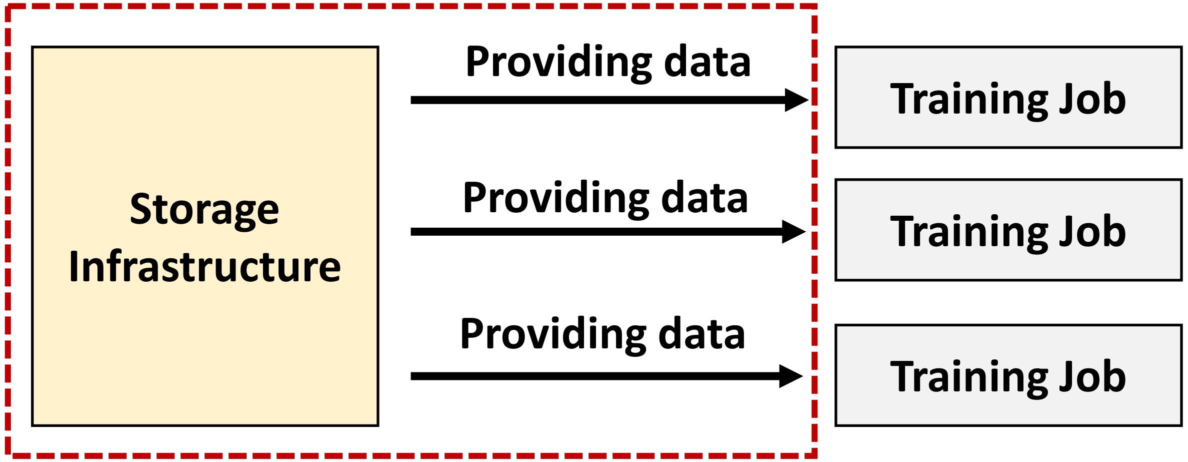
Tectonic-Shift: A Composite Storage Fabric for Large-Scale ML Training

ATC'23

Background

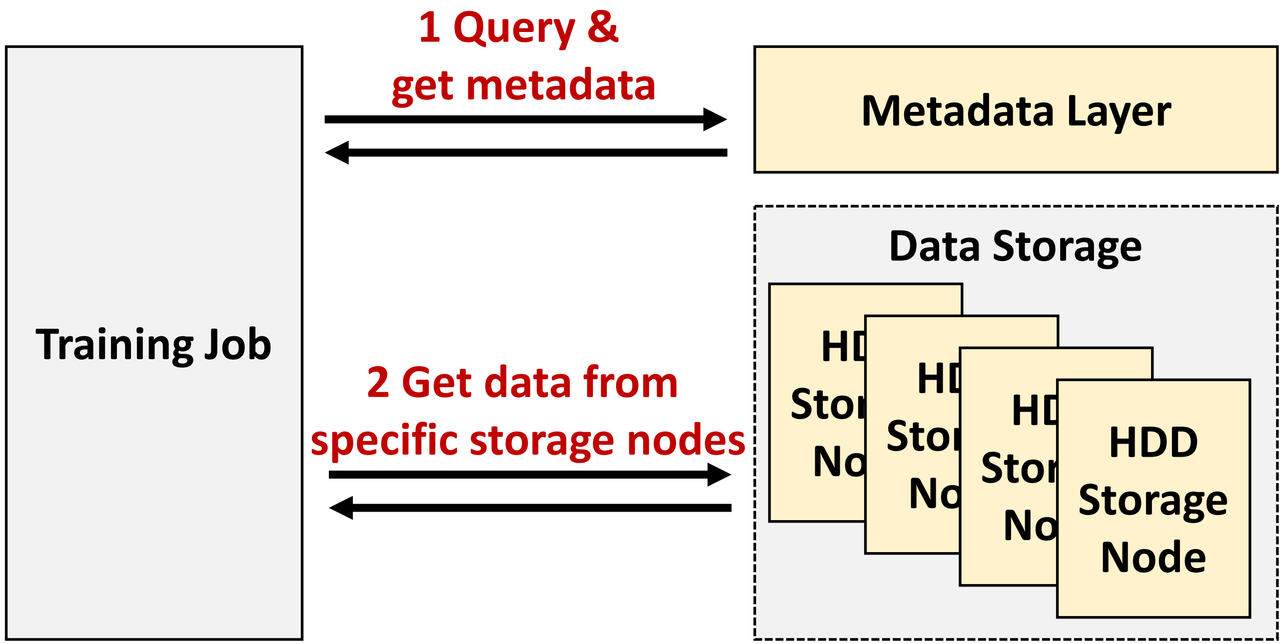
- Machine learning (ML) has been very successful and widely used in industry
- The key in machine learning is model training based on massive amounts of data
  - An efficient and scalable data storage infrastructure becomes an important part of the training process



