ALLUXIO

Data Caching Strategies for Data Analytics and AI

Beinan Wang, Chunxu Tang @ Alluxio & PrestoDB

Dr. Beinan Wang

Senior Staff Engineer @ Alluxio
PrestoDB Committer

Dr. Chunxu Tang

Research Scientist @ Alluxio
PrestoDB Committer

ALLUXIO

Image source: Unsplash

Data Analytics

Artificial Intelligence

Image source: Unsplash

Data cache (files/objects) plays a crucial role as an accelerator for analytics and AI computation, but …

Data analytics (SQL) and AI have different traffic patterns, leading to different performance-oriented cache strategies and recommendations.

Agenda

Caching Strategy Based on Traffic Patterns

1.1 Traffic Pattern for AI Training
1.2 Traffic Pattern for Analytical SQL

Advanced Caching Strategy

2.1 Cache Capacity Planning
2.2 Adaptive Caching Strategy

Takeaways

Traffic Pattern for AI Training

Traffic pattern for AI training and cache strategy recommendation

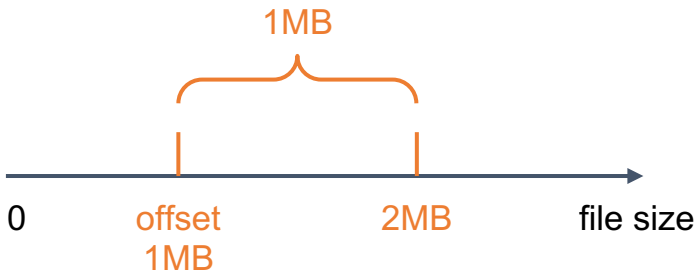
Pattern I: Data Access

Key findings based on our observations on multiple production datasets from tech companies:

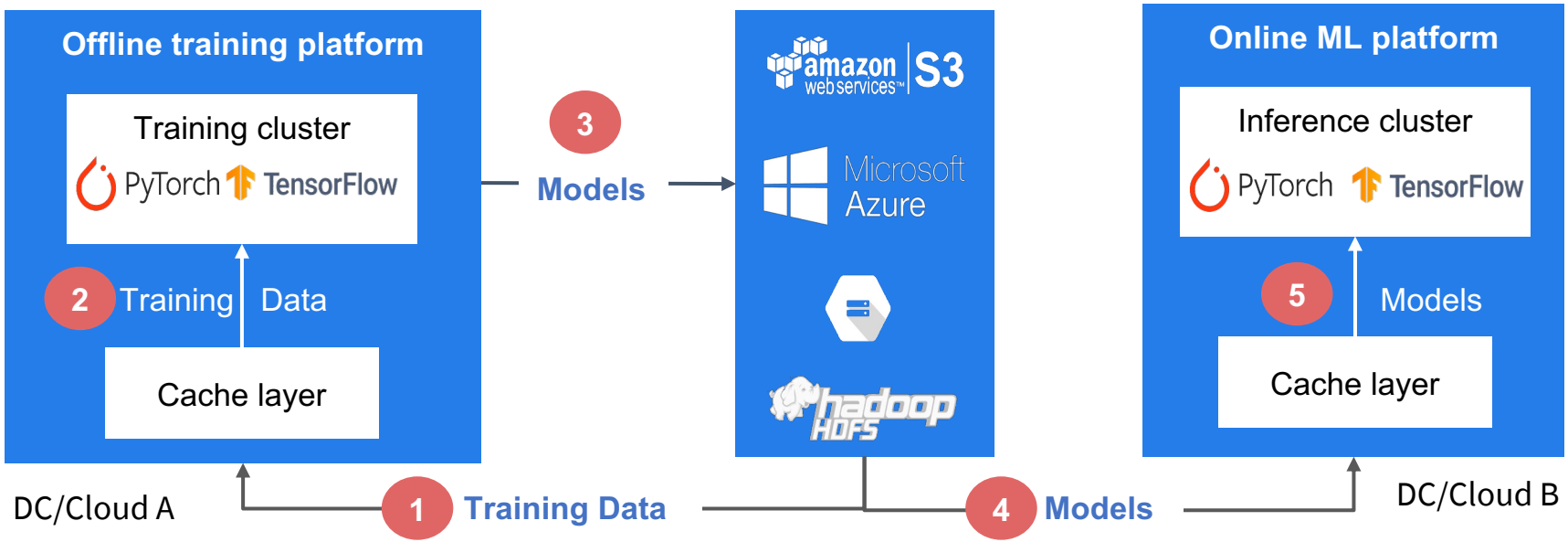
- **Scenario A: Data is stored in some large structured files**
  - Positioned/Random reads are more than sequential reads
  - Block access is (almost) evenly distributed
  - Each read is small (e.g. 4 kB read in Arrow)
- **Scenario B: Data is stored in many small semi-structured/unstructured files**
  - The file number can be > 10 billion
  - File access is (almost) evenly distributed
  - Reads a batch of files

