Kylin – Hadoop OLAP Engine

Jiang Xu, Architect of Kylin

October 2014

ebay inc
GLOBAL PLATFORM
& INFRASTRUCTURE

Kylin Overview

Kylin

- n. (in Chinese art) a mythical animal of composite form

http://kylin.io

Kylin is an open source Distributed Analytics Engine from eBay Inc. that provides SQL interface and multi-dimensional analysis (OLAP) on Hadoop supporting extremely large datasets

What Is Kylin?

- **Extremely Fast OLAP Engine at Scale**

Kylin is designed to reduce query latency on Hadoop for 10+ billions of rows of data

- **ANSI-SQL Interface on Hadoop**

Kylin offers ANSI-SQL on Hadoop and supports most ANSI-SQL query functions

- **Interactive Query Capability**

Users can interact with Hadoop data via Kylin at sub-second latency, better than Hive queries for the same dataset

- **MOLAP Cube**

User can define a data model and pre-build in Kylin with more than 10+ billions of raw data records

- **Seamless Integration with BI Tools**

Kylin currently offers integration capability with BI Tools like Tableau. Integration with Microstrategy and Excel is coming soon

What Is Kylin? - Other Highlights

- Job Management and Monitoring
- Compression and Encoding Support
- Incremental Refresh of Cubes
- Leverage HBase Coprocessor for query latency
- Approximate Query Capability for distinct Count (HyperLogLog)
- Easy Web interface to manage, build, monitor and query cubes
- Security capability to set ACL at Cube/Project Level
- Support LDAP Integration

Glance of SQL-on-Hadoop Ecosystem …

- **SQL translated to MapReduce jobs**
  - Hive
  - Stinger without Tez
- **SQL processed by a MPP Engine**
  - Impala
  - Drill
  - Presto
  - Spark + Shark
- **SQL process by a existing SQL Engine + HDFS**
  - EMC Greenplum (postgres)
  - Taobao Garude (mysql)
- **OLAP on Hadoop in other Companies**
  - Adobe: HBase Cube
  - LinkedIn: Avatara
  - Salesforce.com: Phoenix

Why Do We Build Kylin?

- **Why existing SQL-on-Hadoop solutions fall short?**

The existing SQL-on-Hadoop needs to scan partial or whole data set to answer a user query. Moreover, table join may trigger the data transfer across host. Due to large scan range and network traffic latency, many queries are very slow (minute+ latency).

- **What is MOLAP/ROLAP?**
  - MOLAP (Multi-dimensional OLAP) is to pre-compute data along different dimensions of interest and store resultant values in the cube. MOLAP is much faster but is inflexible. Kylin is more like MOLAP.
  - ROLAP (Relational-OLAP) is to use star or snow-flake schema to do runtime aggregation. ROLAP is flexible but much slower. All existing SQL-on-Hadoop is kind of ROLAP.

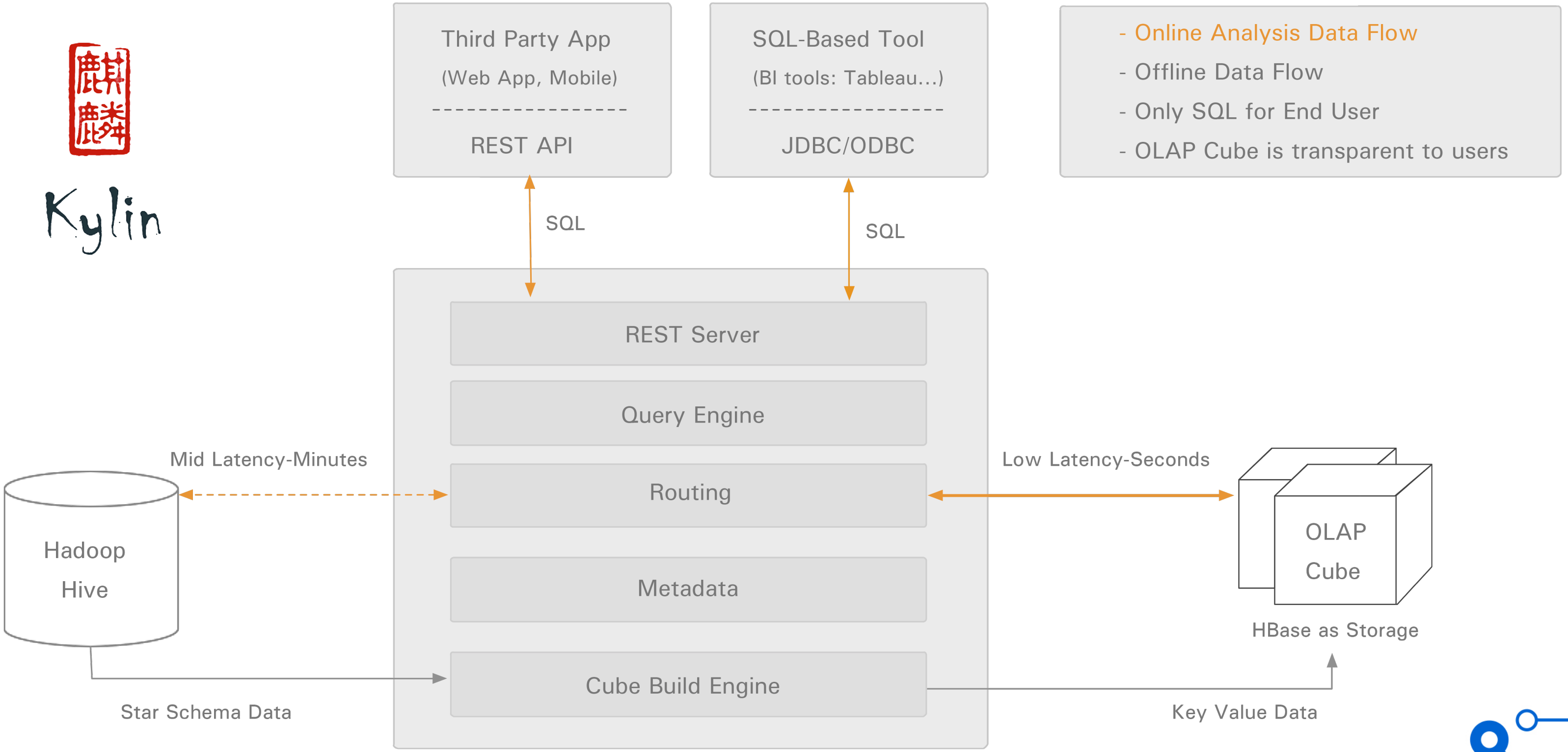
- **How does Kylin support ROLAP/MOLAP?**

Kylin builds data cube (MOLAP) from hive table (ROLAP) according to the metadata definition.

