基于EMR OLAP的开源实时数仓解决方案之ClickHouse事务实现

吴雪扬
高级开发工程师

目录 CONTENT

阿里巴巴 Alibaba.com | DataFun.

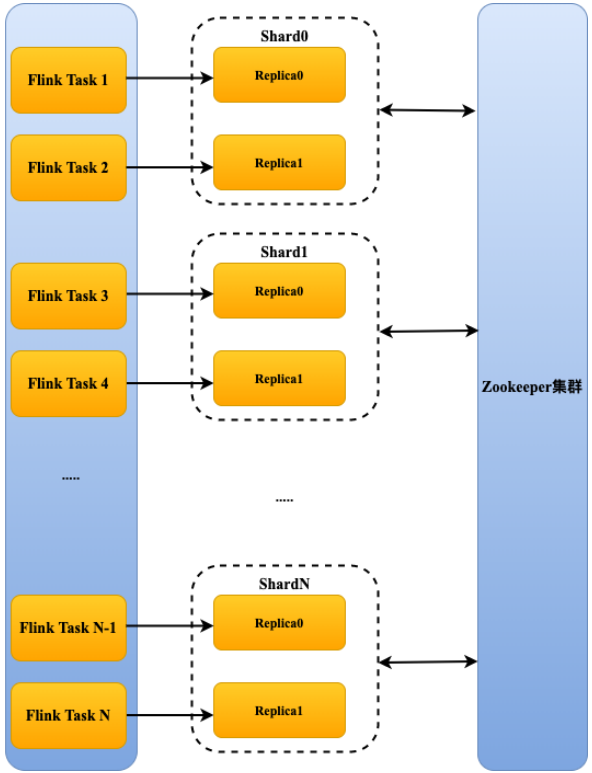
01 现状

为什么需要 ClickHouse 写事务

- 许多用户通过 Flink + ClickHouse 构建“用户画像”、“实时 BI 报表”等业务，有较高的数据准确性要求
- Flink Exactly Once 需要 Sink 端支持
- ClickHouse 社区暂时没有对事务的支持

ClickHouse 当前写入机制

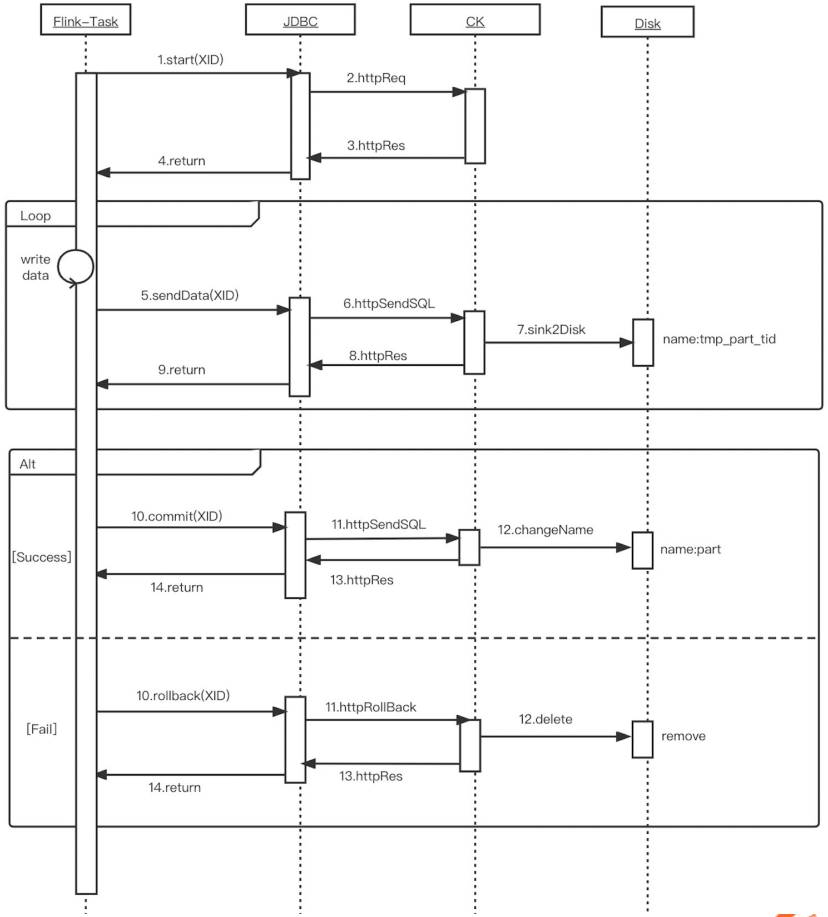
- 按照 Paritition 拆分 Block
- 写入拆分后的 Partitioned Block 成为临时 Data Part
- 重命名这个临时 Data Part 为正式的 Data Part
- 加入到 MergeTreeData 的 Data Part index 中，并对用户可见