Cache Strategy Recommendation I

- Performance
  - Hierarchical caching
  - **Optimization for positioned/random reads**
    - Read by chunks in a fixed size (e.g. 1 MB)
    - Read amplification vs. request number tradeoff
- Scalability/Elasticity
  - Able to grow/shrink capacities
  - Masterless vs. Master-Worker



Pattern II: Hybrid/Multi-Cloud for ML Infrastructure

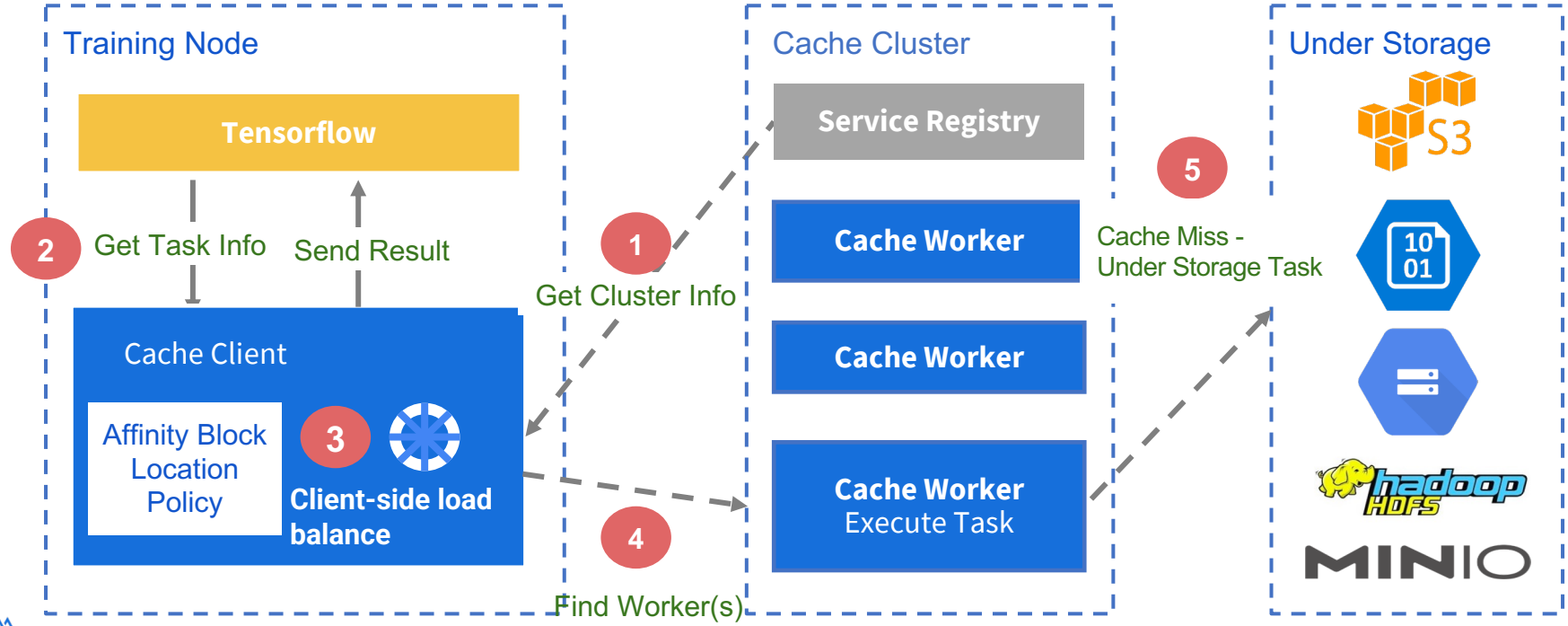


Cache Strategy Recommendation II

- Cloud-friendliness
  - Flexibility for hybrid/multi-cloud
  - Configurable cache admission/eviction policies

- Availability
  - No single point of failure
  - Support fallbacks to under storage

