ALLUXIO

The Journey Along the Way to Data-Locality on Cloud for ML/AI

Lu Qiu, Machine Learning Engineer - lu@alluxio.com
Shawn Sun, Software Engineer - shawn.sun@alluxio.com

Lu Qiu

Machine Learning Engineer
Alluxio PMC
lu@alluxio.com

Shawn Sun

Software Engineer
shawn.sun@alluxio.com

Agenda

1. Benefits of data locality
2. Existing solutions
3. A new design
4. Implementation
5. Production Use Cases
6. Alluxio & Ray Integration

Benefits - Why Data Locality?

1. Performance Gain
   - Faster access to your data compared to remote storage
   - Less time spent on data-intensive applications
     - ML & AI
2. Cost Saving
   - Fewer API calls to cloud storage (Data & Metadata)
   - Higher utilization of GPU → Less GPU time

Existing Solutions

Existing Solutions

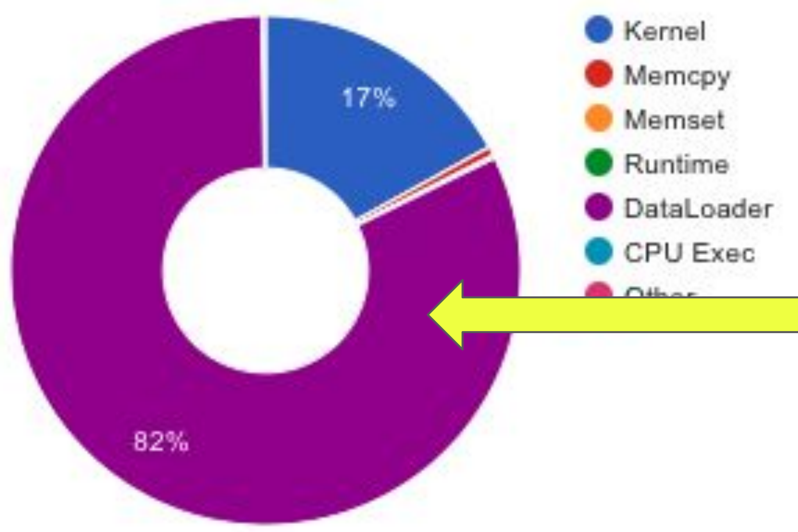
1. Read data directly from remote storage on the fly
2. Copy data from remote to local before training
3. Local cache layer for data reuse
4. Distributed cache system

Always Read From Remote - No Locality

- Easy to set up
- Every epoch needs to re-read all the data from remote.
  - Multiple epochs are almost always needed for better accuracy
  - Reading from remote 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
    - All benefits of data locality
- Management is hard
    - Must manually delete training data after use
- Cache space is limited
    - Dataset is huge - limited benefits