  - If the query can be fulfilled by data cube, Kylin will route the query to data cube that is MOLAP.
  - If the query can't be fulfilled by data cube, Kylin will route the query to hive table that is ROLAP.
  - Basically, you can think Kylin as HOLAP on top of MOLAP and ROLAP.

Architecture Overview

Kylin



Analytics Query Taxonomy

80+% Analytics

High Level Aggregation
- Very High Level, e.g GMV by site by vertical by weeks

Analysis Query
- Middle level, e.g GMV by site by vertical, by category (level x) past 12 weeks

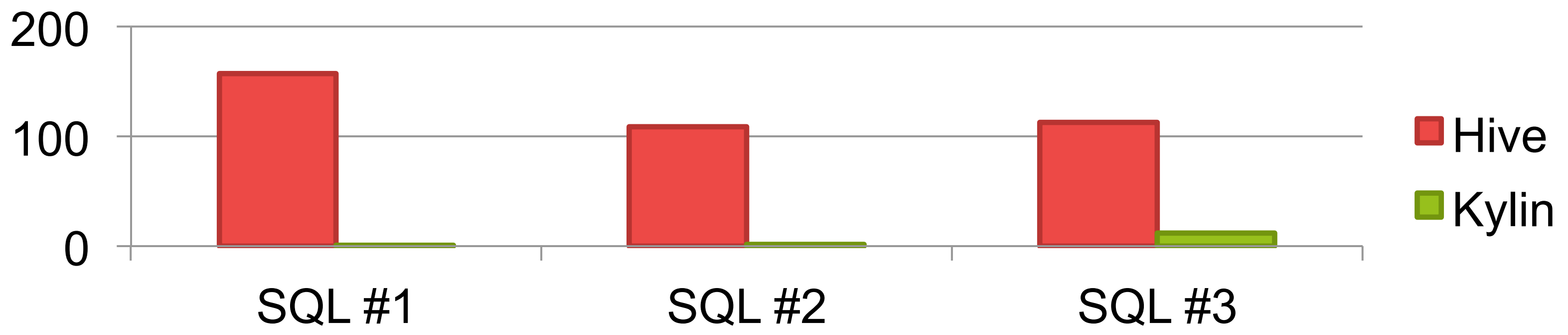
Drill Down to Detail
- Detail Level (SSA Table)

Low Level Aggregation
- Seller ID

Transaction Level
- Transaction ID

Kylin is designed to accelerate ***Analytics*** *queries!*

Query Performance -- Compare to Hive

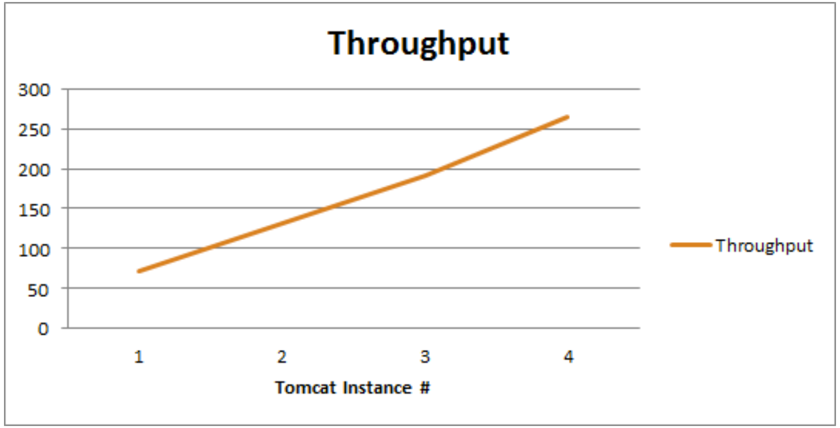


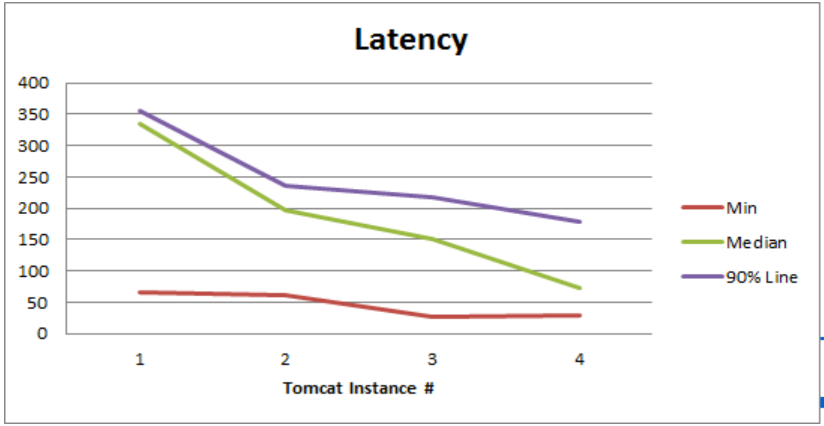
#	Query Type	Return Dataset	Query On Kylin (s)	Query On Hive (s)	Comments
1	High Level Aggregation	4	0.129	157.437	1,217 times
2	Analysis Query	22,669	1.615	109.206	68 times
3	Drill Down to Detail	325,029	12.058	113.123	9 times
4	Drill Down to Detail	524,780	22.42	6383.21	278 times
5	Data Dump	972,002	49.054	N/A	

