ALLUXIO

Cloud Native Data and Model Access Management for AI

Chunxu Tang, Shawn Sun @ Alluxio

Chunxu Tang

Staff Research Scientist
Presto committer
chunxu.tang@alluxio.com

Shawn Sun

Software Engineer
Fluid committer
shawn.sun@alluxio.com

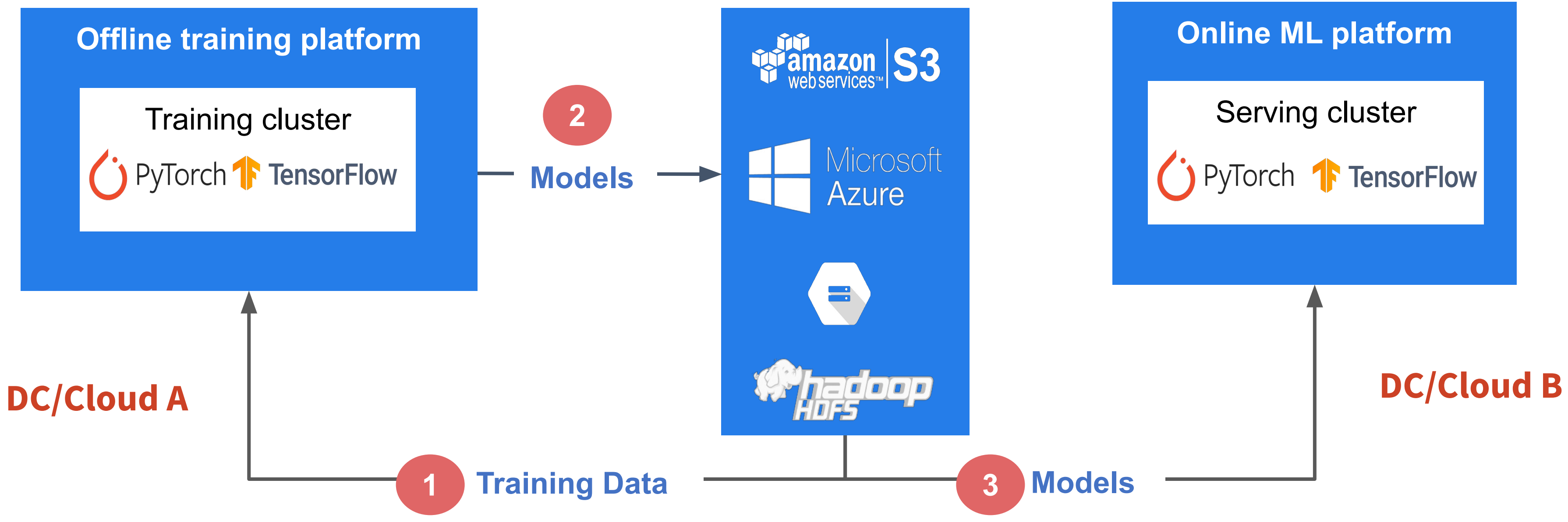
ALLUXIO

Agenda

1. ML in the Cloud
2. Accessing Data and Models in the Cloud
3. A New Design with Alluxio
4. Cloud-Native Alluxio Kubernetes Operator
5. Alluxio CSI-FUSE Driver on Kubernetes
6. Data Access Management for PyTorch
7. Data Access Management for Ray
8. Use Cases

ML in the Cloud

Hybrid/Multi-Cloud ML Platforms



Separation of compute and storage

Data/Model Access Patterns

	Data Ingestion	Data Preprocessing		Model Training			Model Deployment	Model Inference
		Unstructured or Semi Structured	Structured	CV	NLP	Checkpoint Write		
Type of Access	Mostly write	Both read and write	Both read and write	Mostly Read	Mostly Read	Write Only	Mostly Read	Read Only
Access Mode - Read	N/A	Sequential Read	Random Read(4k)	Sequential Read	Random Read(4k)	N/A	Sequential Read	Sequential Read
Access Mode - Write	Sequential Write or Append	Sequential Write or Append	Sequential Write or Append	N/A	N/A	Sequential Write or Append	Sequential Write	N/A
File Size	Small to Large	Small to Large	Medium to Large	Small	Large	Large	Small to Large	Small to Large
Number of Files	Small to Medium	Many	Small	Massive	Small	Small	Small	Small
File Format	Parquet, ORC, Avro, Arrow	jpeg, gif, json or text, mp4	Parquet or ORC	Unstructured data, like jpeg	Structured or semi-structured data	NPZ, HDF5, tf-native	pb, pkl, h5, onnx, mlmodel	pb, pkl, h5, onnx, mlmodel
Requirements for Data & AI Platform	• High throughput • Combine all data sources	• High throughput (batch processing) • Low latency (real-time processing) • High CPU utilization		• High throughput • High read performance • High GPU utilization		• High throughput • High write performance	• Low latency • High concurrency	• Low latency • High throughput • High availability

Data Access Patterns

	Data Ingestion	Data Preprocessing		Model Training			Model Deployment	Model Inference
		Unstructured or Semi Structured	Structured	CV	NLP	Checkpoint Write		
Type of Access	Mostly write	Both read and write	Both read and write	Mostly Read	Mostly Read	Write Only	Mostly Read	Read Only
Access Mode - Read	N/A	Sequential Read	Random Read(4k)	Sequential Read	Random Read(4k)	N/A	Sequential Read	Sequential Read
Access Mode - Write	Sequential Write or Append	Sequential Write or Append	Sequential Write or Append	N/A	N/A	Sequential Write or Append	Sequential Write	N/A
File Size	Small to Large	Small to Large	Medium to Large	Small	Large	Large	Small to Large	Small to Large
Number of Files	Small to Medium	Many	Small	Massive	Small	Small	Small	Small
File Format	Parquet, ORC, Avro, Arrow	jpeg, gif, json or text, mp4	Parquet or ORC	Unstructured data, like jpeg	Structured or semi-structured data	NPZ, HDF5, tf-native	pb, pkl, h5, onnx, mlmodel	pb, pkl, h5, onnx, mlmodel
Requirements for Data & AI Platform	• High throughput • Combine all data sources	• High throughput (batch processing) • Low latency (real-time processing) • High CPU utilization		• High throughput • High read performance • High GPU utilization		• High throughput • High write performance	• Low latency • High concurrency	• Low latency • High throughput • High availability

ALLUXIO

Model Access Patterns

	Data Ingestion	Data Preprocessing Unstructured or Semi Structured	Data Preprocessing Structured	Model Training CV	Model Training NLP	Model Training Checkpoint Write	Model Deployment	Model Inference
Type of Access	Mostly write	Both read and write	Both read and write	Mostly Read	Mostly Read	Write Only	Mostly Read	Read Only
Access Mode - Read	N/A	Sequential Read	Random Read(4k)	Sequential Read	Random Read(4k)	N/A	Sequential Read	Sequential Read
Access Mode - Write	Sequential Write or Append	Sequential Write or Append	Sequential Write or Append	N/A	N/A	Sequential Write or Append	Sequential Write	N/A
File Size	Small to Large	Small to Large	Medium to Large	Small	Large	Large	Small to Large	Small to Large
Number of Files	Small to Medium	Many	Small	Massive	Small	Small	Small	Small
File Format	Parquet, ORC, Avro, Arrow	jpeg, gif, json or text, mp4	Parquet or ORC	Unstructured data, like jpeg	Structured or semi-structured data	NPZ, HDF5, tf-native	pb, pkl, h5, onnx, mlmodel	pb, pkl, h5, onnx, mlmodel
Requirements for Data & AI Platform	• High throughput • Combine all data sources	• High throughput (batch processing) • Low latency (real-time processing) • High CPU utilization		• High throughput • High read performance • High GPU utilization		• High throughput • High write performance	• Low latency • High concurrency	• Low latency • High throughput • High availability