Integration with AI Training

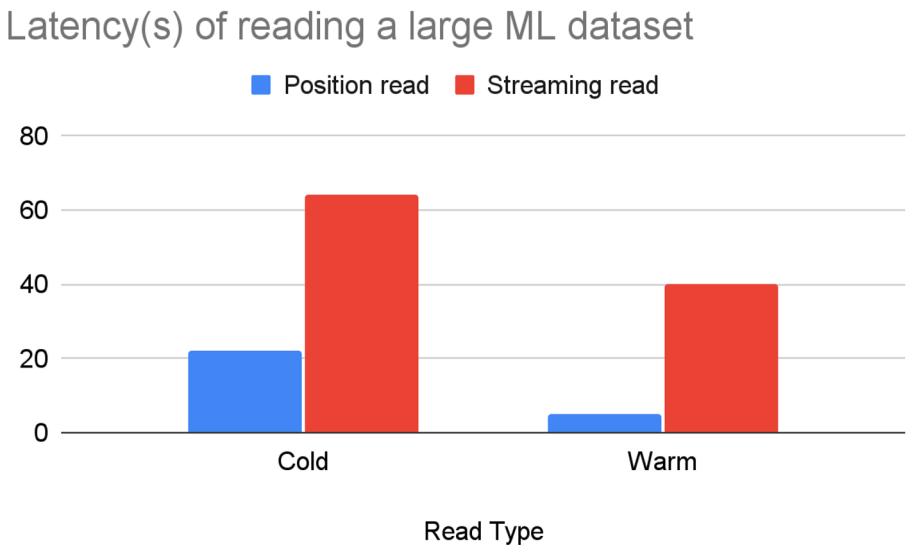


Evaluation - Scenario A (Large Structured Files)

Setup

- Evaluated on a production ML dataset
- Ran read operations with 4 threads
- Differentiated warm and cold reads
- Read from local NVMe storage

Positioned read outperforms streaming read when reading large ML datasets

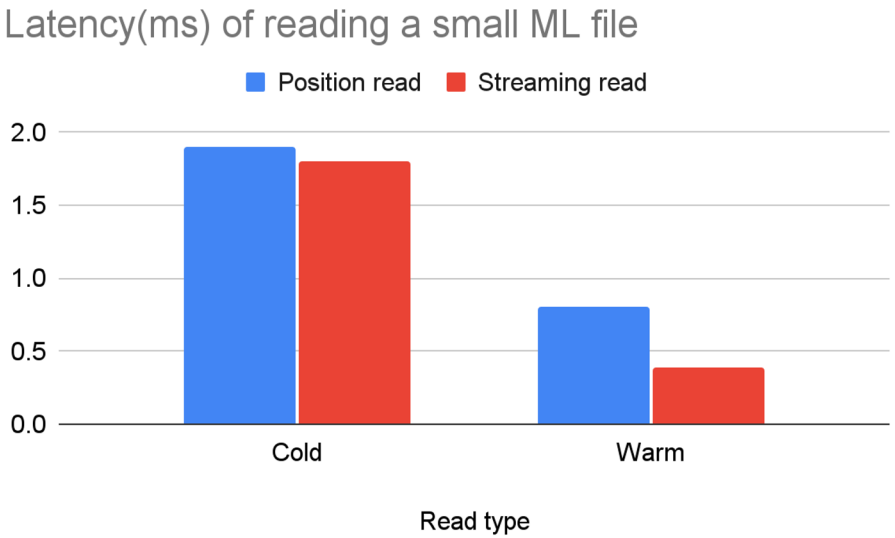


Evaluation - Scenario B (Small Unstructured Files)

