Kylin Overview

--Hadoop OLAP Engine

Jiang Xu xjiang@ebay.com
Chief Architect of Kylin
Oct 2014

Agenda

- Background
- Tech Highlights
- Q & A

What's Kylin

- **kylin** / ˈkiːˈlɪn /
- --n. (in Chinese art) a mythical animal of composite form

An distributed Analytical Engine to provide SQL interface and multi-dimensional analysis (OLAP) on Hadoop to support TB to PB size analytics.

What We are Looking for…

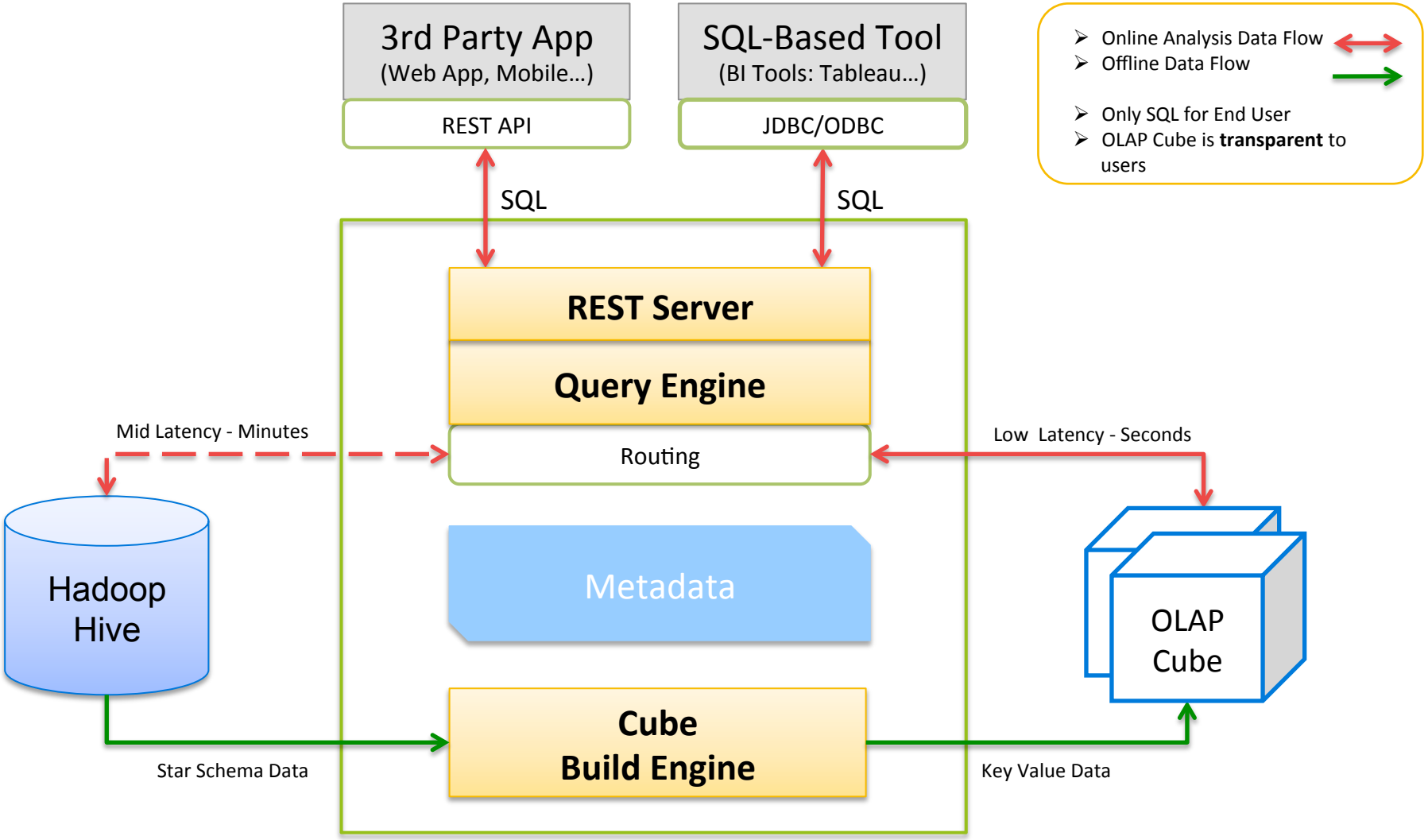
Capability and Features

- Full OLAP Analysis Engine on top of Hadoop Stack
- Unified SQL Interface for Analysts and Developers
- Distributed & Scale-Out Architecture for TB~PB size Analysis
- Low Latency and High Concurrency
- Integration with BI Tools (e.g. Tableau)
- Support High Dimensional and High Cardinality Data Set
- From Open Source to Open Source