Accessing Data and Models In the Cloud

Existing Solutions

Data access:

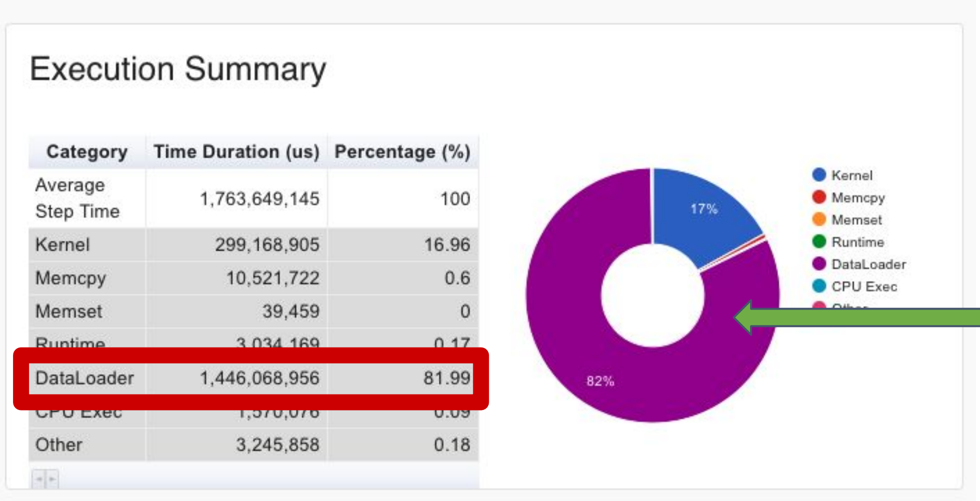
1. Read data directly from cloud storage
2. Copy data from cloud to local before training
3. Local cache layer for data reuse
4. Distributed cache system

Model access:

1. Pull models directly from cloud storage

Always Read From Cloud Storage

- Easy to set up
- Performance are not ideal
  - Model access: Models are repeatedly pulled from cloud storage
  - Data access: Reading data can take more time than actual training



Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	1,763,649,145	100
Kernel	299,168,905	16.96
Memcpy	10,521,722	0.6
Memset	39,459	0
Runtime	3,034,169	0.17
DataLoader	1,446,068,956	81.99
CPU Exec	1,570,076	0.09
Other	3,245,858	0.18

82% of the time spent by DataLoader

Copy Data To Local Before Training

- Data is now local
  - Faster access + less cost
- Management is hard
  - Must manually delete training data after use
- Local storage space is limited
  - Dataset is huge - limited benefits

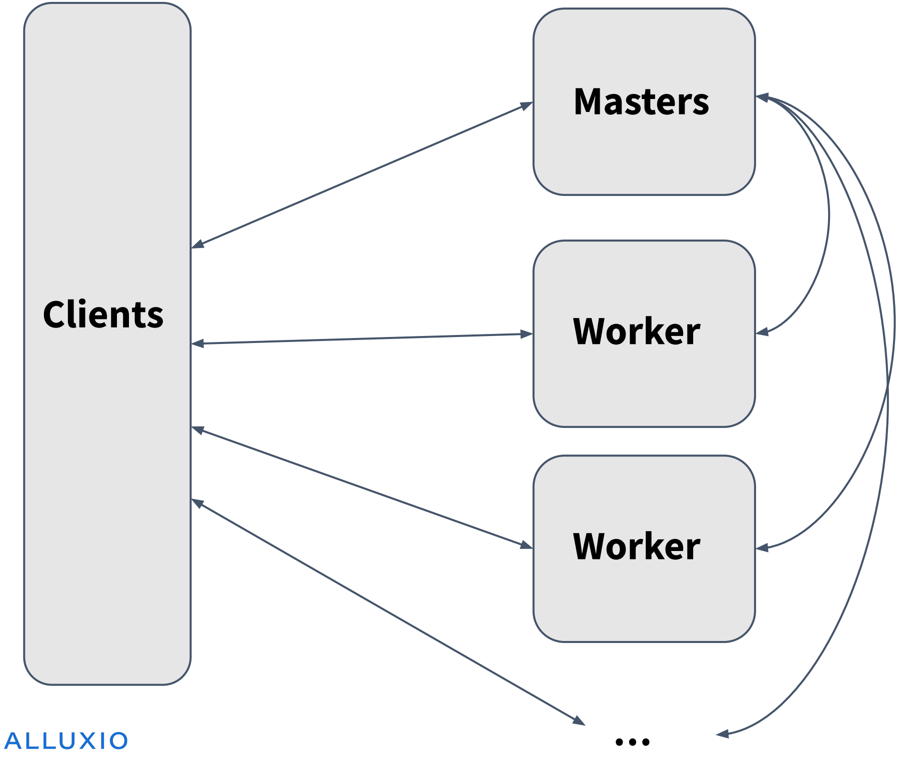
Local Cache Layer for Data Reuse

Examples: S3FS built-in local cache, Alluxio Fuse SDK

- Reused data is local
  - Faster access + less cost
- Cache layer provider helps data management
  - No manual deletion/supervision
- Cache space is limited
  - Dataset is huge - limited benefits

Legacy Distributed Cache System

Alluxio 2.x



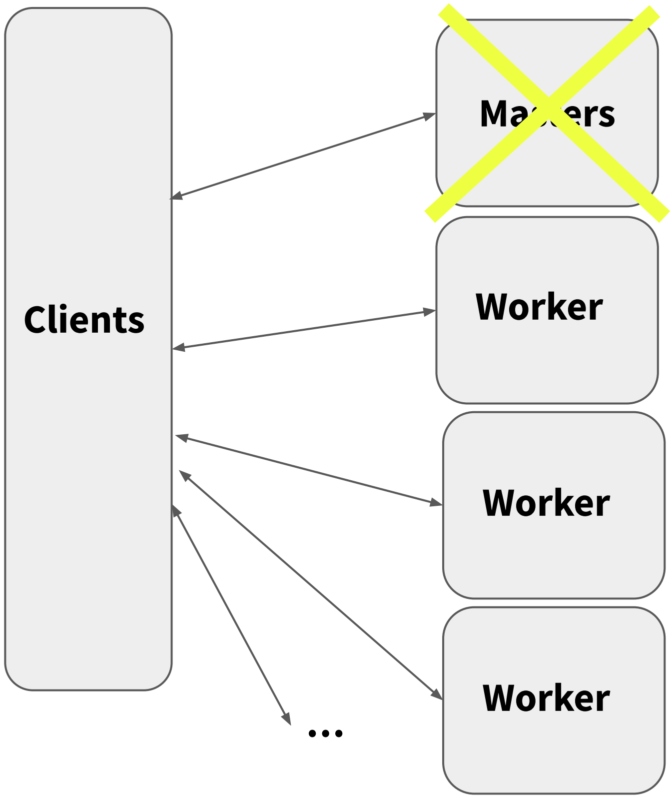
- Training data and trained models can be kept in cache - unified solution.
- Data management functionalities.
- Masters are “single” point of failure.
- The huge number of files makes masters the bottleneck of the overall performance.