Query Performance – Latency & Throughput

Single Tomcat Instance on a Single Machine

	Parallel Thread #	Data			Latency (ms)				Throughput
		Raw Recors	HBase Scan	Return	Min	Max	Median	90% Line	
High Level Aggregation Query	30	1,940,304,293	5	5	67	1809	334	355	72.5/sec
Detail Level Query (with Seller ID)	30	13,683,834,542	43934	7283	1758	4534	2182	3171	9.7/sec

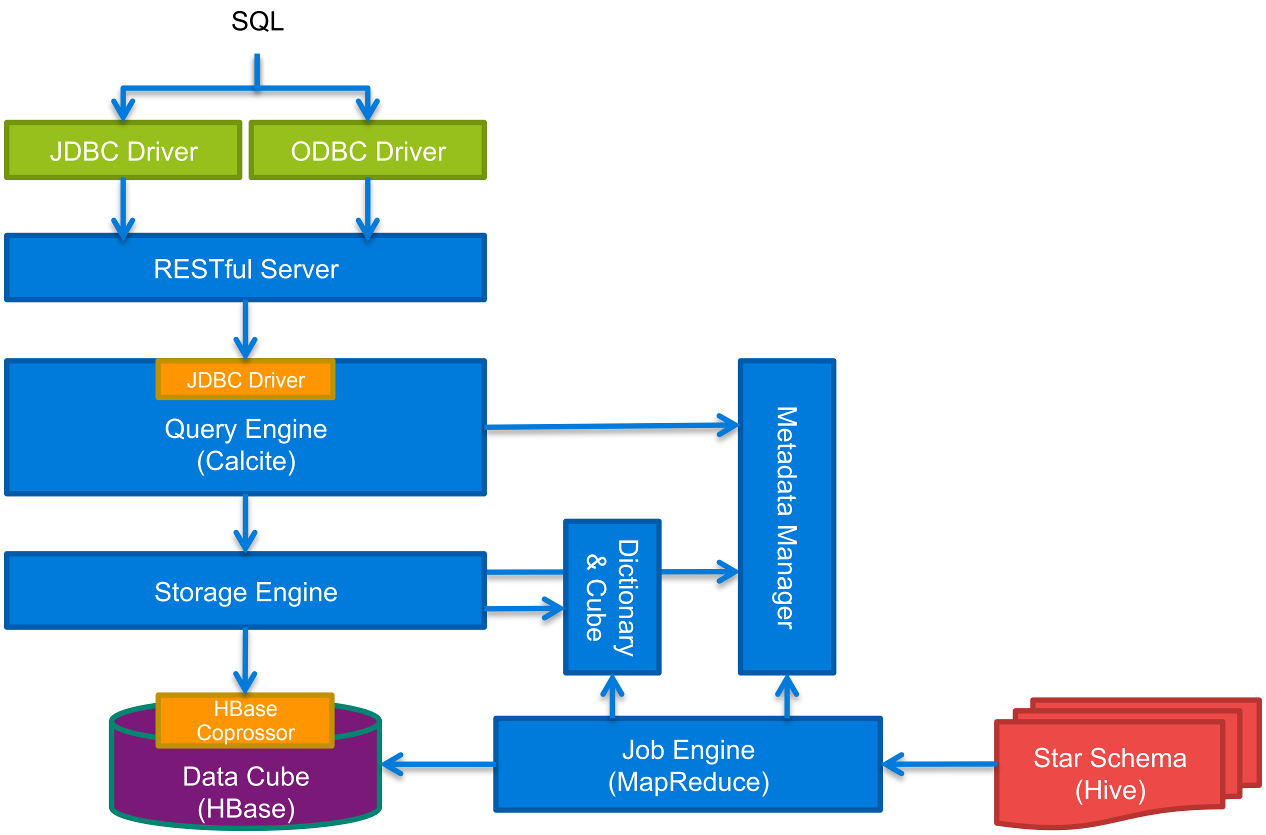




Tech Highlights

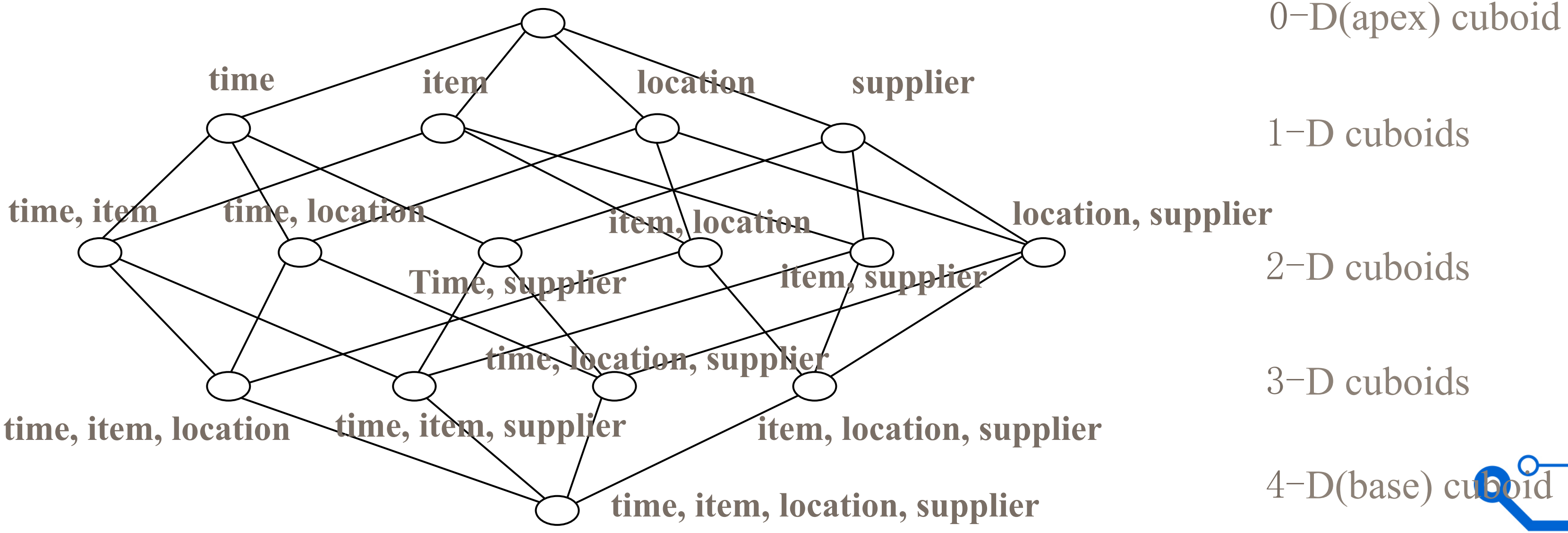
ebay inc
GLOBAL PLATFORM
& INFRASTRUCTURE

