Sparak、Impala、Hive 基于 TPC-DS 对比测试

目录

一、测试概要

分别针对三种查询引擎基于 TPC-DS 数据集及三组 SQL 做下对比测试，大致了解 Spark SQL 的性能水平，并横向与 Hive 及 Impala 做下对比，同时对 Sql 兼容性做下总结。

二、测试环境

1、硬件环境

名称	节点配置	数量	安装服务	备注
UDH 集群	CPU 8*CORE Memory 16G Disk 300G	4	HDFS YARN HIVE ZOOKEEPER IMPALA SPARK	机器是 UAP 云平台虚拟机

2、软件环境

UDH1.2.0 (SPARK:1.3.0+cdh5.4.3 IMPALA:2.2.0+cdh5.4.3 HIVE:1.1.0+cdh5.4.3)

三、测试方法

1、Spark 测试方法

采用 Spark on hive 方式，Spark 采用 stand-alone 方式调度。

2、Impala 测试方法

大表数据格式采用 Parquet，小表采用文本。

通过 Impala shell 直接提交查询 SQL。

Impala 运行方式为 stand-alone。

3、Hive 测试方法

数据格式采用文本。

通过 hive shell 直接提交查询 SQL。

4、测试 SQL

见附录

5、数据集

基于 TPC-DS 生成 10G 的数据集，具体如下：

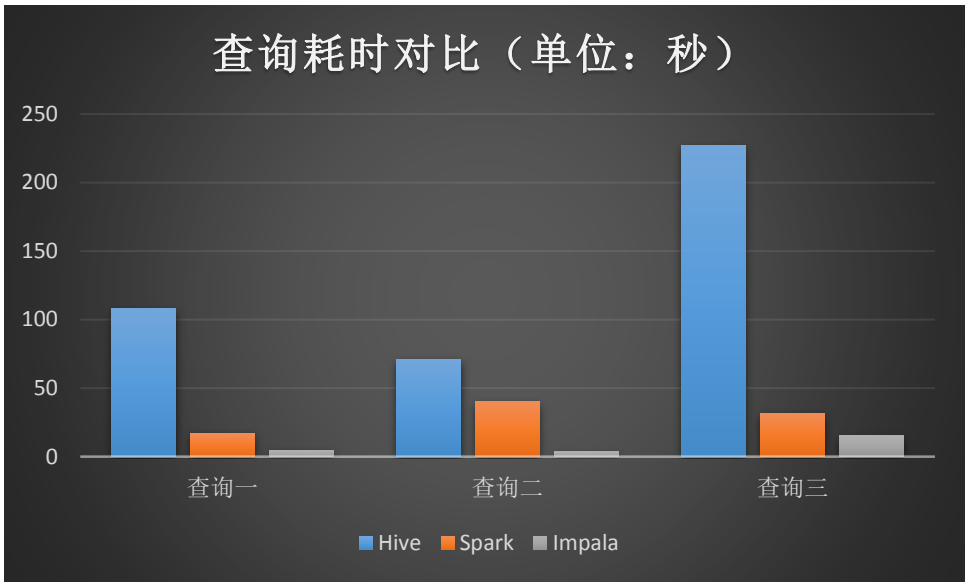
表名	大小	行数
catalog_page	1.6M	12000
catalog_sales	2.9G	14401261
customer	64M	500000
customer_address	27M	250000
customer_demographics	77M	1920800
date_dim	9.9M	73049
income_band	328B	20
household_demographics	149K	7200
inventory	2.6G	133110000
item	28M	102000
ship_mode	1.1K	20
reason	1.7K	45
promotion	61K	500
store	27K	102
store_sales	3.8G	28800991
store_returns	323M	2875432
time_dim	4.9M	86400
warehouse	1.2K	10

四、测试结果

测试数据：

	查询耗时（单位：秒）		
	查询一	查询二	查询三
Hive	108	71	227
Spark	17	40	31.7
Impala	4.5	3.6	15.3

统计图：



五、SQL 兼容性

Spark SQL、Impala 语法方面都是向 Hive 看齐。

Spark SQL 几乎完全兼容 HIVE SQL 语法，只是 HIVE 特有的一些优化参数及极少用语法不支持。

Impala SQL 与 HIVE SQL 高度兼容，但不局限于 HIVE 已有的查询 SQL，同时 Impala 还支持 insert into values … 。