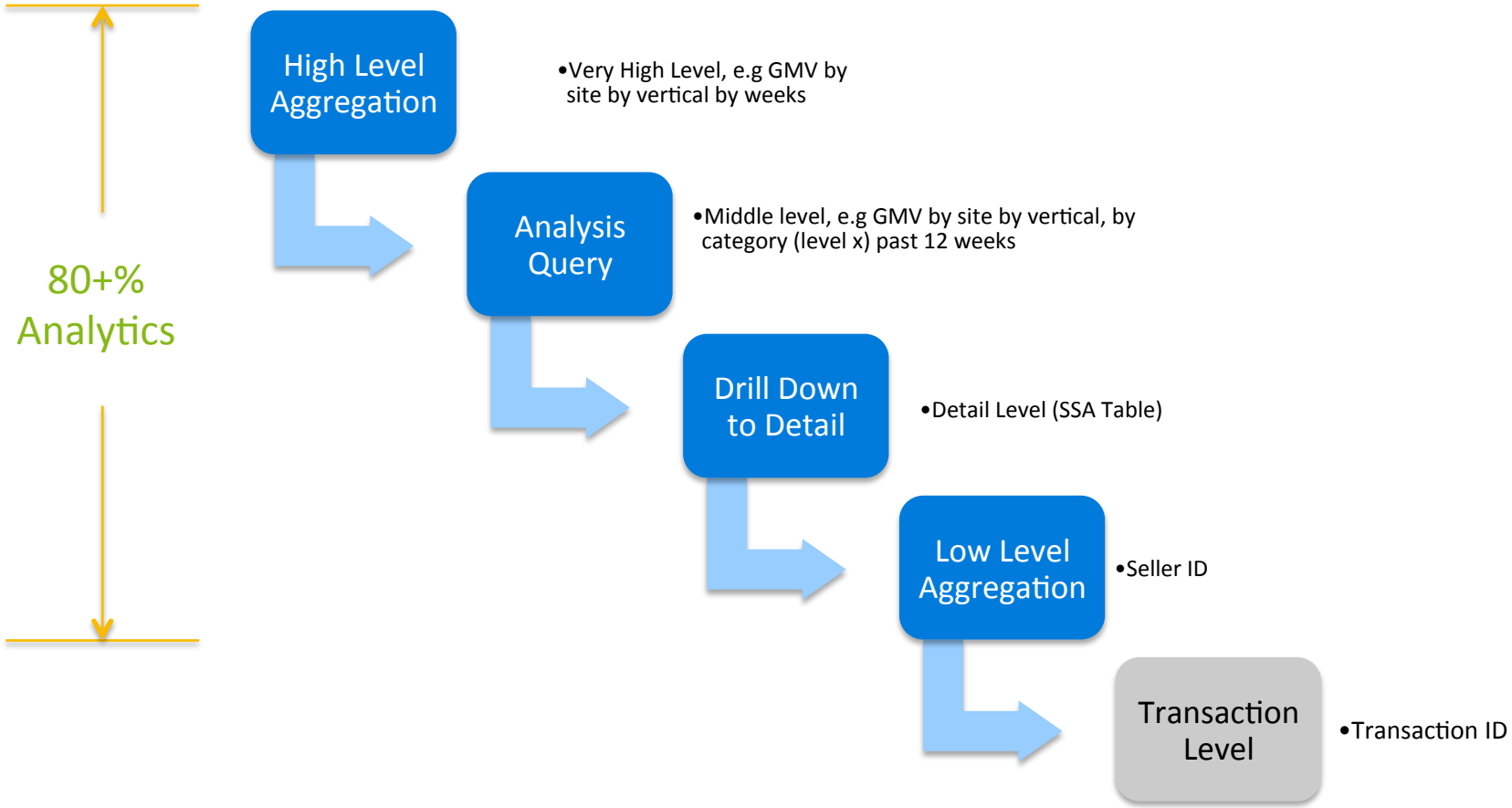
Glance of SQL-On-Hadoop Ecosystem…

- 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

Kylin Overview



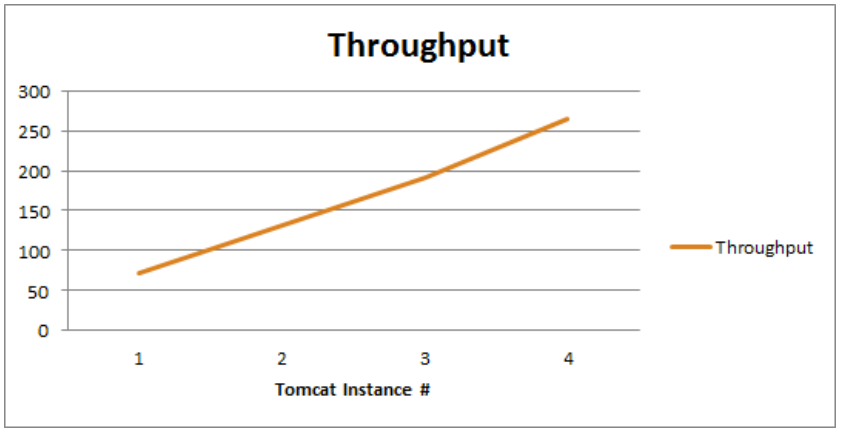
Analytics Query Taxonomy

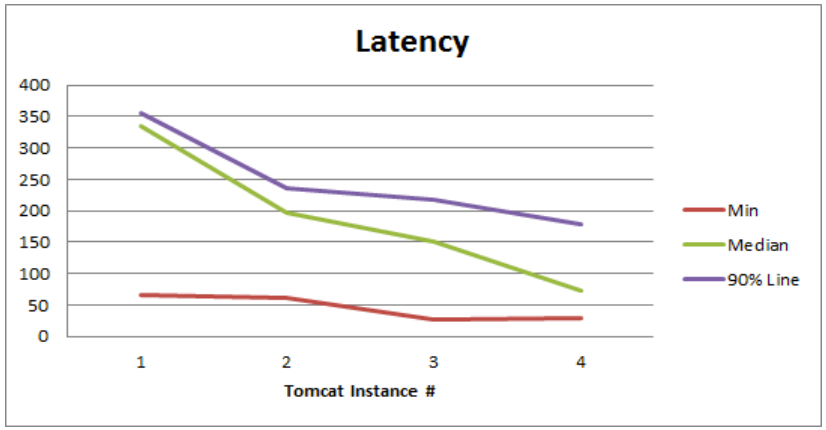


Query Performance - Latency & Throughput