How efficiently the data is provided affects how fast the training is done

Background

➢ Tectonic: a prior storage fabric for Meta before this paper



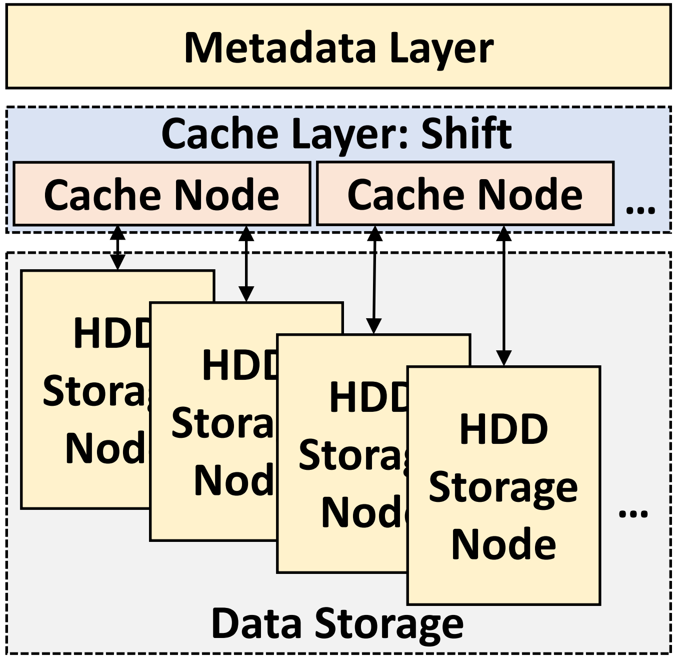
Problems & Goals

- Increasing demand for IO for training tasks
- HDD storage nodes in Tectonic can't meet growth IO demands
  - HDDs have low storage cost overhead but high IO overhead (about 10x over storage cost)

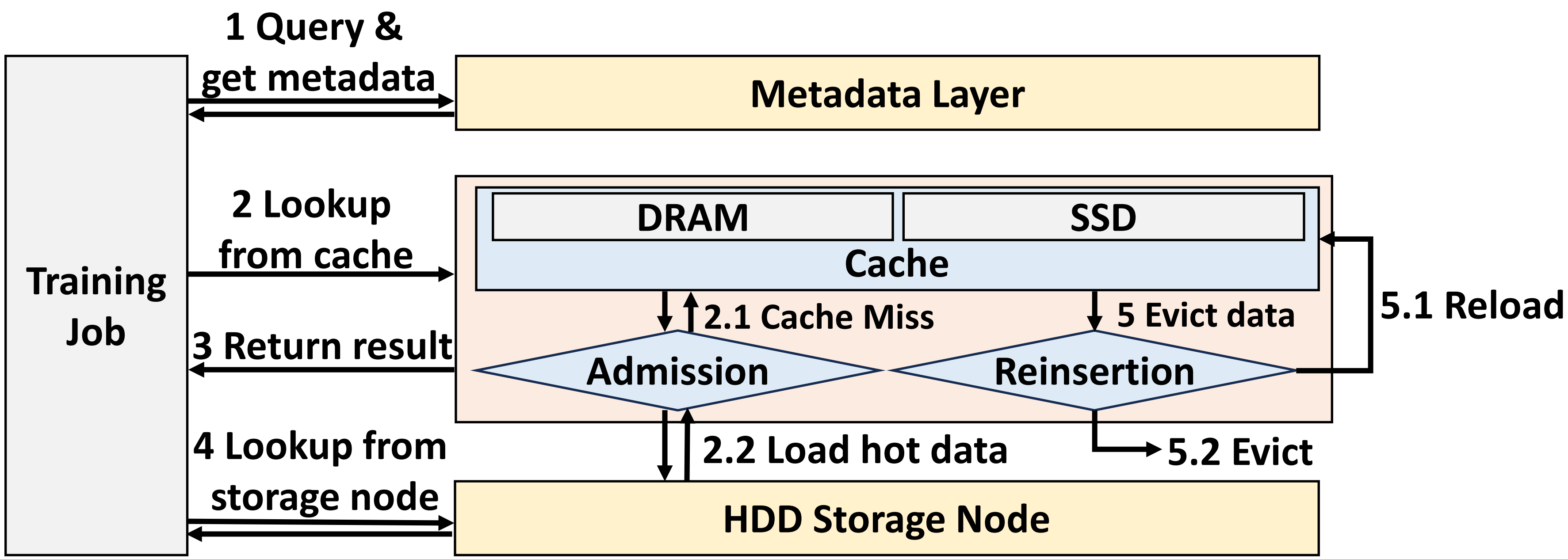
- Goals: exploring a new file system for training tasks
  - IO efficient
  - User-transparent
  - Scalable

