深入分析Parquet列式存储格式

2015-08-11 大数据邦

Parquet是面向分析型业务的列式存储格式，由Twitter和Cloudera合作开发，2015年5月从Apache的孵化器里毕业成为Apache顶级项目，最新的版本是1.8.0。

列式存储

列式存储和行式存储相比有哪些优势呢？

1，可以跳过不符合条件的数据，只读取需要的数据，降低IO数据量。

2，压缩编码可以降低磁盘存储空间。由于同一列的数据类型是一样的，可以使用更高效的压缩编码（例如Run Length Encoding和DeltaEncoding）进一步节约存储空间。

3，只读取需要的列，支持向量运算，能够获取更好的扫描性能。

当时Twitter的日增数据量达到压缩之后的100TB+，存储在HDFS上，工程师会使用多种计算框架（例如MapReduce, Hive, Pig等）对这些数据做分析和挖掘；日志结构是复杂的嵌套数据类型，例如一个典型的日志的schema有87列，嵌套了7层。所以需要设计一种列式存储格式，既能支持关系型数据（简单数据类型），又能支持复杂的嵌套类型的数据，同时能够适配多种数据处理框架。

关系型数据的列式存储，可以将每一列的值直接排列下来，不用引入其他的概念，也不会丢失数据。关系型数据的列式存储比较好理解，而嵌套类型数据的列存储则会遇到一些麻烦。如图1所示，我们把嵌套数据类型的一行叫做一个记录(record)，嵌套数据类型的特点是一个record中的column除了可以是Int,Long, String这样的原语(primitive)类型以外，还可以是List, Map, Set这样的复杂类型。在行式存储中一行的多列是连续的写在一起的，在列式存储中数据按列分开存储，例如可以只读取A.B.C这一列的数据而不去读A.E和A.B.D，那么如何根据读取出来的各个列的数据重构出一行记录呢？

