TPCH ORC 100G DorisDB & Presto

hw & data & config

hardware

data: 100GB, ORC格式，snappy/lz4压缩

2 worker nodes: 8cores, 16vcpu, 64GB

这边100G是snappy/lz4格式压缩，然后数据在高效云盘500GB（5000IOPS）上，测试机型是ecs.g6.4xlarge （8 cores, 16 cpu, 64GB)

跑22个tpch sql, 运行5次，取后面3次平均值
https://doris.feishu.cn/sheets/shtcn4iueGhBthAdcN9DqYbWr0f?sheet=5c15fd

测试dorisdb的时候关闭presto, 测试presto的时候关闭dorisdb.

dorisdb config

Python

```python
GB = 1 << 30

def set_mem_limit(gb_unit):
    v = gb_unit * GB
    cursor.execute('set global exec_mem_limit = %d;' % v)

# set_mem_limit(16)
set_mem_limit(32)

cfg = [
    "parallel_fragment_exec_instance_num = %d" % concurrency,
    "enable_new_planner = true",
    "enable_new_planner_mock_tpch_statistic = true",
    "enable_new_planner_push_down_join_to_agg = true",
    "new_planner_max_transform_reorder_joins = 8",
    "new_planner_tpch_scale = 100"
]
```

native config

Native 配置格式如下:

- Repl = 2 (因为只有2台机器)
- Storage format = default
- Bucket number = 8

SQL

```sql
    ENGINE=OLAP
    DUPLICATE KEY(`n_nationkey`)
    COMMENT 'OLAP'
    DISTRIBUTED BY HASH(`n_nationkey`) BUCKETS 8
properties (
"replication_num" = "2",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);
```

presto config

Plain Text

```
node-scheduler.max-splits-per-node 100
task.concurrency 16
query.max-total-memory-per-node 40GB
query.max-memory-per-node 35GB
query.max-memory 80GB
```

jvm config

worker

Plain Text

```
-server
-Xmx60G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=128M -Xloggc:/mnt/disk1/log/presto/var/log/presto-worker-gc.log
```

coodinator

Plain Text

```
-server
-Xmx60G
-Dcom.sun.management.jmxremote.rmi.port=9081
-Djdk.attach.allowAttachSelf=true
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=128M -Xloggc:/mnt/disk1/log/presto/var/log/presto-worker-gc.log
```

测试结果(native普通配置, 2台机器)

优化点：

1. New planner
2. Date,Datetime 转换优化
3. Hdfs short circuit read

3Nodes Test 是 Planner 新旧对比 之前运行TPCH的时间

native导入有个问题是，我观察到一个机器上磁盘分布不均匀，不知道这个有什么解决办法

Apache

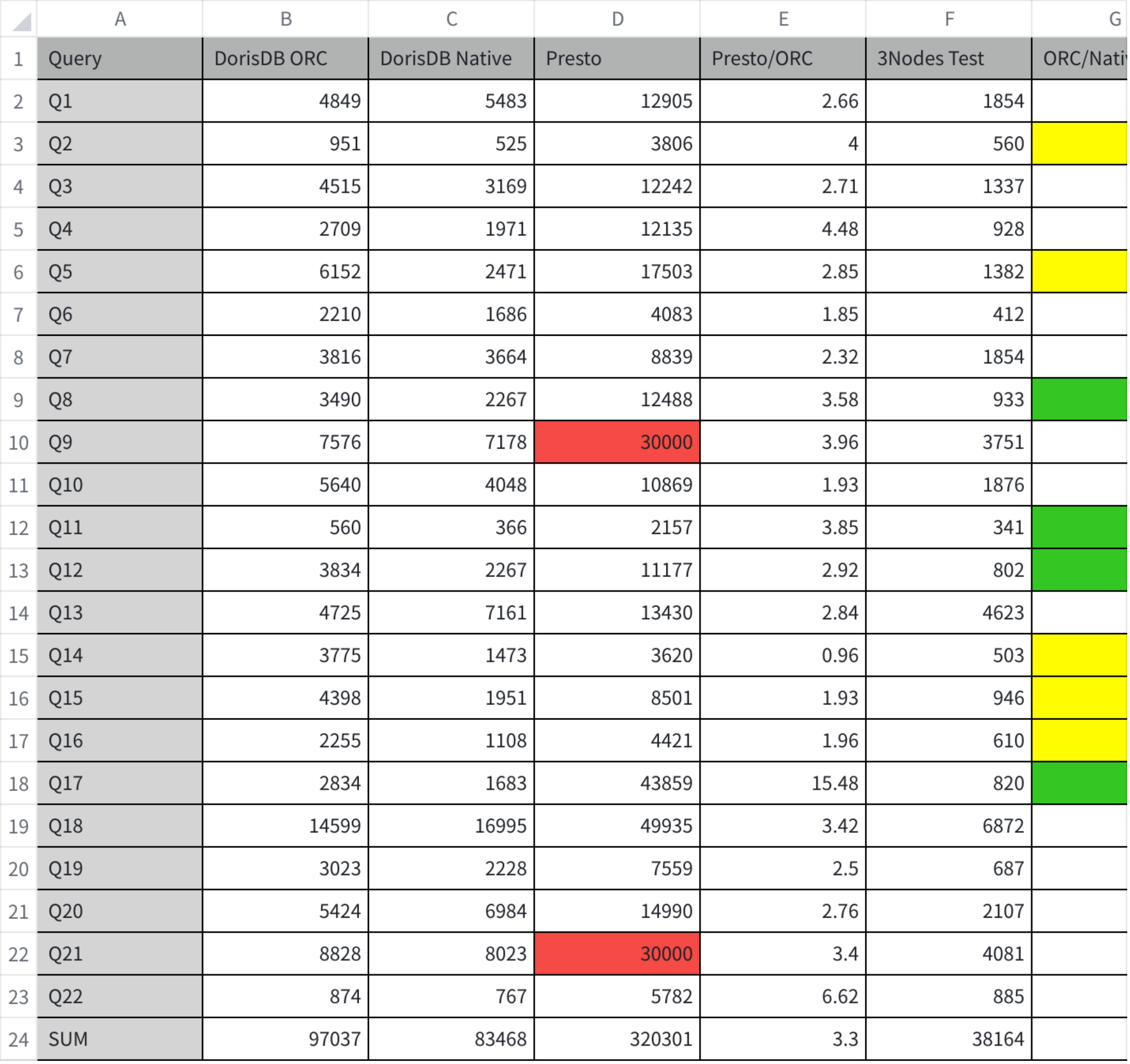
```
# 用了/mnt/disk[1,2,3,4]
storage_root_path = /mnt/disk1/dorisdb-storage;/mnt/disk2/dorisdb-storage;/mnt/disk3/dorisdb-storage;/mnt/disk4/dorisdb-stora

# 但是其中disk1/disk2使用很少
5.4G    /mnt/disk1/dorisdb-storage
620M    /mnt/disk2/dorisdb-storage
15G     /mnt/disk3/dorisdb-storage
16G     /mnt/disk4/dorisdb-storage