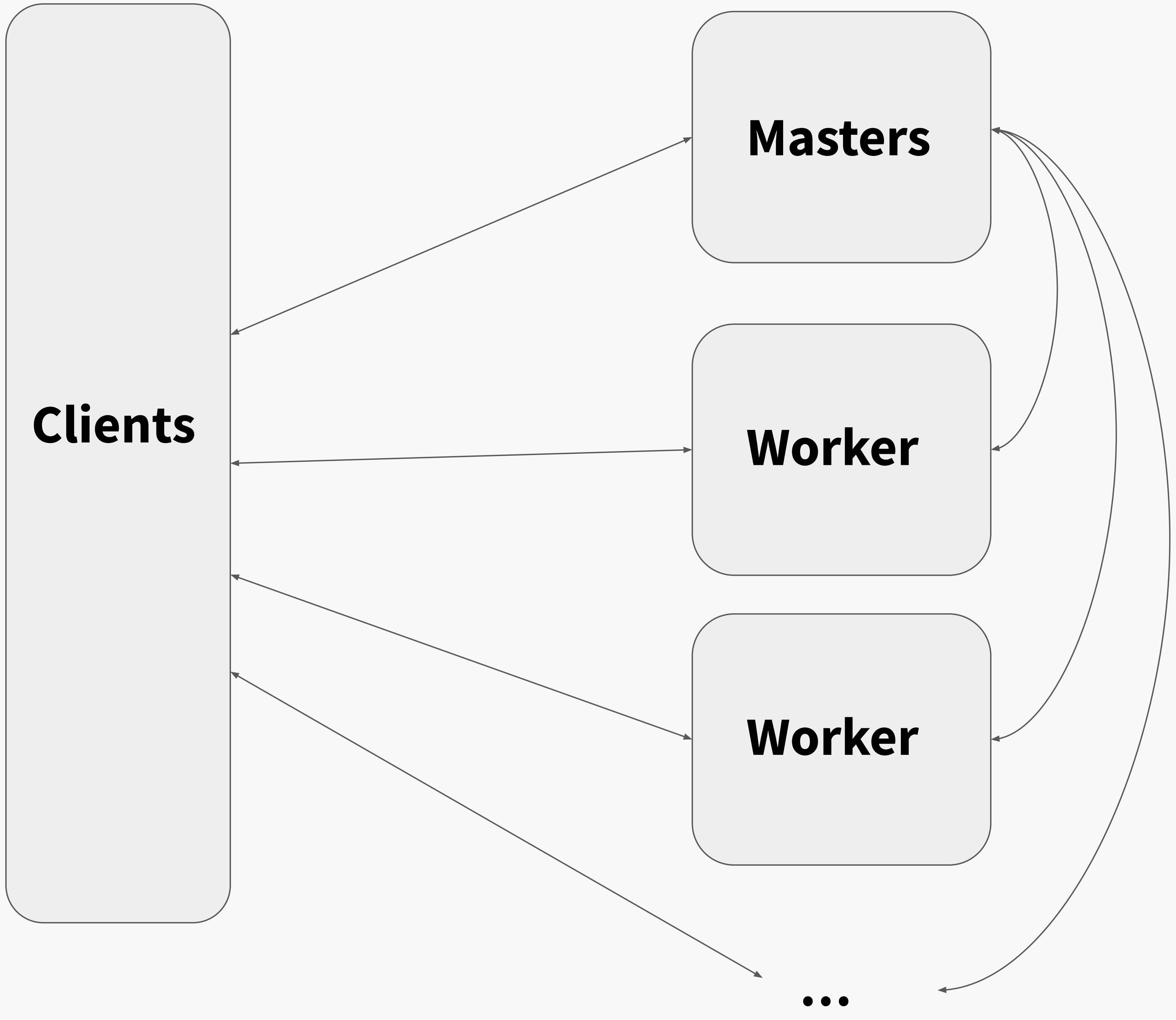
Local Cache Layer for Data Reuse

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

- Reused data is local
- 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



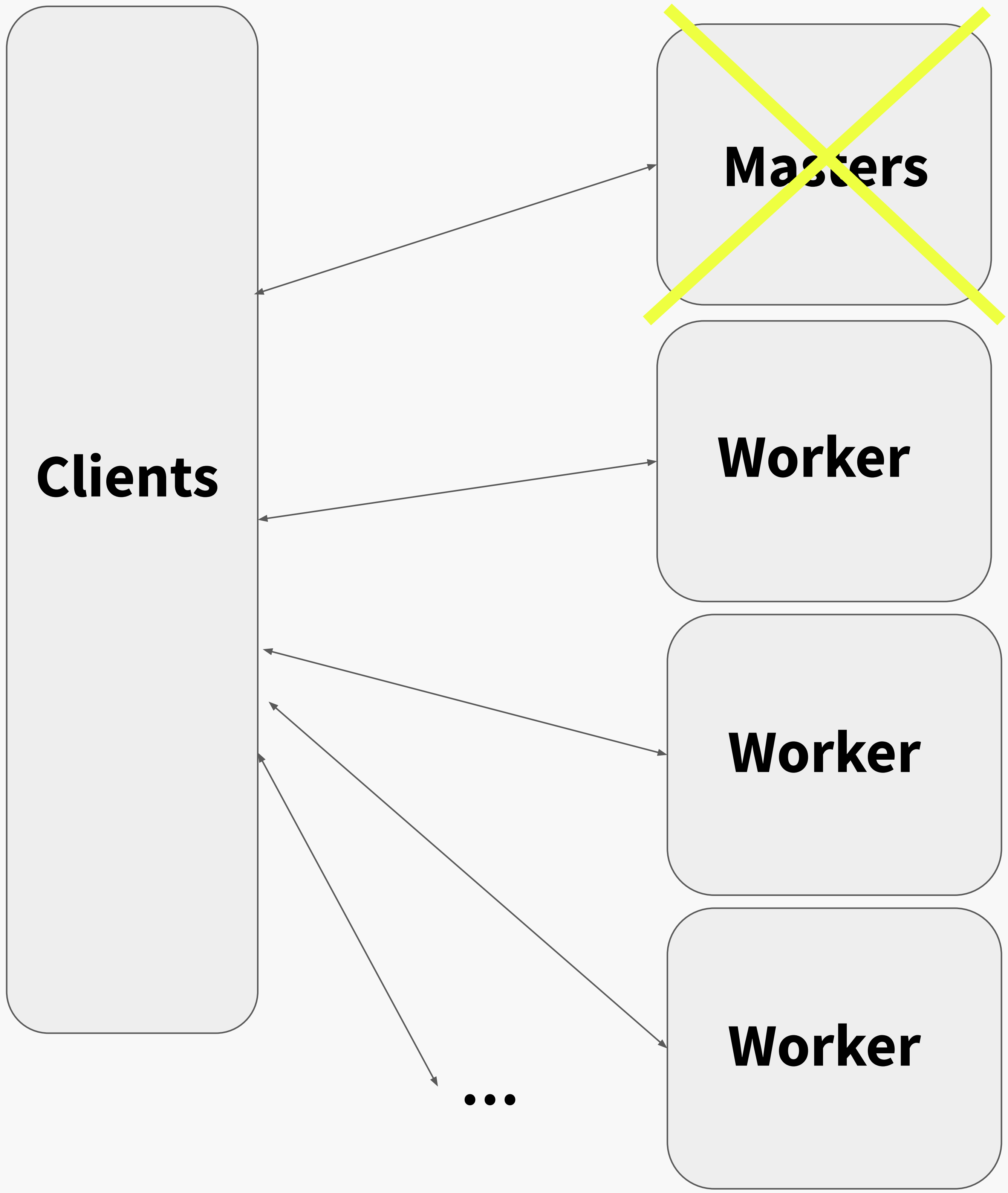
- Can store much more cached data compared to local cache.
- Data management functionalities.
- Masters are “single” point of failure.
- The huge number of files makes masters the bottleneck of the overall performance.

Challenges

1. Local storage space is limited
    - Amount of data is growing fast
2. Reliability
    - Availability is the key for every service
3. Scalability
    - Number of files for training is huge - in order of billions
4. Data Management
    - Manual work is unfavorable