Challenges

1. Performance
    - Pulling data from cloud storage is hurting training/serving.
2. Cost
    - Repeatedly requesting data from cloud storage is costly.
3. Reliability
    - Availability is the key for every service in cloud.
4. Data Management
    - Manual work is unfavorable.

A New Design with Alluxio

Consistent Hashing for caching



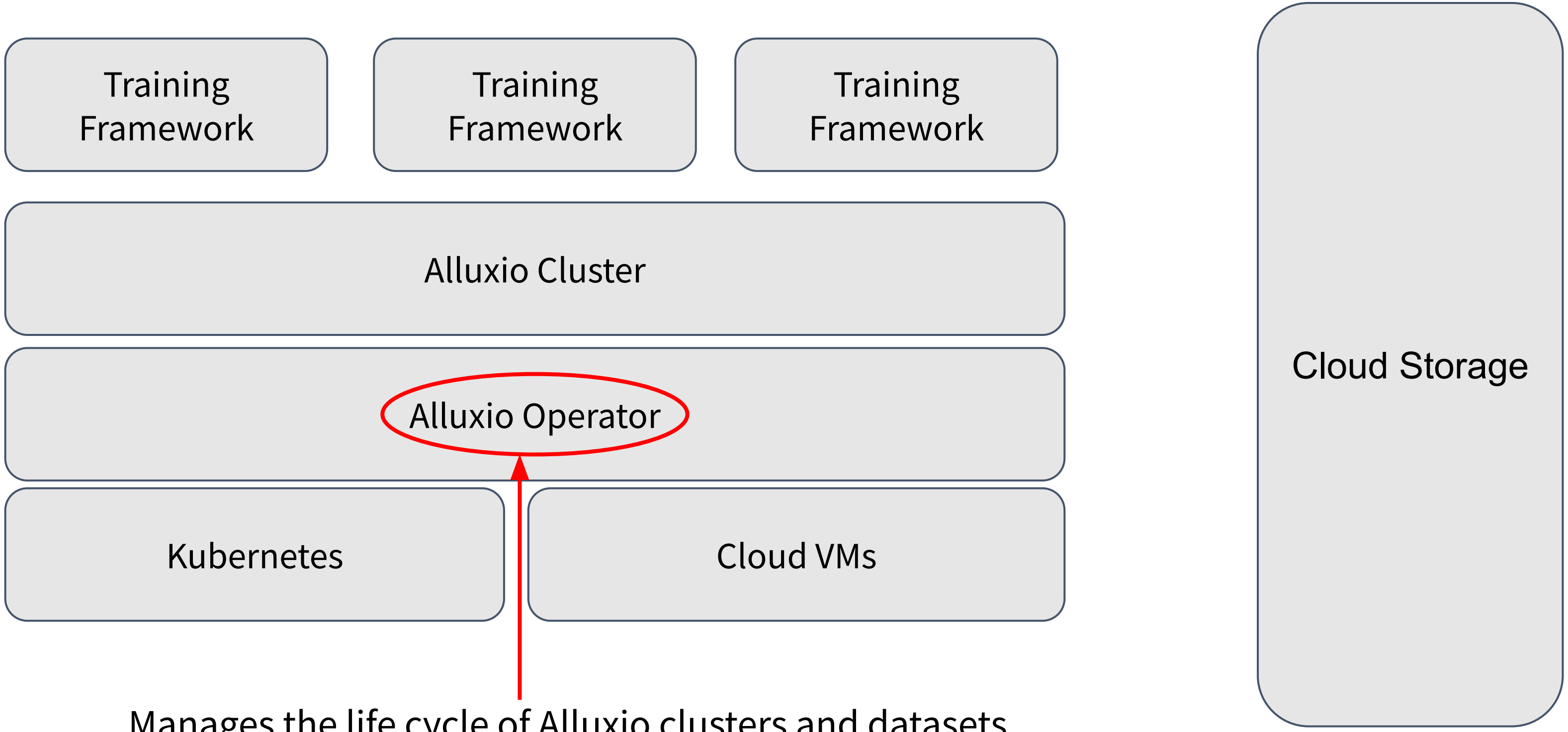
- Use **consistent hashing** to cache both data and metadata on workers.
- Worker nodes have plenty space for cache. Training data and models only need to be pulled once from cloud storage. **Cost --**
- No more single point of failure. **Reliability ++**
- No more performance bottleneck on masters. **Performance ++**
- Data management system.

Alluxio 3xx

- High Scalability
  - One worker supports 30 - 50 million files
  - Scale linearly - easy to support 10 billions of files
- High Availability
  - 99.99% uptime
  - No single point of failure
- High Performance
  - Faster data loading
- Cloud-native K8s Operator and CSI-FUSE for data access management

Cloud-Native Alluxio Kubernetes Operator

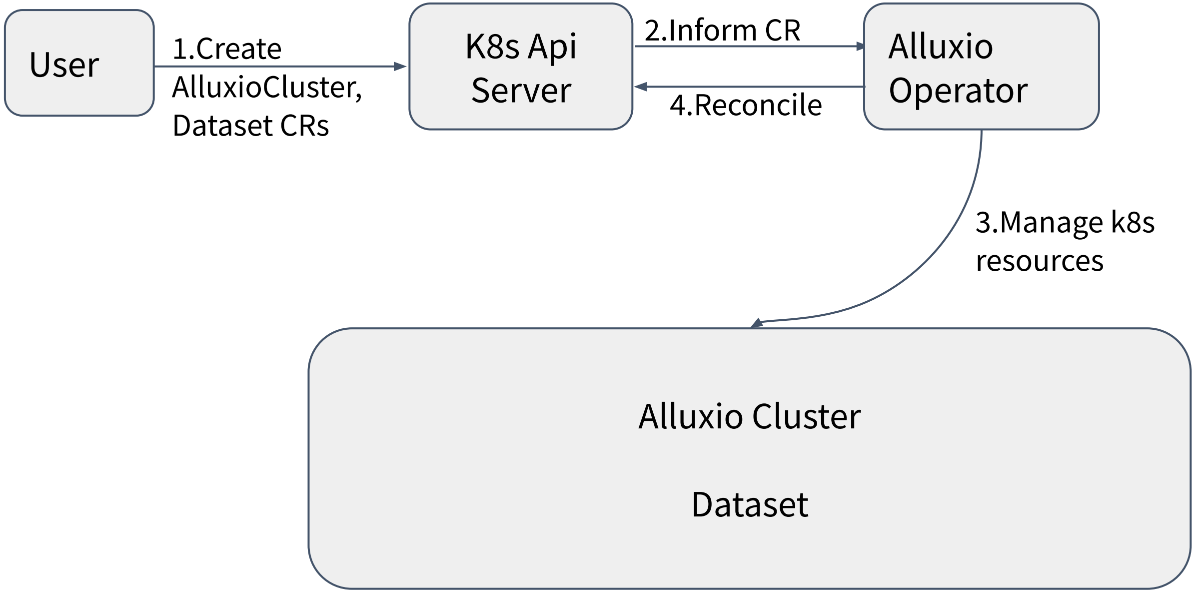
Alluxio Operator



Manages the life cycle of Alluxio clusters and datasets

Alluxio Cluster CRD

Alluxio Operator follows the Kubernetes Operator pattern



- **Zero-downtime Upgrade**
- **High-availability**
- **Auto-scaling**

Fully Managed Cache

```
alluxio_client.load(path)
```

1. Datasets/Models are loaded into the Alluxio cluster
2. Old data get evicted if cache space is full according to eviction policy - no manual work
3. Alluxio server will read and cache from cloud storage if there is any cache miss

