Caching in the multiverse

Mania Abdi*, Amin Mosayyebzadeh†, Mohammad Hossein Hajkazemi*,
Ata Turk†, Orran Krieger†, Peter Desnoyers*

Northeastern University*, State Street†, Boston University†
Typical Data analytic Cluster

Analytic Frameworks

Execution Framework
e.g. Hadoop, Spark, Tez

Datacenter Network

Cluster Network

TOR

Compute

Storage
Typical Data analytic Cluster

- Storage disaggregation enables:
  - Scalability
  - Flexibility
- Execution frameworks perform better when data is local to cluster
- A distributed cache can help, e.g. Alluxio
Data analytic executions

User queries -> Query optimizer -> Job DAG e.g. PIG

• DAGs have complex and deep structure, e.g. 200 or more nodes for complex jobs. In DAGs:
  • vertices represent analytic jobs
  • Edges represent dependencies between jobs.

• Dependency within a DAG + run time \(\rightarrow\) estimated critical path.

Query #8 from TPC-H benchmark
Taxonomy of caching

Frequent Management Approaches

- History based
  - LRU
    - Most practice today
      - e.g. Alluxio
  - All-or-nothing
    - e.g. Pacman (NSDI12)

- Informed hint based
  - Application hints
    - e.g. TIP (SOSP95), MC2 (TOCS11)
    - Deadline aware
      - e.g. NetCo (SOCC18)
  - DAG aware
    - MRD (ICPP18), LRC (InfoComm 17)
Optimizing Data Analytic caches

Cache performance metric:

\[ \text{Query completion time} = T_{\text{job finish}} - T_{\text{job submission}} \]

Goal: minimize query completion time
Our approach

Adapt with schedule

Adapt with time

Execution History

Storage bandwidth

DAGs of Jobs

Cache Status

The first system that solve the joint problem of DAGs scheduling and caching

Goal: Calculate optimize cache plan
- Prefetch control
- Admission control
- Eviction control

Execution Framework
How to find cache/prefetch plan?

I1 Input
J1 MR-Job
O1 Output

Store

I2
J2
O2

I3-1
J3
O3

I3-2
J5
O5

I4
J4
O4

I5-1
J6
O6

I5-2
J7
O7

Stage 2
Stage 3
Stage 4

Time

I1 I2 I3 I4 I5 I6 I7 I8 I9 I10
How to find cache/prefetch plan?

• Predict job run time with and without prefetching.
• Find critical path based on dependencies and history of execution.
• Incorporate dependencies, bandwidth to storage, current cache status to choose:
  • Dataset to be cached
  • Dataset to be prefetched
  • Dataset to be evicted
Experimental environment

Analytic framework: Pig
Execution framework: MapReduce
• 4 bare metal nodes:
  • 60GB RAM
  • 16 CPU
  • 10 Gbps Ethernet
Distributed cache: Alluxio
• 4 bare metal nodes:
  • Collocate with Hadoop nodes.
  • 68G per cache = 24GB cache
Remote storage: Ceph
Benchmark: TPC-H queries
• 30 GB dataset size
Results

- 2X improvement over LRU.
- Up to 22% improvement over MRD
Current work: implementation

- Workflow DAG
- Job runtime estimator
- History of execution
- Critical path estimator
- Job improvement estimator
- I/O planner
- Distributed Cache
- Remote Storage
Discussion / Challenges

• Benchmark:
  • New benchmark to evaluate our approach.

• Multi-query execution:
  • Approaches: two step cache management
    • Create plan for a single query
    • Create plan for multiple queries.
  • Competition queries with contradicting requests
    • Initial approaches: prioritize nearest future access.
  • Bandwidth allocation for multiple queries
    • Initial approaches: prioritize shortest job first.

• Where to prefetch?
Conclusion

**Goal:** minimize end-to-end latency of query execution

**Approach:** scheduling aware cache management policy

**Key insight:** incorporate execution history and current cache status to optimize the critical path through caching and prefetching.

**Results:** 2X improvement over LRU