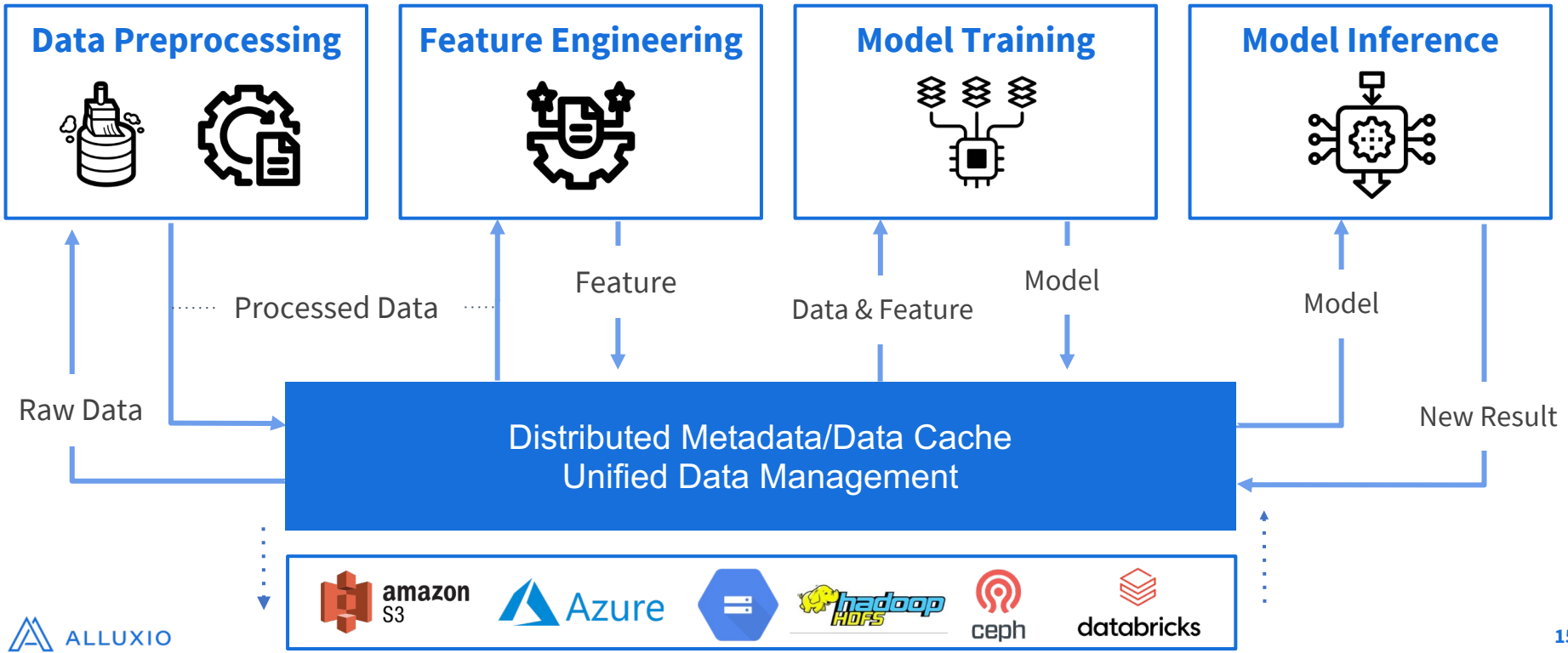
Setup

- Evaluated on a production ML dataset with >10k files
- Each file's size is ~100kB
- Read from local NVMe storage

Streaming read outperforms positioned read when reading small ML datasets



Future - Integration with ML Lifecycle

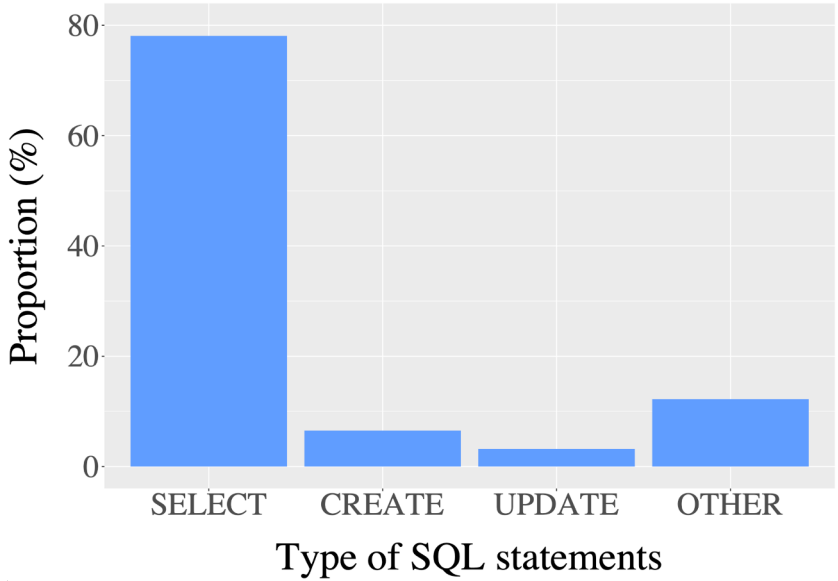


Traffic Pattern for Analytical SQL

Traffic pattern for analytical SQL and cache strategy recommendation

Analytical SQL

- Most workloads are read only
- The data accessed daily could be dozens of PB
- Data might be located in different DCs or cloud
- Using temporary files for uncommitted data



A typical Twitter's OLAP workload

Pattern I: Data Access

- Read distribution in compute engine (Presto)

	Sum	Count	Min	Max
Bytes read	347506	39	1687	16384
Read latency (Nano)	48,964,641,456	39	94343	11,434,222,029