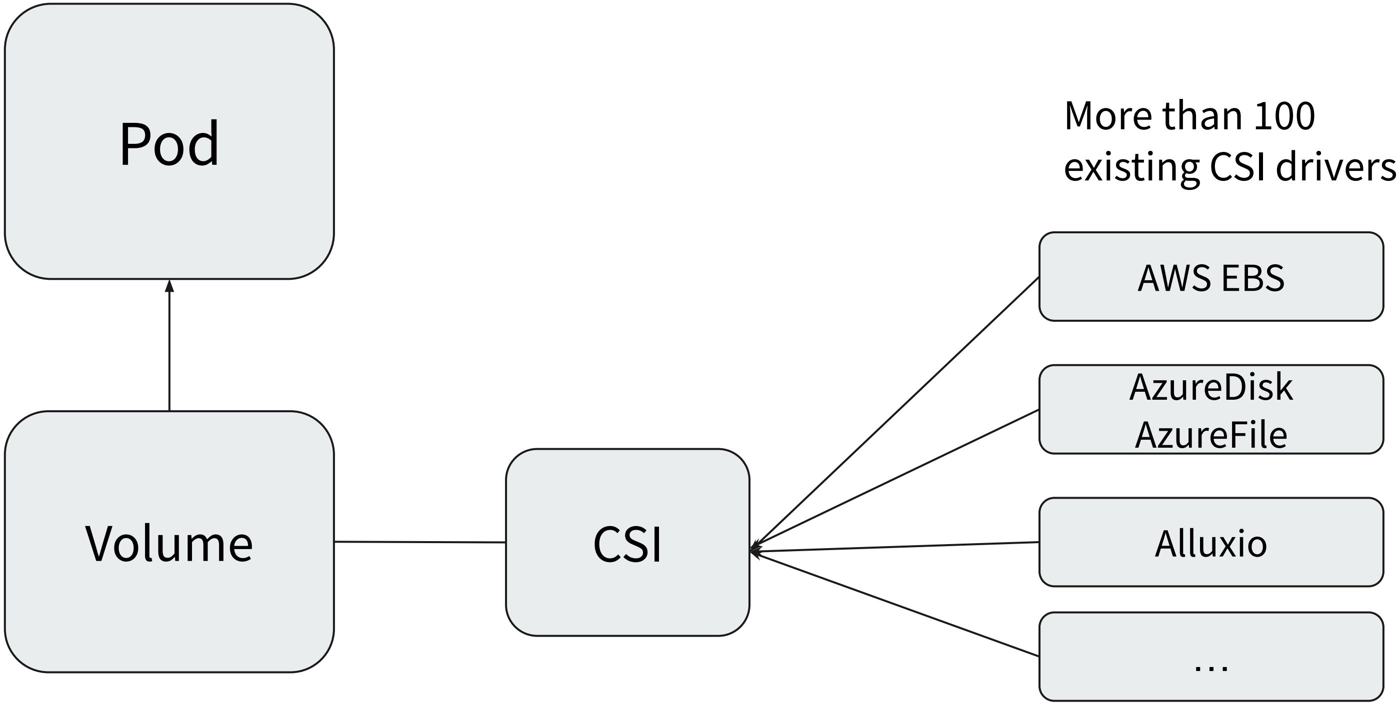
Alluxio CSI-FUSE Driver on Kubernetes

Alluxio FUSE

- Expose the Alluxio file system as a local file system.
- Can access the cloud storage just as accessing local storage.
  - `cat, ls`
  - `f = open("a.txt", "r")`
- Very low impact for end users

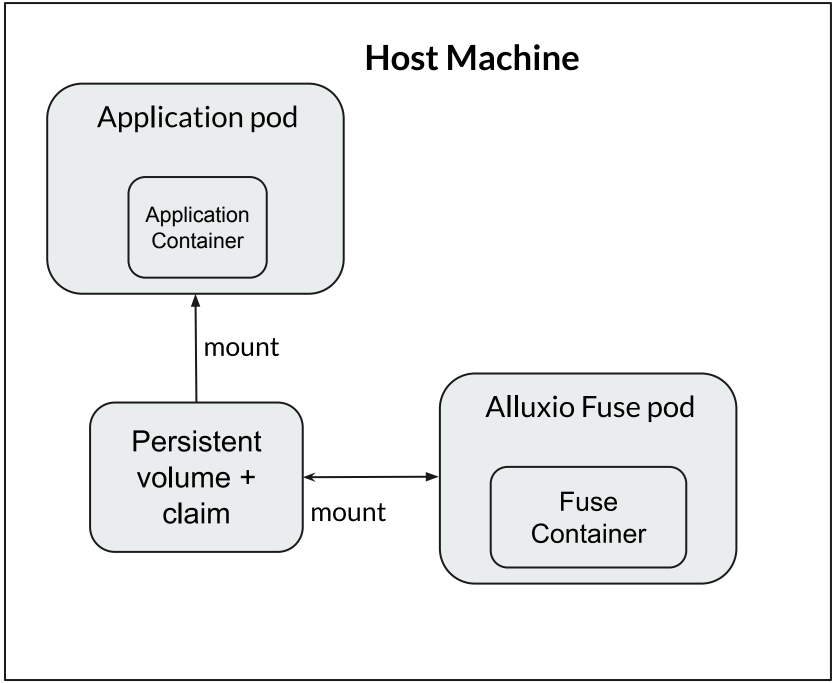
```
root@alluxio-fuse-ip-10-0-4-38:/mnt/alluxio/fuse-alluxio-5256dac9-b7b3-412f-ae74-429aeea1789f# ls
 a.txt                   edge         sample.txt           test.txt
 alluxio                 h5test.h5    small-dataset        wwm_uncased_L-24_H-1024_A-16
 alluxio_backups         imagenet     stress-master-base   x.txt
 default_tests_files     overmind     test                ''$'\346\265\213\350\257\225\346\226\207\344\273
```