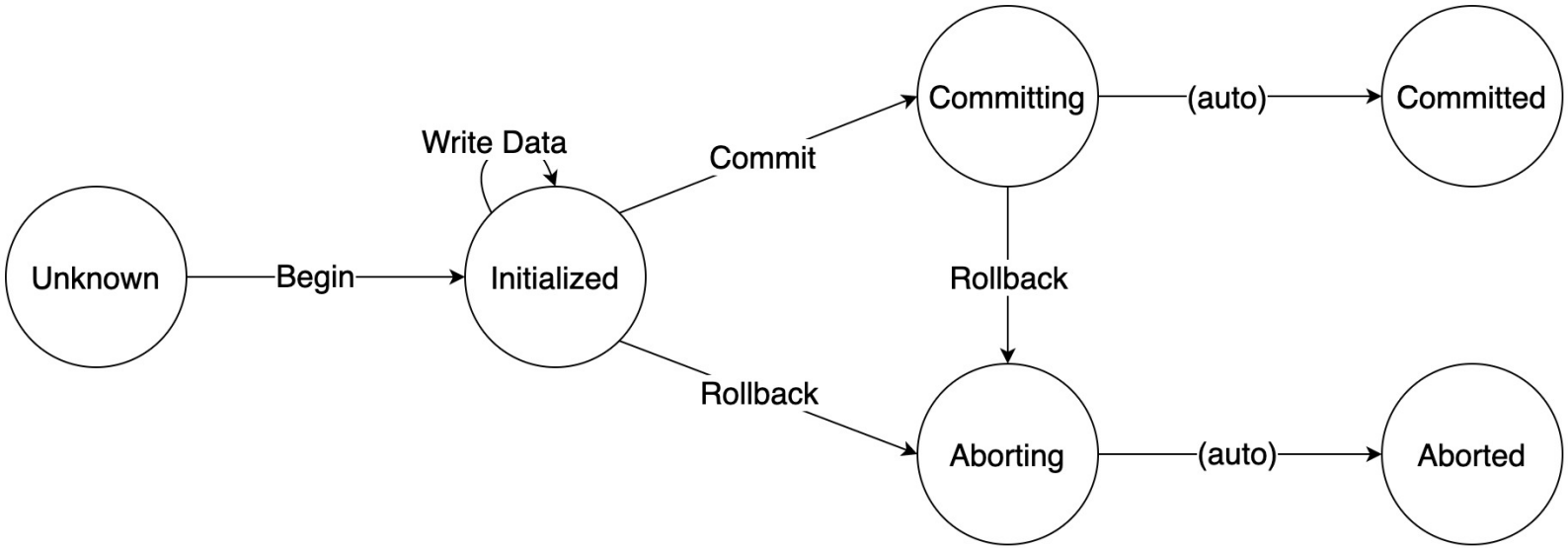
阿里巴巴 Alibaba.com | DataFun.

