ALLUXIO

Scalable Filesystem Metadata Services

Featuring RocksDB

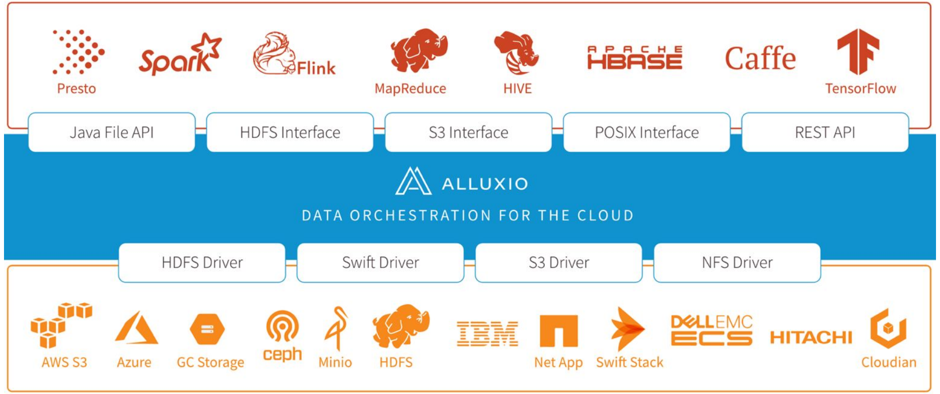
Calvin Jia - 07/11 RocksDB Meetup

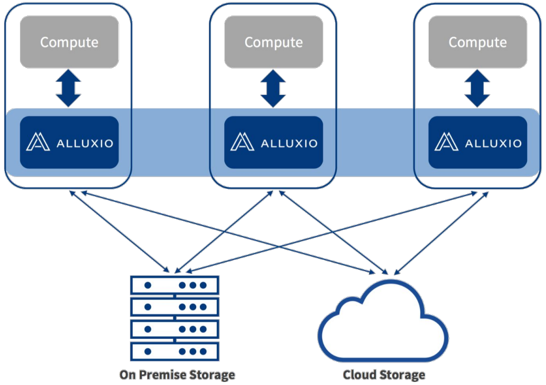
About Me

Calvin Jia

- Release Manager for Alluxio 2.0.0
- Contributor since Tachyon 0.4 (2012)
- Founding Engineer @ Alluxio

Alluxio Overview





- Open source data orchestration
- Commonly used for data analytics such as OLAP on Hadoop
- Deployed at Huya, Two Sigma, Tencent, and many others
- Largest deployments of over 1000 nodes