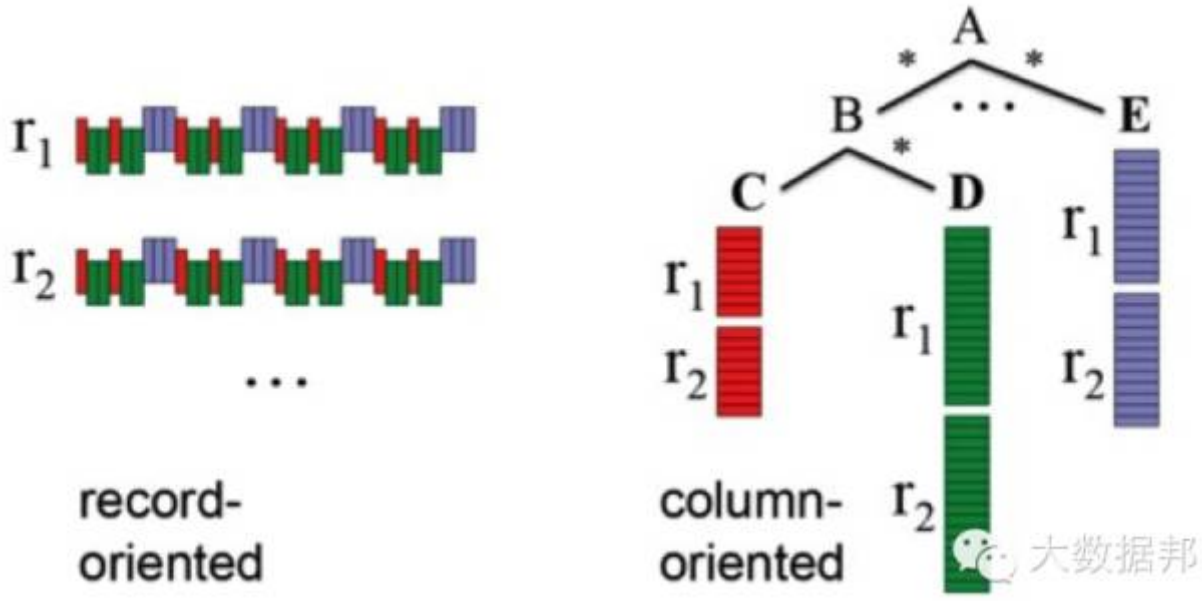

图1 行式存储和列式存储

Google的Dremel系统解决了这个问题，核心思想是使用“record shreddingand assembly algorithm”来表示复杂的嵌套数据类型，同时辅以按列的高效压缩和编码技术，实现降低存储空间，提高IO效率，降低上层应用延迟。Parquet就是基于Dremel的数据模型和算法实现的。
Parquet适配多种计算框架

Parquet是语言无关的，而且不与任何一种数据处理框架绑定在一起，适配多种语言和组件，能够与Parquet配合的组件有：
查询引擎: Hive, Impala, Pig, Presto, Drill, Tajo, HAWQ, IBM Big SQL
计算框架: MapReduce, Spark, Cascading, Crunch, Scalding, Kite
数据模型: Avro, Thrift, Protocol Buffers, POJOs

那么Parquet是如何与这些组件协作的呢？这个可以通过图2来说明。数据从内存到Parquet文件或者反过来的过程主要由以下三个部分组成：
1, 存储格式(storage format)
parquet-format项目定义了Parquet内部的数据类型、存储格式等。
2, 对象模型转换器(object model converters)
这部分功能由parquet-mr项目来实现，主要完成外部对象模型与Parquet内部数据类型的映射。
3, 对象模型(object models)
对象模型可以简单理解为内存中的数据表示，Avro, Thrift, Protocol Buffers, Hive SerDe, Pig Tuple, Spark SQLInternalRow等这些都是对象模型。Parquet也提供了一个example object model 帮助大家理解。
例如parquet-mr项目里的parquet-pig项目就是负责把内存中的Pig Tuple序列化并按列存储成Parquet格式，以及反过来把Parquet文件的数据反序列化成Pig Tuple。

这里需要注意的是Avro, Thrift, Protocol Buffers都有他们自己的存储格式，但是Parquet并没有使用他们，而是使用了自己在parquet-format项目里定义的存储格式。所以如果你的应用使用了Avro等对象模型，这些数据序列化到磁盘还是使用的parquet-mr定义的转换器把他们转换成Parquet自己的存储格式。