02
整体方案

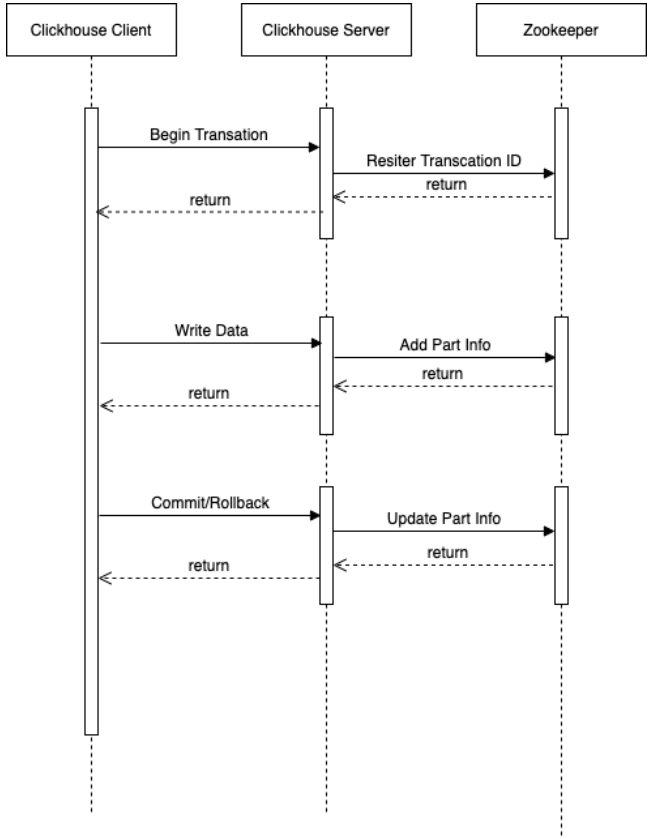
整体流程



ClickHouse 事务状态机



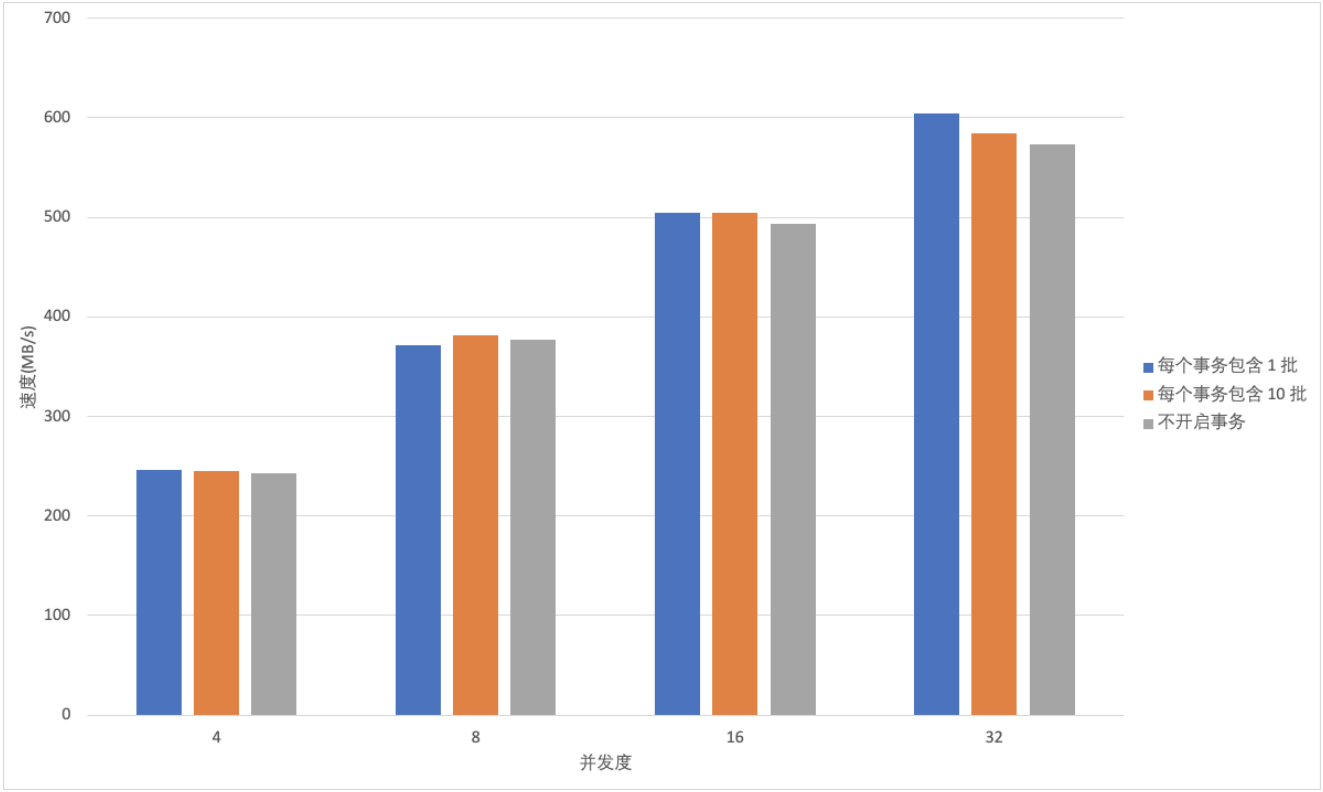
