Quartet: Harmonizing task scheduling and caching for cluster computing

UNIVERSITY OF TORONTO

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

June 23, 2016

Analyses for the masses

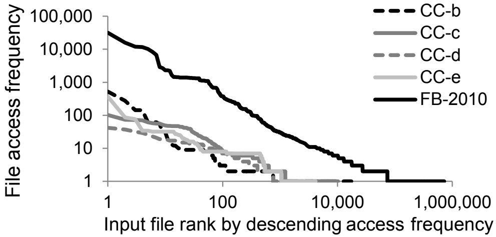
- Data collection is cheap, so datasets are growing exponentially
- Cluster computing makes it easy to analyze these datasets, enabling:
  - Queries on entire datasets
  - Analysts running queries on the same corpus
  - Tuning queries

Data is often re-accessed

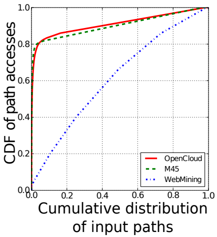
- Result is many queries running on the same large datasets
- Leads to significant data reuse

Data is often re-accessed

- Result is many queries running on the same large datasets
- Leads to significant data reuse



Facebook, and Cloudera customers [VLDB'12]

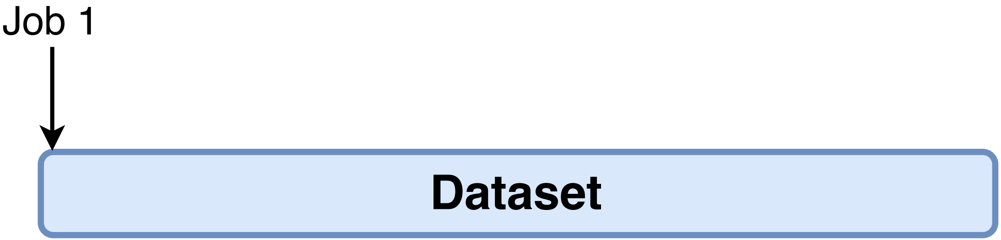


CMU academic clusters [VLDB'13]

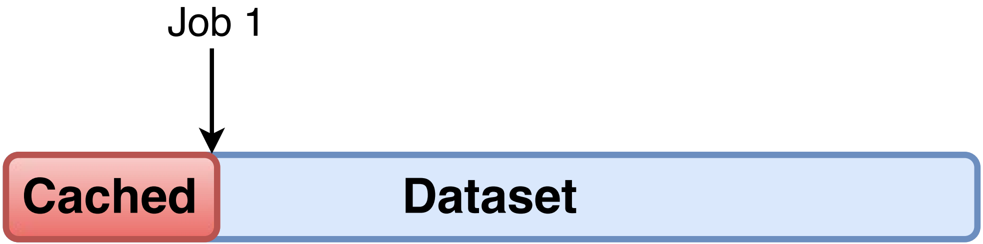
Data reuse does not help

- We should expect data reuse improves performance due to caching
- We find that jobs do not see the benefits of reuse

