Data Caching for Enterprise-Grade Petabyte-Scale OLAP

Authors: Chunxu Tang and Bin Fan, Alluxio; Jing Zhao and Chen Liang, Uber, Inc; Yi Wang and Beinan Wang, Alluxio; Ziyue Qiu, Carnegie Mellon University and Uber, Inc.; Lu Qiu, Bowen Ding, Shouzhuo Sun, Saiguang Che, Jiaming Mai, Shouwei Chen, Yu Zhu, and Jianjian Xie, Alluxio; Yutian (James) Sun, Meta, Inc.; Yao Li and Yangjun Zhang, Uber, Inc.; Ke Wang, Meta, Inc.; Mingmin Chen, Uber, Inc.

Yi (Hope) Wang, Developer Advocate @ Alluxio
Ziyue Qiu, Intern @ Uber & PhD student @ Carnegie Mellon University

TABLE OF CONTENTS

Problem Statement

Motivation: Uber's Data Analytical & Storage Systems

Data Query

Presto®, an open-source distributed SQL query engine for OLAP

Interactive & batch queries

~12,000 weekly active users
~500,000 daily queries
50 PB data
100 PB HDFS bytes read daily
10,000 nodes
20 clusters
2 regions

Data Storage

Apache Hadoop® Distributed File System (**HDFS**)

Supporting data analytical requests like fraud detection, machine learning & ETA calculation

Transitioning to higher-density HDDs to save costs
(from 4TB to 16+TB SKUs)

Motivation: Observations in Workload Characteristics

Data-intensive Analytics at Massive-Scale

Host	Host1	Host 2	Host 3	Host 4
Total reads (M)	13.5	12.8	8.5	14.3
Total writes (K)	3.3	4.7	4.6	45
Reads / writes	4091.0	2723.4	1847.8	317.8
Read traffic on top 10K blocks	89%	94%	99%	99%

Table: Production traffic of Uber's HDFS clusters

Skewness in Data Access Patterns

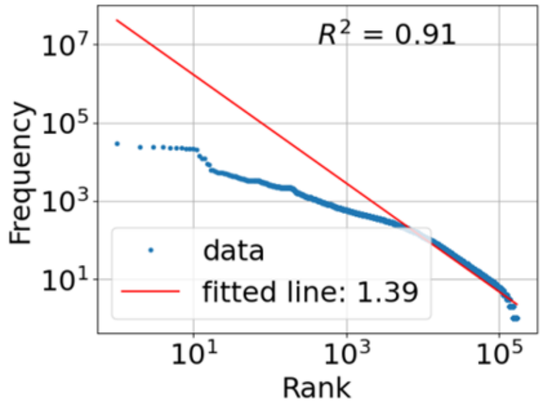


Figure: Popularity rank and Zipfian distribution, indicating a heavy skewness in data access distribution

Limited I/O Bandwidth with High-Density Disks

- I/O bandwidth of each HDD cannot match the HDD capacity
- I/O throttling within HDFS DataNodes
- Several thousand of blocked processes within one minute

Fragmented Access to Columnar Files

- Columnar formats (Apache Parquet®, Apache ORC®) are favored for read operation efficiency
- > 50% SQL requests on HDFS access less than 10 KB of data
- > 90% involve less than 1 MB

TAKEAWAY

A Simple & Flexible Cache Layer Is Needed

REQUIREMENTS

Design Requirements

- An application-agnostic cache
- Minimal socket communication with applications
- Non-volatile data storage
- Capability to support both random access and sequential data access
- High performance

ARCHITECTURAL DESIGN

Overall Architectural Design

- Goal: Enhance **read** performance
- Admission Controller
  - Prioritize the most frequently accessed files
  - Only admit essential data
- Page Store
  - Transforms file-level read operations into more granular page-level operations
- Cache Manager
  - Hit or miss
    - On cache hit: return requested page
    - On cache miss: fetch page from external data sources, cache locally, and serve to the client
  - Cache eviction
    - Policy- based: FIFO, random, LRU
    - Time-based: set time-to-live

