在Python中执行SQL：200G数据分析秒级出结果

在Python数据分析领域，除了常用的DataFrame分析之外，SQL分析依靠其简单易用、语句自动优化、易于处理结构化数据等特点在数据分析领域中也很受欢迎，这里为大家介绍一款单机SQL分析引擎：Blackhole-SQL，

这款引擎在分析200G的数据时能够做到秒级出结果，并且十分简单易用，接下来我们来认识一下这款Python数据分析领域里的SQL利器吧。

1.什么是Blackhole-SQL

Blackhole-SQL是百度自研的科学计算Python库Blackhole中的SQL子模块，主要是为了那些喜欢使用SQL做数据分析的小伙伴而量身打造的一款单机引擎，能够处理TB级别的数据，兼容标准SQL语法，提供了丰富的数

据聚合算子，具备强大的数据分析、统计、聚合能力，同时提供了数据导入/导出的能力。

2.安装Blackhole-SQL

安装Blackhole-SQL非常简单，因为SQL模块是内嵌在Blackhole中的，安装Blackhole即可：

pip install https://conda-codelab.bj.bcebos.com/install-packages/blackhole/v1.0.1/blackhole-v1.0.1-cp37-cp37m-linux_x86_64.tar.gz -i https://mirrors.aliyun.com/pypi/simple

因为Blackhole还未开源，因此不能直接pip + 包名安装，后续开源了可以。

3.开始体验之旅

接下来给大家展示一下Blackhole-SQL的用法，废话不多说，直接上代码：

```
from blackhole.sql.context import Context


context = Context()
data_schema = '''user String,age Int32,fees Float32'''
```

```
data_format = 'CSV'
data_path = '/tmp/data/256G/user.csv'

ds = context.load(data_path).schema(data_schema).format(data_format).primary_keys('user').create_table('test')
```

上面这段代码展示了如何引入Blackhole-SQL、确定数据源的schema以及建表的过程

i.首先导入Blackhole-SQL的Context，这个Context是与SQL引擎交互的上下文，是执行SQL查询的入口

ii.创建好Context类型的对象context

iii.data_schema指定了这个数据集的数据schema，这个例子里的数据集有3列，列名分别是：user，age，fees，在列名后跟着该列的数据类型，每列定义之间用","分隔

iv.通过context使用SQL引擎的数据导入load接口导入外部的数据，通过schema方法指定schema信息，format方法指定数据格式，目前的格式支持csv，parquet，orc，json，primary_keys方法指定了数据集的主键，

这里的主键与传统意义上的主键意义不同，这里的主键只是用来创建索引用的，也可以不知道，但是指定了之后如果聚合排序发生在索引列上可以显著提升查询速度，create_table方法是指定创建的表名，在Blackhole-SQL

中，和传统的数据库表一样，针对数据的dml操作都是在表上进行的，执行完create_table之后，原始数据就被导入到Blackhole-SQL引擎中了，并且是结构化保存的，接下来就可以针对表来进行操作了。

(1)查询表中的数据

```
context.sql('select * from test limit 5').show()
```

执行上面的代码会有如下输出：

```
user         age         fees
Alice         23         7854
Bob          43         28314
Herry         28         9845
Jack          31         14523
Tom          32         13205
```

看，是不是和使用传统数据库比如MySQL一样呢！

当然，也可以设置条件来进行查询：

```
context.sql('select * from test where fees < 10000.0 limit 2').show()
```

输出：

```
user        age        fees
Alice        23        7854
Herry        28        9845
```

语法和标准SQL一样。

(2)聚合与统计

Blackhole-SQL提供了很多丰富的聚合统计函数，常用的比如有:min，max，avg，sum等，例如执行下列代码：

context.sql('select sum(fees) from test').show()

context.sql('select avg(age) from test').show()

context.sql('select min(age) from test').show()

context.sql('select max(fees) from test').show()

分别会有如下输出：

```
sum(fees)
73741
```

```
avg(age)
31.4
```

```
min(age)
23
```

```
max(fees)
28314
```

(3)排序

排序的使用方式和标准SQL语法一样：

context.sql('select * from test order by age desc limit 5').show()

输出：

```
user        age        fees
Bob        43        28314
Tom        32        13205
Jack        31        14523
Herry        28        9845
Alice        23        7854
```

4.与Spark SQL比较

在笔者的笔记本上，当数据膨胀到50G，约32亿行时，执行上面的操作都不到1秒就完成了，这速度是不是很犀利呢，当然有的童鞋可能会质疑，上面的数据集和查询语句都太简单了，面对一些大宽表和复杂的聚合查询时

