Hortonworks

Discardable, In-Memory Materialized Queries

Julian Hyde

2014-05-23

Hadoop today

Brute force

Hadoop brings a lot of CPU, disk, IO

Yarn, Tez, Vectorization are making Hadoop faster

How to use that brute force is left to the application

Business Intelligence

Best practice is to pull data out of Hadoop

- Populate EDW e.g. Teradata
- In-memory analytics, e.g. Platfora
- Custom analytics, e.g. Lambda architecture

Ineffective use of memory

Opportunity to make Hadoop smarter

Materialized views - Classic

Classic materialized view (Oracle, DB2, Teradata, MSSql)

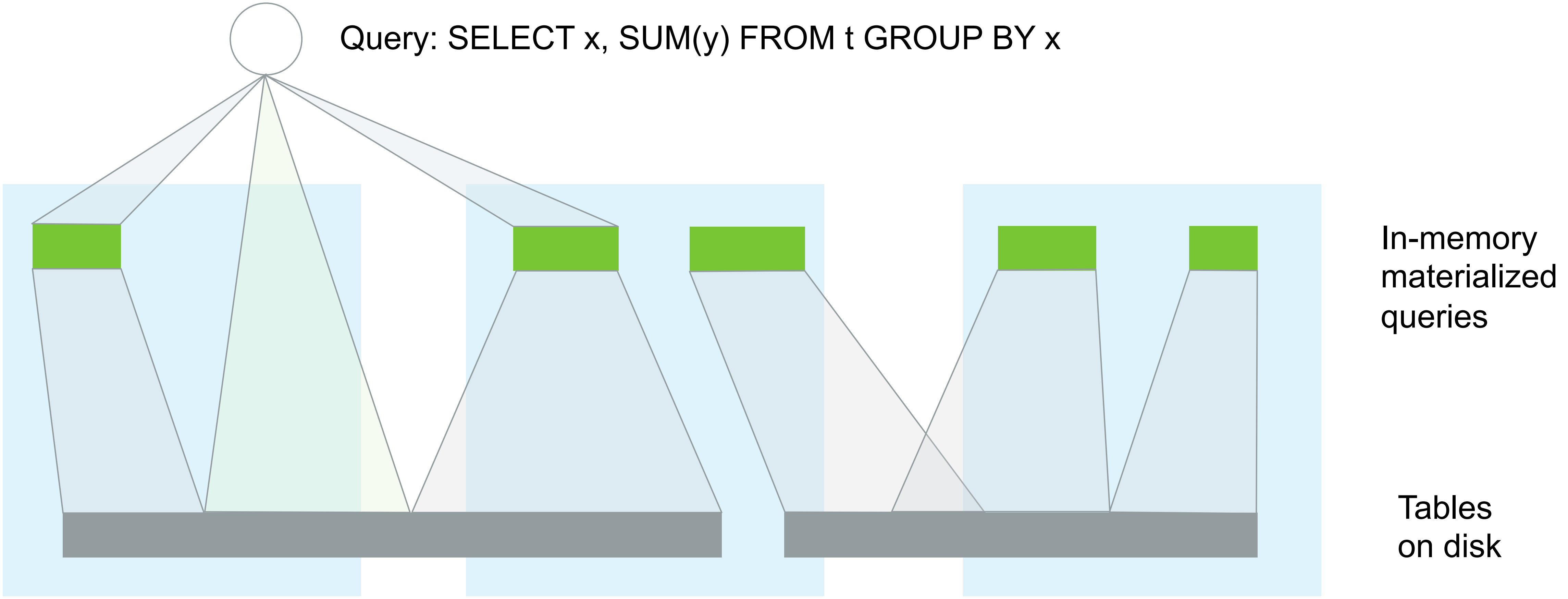
1. A table defined using a SQL query
2. Designed by DBA
3. Storage same as a regular table
    1. On disk
    2. Can define indexes
4. DB populates the table
5. Queries are rewritten to use the table**
6. DB updates the table to reflect changes to source data (usually deferred)*

*Magic required

```
CREATE MATERIALIZED VIEW SalesMonthZip AS
SELECT t.year, t.month,
  c.state, c.zipcode,
  COUNT(*), SUM(s.units), SUM(s.price)
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
JOIN CustomerDim AS c USING (customerId)
GROUP BY t.year, t.month,
  c.state, c.zipcode;
```

```
SELECT t.year, AVG(s.units)
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
GROUP BY t.year;
```

Tables and in-memory materialized queries



Query: SELECT x, SUM(y) FROM t GROUP BY x
In-memory materialized queries
Tables on disk

Hortonworks

Materialized views - DIMMQ

DIMMQs - Discardable, In-memory Materialized Queries

Differences with classic materialized views

1. May be in-memory
2. HDFS may discard – based on DDM (Distributed Discardable Memory)
3. Lifecycle support:
   1. Assume table is populated
   2. Don’t populate & maintain
   3. User can flag as valid, invalid, or change definition (e.g. date range)
   4. HDFS may discard
4. More design options:
   1. DBA specifies
   2. Retain query results (or partial results)
   3. An agent builds MVs based on query traffic

Hortonworks

Data independence

This is not just about SQL standards compliance!

