KIP-1: Parquet storage

Author: Yiming Xu, Mingming Ge

1. Background: Why Kylin on Parquet

Currently, Kylin uses Apache HBase as the storage for OLAP cubes.

HBase is very fast, while it also has some drawbacks:

- HBase is not real columnar storage;
- HBase has no secondary index; Rowkey is the only index;
- HBase has no encoding, Kylin has to do the encoding by itself;
- HBase does not fit for cloud deployment and auto-scaling;
- HBase has different API versions and has compatible issues (e.g, 0.98, 1.0, 1.1, 2.0);
- HBase has different vendor releases and has compatible issues (e.g, Cloudera's is not compatible with others);

This proposal is to use Apache Parquet + Spark to replace HBase:

- Parquet is an open-source columnar file format;
- Parquet is more cloud-friendly, can work with most FS including HDFS, S3, Azure Blob store, Ali OSS, etc;
- Parquet can integrate very well with Hadoop, Hive, Spark, Impala, and others;
- Support custom index;
- It is mature and stable;

2. Parquet file layouts on HDFS

- Storage layout is important for I/O optimizations
- Do as much as possible pruning before reading the file
    - Filter by folder, file name, etc
- Each Cuboid uses a dedicated folder
- Cube
    - Segment A
        - Cuboid-1111
            - part-0000-XXX.snappy.parquet
            - part-0001-XXX.snappy.parquet
        - Cuboid-1001
            - part-0000-XXX.snappy.parquet
            - part-0001-XXX.snappy.parquet
    - Segment B
        - Cuboid-1111
            - part-0000-XXX.snappy.parquet
            - ...
- Advantages
    - Filter by folder is good enough
    - Can dynamically add/remove cuboid without impact others (ShaofengShi: Currently Kylin won't dynamically add/remove cuboid)
- Disadvantage
    - Many folders when the cube has many cuboids (ShaofengShi: I think this will bring too many small files, it will increase the burden to HDFS, how can we overcome it?)

3. Dimension/measure layouts in Parquet

- Dimension and measures layouts in parquet files
  If there is a dimension combination of [d1, d2, d3] and measures of [m1, m2]then a parquet file like this will be generated:
  Columns 1, 2, and 3 correspond to Dimension d1, d2, and d3, respectively
  Column 110000 and 110001 respectively correspond to Measure m1, m2
- (ShaofengShi: Is "1", "110000" the column name in parquet? or the column name should be the original column name?)
- (ShaofengShi: How does the "110000" number come?)

```
Parquet file schema:
    1:           OPTIONAL INT64 R:0 D:1
    2:           REQUIRED DOUBLE R:0 D:0
    3:           OPTIONAL INT64 R:0 D:1
    110000:      OPTIONAL INT64 R:0 D:1
    110001:      OPTIONAL INT64 R:0 D:1
```

- "REQUIRED" and "OPTIONAL" correspond to "nullable" in database system.
- Parquet data type includes *BOOLEAN, INT32, INT64, INT96, FLOAT, DOUBLE and BYTE_ARRAY.* The data with string type in hive will be stored as BYTE_ARRAY in parquet.

- How to deal with the order of dimension and measure
    - In a parquet file, the order of the columns is always dimension first and measure last
    - There is no order between dimensions and between measures
- Parquet file split
    - parquet.block.size default 128mb
    - (ShaofengShi: How many row groups in a parquet file?)

