Parquet文件

Parquet是一种开源的面向列的数据存储格式，它提供了各种存储优化，尤其适合数据

分析。Parquet提供列压缩从而可以节省空间，而且它支持按列读取而非整个文件地读

取。作为一种文件格式，Parquet与Apache Spark配合得很好，而且实际上也是Spark

的默认文件格式。我们建议将数据写到Parquet以便长期存储，因为从Parquet文件读

取始终比从JSON文件或CSV文件效率更高。Parquet的另一个优点是它支持复杂类

型，也就是说如果列是一个数组（CSV文件无法存储数组列）、map映射或struct结构

体，仍可以正常读取和写入，不会出现任何问题。以下代码指定Parquet为文件读取格

式：

spark.read.format("parquet")

读Parquet文件

Parquet的可选项很少，因为它在存储数据时执行本身的schema，因此，你只需要指定

一下Parquet格式。如果我们对DataFrame的模式（schema）有严格的要求，则可以设

置schema。一般来说，在读取的时候使用默认的schema，所以就不需要再设置了，这

类似于CSV文件的infer Schema。然而由于schema内置于文件中，不需要推断，所以

Parquet文件格式更强大。

以下是一些从parquet中读取数据的简单例子：

spark.read.format(“parquet")

// in Scala

spark.read.format(“parquet")

.load(“/data/flflight-data/parquet/2010-summary.parquet").show(5)

# in Python

spark.read.format(“parquet")\

.load(“/data/flflight-data/parquet/2010-summary.parquet").show(5)

Parquet可选项

由于Parquet含有明确定义且与Spark概念密切一致的规范，所以它只有很少的可选

项，实际上只有两个，表9-5列出了可选项。

虽然只有两个选项，如果你使用的是不兼容的Parquet文件，仍然会遇到问题。

当使用不同版本的Spark（尤其是旧版本时）写入Parquet文件时要小心，因为

这可能会导致让人头疼的问题。

表9-5：Parquet数据源选项

read/

write Key

取值范围

默认值

说明

write compression None, uncom None

声明Spark应该使用什

或codec pressed, bzip2, deflate,

么压缩编解码器来读取

gzip, lz4,或 snappy

或写入文件

read merge Schema true，false

配置值spark. 增量地添加列到同一表

sql.par quet. /文件夹中的Parquet文

mergeSchema 件里，此选项用于启用

或禁用此功能

写Parquet文件

写Parquet文件和读取它一样简单，只需指定文件的位置即可。这里应用相同的分片规

则：

// in Scala

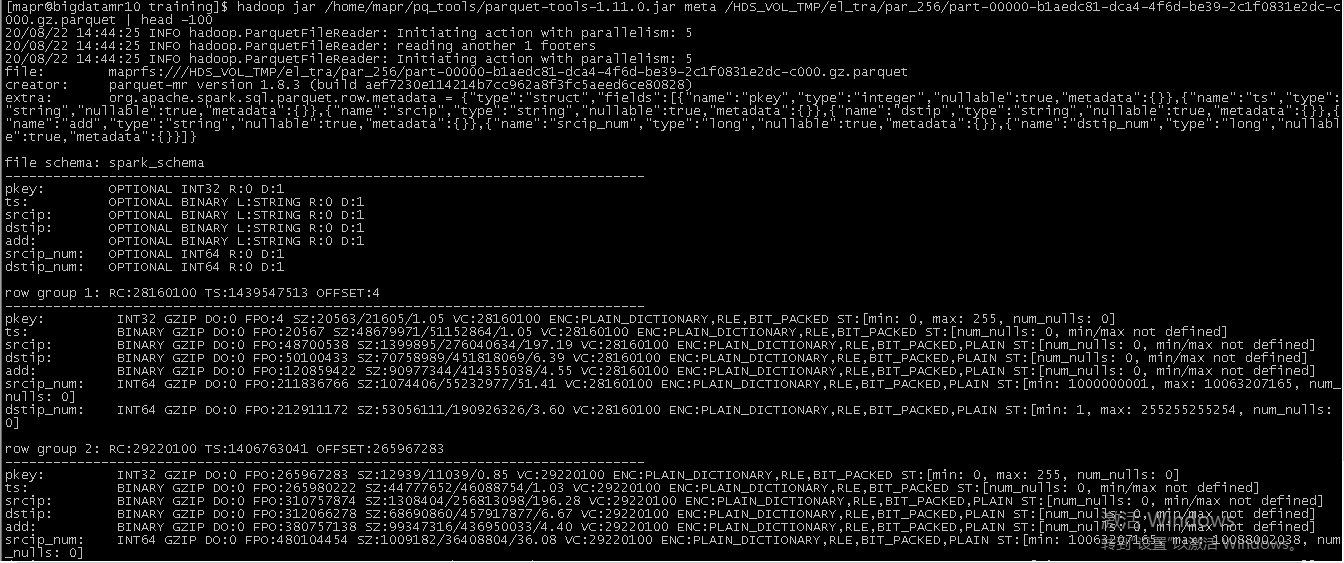
csvFile.write.format(“parquet").mode(“overwrite")

