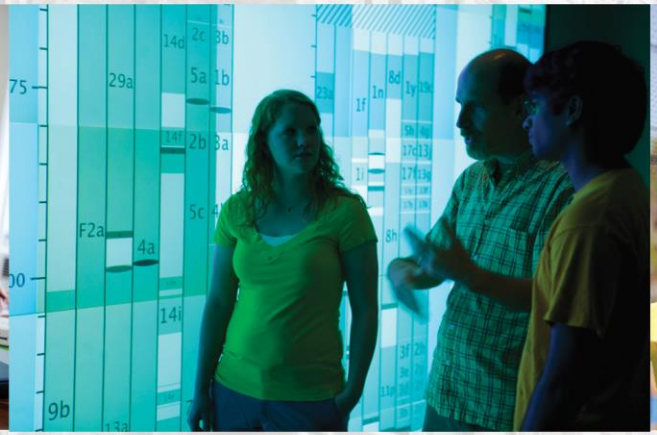
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

Towards enabling cooperation between scheduler and storage layer to improve job performance

Mayank Pundir, John C. Bellessa, Shadi A. Noghabi, Cristina L. Abad, Roy H. Campbell

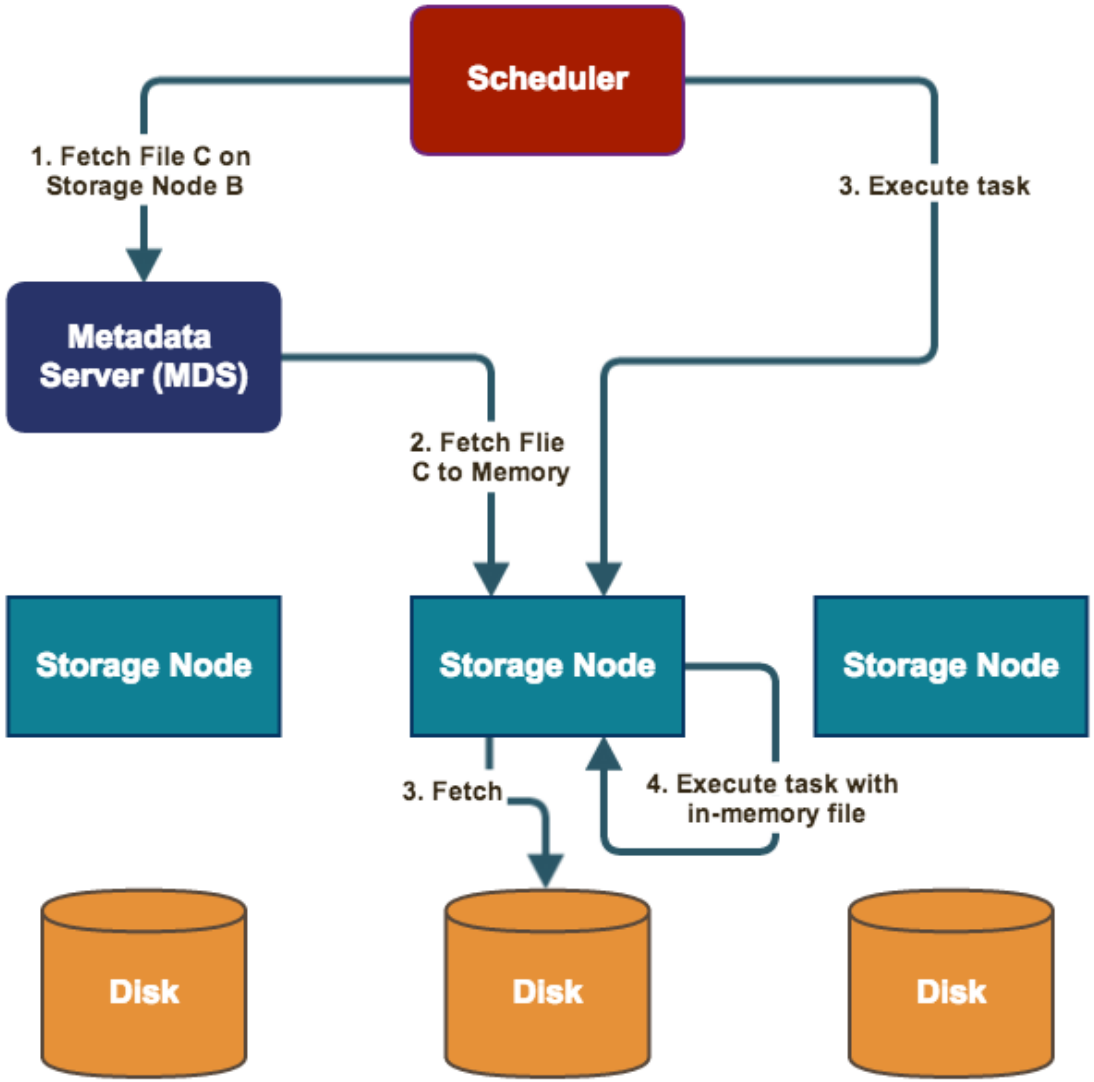


Introduction

- In distributed, data-intensive frameworks, fast access to data is critical
- One way to improve data access time is data locality
  - i.e., tasks should always execute on the node storing the data
- Existing approaches
  - Replication (disk locality)
  - Caching (memory locality)
- Our approach: Scheduled Caching
  - Memory locality at *just the right time*

Scheduled Caching

- Enable coordination between the scheduler and the storage layer
- Scheduler is aware of…
  - *What* data certain tasks will require
  - *Where* the tasks will be executed
  - *When* the tasks will be executed
- Provides hints to the storage layer
  - Fetch file
  - Do not evict file


Scheduler
1. Fetch File C on Storage Node B
3. Execute task
Metadata Server (MDS)
2. Fetch Flie C to Memory
Storage Node
Storage Node
Storage Node
3. Fetch
4. Execute task with in-memory file
Disk
Disk
Disk

Proof-of-Concept

- Implemented using a simplified version of the architecture for Hadoop
- Dual-HDFS System
  - one layer on disk
  - another layer in memory (uses tmpfs)
- Simplified problem: Fetch files from disk HDFS to memory HDFS when scheduler gives the pre-fetch hint

Evaluation

- SWIM benchmark
  - Facebook-like Hadoop workload
  - 46 jobs with varying levels of map-heavy, shuffle-heavy and reduce-heavy jobs

- Compared 3 configurations
  - Unmodified, on-disk HDFS
  - Unmodified, in-memory HDFS (represents ideal case)
  - Dual-HDFS with Scheduled Caching

Results

	Unmodified, on-disk	Unmodified, in-memory	Dual-HDFS w/ pre-fetching
Average Job Completion Time	2574.96 seconds	1105.37 seconds	1130.69 seconds

- With pre-fetching, we saw an average performance improvement of about 2.3x over disk.
- Incurs a penalty of only 2.3% over ideal configuration

Future Work

- Need a model for the workloads we are targeting
  - I/O-intensive vs. computationally intensive
- Intelligent Cache-eviction
  - Aging algorithms
- Streaming input
  - Enable tasks to begin processing before entire input is available
  - Also addresses files too large for available memory
- Intelligent Scheduling
  - Schedule tasks whose data are already in memory

THANK YOU!