Kafka Needs No Keeper

Colin McCabe
Introduction

- Kafka has gotten its mileage out of Zookeeper
- But it is still a second system
- KIP-500 has been adopted by the community
- This is not a 1-1 replacement
- We’ve been headed this direction for years
Evolution of Apache Kafka Clients
Producer
Consumer
Admin Tools
write to
topics

Producer

Consumer

Admin Tools
write to topics

Producer

Consumer

read from topics

Admin Tools
write to topics

Producer

read from topics

Consumer

offset fetch/commit

group partition assignment

Admin Tools
Consumer Group Coordinator
Consumer

- read from topics
- offset fetch/commit
- group partition assignment
Consumer APIs

- Fetch

Consumer

- read from topics
- group partition assignment
- offset fetch/commit
Consumer APIs
- Fetch

Consumer

- read from topics
- group partition assignment
- offset fetch/commit
Consumer APIs

- Fetch
- __offsets

Consumer

- read from topics
- group partition assignment
- offset fetch/commit
Consumer

- group partition assignment
- read from topics

Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch

__offsets
Consumer group partition assignment read from topics

Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch

offset fetch/commit

Consumer

read from topics

group partition assignment

_offsets
Consumer group partition assignment

Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch
Consumer APIs

- Fetch
- OffsetCommit
- OffsetFetch
- JoinGroup
- SyncGroup
- Heartbeat

Consumer

- read from topics
- offset fetch/commit
- group partition assignment

__offsets
Consumer read from topics

Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch
- JoinGroup
- SyncGroup
- Heartbeat

Group partition assignment

Offset fetch/commit

_offsets
Consumer

- read from topics
- group partition assignment
- offset fetch/commit

Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch
- JoinGroup
- SyncGroup
- Heartbeat
Consumer APIs
- Fetch
- OffsetCommit
- OffsetFetch
- JoinGroup
- SyncGroup
- Heartbeat
Producer

Consumer

Admin Tools

create/delete topics
Kafka Security and the Admin Client
Producer

Consumer

Admin Tools

create/delete topics
ACL Enforcement

Producer

Consumer

Admin Tools

create/delete topics
ACL Enforcement

Producer

Consumer

Admin Tools

create/delete topics
ACL Enforcement

Admin Tools

create/delete topics
ACL Enforcement

AdminClient

Admin Tools

create/delete topics
AdminClient 

ACL Enforcement 

Admin APIs: 
- CreateTopics 
- DeleteTopics 
- AlterConfigs 
- ... 

Admin Tools 

create/delete topics
Admin APIs:
- CreateTopics
- DeleteTopics
- AlterConfigs
- ...

ACL Enforcement
Producer

Consumer

AdminClient

Client APIs:
- Produce
- Fetch
- Metadata
- CreateTopics
- DeleteTopics
- ...
Producer

Consumer

AdminClient

Client APIs:
- Produce
- Fetch
- Metadata
- CreateTopics
- DeleteTopics
- ...

- Encapsulation
- Security
- Validation
- Compatibility
Inter Broker Communication
Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller Election
Controller

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller Election
Controller APIs:
- LeaderAndIsr
- UpdateMetadata
- StopReplica

Broker Registration
- ACL Management
- Dynamic Configuration
- ISR Management
- Controller Election
Controller APIs:
- LeaderAndIsr
- UpdateMetadata
- StopReplica

Controller:
- Leader/ISR Push
- Update Metadata
- Stop/Delete Replica

Broker Registration
- ACL Management
- Dynamic Configuration
- ISR Management
- Controller Election
Controller APIs:
- LeaderAndIsr
- UpdateMetadata
- StopReplica

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller Election
Controller APIs:
- LeaderAndIsr
- UpdateMetadata
- StopReplica
- AlterIsr