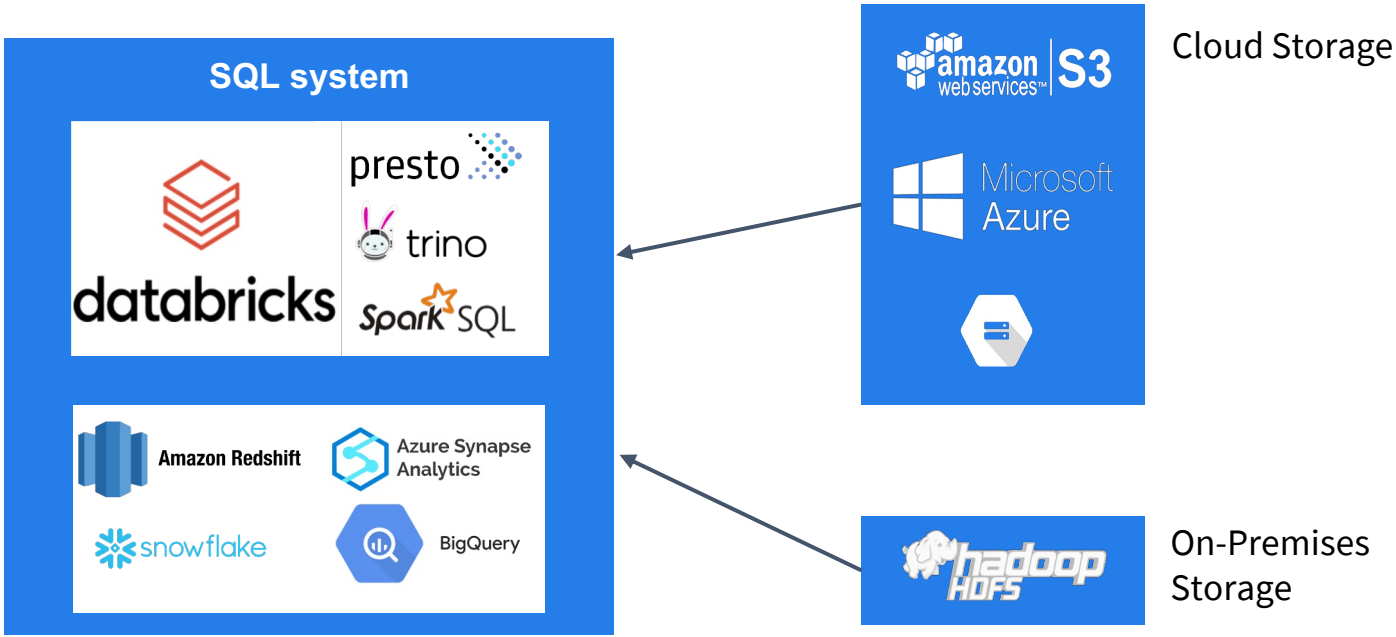
- Read distribution from under storage (HDFS)

<64K bytes	< 128K bytes	<512K bytes	< 1M bytes
50%	70%	80%	95%

Cache Strategy Recommendation I

- Performance
  - Hierarchical caching
    - File buffer
    - Page cache
    - Remote cache
  - Seekable streaming read
    - Preload
    - Eliminate read amplification when selectivity is high

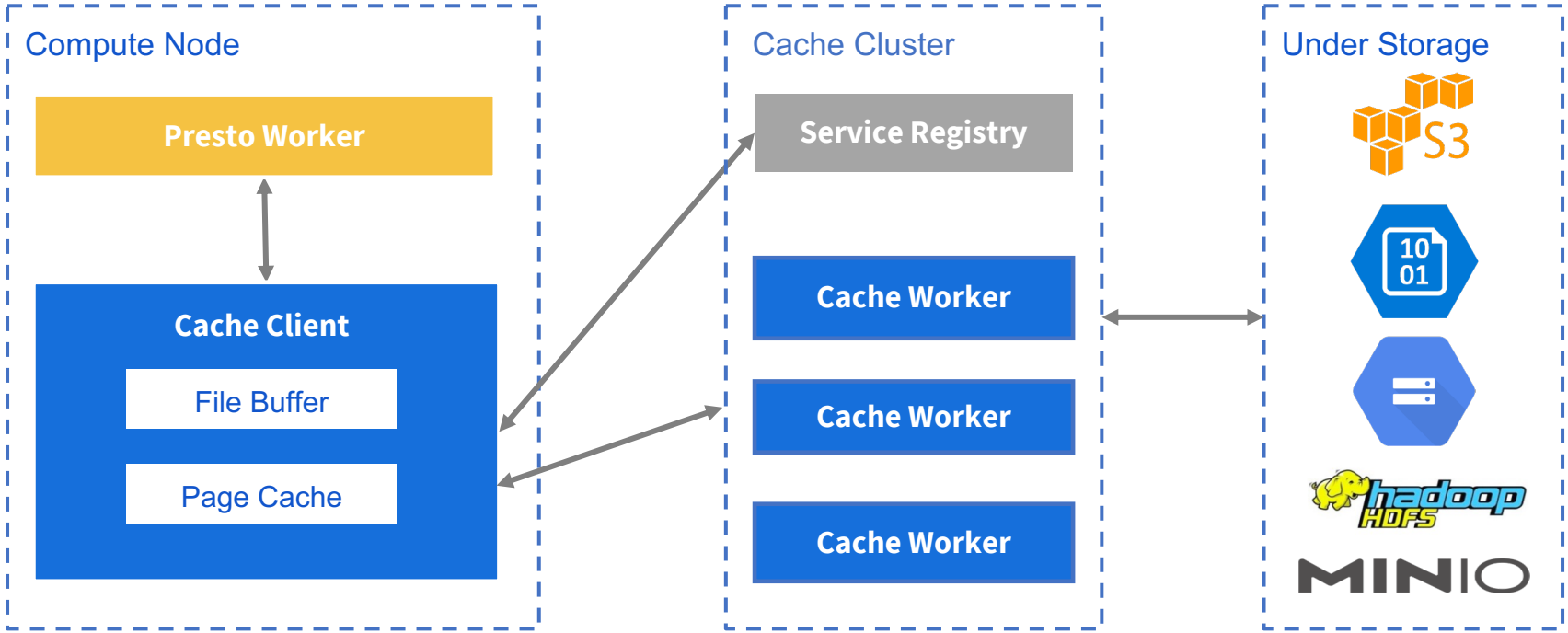
Pattern II: Hybrid/Multi-Cloud for Data Infrastructure