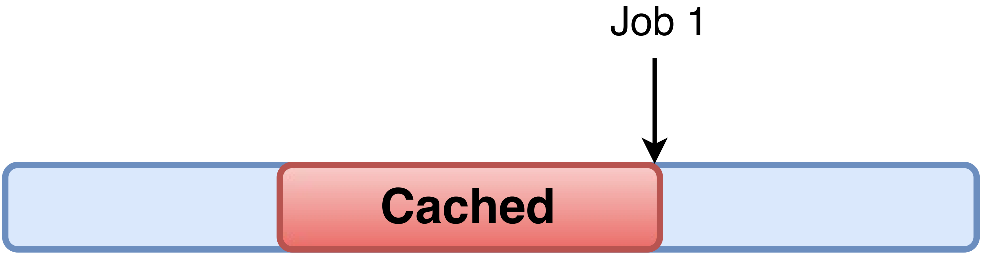
Example: Repeating a Hadoop job



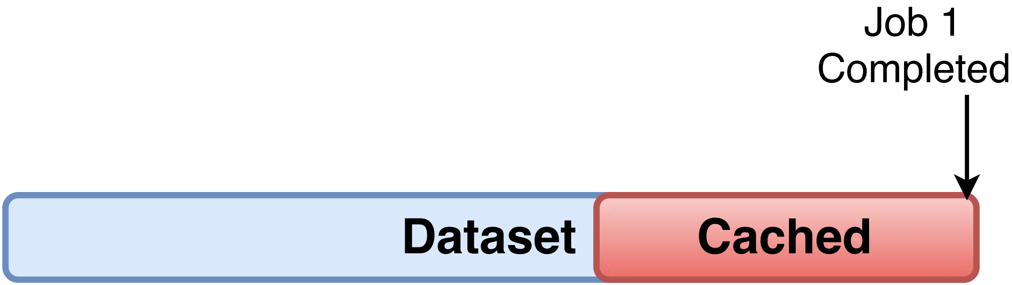
Example: Repeating a Hadoop job



Example: Repeating a Hadoop job

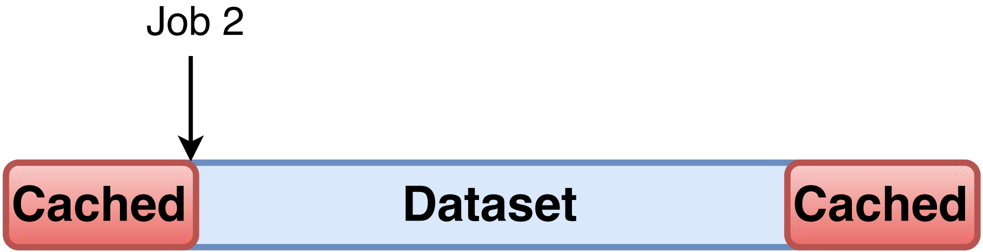


Example: Repeating a Hadoop job

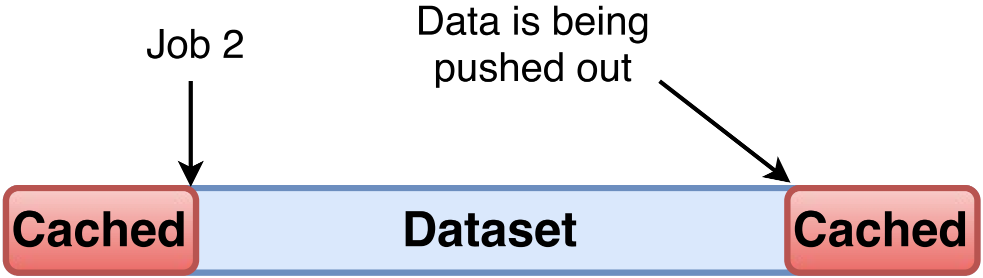


Example: Repeating a Hadoop job

Example: Repeating a Hadoop job



Example: Repeating a Hadoop job



Missed Opportunities

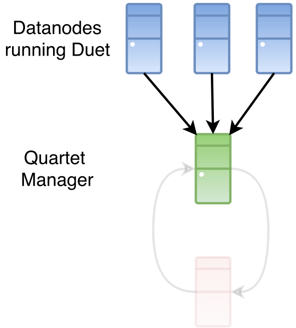
- Working sets don't fit in the page cache
- Jobs consume data independently of one another

Solution

- Key idea
  - Reorder work to first consume cached data
  - Jobs are made of small tasks with no ordering requirements
- Challenges
  - Cache visibility: Jobs need to know what data is cached on the different nodes
  - Task reordering: Jobs need to reorder their tasks based on this knowledge
- Our Quartet system addresses both these challenges

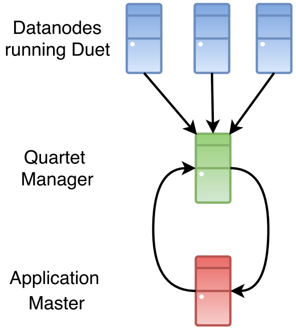
Challenge 1: Cache Visibility

- Datanodes collect information about HDFS blocks that reside in memory
  - Requires the Duet kernel module that informs applications when pages are cached or evicted
- Nodes send this information periodically to the Quartet Manager
  - Changes to the number of resident pages of each block