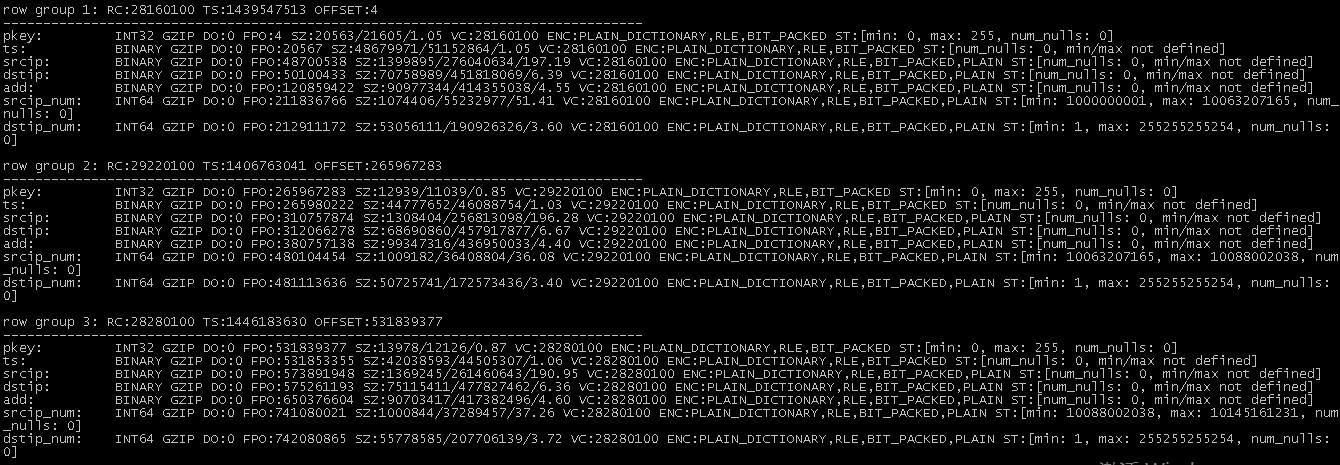
.save(“/tmp/my-parquet-fifile.parquet")