Cache Strategy Recommendation II

- Cloud-friendliness
  - Flexibility for hybrid/multi-cloud
  - Configurable cache admission/eviction policies
- Availability
  - No single point of failure
  - Support fallbacks to under storage
- **Cost-effectiveness**
  - Reduce the number of API calls
  - Reduce the data transfer cost (egress cost)
  - Minimize the read amplification
  - Do not persist temporary files

Integration with SQL Systems



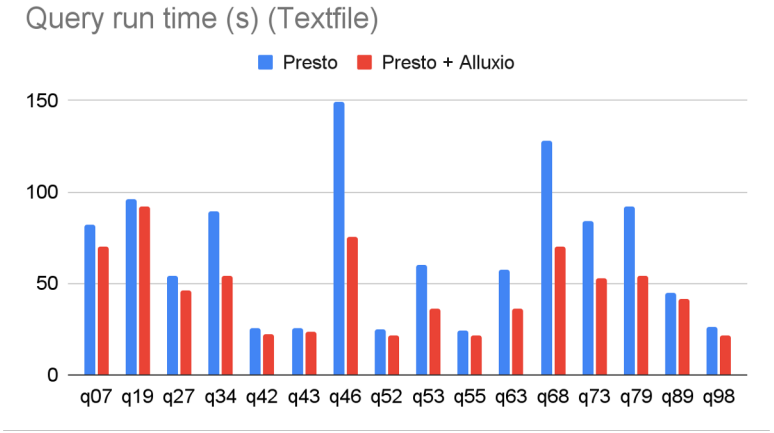
Evaluation

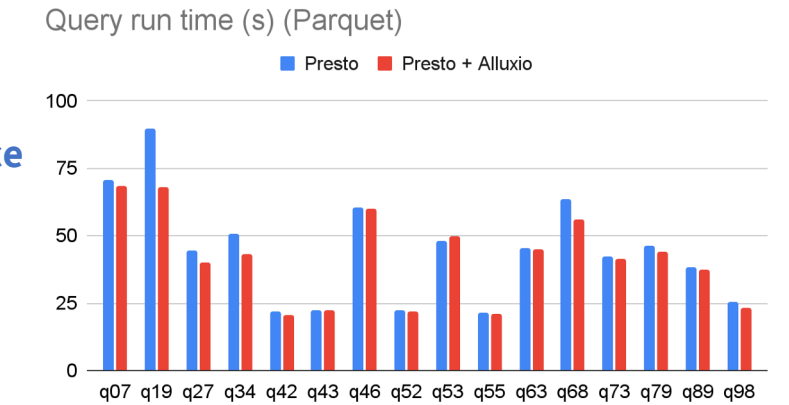
Setup

- Evaluated with Presto and Alluxio
- Deployed Presto with 10 worker nodes
- Tested sampled TPC-DS queries

Observations

- **Presto with Alluxio can achieve up to 2x performance Improvement on text files**
- **Performance gain on Parquet files is not significant**
  - **Positioned reads are common in Parquet files**
  - **Observed read amplification**





Cache Capacity Planning

Cache capacity planning based on real-time metrics of the working set

Motivation

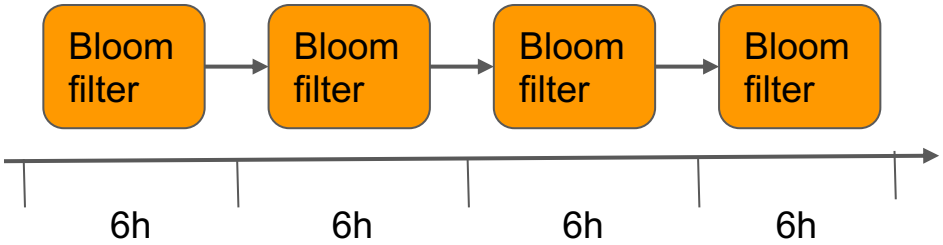
In a multi-tenant SQL system,

- **How to size the cache for each tenant?**
  We need to tell the administrator how many non-duplicate bytes the cache has received in the past 24 hours to estimate the future cache demand.
- **What is the potential cache hit ratio improvement?**
  We need to tell the administrator how many requests hit the cache if the cache can keep all the requests over the last 24 hours.