ClickHouse 写事务处理



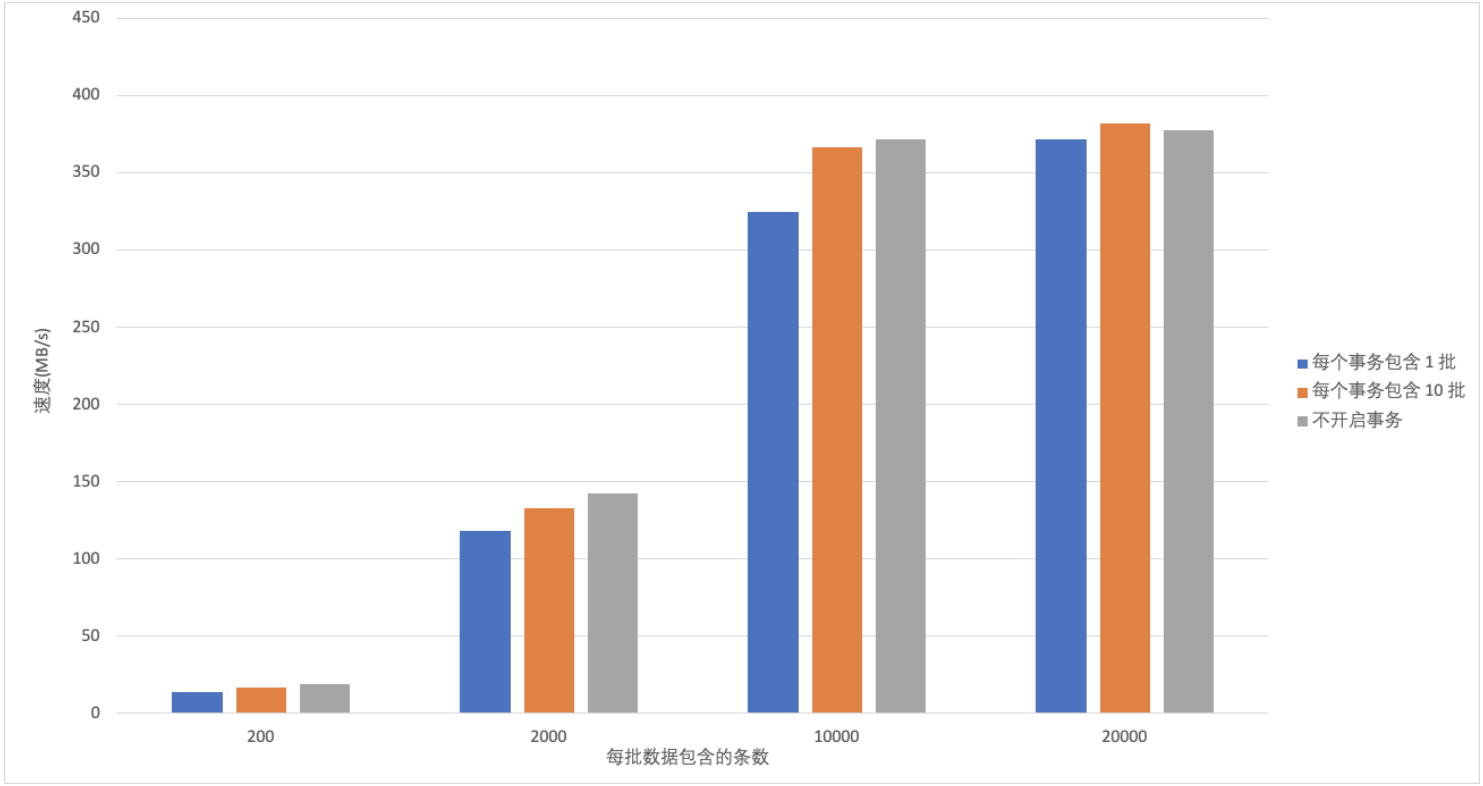
阿里巴巴 Alibaba.com | DataFun.

