Yandex

Yandex

ClickHouse Keeper

Alexander Sapin, Software Engineer

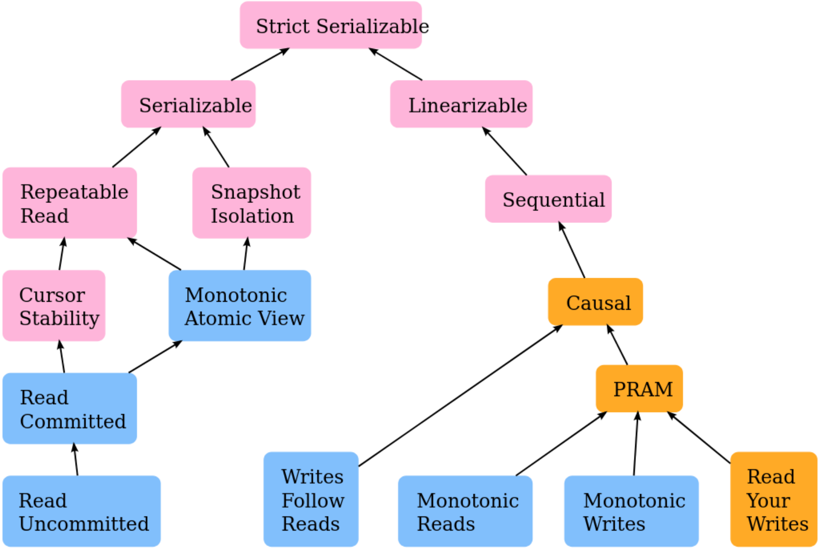
Consensus Problem

- In modern world applications are distributed
  - Multiple independed servers (processes) involved
  - Communication via unreliable network
- Sometimes agreement on some data required
  - Many cases: leader election, load balancing, value increment
  - Failures may happen: network errors, processes failures
  - No reliable clocks exists (in fact not true)
- Required properties
  - Termination – every alive processes agrees some value v;
  - Integrity – if all the alive processes propose value *v*, then any correct process must agree on *v*;
  - Agreement – every alive process must agree on the same value.

Consensus in Real World: State Machine

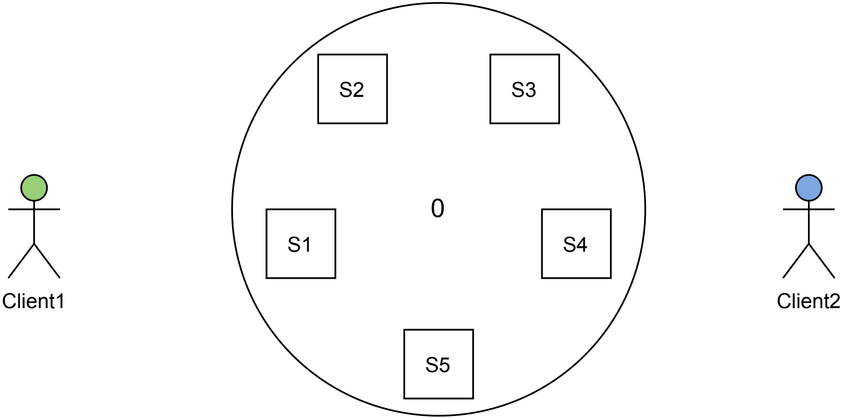
- Agreement on a single value is not enough
  - Consensus algorithms works on state machines
  - State machines has some *state* and *operations* (*transitions*)
  - Often operations stored in *log* and applied to state machine
- Example: distributed hash table
  - State machine – in-memory hash table
  - Operations: *set(key, value)*, *get(key)*
  - Log: *set('x', 5)*, *set('y', 10)*, ...
- Consistency models: history of operations
  - Linearizeability – equal to some sequential order for everyone
  - Sequential consistency – equal to some sequential order for a single process
  - ...

Consistency Models



Consensus in Real World: Example

State machine: `bank account`
Operations: `deposit(x)`, `withdraw(x)`
Log:

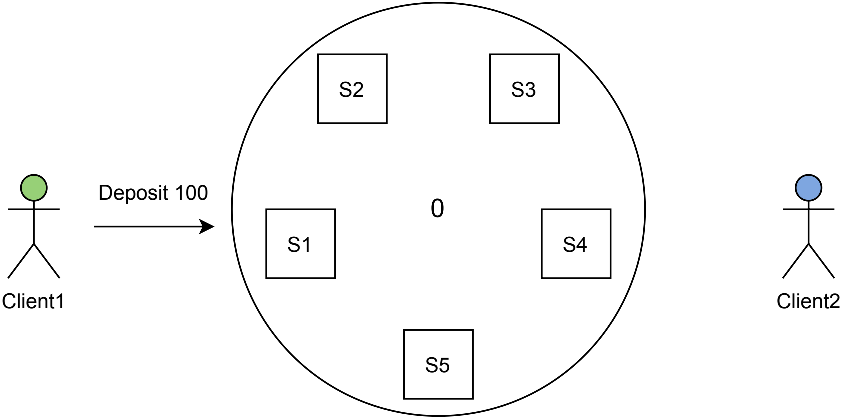


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x)`, `withdraw(x)`

Log:

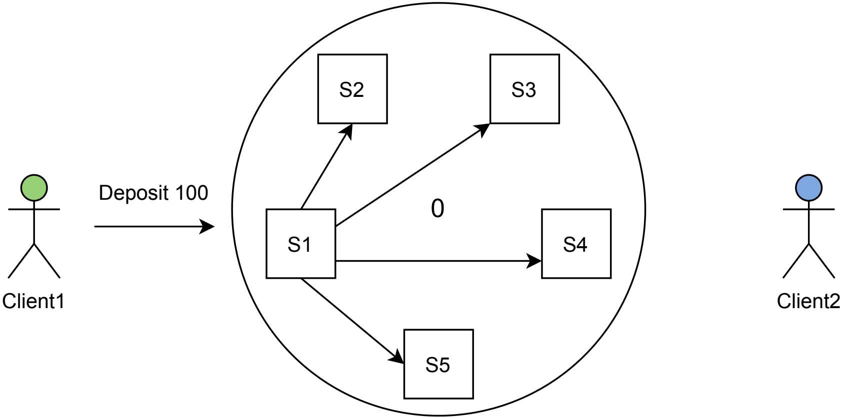


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x), withdraw(x)`

Log:

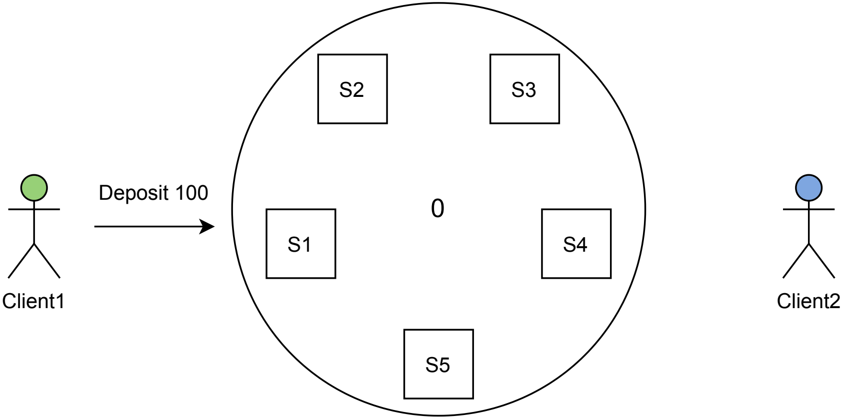


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x), withdraw(x)`

Log: `d(100)`

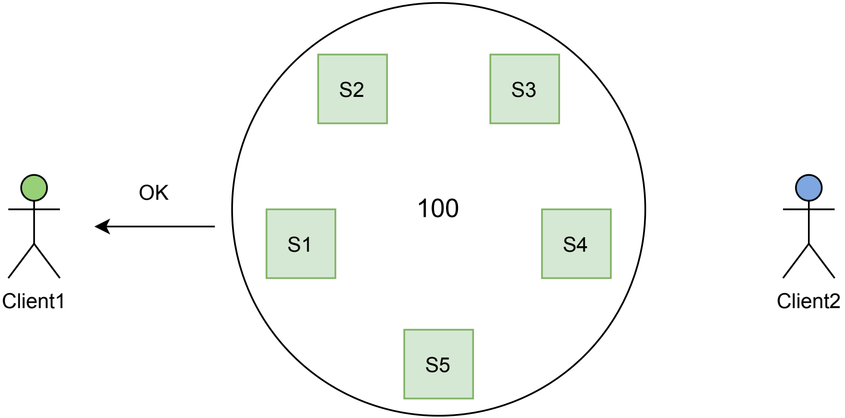


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x)`, `withdraw(x)`

Log: `d(100)`

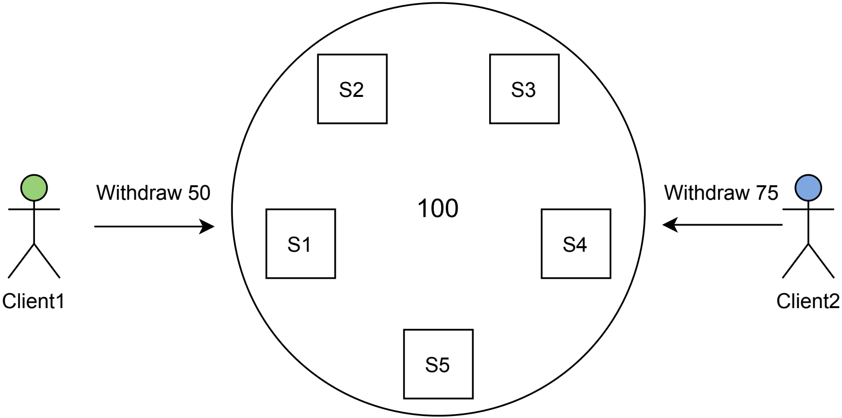


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x), withdraw(x)`

Log: `d(100)`

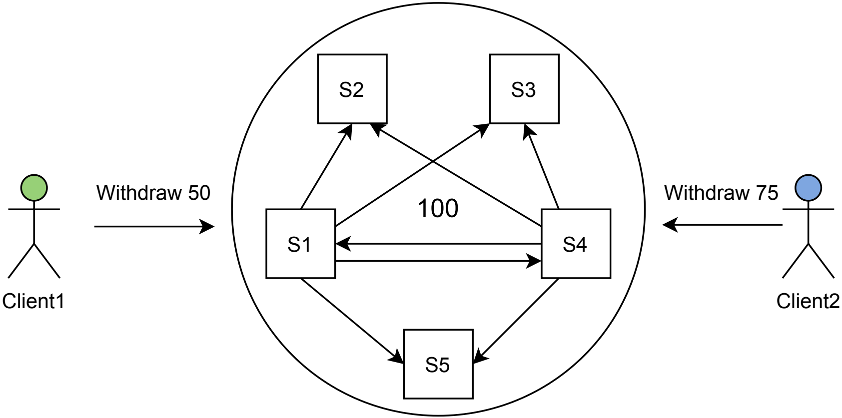


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x), withdraw(x)`

Log: `d(100)`

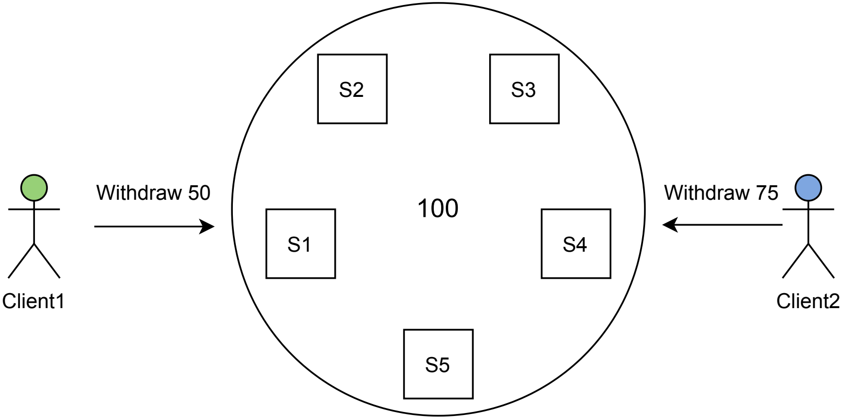


Consensus in Real World: Example

State machine: `bank account`

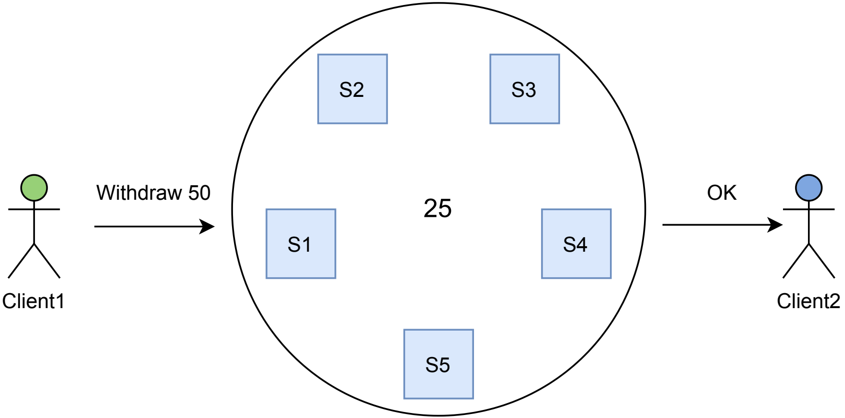
Operations: `deposit(x), withdraw(x)`