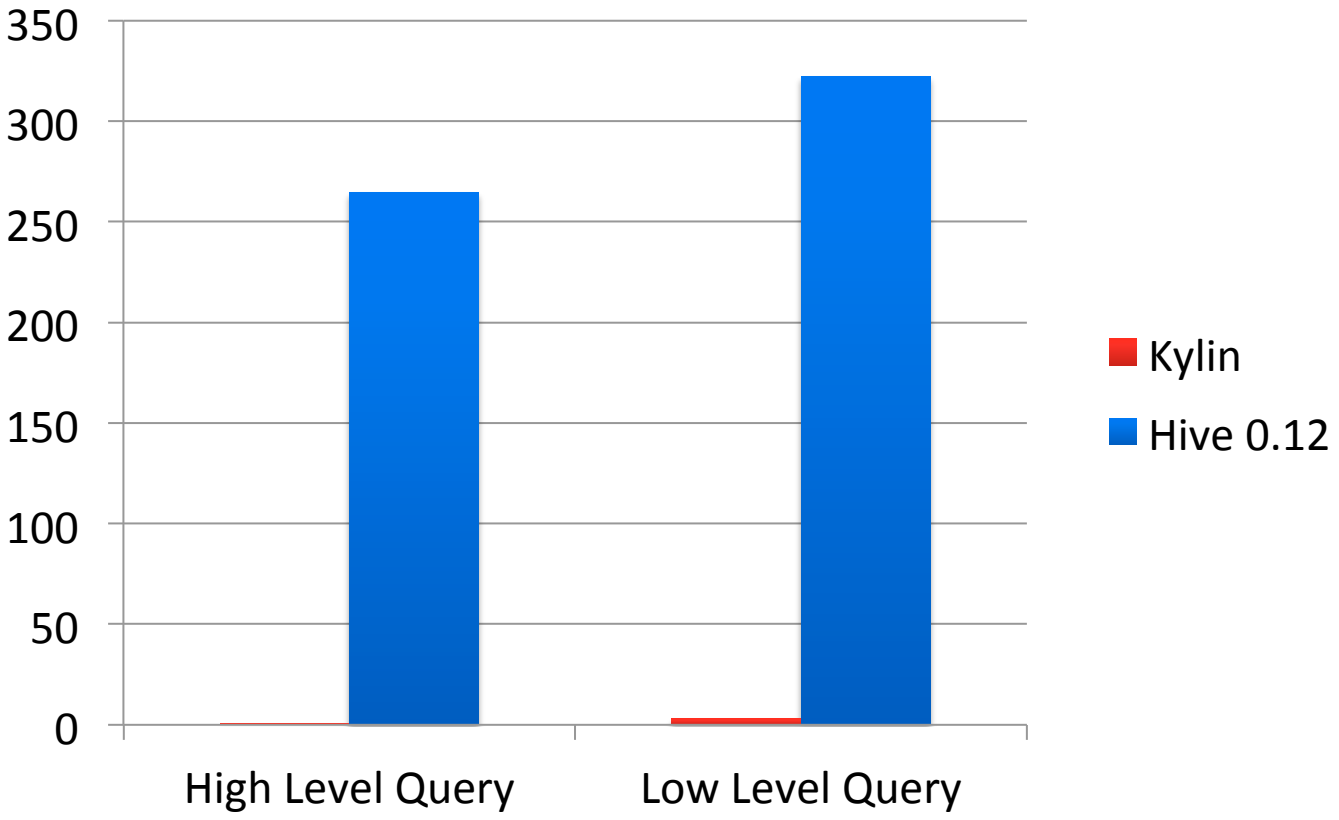
Single Tomcat Instance on a Single Machine

	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





Query Performance -- Compare to Hive



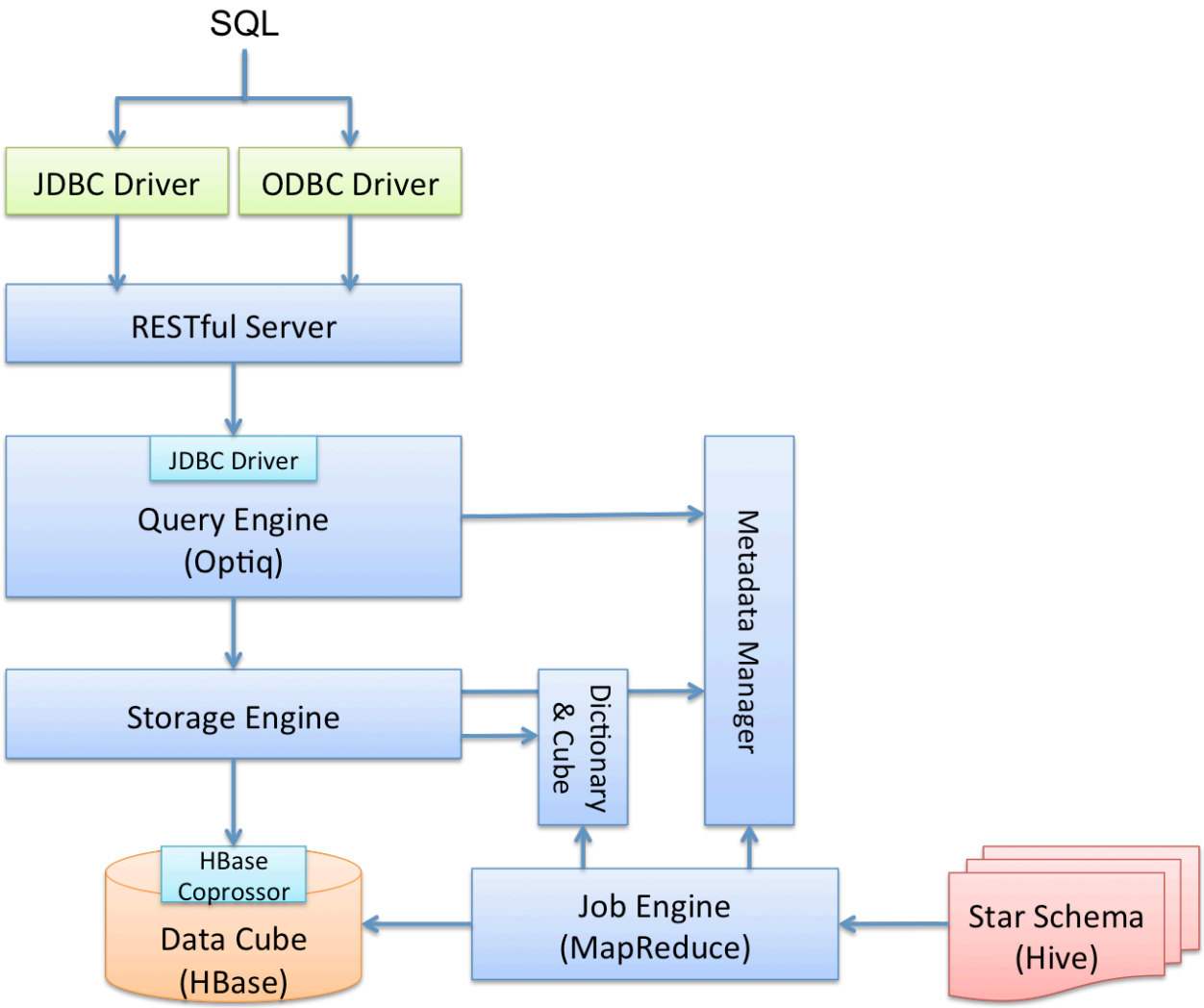
- *Kylin: 0.355 sec*
- *Hive: 264.226 sec*

