R-Store: A Scalable Distributed System for Supporting Real-time Analytics

Feng Li, M. Tamer Özsu, Gang Chen, Beng Chin Ooi @ICDE 2014

Presented by: Xiao Meng

CS848, University of Waterloo
Outline

• Background & Motivation
• System Overview
• System Design
• RTOLAP in R-Store
• Evaluation
• Conclusion
• Q & A
Background & Motivation

- **Situation for large scale data processing**
  - Systems classified into 2 categories: OLTP, OLAP
  - Data periodically transport to OLAP through ETL

- **Demand**
  - Time critical decision making (RTOLAP)
    - the freshness of OLAP results
    - Fully RTOLAP entail executing query directly on OLTP data
  - OLAP & OLTP processed by one integrated system
Background & Motivation

- **Problem on simple combination**
  - Resource contention
    - OLTP query blocked by OLAP
  - Inconsistency
    - Long running OLAP may access same data sets several times, updates by OLTP could lead to incorrect OLAP results

- **Solution – R-Store**
  - Resource contention
    - Computation resource isolation
  - Inconsistency
    - Multi-versioning storage system
System Overview – A glimpse of R-Store

- OLAP query data based on timestamp of query submission from multi-versioning storage system
  - Modified HBase as storage
  - Mapreduce job for query execution

- Periodically materialize real-time data into data cube
  - Fully HBaseScan every time is time-consuming
    - Entire table is scanned & shuffled during MR
  - Streaming Mapreduce to maintain data cube
System Overview – R-Store Architecture

OLTP submitted to KV Store
OLAP query processed by
MapReduce – Scan on HBase
Refresh data cube through
streaming MapReduce
MetaStore to generate query
timestamp T Q & metadata
System Design – A Glimpse of HBase
System Design – Storage Design based on HBase

• Extend Scan to 2 versions
  – FullScan for querying data cube
  – IncrementalScan for querying real-time data

• Infinite versions of data to maintain query consistency
  – Compaction to remove stale versions
  – Global compaction
    • Immediately following data cube refresh
  – Local compaction
    • Compact old versions not accessed by any scan process
System Design – IncrementalScan in detail

• **Target:** Find out changes since last data cube materialization

• **Method**
  – Take 2 timestamps as input $T_{DC}$ & $T_Q$, return the values with largest timestamp before $T_{DC}$ & $T_Q$

• **Implementations**
  – Naïve: Accessing memstore & storefile in parallel
  – Adaptive: Maintain key modified since last materialization, first scan memstore, scan or random access keys based on cost
System Design – Compaction in detail

- **Global compaction**
  - Similar to Hbase’s default, retain only one version of each key
  - Triggered by data cube’s refresh completion

- **Local compaction**
  - Compacted data stored in different file in case block scan process
  - Files can be removed when not accessed by any scan
  - Triggered when #tuple/#key exceeds threshold
System Design – Data cube

- Define a data cube for “Customer Profiles”
- Dimensions: age, income, buys
System Design – Data cube maintenance

• Re-computation
  – First run
  – FullScan on one region, generate a KV pair for each cuboid in mapper, aggregate & output in reducer

• Incremental Update
  – Consequent runs
  – Propagation step to computes change & update step to update cube
  – Streaming system updates cube inside & periodically materialize it into storage
System Design – HStreaming for cube maintenance

• Each mapper responsible for processing update within a key range
  – Maintain KVs locally, cache hot keys in memory
  – For updates, emit 2 KV pair for each cubiod (+, -)

• Reducer cache the output KV of mapper and invoke reduce every $W_r$, refresh cube every $W_{cube}$
System Design – Data Flow of R-Store

1. Updates arrives Hbase-R
2. stream updates to a Hstreaming mapper
3. Reducer periodically materialize local data cube to Hbase-R & notifies Metastore
RTOLAP in R-Store – Query Processing

- Map
  - Tag the values with ‘Q’ ‘+’, ‘-’
- Reduce
  - Do calculation based on aggregation function & three values
Evaluation

- Cluster of 144 nodes
  - Intel X3430 2.4 GHz processor
  - 8 GB of memory
  - 2x500 GB SATA disks
  - gigabit Ethernet

- TPC-H data
Evaluation - Performance of Maintaining Data cube

- Hstreaming with 10 nodes have higher throughput than 40 Hbase-R nodes

- 1.6 billion keys, 1% updated, update algorithm fast enough,
- latency equals to Hbase-R input speed
Evaluation - Performance of RT querying

- Small key range updates scans few data in Hbase-R, process fewer data
Evaluation - Performance of OLTP

(a) Throughput

(b) Latency
Related Work

• Database
  – C-Store(VLDB 05)

• Main-memory database
  – HyPer(ICDE 11), HYRISE(VLDB 10)

• Druid(SIGMOD 14)
Conclusion

- Multi-version concurrent control to support RTOLAP
- Data cube to reduce storage requirement & improve performance
- Streaming system to refresh data cube
- Available at https://github.com/lifeng5042/RStore
Q&A
Backup – OLAP Cube

• A multi-dimensional generalization of a two- or three-dimensional spreadsheet. Hypercube for dataset with more than three d’s.

• Dimensions: Product, time, cities…

• Cells: each cell of the cube holds a number that represents some measure of the business, e.g. sales, profits…

• Slicer: the dimension held constant for all cells so that multi-dimensional information can be shown in a 2D physical space of a spreadsheet.
Backup – OLAP Cube

- Data cube can be viewed as a lattice of cuboids
- The bottom-most cuboid is the base cuboid
- The top-most cuboid (apex) contains only one cell
- How many cuboids in an n-dimensional cube with $L$ levels?

$$T = \prod_{i=1}^{n} (L_i + 1)$$