Log: `d(100), w(75), w(50)`



Consensus in Real World: Example

State machine: `bank account`
Operations: `deposit(x)`, `withdraw(x)`
Log: `d(100)`, `w(75)`, `w(50)`

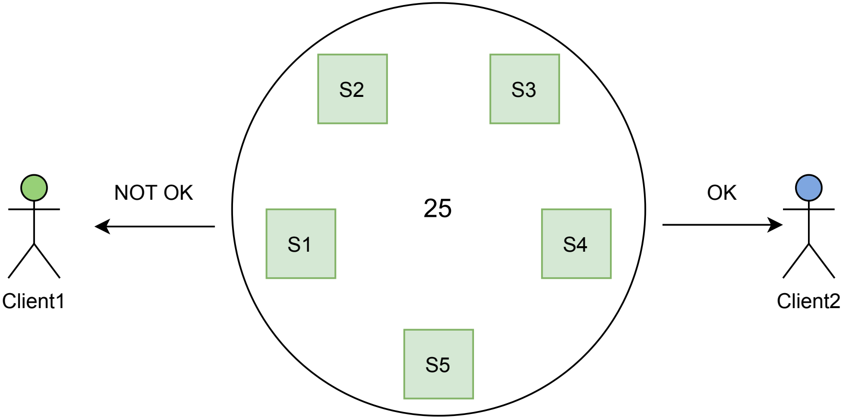


Consensus in Real World: Example

State machine: `bank account`

Operations: `deposit(x), withdraw(x)`

Log: `d(100), w(75), w(50)`



Consensus in Real World: Algos and Apps

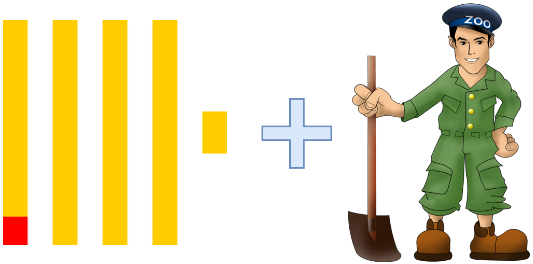
- Some consensus algorithms
  - Paxos, MultiPaxos (around 2000)
  - ZAB (2011)
  - Raft (2014)
- Solves consensus problem:
  - **Coordination:** Chubby, ZooKeeper, etcd, Consul
  - **KV Storage:** DynamoDB, Cassandra, Riak
  - **Distributed Databases:** Spanner, CockroachDB
  - **Stream Processing:** Kafka, Millwheel
  - **Resource Management:** Kubernetes, Mesos

Why ClickHouse needs Consensus?