三种查询引擎在窗口函数支持上有限，另外，也不支持 where 条件中含有多个子查询的 or 操作。

六、小结

查询性能上 HIVE<SPARK<IMPALA

七、附录

查询一：

```
select
c_last_name,c_first_name,ca_city ,bought_city ,ss_ticket_number,amt,profit from
   (select                ss_ticket_number,ss_customer_sk                ,ca_city
bought_city,sum(ss_coupon_amt) amt ,sum(ss_net_profit) profit
    from store_sales_p,date_dim,store,household_demographics,customer_address
    where store_sales_p.ss_sold_date_sk = date_dim.d_date_sk
    and store_sales_p.ss_store_sk = store.s_store_sk
    and store_sales_P.ss_hdemo_sk = household_demographics.hd_demo_sk
    and store_sales_p.ss_addr_sk = customer_address.ca_address_sk
    and        (household_demographics.hd_dep_count        =        6        or
household_demographics.hd_vehicle_count= 2)
    and date_dim.d_dow in (6,0)
    and date_dim.d_year in (1999,2000,2001)
    and        store.s_city        in        ('Midway','Oak        Grove','Pleasant
Hill','Fairview','Riverside')
    group        by        ss_ticket_number,ss_customer_sk,ss_addr_sk,ca_city)
dn,customer,customer_address current_addr
    where ss_customer_sk = c_customer_sk
      and customer.c_current_addr_sk = current_addr.ca_address_sk
      and current_addr.ca_city <> bought_city
  order by c_last_name,c_first_name,ca_city,bought_city,ss_ticket_number
  limit 100;
```

查询二：

```
select  * from(select w_warehouse_name,i_item_id
           ,sum(case when (cast(d_date as TIMESTAMP) < cast ('2001-05-04' as
TIMESTAMP)) then inv_quantity_on_hand else 0 end) as inv_before
           ,sum(case when (cast(d_date as TIMESTAMP) >= cast ('2001-05-04' as
TIMESTAMP)) then inv_quantity_on_hand  else 0 end) as inv_after
   from inventory_p ,warehouse,item ,date_dim_p
   where i_current_price between 0.99 and 1.49
     and i_item_sk          = inv_item_sk
     and inv_warehouse_sk   = w_warehouse_sk
     and inv_date_sk    = d_date_sk
     and d_date between cast ('2001-04-04' as TIMESTAMP) and cast ('2001-06-04'
as TIMESTAMP)
   group by w_warehouse_name, i_item_id) x
 where (case when inv_before > 0  then inv_after / inv_before  else null end)
between 2.0/3.0 and 3.0/2.0
```

```
 order by w_warehouse_name ,i_item_id
 limit 100;
```
查询三：
```
select  i_item_id,i_item_desc ,s_state
       ,count(ss_quantity) as store_sales_quantitycount
       ,avg(ss_quantity) as store_sales_quantityave
       ,stddev_samp(ss_quantity) as store_sales_quantitystdev
       ,stddev_samp(ss_quantity)/avg(ss_quantity) as store_sales_quantitycov
       ,count(sr_return_quantity) as store_returns_quantitycount
       ,avg(sr_return_quantity) as store_returns_quantityave
       ,stddev_samp(sr_return_quantity) as store_returns_quantitystdev
       ,stddev_samp(sr_return_quantity)/avg(sr_return_quantity)              as
store_returns_quantitycov
       ,count(cs_quantity) as catalog_sales_quantitycount ,avg(cs_quantity) as
catalog_sales_quantityave
       ,stddev_samp(cs_quantity)/avg(cs_quantity)                            as
catalog_sales_quantitystdev
       ,stddev_samp(cs_quantity)/avg(cs_quantity) as catalog_sales_quantitycov
 from  store_sales_p,store_returns  ,catalog_sales_p  ,date_dim  d1  ,date_dim
d2 ,date_dim d3,store ,item
 where d1.d_quarter_name = '2000Q1'
   and d1.d_date_sk = ss_sold_date_sk
   and i_item_sk = ss_item_sk
   and s_store_sk = ss_store_sk
   and ss_customer_sk = sr_customer_sk
   and ss_item_sk = sr_item_sk
   and ss_ticket_number = sr_ticket_number
   and sr_returned_date_sk = d2.d_date_sk
   and d2.d_quarter_name in ('2000Q1','2000Q2','2000Q3')
   and sr_customer_sk = cs_bill_customer_sk
   and sr_item_sk = cs_item_sk
   and cs_sold_date_sk = d3.d_date_sk
   and d3.d_quarter_name in ('2000Q1','2000Q2','2000Q3')
 group by i_item_id,i_item_desc,s_state
 order by i_item_id,i_item_desc,s_state
limit 100;
```