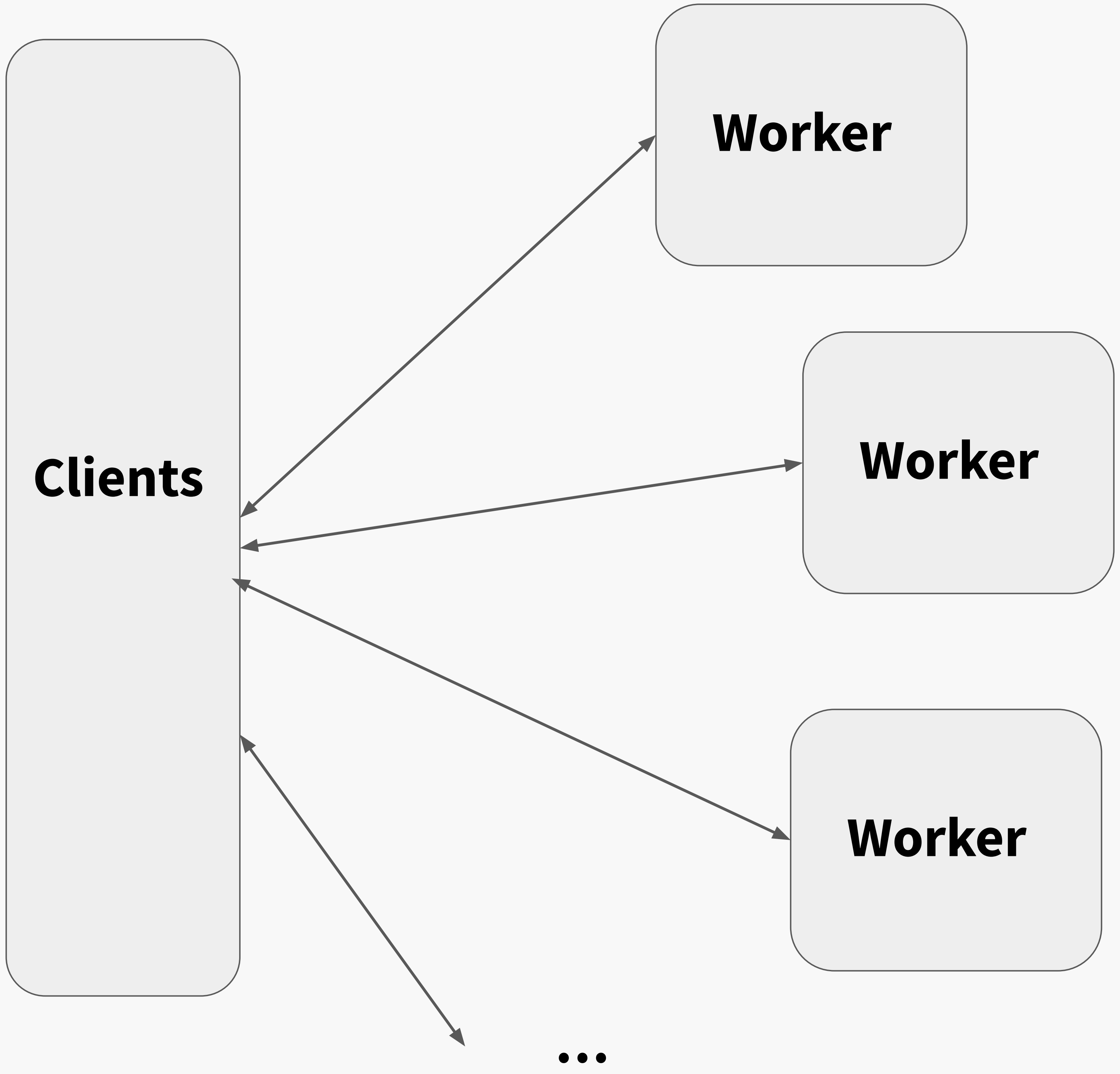
A New Design

Consistent Hashing for caching



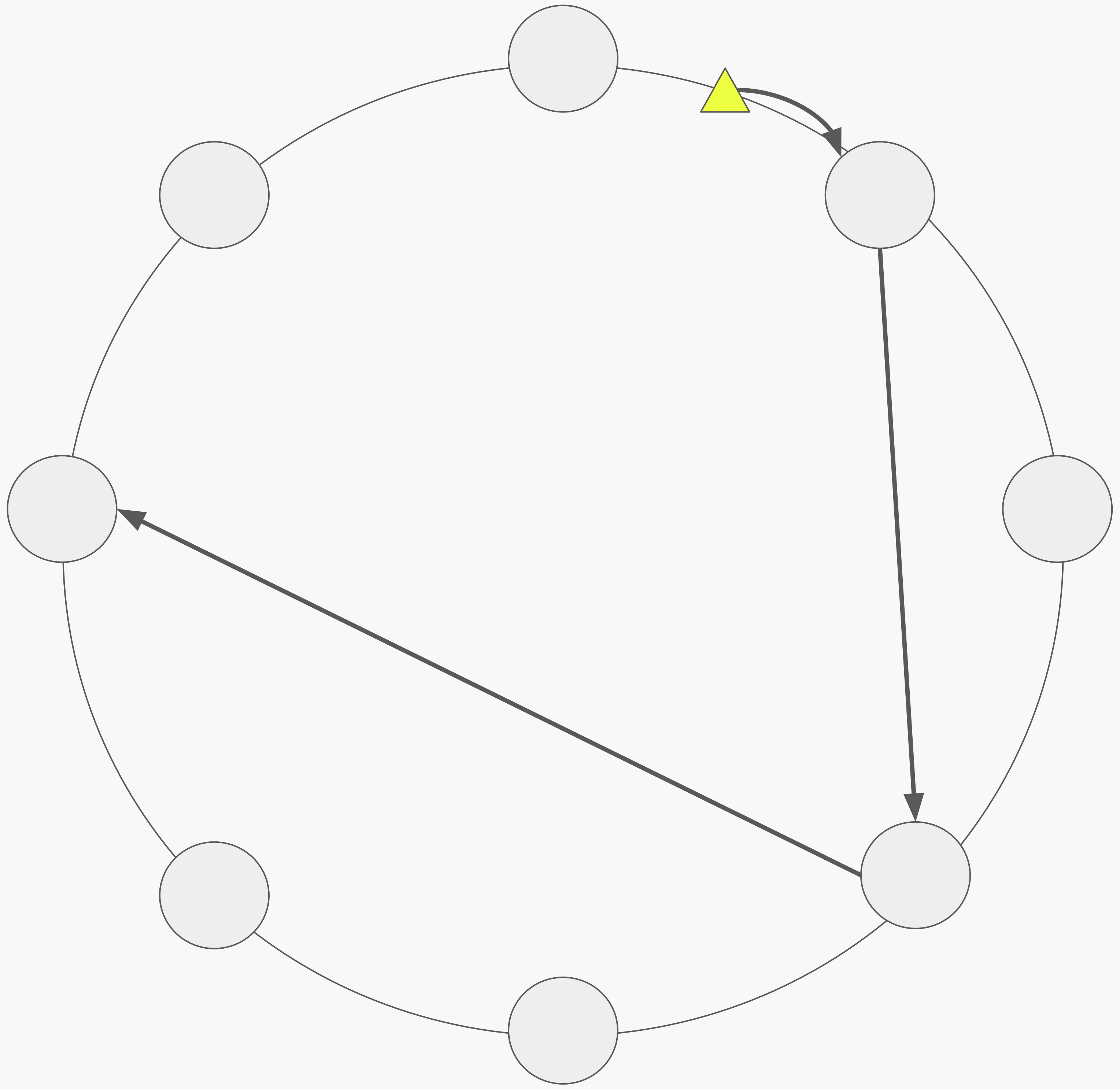
- Use **consistent hashing** to cache both data and metadata on workers.
- Worker nodes have plenty space for cache.
- No more single point of failure.
- No more performance bottleneck on masters.
- Data management system.

New Challenges



- Potential load imbalance.
- Some worker can be very busy, while others being idle.
- Hurting overall performance.
- May lead to outages.