Component Design

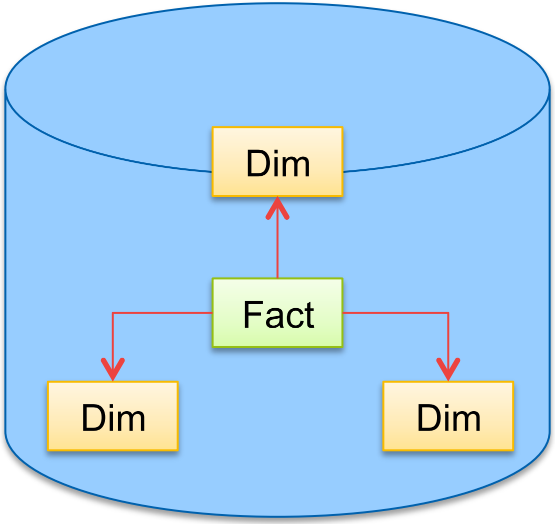


Background – Cube & Cuboid

- Cuboid = one combination of dimensions
- Cube = all combination of dimensions (all cuboids)



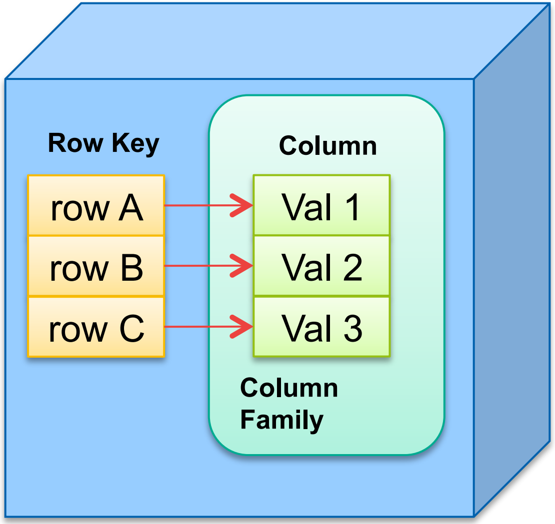
Metadata- Overview



Source
Star Schema

Cube Metadata

Cube: …
Fact Table: …
Dimensions: …
Measures: …
Row Key: …
HBase Mapping:
…



Target
HBase Storage

Metadata – Dimension

- Dimension
  - Normal
  - Mandatory
  - Hierarchy
  - Derived

Metadata – Measure

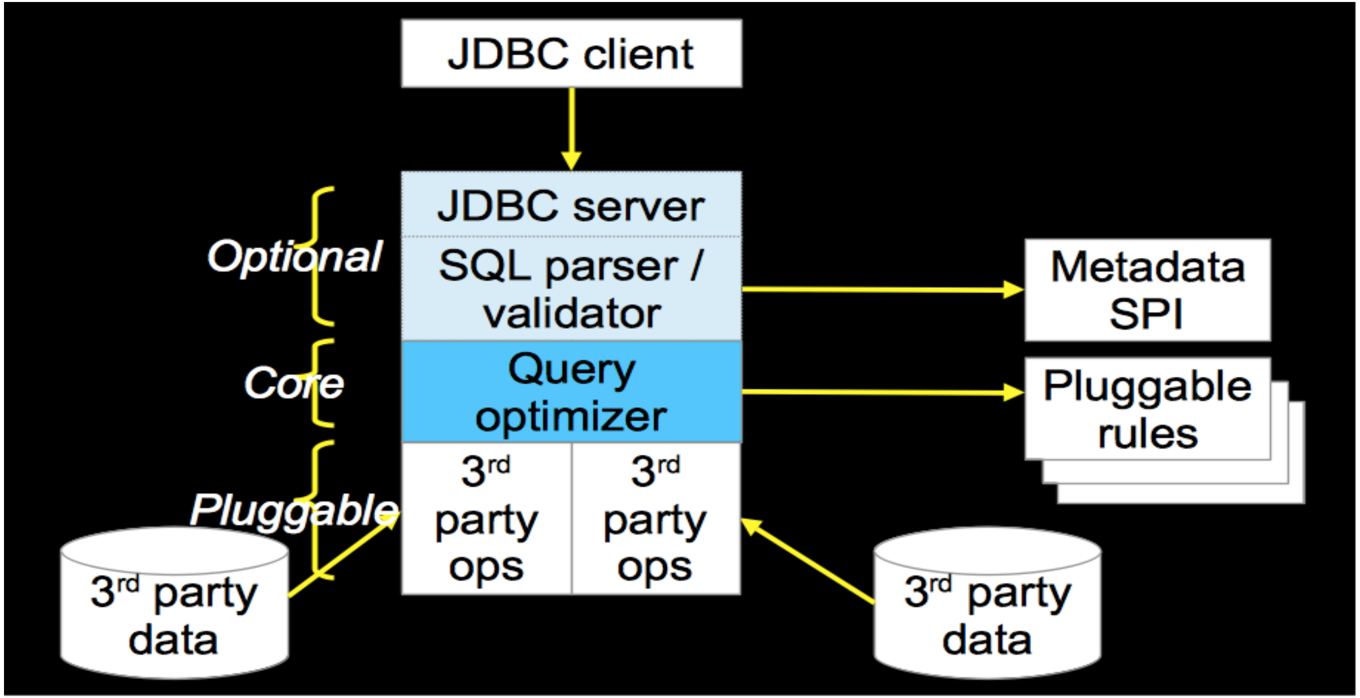
- Measure
  - Sum
  - Count
  - Max
  - Min
  - Average
  - Distinct Count (based on HyperLogLog)

Query Engine - Overview

- Query engine is based on Calcite
- Query Execution Plan
- Optiq Plug-ins in Query Engine

Query Engine – Calcite

- **Calcite** (http://incubator.apache.org/projects/calcite.html) is an *extensible* open source SQL engine that is also used in Stinger/Drill/Cascading.