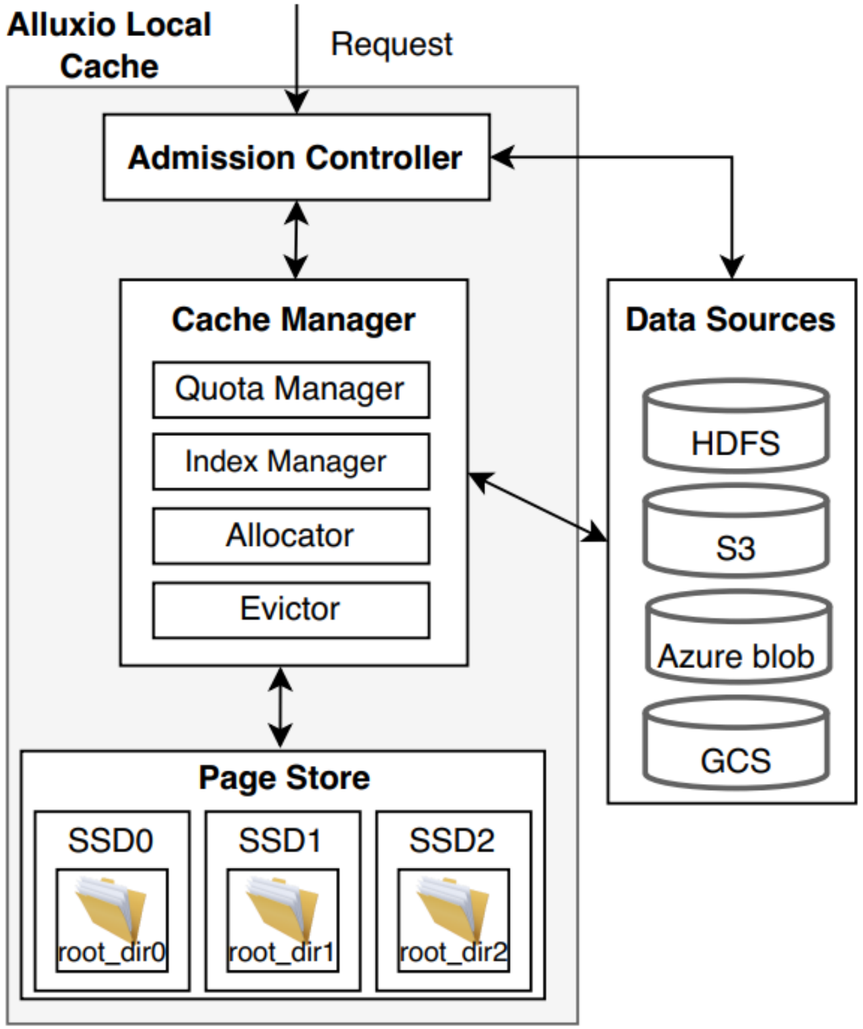


Figure: The workflow of Alluxio local cache

Basic Design

Leverage Local SSDs

- Not constrained by the available memory in the node
- Balance between cost-efficiency and high-speed access
- Local SSDs are usually underutilized