Soft Affinity Caching Solution



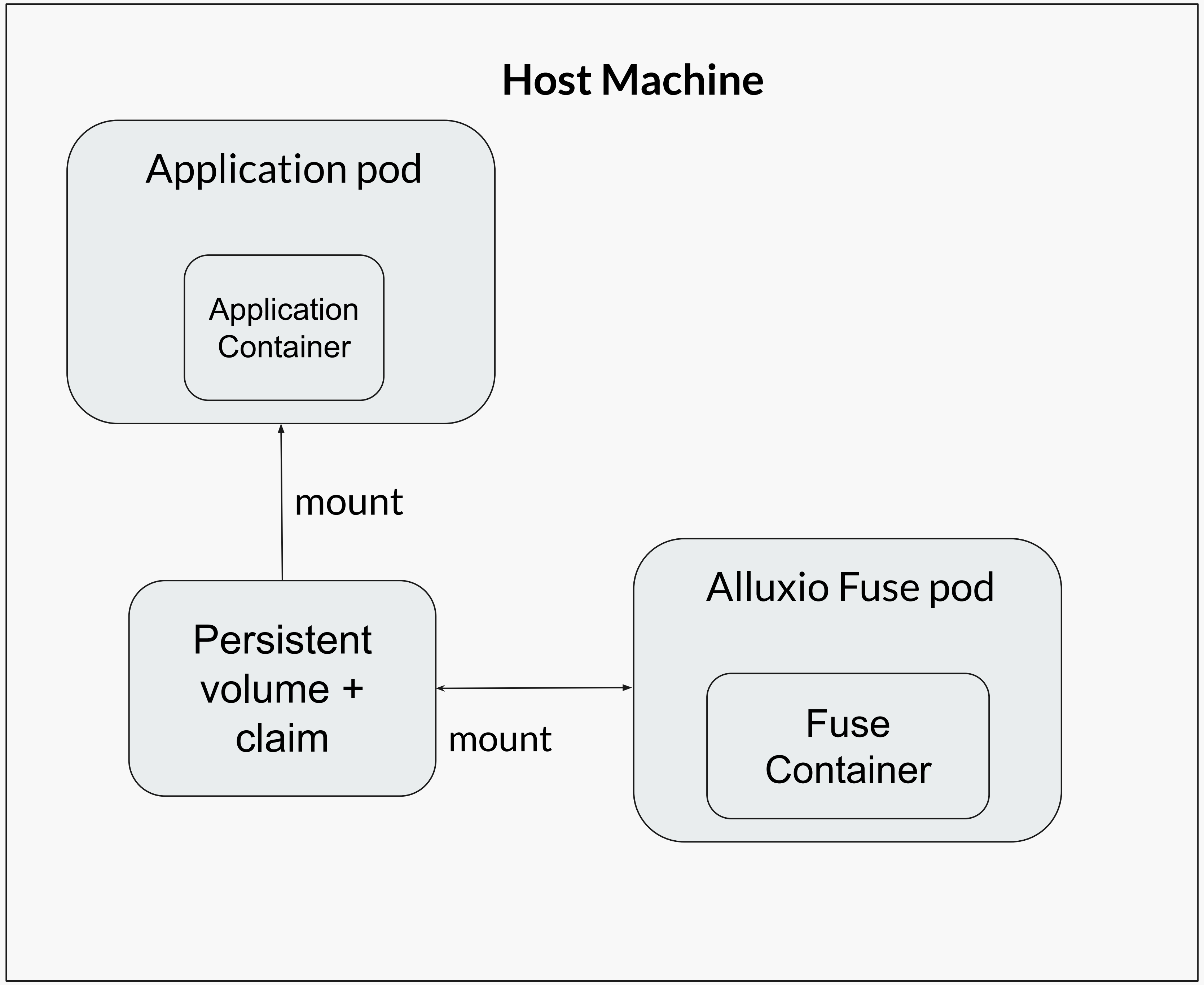
Implementation

Alluxio 3xx

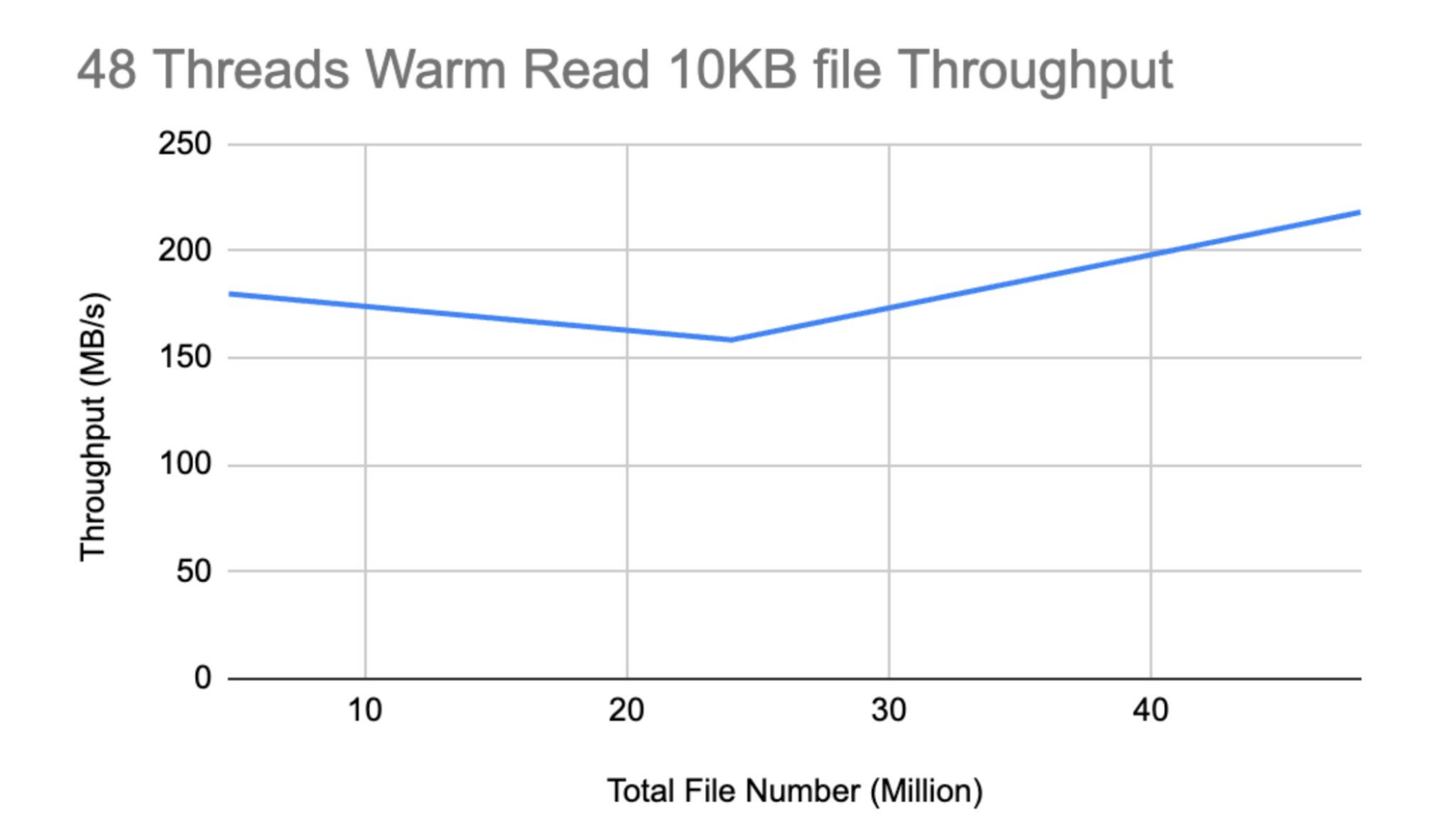
- Implements the soft-affinity data cache scheduling algorithm for caching data
- High scalability
  - One worker supports 30 - 50 million files
- High availability
  - 99.99% uptime
  - No single point of failure
- Cloud-native K8s Operator for deployment and management

CSI x FUSE for Training

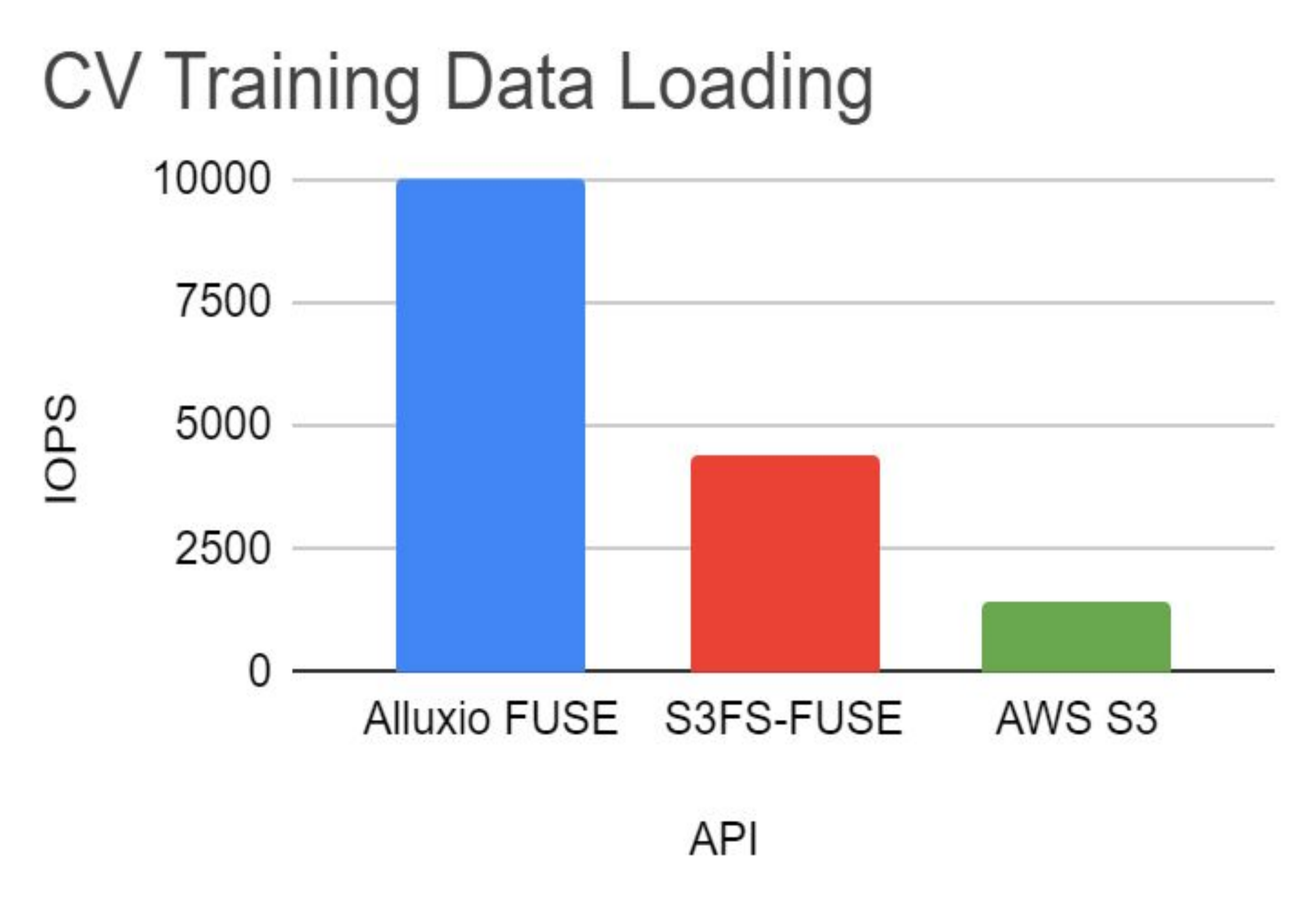
- FUSE: Turn remote dataset into local folder for training
- CSI: Launch FUSE pod only when dataset is needed!
- Three layers of caching
  - kernel cache
  - local disk cache
  - distributed cache



