Moving from Zookeeper to Raft

David Zhu, ChanChan Mao @Alluxio
David Zhu
Tech Lead Manager,
Alluxio PMC

ChanChyan Mao
Community Evangelist
Analytics & AI in the Hybrid & Multi-Cloud Era

Available:

- AWS
- Google Cloud
- Microsoft Azure
- Alibaba Cloud
- Tencent Cloud

Platforms:

- Spark
- Presto
- TensorFlow
- Hive
- Hadoop MapReduce
- PyTorch
- Hudi

Cloud Providers:

- Amazon S3
- Microsoft Azure
- Google Cloud
- Alibaba Cloud
- Tencent Cloud
Open Source Started From UC Berkeley AMPLab in 2014

Join the conversation on Slack alluxio.io/slack

1,200+ contributors & growing

Top 10 Most Critical Java Based Open Source Project

GitHub’s Top 100 Most Valuable Repositories Out of 96 Million

10,000+ Slack Community Members
Inefficient Manual Copy Across Data Centers, Regions, Clouds
Solution

Multi-Cloud Ready Analytics & AI Platform

No-copy data access across silos agnostic to compute engine

Foundation of a heterogeneous data platform across geos
Stateful Distributed Coordination Services
Alluxio

DATA ORCHESTRATION FOR ANALYTICS AND AI

HDFS Interface | Java File API | POSIX Interface | S3 Interface | REST API

HDFS Driver | S3 Driver | GCS Driver | Azure Driver | NFS Driver

AWS S3 | GC Storage | Azure | HDFS | ceph | MinIO | IBM | Net App | DELL EMC | HITACHI | Cloudian
Key features

- Alluxio Caching provides cost saving, faster access
- Long-living cache scales independently of ephemeral compute resources
- Multiple interface support allow chaining of data pipeline across different compute engines / frameworks
- Unified namespace, physical data migration without needing to change application code
Alluxio Architecture

- UFS (Under File System)
  - E.g. Amazon S3, HDFS, etc
  - Object stores
Fault tolerance

- Workers can be replicated, act as a cache for the various UFS
- Many UFS have high availability, fault tolerance guarantees
- Master becomes single point of failure
Journal details

- Total order log of state changes
- Can recover state by replaying the operations
- Snapshots to efficiently store state
  - Faster recovery
  - Smaller size
Basic fault tolerance

- Create a fault tolerant journal
- If the master crashes
  - Start a new master
  - Replay the journal
  - Start serving clients
- Detecting a failure, starting a new node, and replaying the journal takes time
  - The system will be unavailable during this time
Basic high availability

- Run multiple masters
- A primary master will serve requests
- Secondary master(s) will replicate the state of the primary master, and take over in case of failure
Basic highly available/ fault tolerant architecture

Diagram showing the architecture:
- Clients
- Workers
- UFS
- Primary Master
- Secondary Masters
- File system metadata
- Inode tree
- Append
- Read
- Journal
  - 1
  - 2
  - 3
  - 4
  - 5
Problems to solve

- Ensure a single primary master running at all times
- Journal needs to be
  - Fault tolerant
  - Masters must agree on a single valid order of journal entries
    - Consensus
Implementation 1: Zookeeper + UFS Journal
Ensure a single primary master running at a time

- Leader election using Zookeeper
- Apache Zookeeper an open-source server which enables highly reliable distributed coordination
  - Provides a file-system (hierarchical namespace) like abstraction built on top of an Atomic Broadcast (consensus) protocol
  - Run on a cluster of nodes to provide fault tolerance/high availability
Apache Zookeeper Curator leader election recipe

- Each master subscribes to the leader election recipe
- The recipe will elect one of the nodes leader
- If the leader fails or is unable to be reached due to network issues, the recipe will elect a new leader
UFS Journal

- Write journal entries to the UFS
- Use the availability / fault tolerance guarantees of the UFS
Does leader election solve all our problems?

- Not quite
- The Zookeeper recipe will do its best to ensure only a single node is chosen leader at a time, but due to asynchrony in the system two nodes may believe they are leader at the same time
- Multiple nodes may be trying to write to the journal concurrently
  - Note leader election is sufficient to provide consensus, but is not a consensus protocol itself
Defer to the UFS

- Can use the consistency guarantees of the UFS to ensure only a single writer to the journal

- Alluxio Primary/secondary master state transition
Ensure a single journal writer to the UFS

- Use a mutual exclusion protocol based on log file names
- Rename is atomic on HDFS
  - (1, 12) (13, 20) (21, 300) (301, MAX_INT)
- New primary master must ensure current log file is “complete” before writing to a new log
  - (1, 12) (13, 20) (21, 300) (301, 327)
- Create a new log
  - (1, 12) (13, 20) (21, 300) (301, 327) (328, MAX_INT)
Zookeeper + UFS architecture
Issues

- Relies on multiple systems
  - Each having their own fault tolerance/availability models
  - More complicated
- Different UFS have different consistency models and performance
  - May not be efficient for appending log entries
Implementation 2: Raft Journal
The journal as a (deterministic) state machine

- Input: command
  - Append
- State transition (deterministic):
  - Add the journal entry to the log
Raft - replicated state machine

- Clients interact with the state machine as if it was a single instance (linearizability)
  - Clients send commands to the state machine and receive responses
- But the state machine is fault tolerant and has high availability
- **Apache Ratis** (java implementation of Raft)
Forget the Journal as a Log

Key-value store state machine commands:
- Write(key, value)
- Remove(key)
- Read(key) -> value

<table>
<thead>
<tr>
<th>Inode ID</th>
<th>Inode metadata</th>
<th>Worker location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Alluxio metadata as a replicated state machine

- Raft simplifies the task of implementing the journal.
- It handles snapshotting and recovery.
- The journal just becomes a key-value store keeping track of the file-system meta-data.
- Raft is colocated with the Alluxio masters.
RocksDB as the key-value store

- Log-structured merge tree
  - Efficient inserts
- Key-value store
  - Inode tree as a key-value map
- Alluxio provides an additional in-memory cache for fast reads
- Efficient snapshots
Alluxio + Raft architecture
Advantages

- Simplicity
  - No external systems (raft colocated with masters)
  - Journal as a key-value store (higher level abstraction)
  - No longer relying on UFS rename semantics

- Performance
  - Journal operations stored locally on masters
  - RocksDB as an efficient key-value store
Other systems

- Kubernetes - etcd - key/value store run on Raft
- Kafka - coordinator running Zookeeper -> Now on Raft
- FoundationDb - coordinators running Paxos

- Generally a paxos based replicated state machine of a simple data store with some custom application logic running over this
Questions?