- *Kylin: 3.171 sec*
- *Hive: 321.885 sec*

Agenda

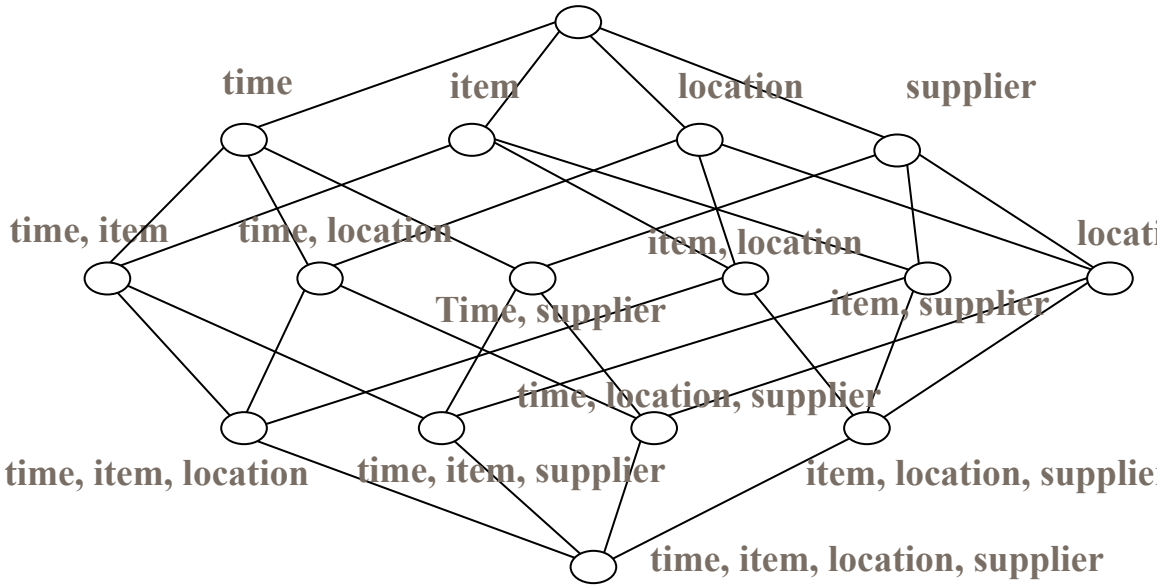
- Background
- Tech Highlights
- Q & A

Component Design



Background - Cube & Cuboid

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



0−D(apex) cuboid

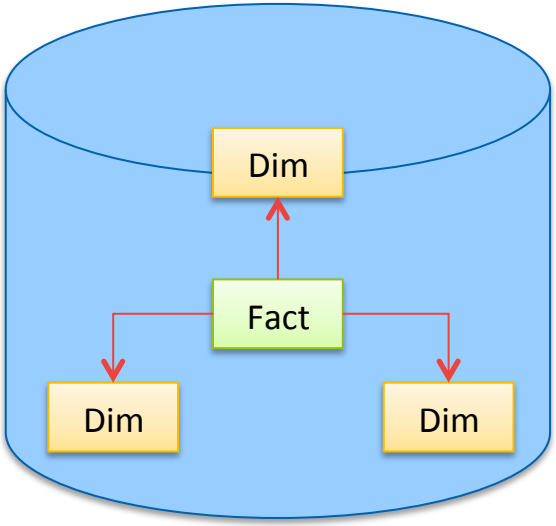
1−D cuboids

2−D cuboids

3−D cuboids

4−D(base) cuboid

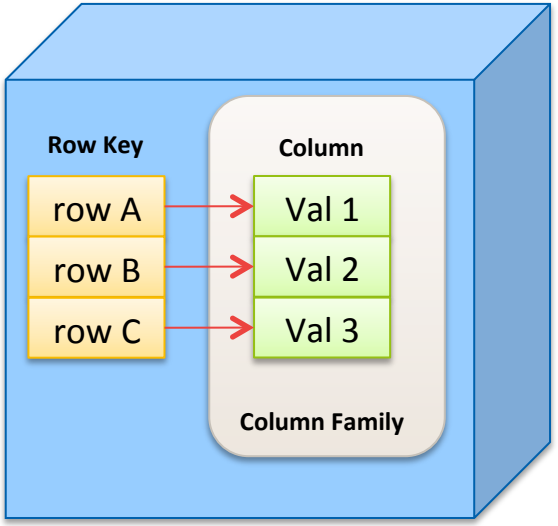
Metadata- Overview



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

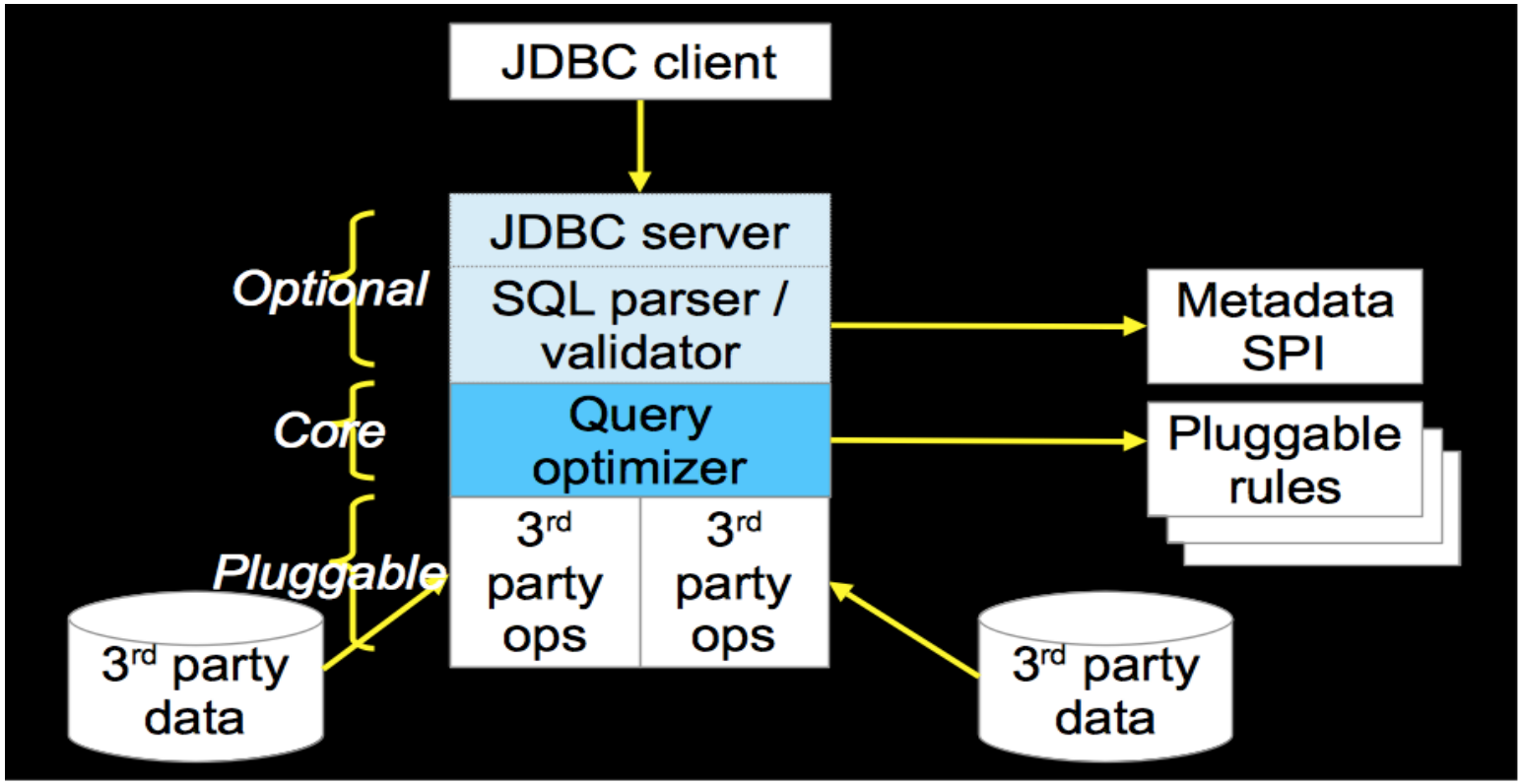
- Measure
  - Sum
  - Count
  - Max
  - Min
  - Average
  - Distinct Count (based on HyperLogLog)

Query Engine - Overview

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

Query Engine - Optiq

- **Optiq** is an *extensible* open source SQL engine that is also used in Stinger/Drill/Cascading