Query Engine – Explain Plan

SELECT test_cal_dt.week_beg_dt, test_category.lv1_categ, test_category.lv2_categ, test_category.lv3_categ, test_kylin_fact.format_name, test_sites.site_name, sum(test_kylin_fact.price) as total_price, count(*) as total_count
FROM test_kylin_fact
LEFT JOIN test_cal_dt ON test_kylin_fact.cal_dt = test_cal_dt.cal_dt
LEFT JOIN test_category_groupings ON test_kylin_fact.leaf_categ_id = test_category_groupings.leaf_categ_id AND test_kylin_fact.lstg_site_id = test_category_groupings.site_id
LEFT JOIN test_sites ON test_kylin_fact.lstg_site_id = test_sites.site_id
WHERE test_kylin_fact.seller_id = 123456 OR test_kylin_fact.lstg_format_name = ’New'
GROUP BY test_cal_dt.week_beg_dt, test_category.lv1_categ, test_category.lv2_categ, test_category.lv3_categ, test_kylin_fact.format_name, test_sites.site_name

OLAPToEnumerableConverter
OLAPProjectRel(WEEK_BEG_DT=[$0], LV1_CATEG=[$1], LVL2_CATEG=[$2], LVL3_CATEG=[$3], FORMAT_NAME=[$4], SITE_NAME=[$5], TOTAL_PRICE=[CASE(=($7, 0), null, $6)], TOTAL_COUNT=[$8])
OLAPAggregateRel(group=[{0, 1, 2, 3, 4, 5}], agg#0=[$SUM0($6)], agg#1=[COUNT($6)], TRANS_CNT=[COUNT()])
OLAPProjectRel(WEEK_BEG_DT=[$13], LV1_CATEG=[$21], LVL2_CATEG=[$15], LVL3_CATEG=[$14], FORMAT_NAME=[$5], SITE_NAME=[$23], PRICE=[$0])
OLAPFilterRel(condition=[OR(=($3, 123456), =($5, ‘New'))])
OLAPJoinRel(condition=[=($2, $25)], joinType=[left])
OLAPJoinRel(condition=[AND(=($6, $22), =($2, $17))], joinType=[left])
OLAPJoinRel(condition=[=($4, $12)], joinType=[left])
OLAPTableScan(table=[[DEFAULT, TEST_KYLIN_FACT]], fields=[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]])
OLAPTableScan(table=[[DEFAULT, TEST_CAL_DT]], fields=[[0, 1]])
OLAPTableScan(table=[[DEFAULT, TEST_CATEGORY_GROUPINGS]], fields=[[0, 1, 2, 3, 4, 5, 6, 7, 8]])
OLAPTableScan(table=[[DEFAULT, TEST_SITES]], fields=[[0, 1, 2]])

Query Engine – Calcite Plugin-ins

- **Metadata SPI**
  - Provide table schema from kylin metadata
- **Optimize Rule**
  - Translate the logic operator into kylin operator
- **Relational Operator**
  - Find right cube
  - Translate SQL into storage engine api call
  - Generate physical execute plan by linq4j java implementation
- **Result Enumerator**
  - Translate storage engine result into java implementation result.
- **SQL Function**
  - Add HyperLogLog for distinct count
  - Implement date time related functions (i.e. Quarter)

Storage Engine - Overview

- **Provide cube query for query engine**
  - Common iterator interface for storage engine
  - Isolate query engine from underline storage

- **HBase Storage**
  - Pre-join & pre-aggregation
  - Dictionary
  - HBase coprocessor

Storage Engine – Pre-join & pre-aggregation

- **Pre-join (Hive)**
  - Kylin will generate pre-join HQL based on metadata that will join the fact table with dimension tables
  - Kylin will use Hive to execute the pre-join HQL that will generate the pre-joined flat table
- **Pre-aggregation (MapReduce)**
  - Kylin will generate a minimum spanning tree of cuboid from cube lattice graph.
  - Kylin will generate MapReduce job base on the minimum spanning tree
  - MapReduce job will do pre-aggregation to generate all cuboids from N dimensions to 1 dimension.
- **MOLAP Cube (HBase)**
  - The pre-join & pre-aggregation results (i.e. MOLAP Cube) is stored in HBase

Storage Engine – Dictionary

- Dictionary maps dimension values into IDs that will reduce the memory and storage footprint.
- Metadata can define whether one dimension use “dictionary” or not
- Dictionary is based on Trie

```
          '' (6)
          |
          pa (6)
        /    \
   int (1*)    r (5*)
               |
               t (4*)
             /    \
          i (2)    y (1*)
        /    \
   es (1*)  tion (1*)
```

```
[value]        [seq no]
par        ->  1
part       ->  2
party      ->  5
parties    ->  3
partition  ->  4
paint      ->  0
```

