每个 Spark 开发者都应该知道的开发技巧

尽量不要把数据 collect 到 Driver 端

如果你的 RDD/DataFrame 非常大，drive 端的内存无法放下所有的数据时，千万别这么做

```
data = df.collect()
```

Collect 函数会尝试将 RDD/DataFrame 中所有的数据复制到 driver 端，这时候肯定会导致 driver 端的内存溢出，然后进程出现 crash。

相反，我们可以通过调用 take 或 takeSample，或者 filter 或采样来减少 RDD/DataFrame 返回的元素。

同样，除非你确定你的数据集足够小，而且 driver 端的内存足够放得下这些元素，否则下面的函数也需要小心使用：

- countByKey
- countByValue
- collectAsMap

广播比较大的变量

Spark 官方文档对广播的描述：

> Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

使用 SparkContext 中的广播变量可以显著减少每个序列化任务的大小，以及在集群上运行任务的成本。如果你的任务使用来自 Driver 程序的一个大对象（例如一个静态搜索表，一个大的 list），请考虑把它转换成一个广播变量。

如果你不这样做，那么相同的变量将发送到 Executor 中的每个分区里面。广播变量允许程序员在每台机器上以反序列化的形式缓存只读变量，而不是随任务发送该变量的副本。

Spark在应用程序的 Driver 上打印每个任务的序列化大小，因此您可以查看这个，看看您的任务是否太大；一般来说，超过20KB的任务都是值得优化的。下面就是使用广播的例子：

```
import contextlib

@contextlib.contextmanager
def persisted_broadcast(sc, variable):
    variable_b = sc.broadcast(variable)
    yield variable_b
    variable_b.unpersist()

with persisted_broadcast(sc, my_data) as broadcasted_data:
    pass
```

值得注意的是，Spark 有一个名为 ContextCleaner 的类，它会定期运行来删除没有使用的广播变量。

使用最合适的文件格式

为了提高程序的运行效率，我们需要选择合适的文件格式。根据 Spark
作业的特定应用程序或单个功能，格式可能会有所不同。

许多格式都有自己的细节，例如 Avro
具有容易的序列化/反序列化，这允许有效地和其他摄入程序进行集成。同时，Parquet
允许您在选择特定列时有效地工作，并且可以有效地存储中间文件。但是 parquet
文件是不可变化的，修改需要覆盖整个数据集，然而，Avro 文件可以轻松应对频繁的模式更改。