Page-based Storage

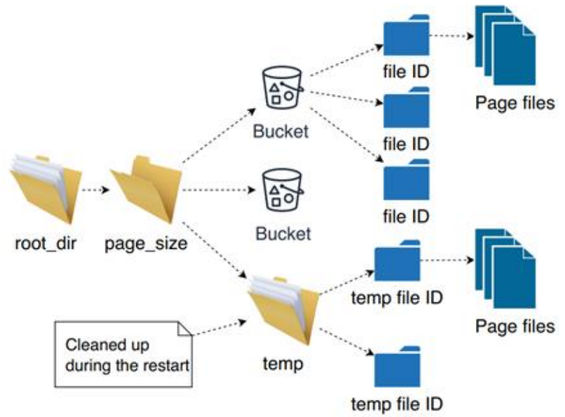


Figure: Organization of cached data files

Simple & Efficient Index

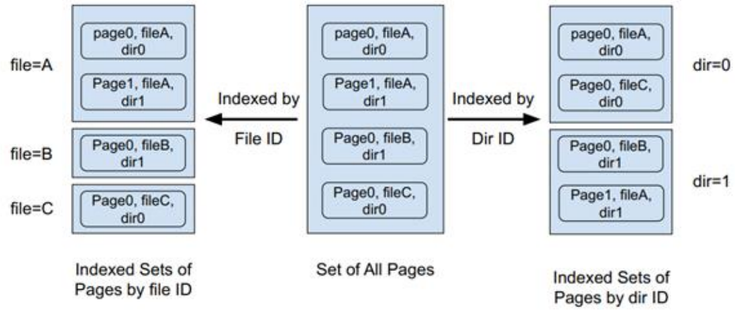


Figure: Using indexed sets for organizing pages

CACHE OPTIMIZATIONS & IMPROVEMENTS

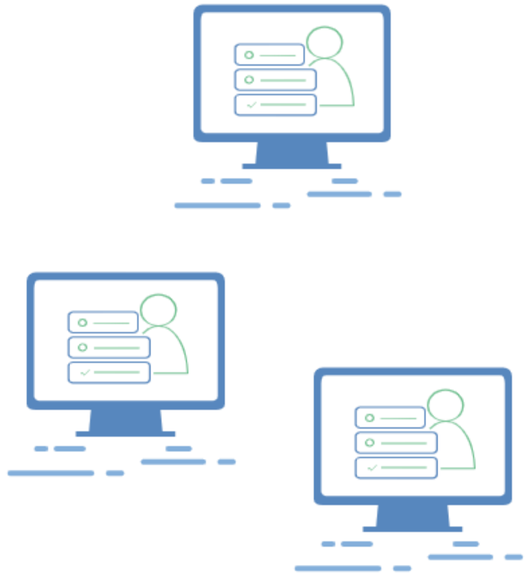
Optimization: Cache Admission Strategies

- Regular expressions or JSON-format expressions
  - Example: Presto local cache
  - Static filtering rules applied to tables
  - Limits on cached partitions set by platform owners
  - At Uber
    - <10% of requests need remote storage access after filtering
- Analysis of historical data access patterns
  - Uses sliding windows to assess data block "hotness"
  - Example: HDFS local cache
