Caching Framework for Exabyte-Scale Data Lakes
(COMMUNITY OVER CODE 2023)

Jing Zhao, Uber
Hope Wang, Alluxio
Agenda

● Data locality challenges in large-scale data lakes
● Alluxio: An open-source caching framework
  ○ Improve both performance and cost efficiency
  ○ Distributed cache and embedded cache
● Uber: Using cache for exabyte-scale data lake
  ○ HDFS DataNode local cache for high-density HDD adoption
  ○ Presto local cache
The Evolution of the Modern Data Stack

10yr Ago
- Tightly-Coupled MapReduce & HDFS
- On-Prem HDFS
- YARN

Today
- Compute-Storage Separation
- Cloud Data Lake
- K8s/Containerization

More Elastic, Cheaper, Easier to Manage, More Scalable
We are Losing Data Locality

Today

- Compute-Storage Separation
- Cloud Data Lake
- K8s/Containerization

Data locality is missing ...
The Implications of Losing Data Locality

**Slow & Expensive**
- Slow and inconsistent data access performance
- Fast-growing cloud storage costs, including API and egress costs
- High data operation costs when migrating to the cloud

**Complex Platform**
- Data copies, synchronization costs
- Multiple APIs necessitate integration and application rewrites
- On-premise, cloud, hybrid, multi-cloud environments all have different environment properties
10% Of Your Data is Hot Data

Source: Alluxio
Caching’s Values

- Boost Performance
- Prevent Network Congestion
- Save Costs
- Offload Under Storage
Alluxio: A Critical Caching Framework in the Open-Source Data Stack
Open-Source Started From UC Berkeley AMPLab in 2014

1,200+ contributors & growing

11,000+ Slack Community Members

Top 10 Most Critical Java Based Open Source Project

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

Join the conversation on Slack
alluxio.io/slack
Alluxio Technology Journey

EXPLOSION OF DATA
rise of big data & analytics

- Started from UC Berkeley AMPLab
- 1000+ nodes Largest deployment by Baidu

CLOUD ADOPTION
Single to hybrid cloud, multi-cloud, cross region

- 1 Billion Files supported by Alluxio with 2.0 release
- 1000+ Contributors Open Source

GENERATIVE AI
Large-scale model training and deployment

- 9/10 top Internet Co powered by Alluxio
- 7/10 top Internet Co powered by Alluxio
- 100% Presto @ Meta Fully on-boarded to Alluxio
- Zhihu LLM Model training served by Alluxio

1000+ Attendees Data Orchestration Summit

2014

2019

2023
A Caching Framework to Fit Different Needs

Alluxio Embedded/Local Cache
- Run as a library in the application processes (Presto, HDFS DataNode)
- Leverage local disk NVMe or memory
- When the size of hot data fits local disks

Alluxio Distributed Cache
- Standalone cache service shareable across applications
- Cache capacity scales horizontally
Multi-level Caching
L1 Embedded Cache + L2 Distributed Cache
Alluxio Embedded Cache in Presto

- Battle tested in Uber, Meta, Tiktok and etc.
- Support Iceberg, Hudi, Delta Lake and Hive tables
- Support varied file format such as Parquet, ORC and CSV
- Fully optimized for local NVMe storages

Presto Worker

HDFS API Calls

Alluxio Caching File System

On Cache Miss

External File System

On Cache Hit

Alluxio Cache Manager

External Storage

Local cache storage
Alluxio Embedded Cache Data Management

Alluxio embedded cache provides cache eviction and admission

- Support LRU and FIFO cache eviction policy
- Support customized cache admission policy
- Support TTL
- Support data quota
TPC-DS Benchmark of Presto+Embedded Cache

- Accelerate 70 out of 73 queries that last > 2s
- Total running time q1-q99 improved by 63% on average
Alluxio Distributed Cache

Diagram showing the integration of Alluxio with Compute, HDFS, and Object Store.
Uber: Using Cache for Exabyte-scale Data Lake
Data informs **every decision** at Uber

- Community Operations
- Marketplace Pricing
- Eats
- Compliance
- Growth Marketing
- Data Science
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusters</td>
<td>30</td>
</tr>
<tr>
<td>Regions</td>
<td>2</td>
</tr>
<tr>
<td>Nodes</td>
<td>11K</td>
</tr>
<tr>
<td>Data Footprint</td>
<td>1.5EB</td>
</tr>
</tbody>
</table>
Adopting High-Density HDD in HDFS

- Capacity per Host: 4TB * 24 → 16TB * 35
- Efficiency: >50% HW cost reduction
- Challenges
  - DataNode IO performance
  - HDFS reliability (blast radius)
- Opportunities
  - Traffic focuses on a small group of extremely hot blocks
  - Top 10K blocks attracted >90% read traffic

