Towards Enabling Cooperation Between Scheduler and Storage Layer to Improve Job Performance

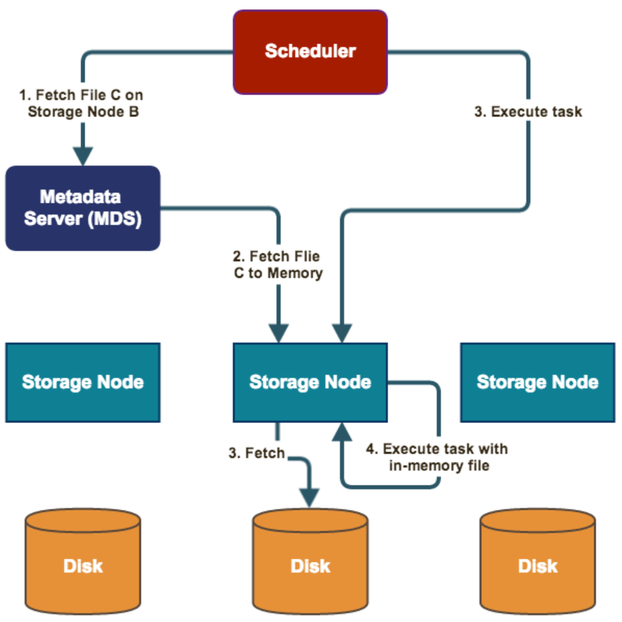
Mayank Pundir, John Bellessa, Shadi A. Noghabi, Cristina L. Abad, Roy H. Campbell
Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign

Motivation

- Fast access to data is critical in data-intensive computing applications
- Fast access achieved by data locality
- Existing approaches deal with “hot” data
  - Replication
  - Caching
- These approaches do not improve the first access to files, as they have not yet become hot.

Scheduled Caching

- Locality-aware schedulers know which data a task requires
- To leverage this awareness, we envision collaboration between the scheduler and storage layer
- The scheduler may pass the information to the storage layer as hints
  - *Pre-fetch*: Bring soon-to-be-used files into memory
  - *Do not evict*: Files will be needed in the future
- Scheduler issues hints to a metadata server API



Implementation

- Our proof-of-concept implementation on Hadoop 1.2.1 required less than 100 additional lines of code
- Based on “dual-HDFS” design
  - One instance on disk; another in memory (using tmpfs)
  - fetch() call brings files from on-disk HDFS to in-memory HDFS

More questions?
Contact Mayank: pundir2@illinois.edu

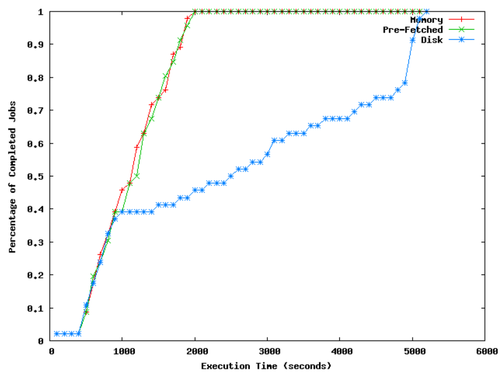
ILLINOIS

Evaluation

We tested our prototype using the SWIM benchmark on an 11 node cluster: one master node with 10 GB of memory; 10 slave nodes, each with 5 GB memory.

We compare performance of three different configurations:
1. Unmodified, on-disk HDFS
2. Unmodified, in-memory HDFS (represents ideal case)
3. HDFS with Scheduled Caching

In both configuration 2 and 3, in-memory HDFS (see Implementation) have half of the system memory available: 5 GB in the master node, and 2.5 GB in the slave nodes.

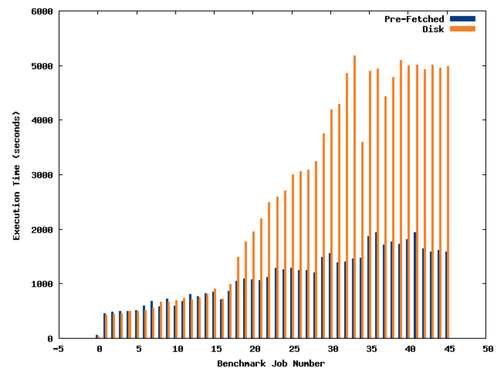


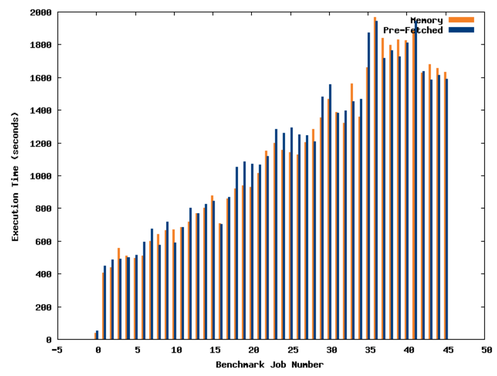
With our prototype, all jobs complete in at most 2000 sec.

Less than half of the jobs have completed with the on-disk version at 2000 sec.

From this figure, we see that many of the tasks in the SWIM benchmark have significant overhead when using the on-disk HDFS.

Our prototype improves average job performance by 2.3x





This figure shows that our implementation performs almost as well as the “ideal” configuration.

In fact, we only incur a 2.3% time penalty.

Future Work

- Our immediate plans are to enable Scheduled Caching in the Hive and Pig frameworks
- When memory presence is achieved, but memory locality is not, RDMA could possibly be employed to speed up network transfer
- Another optimization we have considered is enabling MapReduce tasks to begin operating on partial data; that is, before all the data is available in memory
- A memory-aware scheduler could intelligently schedule tasks whose input data are already in memory ahead of other tasks