    - Only ~1% requests require slower storage access

```
{
  "databases": [
    {
      "name": "database_foo",
      "tables": [
        {
          "name": "table_bar",
          "maxCachedPartitions": 100
        }
      ]
    }
  ]
}
```

Optimization: Quota Management for Multi-Tenancy

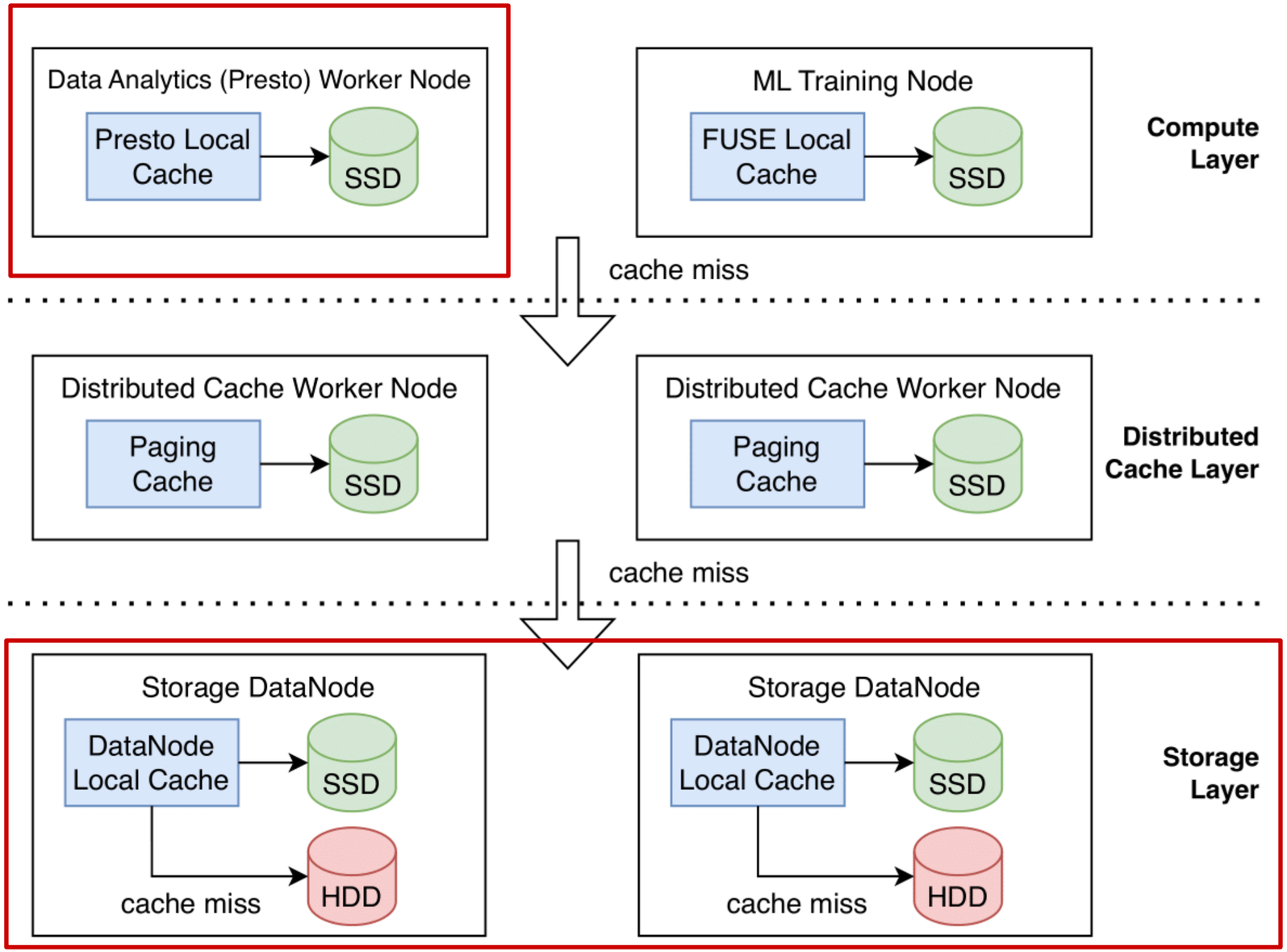
- Flexible multi-layer quota management
  - User tenants (teams, units, regions)
  - Data tenants (partitions, tables)
  - Custom tenants (projects, applications)
- Hierarchical quota verification process
- Quota violation handling
  - Partition-level eviction
  - Table-level sharing and random eviction
- Benefits
  - Fairness
  - Efficient resource utilization
  - Prevention of resource monopolization