```

其中Presto在Q9,Q21上内存不够失败，我暂且使用30s代替，因为最后要统计整体的加速比。除去Q14这个case(0.96并且整体时间在3.6s左右，所以参考意义也不是很大）之外，所有情况都比presto要好。最好情况在Q17上15.48倍，整体加速是3.3倍（所有的query时间）

presto性能上来的原因可能和做了 `analyze table` 有关系 https://cwiki.apache.org/confluence/display/Hive/StatsDev#StatsDev-TableandPartitionStatistics 更新了对table, partition, column的统计信息

SQL

```sql
ANALYZE TABLE tpch_snappy.nation COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.customer COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.lineitem COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.orders COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.part COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.partsupp COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.region COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.supplier COMPUTE STATISTICS FOR COLUMNS;
ANALYZE TABLE tpch_snappy.nation COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.customer COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.lineitem COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.orders COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.part COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.partsupp COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.region COMPUTE STATISTICS;
ANALYZE TABLE tpch_snappy.supplier COMPUTE STATISTICS;
```



	A	B	C	D	E	F	G
1	Query	DorisDB ORC	DorisDB Native	Presto	Presto/ORC	3Nodes Test	ORC/Nati
2	Q1	4849	5483	12905	2.66	1854	
3	Q2	951	525	3806	4	560	🟨
4	Q3	4515	3169	12242	2.71	1337	
5	Q4	2709	1971	12135	4.48	928	
6	Q5	6152	2471	17503	2.85	1382	🟨
7	Q6	2210	1686	4083	1.85	412	
8	Q7	3816	3664	8839	2.32	1854	
9	Q8	3490	2267	12488	3.58	933	🟩
10	Q9	7576	7178	30000	3.96	3751	
11	Q10	5640	4048	10869	1.93	1876	
12	Q11	560	366	2157	3.85	341	🟩
13	Q12	3834	2267	11177	2.92	802	🟩
14	Q13	4725	7161	13430	2.84	4623	
15	Q14	3775	1473	3620	0.96	503	🟨
16	Q15	4398	1951	8501	1.93	946	🟨
17	Q16	2255	1108	4421	1.96	610	🟨
18	Q17	2834	1683	43859	15.48	820	🟩
19	Q18	14599	16995	49935	3.42	6872	
20	Q19	3023	2228	7559	2.5	687	
21	Q20	5424	6984	14990	2.76	2107	
22	Q21	8828	8023	30000	3.4	4081	
23	Q22	874	767	5782	6.62	885	
24	SUM	97037	83468	320301	3.3	38164	

和 native 的对比上也有些差距:

1. 最坏的几个cases (>1.8) Q2, Q5, Q14,Q15,Q16 (bucket shuffle原因)
2. 其次差距比较大(>1.5) Q8, Q11, Q12,Q17
    a. Q8,Q12 bucket shuffle
    b. Q11 计划相同, Scan 差距
    c. Q17 计划相同, Scan 差距

测试结果2(native优化配置, 3台机器)