Query Engine - OLAP Explain Plan

```
SELECT test_cal_dt.week_beg_dt, test_category_groupings.meta_categ_name, test_category_groupings.categ_lvl2_name,
test_category_groupings.categ_lvl3_name, test_kylin_fact.lstg_format_name,  test_sites.site_name, SUM(test_kylin_fact.price) AS GMV,
COUNT(*) AS TRANS_CNT
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 = 10000002 OR test_kylin_fact.lstg_format_name = 'FP-non GTC'
GROUP BY test_cal_dt.week_beg_dt, test_category_groupings.meta_categ_name, test_category_groupings.categ_lvl2_name,
test_category_groupings.categ_lvl3_name, test_kylin_fact.lstg_format_name,test_sites.site_name
```

```
OLAPToEnumerableConverter
  OLAPProjectRel(WEEK_BEG_DT=[$0], META_CATEG_NAME=[$1], CATEG_LVL2_NAME=[$2], CATEG_LVL3_NAME=[$3],
LSTG_FORMAT_NAME=[$4], SITE_NAME=[$5], GMV=[CASE(=($7, 0), null, $6)], TRANS_CNT=[$8])
    OLAPAggregateRel(group=[{0, 1, 2, 3, 4, 5}], agg#0=[$SUM0($6)], agg#1=[COUNT($6)], TRANS_CNT=[COUNT()])
      OLAPProjectRel(WEEK_BEG_DT=[$13], META_CATEG_NAME=[$21], CATEG_LVL2_NAME=[$15], CATEG_LVL3_NAME=[$14],
LSTG_FORMAT_NAME=[$5], SITE_NAME=[$23], PRICE=[$0])
        OLAPFilterRel(condition=[OR(=($3, 10000002), =($5, 'FP-non GTC'))])
          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 – Optiq Plugin-ins

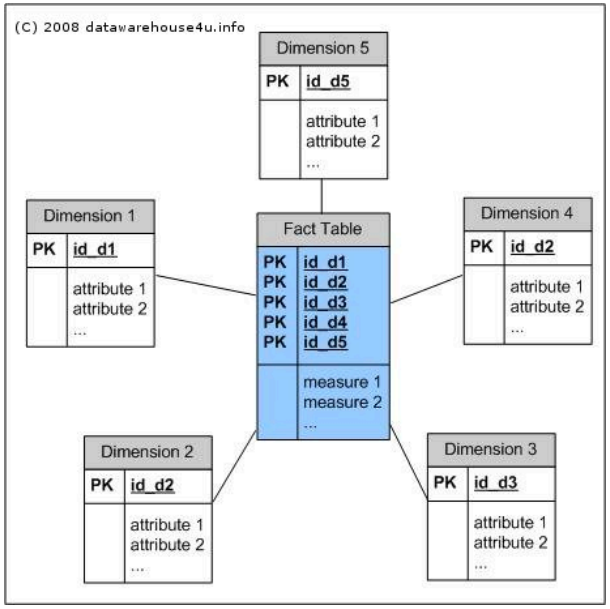
- Query Engine add following plugins into Optiq framework:
  - Metadata SPI:
    - Provide table schema from kylin metadata
  - Optimize Rule:
    - Translate the logic operator into kylin operator
  - Relational Operator:
    - Find right cube
    - Translate SQL into storage engine api call
    - Generate 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 - HBase Schema



Pre-Joined Flat Table	
Dimensions	D1
	D2
	…
	DX
Measures	M1
	M2
	…
	MY