Main Idea

- Present a new storage fabric **Tectonic-Shift** for Meta's production machine learning training infrastructure
  - Add a cache layer called Shift based on Tectonic, achieving efficient IO
  - Provides the same interface for training tasks, achieving user transparency
  - Completely independent of the Tectonic and consists of a number of cache nodes named Shift Node, achieving high scalability

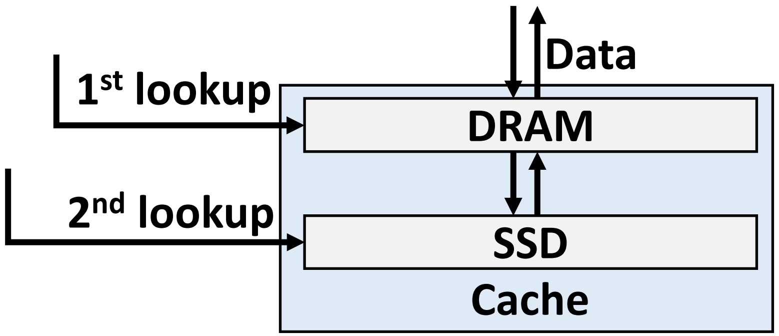


Data Reading Process



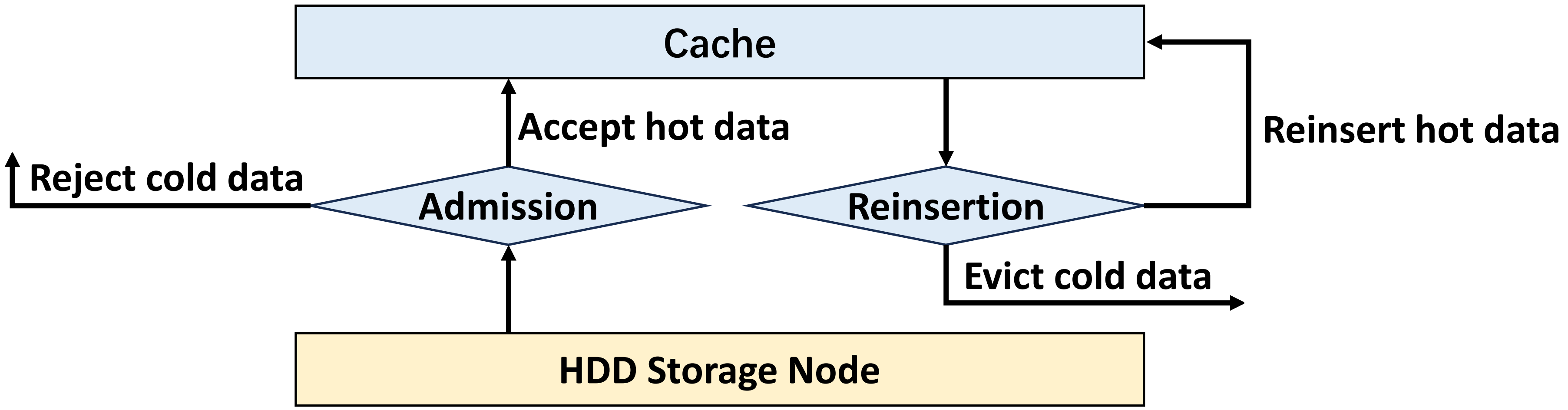
Design1: Cache

- Uses both DRAM and SSD for data storage
  - Just using DRAM tends to make the network interface card an IO bottleneck
  - SSD only have about 20% the IO overhead compared to HDD