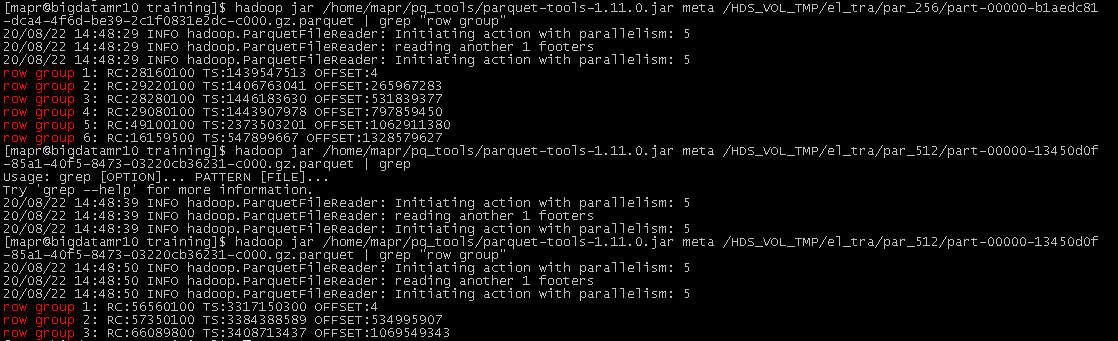
# in Python

csvFile.write.format(“parquet").mode(“overwrite")\

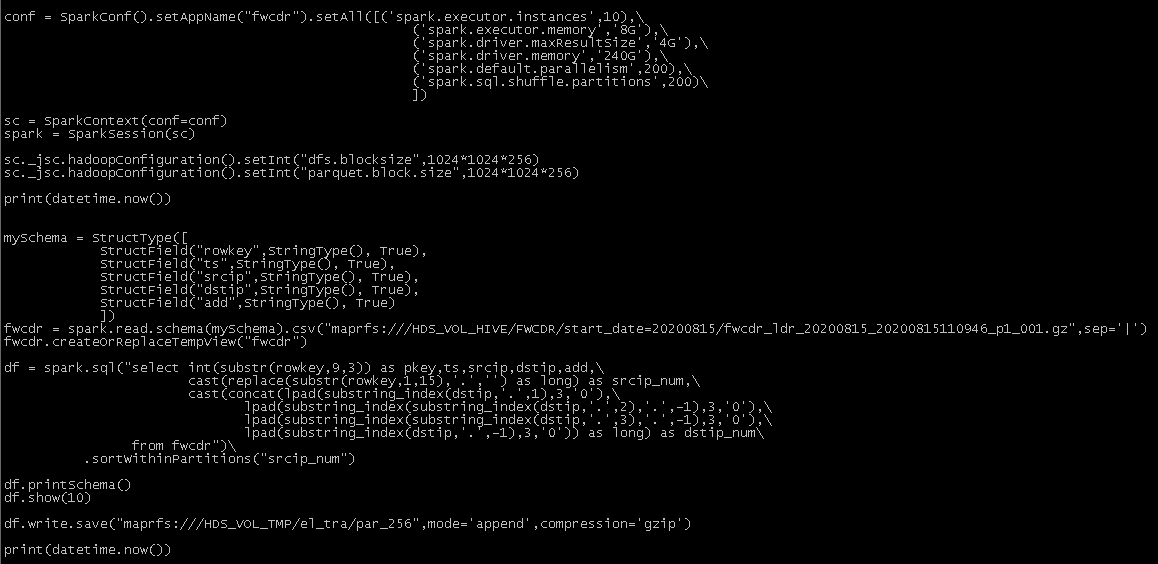
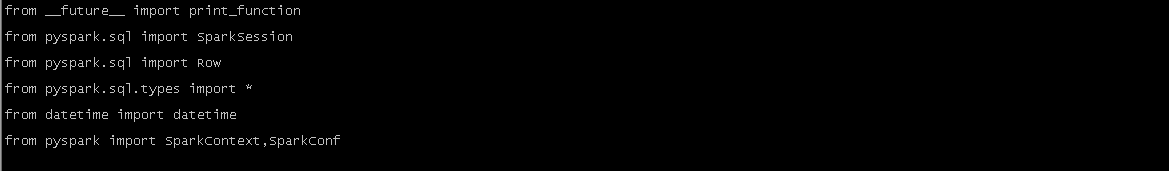
.save(“/tmp/my-parquet-fifile.parquet")