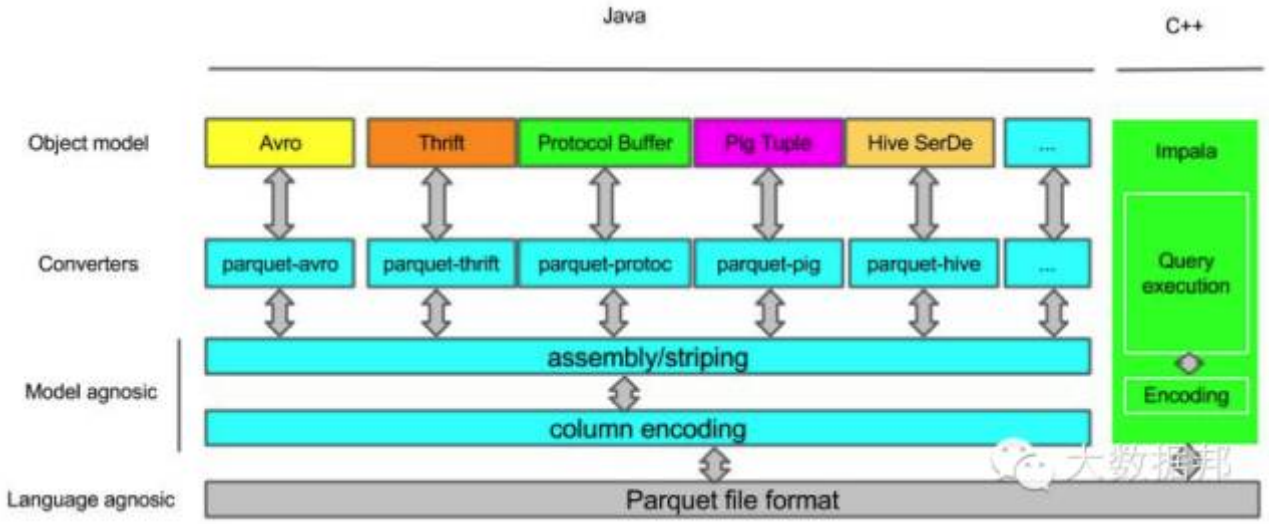

图2 Parquet项目的结构

Parquet数据模型

理解Parquet首先要理解这个列存储格式的数据模型。我们以一个下面这样的schema和数据为例来说明这个问题。

```
message AddressBook {
 required string owner;
 repeated string ownerPhoneNumbers;
 repeated group contacts {
  required string name;
  optional string phoneNumber;
 }
}
```

这个schema中每条记录表示一个人的AddressBook。有且只有一个owner，owner可以有0个或者多个ownerPhoneNumbers，owner可以有0个或者多个contacts。每个contact有且只有一个name，这个contact的phoneNumber可有可无。这个schema可以用图3的树结构来表示。

每个schema的结构是这样的：根叫做message，message包含多个fields。每个field包含三个属性：repetition, type, name。repetition可以是以下三种：required（出现1次），optional（出现0次或者1次），repeated（出现0次或者多次）。type可以是一个group或者一个primitive类型。

Parquet格式的数据类型没有复杂的Map，List，Set等，而是使用repeated fields 和groups来表示。例如List和Set可以被表示成一个repeated field，Map可以表示成一个包含有key-value 对的repeated field，而且key是required的。