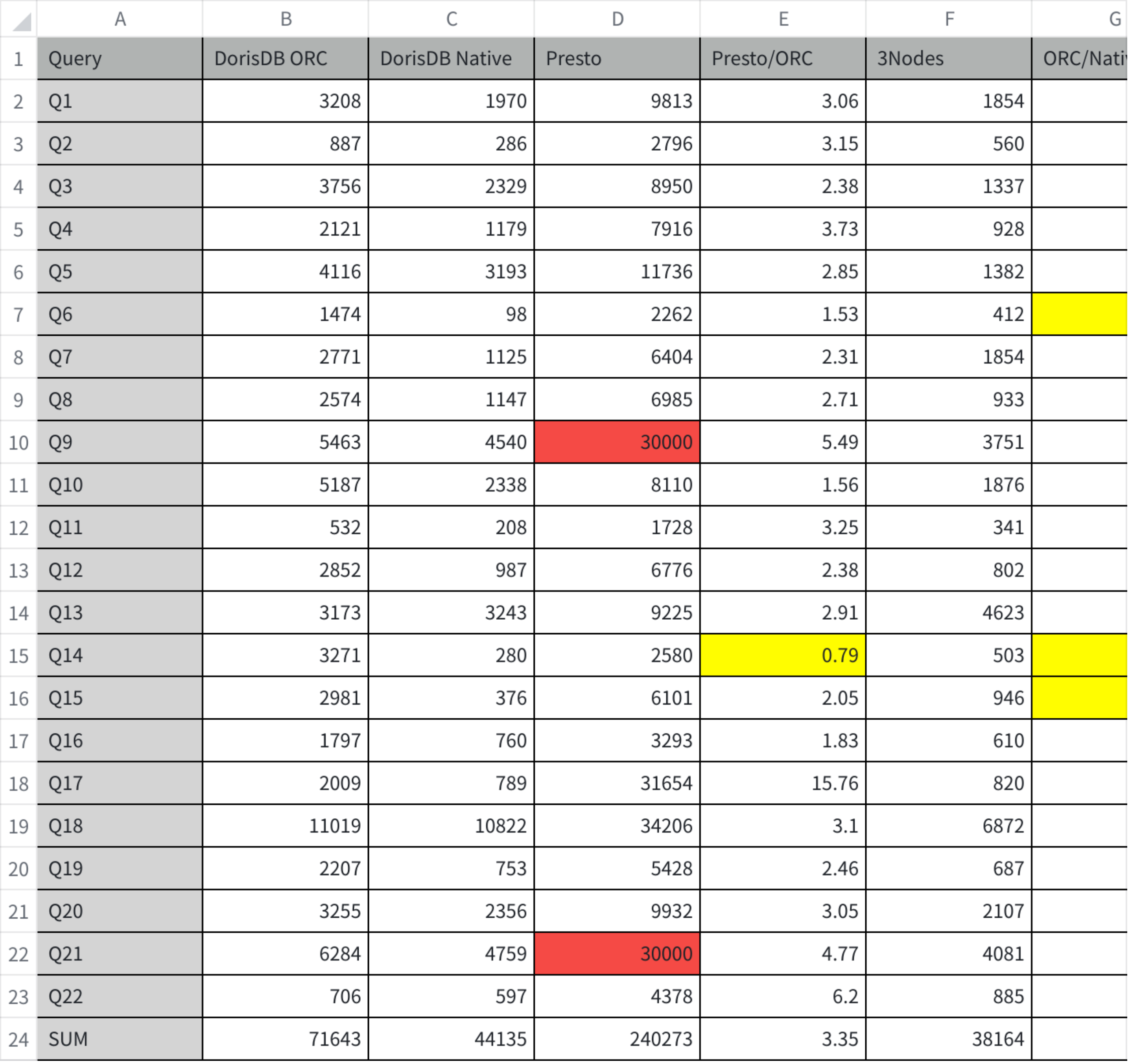
对hdfs做了replication factor调整为3，这样HDFS文件都可以short-circuit read.

优化之后的native配置增加了partition, 并且调整了bucket数量。具体配置看最后Appendix.

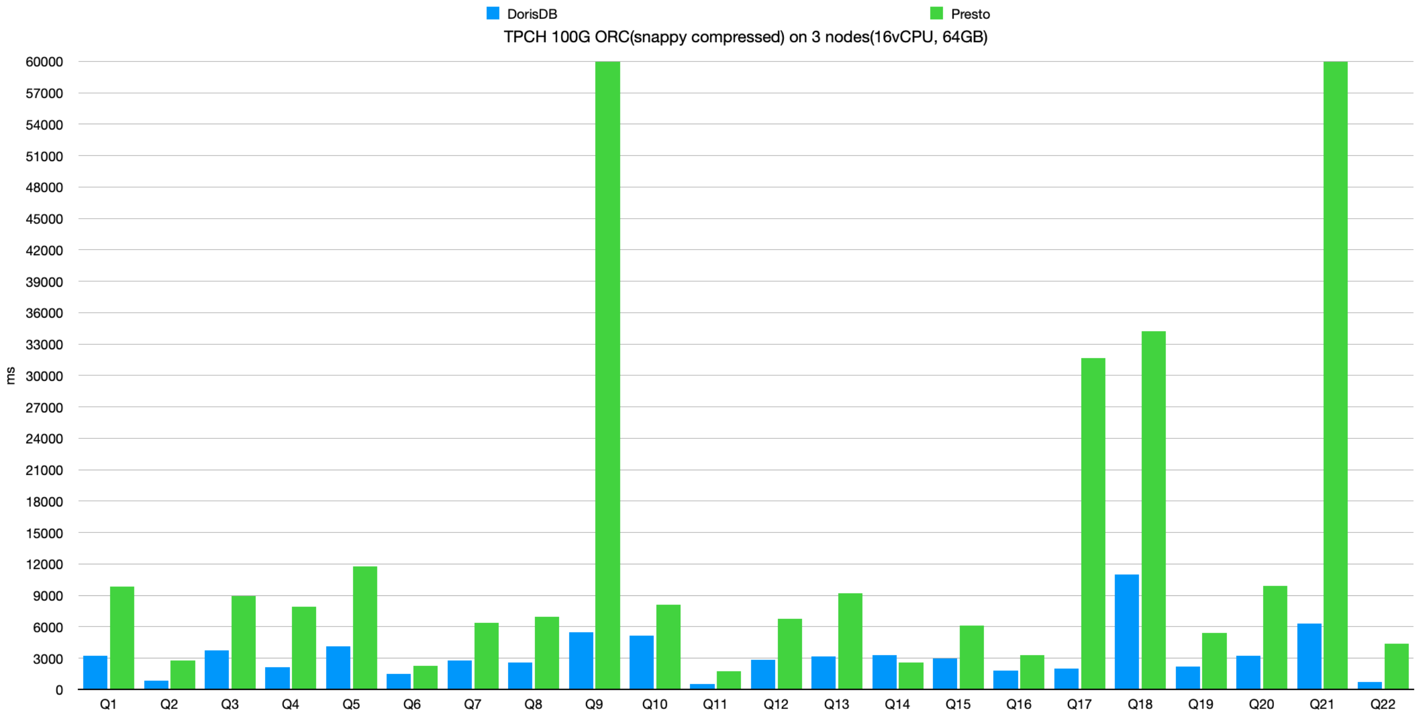
其中Presto在Q9,Q21上内存不够失败，我暂且使用30s代替，因为最后要统计整体的加速比。除去Q14这个case之外，所有情况都比presto要好。最好情况在Q17上15.76倍，整体加速是3.35倍（所有的query时间）

