hardware used in evaluation was underpowered for 2010.

References

N. Schiper and S. Toueg. A robust and lightweight stable leader election service for dynamic systems. In DSN, 2008.

A. Sherman, P.A. Lisiecki, A. Berkheimer, and J. Wein. ACMS: The Akamai configuration management system. In NSDI, 2005.

M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In Proceedings of the 7th ACM/USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2006.

K.P. Birman. Replication and fault-tolerance in the ISIS system. Vol. 19. No. 5. ACM, 1985.

L. Lamport. "The part-time parliament." ACM Transactions on Computer Systems (TOCS) 16.2 (1998): 133-169.

J. MacCormick, et al. "Boxwood: Abstractions as the Foundation for Storage Infrastructure." OSDI. Vol. 4. 2004.

M. K. Aguilera, et al. "Sinfonia: a new paradigm for building scalable distributed systems." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.

G. DeCandia, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS operating systems review. Vol. 41. No. 6. ACM, 2007.