- Replicated Merge Tree
  - Leader-leader eventually-consistent replication
  - Distributed state machine with own log
  - Consensus: block numbers allocation, merges assignment
- Distributed DDL queries (`ON CLUSTER`)
  - Distributed state machine with own log
  - Sequential queries execution for each shard
  - Consensus: order of queries, executor choice
- Main properties
  - Small amount of data
  - Linearizeability for writes
  - Highly available

Consensus for ClickHouse

- ClickHouse use ZooKeeper for
  - Replicated merge tree metadata
  - DDL queries log storage
  - Distributed notification system
- Why ZooKeeper?
  - Historical reasons
  - Simple and powerful API
  - MultiTransactions
  - Watches
  - Good performance for reads



ZooKeeper Coordination System

- State Machine (Data Model)
  - Filesystem-like distributed hash-table
  - Each node can have both data and children
  - Nodes have stats (version of data, of children, ...)
  - No data types, everything is string
- Client API
  - Own TCP full-duplex protocol
  - Persistent session for each client (unique `session_id`)
- Main operations
  - Read: `get(node)`, `list(node)`, `exists(node)`, `check(node, version)`
  - Write: `set(node, value)`, `create(node)`, `remove(node)`
  - Writes can be combined into MultiTransactions

ZooKeeper Features

- State Machine features
  - Ephemeral nodes – disappear with session disconnect
  - Sequential nodes – unique names with ten digits number
  - Node can be both sequential and ephemeral
- Client API features
  - Watches – server-side one-time triggers
  - Session restore – client can reconnect with the same session_id
- Pluggable ACL + authentication system
  - The most strange implementation I've ever seen

ZooKeeper Internals

Consistency Guarantees

- Linearizeability for write operations
- Sequential consistency for reads (reads are local)
- Atomicity of MultiTransactions
- No rollbacks of commited writes

Consensus Implementation

- Own algorithm: ZooKeeper Atomic Broadcast
- Operations are idempotent and stored in log files on filesystem
- Regular state machine snapshots

Scalability

- Linear for reads (more servers, faster reads)
- Inverse linear for writes (more servers, slower writes)

ZooKeeper Pros and Cons for ClickHouse

Pros:

- Battle tested consensus
- Appropriate data model
- Simple protocol
- Has own client implementation

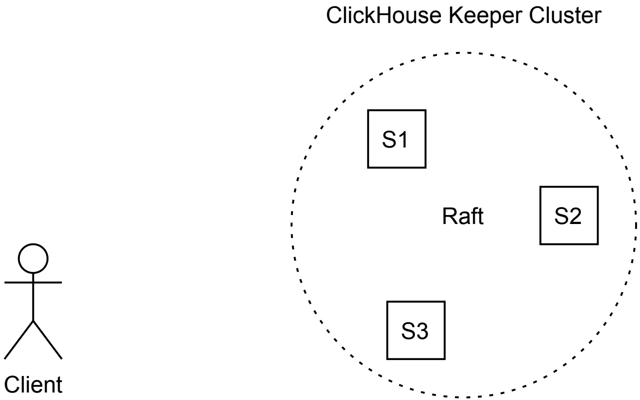
Cons:

- Writen in Java
- Difficult to operate
- Require separate servers
- ZXID overflow
- Uncompressed logs and snapshots
- Checksums are optional
- Unreliable reconnect semantics
- Almost does not develop

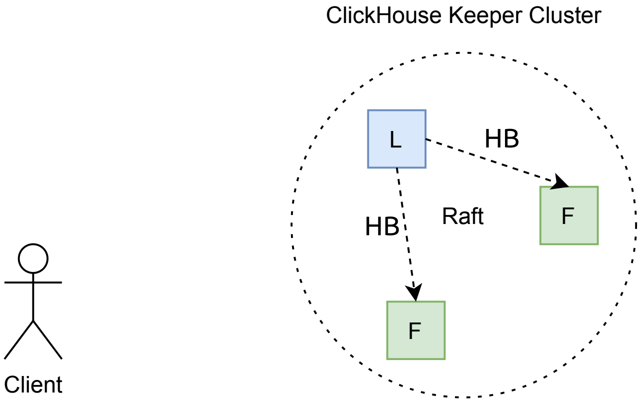
ClickHouse Keeper

- Replacement for ZooKeeper
  - Compatible client protocol (all clients work out of the box)
  - The same state machine (data model)
  - Better guarantees (optionally allows linearizeable reads)
  - Comparable performance (better for reads, similar for writes)
- Implementation
  - Written in C++, bundled into `clickhouse-server` package
  - Uses Raft algorithm (NuRaft implementation)
  - Can be embedded into ClickHouse server
  - Optional TLS for clients and internal communication
- Advantages over ZooKeeper
  - Checksums in logs, snapshots and internal protocol
  - Compressed snapshots and logs