Shadow Cache Design- Multiple Bloom Filters

Memory efficient solution: Using a chain of Bloom filters, each tracking the unique objects in one time window

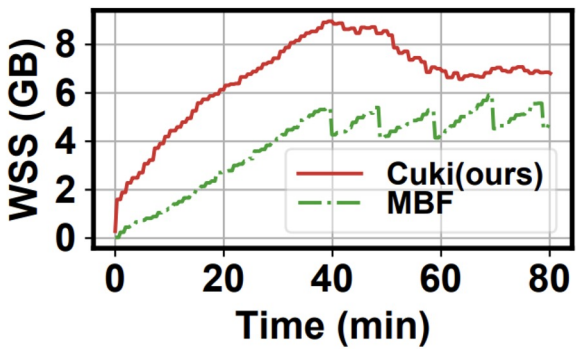
Track working set of 6 TB, only 35 MB memory is required (4 bloom filters, < 3% err)

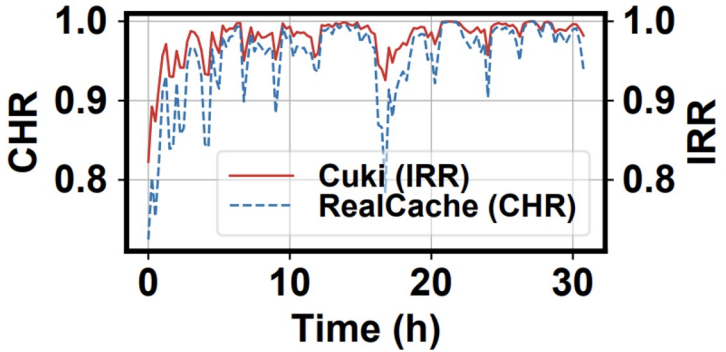


Shadow Cache Design - Cuckoo Filter

Access counting solution: Using a cuckoo filter with aging operation

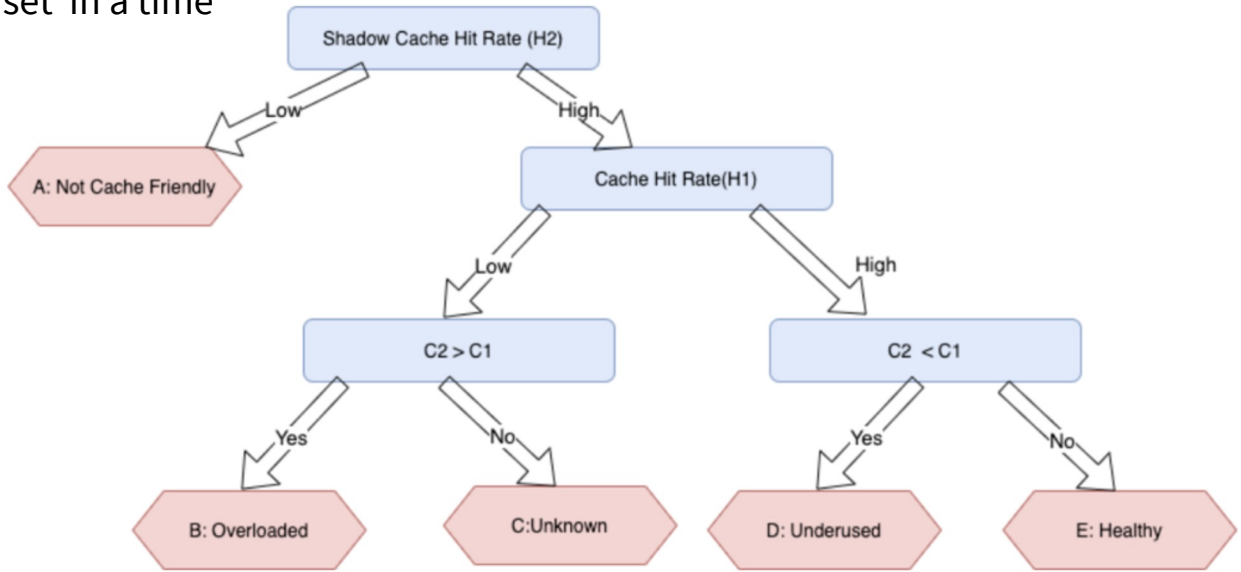
Track working set of 8 TB, only < 400MB memory is required (< 3% err)





Cache Status Estimation

- C1: Real cache usage at a certain point in time
- C2: Shadow cache working set in a time window (1 day / 1 week)
- H1: Real cache hit-rate
- H2: Shadow cache hit-rate