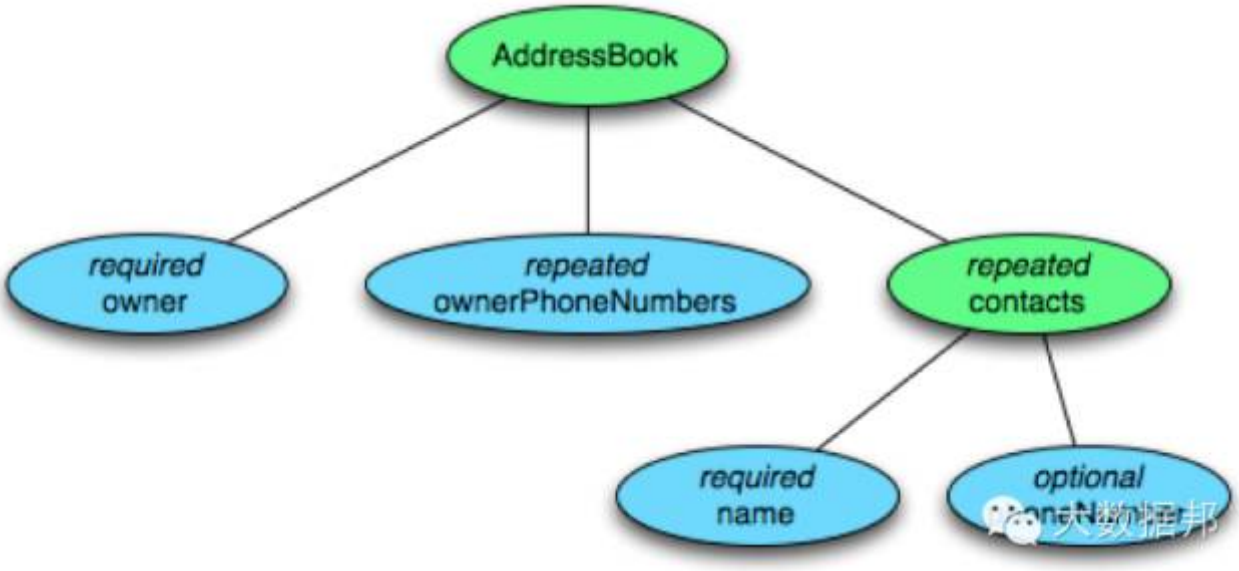

图3 AddressBook的树结构表示

Parquet文件的存储格式

那么如何把内存中每个AddressBook对象按照列式存储格式存储下来呢？
在Parquet格式的存储中，一个schema的树结构有几个叶子节点，实际的存储中就会有多少column。例如上面这个schema的数据存储实际上有四个column，如图4所示：

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

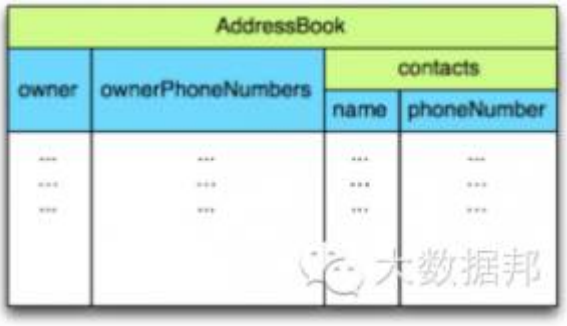

图4 AddressBook实际存储的列

Parquet文件在磁盘上的分布情况如图5所示。所有的数据被水平切分成Row group，一个Row group包含这个Row group对应的区间内的所有列的column chunk。一个column chunk负责存储某一列的数据，这些数据是这一列的Repetition levels，Definition levels和values（详见后文）。一个column chunk是由Page组成的，Page是压缩和编码的单元，对数据模型来说是透明的。一个Parquet文件最后是Footer，存储了文件的元数据信息和统计信息。Row group是数据读写时候的缓存单元，所以推荐设置较大的Row group从而带来较大的并行度，当然也需要较大的内存空间作为代价。一般情况下推荐配置一个Row group大小1G，一个HDFS块大小1G，一个HDFS文件只含有一个块。