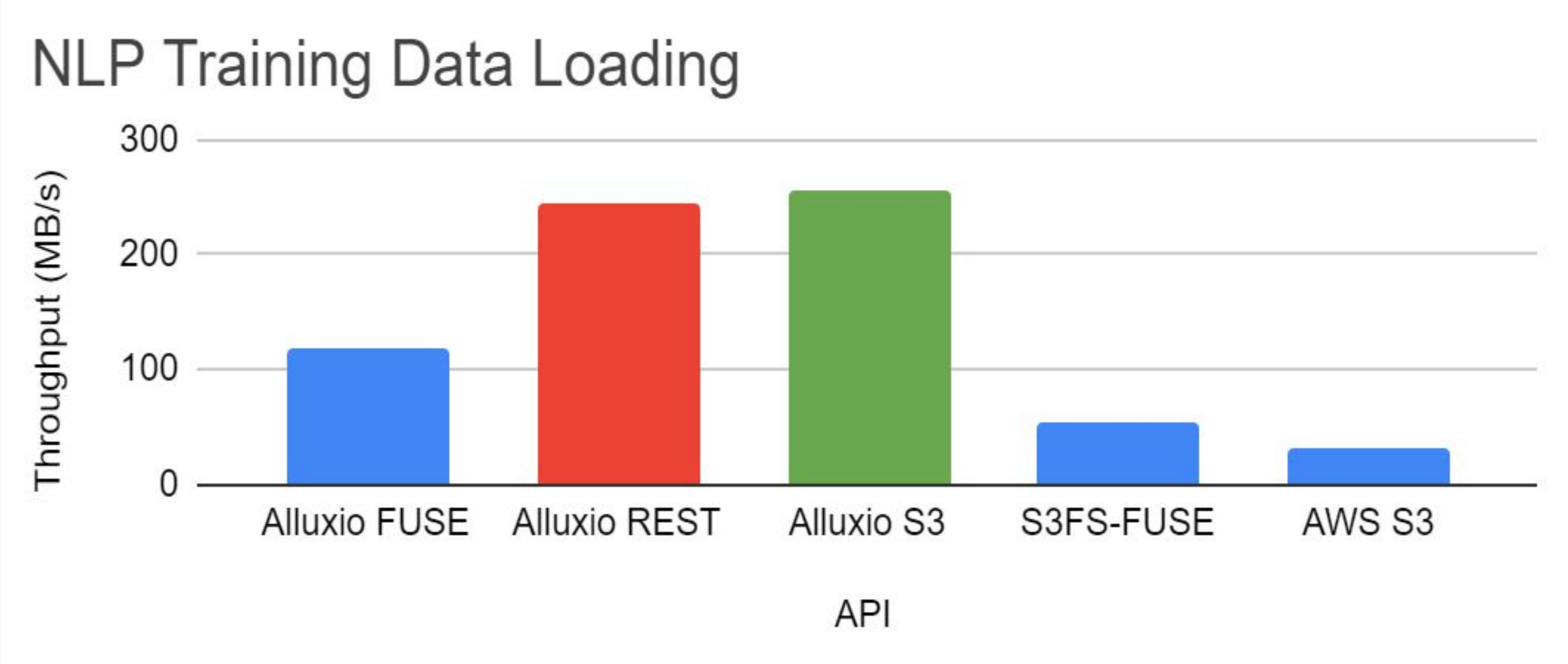
Single Worker Performance Benchmark



Data Loading Performance



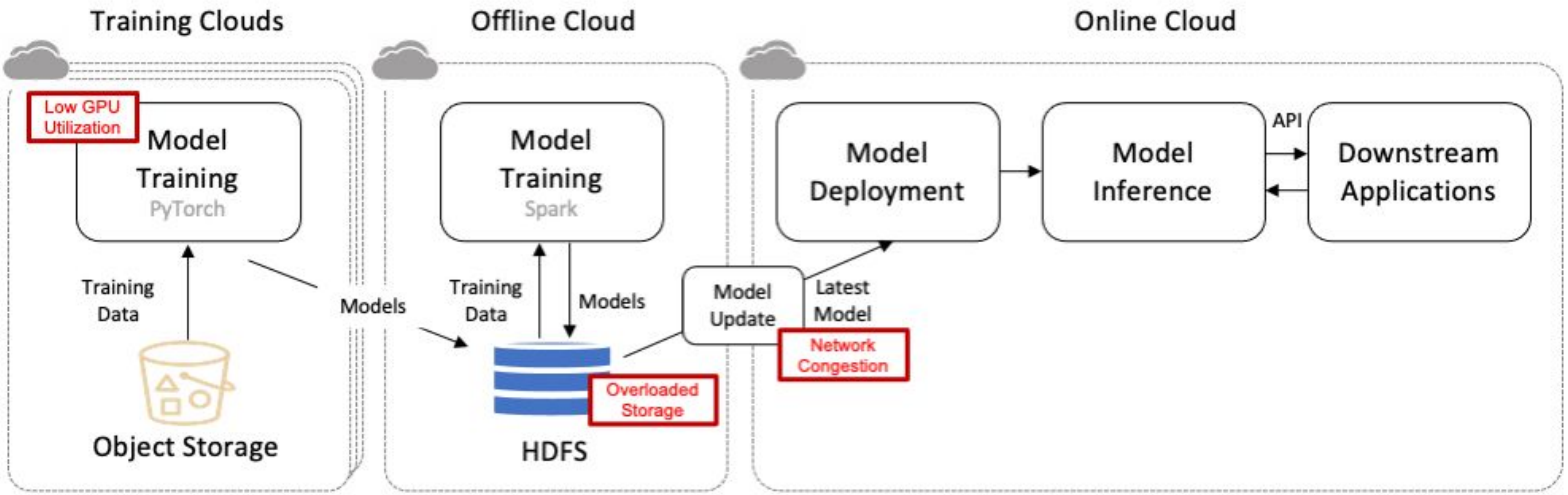
Subset of imagenet



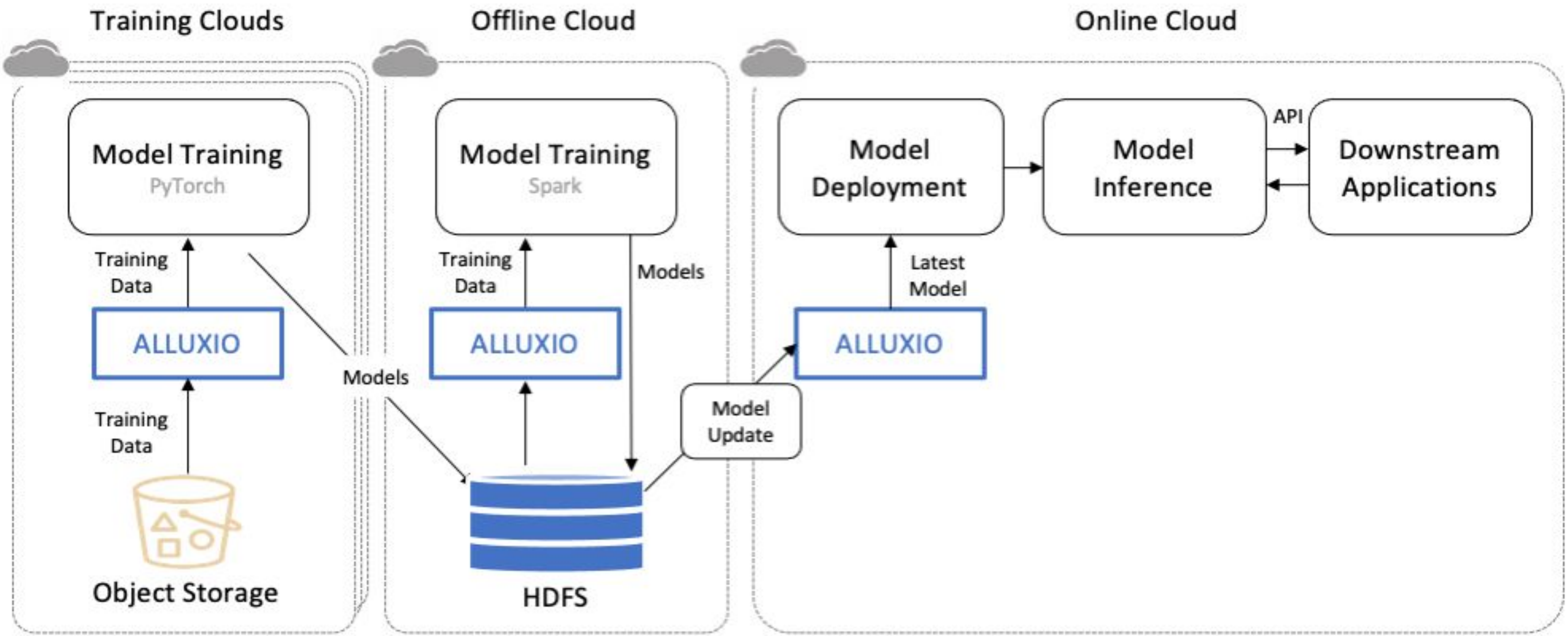
yelp academic dataset

Solve Data/Model Locality Issue @ Microsoft, Shopee, Zhihu & Others

Challenges of LLM Pipeline



LLM Cache Strategy (Alluxio)



