hulu

Optimize Ad Exposure Data Loading

Contents

hulu

What is Ad Exposure

- Ad Exposure - Events of user watching ads
- Hive Table:
  - a denormalized table with 33 dimensions and 3 measurements
  - hourly partitioned
- User PII is hashed for CCPA compliance
- Mainly loaded by a daily spark job
- An important data source for ad attribution, ad exposure
- millions of rows per hour

Problems of ad exposure data loading

- Too large file size
  - ~ 20GB per hour
  - Mainly caused by hash PII columns

- Too many stripes
  - ~ 400 stripes per hour
  - When data is used in downstream application, a stripe correspond to a partition, lead to a potential performance issue

- Poor scalability
  - For daily loading job, we partition data by hour to make sure a single file is generated for each hour. But it indicates we can only use up to 24 cores.

ORC Data Placement

- ORC use a hybrid data placement
  - Data is splitted by row into "stripe"s
  - Within a stripe, data is splitted by column into "stream"s
  - Pros: Flexible for different workload

user id	device id	starts
1	d1	1
2	d2	1
3	d3	0
4	d4	1
5	d5	1

A Logical Table

Stripe 1:
  Stream 1: [1, 2, 3]
  Stream 2: [d1, d2, d3]
  Stream 3: [1, 1, 0]

Stripe 2:
  Stream 1: [4, 5]
  Stream 2: [d4, d5]
  Stream 3: [1, 1]

A Physical ORC File

RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems

ORC Data Encoding

- ORC support a variety of encodings, and leverage a heuristic strategy to choose a suitable encoding
  - String encodings
    - Direct encoding
    - Dictionary encoding

Raw Data Stream	Direct Encoding Result	Dictionary Encoding Result
"foo", "bar", "foo"	"foo", "bar", "foo"	data:[0, 1, 0], dic:["foo", "bar"]

  - Integer encodings
    - Fixed encoding
    - Variable encoding

Raw Data	Fixed Encoding Result	Variable Encoding Result
127	0x00, 0x00, 0x00, 0x7F	0x7F
128	0x00, 0x00, 0x00, 0x10	0x01, 0x80

ORC Specification

ORC Tuning Properties

- Some ORC behaviors can be controlled by properties, and there are two types of properties
  - Table properties
    - orc.stripe.size
    - orc.compress
    - orc.compress.size
    - ...
  - Configuration properties
    - hive.exec.orc.default.stripe.size
    - hive.exec.orc.default.compress
    - hive.exec.orc.default.block.size
    - ...

ORC Properties

ORC Tuning Practice

- How to set ORC properties in a spark job

```
// to set table properties
dataframe
  .option("orc.stripe.size", "67108864")
  .option("orc.compress", "ZLIB")
  .orc("path-to-file")

// to set configuration properties
sparkSession.conf.set("hive.exec.orc.default.stripe.size", "67108864")
sparkSession.conf.set("hive.exec.orc.default.compress", "ZLIB")
```

- How to check ORC tuning result

```
hive --orcfiledump /hive/warehouse/adi.db/ad_exposure_data/realdate_utc=2019-03-01/hour=430947/063779_0
File Version: 0.12 with HIVE_8732
Rows: 295000
Compression: ZLIB
Compression size: 262144
...

Stripe Statistics:
  Stripe 1:
    Column 0: count: 67685 hasNull: false
    Column 1: count: 67685 hasNull: false min: 1551409200000 max: 1551412799000 sum: 105007252514515000
    Column 2: count: 67685 hasNull: false min: -1 max: 132353314 sum: 6072485150902
...
```

ORC Properties

Optimization 1 - Enlarge stripe size to reduce stripe number

- The default stripe size 64MB is too small for denormalized tables, which causes too many stripes
- Instead, we use 256MB stripe size, by property 'hive.exec.orc.default.stripe.size'
- Note that this parameter control stripe size before compression, the actual generated stripe will have a lower size

	Before Optimizing	After Optimizing
Expected Strip Size before Compression	64MB	256MB
Actual Stripe Size	30MB	90MB

Stripe size increase ✔️
Stripe number decrease ✔️
Data compression ratio increase ✔️

hulu

Optimization 2 - Use an efficient data encoding for hashed PII

- Before optimization, data is randomly distributed, and ORC use direct encoding for hashed PII. Instead, we can enforce a dictionary encoding for a higher space utility, by property "hive.exec.orc.dictionary.key.size.threshold"

- Further, as PII is dependent on user, we can cluster data by user id to improve dictionary encoding efficiency.

```
dataframe
  .repartition("user_id")
  .sortWithinPartition("user_id")
```

Rows within a stripe increase ✔️
Stripe number decrease ✔️
File size decrease ✔️

Optimization 3 - Use a optimized partitioner for user id clustering

- Problems of partition by user id: A partition may include data of multiple hours, which cause a task output data to multiple files.
  - High Memory Pressure for loading job(too many orc file writer for a partition)
  - Small file on HDFS

- As the table is partitioned by hour, we can also partition by hour for loading job. But still have problem: Data of a hour can only be processed by a core.
  - Poor scalability
  - High Memory Pressure for loading job(too many data for a partition)

hulu

Optimization 3 - Use a optimized partitioner for user id clustering cont.

- What a partitioner is required?
  - Data with same "user id + hour" should be distributed to same partition
  - Data with same "hour" should be distributed to a few partitions
- Besides following assumptions,
  - Distribute data of each hour to a fixed number of partitions
  - Each partition only contains data of same hour
- We get a two-level partitioner.

```
case class HourUserIdPartitioner(startHourId: Int, endHourId: Int, partitionsPerHour: Int)
  extends Partitioner {

  override val numPartitions: Int = (endHourId - startHourId) * partitionsPerHour

  override def getPartition(key: Any): Int = {
    key match {
      case (hourId: Int, userId: Option[Int]) =>
        (hourId - startHourId) * partitionsPerHour + (userId.getOrElse(0).abs % partitionsPerHour)
    }
  }
}
```

Job scalability increase ✔️
HDFS File number decrease ✔️

Optimization 4 - Scale backfill job

- Consider data backfill scenario, we need to load data of a long time range.
- Firstly, we cannot load data of full time range, as it leads to an overwhelming data amount. So, we need to split the job into tasks by several hours.

```
for ( ; startHourId < endHourId; startHourId += hoursPerTask) {
  loadDataForHours(startHourId, startHourId + hoursPerTask)
}
```

- Further, we use an adaptive method to determine for task splitting.
  hoursPerTask * partitionsPerHour = optimalParallelism * totalCoresOfSparkApp

Job scalability increase ✔️

hulu

Performance

For daily loading job with same resources,

	Before optimization	After optimization
Actual Stripe Size Per Hour	30MB	90MB
Rows Count Per Stripe	7w	170w
Stripes Count Per Hour	400	30
Job Running Time	5h	5h
File Size Per Hour	20GB	2GB

hulu

hulu

THANK YOU