

腾讯云数据仓库 TCHouse-C

性能测试



腾讯云

【 版权声明 】

©2013–2024 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

性能测试

测试方案介绍

测试结果参考

性能测试

测试方案介绍

最近更新时间：2023-09-22 20:05:12

前言

本文为您介绍如何使用 Star Schema 数据集对腾讯云 TCHouse-C 进行性能测试，给出数据导入及性能测试的参考方案。

准备工作

购买实例

参照 [快速入门](#) 购买腾讯云 TCHouse-C 实例，可选择标准型、高性能型、大存储型，软件建议选择22.8及以后版本。

准备测试机器

准备能够访问腾讯云 TCHouse-C 服务的 Linux 系统机器，并在该机器上安装 clickhouse-client 工具。测试机器需要能够访问腾讯云 TCHouse-C 服务，至少需1.5TB存储空间。在测试机器上安装`clickhouse client`工具，请参见 [安装文档](#)。

在购买实例后，请在控制台中调整参数如下：

| 参数名称 | 所载文件 | 作用 | 建议值 |
|------------------------------|----------------|-----------------------|-------------|
| max_threads | users.xml | 单个查询允许使用的线程数 | CPU 核数 |
| max_insert_threads | users.xml | 单次写入允许使用的线程数 | CPU 核数 |
| max_memory_usage users.xml | 单次查询允许使用的内存最大值 | 总内存数 | 10GB |
| background_pool_size | users.xml | MergeTree 引擎后台任务线程池大小 | CPU 核数*2 |
| max_thread_pool_size | config.xml | 全局线程池最大分配线程数量 | 20000 |
| max_open_files | config.xml | 允许进程打开的最大文件句柄上 | 1000000 |
| mark_cache_size | config.xml | mark 文件缓存大小 | 10737418240 |

具体参数调整，请参见 [参数配置](#)。

注意：调整完成后，请重启集群。

测试步骤

确认软件版本

使用 `clickhouse client` 访问腾讯云 TCHouse-C 服务，查看软件版本。

```
clickhouse client --host $HOST --port $PORT -q "select version()"
```

注意：请确保软件版本大于22.8.*。

准备数据生成工具

```
$ git clone git@github.com:vadimtk/ssb-dbggen.git$ cd ssb-dbggen$ make
```

生成测试数据

ssb-dbggen工具支持两种规模的数据，使用参数 `-s 100` 可以生成约6亿行规模的数据，使用参数 `-s 1000` 可以生成约60亿行规模的数据。建议采用 `-s 1000`。

```
$ ./dbggen -s 1000 -T c$ ./dbggen -s 1000 -T l$ ./dbggen -s 1000 -T p$ ./dbggen -s 1000 -T s
```