03
测试结果

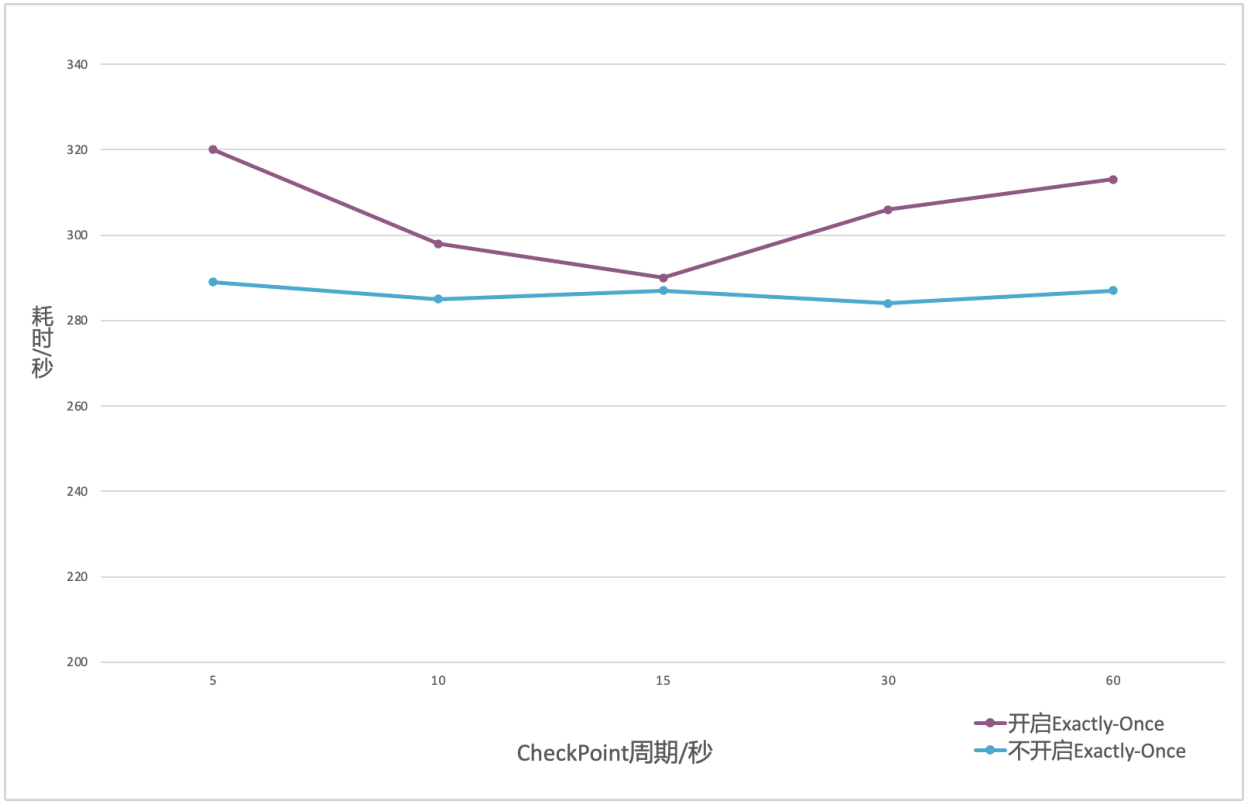
并发对写事务的性能影响



数据大小对写事务性能的影响



Flink Exactly Once 性能



阿里巴巴 Alibaba.com | DataFun.

04

Sharding Key 优化

case1:计算UV场景，将相同 uid 写入到同一 shard 分片

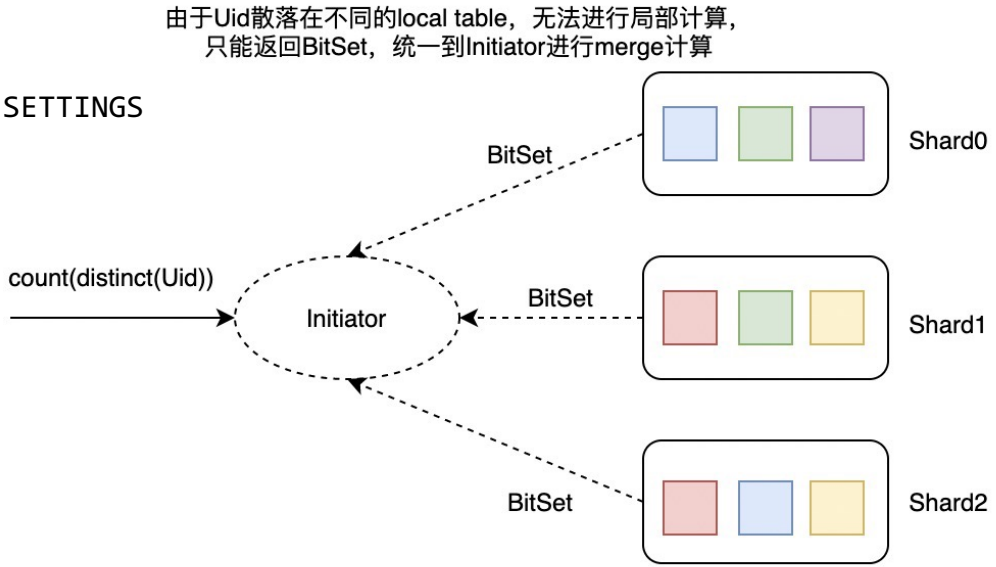
```
CREATE TABLE user_action ON CLUSTER cluster_emr
(
    Uid UInt32,
    Action String
)
ENGINE = ReplicatedMergeTree('/ssb/{layer}-{shard}/user_action', '{replica}') ORDER BY (Uid);

CREATE TABLE user_action_all ON CLUSTER cluster_emr
(
    Uid UInt32,
    Action String
)
ENGINE = Distributed(cluster_emr, 'default', 'user_action', Uid);

INSERT INTO user_action_all(Uid, Action)
SELECT
  number,
  randomPrintableASCII(16)
FROM numbers(100000000);
```