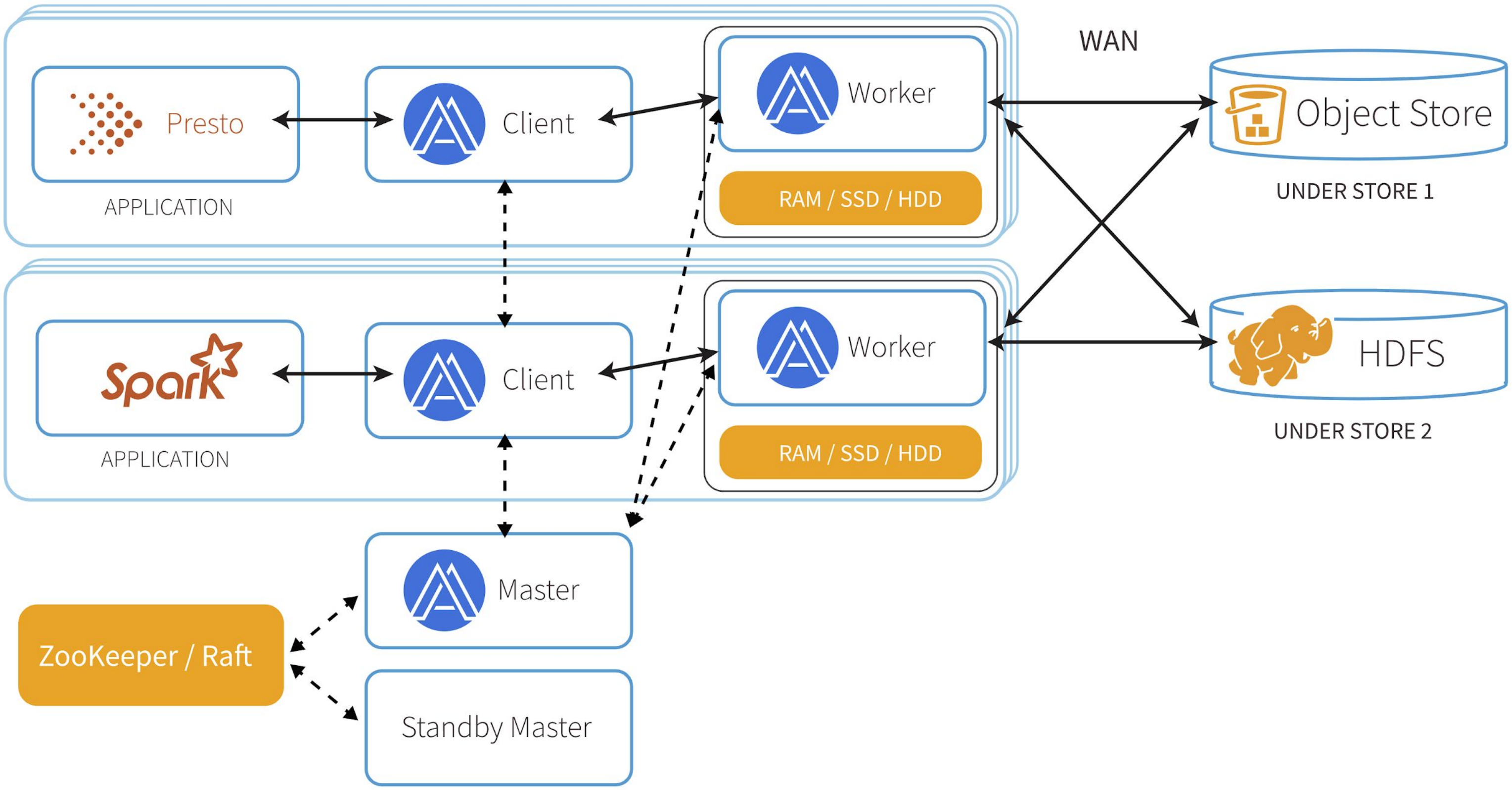
Agenda

1. Architecture
2. Challenges
3. Solutions

ALLUXIO

Architecture

Alluxio Architecture



Alluxio Master

- Responsible for storing and serving metadata in Alluxio
- Alluxio Metadata consists of files and blocks
- Main data structure is the Filesystem Tree
    - The namespace for files in Alluxio
    - Can include mounts of other file system namespaces
    - The size of the tree can be very large!

Challenges

ALLUXIO

Metadata Storage Challenges

- Storing the raw metadata becomes a problem with a large number of files
- On average, each file takes 1KB of on-heap storage
    - 1 billion files would take 1 TB of heap space!
    - A typical JVM runs with < 64GB of heap space
    - GC becomes a big problem when using larger heaps

Metadata Serving Challenges

- File operations (ie. getStatus, create) need to be fast
  - On heap data structures excel in this case
- Operations need to be optimized for high concurrency
  - Generally many readers and few writers

