MISO: Souping Up Big Data Query Processing with a Multistore System

Jeff LeFevre, UC Santa Cruz*
Jagan Sankaranarayanan, NEC Labs
Hakan Hacigümüş, NEC Labs
Junichi Tatemura, NEC Labs
Neoklis Polyzotis, UC Santa Cruz
Michael J. Carey, UC Irvine

Appeared in SIGMOD 2014 *Now at HP Vertica R&D
Analytical System in today’s Organizations

Big Data / Exploratory Queries

- Data that collect but there may be a use for it
- Unknown business value
- Formatted on-the-fly as needed (schema-on-read)
- Evolutionary workload (i.e., adhoc)

Big Data

HDFS

Business Reporting Queries

- Data important to your organization
- High business value
- Well formatted (schema-on-write)
- Well known workload

Business Data

RDBMS
System Characteristics

- **Big Data Store** offers the ability to store massive amounts of data and begin querying right away
  - Executes SQL + UDFs
  - Slow query performance [Pavlo et al. 2009, Dean et al. 2010]

- **RDBMS** offers the ability to achieve high performance for queries
  - Executes SQL
  - High data loading time [Simitsis et al. 2009]

- **Emerging system architectures** combine both
  - Create a “Multistore” system for workload co-processing
Multistore Architecture to Speedup Big Data Processing

Big Data / Exploratory Queries

Query Plan

HV (HDFS)

Business Reporting Queries

DW (RDBMS)

Big Data

DW has limited spare capacity to speedup big data queries
Multistore System Architecture

- Multistore query plan is “split” across both stores
  - Transfers data and computation during query processing

- Two related work [Simitsis et al. 2012, DeWitt et al. 2013] focus on optimizing multistore plans

- Let us look at the performance for all possible splits for a single plan
Execution Profile for all Possible Splits

Each plan represents a different split of the plan

On-the-fly data loading

Up-front data loading
Our Approach: Multistore Design

- Problem is at the core of multistore architectures
  - Up-front data loading too time consuming
  - On-the-fly data loading results in redundant work
  - Need to decide what subset of data will be useful to load into DW

- Multistore physical design problem is a data placement problem that determines:
  - *What* data to materialize (which “views”)  
  - *Where* to materialize the data (which “store”)

- We introduce MISO, our MultiStore Online tuner
Multistore System with MISO

MISO is our “secret sauce” to tune the multistore

Q2 benefits from the opportunistic views of Q1 [SIGMOD 2014 paper on Wednesday]

Periodically moves the useful views to DW [Today’s talk]
Our MultiStore Online Tuner (MISO)

- Periodically reorganizes the data in each store
  - Transfer views between the stores
  - Adapt the design as the workload changes dynamically (system observes a stream of queries)

- High-level goals:
  1. Facilitate earlier splits for each query
  2. Most part of the queries run on DW, even bypassing HV sometimes
  3. Limit the impact on Data Warehouse (DW)

- Challenges
  - Have to solve 2 physical design problems
  - Problems are tied together in a unique way
Problem Statement

Given an observed query stream, current view placement, and budget constraints, compute new placement of views that minimizes the expected future workload cost.
MISO Tuner Solution

- Storage budget $B_h$
- View Transfer budget $B_t$
- Storage budget $B_d$

HV
- Views $V_h$
- Use leftover transfer budget
- Solve HV design SECOND

DW
- Views $V_d$
- Use as much of the transfer budget
- Solve DW design FIRST

View Transfer budget $B_t$
Selecting Views for Each Store

- MISO solves a multi-dimensional 0-1 knapsack problem for each store
  - Inputs are the set of views and budget constraints
  - Dimensions are the storage and transfer budgets

- Knapsack complication: view benefits are not independent
  - A pair of views (x,y) can interact in two ways [Schnaitter et al. 2009]
    - Positive interaction
    - Negative interaction

[Schnaitter et al. 2012]

Packing Heuristics
Experimental Setup

- 2 Independent Clusters connected via 1 GbE
  - Hadoop (HV) 15 nodes (1 head + 14 slaves)
    - Latest version
  - DW 9 nodes (1 head + 8 slaves)
    - Commercial parallel data warehouse

- All machines have identical hardware
  - 2x Xeon CPUs, 16 GB RAM, 2 TB disk
Experimental Setup (2)

- **Workload** *(See our SIGMOD DANAC 2013 paper)*
  - Mimic data analyst/scientist exploration to find value
  - 32 queries: 8 analysts each writing 4 versions of a query
  - Complex queries with UDFs
    - Average query runtime on Hadoop cluster is about 10k secs

- **Data sets**
  - 1 TB Twitter “tweet” stream (social data)
  - 1 TB Foursquare “check-in” stream (social data)
  - 12 GB Landmarks data (static/historic data)
System Performance Comparison

TTI ➔ Time To Insight. Holistic metric that includes everything, ETL, tuning, transfer, execution time.
Breakdown of Execution Time

(a) MS-BASIC

(b) MS-MISO with small storage budget

(c) MS-MISO with large storage budget
MultiStore Tuning Methods

![Bar chart comparing different MultiStore tuning methods: MS-BASIC, MS-OFF, MS-LRU, and MS-MISO. The Y-axis represents TTI (10^3 Sec) ranging from 0 to 300. The chart shows how each method performs, with MS-MISO having the lowest TTI and MS-BASIC having the highest.](image-url)
Varying Storage Budgets ($B_t = 10\text{GB}$)

<table>
<thead>
<tr>
<th>TTI (x$10^3$ seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.125X</td>
</tr>
<tr>
<td>0.5X</td>
</tr>
<tr>
<td>1X</td>
</tr>
<tr>
<td>2X</td>
</tr>
</tbody>
</table>

- **MS-LRU**
- **MS-OFF**
- **MS-MISO**
Impact on DW

- Execute a background workload on DW
  - TPC-DS reporting queries consuming IO and CPU resources
  - Consider DW with 20% spare capacity and 40% spare capacity for both IO and CPU case

- Avg. impact (slowdown) on DW queries < 2%
Summary

- Tuning the multistore important to obtaining good performance

- Our MISO Tuner algorithm periodically reorganizes views in each store

- This improves upon both up-front data loading and on-the-fly data loading

- Outperforms multistore query processing without tuning (MS-BASIC) and naïve tuning approaches such as LRU