创建数据库表

在腾讯云 TCHouse-C 控制台上，获取服务入口信息：访问IP地址和服务端口。分别记录为 HOST 和 PORT。

使用 `clickhouse client` 工具链接腾讯云 TCHouse-C 服务，执行如下 SQL：

```
CREATE TABLE customer
(
    C_CUSTKEY      UInt32,
    C_NAME         String,
    C_ADDRESS      String,
    C_CITY         LowCardinality(String),
    C_NATION       LowCardinality(String),
    C_REGION       LowCardinality(String),
    C_PHONE        String,
    C_MKTSEGMENT   LowCardinality(String)
)
ENGINE = MergeTree ORDER BY (C_CUSTKEY);

CREATE TABLE lineorder
```

```
(
    LO_ORDERKEY          UInt32,
    LO_LINENUMBER        UInt8,
    LO_CUSTKEY            UInt32,
    LO_PARTKEY            UInt32,
    LO_SUPPKEY            UInt32,
    LO_ORDERDATE          Date,
    LO_ORDERPRIORITY      LowCardinality(String),
    LO_SHIPPRIORITY       UInt8,
    LO_QUANTITY           UInt8,
    LO_EXTENDEDPRICE      UInt32,
    LO_ORDTOTALPRICE      UInt32,
    LO_DISCOUNT          UInt8,
    LO_REVENUE            UInt32,
    LO_SUPPLYCOST         UInt32,
    LO_TAX                UInt8,
    LO_COMMITDATE         Date,
    LO_SHIPMODE           LowCardinality(String)
)
ENGINE = MergeTree PARTITION BY toYear(LO_ORDERDATE) ORDER BY (LO_ORDERDATE, LO_ORDERKEY);

CREATE TABLE part
(
    P_PARTKEY      UInt32,
    P_NAME         String,
    P_MFGR         LowCardinality(String),
    P_CATEGORY     LowCardinality(String),
    P_BRAND        LowCardinality(String),
    P_COLOR        LowCardinality(String),
    P_TYPE         LowCardinality(String),
    P_SIZE         UInt8,
    P_CONTAINER    LowCardinality(String)
)
ENGINE = MergeTree ORDER BY P_PARTKEY;

CREATE TABLE supplier
(
    S_SUPPKEY      UInt32,
    S_NAME         String,
    S_ADDRESS      String,
    S_CITY         LowCardinality(String),
    S_NATION       LowCardinality(String),
    S_REGION       LowCardinality(String),
    S_PHONE        String
)
```

```
)  
ENGINE = MergeTree ORDER BY S_SUPPKEY;
```

导入测试数据

首先，导入基础表数据：

```
$ clickhouse client --host $HOST --port $PORT --  
query "INSERT INTO customer FORMAT CSV" < customer.tbl  
$ clickhouse client --host $HOST --port $PORT --  
query "INSERT INTO part FORMAT CSV" < part.tbl  
$ clickhouse client --host $HOST --port $PORT --  
query "INSERT INTO supplier FORMAT CSV" < supplier.tbl  
$ clickhouse client --host $HOST --port $PORT --  
query "INSERT INTO lineorder FORMAT CSV" < lineorder.tbl
```

其次，根据基础表数据生成宽表数据。

这里需要注意已经调整了 `max_memory_usage` 和 `max_insert_threads` 参数。

```
CREATE TABLE lineorder_flat  
ENGINE = MergeTree ORDER BY (LO_ORDERDATE, LO_ORDERKEY)  
AS SELECT  
  I.LO_ORDERKEY AS LO_ORDERKEY,  
  I.LO_LINENUMBER AS LO_LINENUMBER,  
  I.LO_CUSTKEY AS LO_CUSTKEY,  
  I.LO_PARTKEY AS LO_PARTKEY,  
  I.LO_SUPPKEY AS LO_SUPPKEY,  
  I.LO_ORDERDATE AS LO_ORDERDATE,  
  I.LO_ORDERPRIORITY AS LO_ORDERPRIORITY,  
  I.LO_SHIPPRIORITY AS LO_SHIPPRIORITY,  
  I.LO_QUANTITY AS LO_QUANTITY,  
  I.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,  
  I.LO_ORDTOTALPRICE AS LO_ORDTOTALPRICE,  
  I.LO_DISCOUNT AS LO_DISCOUNT,  
  I.LO_REVENUE AS LO_REVENUE,  
  I.LO_SUPPLYCOST AS LO_SUPPLYCOST,  
  I.LO_TAX AS LO_TAX,  
  I.LO_COMMITDATE AS LO_COMMITDATE,  
  I.LO_SHIPMODE AS LO_SHIPMODE,  
  c.C_NAME AS C_NAME,  
  c.C_ADDRESS AS C_ADDRESS,  
  c.C_CITY AS C_CITY,  
  c.C_NATION AS C_NATION,  
  c.C_REGION AS C_REGION,
```

```
c.C_PHONE AS C_PHONE,
c.C_MKTSEGMENT AS C_MKTSEGMENT,
s.S_NAME AS S_NAME,
s.S_ADDRESS AS S_ADDRESS,
s.S_CITY AS S_CITY,
s.S_NATION AS S_NATION,
s.S_REGION AS S_REGION,
s.S_PHONE AS S_PHONE,
p.P_NAME AS P_NAME,
p.P_MFGR AS P_MFGR,
p.P_CATEGORY AS P_CATEGORY,
p.P_BRAND AS P_BRAND,
p.P_COLOR AS P_COLOR,
p.P_TYPE AS P_TYPE,
p.P_SIZE AS P_SIZE,
p.P_CONTAINER AS P_CONTAINER
FROM lineorder AS l
INNER JOIN customer AS c ON c.C_CUSTKEY = l.LO_CUSTKEY
INNER JOIN supplier AS s ON s.S_SUPPKEY = l.LO_SUPPKEY
INNER JOIN part AS p ON p.P_PARTKEY = l.LO_PARTKEY;
```