4. Data types mapping in Parquet

- How do you encode the data into a parquet?
    - Kylin no longer needs to encode columns
    - Parquet will encode needed columns
- All data types can be accurately mapped to Parquet
    - Support with ParquetWriteSupport
        - StructType ArrayType MapType
    - Direct mapping transformation

Type	Spark	Parquet
Numeric types	ByteType	INT32
Numeric types	ShortType	INT32
Numeric types	IntegerType	INT32
Numeric types	LongType	INT64
Numeric types	FloatType	FLOAT
Numeric types	DoubleType	DOUBLE
Numeric types	DecimalType	INT32INT64BinaryTypeFIXED_LEN_BYTE_ARRAY
String type	StringType	BYTE_ARRAY
Binary type	BinaryType	BYTE_ARRAY
Boolean type	BooleanType	BOOLEAN
Datetime type	TimestampType	INT96
Datetime type	DateType	INT32

- How computed columns are stored
    - Bitmap: BYTE_ARRAY
    - TopN: BYTE_ARRAY

5. How to build Cube into Parquet

- Reduced build steps
    - From ten-twenty steps to only two steps
- Build Engine
    - Simple and clear architecture
    - Spark as the only build engine
    - All builds are done via spark
    - Adaptively adjust spark parameters
    - Dictionary of dimensions no longer needed
- Supported measures
    - Sum
    - Count
    - Min
    - Max
    - TopN
    - Bitmap
    - HyperLogLog
- Cube into parquet
  *

(ShaofengShi: this part need detailed info)

6. How to query with Parquet

- Query Engine: Sparder
  - Use spark as a calculation tool
  - Distributed query engineavoid single-point-of-failure
  - Unified calculation engine for building and querying
  - There is a substantial increase in query performance
  - Can benefit from spark new features and ecology

- The basic process of Sparder query
  1. Parser => Sql to AST tree
  2. Validation => Further verify the validity of SQL based on metadata
  3. Optimizer => Generate LogicPlan according to optimization rules
  4. Kylin's Adaptation => Convert AST's nodes to rel nodes(Various classes ending with Rel, such as FilterRel)
  5. Spark Plan => relnode to Spark plan
  6. Query Execution => Read cube data based on the generated spark plan

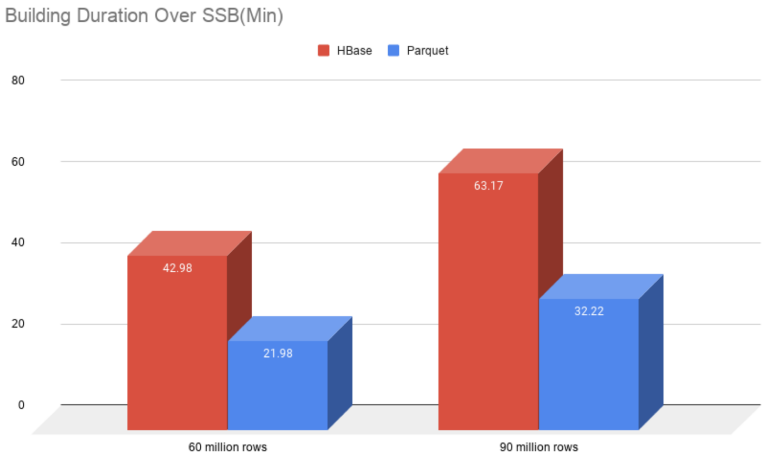
- What are the optimizations of Kylin reading parquet data
  - Segment Pruning
  - Shard by
  - Parquet page index
  - Project Pushdown
  - Predicate Pushdown

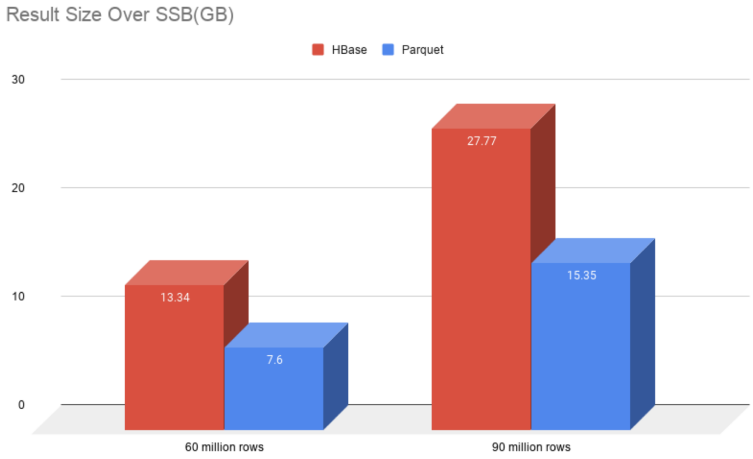
7. Performance

Kyligence provides dataset tool for SSB and TPC-H which contains test SQL case, the repositories are as follows:

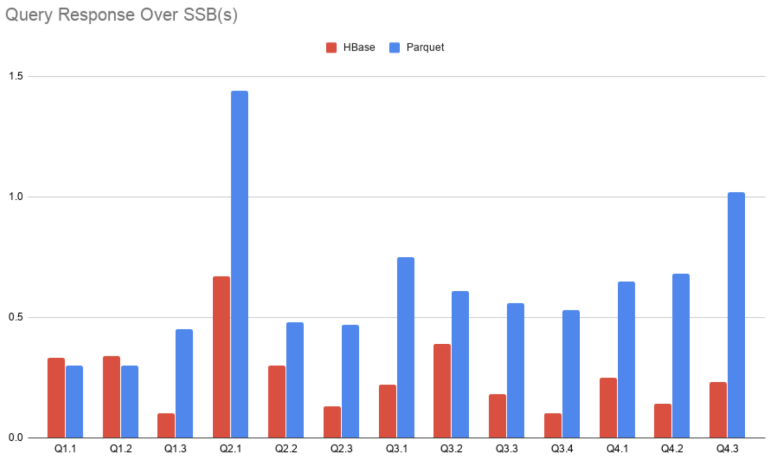
- https://github.com/Kyligence/ssb-kylin
- https://github.com/Kyligence/kylin-tpch

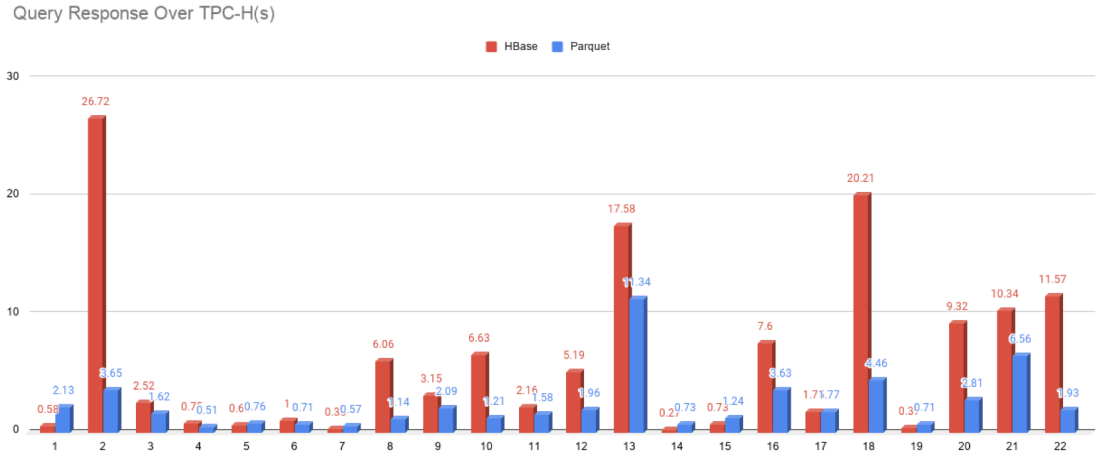
- Environment
  - 4 nodes hadoop cluster
  - Yarn queue has 400G memory and 128 cpu cores
- Build(Over SSB)





- Query(Over SSB and TPC-H)


Query Response Over SSB(s)
HBase
Parquet
1.5
1.0
0.5
0.0
Q1.1
Q1.2
Q1.3
Q2.1
Q2.2
Q2.3
Q3.1
Q3.2
Q3.3
Q3.4
Q4.1
Q4.2
Q4.3


Query Response Over TPC-H(s)
HBase
Parquet
30
20
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

8. Next step