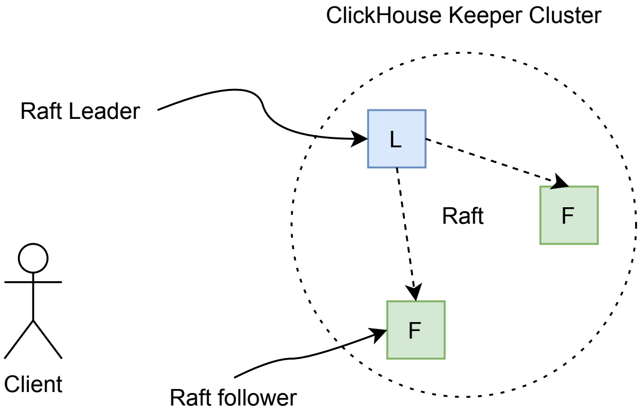
ClickHouse Keeper: In Action



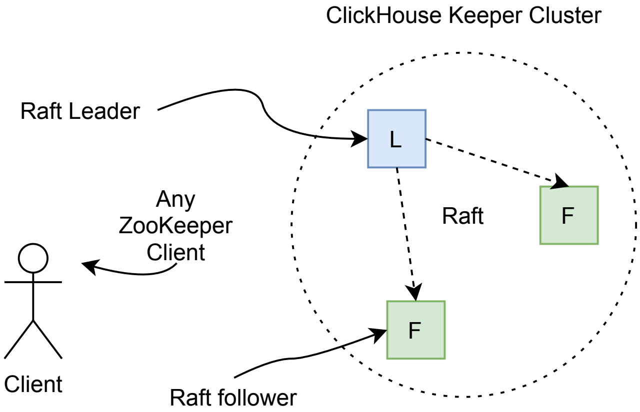
ClickHouse Keeper: In Action



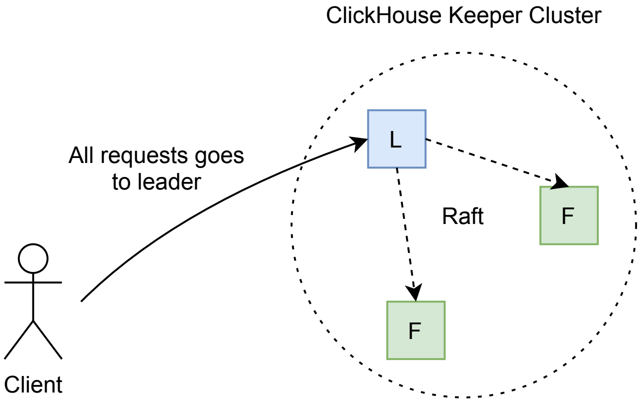
ClickHouse Keeper: In Action



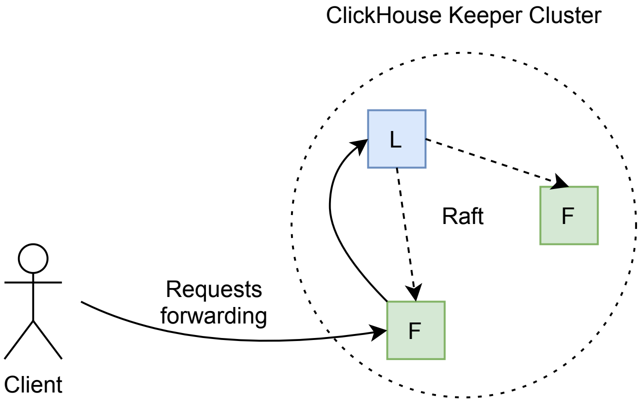
ClickHouse Keeper: In Action



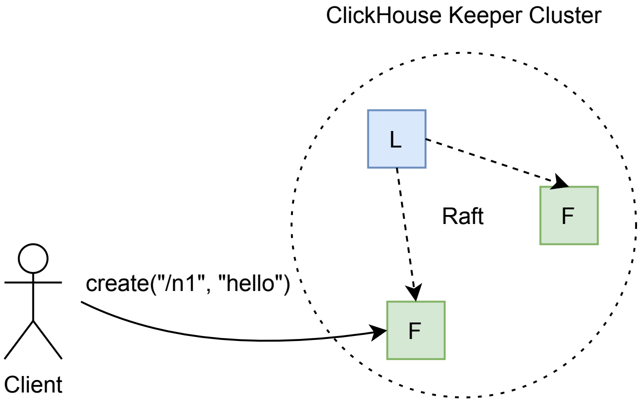
ClickHouse Keeper: In Action



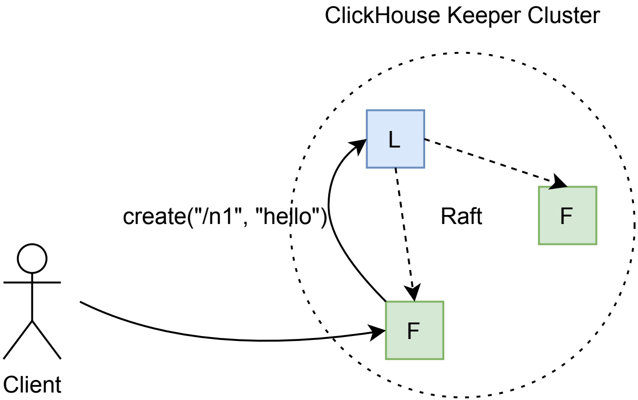
ClickHouse Keeper: In Action



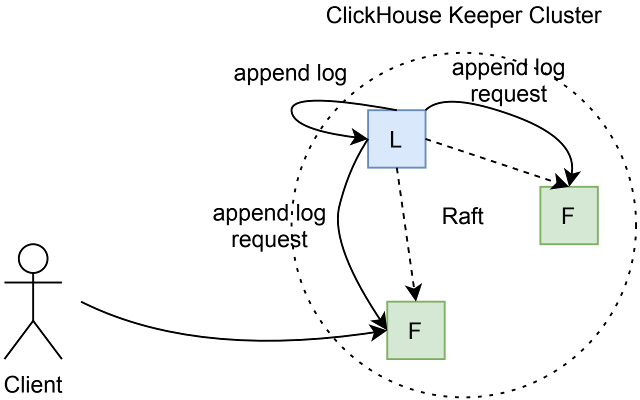
ClickHouse Keeper: In Action



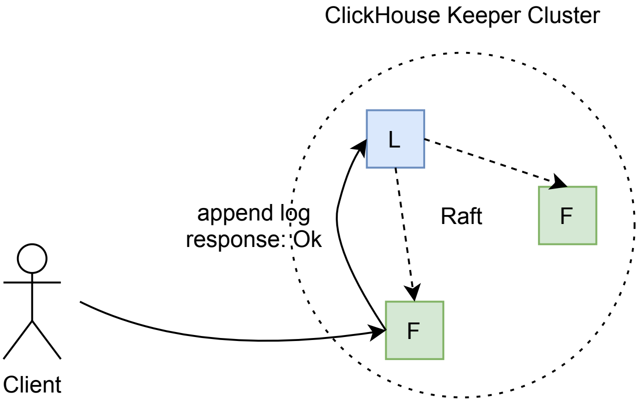
ClickHouse Keeper: In Action



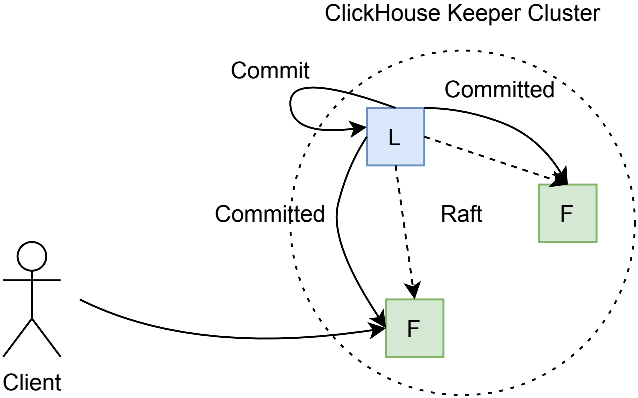
ClickHouse Keeper: In Action



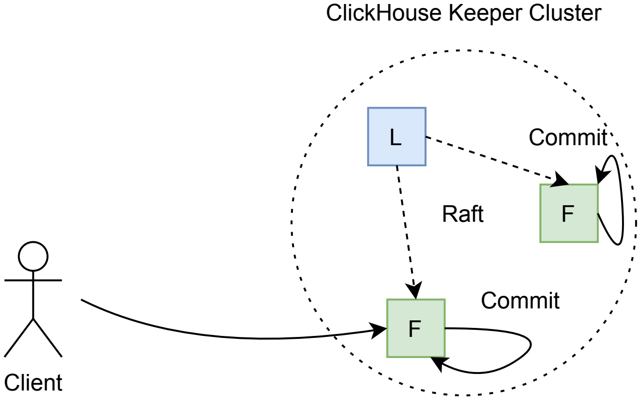
ClickHouse Keeper: In Action



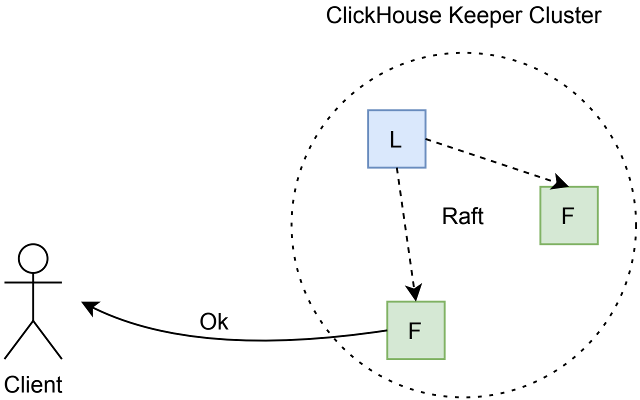
ClickHouse Keeper: In Action



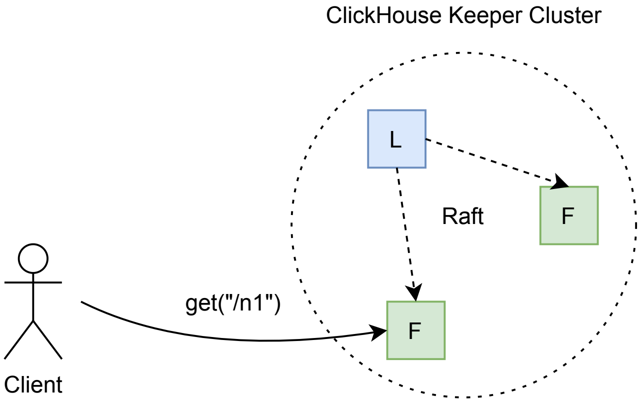
ClickHouse Keeper: In Action



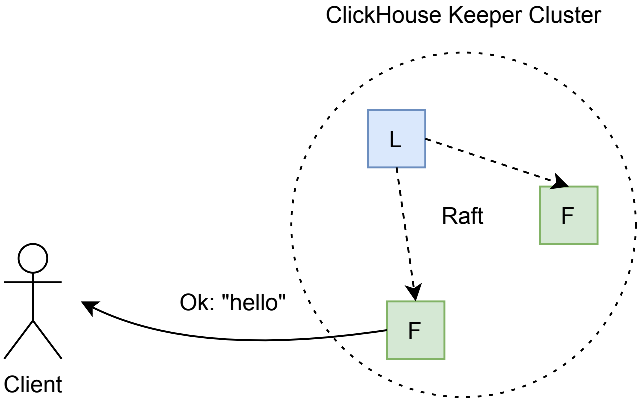