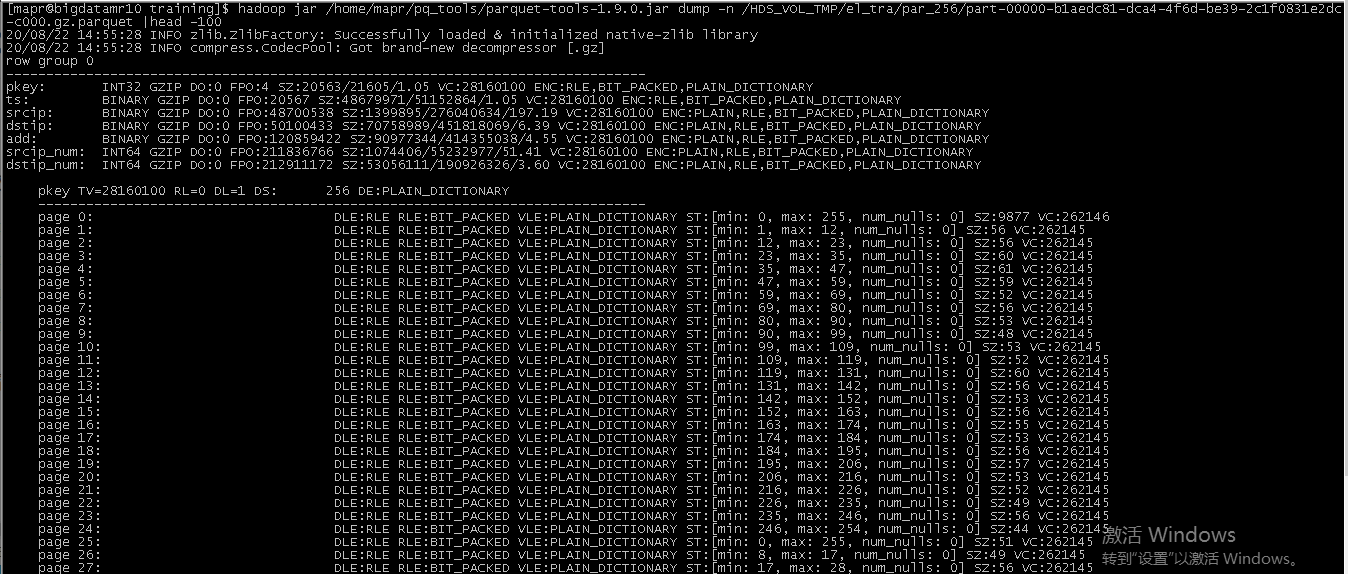




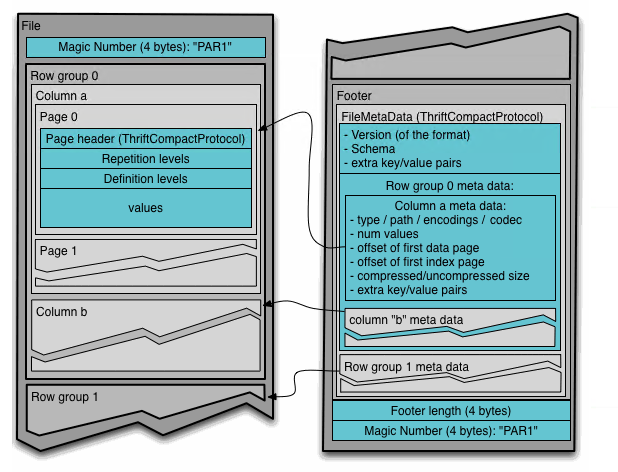


Parquet 内部的数据类型、存储格式





apacheparquet是一种列式数据存储格式，它提供了一种按列存储表格数据的方法。相同日期时间的列以Parquet格式作为行存储在一起，以便提供更好的存储、压缩和数据检索。



## Metadata

元数据有三种类型： file metadata, column (chunk) metadata and page header metadata。所有协议都使用TCompactProtocol序列化。