Blackhole-SQL的表现是不是也很优秀呢，下面我们使用业界公人的最权威的TPC-DS benmark数据集来测试Blackhole-SQL性能，使用的是TPC-DS数据集中数据量最大的表store_sales，数据量从1G到256G，表schema如下所示：

```
ss_sold_date_sk Int32,
ss_sold_time_sk Int32,
ss_item_sk Int32,
ss_customer_sk Int32,
ss_cdemo_sk Int32,
ss_hdemo_sk Int32,
ss_addr_sk Int32,
ss_store_sk Int32,
ss_promo_sk Int32,
ss_ticket_number Int32,
ss_quantity Int32,
ss_wholesale_cost Decimal(7,2),
ss_list_price Decimal(7,2),
ss_sales_price Decimal(7,2),
ss_ext_discount_amt Decimal(7,2),
ss_ext_sales_price Decimal(7,2),
ss_ext_wholesale_cost Decimal(7,2),
ss_ext_list_price Decimal(7,2),
ss_ext_tax Decimal(7,2),
ss_coupon_amt Decimal(7,2),
ss_net_paid Decimal(7,2),
ss_net_paid_inc_tax Decimal(7,2),
ss_net_profit Decimal(7,2)
```

```
    "query1": ["select avg(ss_item_sk) from store_sales"],
    "query2": ["select ss_sold_date_sk,count(*) as cnt from store_sales group by ss_sold_date_sk "
               "order by cnt desc,ss_sold_date_sk limit 10"],
    "query3": ["select ss_sold_date_sk,avg(ss_item_sk) as cnt from store_sales "
               "group by ss_sold_date_sk order by cnt desc,ss_sold_date_sk limit 10"],
    "query4": ["select ss_item_sk,count(*) from store_sales group by "
               "ss_item_sk having count(*)>1 limit 10"],
    "query5": ["select sum(ss_item_sk) from store_sales"],
```

为了有一个对比，我们同时也测试了Spark SQL单机版的性能，以下是Blackhole-SQL在6个query下的性能表现，单位是秒

1g	8g	16g	32g	64g	128g	256g
0.01	0.04	0.1	0.14	0.16	0.35	0.8
0.02	0.16	0.26	0.51	1.03	2.07	3.99
0.03	0.21	0.45	0.87	1.65	3.03	7.58
0.02	0.22	0.26	0.58	1.05	2.06	4.26
0.01	0.06	0.04	0.09	0.16	0.37	0.71

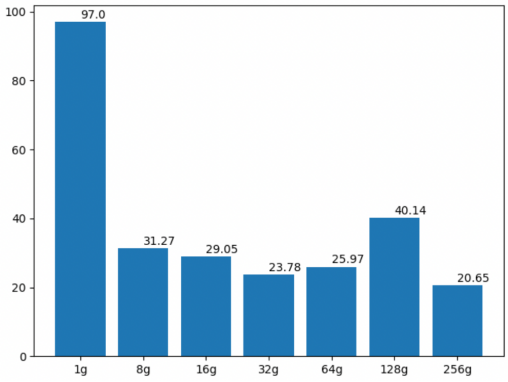
所有query都在几秒以内返回结果了，有的甚至还不到1秒，这性能是不是非常棒？我们可以对比一下Spark SQL的性能：

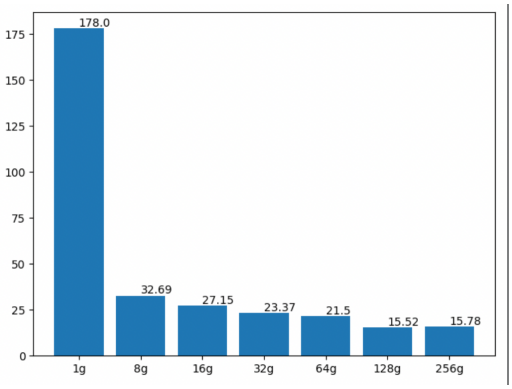
1g	8g	16g	32g	64g	128g	256g
4.53	5.7	9.07	12.52	20.1	41.09	66.1
8.6	8.99	11	15.76	25.05	38.64	74.05
7.03	9.61	11.98	15.87	26.55	40.26	73.72
6.61	9.29	10.3	17.18	26.01	39.75	75.09
4.12	5.98	8.17	12.72	20.18	35.74	64.54