CSI For Kubernetes



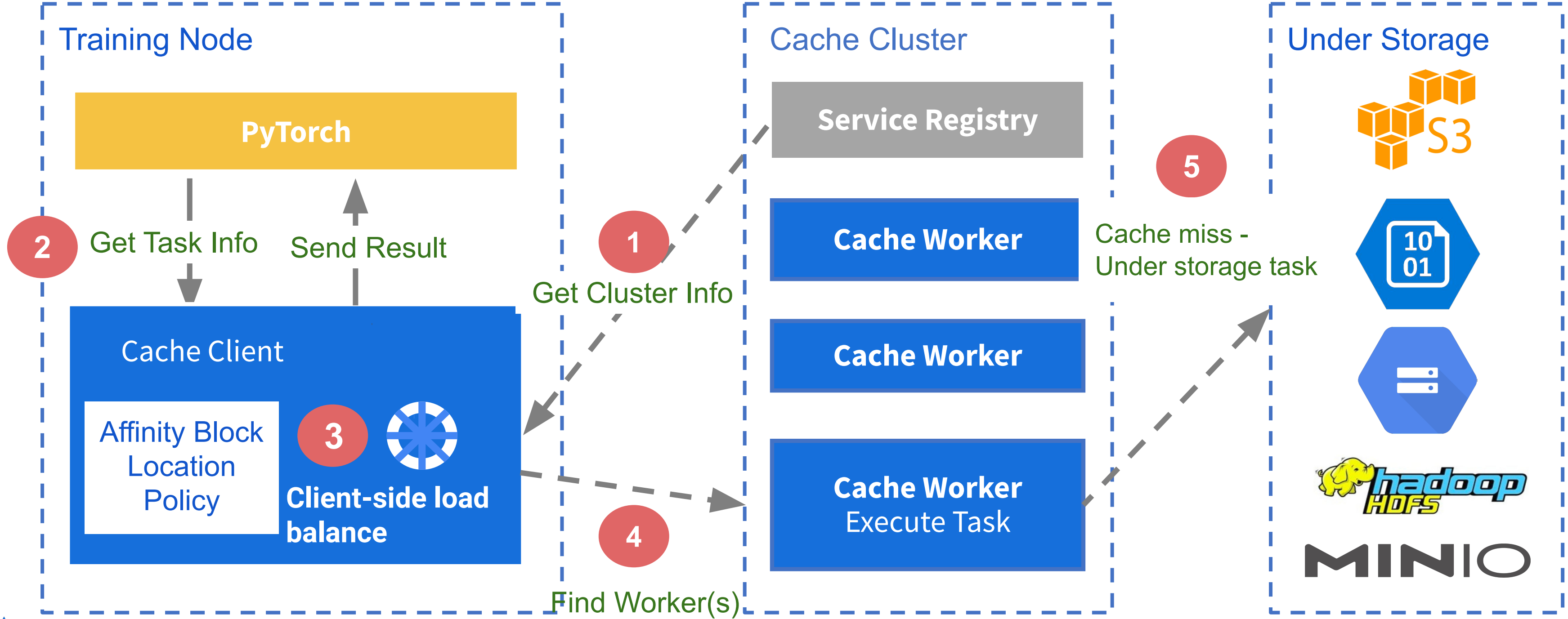
Alluxio CSI x Alluxio FUSE for Data Access

- FUSE: Turn remote dataset in cloud into local folder for training
- CSI: Launch Alluxio FUSE pod only when dataset is needed
- Three layers of caching
  - kernel cache - Kernel Fuse
  - local cache - Alluxio Fuse
  - distributed cache - Alluxio Server

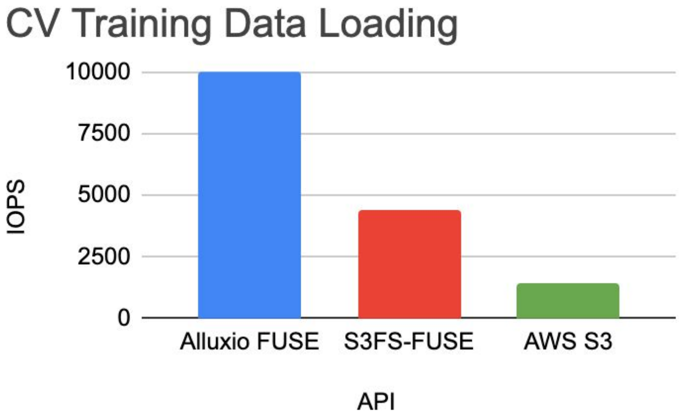


Data Access Management for PyTorch

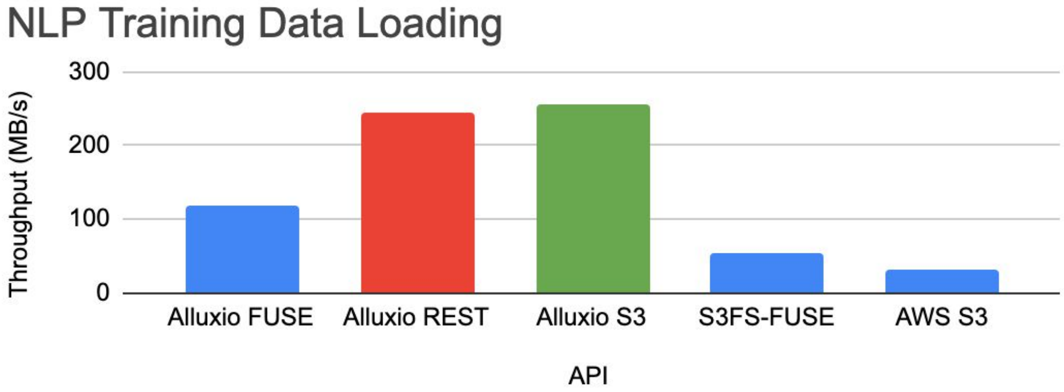
Integration with PyTorch Training (Alluxio)



Data Loading Performance



ImageNet (subset)



Yelp review

GPU Utilization Improvement

Training Directly from Storage (S3-FUSE)

- > 80% of total time is spent in DataLoader
- Result in Low GPU Utilization Rate (<20%)

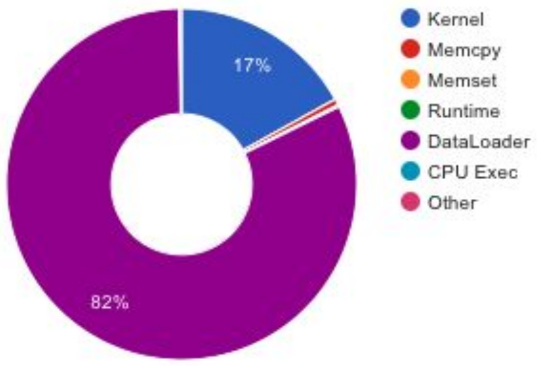
GPU Summary

GPU 0:

Name	Tesla T4
Memory	14.62 GB
Compute Capability	7.5
GPU Utilization	16.96 %
Est. SM Efficiency	16.91 %
Est. Achieved Occupancy	68.75 %
Kernel Time using Tensor Cores	0.0 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	1,763,649,145	100
Kernel	299,168,905	16.96
Memcpy	10,521,722	0.6
Memset	39,459	0
Runtime	3,034,169	0.17
DataLoader	1,446,068,956	81.99
CPU Exec	1,570,076	0.09
Other	3,245,858	0.18



