Apache Kylin Cube 构建原理

作者: 康凯森

日期: 2016-10-06

分类: OLAP (../tag/OLAP.html)

本文主要介绍了Apache Kylin (http://kylin.apache.org/)是如何将Hive表中的数据转化为HBase (http://blog.bcmeng.com/post/hbase-note.html)的KV结构，并简单介绍了Kylin的SQL查询是如何转化为HBase的Scan操作。

Apache Kylin 是什么

Apache Kylin是一个开源的、基于Hadoop生态系统的OLAP查询引擎，能够通过SQL接口对十亿、甚至百亿行的超大数据集实现秒级的多维分析查询。

OLAP 是什么

即联机分析处理：以复杂的分析型查询为主，需要扫描，聚合大量数据。

Kylin如何实现超大数据集的秒级多维分析查询

预计算

对于超大数据集的复杂查询，既然现场计算需要花费较长时间，那么根据空间换时间的原理，我们就可以提前将所有可能的计算结果计算并存储下来，从而实现超大数据集的秒级多维分析查询。

Kylin的预计算是如何实现的

将数据源Hive表中的数据按照指定的维度和指标 由计算引擎MapReduce离线计算出所有可能的查询结果(即Cube)存储到HBase中。

Cube 和 Cuboid是什么

简单地说，一个cube就是一个Hive表的数据按照指定维度与指标计算出的所有组合结果。

其中每一种维度组合称为cuboid，一个cuboid包含一种具体维度组合下所有指标的值。

如下图，整个立方体称为1个cube，立方体中每个网格点称为1个cuboid，图中（A,B,C,D）和（A，D）都是cuboid，特别的，（A,B,C,D）称为Base cuboid。cube的计算过程是逐层计算的，首先计算Base cuboid，然后计算维度数依次减少，逐层向下计算每层的cuboid。

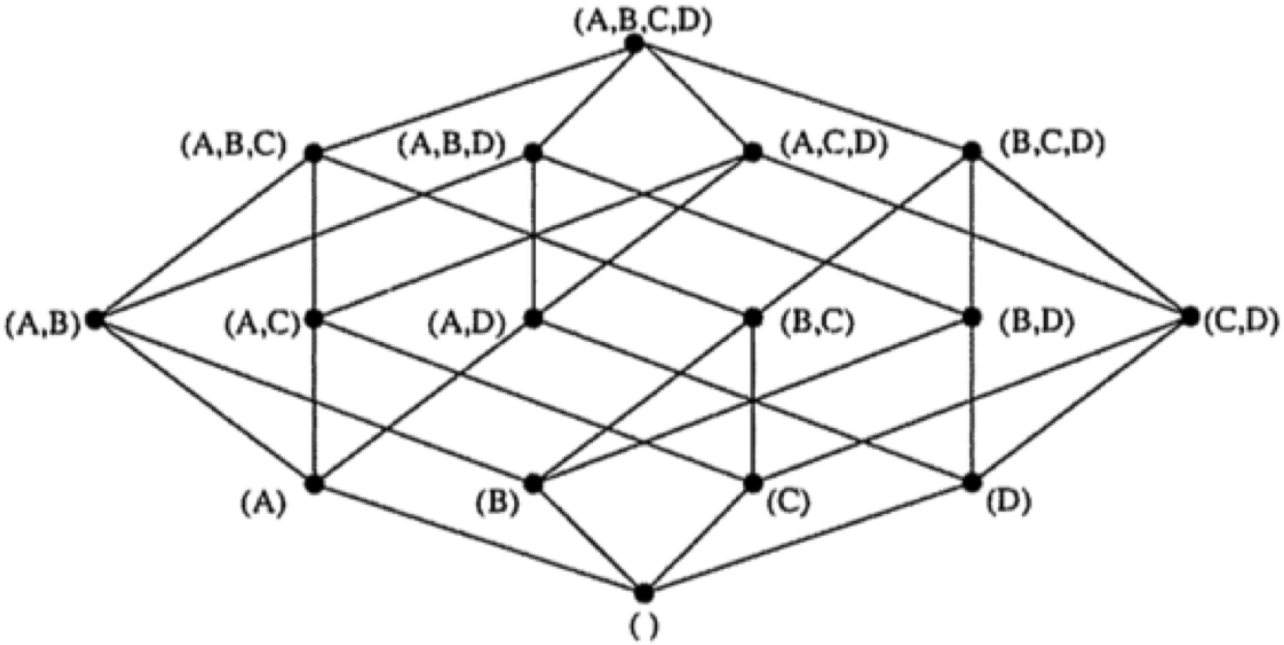

图1

Cuboid 的维度和指标如何转换为HBase的KV结构

简单的说Cuboid的维度会映射为HBase的Rowkey，Cuboid的指标会映射为HBase的Value。如下图所示：

原始表

year	city	price
1993	beijing	10
1993	beijing	30
1994	shanghai	20
1994	beijing	40

预聚合表

year	city	sum(price)
1993	beijing	40
1994	shanghai	20
1994	beijing	40
1993	*	40
1994	*	60
*	beijing	80
*	shanghai	50
*	*	100