<table>
<thead>
<tr>
<th>Host</th>
<th>Total reads</th>
<th>Total writes</th>
<th>Number of Blocks stored in the host</th>
<th>Number of blocks being read</th>
<th>Average Block Size</th>
<th>Capacity Usage</th>
<th>Read traffic on top 10k blocks (1 hour time window)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host1</td>
<td>13.5M</td>
<td>3.3 K</td>
<td>1,074,622</td>
<td>84769</td>
<td>380 MB</td>
<td>77%</td>
<td>89%</td>
</tr>
<tr>
<td>Host2</td>
<td>12.8M</td>
<td>4.7 K</td>
<td>633,923</td>
<td>59376</td>
<td>330 MB</td>
<td>79.89%</td>
<td>94%</td>
</tr>
<tr>
<td>Host3</td>
<td>11.3M</td>
<td>275 K</td>
<td>479,544</td>
<td>49317</td>
<td>300 MB</td>
<td>79.86%</td>
<td>91%</td>
</tr>
<tr>
<td>Host4</td>
<td>8.5M</td>
<td>4.6 K</td>
<td>247,206</td>
<td>31048</td>
<td>300 MB</td>
<td>82.85%</td>
<td>99%</td>
</tr>
<tr>
<td>Host5</td>
<td>14.3M</td>
<td>45 K</td>
<td>463,819</td>
<td>79958</td>
<td>160 MB</td>
<td>81.62%</td>
<td>99%</td>
</tr>
</tbody>
</table>
DataNode Local Cache

- Build a local cache within DataNode
  - 4TB NVMe SSD disk
  - Based on DataNode local traffic
- Leverage **Alluxio** for cache management
  - Page-level cache
  - 1MB default page size matches traffic pattern
- Exclude Non-Client Reads
Support Append to a Block

- HDFS blocks may not be immutable due to Append ops
- Leverage HDFS generation stamp to achieve “snapshot isolation”
Rate Limiter

- 1:140 SSD-to-HDD ratio
- Need mechanisms to control what data can be loaded into Cache
  - Cache hit rate
  - SSD write endurance
- Track block access frequency within a sliding time window

```
<table>
<thead>
<tr>
<th>Blk ID</th>
<th># of access</th>
</tr>
</thead>
<tbody>
<tr>
<td>blk01</td>
<td>1</td>
</tr>
<tr>
<td>blk03</td>
<td>10</td>
</tr>
<tr>
<td>blk40</td>
<td>50</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Blk ID</th>
<th># of access</th>
</tr>
</thead>
<tbody>
<tr>
<td>blk01</td>
<td>10</td>
</tr>
<tr>
<td>blk05</td>
<td>5</td>
</tr>
<tr>
<td>blk10</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Blk ID</th>
<th># of access</th>
</tr>
</thead>
<tbody>
<tr>
<td>blk02</td>
<td>15</td>
</tr>
<tr>
<td>blk15</td>
<td>3</td>
</tr>
<tr>
<td>blk20</td>
<td>30</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
Evaluation and Rollout

- Performance evaluation: DataNode trace replay
  - Test with scale
  - Reproduce production traffic profile
- Alluxio cache “shadow mode”
  - A simulated cache mode to collect metrics
Current Status

- Deployed in all major production clusters (1200 DataNodes)

Effectively reduced the # of processes blocked by IO
Current Status

Offload 60% of IO from HDD

Cache read throughput is significantly higher, nearly twice that of non-cache read throughput.
Presto @ Uber

2
Regions

7K
Weekly Active Users

15
Clusters

5K
Nodes

500K
Queries/day

90PB
HDFS bytes read/day
Workloads

Interactive
Ad hoc queries

Batch
Scheduled

Interactive Presto Cluster

Batch Presto Cluster
Leverage Presto workers’ local NVMe SSD disks
Selective caching based on table
Leverage Alluxio for cache management in Presto worker
Forced cache TTL to meet compliance requirements
Key Challenges @ Uber and Solutions

- **Realtime Partition Updates**
  - **Problem**: Tables/partitions are constantly changing, naive caching could cause returning stale data.
  - **Solution**: Integrated HDFS file modification time as part of cache key

- **Cluster Membership Change**
  - **Problem**: Presto nodes always leave and join cluster, causing file read to route to wrong node and thus cache miss.
  - **Solution**: Introduced consistent hashing to tie files to certain Presto worker nodes.
Key Challenges @ Uber and Solutions

- **Cache Size Restriction**
  - **Problem:** Available cache space is limited compared to total data that needs to be scanned.
  - **Solution:** Introduced cache filter to selectively cache only certain hot data (top accessed tables).

- **Tiny Reads**
  - **Problem:** Uber’s Presto traffic often involves a series of tiny consecutive reads, leading to degraded behavior in Alluxio.
  - **Solution:** Introduced buffer reads in Alluxio caching.
Presto Local SSD Cache Onprem - Today

**Deployment**
- Deployed to Presto production since 2022
- Clusters run with local cache
  - 5 clusters (out of 11 batch clusters)
  - 1500 nodes (43% of all batch nodes in primary region)

**Improvement**
- ~13% Presto batch HDFS read offloaded to cache
- Input read latency reduced by ~44% compared to HDFS reads
Presto Local SSD Cache On Cloud - Initial Evaluation

- Deployed to Presto on cloud for initial evaluation
- Cost reduction AND performance improvement
  - >80% reduction of # of read requests to GCS
  - 228 s → 50 s reduction of P90 query latency