USE CASES & EVALUATION

Use Case Overview

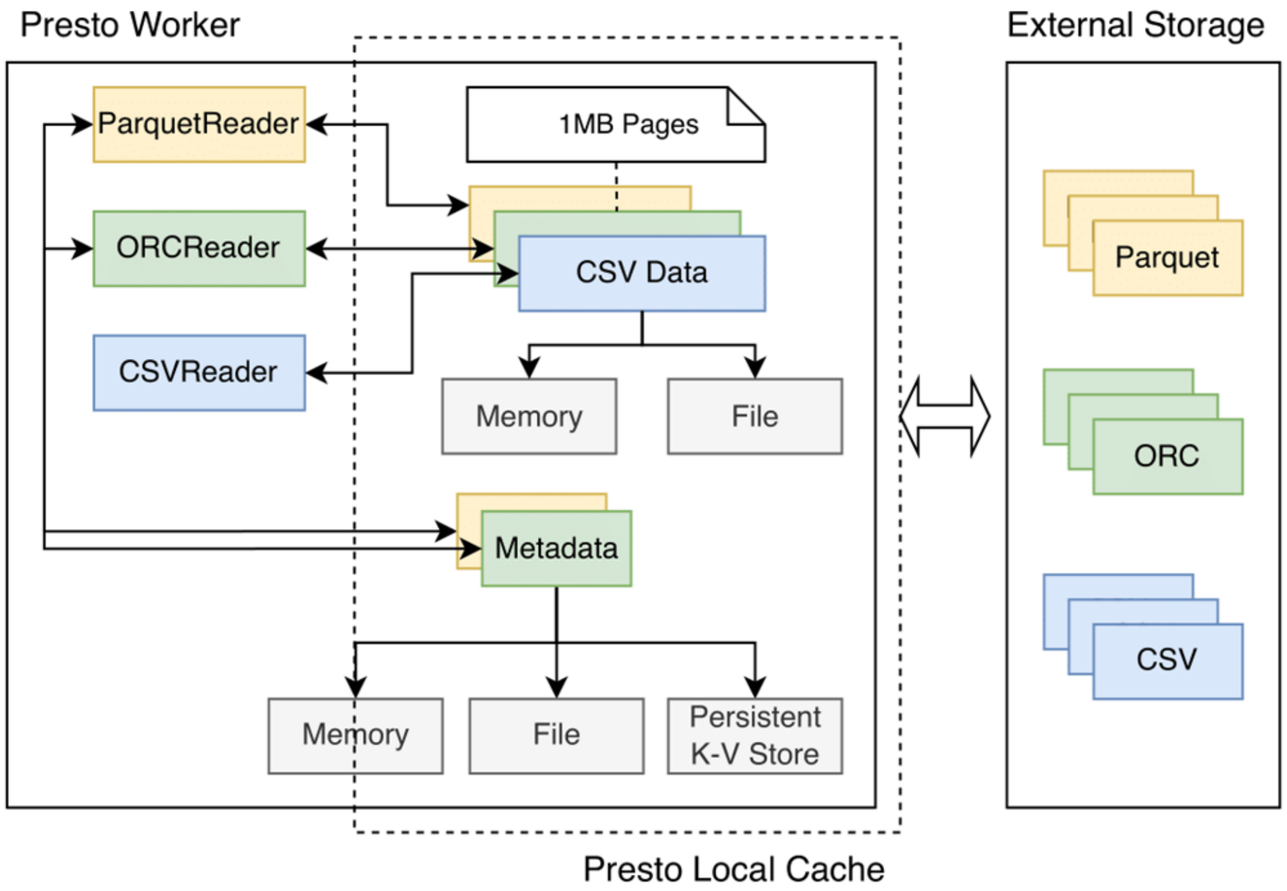
Case Study 1:
Presto Local Cache



Case Study 2:
HDFS Local Cache

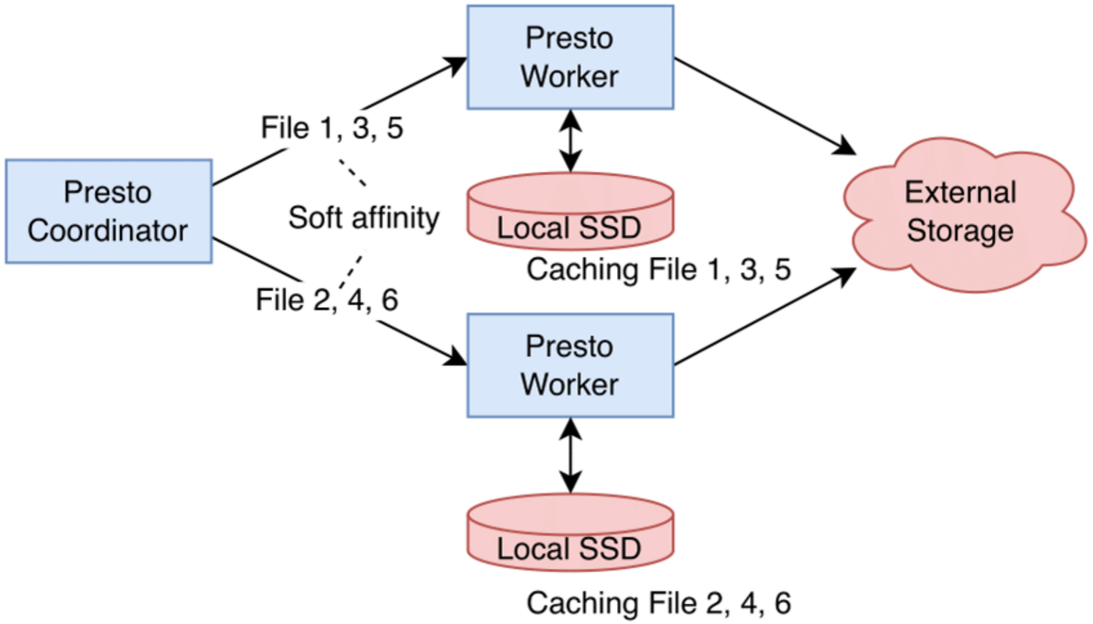
Case Study 1: Presto® Local Cache Overview

- Supports various file reader classes
  - stores data in memory or files
- Caches file metadata as well
  - mitigates CPU usage on file parsing
  - stores in memory, files, or K-V Store
- Employs a read-through strategy