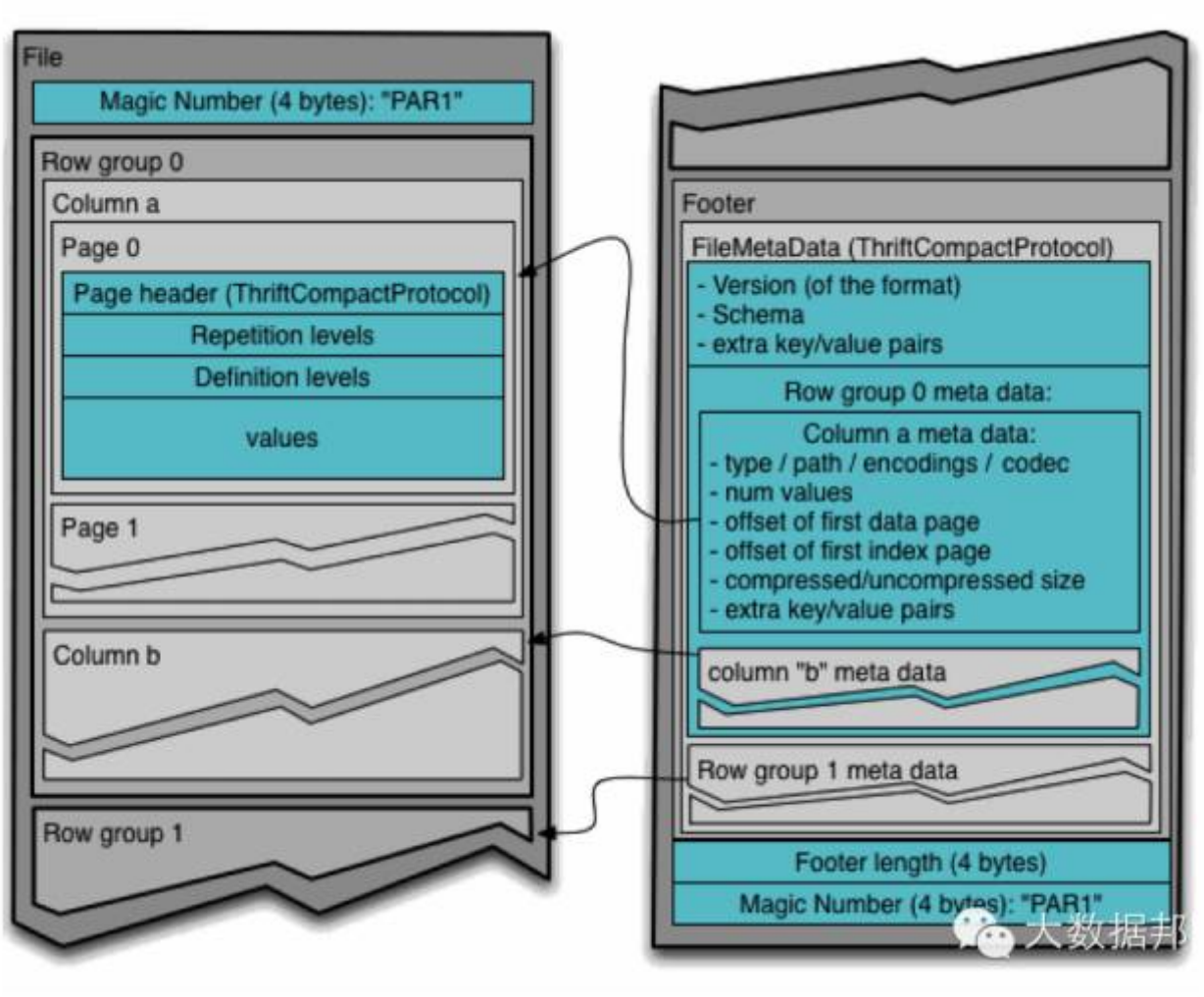

图5 Parquet文件格式在磁盘的分布

拿我们的这个schema为例，在任何一个Row group内，会顺序存储四个column chunk。这四个column都是string类型。这个时候Parquet就需要把内存中的AddressBook对象映射到四个string类型的column中。如果读取磁盘上的4个column要能够恢复出AddressBook对象。这就用到了我们前面提到的“record shredding and assembly algorithm”。

Striping/Assembly算法

对于嵌套数据类型，我们除了存储数据的value之外还需要两个变量Repetition Level(R),Definition Level(D) 才能存储其完整的信息用于序列化和反序列化嵌套数据类型。Repetition Level和 Definition Level可以说是为了支持嵌套类型而设计的，但是它同样适用于简单数据类型。在Parquet中我们只需定义和存储schema的叶子节点所在列的Repetition Level和Definition Level。

DefinitionLevel:

嵌套数据类型的特点是有些field可以是空的，也就是没有定义。**如果一个field是定义的，那么它的所有的父节点都是被定义的。从根节点开始遍历**，当某一个field的路径上的节点开始是空的时候我们记录下当前的深度作为这个field的Definition Level。如果一个field的Definition Level等于这个field的最大Definition Level就说明这个field是有数据的。对于required类型的field

必须是有定义的，所以这个Definition Level是不需要的。在关系型数据中，optional类型的field被编码成0表示空和1表示非空（或者反之）。

RepetitionLevel:
记录该field的值是在哪一个深度上重复的。只有repeated类型的field需要Repetition Level，optional 和required类型的不需要。Repetition Level = 0 表示开始一个新的record。在关系型数据中，repetion level总是0。

下面用AddressBook的例子来说明Striping和assembly的过程。
对于每个column的最大的Repetion Level和 Definition Level如图6所示：