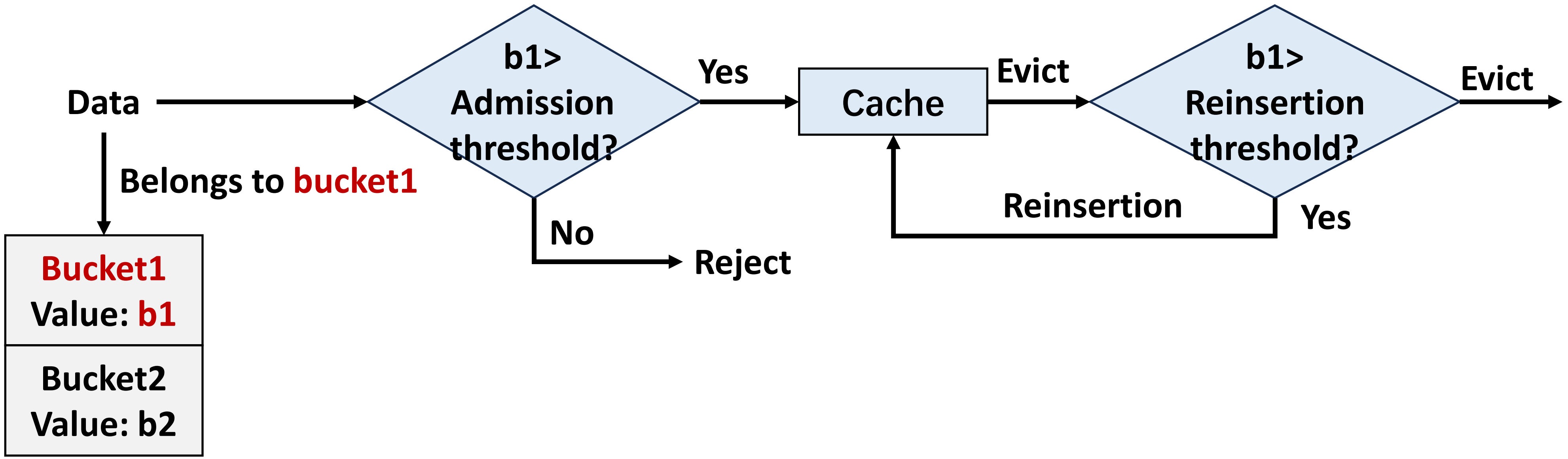
Design2: Policy

- Shift will make additional judgments when loading data and evicting data
  - Just accept hot data into cache and reinsert part of them when eviction
- Rationality stems from 3 reasons
  - 1. SSD and DRAM have greater storage overhead (about 5x) compared to HDD
  - 2. Jobs share frequently accessed data and also access some unique data, resulting in cache pollution
  - 3. Reinsertion further reduces HDD IO and improves hardware lifetime



Design2: Policy

- Using bucket to manage data
  - A bucket is a set of data, cache makes one policy for a bucket
  - Increase granularity to reduce management overhead
- Shift designs 2 policy: history and future
  - 1. History: use the historical access frequency of the data in the bucket to determine
  - 2. Future: depending on each job's schedule for data reading

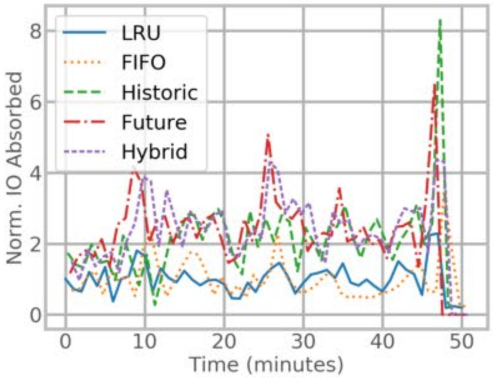


Evaluation

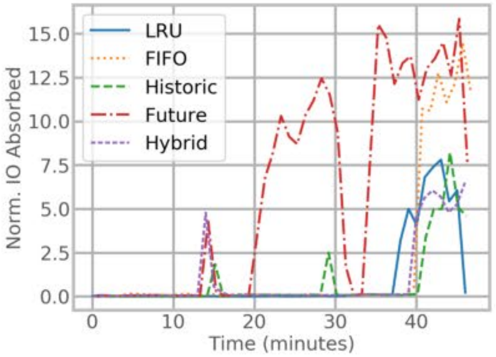
Workload	Jobs & Partitions Read	Description
Synchronized	$\{P_1, P_2, P_3\}_1$, $\{P_4, P_5, P_6\}_2$, $\{P_1, P_2, P_3\}_3$, $\{P_7, P_8, P_9\}_4$, $\{P_1, P_2, P_3\}_5$	Multi-tenant HP tuning or exploratory jobs. Jobs are launched synchronously.
Pipelined	$\{P_1, P_2, P_3\}_1$, $\{P_2, P_3\}_2$, $\{P_3\}_3$	Long-running, pipelined jobs. Jobs are launched synchronously.
Sequential	$\{P_1\}_1$, $\{P_1\}_2$, $\{P_1\}_3$, $\{P_1\}_4$, $\{P_2\}_5$, $\{P_3\}_6$, $\{P_1\}_7$, $\{P_4\}_8$, $\{P_5\}_9$	Queued jobs that launch when training capacity is available. Jobs 1-3, 4-6, and 7-9 launch together.