## Types

BOOLEAN: 1 bit boolean

INT32: 32 bit signed ints

INT64: 64 bit signed ints

INT96: 96 bit signed ints

FLOAT: IEEE 32-bit floating point values

DOUBLE: IEEE 64-bit floating point values

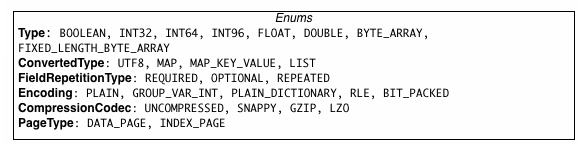
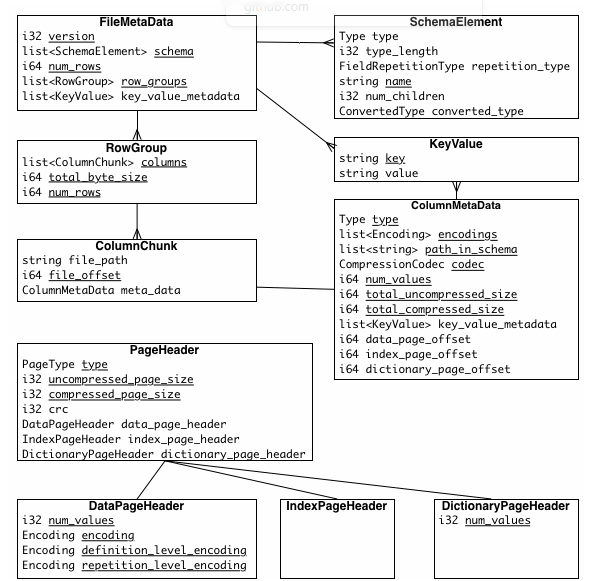
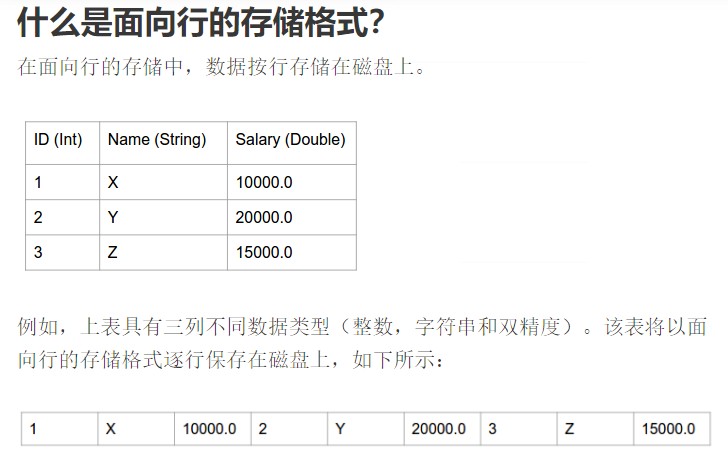
BYTE\_ARRAY: arbitrarily long byte arrays.

## Nested Encoding

Parquet uses the Dremel encoding with definition and repetition levels. Definition levels specify how many optional fields in the path for the column are defined. Repetition levels specify at what repeated field in the path has the value repeated. The max definition and repetition levels can be computed from the schema (i.e. how much nesting there is).