Column	Max Definition level	Max Repetition level
owner	0 (owner is *required*)	0 (no repetition)
ownerPhoneNumbers	1	1 (*repeated*)
contacts.name	1 (name is *required*)	1 (contacts is *repeated*)
contacts.phoneNumber	2 (phoneNumber is *optional*)	1 (contacts is *repeated*)

图6 AddressBook的MaxDefinition Level和Max Repetition Level

下面这样两条record：

```
AddressBook {
 owner:"Julien Le Dem",
 ownerPhoneNumbers: "555 123 4567",
 ownerPhoneNumbers: "555 666 1337",
 contacts: {
  name: "Dmitriy Ryaboy",
  phoneNumber: "555 987 6543",
 },
 contacts: {
   name: "Chris Aniszczyk"
 }
}
AddressBook {
 owner:"A. Nonymous"
}
```

以contacts.phoneNumber这一列为例，"555987 6543"这个contacts.phoneNumber的DefinitionLevel是最大Definition Level=2。而如果一个contact没有phoneNumber，那么它的DefinitionLevel就是1。如果连contact都没有，那么它的DefinitionLevel就是0。

下面我们拿掉其他三个column只看contacts.phoneNumber这个column，把上面的两条record简化成下面的样子：

```
AddressBook {
 contacts: {
  phoneNumber: "555 987 6543"
 }
 contacts: {
 }
}
AddressBook {
}
```

这两条记录的序列化过程如图7所示：

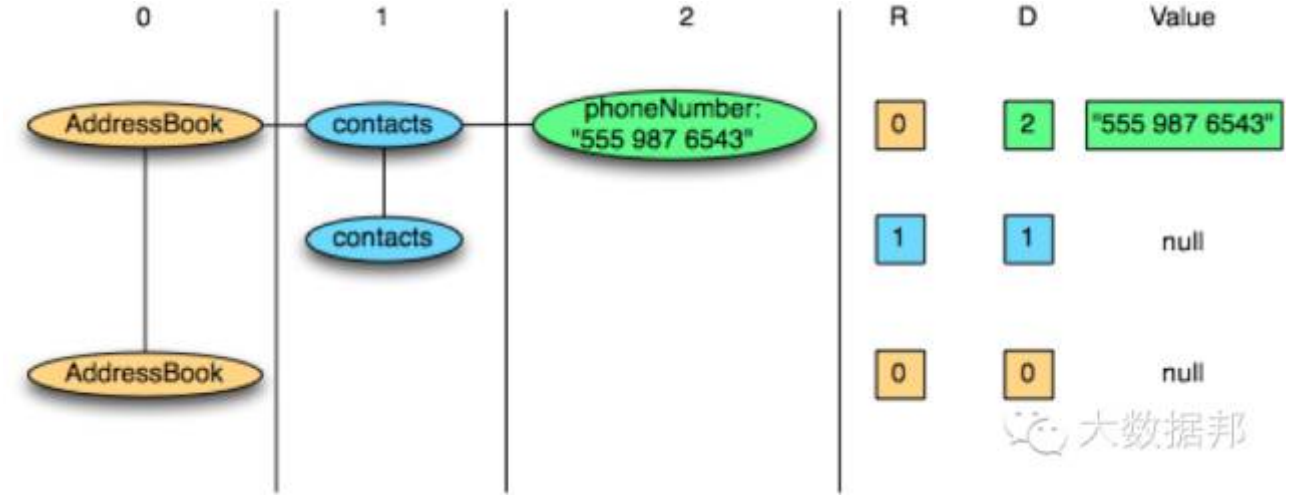

图7 一条记录的序列化过程

如果我们要把这个column写到磁盘上，磁盘上会写入这样的数据（图8）：

R	D	Value
0	2	"555 987 6543"
1	1	NULL
0	0	NULL

图8 一条记录的磁盘存储

注意：NULL实际上不会被存储，如果一个column value的Definition Level小于该column最大Definition Level的话，那么就表示这是一个空值。