Increased GPU Utilization Rate in Model Training

Training Directly from Storage

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

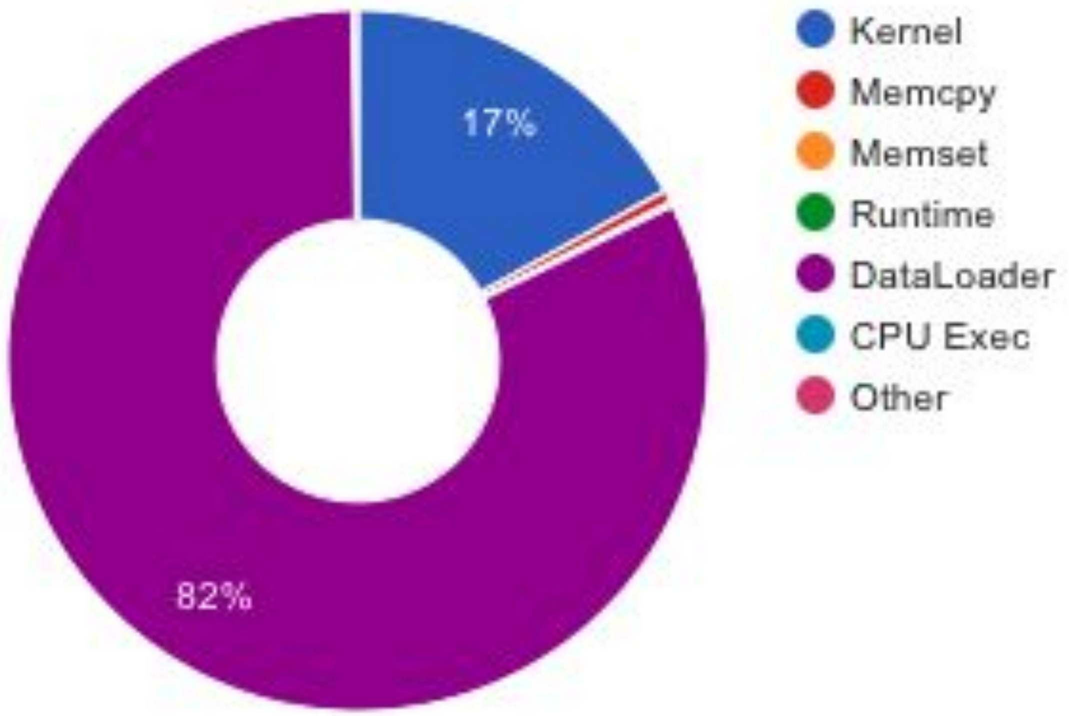
GPU Summary ⓘ

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



Increased GPU Utilization Rate in Model Training

Training with Alluxio

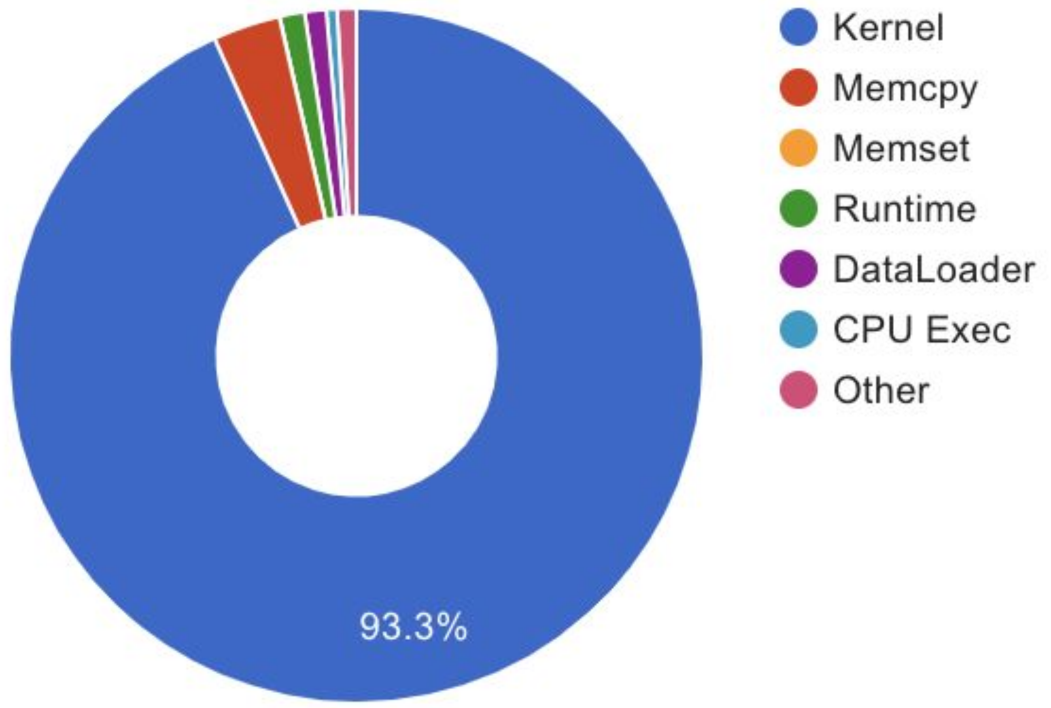
- Reduced DataLoader Rate from 82% to 1% (82X)
- Increase GPU Utilization Rate from 17% to 93% (5X)