Storage Engine – HBase Coprocessor

- HBase coprocessor can reduce network traffic and parallelize scan logic

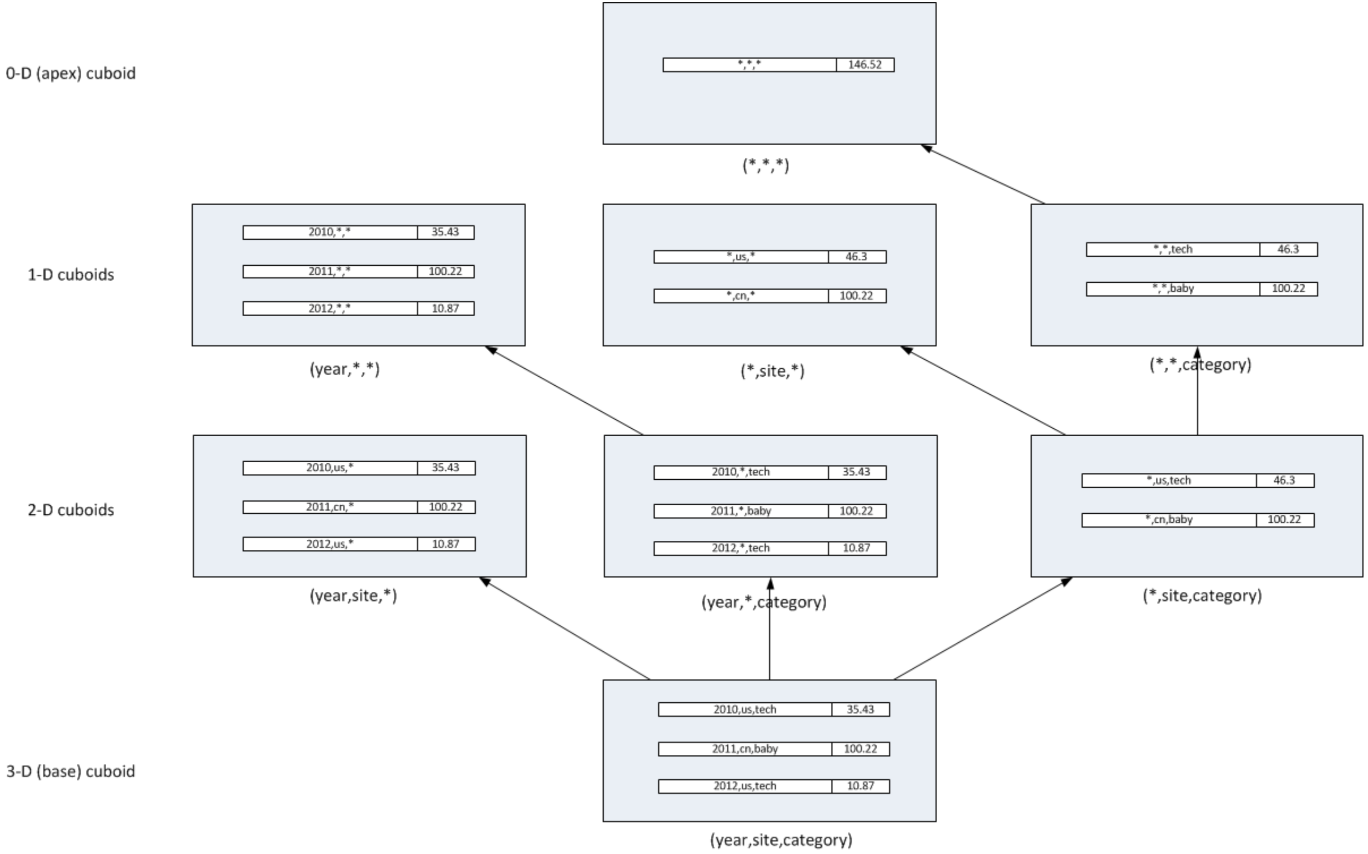
- HBase client
  - Serialize the filter + dimension + metrics into bytes
  - Send the encoded bytes to corprocessor

- HBase Corprocessor
  - Deserialize the bytes to filter + dimension + metrics
  - Iterate all rows from each scan range
    - Filter unmatched rows
    - Aggregate the matched rows by dimensions in an cache
    - Send back the aggregated rows from cache