下面是从磁盘上读取数据并反序列化成AddressBook对象的过程：

1，读取第一个三元组R=0, D=2, Value=”555 987 6543”

R=0 表示是一个新的record，要根据schema创建一个新的nested record直到Definition Level=2。

D=2 说明DefinitionLevel=Max DefinitionLevel，那么这个Value就是contacts.phoneNumber这一列的值，赋值操作contacts.phoneNumber=”555 987 6543”。

2，读取第二个三元组 R=1, D=1

R=1 表示不是一个新的record，是上一个record中一个新的contacts。

D=1 表示contacts定义了，但是contacts的下一个级别也就是phoneNumber没有被定义，所以创建一个空的contacts。

3，读取第三个三元组 R=0, D=0

R=0 表示一个新的record，根据schema创建一个新的nested record直到Definition Level=0，也就是创建一个AddressBook根节点。

可以看出在Parquet列式存储中，对于一个schema的所有叶子节点会被当成column存储，而且叶子节点一定是primitive类型的数据。对于这样一个primitive类型的数据会衍生出三个sub columns (R, D,Value)，也就是从逻辑上看除了数据本身以外会存储大量的Definition Level和Repetition Level。那么这些Definition Level和Repetition Level是否会带来额外的存储开销呢？实际上这部分额外的存储开销是可以忽略的。因为对于一个schema来说level都是有上限的，而且非repeated类型的field不需要RepetitionLevel，required类型的field不需要DefinitionLevel，也可以缩短这个上限。例如对于Twitter的7层嵌套的schema来说，只需要3个bits就可以表示这两个Level了。

对于存储关系型的record，record中的元素都是非空的（NOT NULL in SQL）。Repetion Level和DefinitionLevel都是0，所以这两个subcolumn就完全不需要存储了。所以在存储非嵌套类型的时候，Parquet格式也是一样高效的。

上面演示了一个column的写入和重构，那么在不同column之间是怎么跳转的呢，这里用到了有限状态机的知识，详细介绍可以参考Dremel。

数据压缩算法

列式存储给数据压缩也提供了更大的发挥空间，除了我们常见的snappy, gzip等压缩方法以外，由于列式存储同一列的数据类型是一致的，所以可以使用更多的压缩算法。

压缩算法	使用场景
Run Length Encoding	重复数据
Delta Encoding	有序数据集，例如timestamp，自动生成的ID，以及监控的各种metrics
Dictionary Encoding	小规模的数据集合，例如IP地址
Prefix Encoding	Delta Encoding for strings

性能

Parquet列式存储带来的性能上的提高在业内已经得到了充分的认可，特别是当你们的表非常宽（column非常多）的时候，Parquet无论在资源利用率还是性能上都优势明显。**具体的性能指标详见参考文档。**

Spark已经将Parquet设为默认的文件存储格式，Cloudera投入了很多工程师到Impala+Parquet相关开发中，Hive/Pig都原生支持Parquet。Parquet现在为Twitter至少节省了1/3的存储空间，同时节省了大量的表扫描和反序列化的时间。这两方面直接反应就是节约成本和提高性能。

如果说HDFS是大数据时代文件系统的事实标准的话，Parquet就是大数据时代存储格式的事实标准。

参考文档

http://parquet.apache.org/
https://blog.twitter.com/2013/dremel-made-simple-with-parquet
http://blog.cloudera.com/blog/2015/04/using-apache-parquet-at-appnexus/
http://blog.cloudera.com/blog/2014/05/using-impala-at-scale-at-allstate/

作者简介：

梁堰波，现就职于明略数据，开源爱好者，Apache Hadoop & Spark contributor。北京航空航天大学计算机硕士，曾就职于Yahoo!、美团网、法国电信，具备丰富的大数据、数据挖掘和机器学习领域的项目经验。

回复 美人 大数据分析：如何成为美人？

回复 色情 大数据！你能在色情行业里做什么！