GPU Summary

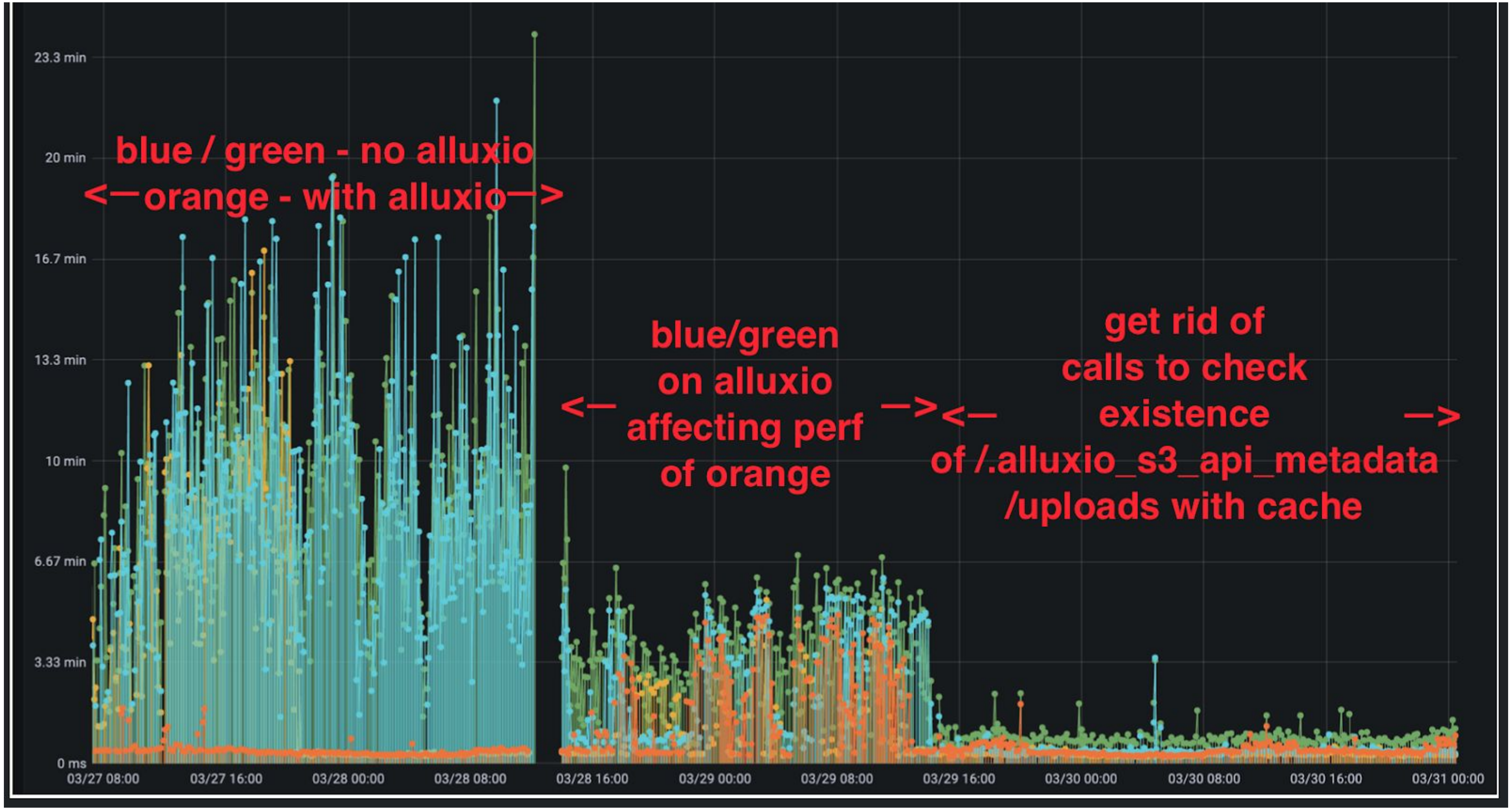
GPU 0:	
Name	**Tesla T4**
Memory	**14.62 GB**
Compute Capability	**7.5**
GPU Utilization	**93.29 %**
Est. SM Efficiency	**92.98 %**
Est. Achieved Occupancy	**68.03 %**
Kernel Time using Tensor Cores	**0.0 %**

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



10X Acceleration in Model Deployment



Alluxio & Ray

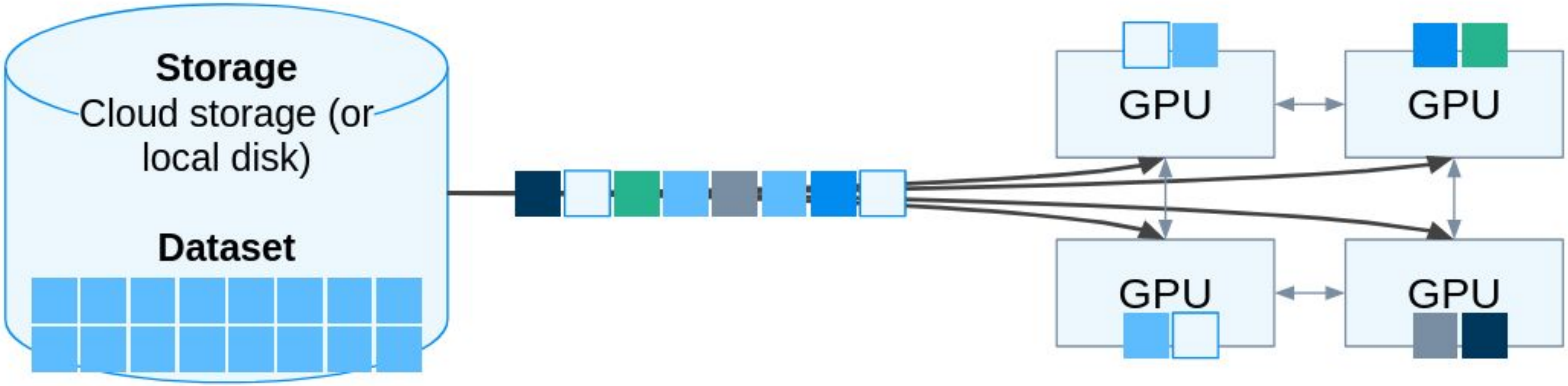
Ray is Designed for Distributed Training

- 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.

Why Streaming

*“Downloading the entire training dataset to local disk may not make sense

- If you don’t have enough disk space on each node to store the entire dataset
- If you want to overlap downloading the data with data preprocessing and training
- If you want each worker node to read a different and random subset of the data on each epoch”



*Source: https://www.anyscale.com/blog/fast-flexible-scalable-data-loading-for-ml-training-with-ray-data

Performance & Cost Implication of Ray

- You might load the entire dataset again and again for each epoch
- You cannot cache the hottest data among multiple training jobs automatically
- You might be suffering from a cold start every time.

5 AM

Hi, we are working with converting our training framework to Ray. Doing the initial conversion with went rather well and are up and running with ray-train. But now we want to speed it up and our biggest bottleneck is our data handling.

We are working with big datasets (order of magnitude Terra-byte) and do different augmentations to the training samples every time they are loaded, so we cannot pre-process the dataset before training but do it when loading each batch. As the code is written for local training, parts of the data is loaded to each worker, which put constraints on the number of workers we need for bigger datasets.

We are now starting to work with re-writing the training code to focus on being focused towards distributed work loads and are exploring doing the pre-processing on separate machines to ge rid of these constraints. I therefore have some questions:

- Is this something ray-data could be used for, or is that more adapted for datasets that can be loaded before training start?
- Or is this a more specific use case and more probable that we would need to write our own data-handling with ray-core?

16 replies Last reply 28 days ago

I would like to pull my models from s3 and mount them to a volume which can then be accessible by any worker / node in my Ray Cluster. Is there a recommended way to do this? Or any guides to help? (edited)

11 replies

The idea is to avoid long download times in the `init` methods, especially when autoscaling is enabled (this would be duplicated work for no reason)...