3Nodes 是 Planner 新旧对比 之前运行TPCH的时间. 这次Native和3Nodes相比有有好有坏，整体来说还是不如之前的3Nodes

优化之后的native基本完爆orc，尤其是在Q6, Q14, Q15这3个cases上。Q14，Q15在上面分析过是因为bucket shuffle. 至于Q6需要做分析。



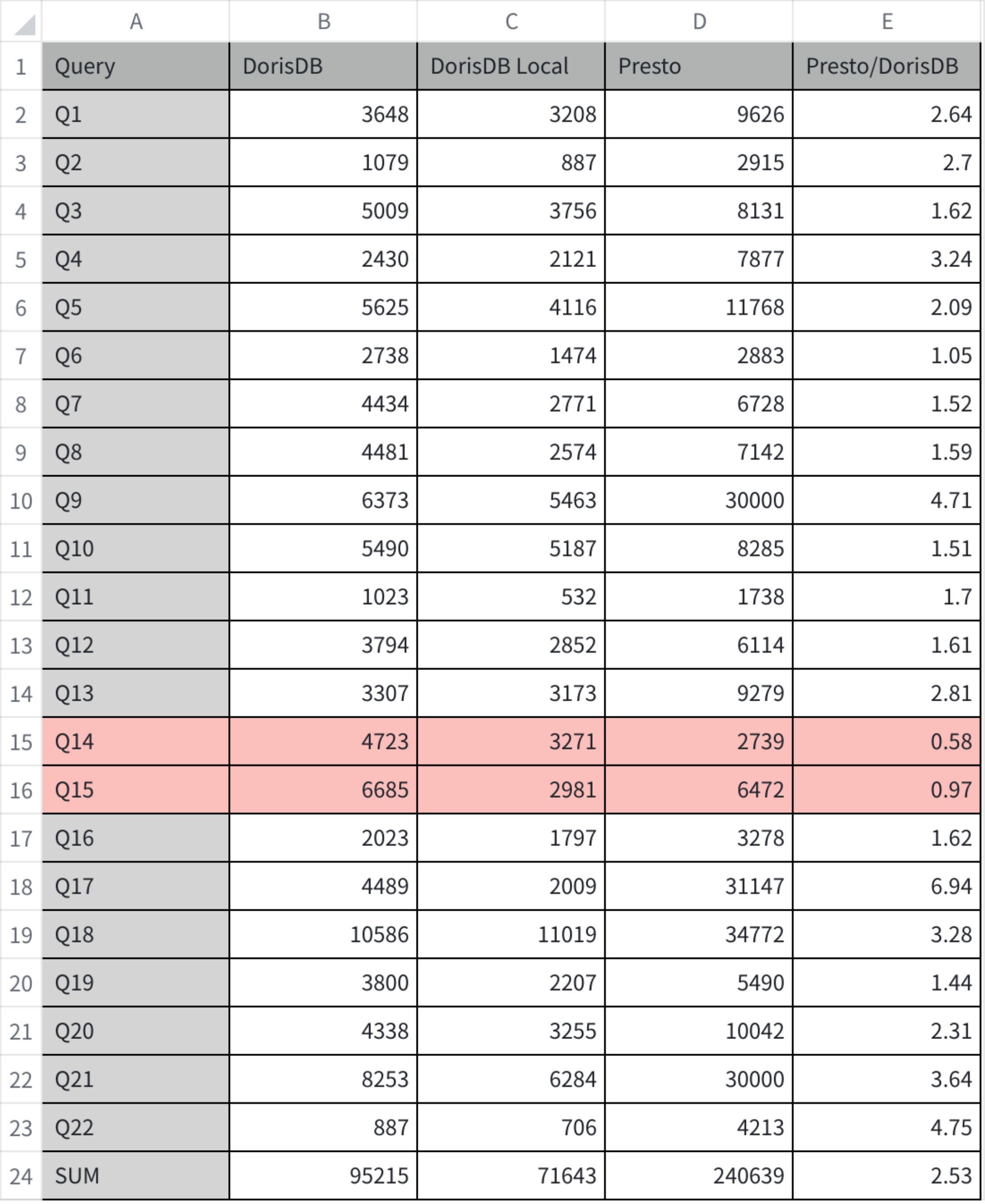
	A	B	C	D	E	F	G
1	Query	DorisDB ORC	DorisDB Native	Presto	Presto/ORC	3Nodes	ORC/Nati
2	Q1	3208	1970	9813	3.06	1854	
3	Q2	887	286	2796	3.15	560	
4	Q3	3756	2329	8950	2.38	1337	
5	Q4	2121	1179	7916	3.73	928	
6	Q5	4116	3193	11736	2.85	1382	
7	Q6	1474	98	2262	1.53	412	
8	Q7	2771	1125	6404	2.31	1854	
9	Q8	2574	1147	6985	2.71	933	
10	Q9	5463	4540	30000	5.49	3751	
11	Q10	5187	2338	8110	1.56	1876	
12	Q11	532	208	1728	3.25	341	
13	Q12	2852	987	6776	2.38	802	
14	Q13	3173	3243	9225	2.91	4623	
15	Q14	3271	280	2580	0.79	503	
16	Q15	2981	376	6101	2.05	946	
17	Q16	1797	760	3293	1.83	610	
18	Q17	2009	789	31654	15.76	820	
19	Q18	11019	10822	34206	3.1	6872	
20	Q19	2207	753	5428	2.46	687	
21	Q20	3255	2356	9932	3.05	2107	
22	Q21	6284	4759	30000	4.77	4081	
23	Q22	706	597	4378	6.2	885	
24	SUM	71643	44135	240273	3.35	38164	