Case Study 1: Presto® Local Cache - Soft-Affinity Scheduler

- Background
  - Presto segments data into row groups (to expedite data processing)
  - Coordinator distributes splits among the worker nodes
- Conventionally, evenly distribute tasks by randomly assigning splits to workers
  - inefficient for caching
  - lead to frequent admission and eviction of data from each worker's local cache
- Soft-affinity Scheduler: allocate splits from the same file to the same worker node



Case Study 1: Presto® Local Cache Evaluation

- Metrics: time spent on reading Parquet files
- At Uber
  - Quantified by ScanFilterProjectOperator (responsible for data input and initial filtering)
  - Over 97% of queries involve reading through this operator

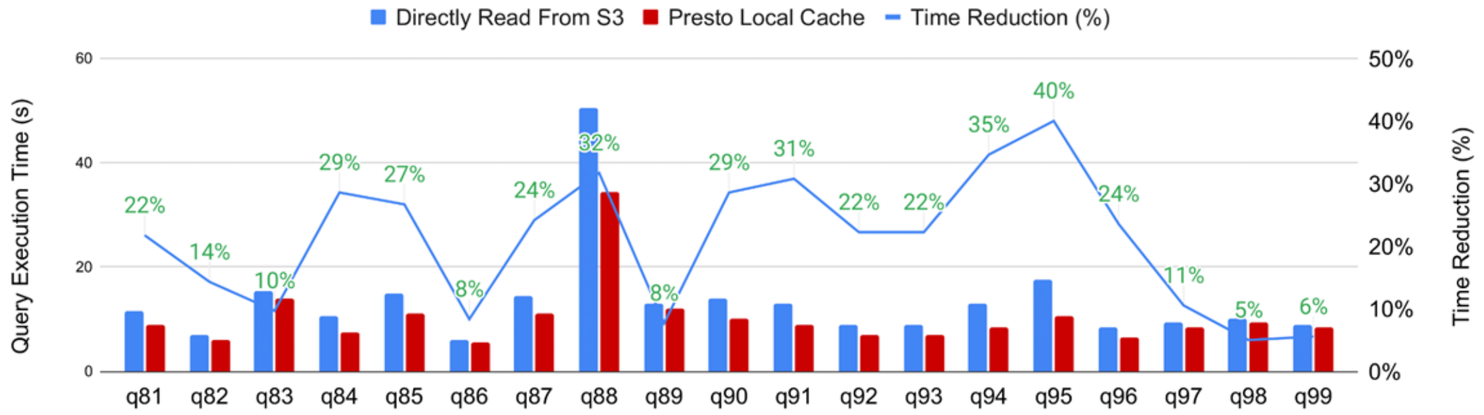
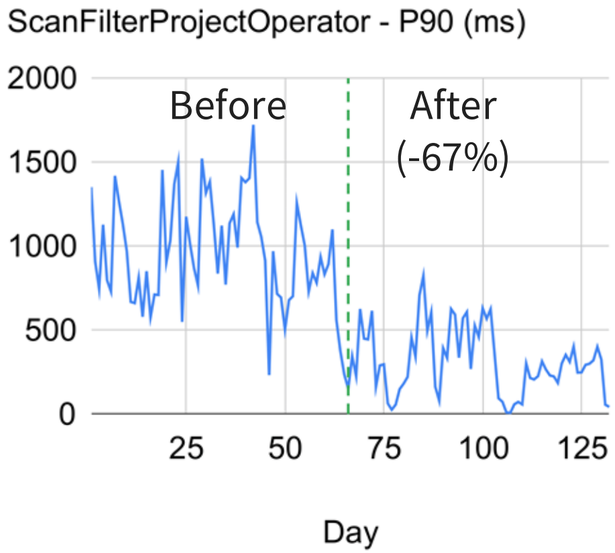
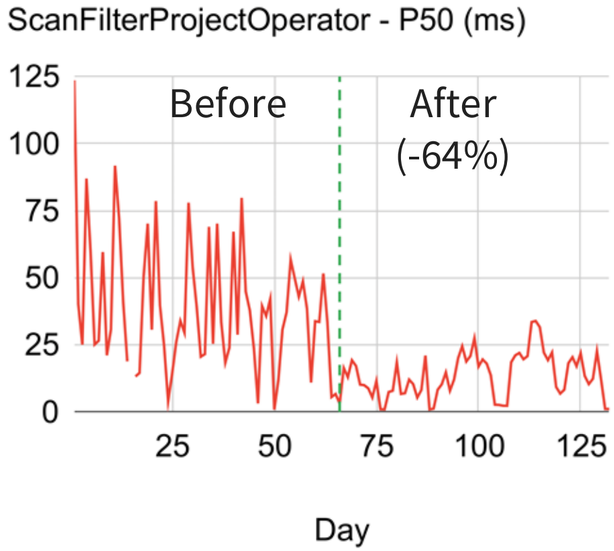