case1:计算UV场景，将相同 uid 写入到同一 shard 分片

```
select uniqExact(Uid) from user_action_all SETTINGS
distributed_group_by_no_merge=0;
```




```
select sum(par_uv) as uv from (
  select uniqExact(Uid) as par_uv
  from user_action_all
) SETTINGS distributed_group_by_no_merge=1;
```

case1:计算UV场景，将相同 uid 写入到同一 shard 分片

```
emr-header-1.cluster-234139 :) select uniqExact(Uid) from user_action_all SETTINGS distributed_group_by_no_merge = 0;

SELECT uniqExact(Uid)
FROM user_action_all
SETTINGS distributed_group_by_no_merge = 0

↘ Progress: 22.43 million rows, 89.72 MB (110.94 million rows/s., 443.75 MB/s.)  22%
┌─uniqExact(Uid)─┐
│      100000000 │
└────────────────┘

1 rows in set. Elapsed: 7.330 sec. Processed 100.00 million rows, 400.00 MB (13.64 million rows/s., 54.57 MB/s.)
```

```
emr-header-1.cluster-234139 :) select sum(par_uv) as uv from (
:-]   select uniqExact(Uid) as par_uv
:-]   from user_action_all
:-] ) SETTINGS distributed_group_by_no_merge = 1;

SELECT sum(par_uv) AS uv
FROM
(
    SELECT uniqExact(Uid) AS par_uv
    FROM user_action_all
)
SETTINGS distributed_group_by_no_merge = 1

┌─────uv─┐
│ 100000000 │
└─────────┘

1 rows in set. Elapsed: 3.003 sec. Processed 100.00 million rows, 400.00 MB (33.30 million rows/s., 133.19 MB/s.)
```

case2: IN 子查询

```
create table orders on cluster cluster_emr (
  uid UInt32,
  date Date,
  skuId UInt32,
  order_revenue UInt8
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/default/orders', '{replica}')
Order by date

CREATE TABLE orders_all ON CLUSTER cluster_emr (
  uid UInt32,
  date Date,
  skuId UInt32,
  order_revenue UInt8
) ENGINE = Distributed(cluster_emr, default, orders, uid)

insert into orders_all (uid, date, skuId, order_revenue)
select
  rand(1)%80000000,
  toDate('2020-01-01')+rand(2)%30,
  rand(3)%1000,
  rand(4)%200
from numbers(300000000)
```

case2: IN 子查询

```
emr-header-1.cluster-238390 :) select count(), sum(order_revenue) from orders_all where date>=toDate('2020-01-21') and order_revenue>=10 and uid global in (
:-]   select distinct(uid) from orders_all
:-]   where date<=toDate('2020-01-20')
:-] )

SELECT
    count(),
    sum(order_revenue)
FROM orders_all
WHERE (date >= toDate('2020-01-21')) AND (order_revenue >= 10) AND (uid GLOBAL IN
(
    SELECT DISTINCT uid
    FROM orders_all
    WHERE date <= toDate('2020-01-20')
))

┌─count()─┬─sum(order_revenue)─┐
│ 87204916 │         9112638793 │
└─────────┴────────────────────┘

1 rows in set. Elapsed: 21.248 sec. Processed 667.46 million rows, 3.37 GB (31.41 million rows/s., 158.63 MB/s.)
```

```
emr-header-1.cluster-238390 :) select count(), sum(order_revenue) from orders_all where date>=toDate('2020-01-21') and order_revenue>=10 and uid in (
:-]   select distinct(uid) from orders
:-]   where date<=toDate('2020-01-20')
:-] )

SELECT
    count(),
    sum(order_revenue)
FROM orders_all
WHERE (date >= toDate('2020-01-21')) AND (order_revenue >= 10) AND (uid IN
(
    SELECT DISTINCT uid
    FROM orders
    WHERE date <= toDate('2020-01-20')
))

┌─count()─┬─sum(order_revenue)─┐
│ 87204916 │         9112638793 │
└─────────┴────────────────────┘

1 rows in set. Elapsed: 2.966 sec. Processed 300.30 million rows, 1.90 GB (101.24 million rows/s., 641.24 MB/s.)
```