Presto Q9,Q21 OOM运行失败
Presto CLI emr-338-1.1.0-2-gf82ea20
DorisDB 1afbd2d

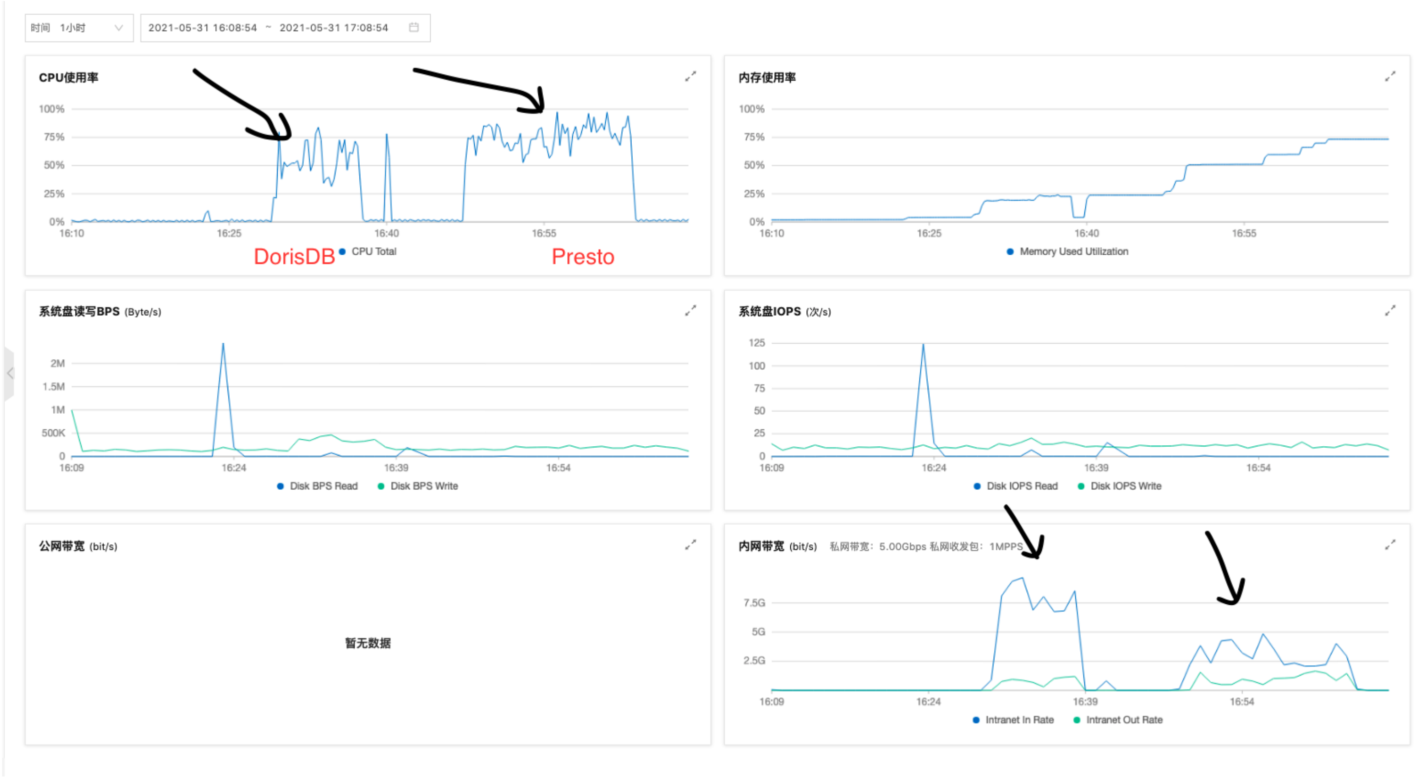
测试结果3(跨网络读取)

在同一个局域网内，HDFS和计算节点分布在不同的节点上：3个hdfs datanode节点，3个计算节点（运行presto server和dorisdb be)．下面是测试结果：可以看到加速比没有像本地读(3.35)那么高，相比Presto, DorisDB在HDFS IO读取上没有什么优势。