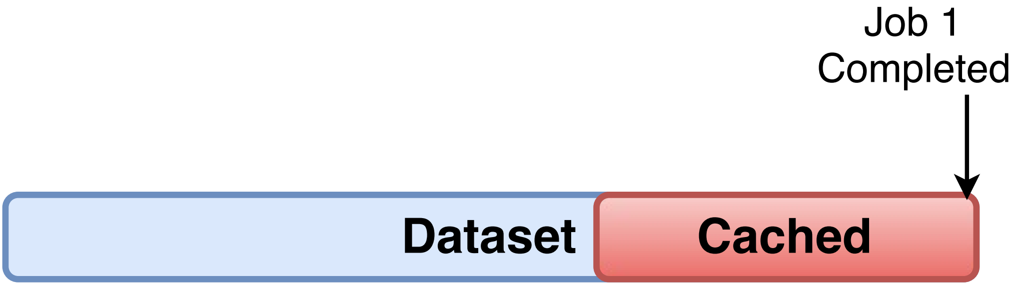


Challenge 2: Task Reordering

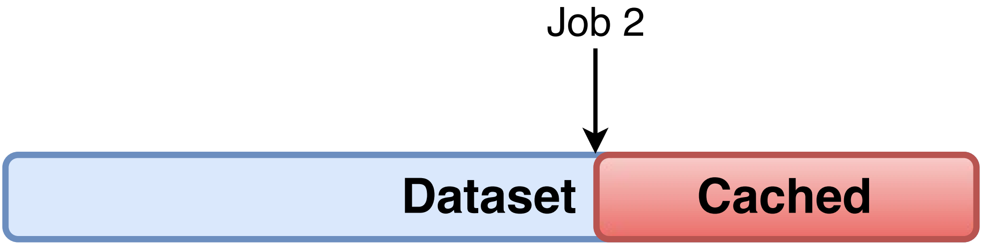
1. Application Master registers blocks of interest with the Quartet manager
2. Quartet manager informs Application Master about cached blocks
3. Application Master prioritizes and places tasks based on block residency information