Store 1B+ files while serving at high performance

ALLUXIO

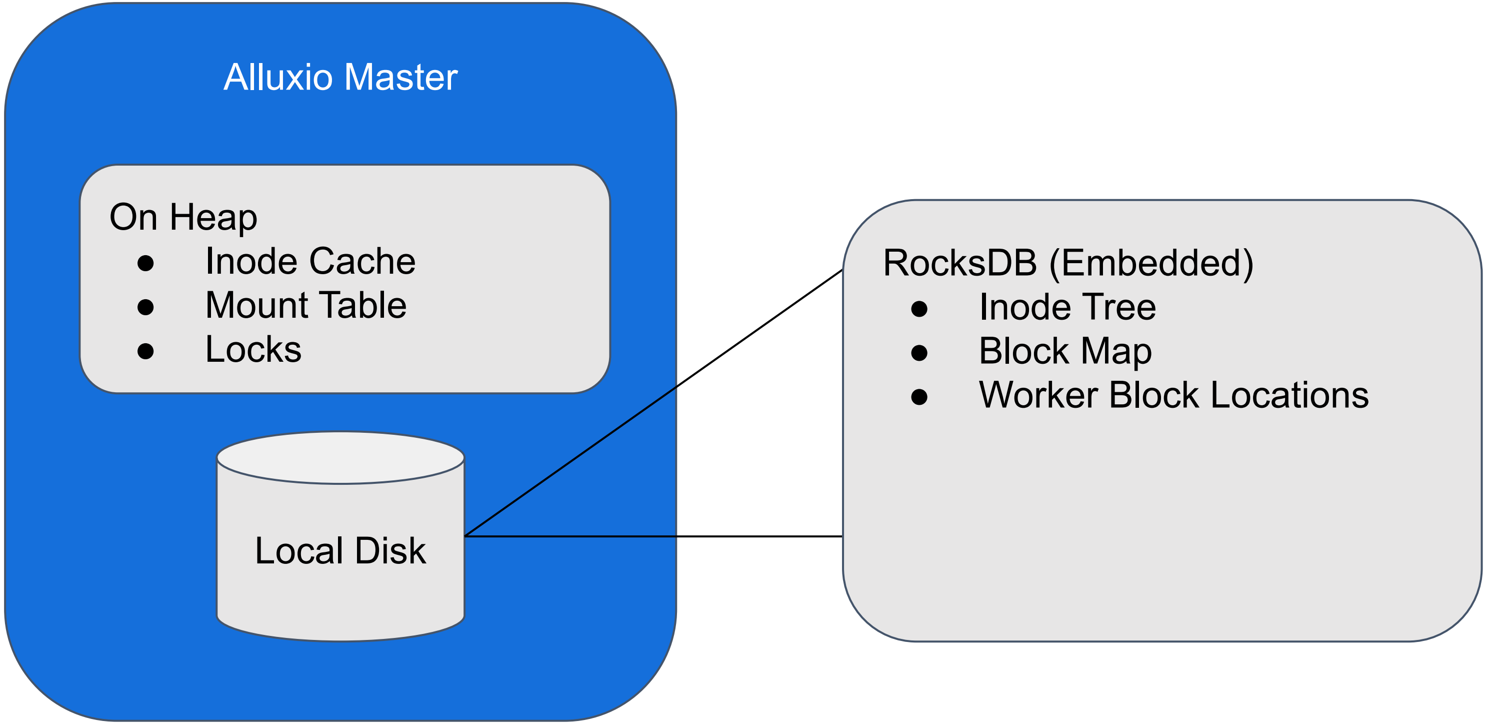
Solutions

ALLUXIO

RocksDB

- Embeddable
- Key-Value interface
- LSMT based storage (sorted)
- Has Java API
- Vibrant community

Tiered Metadata Storage = 1 Billion Files



Working with RocksDB

- Abstract the metadata storage layer
- Redesign the data structure representation of the Filesystem Tree
  - Each inode is represented by a numerical ID
  - Edge table maps <ID,childname> to <ID of child> **Ex: <1foo, 2>**
  - Inode table maps <ID> to <Metadata blob of inode> **Ex: <2, proto>**
- Two table solution provides good performance for common operations
  - One lookup for listing by using prefix scan
  - Path depth lookups for tree traversal
  - Constant number of inserts for updates/deletes/creates

Example RocksDB Operations

- To create a file, /s3/data/june.txt:
  - Look up **<rootID, s3>** in the edge table to get **<s3ID>**
  - Look up **<s3ID, data>** in the edge table to get **<dataID>**
  - Look up **<dataID>** in the inode table to get **<dataID metadata>**
  - Update **<dataID, dataID metadata>** in the inode table
  - Put **<june.txtID, june.txt metadata>** in the inode table
  - Put **<dataId, june.txt>** in the edge table
- To list children of /:
  - Prefix lookup of **<rootId>** in the edge table to get all **<childID>**s
  - Look up each **<childID>** in the inode table to get **<child metadata>**

Effects of the Inode Cache

- Generally can store up to 10M inodes
- Caching top levels of the Filesystem Tree greatly speeds up read performance
    - 20-50% performance loss when addressing a filesystem tree that does not mostly fit into memory
- Writes can be buffered in the cache and are asynchronously flushed to RocksDB
- No requirement for durability - that is handled by the journal

Additional & Future Work

- Fast startup time through using RocksDB checkpoints
- More sophisticated cache management policies

Conclusion

- RocksDB enables us to leverage offheap storage
- Scales our raw metadata storage by an order of magnitude, allowing us to address over 1 billion files
- Available in Alluxio 2.0 - Released June 27th 2019!

Questions?

Alluxio Website - https://www.alluxio.io

Alluxio Community Slack Channel - https://www.alluxio.io/slack

Alluxio Office Hours & Webinars - https://www.alluxio.io/events

ALLUXIO