case3: Join 查询

```
create table dim_users on cluster cluster_emr (
  uid UInt32,
  age UInt8,
  name String,
  addr String
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/default/dim_users', '{replica}')
Order by uid

CREATE TABLE dim_users_all ON CLUSTER cluster_emr (
  uid UInt32,
  age UInt8,
  name String,
  addr String
) ENGINE = Distributed(cluster_emr, default, dim_users, uid)

insert into dim_users_all (uid, age, name, addr)
select
  number,
  20+rand(1)%20,
  randomPrintableASCII(6),
  randomPrintableASCII(20)
from numbers(80000000)
```

case3: Join 查询

```
select date, sum(order_revenue), avg(age) from orders_all global
join dim_users_all using uid group by date;
```

```
emr-header-1.cluster-238390 :)
emr-header-1.cluster-238390 :) select date, sum(order_revenue), max(age) from orders_all global join dim_users_all using uid group by date;

SELECT
    date,
    sum(order_revenue),
    max(age)
FROM orders_all
GLOBAL INNER JOIN dim_users_all USING (uid)
GROUP BY date

↙ Progress: 113.70 million rows, 4.09 GB (9.22 million rows/s., 331.67 MB/s.) ████████████████████████████████████████████████████████████████████████ 99%
Received exception from server (version 20.8.12):
Code: 241. DB::Exception: Received from localhost:9000. DB::Exception: Memory limit (for query) exceeded: would use 10.05 GiB (attempt to allocate chunk of 2147680448 bytes), maximum: 9.31 GiB: While exec
uting CreatingSetsTransform.

0 rows in set. Elapsed: 13.229 sec. Processed 113.70 million rows, 4.09 GB (8.59 million rows/s., 309.05 MB/s.)

emr-header-1.cluster-238390 :)
```

case3: Join 查询

-- 变成子查询方法，少了对name，addr两个字段的查询，join可以跑完，但是很慢

```
select date, sum(order_revenue),
avg(age) from orders_all global join (
  select uid, age from dim_users_all
) as t using uid
group by date
```

```
SELECT
    date,
    sum(order_revenue),
    avg(age)
FROM orders_all
GLOBAL INNER JOIN
(
    SELECT
        uid,
        age
    FROM dim_users_all
) AS t USING (uid)
GROUP BY date
```

date	sum(order_revenue)	avg(age)
2020-01-01	994920069	29.500124749887725
2020-01-02	995020722	29.499393052731264
2020-01-03	995615987	29.500416103682284
2020-01-04	994821841	29.49657777493705
2020-01-05	994719766	29.499153035861
2020-01-06	994743751	29.50074107244062
2020-01-07	995467728	29.49988248366843
2020-01-08	994542276	29.501070465252017
2020-01-09	995521820	29.498968329731532
2020-01-10	994569113	29.49885467373429
2020-01-11	994864744	29.498308566830143
2020-01-12	995085067	29.497976768280527
2020-01-13	995150500	29.502989053334478
2020-01-14	995098243	29.50262326076187
2020-01-15	994756828	29.503588065315462
2020-01-16	995107620	29.500499857240772
2020-01-17	995049328	29.499036160721875
2020-01-18	994679184	29.497565782578047
2020-01-19	995436821	29.501176737777403
2020-01-20	994533632	29.499468647289522
2020-01-21	995405149	29.49742075527755
2020-01-22	994881257	29.502520245698836
2020-01-23	995032419	29.50092529605524
2020-01-24	994133081	29.499583259740408
2020-01-25	995168149	29.499369998235995
2020-01-26	994503337	29.50147485986581
2020-01-27	995022505	29.49817453335268
2020-01-28	995462950	29.501151955455256
2020-01-29	995493813	29.499229766027
2020-01-30	994674709	29.50051307049076

30 rows in set. Elapsed: 25.276 sec. Processed 780.00 million rows, 4.50 GB (30.86 million rows/s., 178.04 MB/s.)