Alluxio's Position In the Ray Ecosystem

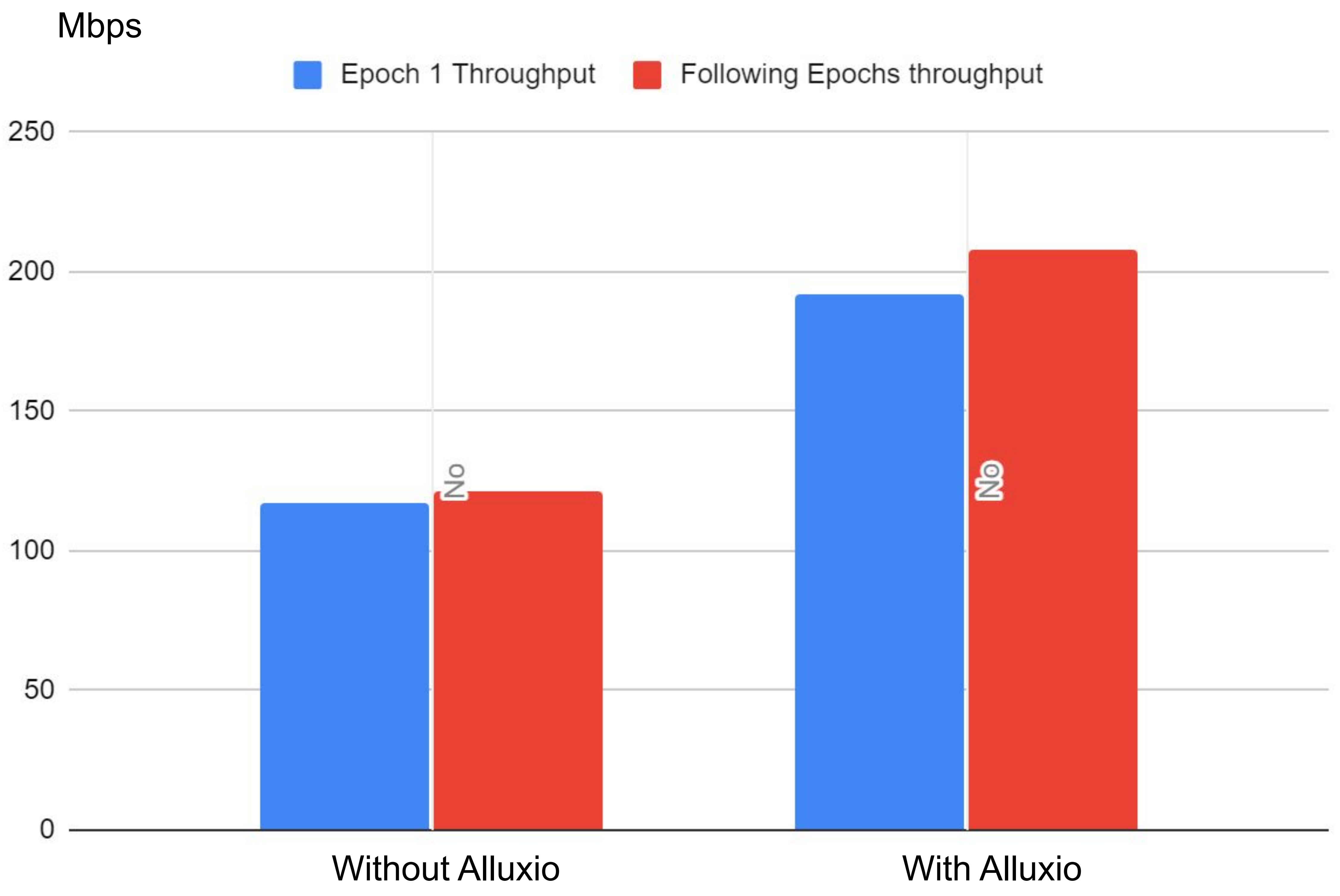
RAY — Unified Compute - ML pipeline orchestration

PyTorch TensorFlow — ML Framework - Model training/inference

ALLUXIO — Alluxio - High performance data access layer

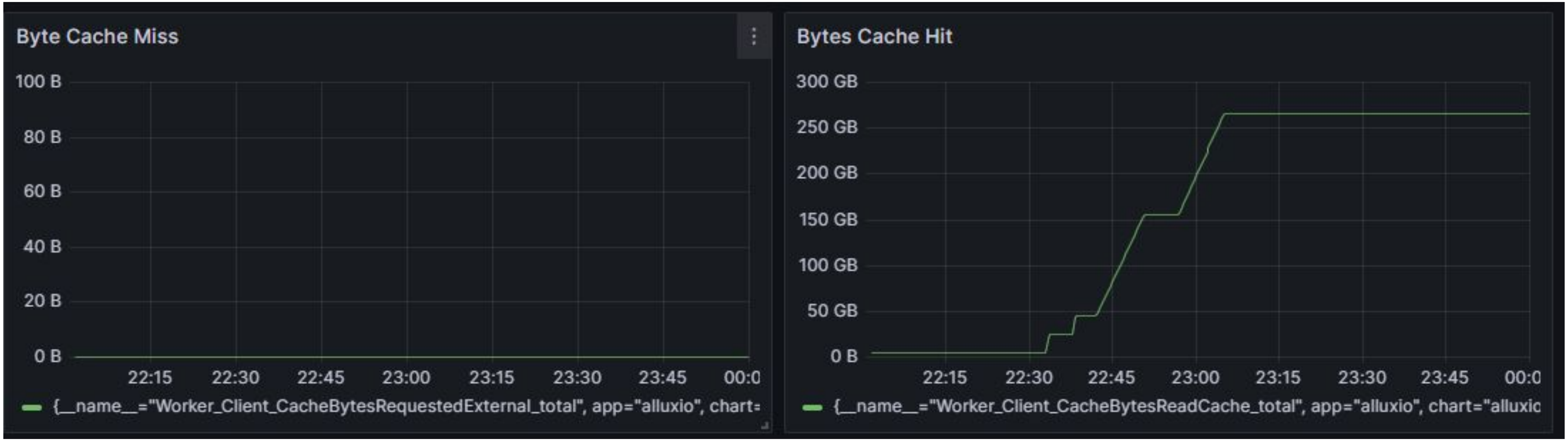
amazon S3 Azure — Storage - Data storage

Alluxio+Ray Benchmark – I/O Throughput

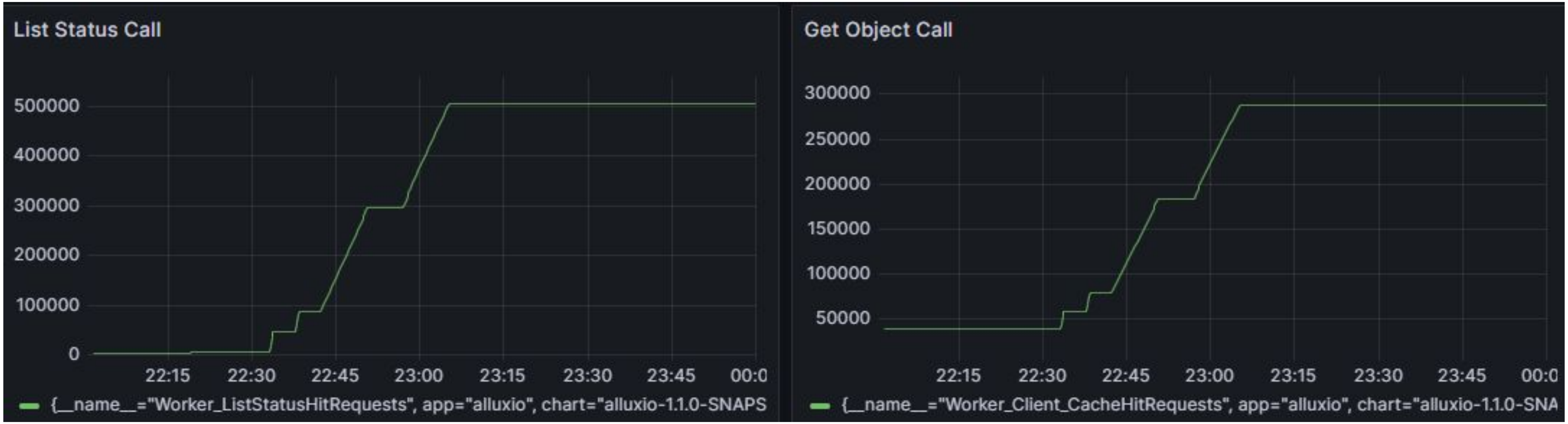


- Instance Type
  - m5.4xlarge 16vCPU 64GB memory
- Ray head resources
  - nohup ray start --head --memory=$((16 * 2**30)) --object-store-memory=$((4 * 2**30)) --dashboard-host=0.0.0.0 --metrics-export-port=8080 --block --num-cpus=14 --system-config='{"automatic_object_spilling_enabled": false}' &
- Ray actual task resources
  - python release/nightly_tests/dataset/multi_node_train_benchmark.py --num-workers 12 --file-type image --data-root s3://ai-ref-arch/imagenet-mini/train --object-store-memory $((4 * 2**30))

Cost Saving – Egress/Data Transfer Fees



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



Lu Qiu
www.linkedin.com/in/luqiuai

Shawn Sun
www.linkedin.com/in/-shawn-sun-/

Website
www.alluxio.io

Slack
https://alluxio.io/slack

Social Media
Twitter.com/alluxio
Linkedin.com/alluxio

Github
https://github.com/Alluxio