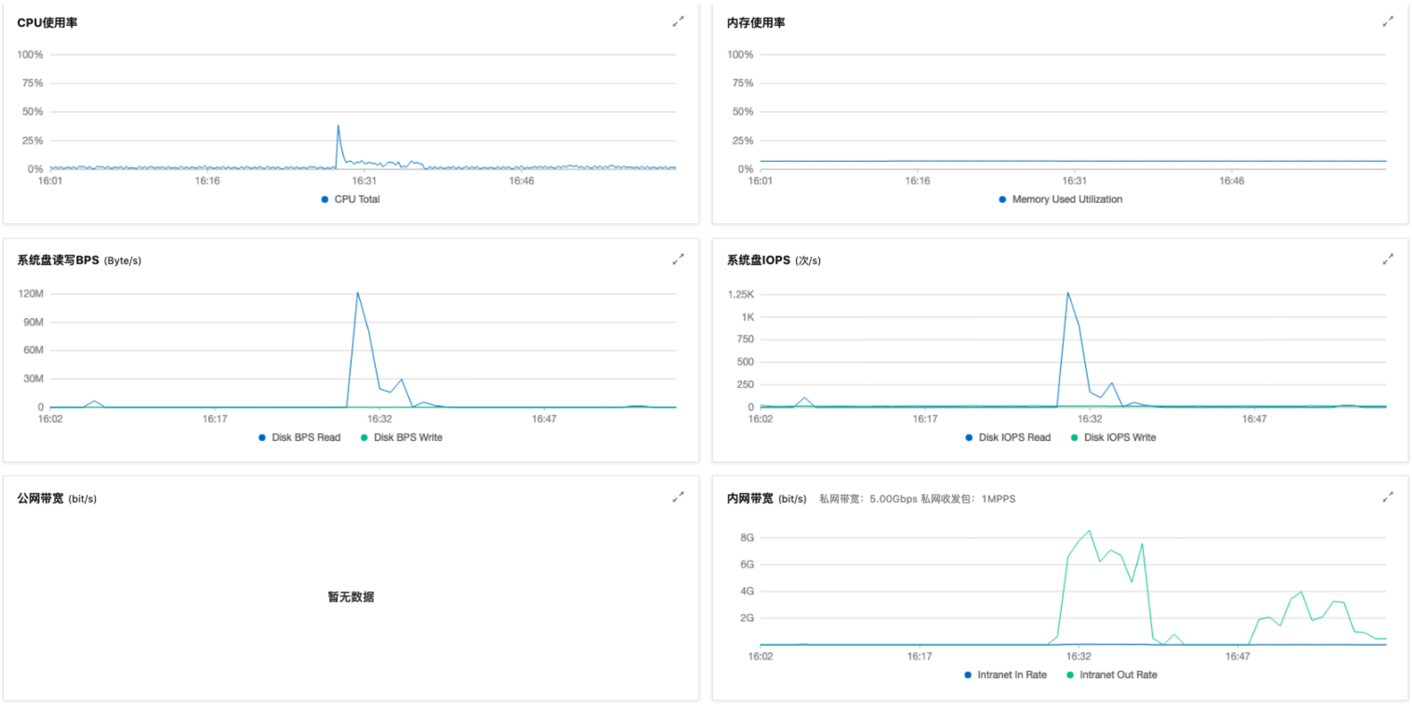
	A	B	C	D	E
1	Query	DorisDB	DorisDB Local	Presto	Presto/DorisDB
2	Q1	3648	3208	9626	2.64
3	Q2	1079	887	2915	2.7
4	Q3	5009	3756	8131	1.62
5	Q4	2430	2121	7877	3.24
6	Q5	5625	4116	11768	2.09
7	Q6	2738	1474	2883	1.05
8	Q7	4434	2771	6728	1.52
9	Q8	4481	2574	7142	1.59
10	Q9	6373	5463	30000	4.71
11	Q10	5490	5187	8285	1.51
12	Q11	1023	532	1738	1.7
13	Q12	3794	2852	6114	1.61
14	Q13	3307	3173	9279	2.81
15	Q14	4723	3271	2739	0.58
16	Q15	6685	2981	6472	0.97
17	Q16	2023	1797	3278	1.62
18	Q17	4489	2009	31147	6.94
19	Q18	10586	11019	34772	3.28
20	Q19	3800	2207	5490	1.44
21	Q20	4338	3255	10042	2.31
22	Q21	8253	6284	30000	3.64
23	Q22	887	706	4213	4.75
24	SUM	95215	71643	240639	2.53

从监控来看，DorisDB网络peak bandwidth是Presto的两倍，因为Presto运行时间是DorisDB的2.53倍，所以可能读取数据的量可能大致是相同的。但是因为DorisDB的peak bandwidth能到7.5Gbps，所以在公有云的环境下面可能会被限制带宽。