GPU Utilization Improvement

Training with Alluxio-FUSE

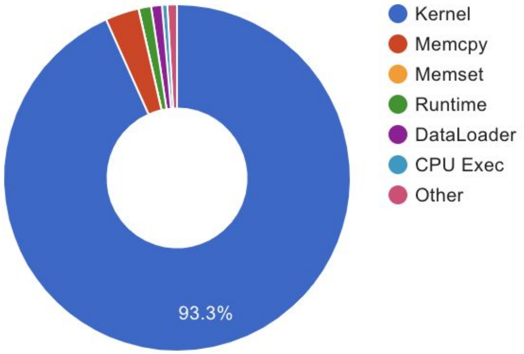
- Reduced DataLoader Rate from **82% to 1% (82X)**
- Increase GPU Utilization Rate from **17% to 93% (5X)**

GPU Summary ⓘ

GPU 0:	
Name	**Tesla T4**
Memory	**14.62 GB**
Compute Capability	**7.5**
GPU Utilization	**93.29 %**
Est. SM Efficiency	**92.98 %**
Est. Achieved Occupancy	**68.03 %**
Kernel Time using Tensor Cores	**0.0 %**

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	334,274,946	100
Kernel	311,847,023	93.29
Memcpy	10,500,126	3.14
Memset	43,946	0.01
Runtime	3,899,241	1.17
DataLoader	3,343,301	1
CPU Exec	1,648,391	0.49
Other	2,992,918	0.9



Data Access Management for Ray

Ray is Designed for Distributed Training

- Ray uses a distributed scheduler to dispatch training jobs to available workers (CPUs/GPUs)
- Enables seamless horizontal scaling of training jobs across multiple nodes
- Provides streaming data abstraction for ML training for parallel and distributed preprocessing.

Alluxio's Position In the Ray Ecosystem

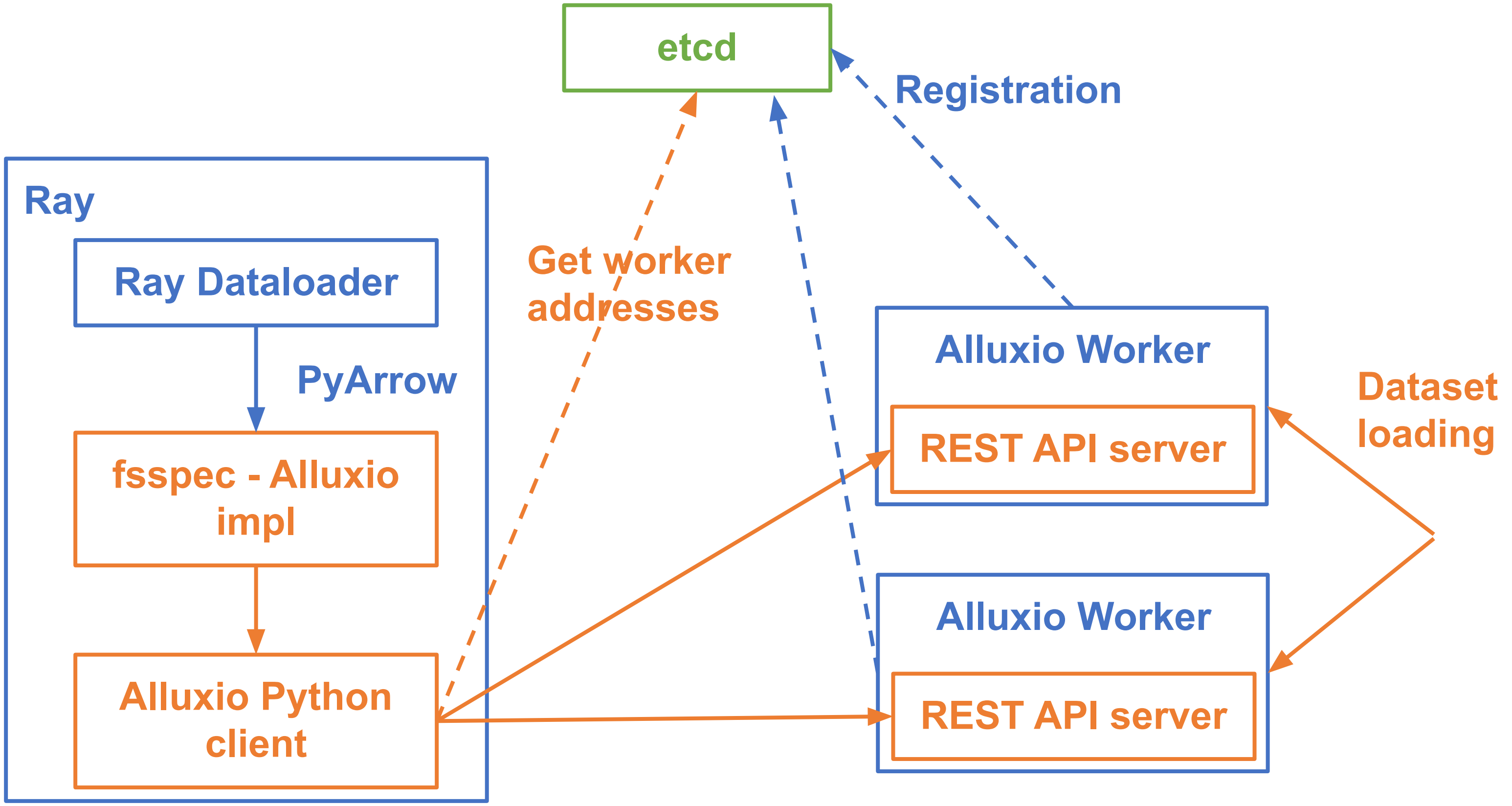
RAY **Unified Compute - ML pipeline orchestration**

PyTorch TensorFlow **ML Framework - Model training/inference**

ALLUXIO **Alluxio - High performance data access layer**

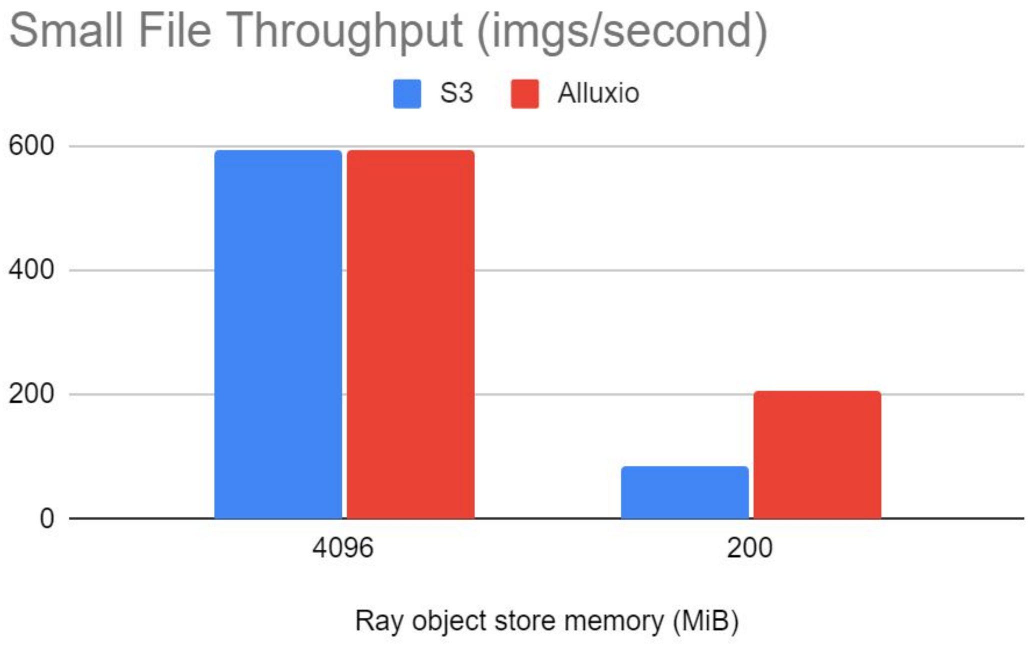
amazon S3 Azure **Storage - Data storage**