Materialized views are supposed to be transparent in creation, maintenance and use.

If not one DBA ever types "CREATE MATERIALIZED VIEW", we have succeeded

Data independence

Ability to move data around and not tell your application

Replicas

Redundant copies

Moving between disk and memory

Sort order, projections (à la Vertica), aggregates (à la Microstrategy)

Indexes, and other weird data structures

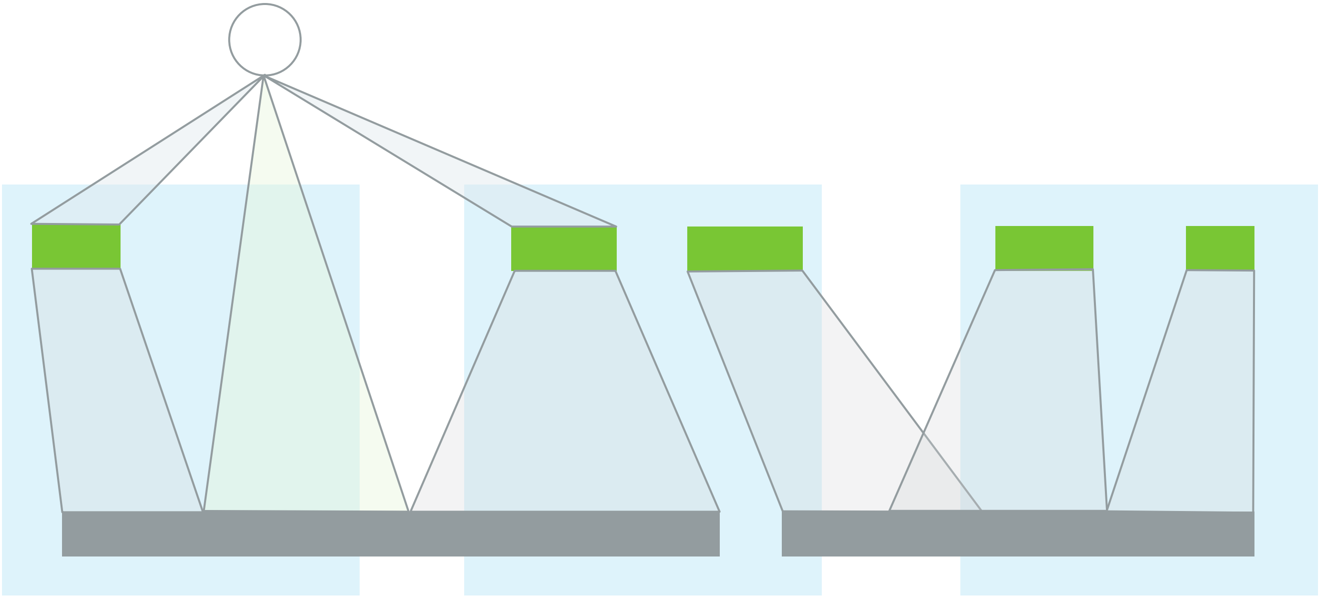
Dynamic equilibrium

Ongoing activities:

- Agent suggests new MVs
- MVs are built in background
- Ongoing query activity uses MVs
- User marks MVs as invalid due to source data changes
- HDFS throws out MVs that are not pulling their weight

(*Rinse and repeat*)

System moves data around to adapt to changing usage patterns.



Hortonworks

Lattice

Space of possible materialized views

A star schema, with mandatory many-to-one relationships

Each view is a projected, filtered aggregation

- Sales by zipcode and quarter in 2013
- Sales by state in Q1, 2012

```
CREATE LATTICE SalesStar AS
SELECT *
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
JOIN CustomerDim AS c USING (customerId);
```

Lattice gathers stats

- "I used MV *m* to answer query *q* and avoided fetching *r* rows"
- Cost of MV = construction effort + memory * time
- Utility of MV = query processing effort saved

Recommends & builds optimal MVs

Hortonworks