时间 1小时 2021-05-31 16:08:54 ~ 2021-05-31 17:08:54

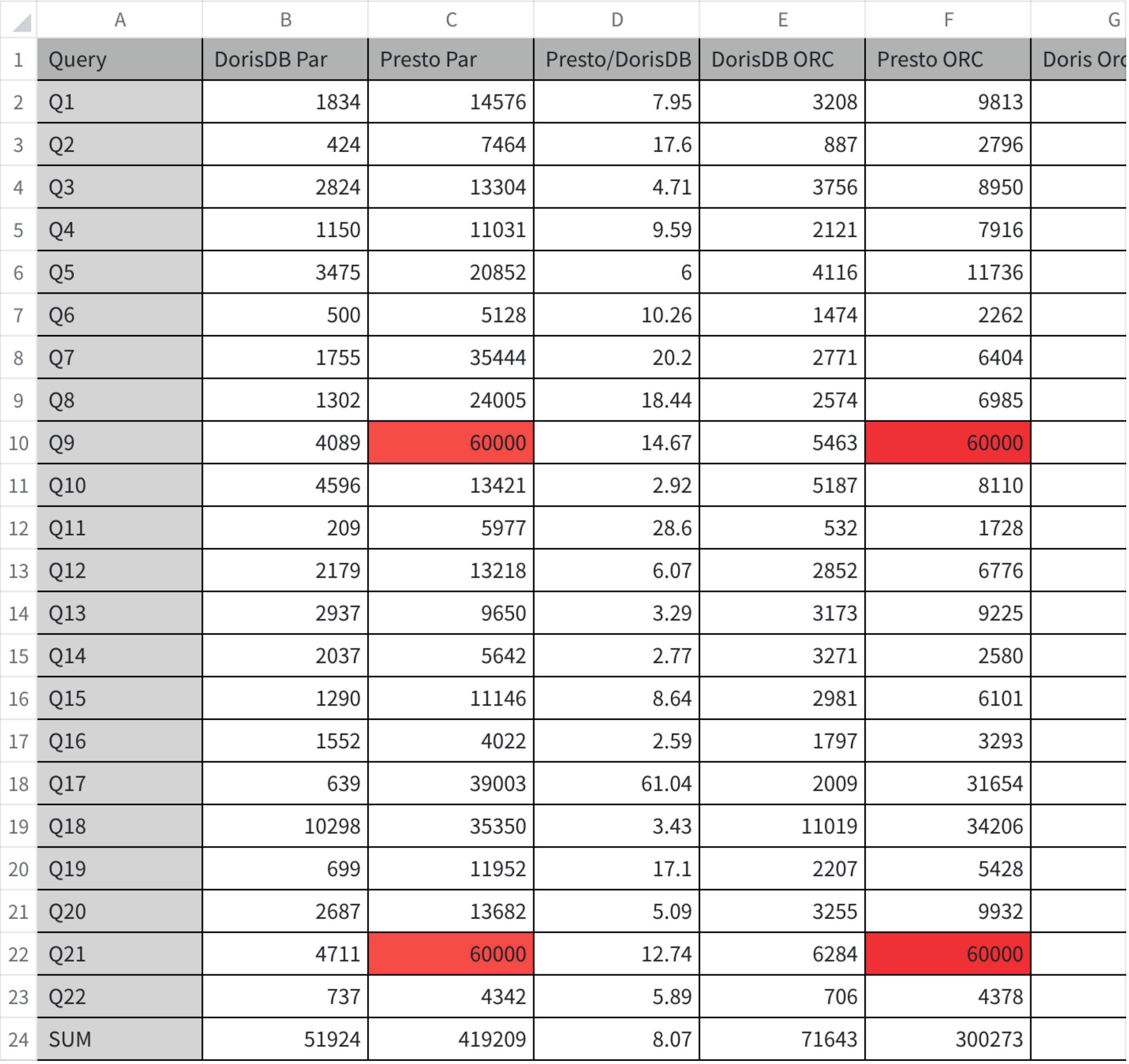
下图是HDFS Datanode上看到的，和上图网络流量趋势基本一致。



测试结果4(本地读取, ORC vs. Parquet)

其中DorisDB Par是parquet格式，DorisDB Orc是ORC格式. Doris Orc/Par是ORC/Parquet比例。几乎在所有的case上(Q22除外），Parquet时间都比ORC时间要少，总的下来ORC多运行1.38倍的时间。这方面差距主要还是在读取IO上。

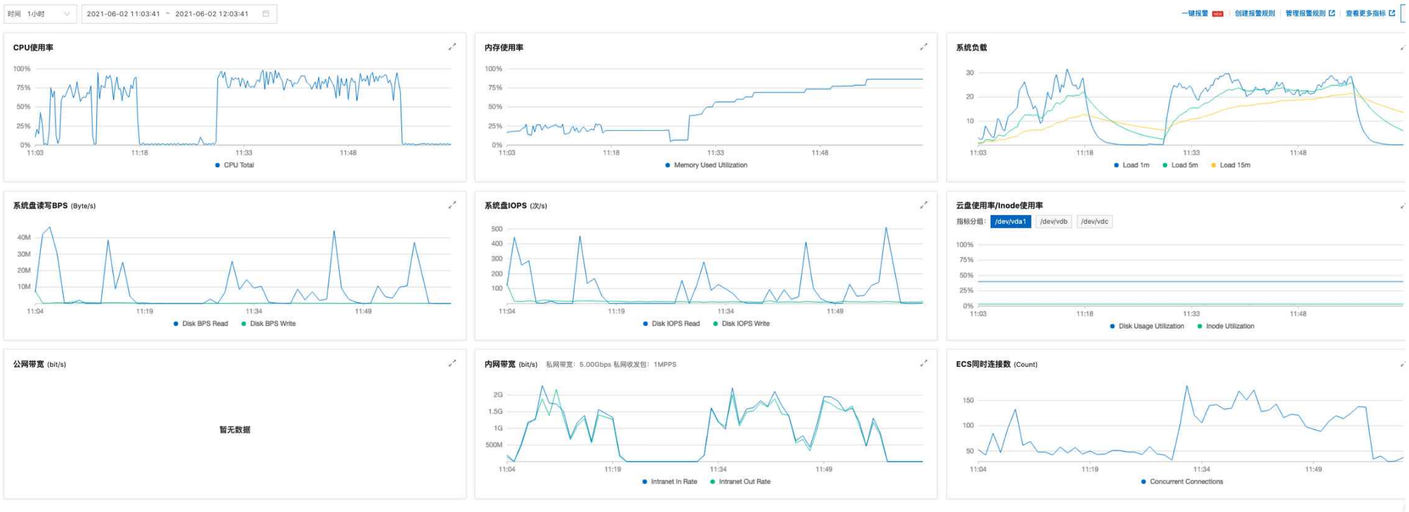
其中Presto Par是Presto读取Parquet时间。Presto在Parquet格式上没有做什么优化，所以对比意义不大。



	A	B	C	D	E	F	G
1	Query	DorisDB Par	Presto Par	Presto/DorisDB	DorisDB ORC	Presto ORC	Doris Orc
2	Q1	1834	14576	7.95	3208	9813	
3	Q2	424	7464	17.6	887	2796	
4	Q3	2824	13304	4.71	3756	8950	
5	Q4	1150	11031	9.59	2121	7916	
6	Q5	3475	20852	6	4116	11736	
7	Q6	500	5128	10.26	1474	2262	
8	Q7	1755	35444	20.2	2771	6404	
9	Q8	1302	24005	18.44	2574	6985	
10	Q9	4089	60000	14.67	5463	60000	
11	Q10	4596	13421	2.92	5187	8110	
12	Q11	209	5977	28.6	532	1728	
13	Q12	2179	13218	6.07	2852	6776	
14	Q13	2937	9650	3.29	3173	9225	
15	Q14	2037	5642	2.77	3271	2580	
16	Q15	1290	11146	8.64	2981	6101	
17	Q16	1552	4022	2.59	1797	3293	
18	Q17	639	39003	61.04	2009	31654	
19	Q18	10298	35350	3.43	11019	34206	
20	Q19	699	11952	17.1	2207	5428	
21	Q20	2687	13682	5.09	3255	9932	
22	Q21	4711	60000	12.74	6284	60000	
23	Q22	737	4342	5.89	706	4378	
24	SUM	51924	419209	8.07	71643	300273	

下面是运行DorisDB Parquet和Presto Parquet的监控，可以看到DorisDB使用CPU还是比较充分的，也说明花费在IO时间上比较少。Presto CPU也非常满，运行时间长可能主要原因会是因为没有做好过