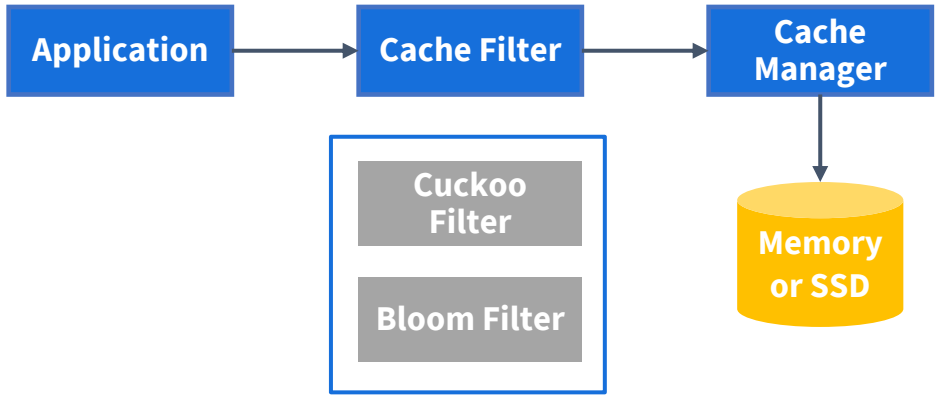


Adaptive Cache Strategy

Adaptive cache admission and eviction for uncertain traffic patterns

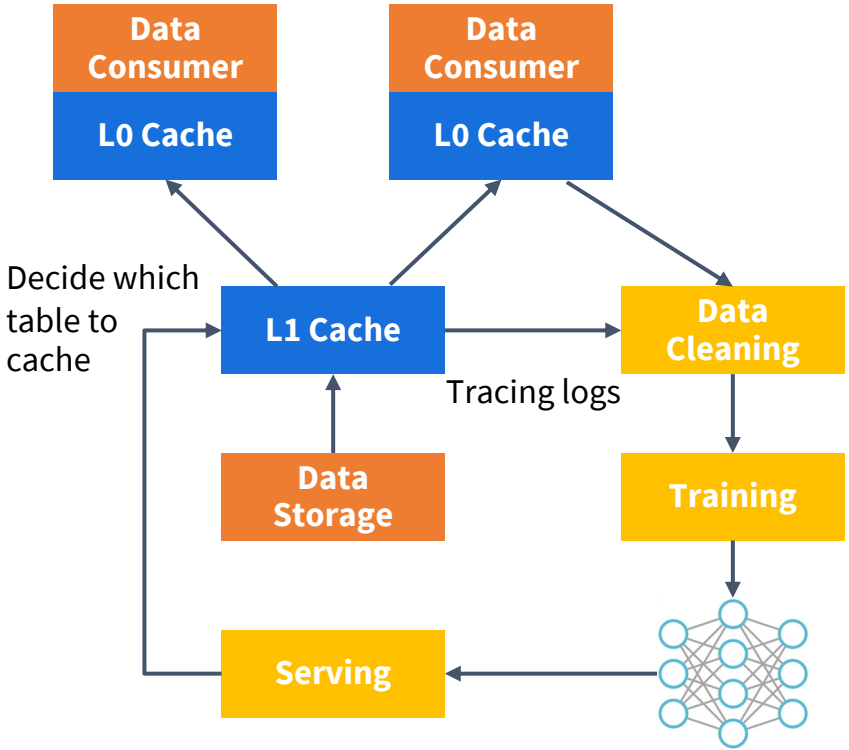
Adaptive Cache Filter

- Based on the data structures that support approximate membership queries such as bloom-filter and cuckoo-filter
- We could use multiple bloom-filters, counting bloom-filter or putting a counter to cuckoo-filter to count the number of visit of each data block.
- The memory consumption might be big for extremely large working sets.



AI-Based Cache Filter

- Distinguish tables with low value and high value.
  - Caching tables with low value will repeatedly flush the storage of L1 cache.
  - Caching tables with high value will result in the cached data being read more times.
- We can provide granularity at the partition or even file level.



Takeaways

Takeaways

- **Positioned read plays a crucial role in cache performance optimization for loading large AI-purposed structured files.**
- **Cache in worker nodes can notably accelerate the computation for data analytics (SQL).**
- **Bloom filter and cuckoo filter are key design blocks to track working sets for cache capacity planning.**
- **More advanced cache strategies such as the adaptive cache strategy are a potential research direction.**

References

- C. Tang, B. Wang et al., “Serving hybrid-cloud SQL interactive queries at Twitter” in European Conference on Software Architecture. Springer, 2022.
- C. Tang, B. Wang et al., “Hybrid-cloud SQL federation system at Twitter” in ECSA (Companion). Springer, 2021.
- “Improving Presto Architectural Decisions with Alluxio Shadow Cache at Meta (Facebook)”, https://www.alluxio.io/blog/improving-presto-architectural-decisions-with-alluxio-shadow-cache-at-meta-facebook/, 2022.
- R. Gu, S. Li, et al., “Adaptive Online Cache Capacity Optimization via Lightweight Working Set Size Estimation at Scale” (in-press) in USENIX ATC, 2023.