Figure: Query execution time of TPC-DS (Query 81 to 99) without and with Presto local cache (see paper)



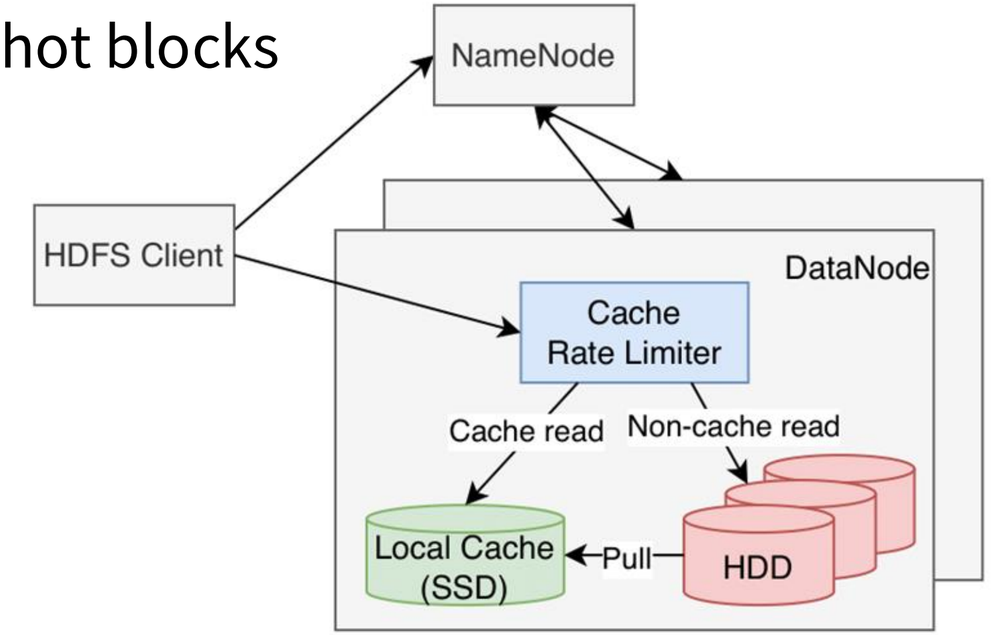


Case Study 2: HDFS Datanode Local Cache

- Rate limiter each DataNode: avoid frequent cache admission/eviction
  - based on historical access patterns, determine admission or not
  - See paper for details
- Why not distributed cache?
  - local caching already offers a substantial return on investment (ROI)
  - Traffic focuses on a small group of extremely hot blocks
  - Top 10K blocks attracted >90% read traffic

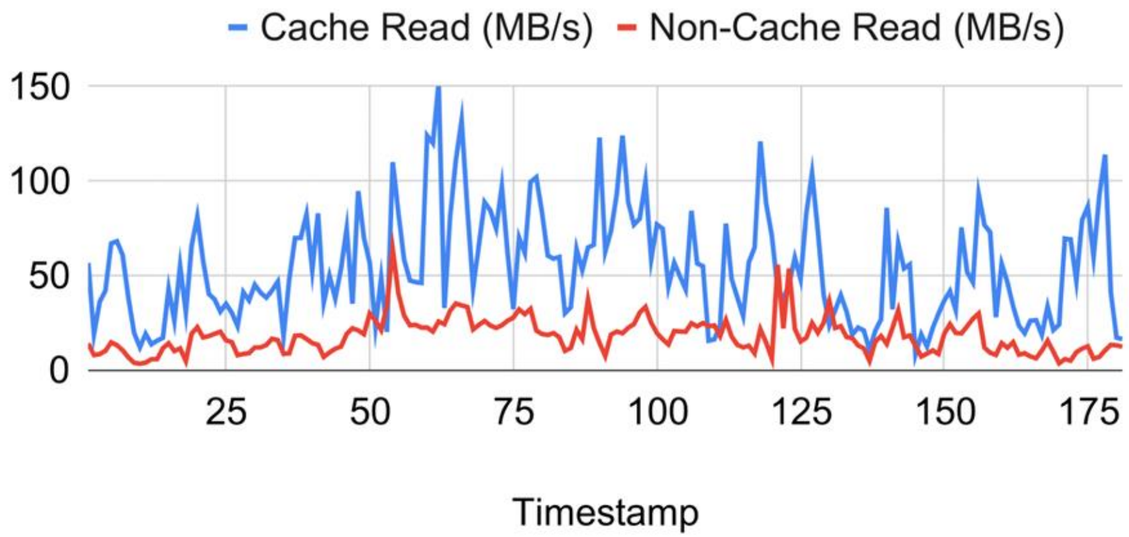
Host	Host1	Host 2	Host 3	Host 4
Total reads (M)	13.5	12.8	8.5	14.3
Total writes (K)	3.3	4.7	4.6	45
Reads / writes	4091.0	2723.4	1847.8	317.8
Read traffic on top 10K blocks	89%	94%	99%	99%