优化查询（可选）

腾讯云 TCHouse-C 提供了预计算能力，加速执行。这里使用 PROJECTION 来加速查询。执行如下 SQL：

```
ALTER TABLE lineorder_flat
ADD PROJECTION p1
(
  SELECT
    toYear(LO_ORDERDATE) AS year,
    sum(LO_REVENUE)
  GROUP BY
    year,
    P_BRAND,
    P_CATEGORY,
    S_REGION
);
ALTER TABLE lineorder_flat
ADD PROJECTION p2
(
  SELECT
    toYear(LO_ORDERDATE) AS year,
    sum(LO_REVENUE)
  GROUP BY
    year,
```



```
C_NATION,  
S_NATION,  
C_REGION,  
S_REGION  
);  
ALTER TABLE lineorder_flat  
ADD PROJECTION p3  
(  
  SELECT  
    toYear(LO_ORDERDATE) AS year,  
    sum(LO_REVENUE)  
  GROUP BY  
    year,  
    C_CITY,  
    S_CITY  
);  
ALTER TABLE lineorder_flat  
ADD PROJECTION p4  
(  
  SELECT  
    toYear(LO_ORDERDATE) AS year,  
    sum(LO_REVENUE)  
  GROUP BY  
    year,  
    C_NATION,  
    C_CITY,  
    S_NATION,  
    S_CITY  
);  
ALTER TABLE lineorder_flat  
ADD PROJECTION p5  
(  
  SELECT  
    toYear(LO_ORDERDATE) AS year,  
    sum(LO_REVENUE - LO_SUPPLYCOST) AS profit  
  GROUP BY  
    year,  
    C_NATION,  
    C_REGION,  
    S_REGION,  
    P_MFGR,  
    P_MFGR  
);  
ALTER TABLE lineorder_flat  
ADD PROJECTION p6
```

```
(
  SELECT
    toYear(LO_ORDERDATE) AS year,
    sum(LO_REVENUE - LO_SUPPLYCOST)
  GROUP BY
    year,
    S_NATION,
    P_CATEGORY,
    C_REGION,
    S_REGION,
    P_MFGR
);
ALTER TABLE lineorder_flat
ADD PROJECTION p7
(
  SELECT
    toYear(LO_ORDERDATE) AS year,
    sum(LO_REVENUE - LO_SUPPLYCOST)
  GROUP BY
    year,
    S_CITY,
    P_BRAND,
    S_NATION,
    P_CATEGORY
);
```

执行上述 SQL 后，需要对存量数据进行处理，使 PROJECTION 在存量数据上生效。执行如下 SQL：

```
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p1;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p2;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p3;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p4;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p5;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p6;
ALTER TABLE lineorder_flat MATERIALIZE PROJECTION p7;
```

注意：该步骤是可选的，使用优化后，性能提升非常明显。

执行测试 SQL 并统计执行时间数据

Q1.1

```
SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
```

```
FROM lineorder_flat
WHERE toYear(LO_ORDERDATE) = 1993 AND LO_DISCOUNT BETWEEN 1 AND 3 AND L
O_QUANTITY < 25;
```

Q1.2

```
SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
FROM lineorder_flat
WHERE toYYYYMM(LO_ORDERDATE) = 199401 AND LO_DISCOUNT BETWEEN 4 AND 6
AND LO_QUANTITY BETWEEN 26 AND 35;
```