Alluxio - Ray Integration



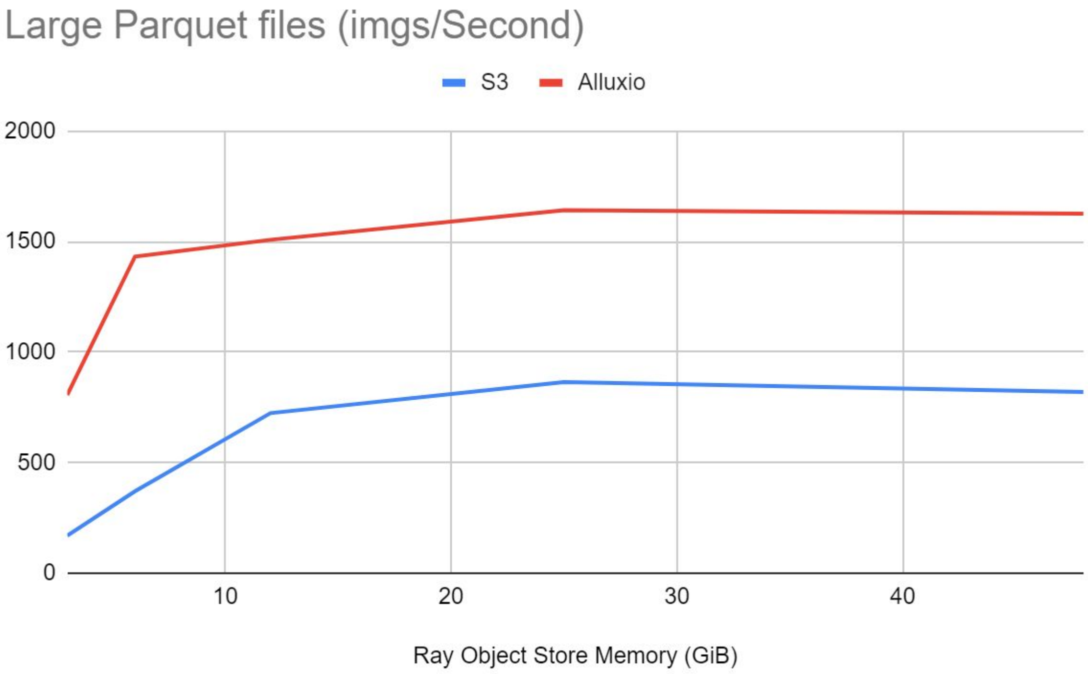
ALLUXIO

Alluxio+Ray Benchmark – Small Files



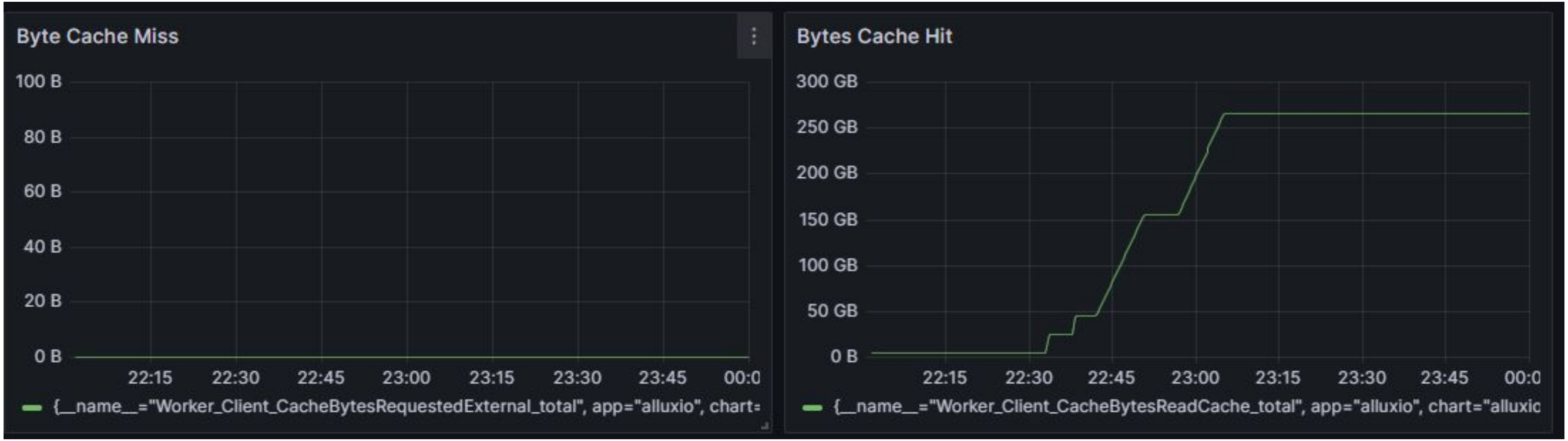
- Dataset
  - 130GB imagenet dataset
- Process Settings
  - 4 train workers
  - 9 process reading
- Active Object Store Memory
  - 400-500 MiB

Alluxio+Ray Benchmark – Large Parquet files

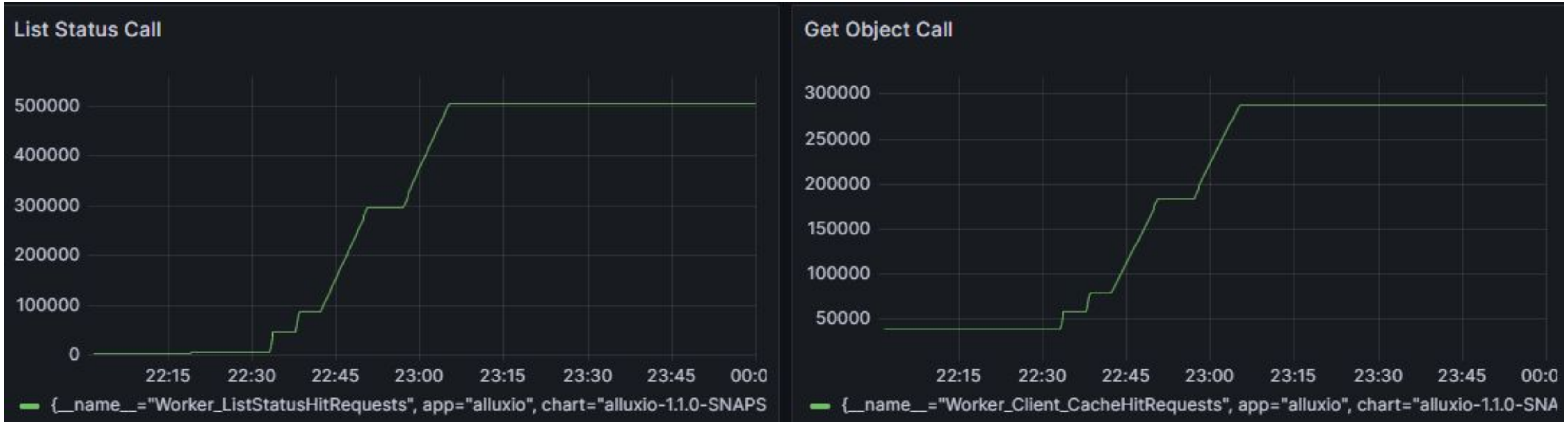


- Dataset
  - 200MiB files, adds up to 60GiB
- Process Settings
  - 28 train workers
  - 28 process reading
- Active Object Store Memory
  - 20-30 GiB