Pre-Aggregated Table	
Dimensions	D3
	D6
	D8
Measures	M1
	M2
	…
	MY

Cuboid ID	D1	D2	…	DX

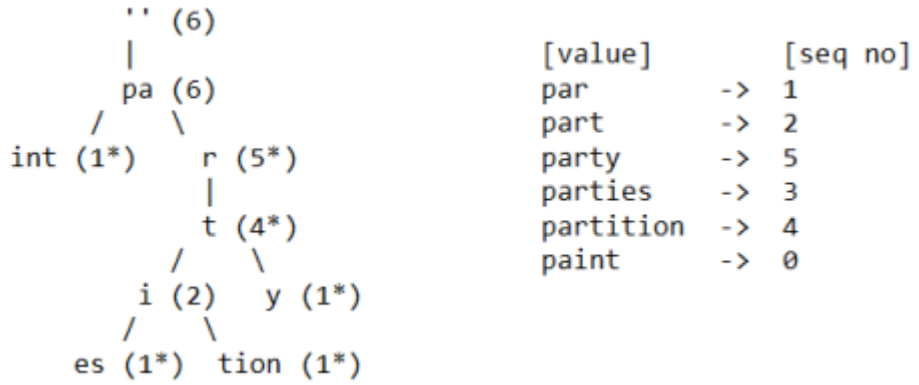
Row Key = Cuboid ID+Dimensions

M1	M2	…	MY

Row Value = Measures

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



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

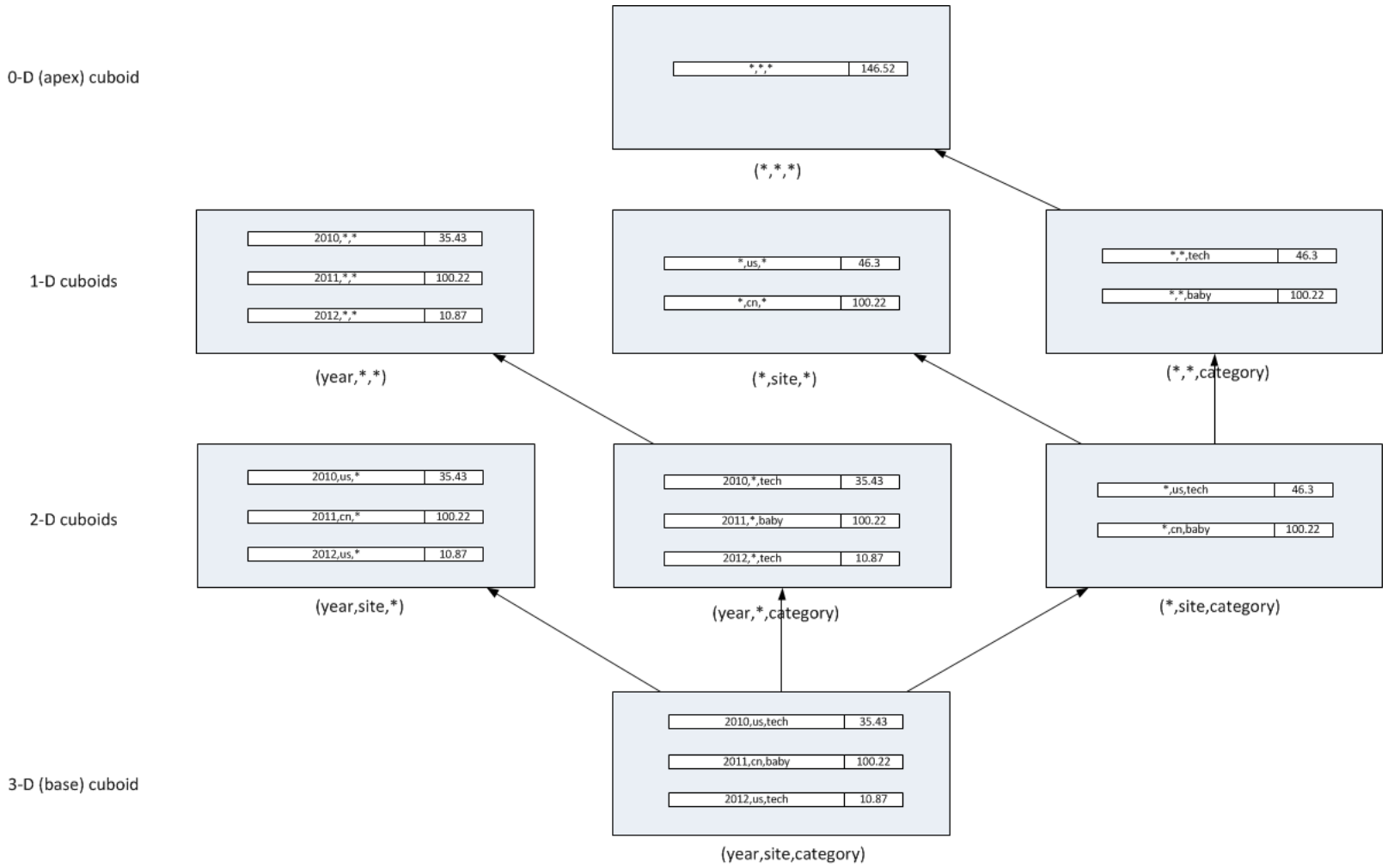
Storage Engine – HBase Coprocessor

- 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

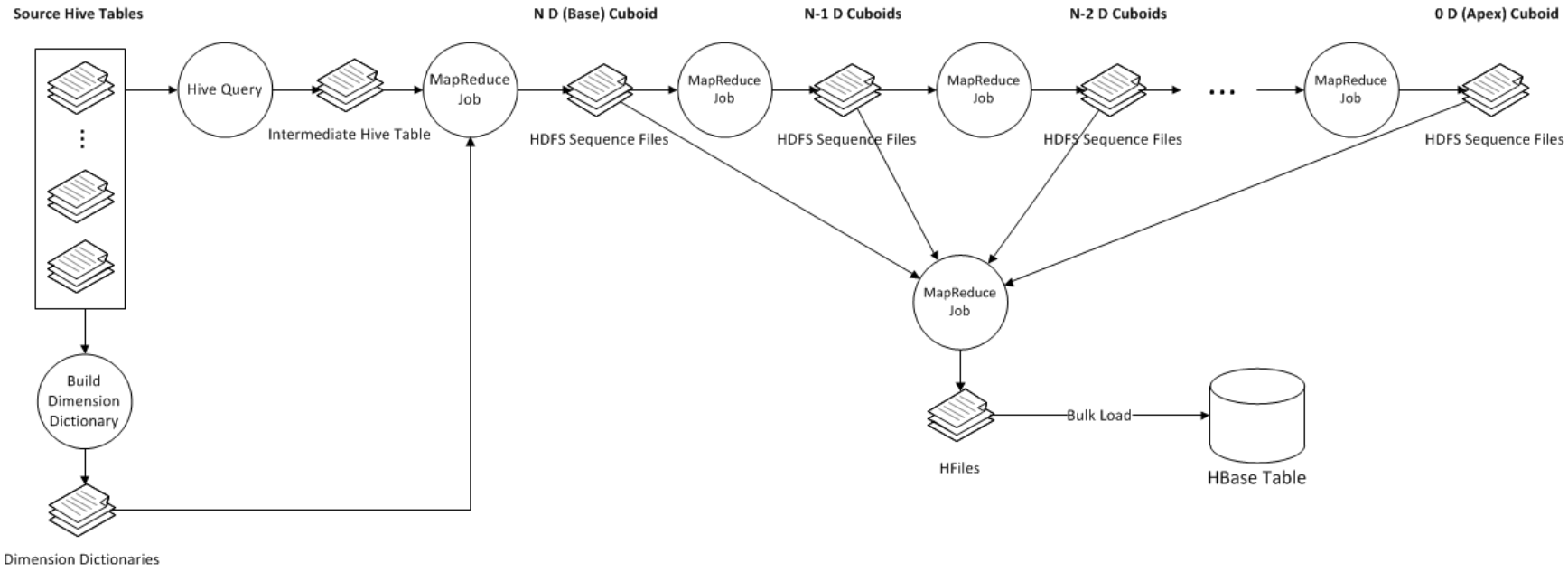
Cube Build - Overview

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

Cube Build – Multi Staged Build



Cube Build - Map Reduce Job Flow



Cube Build – Build Steps

- Build dictionary from dimension tables on local disk. And copy dictionary to HDFS
- Rune 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 2^N 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

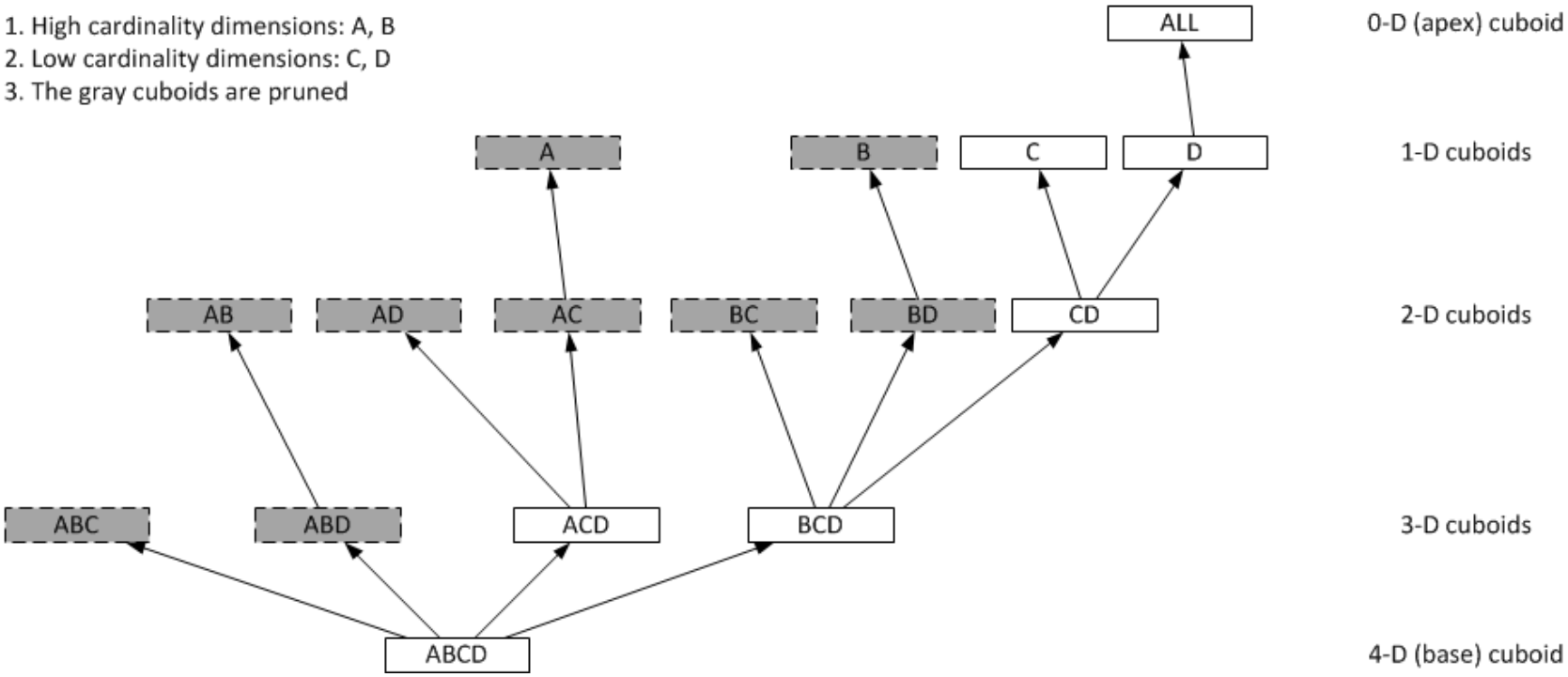