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller Election
Controller

Controller APIs:
- LeaderAndIsr
- UpdateMetadata
- StopReplica
- AlterIsr

Leader/ISR Push
Update Metadata
Stop/Delete Replica
ISR Management

Broker Registration
ACL Management
Dynamic Configuration
ISR Management
Controller Election
- Encapsulation
- Compatibility
- Ownership
Broker Liveness
/brokers/1 -> {
  host: 10.10.10.1:9092
  rack: rack-1
}

/brokers/1 -> {
    host: 10.10.10.1:9092
    rack: rack-1
}
Broker 1 is offline

Watch trigger
Network Partition Resilience
Case 1: Total partition
Case 2: Broker partition
Case 3: Zk Partition
Case 4: Controller partition
Metadata Inconsistency
Metadata Source of Truth
Metadata Source of Truth

Metadata Cache
- sync writes
- async updates
Metadata Cache
- async updates
- sync writes

Metadata Cache
- async update

Metadata Source of Truth

Metadata Cache
- async update
Last Resort:
> rmr /controller
Last Resort:

> rmr /controller

New controller!
Last Resort:
> rmr /controller

Load ALL Metadata
Last Resort:
> rmr /controller

Load ALL Metadata
Last Resort:
> rmr /controller
Last Resort:
> `rmm /controller`

Push ALL Metadata
Last Resort:
> rmr /controller

Push ALL Metadata

How do you know the metadata has diverged?
Performance of Controller Initialization
New controller!
Load ALL Metadata

Complexity: $O(N)$
$N =$ number of partitions
Push ALL Metadata
Complexity: \(O(N \times M)\)

\(N = \) number of partitions

\(M = \) number of brokers
Metadata as an Event Log
Metadata as an Event Log

- Each change becomes a message
- Changes are propagated to all brokers

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>924</td>
<td>Create topic “foo”</td>
</tr>
<tr>
<td>925</td>
<td>Delete topic “bar”</td>
</tr>
<tr>
<td>926</td>
<td>Add node 4 to the cluster</td>
</tr>
<tr>
<td>927</td>
<td>Create topic “baz”</td>
</tr>
<tr>
<td>928</td>
<td>Alter ISR for “foo-0”</td>
</tr>
<tr>
<td>929</td>
<td>Add node 5 to the cluster</td>
</tr>
<tr>
<td>No.</td>
<td>Action Description</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>924</td>
<td>Create topic &quot;foo&quot;</td>
</tr>
<tr>
<td>925</td>
<td>Delete topic &quot;bar&quot;</td>
</tr>
<tr>
<td>926</td>
<td>Add node 4 to the cluster</td>
</tr>
<tr>
<td>927</td>
<td>Create topic &quot;baz&quot;</td>
</tr>
<tr>
<td>928</td>
<td>Alter ISR for &quot;foo-0&quot;</td>
</tr>
<tr>
<td>929</td>
<td>Add node 5 to the cluster</td>
</tr>
</tbody>
</table>

Metadata as an Event Log

- Clear ordering
- Can send deltas
- Offset tracks consumer position
- Easy to measure lag
Implementing the Controller Log

Can we use the existing Kafka log replication protocol?
  - How do we elect the leader?

We need a self-managed quorum.
Can we use the existing Kafka log replication protocol?
- How do we elect the leader?

We need a self-managed quorum.

Enter Raft.

Leader election is by simple majority.
<table>
<thead>
<tr>
<th></th>
<th>Kafka</th>
<th>Raft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Writes</strong></td>
<td>Single Leader</td>
<td>Single Leader</td>
</tr>
<tr>
<td><strong>Fencing</strong></td>
<td>Monotonically increasing epoch</td>
<td>Monotonically increasing term</td>
</tr>
<tr>
<td><strong>Log reconciliation</strong></td>
<td>Offset and epoch</td>
<td>Term and index</td>
</tr>
<tr>
<td><strong>Push/Pull</strong></td>
<td>Pull</td>
<td>Push</td>
</tr>
<tr>
<td><strong>Commit Semantics</strong></td>
<td>ISR</td>
<td>Majority</td>
</tr>
<tr>
<td><strong>Leader Election</strong></td>
<td>From ISR through Zookeeper</td>
<td>Majority</td>
</tr>
</tbody>
</table>
The Controller Quorum
The Controller Raft Quorum
- The leader is the active controller
- Controls reads / writes to the log
- Typically 3 or 5 nodes, like ZK
Instant Failover
- Low-latency failover via Raft election
- Standbys contain all data in memory
- Brokers do not need to re-fetch

offset=1

offset=2
Metadata Caching

- Brokers can persist metadata to disk
- Only fetch what they need
- Use snapshots if we’re too far behind