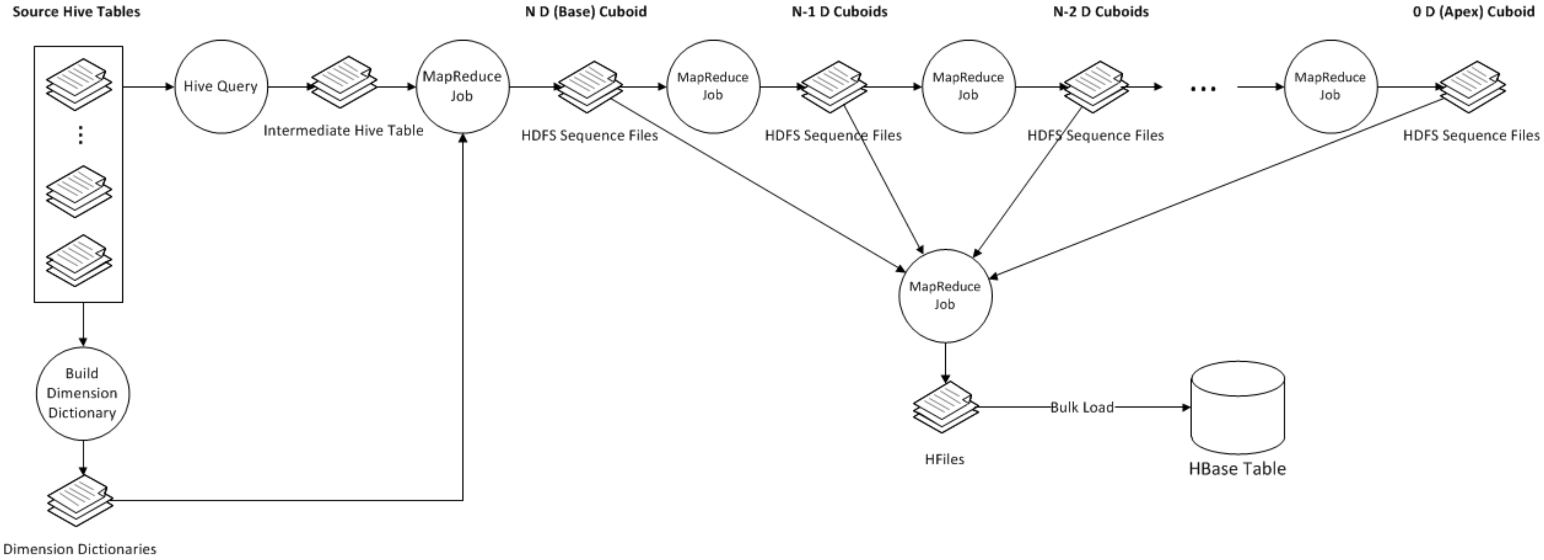
Cube Build - Overview

- Multi Staged Build
- Map Reduce Job Flow
- Cube Build Steps

Cube Build – Multi Staged Build



Cube Build – Map Reduce Job Flow



Cube Build – Steps

- Build dictionary from dimension tables on local disk. And copy dictionary to HDFS
- Run Hive query to build a joined flatten table
- Run map reduce job to build cuboids in HDFS sequence files from tier 1 to tier N
- Calculate the key distribution of HDFS sequence files. And evenly split the key space into K regions.
-
- Translate HDFS sequence files into HBase Hfile
- Bulk load the HFile into HBase

Cube Optimization - Overview

- "Curse of dimensionality": N dimension cube has 2N cuboid
  - Full Cube vs. Partial Cube
- Hugh data volume
  - Incremental Build
- Slow Table Scan – TopN Query on High Cardinality Dimension
  - Bitmap inverted index
  - Time range partition
  - In-memory parallel scan: block cache + endpoint coprocessor

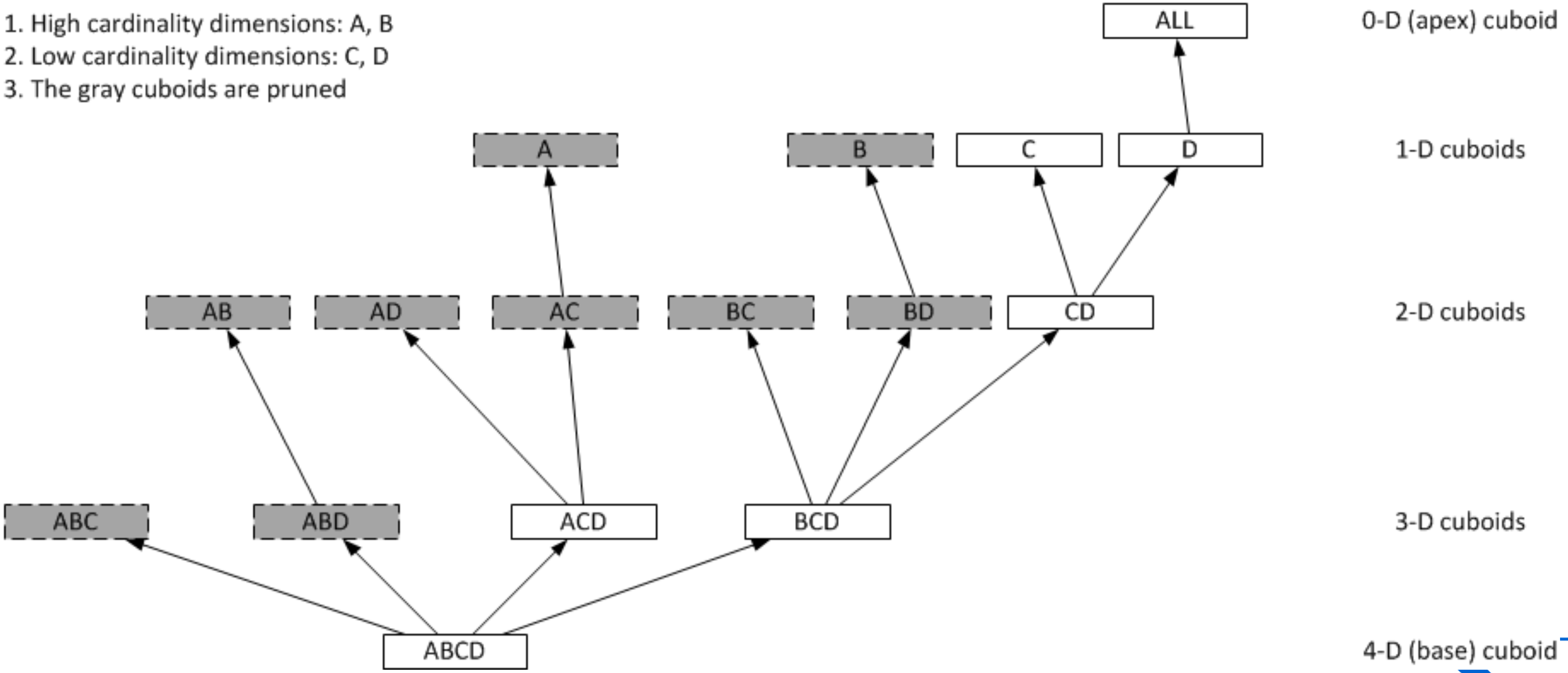
Cube Optimization – Full Cube vs. Partial Cube

- **Full Cube**
  - Pre-aggregate all dimension combinations
  - “Curse of dimensionality”: N dimension cube has 2N cuboid.