Evaluation

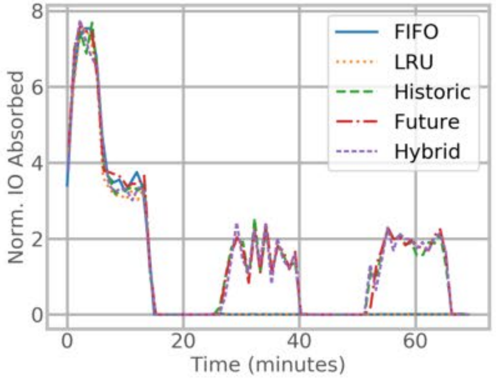
Workload	Jobs & Partitions Read
Synchronized	$\{P_1,P_2,P_3\}_1$, $\{P_4,P_5,P_6\}_2$, $\{P_1,P_2,P_3\}_3$, $\{P_7,P_8,P_9\}_4$, $\{P_1,P_2,P_3\}_5$
Pipelined	$\{P_1,P_2,P_3\}_1$, $\{P_2,P_3\}_2$, $\{P_3\}_3$
Sequential	$\{P_1\}_1$, $\{P_1\}_2$, $\{P_1\}_3$, $\{P_1\}_4$, $\{P_2\}_5$, $\{P_3\}_6$, $\{P_1\}_7$, $\{P_4\}_8$, $\{P_5\}_9$



(a) Synchronized workload



(b) Pipelined workload



(c) Sequential workload

- All new policies is better since they do not load P7 P8 P9 for job4

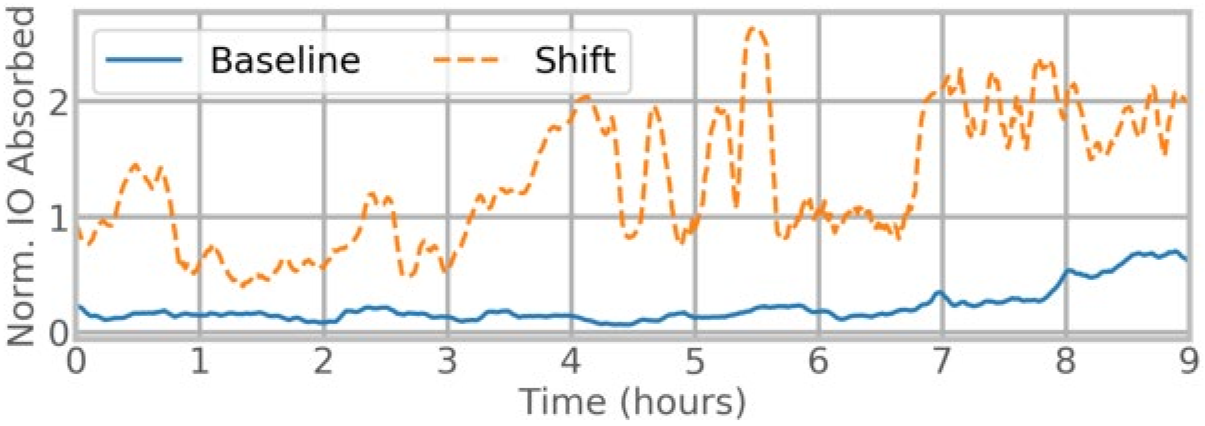
- Future is better since it just keeps P3 in cache
- Historic is worse since it load P2 and evict P3 when doing job1

- All new policies is better since they do not load P2 and P5

Evaluation

	Normalized Hit Rate	Normalized HDD IO
Hybrid with Reinsertion	1.03	0.82

Evaluation



Conclusion

- High IO requirements for training tasks cause existing data systems to become unsuitable
- The authors explore and propose a new file system Tectonic-Shift, a file system builds on Tectonic and improves IO by cache
- Uses DRAM and SSD to store cached data, realizing high IO while saving storage costs
- Introduces admission and reinsertion on top of the existing common cache structure to further improve cache utilization