/mnt/logs/kafka/metadata
offset=1

/mnt/logs/kafka/metadata
offset=2
Broker Registration
- Building a map of the cluster
- What brokers exist in the cluster?
- How can they be reached?
Broker Registration

- Brokers send heartbeats to the active controller
- The controller uses this to build a map of the cluster
Broker Registration
- Brokers send heartbeats to the active controller
- The controller uses this to build a map of the cluster
- The controller also tells brokers if they should be fenced or shut down
Fencing

- Brokers need to be fenced if they’re partitioned from the controller, or can’t keep up
- Brokers self-fence if they can’t talk to the controller
Handling network partitions
Case 1: Total partition
Case 1: Total partition
Case 2: Broker partition
Case 3:
Controller partition
Case 3: Controller partition
## Deployment

<table>
<thead>
<tr>
<th></th>
<th>Current</th>
<th>KIP-500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File</td>
<td>Kafka and ZooKeeper</td>
<td>Kafka</td>
</tr>
<tr>
<td>Metrics</td>
<td>Kafka and ZK</td>
<td>Kafka</td>
</tr>
<tr>
<td>Administrative Tools</td>
<td>ZK Shell, Four letter words, Kafka tools</td>
<td>Kafka tools</td>
</tr>
<tr>
<td>Security</td>
<td>Kafka and ZK</td>
<td>Kafka</td>
</tr>
</tbody>
</table>
Shared Controller Nodes

- Fewer resources used
- Single node clusters (eventually)
Separate Controller Nodes

- Better resource isolation
- Good for big clusters
Roadmap
Remove Client-side ZK dependencies

Remove Broker-side ZK dependencies

Controller Quorum
Remove Client-side ZK dependencies

Remove Broker-side ZK dependencies

Incremental KIP-4 Improvements
- Create new APIs
- Deprecate direct ZK access

Controller Quorum
Remove Client-side ZK dependencies

Remove Broker-side ZK dependencies

Broker-Side Fixes
- Remove deprecated direct ZK access for tools
- Create broker-side APIs
- Centralize ZK access in the controller

Controller Quorum
Remove Client-side ZK dependencies

Remove Broker-side ZK dependencies

First Release without ZooKeeper

- Raft
- Controller quorum

Controller Quorum
Upgrade Issues

- Tools using ZK
- Brokers accessing ZK
- State in ZK
Bridge Release

- No ZK access from tools, brokers (except controller)
Upgrading
- Starting from the bridge release
Upgrading
- Start new controller nodes (possibly combined)
- Quorum elects leader
- Claims leadership in ZK
Upgrading
- Roll nodes one by one as usual
- Controller continues sending LeaderAndIsr, etc. to old nodes
Upgrading
- When all brokers have been rolled, decommission ZK nodes
Conclusion
Apache ZooKeeper has served us well
- KIP-500 is not a 1:1 replacement, but a different paradigm
We have already started removing ZK from clients
- Consumer, AdminClient
- Improved encapsulation, security, upgradability
Metadata should be managed as a log
- Deltas, ordering, caching
- Controller Failover, Fencing
- Improved scalability, robustness, easier deployment

The metadata log must be self-managed
- Raft
- Controller quorum
It will take a few releases to implement KIP-500
- Additional KIPs for APIs, Raft, Metadata, etc.
Rolling upgrades will be supported
- Bridge release
- Post-ZK release
Kafka needs no Keeper
THANK YOU
Colin McCabe
cmccabe@confluent.io