case3: Join 查询

```
-- local join查询
select date, sum(order_revenue),
avg(age) from orders_all join dim_users
using uid group by date;
```

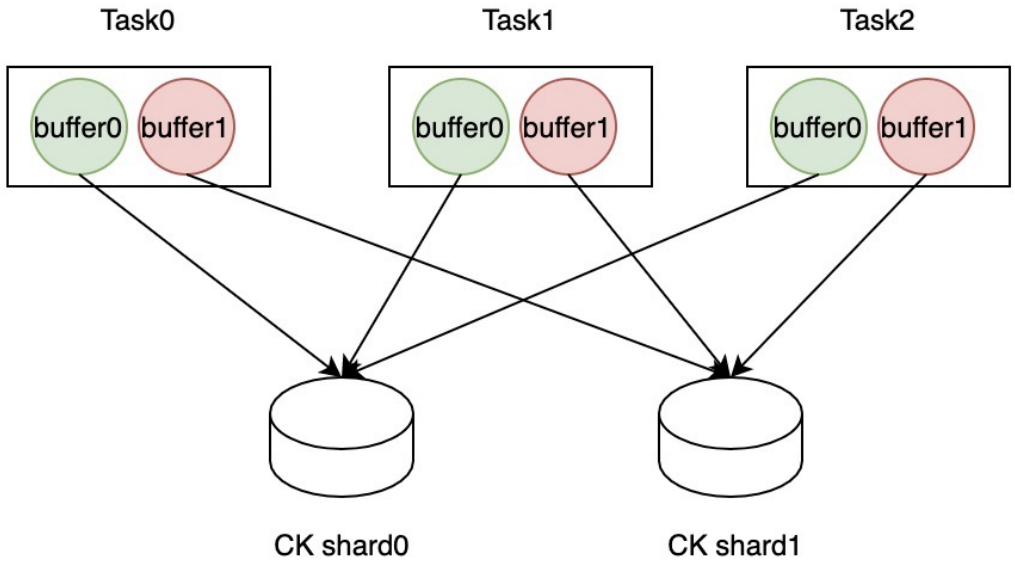
```
SELECT
    date,
    sum(order_revenue),
    avg(age)
FROM orders_all
INNER JOIN dim_users USING (uid)
GROUP BY date
```

date	sum(order_revenue)	avg(age)
2020-01-01	994920069	29.500124749887725
2020-01-02	995020722	29.499393052731264
2020-01-03	995615987	29.500416103682284
2020-01-04	994821841	29.49657777493705
2020-01-05	994719766	29.499153035861
2020-01-06	994743751	29.50074107244062
2020-01-07	995467728	29.49988248366843
2020-01-08	994542276	29.501070465252017
2020-01-09	995521820	29.498968329731532
2020-01-10	994569113	29.49885467373429
2020-01-11	994864744	29.498308566830143
2020-01-12	995085067	29.497976768280527
2020-01-13	995150500	29.502989053334478
2020-01-14	995098243	29.50262326076187
2020-01-15	994756828	29.503588065315462
2020-01-16	995107620	29.500499857240772
2020-01-17	995049328	29.499036160721875
2020-01-18	994679184	29.497565782578047
2020-01-19	995436821	29.501176737777403
2020-01-20	994533632	29.499468647289522
2020-01-21	995405149	29.49742075527755
2020-01-22	994881257	29.502520245698836
2020-01-23	995032419	29.50092529605524
2020-01-24	994133081	29.499583259740408
2020-01-25	995168149	29.499369998235995
2020-01-26	994503337	29.50147485986581
2020-01-27	995022505	29.49817453335268
2020-01-28	995462950	29.501151955455256
2020-01-29	995493813	29.499229766027
2020-01-30	994674709	29.50051307049076

30 rows in set Elapsed: 5.659 sec Processed 380.00 million rows, 2.50 GB (67.15 million rows/s., 441.78 MB/s.)

JDBC Sharding Key 优化

- 直接在task级别计算shardId = key % shardNum，打到对应的local shard上
- 需要task的内存比较多，因为每个task需要hold outputBuffer * shardNum，在shardNum不大时可接受



阿里巴巴 Alibaba.com | DataFun.

05
未来规划

未来规划

- 支持分布式事务
- Sharding Key 支持 Flink Exactly Once 导入
- Meta Server
- ...

EMR OLAP(ClickHouse&...

499人

扫一扫群二维码，立刻加入该群。

非常感谢您的观看

阿里巴巴 Alibaba.com | DataFun.