Q1.3

```
SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
FROM lineorder_flat
WHERE toISOWeek(LO_ORDERDATE) = 6 AND toYear(LO_ORDERDATE) = 1994
AND LO_DISCOUNT BETWEEN 5 AND 7 AND LO_QUANTITY BETWEEN 26 AND 35;
```

Q2.1

```
SELECT
    sum(LO_REVENUE),
    toYear(LO_ORDERDATE) AS year,
    P_BRAND
FROM lineorder_flat
WHERE P_CATEGORY = 'MFGR#12' AND S_REGION = 'AMERICA'
GROUP BY
    year,
    P_BRAND
ORDER BY
    year,
    P_BRAND;
```

Q2.2

```
SELECT
    sum(LO_REVENUE),
    toYear(LO_ORDERDATE) AS year,
    P_BRAND
FROM lineorder_flat
```

```
WHERE P_BRAND >= 'MFGR#2221' AND P_BRAND <= 'MFGR#2228' AND S_REGION =  
'ASIA'  
GROUP BY  
    year,  
    P_BRAND  
ORDER BY  
    year,  
    P_BRAND;
```

Q2.3

```
SELECT  
    sum(LO_REVENUE),  
    toYear(LO_ORDERDATE) AS year,  
    P_BRAND  
FROM lineorder_flat  
WHERE P_BRAND = 'MFGR#2239' AND S_REGION = 'EUROPE'  
GROUP BY  
    year,  
    P_BRAND  
ORDER BY  
    year,  
    P_BRAND;
```

Q3.1

```
SELECT  
    C_NATION,  
    S_NATION,  
    toYear(LO_ORDERDATE) AS year,  
    sum(LO_REVENUE) AS revenue  
FROM lineorder_flat  
WHERE C_REGION = 'ASIA' AND S_REGION = 'ASIA' AND year >= 1992 AND year <=  
1997  
GROUP BY  
    C_NATION,  
    S_NATION,  
    year  
ORDER BY  
    year ASC,  
    revenue DESC;
```

Q3.2

```
SELECT
  C_CITY,
  S_CITY,
  toYear(LO_ORDERDATE) AS year,
  sum(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE C_NATION = 'UNITED STATES' AND S_NATION = 'UNITED STATES' AND year >=
  1992 AND year <= 1997
GROUP BY
  C_CITY,
  S_CITY,
  year
ORDER BY
  year ASC,
  revenue DESC;
```

Q3.3

```
SELECT
  C_CITY,
  S_CITY,
  toYear(LO_ORDERDATE) AS year,
  sum(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE (C_CITY = 'UNITED KI1' OR C_CITY = 'UNITED KI5') AND (S_CITY = 'UNITED KI1'
  OR S_CITY = 'UNITED KI5') AND year >= 1992 AND year <= 1997
GROUP BY
  C_CITY,
  S_CITY,
  year
ORDER BY
  year ASC,
  revenue DESC;
```

Q3.4

```
SELECT
  C_CITY,
  S_CITY,
  toYear(LO_ORDERDATE) AS year,
  sum(LO_REVENUE) AS revenue
FROM lineorder_flat
```

```
WHERE (C_CITY = 'UNITED KI1' OR C_CITY = 'UNITED KI5') AND (S_CITY = 'UNITED KI1'
OR S_CITY = 'UNITED KI5') AND toYYYYMM(LO_ORDERDATE) = 199712
GROUP BY
    C_CITY,
    S_CITY,
    year
ORDER BY
    year ASC,
    revenue DESC;
```

Q4.1

```
SELECT
    toYear(LO_ORDERDATE) AS year,
    C_NATION,
    sum(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE C_REGION = 'AMERICA' AND S_REGION = 'AMERICA' AND (P_MFGR = 'MFGR#
1' OR P_MFGR = 'MFGR#2')
GROUP BY
    year,
    C_NATION
ORDER BY
    year ASC,
    C_NATION ASC;
```