在读取 CSV 和 JSON 文件时，通过指定模式（而不是使用推理机制）可以获得更好的性能——指定模式可以减少错误，建议用于生产代码。以下就是一个例子：

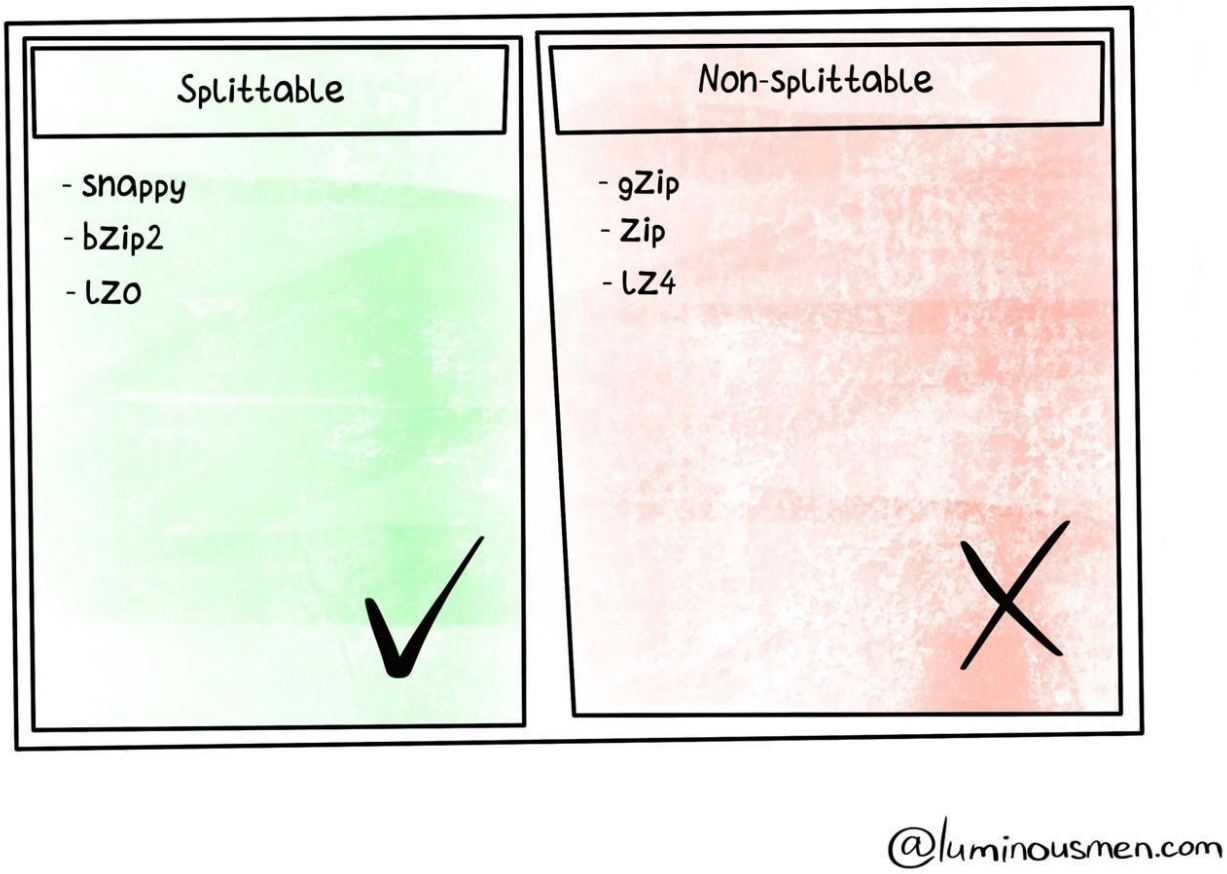
```
from pyspark.sql.types import (StructType, StructField,
   DoubleType, IntegerType, StringType)

schema = StructType([
   StructField('A', IntegerType(), nullable=False),
   StructField('B', DoubleType(), nullable=False),
   StructField('C', StringType(), nullable=False)
])

df = sc.read.csv('/some/input/file.csv', inferSchema=False)
```

为文件选择合适的编码

我们处理的文件类型可以分为两种：

- 可分割的，比如 lzo, bzip2, snappy
- 不可分割的，比如 gzip, zip, lz4

出于讨论的目的，“可分割文件”意味着它们可以以分布式的方式并行处理，而不是在单个机器上(不可分割)。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

不要使用 zip/gzip 格式的大型源文件，它们是不可分割的。不可能使用 Spark 并行地读取这些文件。首先，Spark 需要在一个 Executor 上下载整个文件，在一个内核上解压，然后将分区重新分发到集群节点。可以想象，这将成为分布式处理中的一个巨大瓶颈。如果文件存储在 HDFS 上，那么应该在下载它们之前解压缩它们。

Bzip2 文件也有类似的问题。尽管它们是可拆分的，但它们被压缩得非常少，因此可能分布得很差。如果压缩时间和 CPU 负载没有限制，则使用 Bzip2，例如一次性打包大量数据。

毕竟，我们看到未压缩文件的性能明显优于压缩文件。这是因为未压缩的文件是受 I/O 限制的，而压缩的文件是受 CPU 限制的，但是这里的 I/O 已经足够好了。Apache Spark 很好地支持这一点，但是其他库和数据仓库可能不支持。

当输入和输出值类型不同时，避免使用 reduceByKey

如果出于任何原因您有基于 rdd 的作业，请明智地使用 reduceByKey 操作。

考虑以下的例子：

```
rdd.map(lambda p: (p[0], {p[1]})) ₩
   .reduceByKey(lambda x, y: x | y) ₩
   .collect()
```

请注意，输入值是字符串，输出值是集合。map
操作创建了许多临时的小对象。处理这种情况的更好方法是使用 aggregateByKey:

```
def merge_vals(xs, x):
   xs.add(x)
   return xs

def combine(xs, ys):
   return xs | ys

rdd.aggregateByKey(set(), merge_vals, combine).collect()
```

当不需要返回确切的行数时，不要使用 count

当我们不需要返回确切的行数使用以下的方法：

```
df = sqlContext.read().json(...);
if not len(df.take(1)):
   ...
```

or

```
if df.rdd.isEmpty():
   ...
```

而不是使用以下的方法：

```
if not df.count():
   ...
```

如果是 RDD 的话，可以使用 isEmpty()。