ClickHouse Keeper: In Action



ClickHouse Keeper: In Action



ClickHouse Keeper: In Action



ClickHouse Keeper: Testing

- Ordinary ClickHouse tests
  - Functional tests use `clickhouse-keeper` in single node mode
  - Integration tests use `clickhouse-keeper` in three nodes mode
  - Separate integration tests for basic functionality
- Jepsen tests (`http://jepsen.io/`)
  - General framework for distributed systems testing
  - Written in Clojure with consistency checks
  - Failures: crashes, network partitions, disk corruption, network slow downs
  - More tests than for ZooKeeper
  - About 5 serious bugs found both in NuRaft and our code

ClickHouse Keeper: How to use?

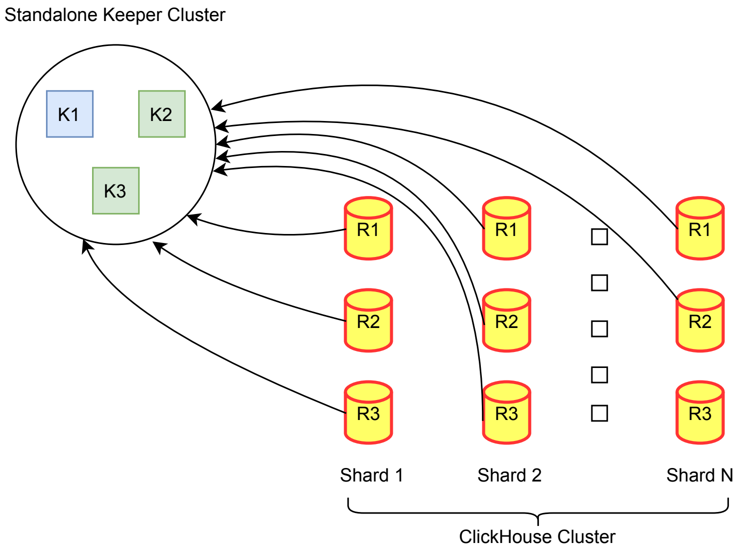
- Two modes
  - As standalone application (`clickhouse-keeper`)
  - Inside `clickhouse-server`
- Configuration
  - Very similar to clickhouse-server `.xml` (or `.yml`) file