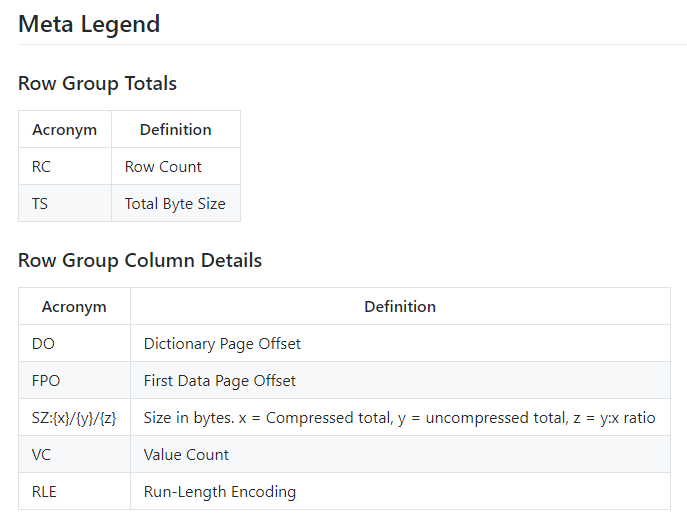
Parquet使用具有定义和重复级别的Dremel编码。定义级别指定在列的路径中定义了多少个可选字段。重复级别指定路径中哪个重复字段的值重复。最大定义和重复级别可以从模式中计算（即有多少嵌套）。

较大的行组允许更大的列块，从而可以执行更大的顺序IO。较大的组也需要在写入路径（或两次写入）中有更多的缓冲区。我们建议大行组（512MB-1GB）。因为可能需要读取整个行组，所以我们希望它完全适合一个HDFS块。因此，HDFS块大小也应设置为更大。优化的读取设置是：1GB行组、1GB HDFS块大小、每个HDFS文件1个HDFS块。







Head cat

Block（hdfs Block）：这意味着hdfs中的一个块，其含义对于描述这个文件格式没有改变。该文件格式设计用于在hdfs之上很好地工作。

file：必须包含文件元数据的hdfs文件。它不需要实际包含数据。

Row group：将数据分为行的逻辑水平分区。对于行组没有保证的物理结构。行组由数据集中每个列的列块组成。

Column chunk：特定列的数据块。它们位于特定的行组中，并保证在文件中是连续的。

page：列块被分成多个页面。页面在概念上是一个不可分割的单元（就压缩和编码而言）。在一个列块中可以有多个页面类型交错。

在层次结构上，文件由一个或多个行组组成。一个行组每列只包含一个列块。列块包含一个或多个页面。

列式存储布局（比如 Parquet）可以加速查询，因为它只检查所有需要的列并对它们的值执行计算，因此只读取一个数据文件或表的小部分数据。Parquet 还支持灵活的压缩选项，因此可以显著减少磁盘上的存储。

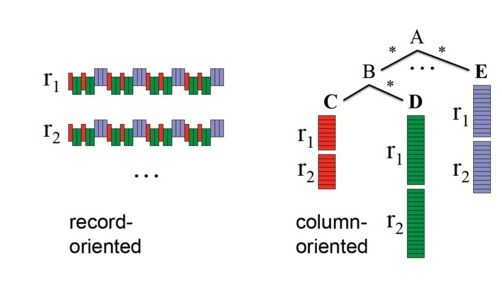
如果您在 HDFS 上拥有基于文本的数据文件或表，而且正在使用 Spark SQL 对它们执行查询，那么强烈推荐将文本数据文件转换为 Parquet 数据文件，以实现性能和存储收益。当然，转换需要时间，但查询性能的提升在某些情况下可能达到 30 倍或更高，存储的节省可高达 75%！

查询性能

Parquet 文件是自描述性的，所以保留了模式

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 Delta Encoding）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。

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？





record shredding and assembly algorithm

记录分解与组装算法

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

## 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

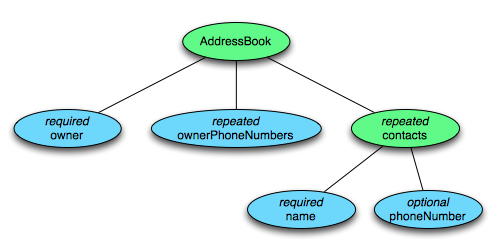
|  |
| --- |
| message AddressBook { |
|  | required string owner; |
|  | repeated string ownerPhoneNumbers; |
|  | repeated group contacts { |
|  | required string name; |
|  | optional string phoneNumber; |
|  | } |
|  | } |



