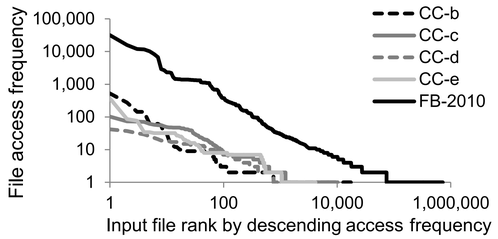
Quartet: Harmonizing task scheduling and caching for cluster computing

Francis Deslauriers, Peter McCormick, George Amvrosiadis, Ashvin Goel, Angela Demke-Brown

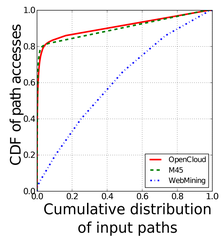
University of Toronto

Buried under data analyses

- Data collection is cheap => datasets grow exponentially
- Cluster computing makes it easy to analyze large datasets
  - Queries can span entire datasets
- Data is often re-accessed
  - Queries running on the same large datasets
- Leads to significant data reuse



Facebook, and Cloudera customers [VLDB'12]



CMU academic clusters [VLDB'13]

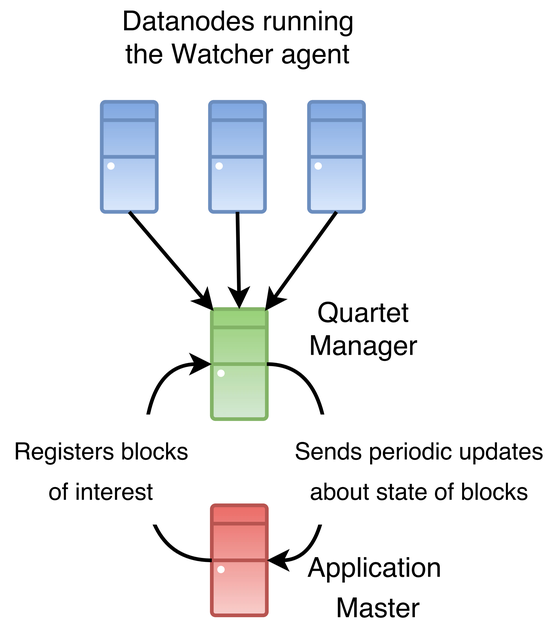
- Data reuse should improve performance due to caching
- **Problem:** We find that jobs see no benefits from reuse
  - Working sets do not fit in the page cache
  - Jobs consume data independently of one another

Goal: Jobs should consume data cached across the cluster first, not evict it

Quartet Architecture

Finding what is in the cache

- **Watchers:** DataNode daemons that listen for changes in node cache
  - Duet kernel module reports on additions, removals etc. in the cache [SOSP15]
- **Quartet Manager:** Central agent that aggregates watcher updates



Cache-aware scheduling

1. Application Master reports blocks of interest to Quartet Manager
2. Quartet Manager informs Application Master on cached blocks
3. Application Master schedules tasks with cached blocks **first**

Experimental results

- Implemented on **Spark** and **Hadoop MapReduce**
- 24 nodes with a total of 384 GB of memory
- Simple IO-bound application
- 2 jobs with 100% input overlap ran sequentially

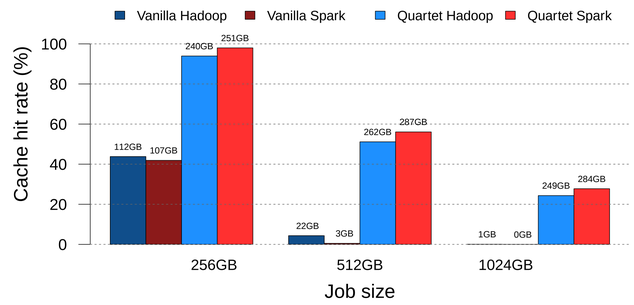


Figure 1: Cache hit ratio

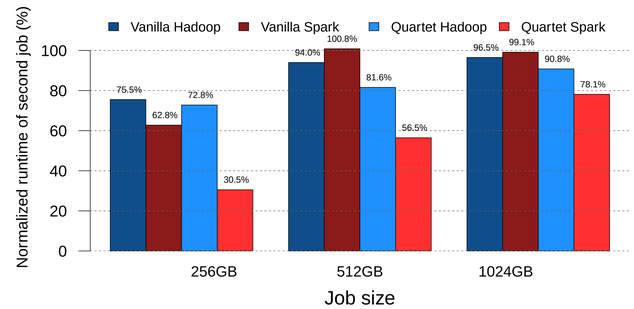


Figure 2: Runtime reduction

Vanilla (baseline) misses sharing opportunities

- Uninformed replica selection
  - 40-45% of cache hit rate with resident dataset
- Processing order independent of cache content
  - Close to no reuse from cache on large datasets

Quartet improves the efficiency of Hadoop and Spark

- Cache hit rate increased to 92-98%
- Up to 45% reduction of runtime