字典编码

维度值	编码
1993	0
1994	1
beijing	0
shanghai	1

HBase KV存储

Rowkey	Value
00000000	100
00000001+0	80
00000001+1	50
00000010+0	40
00000010+1	60
00000011+00	40
00000011+10	40
00000011+11	20

图2

如上图原始表所示：Hive表有两个维度列 `year` 和 `city` ，有一个指标列 `price` 。

如上图预聚合表所示：我们具体要计算的是 `year` 和 `city` 这两个维度所有维度组合（即4个cuboid）下的 `sum(priece)` 指标，这个指标的具体计算过程就是由MapReduce完成的。

如上图字典编码所示：为了节省存储资源，Kylin对维度值进行了字典编码。图中将 `beijing` 和 `shanghai` 依次编码为0和1。

如上图HBase KV存储所示：在计算cuboid过程中，会将Hive表的数据转化为HBase的KV形式。Rowkey的具体格式是 `cuboid id + 具体的维度值`（最新的Rowkey中为了并发查询还加入了ShardKey），以预聚合表内容的第2行为例，其维度组合是（year，city），所以cuboid id就是00000011，cuboid是8位，具体维度值是1994和shanghai，所以编码后的维度值对应上图的字典编码也是11，所以HBase的Rowkey就是0000001111，对应的HBase Value就是 `sum(priece)` 的具体值。

所有的cuboid计算完成后，会将cuboid转化为HBase的 `KeyValue` 格式生成HBase的HFile，最后将HFile load进cube对应的HBase表中。

Cube 构建过程重要源码分析

1 从Hive表生成Base Cuboid

在实际的cube构建过程中，会首先根据cube的Hive事实表和维表生成一张大宽表，然后计算大宽表列的基数，建立维度字典，估算cuboid的大小，建立cube对应的HBase表，再计算base cuboid。

计算base cuboid就是一个MapReduce作业，其输入是上面提到的Hive大宽表，输出是的key是各种维度组合，value是Hive大宽表中指标的值。

```
org.apache.kylin.engine.mr.steps.BaseCuboidJob

map 阶段生成key-value的代码如下：

    protected void outputKV(Context context) throws IOException, InterruptedException {
        intermediateTableDesc.sanityCheck(bytesSplitter);

        byte[] rowKey = buildKey(bytesSplitter.getSplitBuffers());
        outputKey.set(rowKey, 0, rowKey.length);

        ByteBuffer valueBuf = buildValue(bytesSplitter.getSplitBuffers());
        outputValue.set(valueBuf.array(), 0, valueBuf.position());
        context.write(outputKey, outputValue);
    }

所有计算cuboid的reduce阶段代码都一样，见下面。
```

2 从Base Cuboid 逐层计算 Cuboid。

从base cuboid 逐层计算每层的cuboid，也是MapReduce作业，map阶段每层维度数依次减少，reduce阶段对指标进行聚合。

```
org.apache.kylin.engine.mr.steps.CuboidReducer

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        aggs.reset();  //MeasureAggregators 根据每种指标的不同类型对指标进行聚合

        for (Text value : values) {
            codec.decode(ByteBuffer.wrap(value.getBytes(), 0, value.getLength()), input);
            if (cuboidLevel > 0) { // Base Cuboid 的 cuboidLevel 是0
                aggs.aggregate(input, needAggr); //指标进行进一步聚合
            } else {
                aggs.aggregate(input);
            }
        }
        aggs.collectStates(result);

        ByteBuffer valueBuf = codec.encode(result);

        outputValue.set(valueBuf.array(), 0, valueBuf.position());
        context.write(key, outputValue);
```

3 Cuboid 转化为HBase的HFile。

主要就是数据格式的转化。详情请参考： Hive 数据 bulkload 导入 HBase (http://blog.bcmeng.com/post/hbase-bulkload.html)

不同类型的指标是如何进行聚合的

每种不同的指标都会有对应的聚合算法，所有指标聚合的基类是 `org.apache.kylin.measure.MeasureAggregator` 。其核心方法如下：

```
    abstract public void reset();
    //不同类型的指标算法会实现该方法
    abstract public void aggregate(V value);

    abstract public V getState();
```

以最简单的long类型的sum指标为例：

```
public class LongSumAggregator extends MeasureAggregator<LongMutable> {

    LongMutable sum = new LongMutable();

    @Override
    public void reset() {
        sum.set(0);
    }

    @Override
    public void aggregate(LongMutable value) {
        sum.set(sum.get() + value.get());
    }

    @Override
    public LongMutable getState() {
        return sum;
    }
}
```

SQL查询是如何转化为HBase的Scan操作的

还是以图2举例，假设查询SQL如下：

```
select year, sum(price)
from table
where city = "beijing"
group by year
```

这个SQL涉及维度 `year` 和 `city` ，所以其对应的cuboid是00000011，又因为city的值是确定的 `beijing` ,所以在Scan HBase时就会Scan Rowkey以00000011开头且city的值是 `beijing` 的行，取到对应指标 `sum(price)` 的值，返回给用户。

总结

本文主要介绍了Apache Kylin是如何将Hive表中的数据转化为HBase的KV结构，并简单介绍了Kylin的SQL查询是如何转化为HBase的Scan操作。希望对大家有所帮助。

赏

康凯森 (https://www.bcmeng.com/)