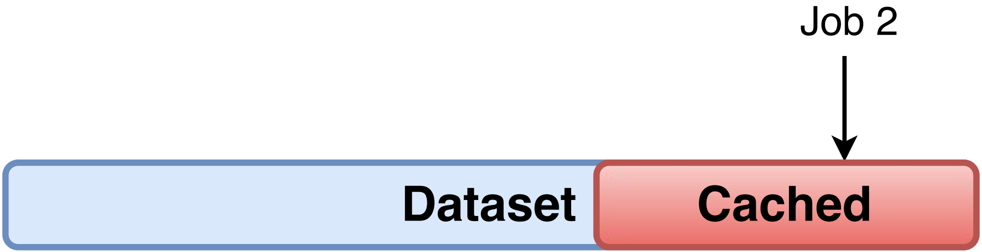
Example: Repeating a Hadoop job



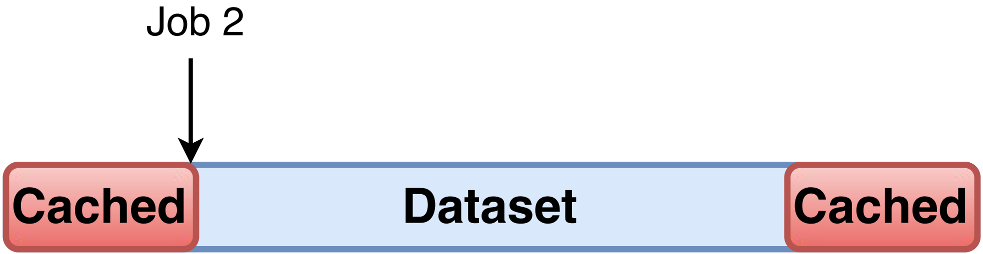
Example: Repeating a Hadoop job



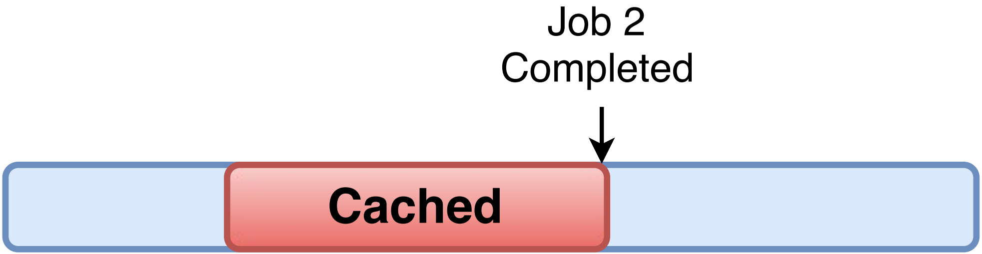
Example: Repeating a Hadoop job



Example: Repeating a Hadoop job



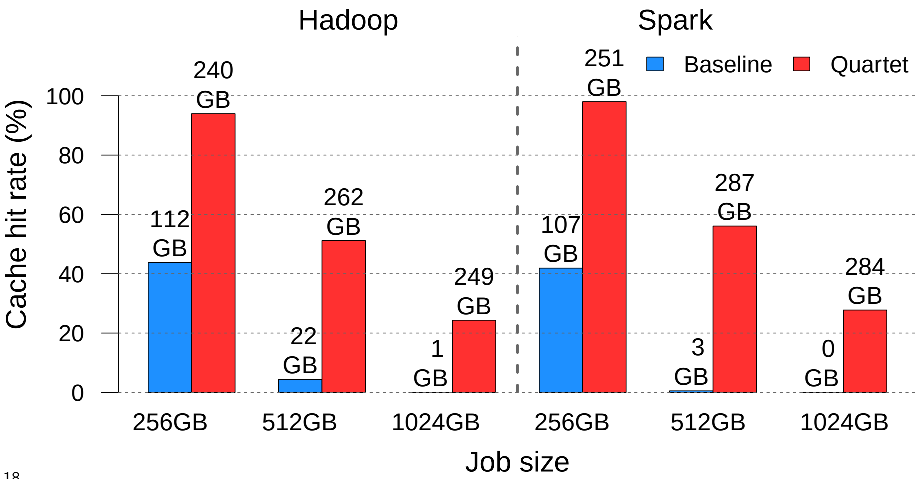
Example: Repeating a Hadoop job



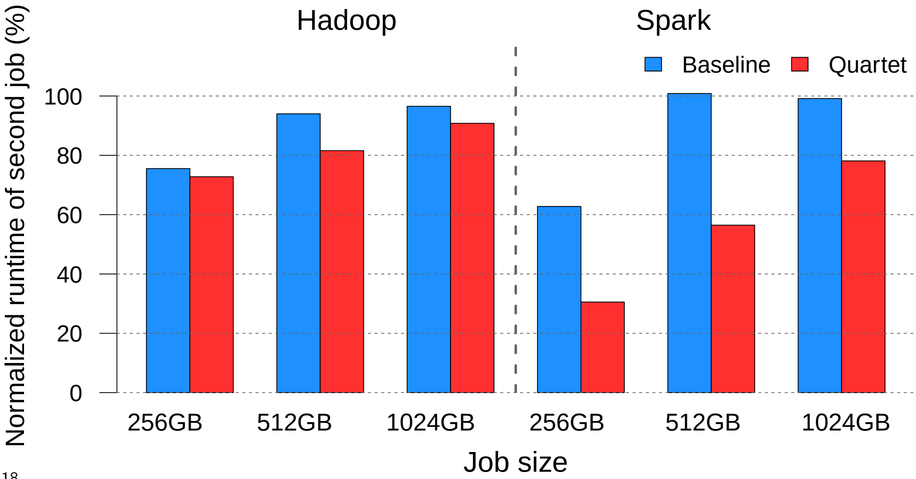
Experiments

- Spark and Hadoop implementations
- 24 nodes with a total of 384 GB of memory
- Different input sizes:
  - Smaller than physical memory (256 GB)
  - Slightly larger (512 GB)
  - Approximately 3 times (1024 GB)
- 3 replicas per block

Results - Cache Hit Rate of the second job



Results - Runtime reduction of the second job



Conclusions

- Observation: Workloads show significant amount of reuse
- Problem: Jobs are unable to take advantage of this reuse
- Solution:
    - Add visibility on what is cached in each of the cluster nodes
    - Reorder tasks to take advantage of this cached data
- Future work: More realistic workloads and scalability

Conclusion

Thank you!

Conclusion

Questions?

Network Overhead

- Watchers updates are aggregated per HDFS blocks (128-256MB)
- Upper bound is the storage bandwidth
- Manager notifications is proportional to the size of the input and hardware
  - 10-100 KB/s in our experiments

Related Work

HDFS Cache Manager

- Requires manual changes in case of change of popularity
- Can't be used when input is larger than memory

PACMan

- Avoiding stragglers in single wave of Mappers
- Modify cache eviction policy to ensure that entire computation stages have memory locality