Cost Saving – Egress/Data Transfer Fees



Cost Saving – API Calls/S3 Operations (List, Get)



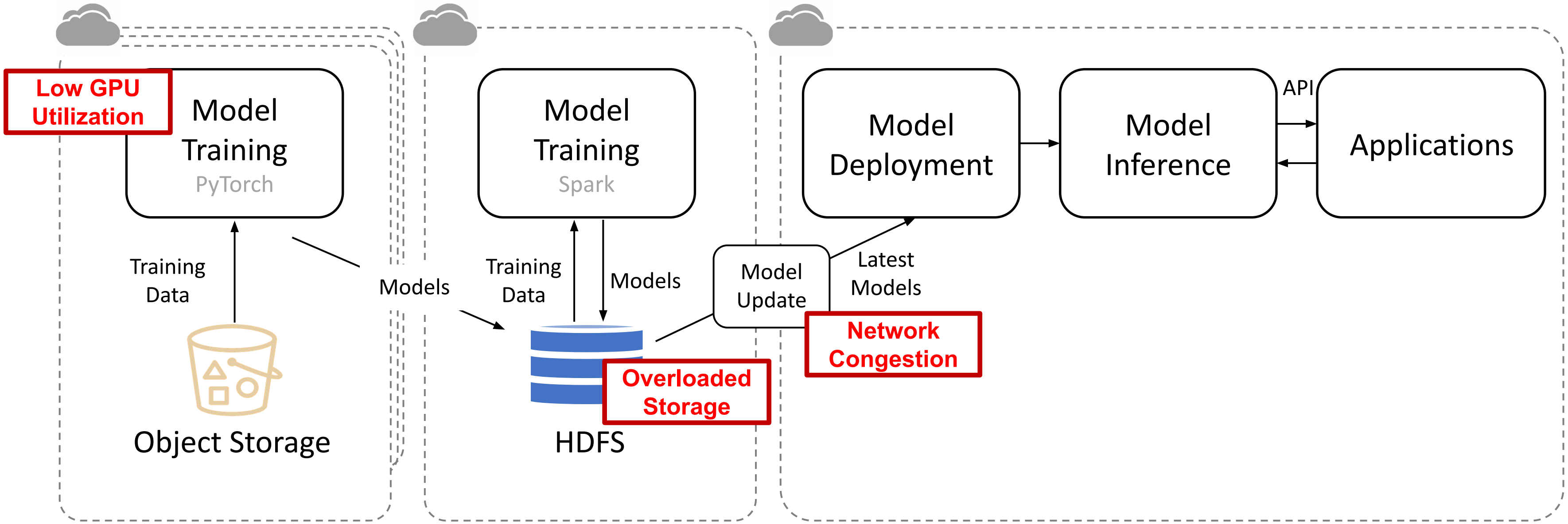
List/Get API calls only access Alluxio

ALLUXIO

Use Cases

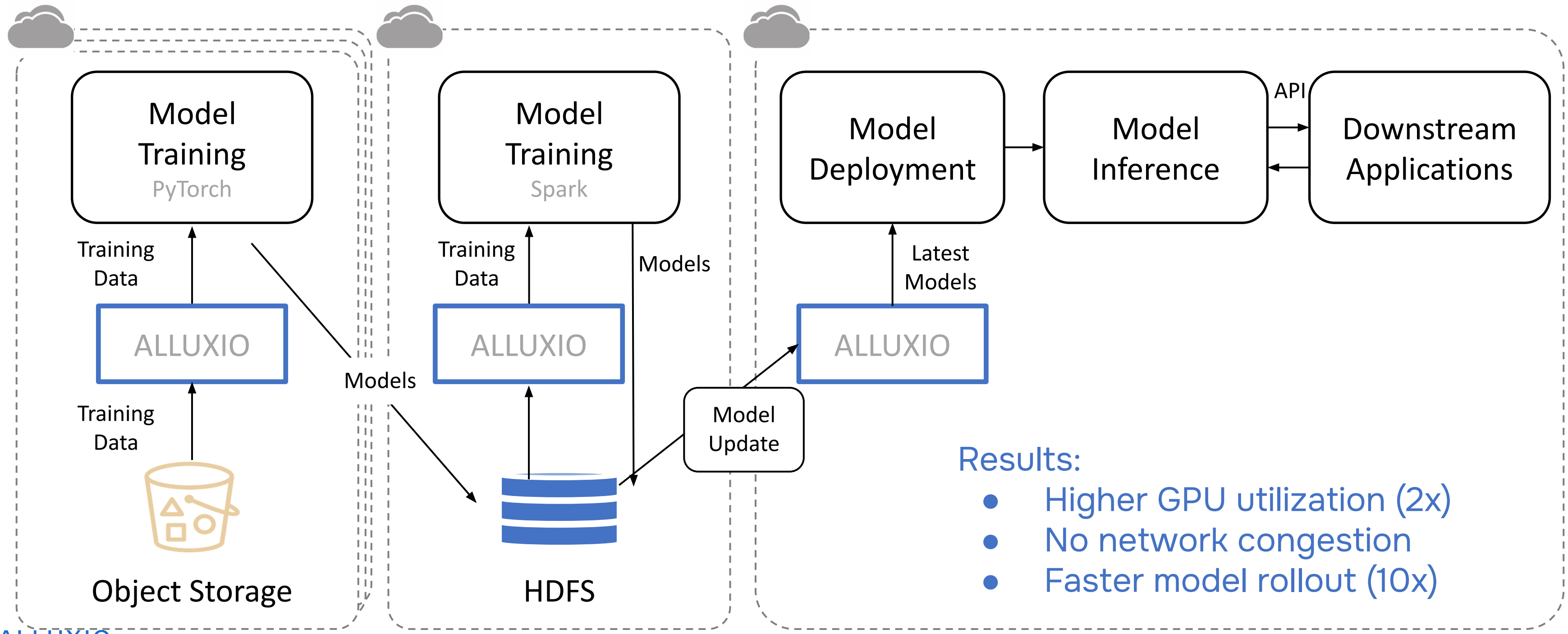
Data Caching Across ML Pipelines

Without cache



Data Caching Across ML Pipelines

With cache



Results:

- Higher GPU utilization (2x)
- No network congestion
- Faster model rollout (10x)

THANKS

Any Questions?

Scan the QR code for a Linktree including great learning resources, exciting meetups & a community of data & AI infra experts!