Slide 2 - Introduction

On systems like Frontier, deep learning training jobs spend up to 70% of time on I/O. 25% of these jobs fail - and half of those are due to node outages. Solutions like HVAC cache data locally to avoid PFS load, but they lack fault tolerance. When a node fails, cached data is lost - that's the problem this work addresses.

Slide 3 - Frontier Failures

A six-month Frontier log analysis showed that 25% of training jobs failed - and over 1,100 of those were from node outages. Even rare, node failure is costly because it erases the local cache, forcing a fallback to slower storage.

Slide 4 - Background & Objective

CosmoFlow repeatedly reads thousands of small files, causing I/O bottlenecks. HVAC solves this with NVMe caching, but it lacks fault recovery. This work extends HVAC using a consistent hash ring to automatically recache lost data and rebalance load after failure.

Slide 5 - Fallback vs Hash Ring

Here's how the fault-tolerant caching system compares to HVAC. On the left, node failure forces a fallback to PFS. On the right, we use a hash ring to automatically recache lost files and redistribute data to remaining nodes. This avoids unnecessary PFS access and maintains training speed.

Slide 6 - Hash Ring, TTL, Warm Cache

To test this, I simulated a node failure right after the first epoch - once the cache was fully populated. I used a hash ring with 100 virtual nodes per physical node, and added TTL-based eviction to avoid unbounded cache growth.

Slide 7 - Failure, Recache, Rebalance

Failure is simulated by removing a node and clearing its cache. The system then recaches missing files from PFS using updated ring mapping. When the node returns, we rebalance the cache to restore even distribution.

Slide 8 - Implementation Summary

I implemented a simplified version in C++, using consistent hashing and synthetic file simulation. Recovery was tested in two modes: NVMe recache and PFS fallback. I logged metrics like runtime, file movement, and node load.

Slide 9 - Experiment Setup

The experiment used three simulated nodes and about 300 synthetic files. After cache warm-up, one node was removed to simulate failure. The system then recached and rebalanced using the hash ring.

Slide 10 - Recovery Timing

The paper showed that NVMe recovery adds just 12.5% runtime overhead, while PFS fallback adds up to 68.7%. In my smaller setup, I measured total access times over 4 epochs and found that NVMe was significantly faster.

Slide 11 - Speedup Calculation

Across 150 trials, NVMe access averaged 146 milliseconds, compared to 424 milliseconds with PFS. That's a 65.5% speedup. This result shows that even in small-scale systems, fault-tolerant NVMe caching can dramatically reduce recovery overhead.

Slide 12 - Paper vs. My Results

Here's a side-by-side comparison. The paper reported a 14.8% improvement with NVMe. In my 3-node simulation, I saw a 60% improvement. The absolute values differ, but the trend is consistent - NVMe caching outperforms PFS fallback.

Slide 13 - Visual Results Comparison

On the left, you see the paper's results at scale. On the right, my 3-node simulation shows the same performance pattern - NVMe consistently outpaces fallback in access time. This confirms the scalability of the caching model.

Slide 14 - Conclusion & Future Work

In summary, this fault-tolerant design reduces PFS pressure and ensures data availability after node failures. While this is a prototype, future work includes smarter eviction, health monitoring, and persistent metadata to make the cache truly resilient.

Slide 15 - Q&A

Thank you for listening - I'm happy to take any questions.