滤。



Appendix. native优化配置

```SQL
create database if not exists tpch_native;
use tpch_native;

CREATE TABLE IF NOT EXISTS `customer` (
  `c_custkey` bigint(20) NOT NULL COMMENT "",
  `c_name` varchar(25) NOT NULL COMMENT "",
  `c_address` varchar(40) NOT NULL COMMENT "",
  `c_nationkey` int(11) NOT NULL COMMENT "",
  `c_phone` char(15) NOT NULL COMMENT "",
  `c_acctbal` double NOT NULL COMMENT "",
  `c_mktsegment` char(10) NOT NULL COMMENT "",
  `c_comment` varchar(117) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`c_custkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`c_custkey`) BUCKETS 12
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `lineitem` (
  `l_orderkey` bigint(20) NOT NULL COMMENT "",
  `l_partkey` bigint(20) NOT NULL COMMENT "",
  `l_suppkey` int(11) NOT NULL COMMENT "",
  `l_linenumber` int(11) NOT NULL COMMENT "",
  `l_quantity` double NOT NULL COMMENT "",
  `l_extendedprice` double NOT NULL COMMENT "",
  `l_discount` double NOT NULL COMMENT "",
  `l_tax` double NOT NULL COMMENT "",
  `l_returnflag` char(1) NOT NULL COMMENT "",
  `l_linestatus` char(1) NOT NULL COMMENT "",
  `l_shipdate` date NOT NULL COMMENT "",
  `l_commitdate` date NOT NULL COMMENT "",
  `l_receiptdate` date NOT NULL COMMENT "",
  `l_shipinstruct` char(25) NOT NULL COMMENT "",
  `l_shipmode` char(10) NOT NULL COMMENT "",
  `l_comment` varchar(44) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`l_orderkey`)
COMMENT "OLAP"
PARTITION BY RANGE(`l_shipdate`)
(PARTITION p1992 VALUES [('0000-01-01'), ('1993-01-01')),
PARTITION p1993 VALUES [('1993-01-01'), ('1994-01-01')),
PARTITION p1994 VALUES [('1994-01-01'), ('1995-01-01')),
PARTITION p1995 VALUES [('1995-01-01'), ('1996-01-01')),
PARTITION p1996 VALUES [('1996-01-01'), ('1997-01-01')),
PARTITION p1997 VALUES [('1997-01-01'), ('1998-01-01')),
PARTITION p1998 VALUES [('1998-01-01'), ('1999-01-01')))
DISTRIBUTED BY HASH(`l_orderkey`) BUCKETS 48
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `nation` (
  `n_nationkey` int(11) NOT NULL COMMENT "",
  `n_name` char(25) NOT NULL COMMENT "",
  `n_regionkey` int(11) NOT NULL COMMENT "",
  `n_comment` varchar(152) NULL COMMENT ""
) ENGINE=OLAP
```

```
DUPLICATE KEY(`n_nationkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`n_nationkey`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `orders` (
  `o_orderkey` bigint(20) NOT NULL COMMENT "",
  `o_custkey` bigint(20) NOT NULL COMMENT "",
  `o_orderstatus` char(1) NOT NULL COMMENT "",
  `o_totalprice` double NOT NULL COMMENT "",
  `o_orderdate` date NOT NULL COMMENT "",
  `o_orderpriority` char(15) NOT NULL COMMENT "",
  `o_clerk` char(15) NOT NULL COMMENT "",
  `o_shippriority` int(11) NOT NULL COMMENT "",
  `o_comment` varchar(79) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`o_orderkey`)
COMMENT "OLAP"
PARTITION BY RANGE(`o_orderdate`)
(PARTITION p1992 VALUES [('0000-01-01'), ('1993-01-01')),
PARTITION p1993 VALUES [('1993-01-01'), ('1994-01-01')),
PARTITION p1994 VALUES [('1994-01-01'), ('1995-01-01')),
PARTITION p1995 VALUES [('1995-01-01'), ('1996-01-01')),
PARTITION p1996 VALUES [('1996-01-01'), ('1997-01-01')),
PARTITION p1997 VALUES [('1997-01-01'), ('1998-01-01')),
PARTITION p1998 VALUES [('1998-01-01'), ('1999-01-01')))
DISTRIBUTED BY HASH(`o_orderkey`) BUCKETS 48
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `part` (
  `p_partkey` bigint(20) NOT NULL COMMENT "",
  `p_name` varchar(55) NOT NULL COMMENT "",
  `p_mfgr` char(25) NOT NULL COMMENT "",
  `p_brand` char(10) NOT NULL COMMENT "",
  `p_type` varchar(25) NOT NULL COMMENT "",
  `p_size` int(11) NOT NULL COMMENT "",
  `p_container` char(10) NOT NULL COMMENT "",
  `p_container` char(10) NOT NULL COMMENT "",
  `p_retailprice` double NOT NULL COMMENT "",
  `p_comment` varchar(23) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`p_partkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`p_partkey`) BUCKETS 12
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `partsupp` (
  `ps_partkey` bigint(20) NOT NULL COMMENT "",
  `ps_suppkey` bigint(20) NOT NULL COMMENT "",
  `ps_availqty` int(11) NOT NULL COMMENT "",
  `ps_supplycost` double NOT NULL COMMENT "",
  `ps_comment` varchar(199) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`ps_partkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`ps_partkey`) BUCKETS 48
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `region` (
  `r_regionkey` int(11) NOT NULL COMMENT "",
  `r_name` char(25) NOT NULL COMMENT "",
  `r_comment` varchar(152) NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`r_regionkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`r_regionkey`) BUCKETS 1
PROPERTIES (
"replication_num" = "3",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);

CREATE TABLE IF NOT EXISTS `supplier` (
  `s_suppkey` int(11) NOT NULL COMMENT "",
  `s_name` char(25) NOT NULL COMMENT "",
  `s_address` varchar(40) NOT NULL COMMENT "",
  `s_nationkey` int(11) NOT NULL COMMENT "",
  `s_phone` char(15) NOT NULL COMMENT "",
  `s_acctbal` double NOT NULL COMMENT "",
  `s_comment` varchar(101) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`s_suppkey`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`s_suppkey`) BUCKETS 3
PROPERTIES (
"replication_num" = "1",
"in_memory" = "false",
"storage_format" = "DEFAULT"
);
```