  - Must be equal for all quorum participants
- General recommendations
  - Place directory with logs to the independet SSD if possible
  - Don’t try to have more than 9 quorum participants
  - Don’t change configuration for more than 1 server at once
- Run standalone

```
clickhouse-keeper --daemon
  --config /etc/your_path_to_config/config.xml
```

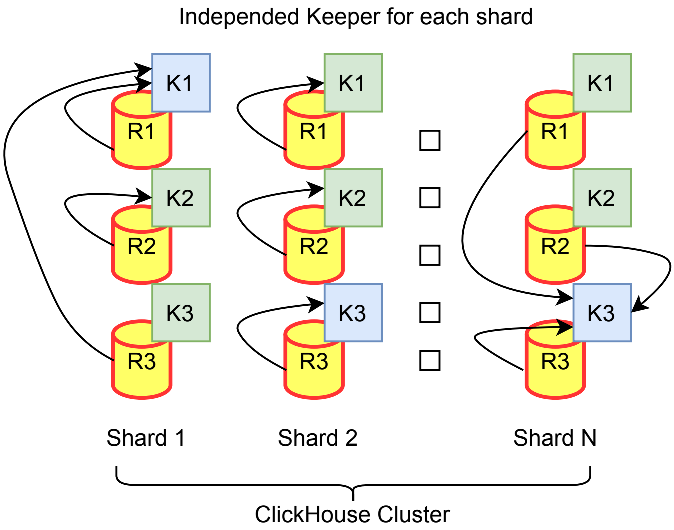
ClickHouse Keeper: Simpliest Configuration

```
<keeper_server>
    <tcp_port>9181</tcp_port>
    <server_id>1</server_id>
    <storage_path>/var/lib/clickhouse/coordination/</storage_path>
    <coordination_settings>
        <force_sync>false</force_sync>
    </coordination_settings>
    <raft_configuration>
        <server>
            <id>1</id>
            <hostname>localhost</hostname>
            <port>44444</port>
        </server>
    </raft_configuration>
</keeper_server>
```