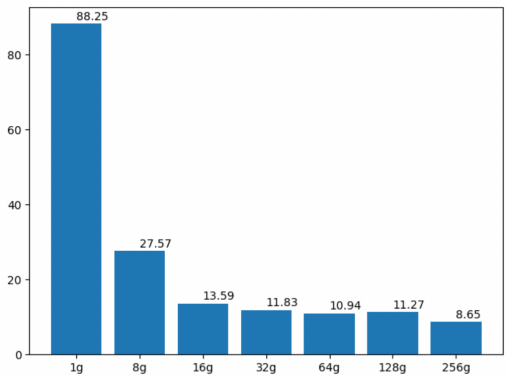
通过对比，Blackhole比Spark SQL单机版快了不是一点半点，如果这个不直观我们直接上加速比的图：

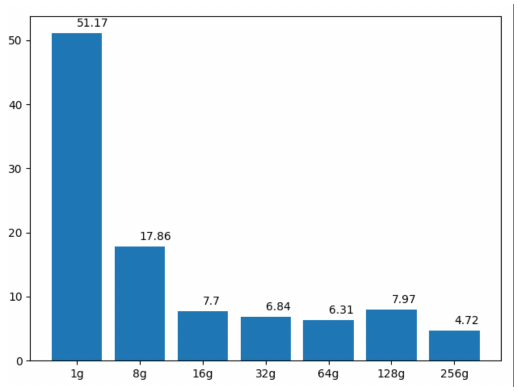
query1: query2: query3: query4:

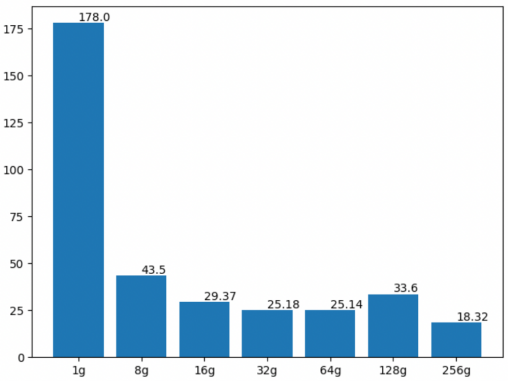
query5:











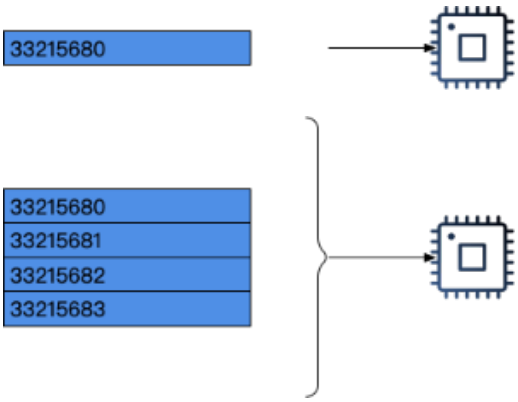
通过上面的对比，我们可以看到Blackhole-SQL对Spark SQL的加速比基本都超过了10，也就是意味着一个数量级

5.为什么这么快

Blackhole-SQL之所以这么快得益于它有一个采用C++编写的SQL执行引擎，这个引擎使用了大量的黑科技来提速，就像一辆使用特殊材料和工艺打造的小跑车一样，总的来说有如下几个方面：

（1）向量化执行的方式

向量化执行想必大家都不陌生，例如利用现代CPU的SSE指令集，可以让CPU执行SIMD指令，一个指令周期内可以处理多条数据，这样显然可以提供系统的吞吐，如下图所示：



普通方式执行　　SIMD方式执行

（2）列式存储引擎

仅从存储系统中读取必要的列数据，无用列不读取，速度非常快。

（3）稀疏索引

查询时能够裁剪不必要的记录读取，提高查询性能。

（4）数据压缩

因为磁盘IO性能是远低于内存操作的，如果将磁盘上的数据压缩到原始数据的十分之一，借助索引，一次读取就能覆盖到更多的数据记录，从而间接提升查询性能。

（5）动态代码生成方式

借助LLVM框架的编译能力，将执行物理计划时生成满足符合流水线形式的执行代码，从而利用现代CPU深度流水线和错序执行能力，最大程度上提高数据处理性能

6.结束语

以上就是Blackhole-SQL的简单介绍，这个Python库最大的亮点就是极致的SQL查询性能，最近刚发布了1.0版，对性能有高要求的小伙伴们不妨下载下来试用一下。