Q4.2

```
SELECT
    toYear(LO_ORDERDATE) AS year,
    S_NATION,
    P_CATEGORY,
    sum(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE C_REGION = 'AMERICA' AND S_REGION = 'AMERICA' AND (year = 1997 OR ye
ar = 1998) AND (P_MFGR = 'MFGR#1' OR P_MFGR = 'MFGR#2')
GROUP BY
    year,
    S_NATION,
    P_CATEGORY
ORDER BY
    year ASC,
    S_NATION ASC,
    P_CATEGORY ASC;
```

Q4.3

```
SELECT
  toYear(LO_ORDERDATE) AS year,
  S_CITY,
  P_BRAND,
  sum(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE S_NATION = 'UNITED STATES' AND (year = 1997 OR year = 1998) AND P_CATEGORY = 'MFGR#14'
GROUP BY
  year,
  S_CITY,
  P_BRAND
ORDER BY
  year ASC,
  S_CITY ASC,
  P_BRAND ASC;
```

总结

性能测试作为腾讯云 TCHouse-C 业务接入前的重要一环，是性能评估以及资源评估的重要依据。

通常业务面临多种系统选型时，也会进行性能对比测试。在对比测试过程中，需要注意以下几点：

- 腾讯云 TCHouse-C 的一些关键参数会影响性能，务必调整合理，才能充分发挥其性能优势。
- 需要对齐资源。例如，一次性分布式查询，腾讯云 TCHouse-C 只有1/2节点参与计算；而其他系统则是全部节点参与计算。在这种情况下，在等同数据规模下，腾讯云 TCHouse-C 可能性能数据不占优。在这种情况下，可以调整集群备份策略，让所有节点参与计算。
- 腾讯云 TCHouse-C 有很多特有的性能优化机制，开启这些机制，能够明显提升查询性能。

测试结果参考

最近更新时间：2023-09-22 20:05:13

关于 SSB 性能测试

SSB (Star Schema Benchmark) 是一个轻量级的数仓场景下的性能测试集。SSB 将 TPC-H 的雪花模式简化为了星型模式，将基准查询由 TPC-H 复杂的 Ad-Hoc 查询改为了结构更固定的 OLAP 查询，主要用于测试在星型模型下，多表关联查询的性能表现。Clickhouse 官方将 SSB 的星型模型打平转化成宽表，改造成了一个单表测试集（以下简称：SSB FLAT）来测试查询引擎的性能。本文将给出腾讯云数据仓库 TCHouse-C 在 SSB 单表数据集上的性能测试结果。

性能测试方案

监控工具

使用腾讯云数据仓库 TCHouse-C 的集群监控页面获取集群和节点监控信息。

测试内容

步骤一：使用 dbgen 工具初始化指定大小的数据存放于待测试的机型集群磁盘空间上。

步骤二：在腾讯云上购买TCHouse-C集群，并在该集群中创建所需的表。

步骤三：将步骤一中生成的数据导入到测试集群中。

步骤四：执行基准性能测试 SQL Q1.1-Q4.3指令。

步骤五：反复执行3次基准性能测试 SQL Q1.1-Q4.3指令。

步骤六：记录3次基准 SQL 执行的数据量，时间和速度。

查看结果

基准 SQL 执行后会输出如下信息，包含：执行查询结果行数、SQL 查询执行消耗的时间、SQL 查询的记录数量、SQL 查询的数据量、SQL 查询的数据记录数量速率和 SQL 查询的数据量速率。

```
9.0.16.17 :) SELECT sum(LO_EXTENDEDPRICE - LO_DISCOUNT) AS revenue FROM lineorder_flat WHERE toISOWeek(LO_ORDERDATE) = 6 AND toYear(LO_ORDERDATE) = 1994 ;
6 AND 35

sum(LO_EXTENDEDPRICE - LO_DISCOUNT)    revenue
lineorder_flat
(toISOWeek(LO_ORDERDATE) = 6) AND (toYear(LO_ORDERDATE) = 1994) AND ((LO_DISCOUNT >= 5) AND (LO_DISCOUNT <= 7)) AND ((LO_QUANTITY >= 26) AND (LO_QUANTITY <= 35))
26140715109448

1 rows in set. Elapsed: 0.040 sec. Processed 17.48 million rows, 139.81 MB (439.32 million rows/s., 3.51 GB/s.)
```