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。

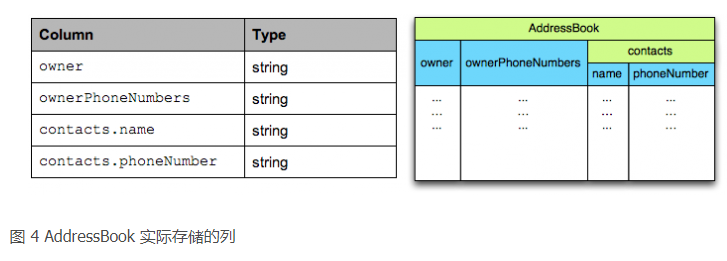




## Parquet 文件的存储格式

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



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。

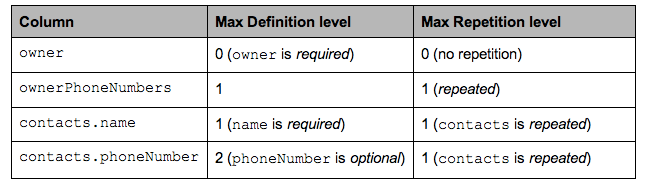
Repetition Level 和 Definition Level 可以说是为了支持嵌套类型而设计的，但是它同样适用于简单数据类型。在 Parquet 中我们只需定义和存储 schema 的叶子节点所在列的 Repetition Level 和 Definition Level。

### Definition Level

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）。

### Repetition Level

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。



列裁剪与谓词下推：列裁剪，意思是只读取需要的列，实现高效的列扫描，减少 IO 操作；谓词下推，可以过滤掉不符合条件的数据，只读取需要的数据，进一步减少 IO 操作。

更高效的压缩与编码：因为同一列的数据类型相同，所以可以针对不同列使用更合适的压缩与编码方式，降低磁盘存储空间。

值得说明的是文件 header 部分中的 Magic Number，它的作用主要是为了做文件校验，验证文件是否是一个Parquet文件。

****1、通过 Hive 创建 Parquet 表****

****·**** 标准建表语句。举一个最简单的建表示例，如下：

create table t1 (id int) stored as parquet;

****·**** 通过 TextFile 表创建 Parquet 表。这是离线数仓中的常规操作，假设已经准备好了一张 TextFile 表 catalog\_sales，在 hive shell 命令行中演示如下操作：

-- 创建parquet表（不压缩）

> drop table if exists catalog\_sales\_par;

> create table catalog\_sales\_par stored as parquet as select \* from catalog\_sales;

-- 设置parquet为snappy压缩

> set parquet.compression=SNAPPY;

> create table catalog\_sales\_par\_snappy stored as parquet as select \* from catalog\_sales;

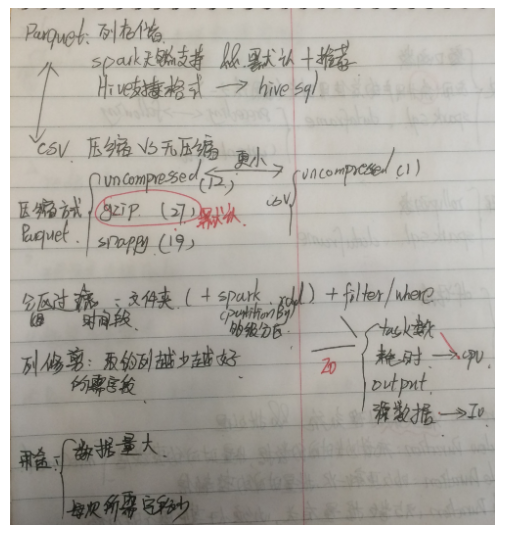
-- 设置parquet块大小

> set parquet.block.size=268435456;

> create table catalog\_sales\_par\_snappy2 stored as parquet as select \* from catalog\_sales;