Cube Optimization – Full Cube vs. Partial Cube

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

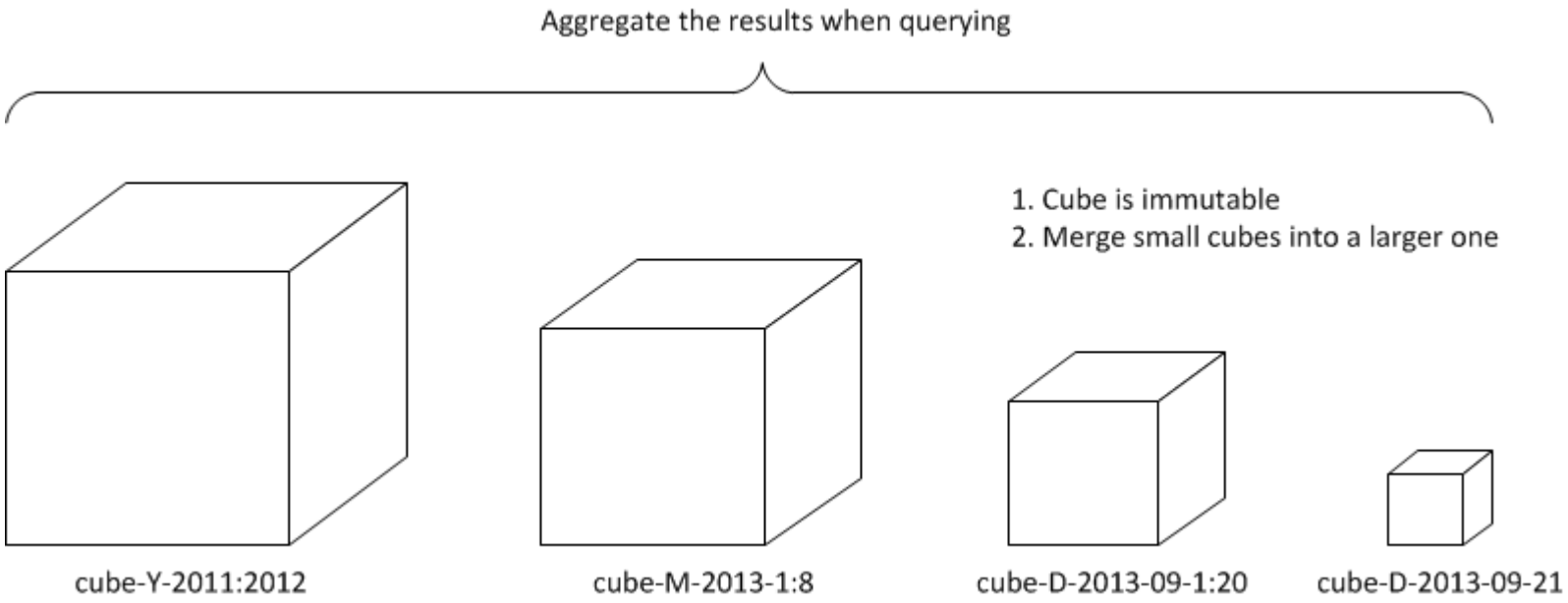
- Partial Cube
  - To avoid dimension explosion, we divide the dimensions into different aggregation groups
  - For cube with 30 dimensions, if we divide these dimensions into 3 group, the cuboid number will reduce from 1 Billion to 3 Thousands
  - Tradeoff between online aggregation and offline pre-aggregation

Cube Optimization – Partial Cube

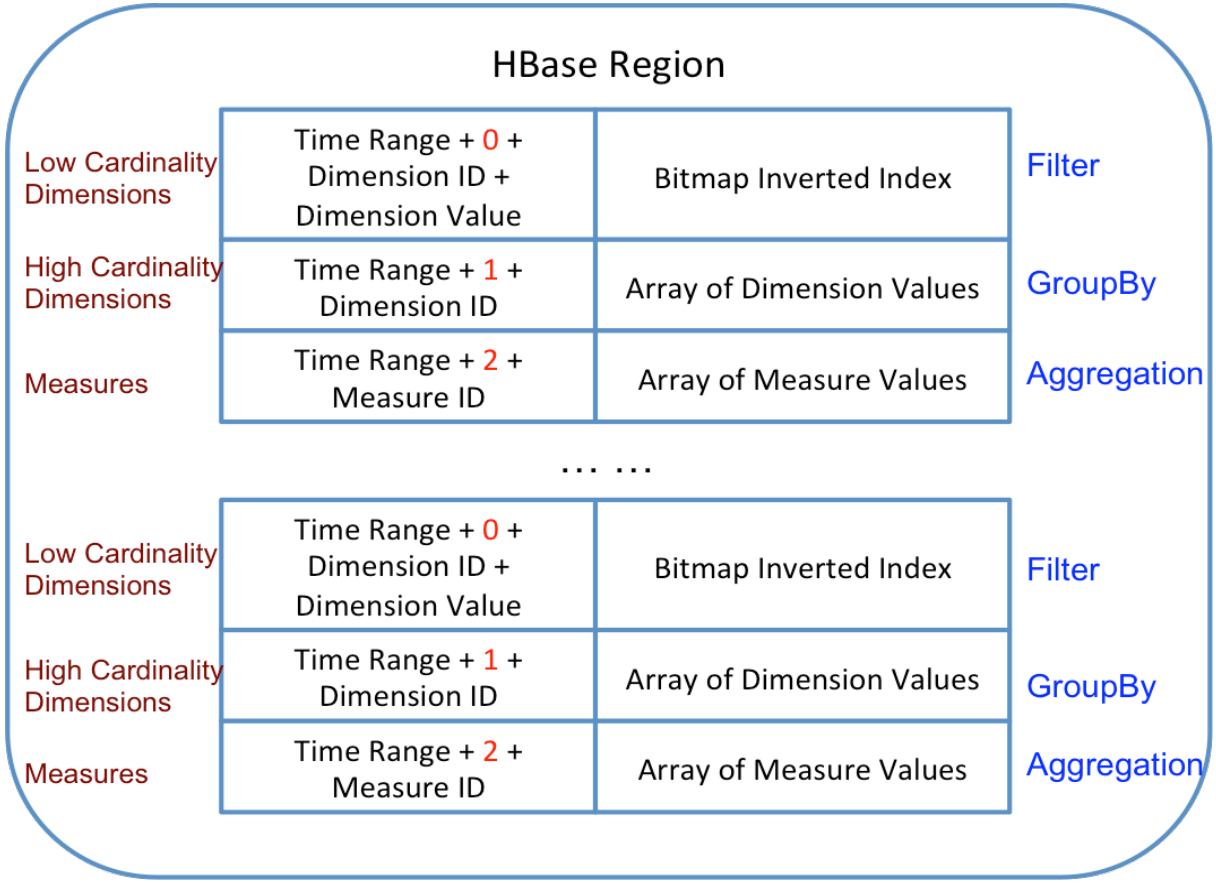
1. High cardinality dimensions: A, B
2. Low cardinality dimensions: C, D
3. The gray cuboids are pruned



Cube Optimization – Incremental Build



Cube Optimization – TopN Query on High Cardinality Dimension



- Bitmap inverted index
- Split high cardinality columns
- Time range partition
- In-memory (block cache)
- Parallel scan (coprocessor)

http://kylin.io