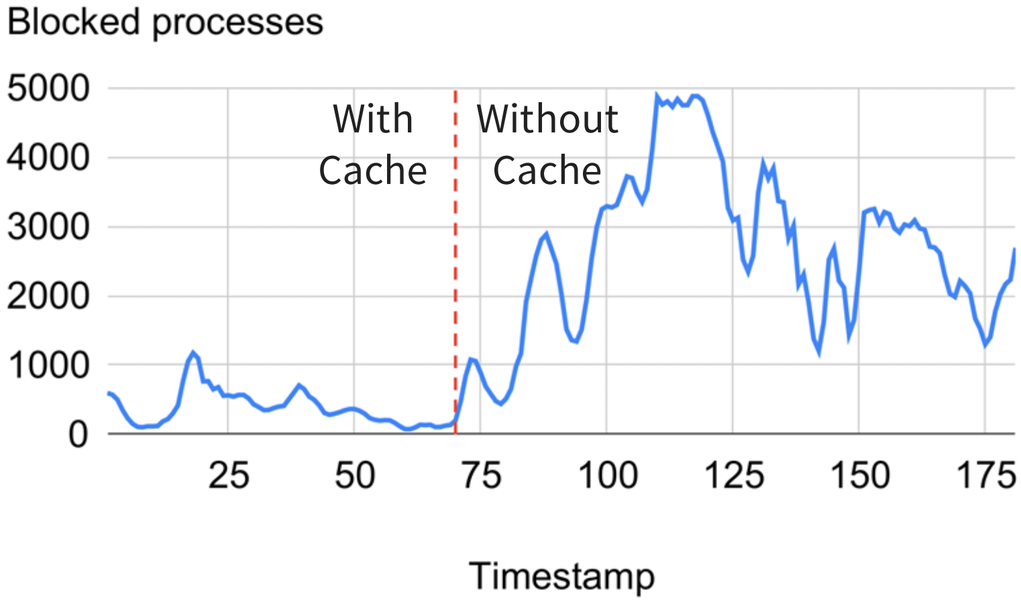
Table: Production traffic of Uber's HDFS clusters



Case Study 2: HDFS Datanode Local Cache

- Read bytes: served >70%
- The number of processes blocked by IO: reduced by an average of 86%





LESSONS LEARNED

Lessons Learned

1. Caching metadata can significantly alleviate CPU overhead
2. Tuning the cache page size from 1 MB
   a. trade-off between read amplification and metadata footprint/# of remote IOs
3. An aggregated metrics system is crucial for cache tuning and debugging
4. Keeping the seats for temporary offline nodes
5. Considering data imbalance and fallback when setting the number of cache replicas
6. Trade-offs between local cache and distributed cache

FAILURE CASE STUDY

Failure Case Study

Case	Solution
File read hanging	fallback to remote storage
Corrupted files	early eviction
Insufficient disk capacity	catch the “No space left on device” exception, early eviction

Join Alluxio community
https://alluxio.io/slack

Q & A

Try Alluxio local cache yourself

- Alluxio local cache for Presto: https://prestodb.io/docs/current/cache/local.html
- Alluxio local cache for Trino: https://trino.io/docs/current/object-storage/file-system-cache.html
- Alluxio Github: https://github.com/Alluxio/alluxio