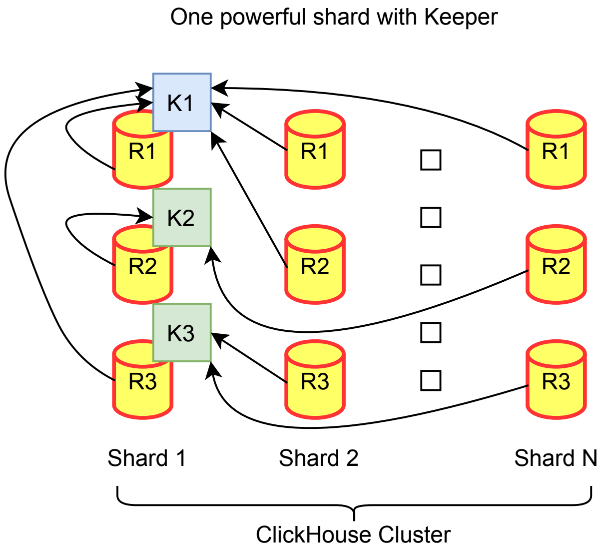
ClickHouse Keeper: Configuration №1



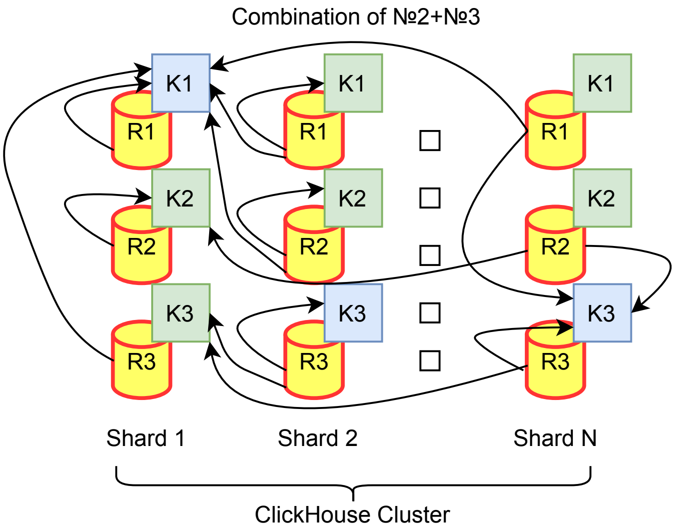
ClickHouse Keeper: Configuration №2



ClickHouse Keeper: Configuration Nº3



ClickHouse Keeper: Configuration №4



ClickHouse Keeper: Some Settings

- If using slow disk or have big network latency try to increase
  - `heart_beat_interval_ms` – how often leader will send heartbeats
  - `election_timeout_lower_bound_ms` – how long followers will wait for HB
  - `election_timeout_upper_bound_ms` – how long followers will wait for HB
- Quorum priorities in `raft_configuration` of server
  - `can_become_leader` – server will be *observer*
  - `priority` – server will become leader more often according to priority
- If you need reads linearizeability [experimental]
  - `quorum_reads` – read requests go through Raft

ClickHouse Keeper: Migration from ZooKeeper

- Separate tool `clickhouse-keeper-converter`
  - Allows to convert ZooKeeper data to `clickhouse-keeper` snapshot
  - Checked for ZooKeeper 3.4+
  - Bundled into `clickhouse-server` package
- How to migrate
  - Stop all ZooKeeper nodes
  - Found leader for migration
  - Start ZooKeeper on leader node and stop again (force snapshot)
  - Run clickhouse-keeper-converter:

```
clickhouse-keeper-converter
    --zookeeper-logs-dir /path_to_zookeeper/version-2
    --zookeeper-snapshots-dir /path_to_zookeeper/version-2
    --output-dir /path/to/clickhouse/keeper/snapshots
```

  - Copy resulted snapshot to all `clickhouse-keeper` nodes

ClickHouse Keeper: Current Status

- Preproduction (available since 21.8)
  - Testing in Yandex.Cloud installations
  - Testing by some experienced users
- What to read
  - Documentation:

    `https://clickhouse.tech/docs/en/operations/clickhouse-keeper/`
  - Integration tests:

    `https://github.com/ClickHouse/ClickHouse/tree/master/tests/integration`
  - NuRaft docs:

    `https://github.com/eBay/NuRaft`
- Next steps
  - Four-letter words introspection interface
  - Compressed logs
  - Elastic quorum configuration

Thank you

QA