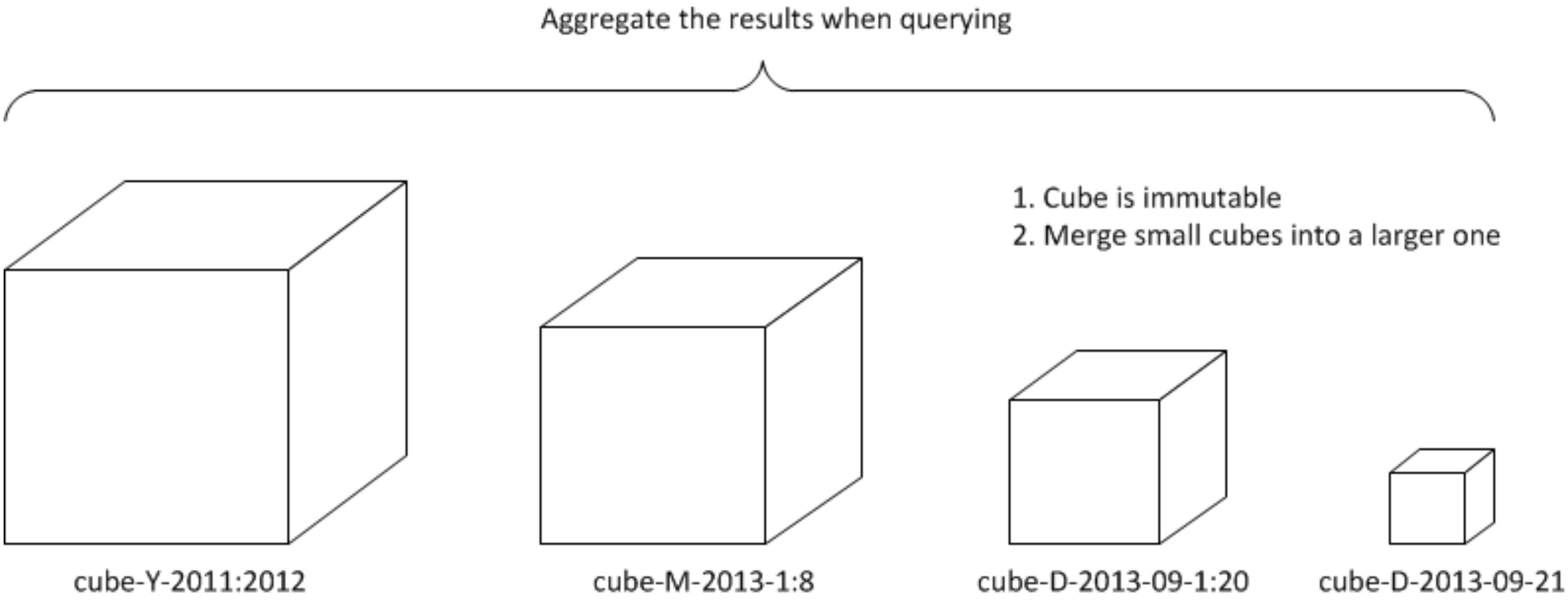
- **Partial Cube**
  - To avoid dimension explosion, we divide the dimensions into different aggregation groups
  - For cube with 30 dimensions, if we divide these dimensions into 3 group, the cuboid number will reduce from 1 Billion to 3 Thousands
  - Tradeoff between online aggregation and offline pre-aggregation

Cube Optimization – Partial Cube

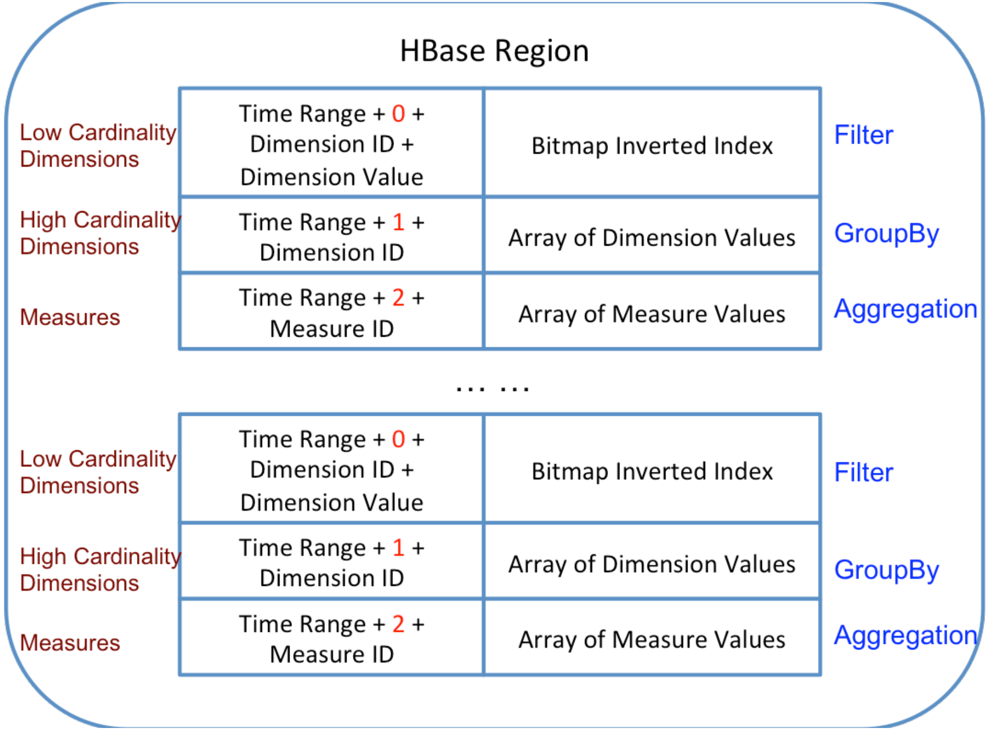
1. High cardinality dimensions: A, B
2. Low cardinality dimensions: C, D
3. The gray cuboids are pruned



Cube Optimization – Incremental Build



Cube Optimization – TopN Query on High Cardinality Dimension



- Bitmap inverted index
- Separate high cardinality dimension from low cardinality dimension
- Time range partition
- In-memory (block cache)
- Parallel scan (endpoint coprocessor)

Kylin Resources

- Web Site
http://kylin.io

- Google Groups
https://groups.google.com/forum/#!forum/kylin-olap

- Source Code
https://github.com/KylinOLAP/Kylin

Q & A

eBay Inc
GLOBAL PLATFORM
& INFRASTRUCTURE