****查看 Parquet 文件 Schema****

Parquet也是Spark SQL 的默认数据源，可通过参数spark.sql.sources.default 进行配置。



 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。

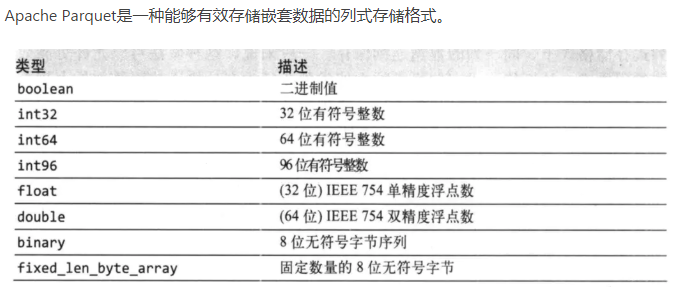
## **性能**

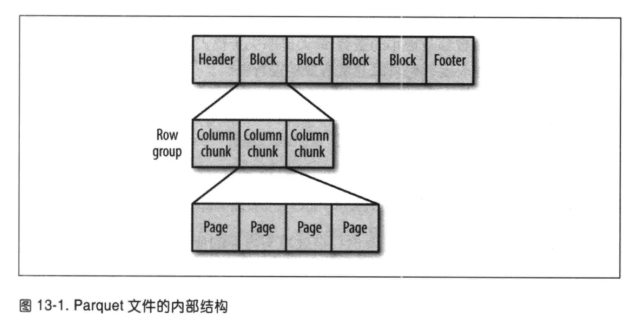
Parquet 列式存储带来的性能上的提高在业内已经得到了充分的认可，特别是当你们的表非常宽（column 非常多）的时候，Parquet 无论在资源利用率还是性能上都优势明显。具体的性能指标详见参考文档。

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。



Parquet文件由一个文件头（header），一个或多个紧随其后的文件块（block），以及一个用于结尾的文件尾（footer）构成。文件头仅包含  
Parquet文件的每个文件块负责存储一个行组，行组由列块组成，且一个列块负责存储一列数据。每个列块中的的数据以页为单位。

作者：BigBigFlower  
链接：https://www.jianshu.com/p/76b7776ed567  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

A、公司当时已经上线spark 集群，而spark天然支持parquet，并为其推荐的存储格式(默认存储为parquet)。

B、hive 支持parquet格式存储，如果以后使用hiveql 进行查询，也完全兼容。

Drill可识别

****3.2.1分区过滤****

parquet结合spark，可以完美的实现支持分区过滤。如，需要某个产品某段时间的数据，则hdfs只取这个文件夹。

spark sql、rdd 等的filter、where关键字均能达到分区过滤的效果。

使用spark的partitionBy 可以实现分区，若传入多个参数，则创建多级分区。第一个字段作为一级分区，第二个字段作为2级分区。。。。。

****3.2.2 列修剪****

列修剪：其实说简单点就是我们要取回的那些列的数据。

当取得列越少，速度越快。当取所有列的数据时，比如我们的120列数据，这时效率将极低。同时，也就失去了使用parquet的意义。

* 当我们取出所有记录时，三种压缩方式耗时差别不大。耗时大概7分钟。
* 当我们仅取出某一天时，parquet的分区过滤优势便显示出来。仅为6分之一左右。貌似当时全量为七八天左右吧。
* 当我们仅取某一天的一个字段时，时间将再次缩短。这时，硬盘将只扫描该列所在rowgroup的柱面。大大节省IO。如有兴趣，可以参考[深入分析Parquet列式存储格式](http://www.infoq.com/cn/articles/in-depth-analysis-of-parquet-column-storage-format)
* ****分区过滤和列修剪可以帮助我们大幅节省磁盘IO。以减轻对服务器的压力。****
* ****如果你的数据字段非常多，但实际应用中，每个业务仅读取其中少量字段，parquet将是一个非常好的选择。****