本次测试采用 SQL 查询执行消耗时间（s）指标项作为输出指标。

测试环境

硬件环境

本文共针对标准型+高性能云硬盘、标准型+增强型SSD云硬盘、高性能型+NVMe SSD 硬盘、大存储型+SATA HDD 本地硬盘四种机型场景进行性能测试，具体配置如下：

| 机型场景 | 节点规格 |
|----------------------|--|
| 标准型 + 高性能云硬盘 | CPU: 32 cores 内存: 128 GB 磁盘: 高性能云盘7000 GB |
| 标准型 + 增强型 SSD 云硬盘 | CPU: 32 cores 内存: 128 GB 磁盘: 增强型 SSD 云盘5000 GB |
| 高性能型 + NVMe SSD 硬盘 | CPU: 32 cores 内存: 128 GB 磁盘: NVMe SSD 硬盘7140 GB * 2 |
| 大存储型 + SATA HDD 本地硬盘 | CPU: 32 cores 内存: 128 GB 磁盘: SATA HDD 本地硬盘44640 GB * 2 |

软件版本

腾讯云 TChouse-C 21.8.12.29

测试数据集

在 -s 1000参数下，生成的数据集大小为：

| SSB 表名 | 行数 | 备注 |
|----------------|-------|------------|
| LINEORDER | 60亿 | 商品订单明细表表 |
| CUSTOMER | 3000万 | 客户信息表 |
| PART | 200万 | 零件信息表 |
| SUPPLIER | 200万 | 供应商信息表 |
| DATE | 2,556 | 日期表 |
| LINEORDER_FLAT | 60亿 | SSB 打平后的宽表 |

性能测试结果

| Query 编号 | 标准型 + 高性能云硬盘 | 标准型 + 增强型 SSD 云硬盘 | 高性能型 + NVMe SSD 硬盘 | 大存储型 + SATA HDD 本地硬盘 | | | | | | | | | |
|----------|--------------|-------------------|--------------------|----------------------|--|--|--|--|--|--|--|--|--|
| Q1.1 | 0.99 | 0.88 | 0.28 | 2.21 | | | | | | | | | |
| Q1.2 | 0.12 | 0.12 | 0.01 | 0.09 | | | | | | | | | |
| Q1.3 | 0.04 | 0.05 | 0.05 | 0.04 | | | | | | | | | |
| Q2.1 | 6.63 | 5.94 | 5.96 | 5.91 | | | | | | | | | |
| Q2.2 | 5.46 | 4.93 | 1.79 | 4.93 | | | | | | | | | |
| Q2.3 | 5.1 | 4.63 | 1.62 | 4.59 | | | | | | | | | |
| Q3.1 | 8.74 | 8.3 | 2.74 | 8.19 | | | | | | | | | |
| Q3.2 | 6.3 | 5.73 | 1.88 | 5.39 | | | | | | | | | |
| Q3.3 | 5.53 | 5.27 | 3.45 | 4.9 | | | | | | | | | |
| Q3.4 | 0.13 | 0.13 | 0.06 | 0.1 | | | | | | | | | |
| Q4.1 | 8.55 | 7.69 | 5.45 | 11.42 | | | | | | | | | |
| Q4.2 | 3.1 | 2.47 | 0.83 | 2.29 | | | | | | | | | |
| Q4.3 | 2.46 | 2.28 | 1.43 | 2.02 | | | | | | | | | |
| 总 | 53.15 | 48.42 | 25.55 | 52.08 | | | | | | | | | |

| | | | | | | | | | | | | | |
|----|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 时间 | | | | | | | | | | | | | |
|----|--|--|--|--|--|--|--|--|--|--|--|--|--|