腾讯云数据仓库 TCHouse-C

实践教程

腾讯云

【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

实践教程
使用 MySQL 关联 HBase 维表数据到 TCHouse-C
ClickHouse 内核使用建议与规范

实践教程
使用 MySQL 关联 HBase 维表数据到 TCHouse-C

最近更新时间：2023-09-05 11:25:16

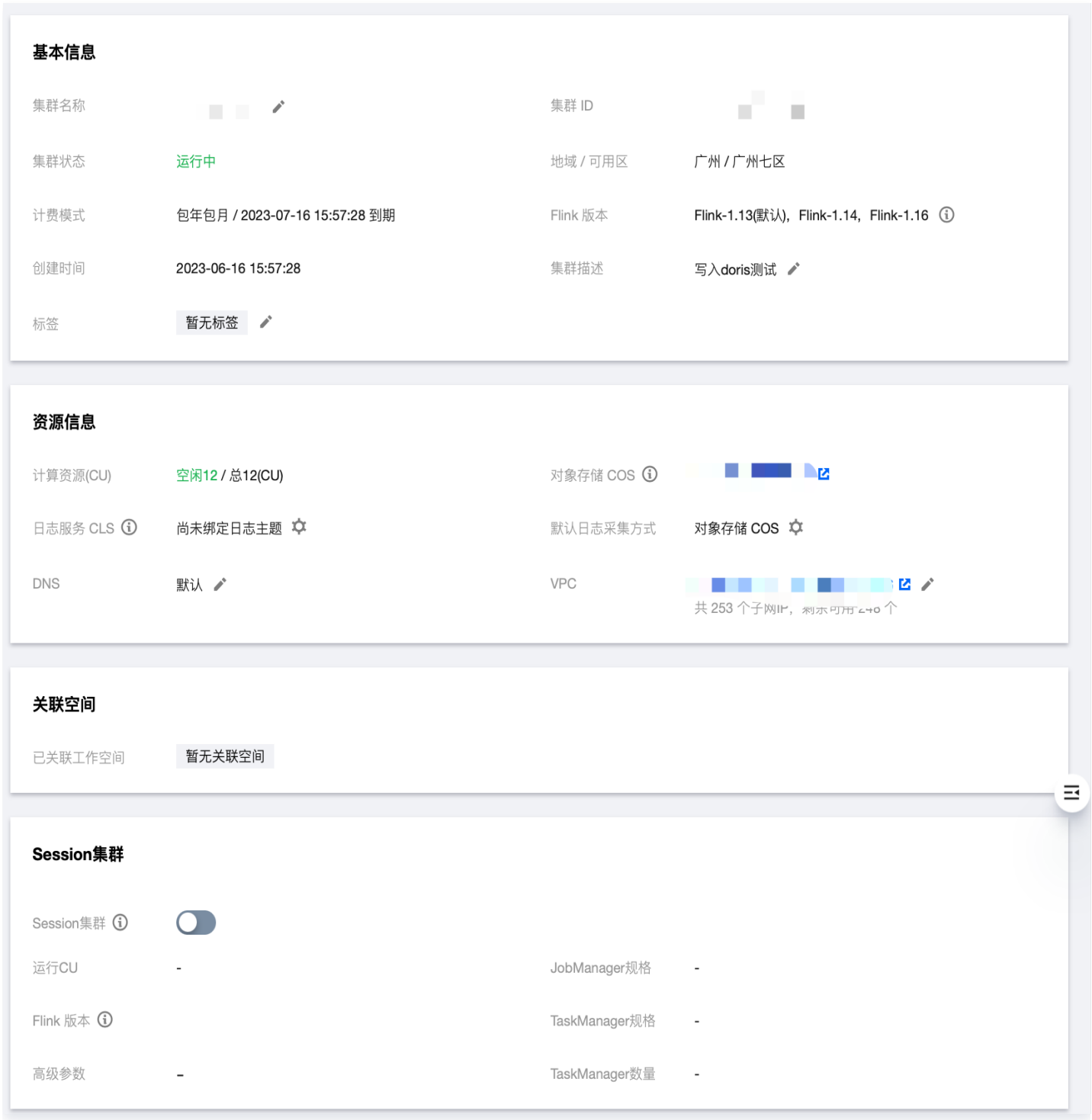
本文介绍了结合 MySQL 数据库、流计算 Oceanus、HBase 以及云数据仓库 TCHouse-C 来构建实时数仓，并通过流计算 Oceanus 读取 MySQL 数据、关联 HBase 中的维表，最终将数据存入云数据仓库 TCHouse-C 进行指标分析，实现完整实时数仓的全流程操作指导。

环境搭建

创建 Oceanus 集群

在 流计算 Oceanus 控制台的**计算资源 > 新建**，创建集群，选择地域、可用区、VPC、存储、日志、设置初始密码等。创建完后的集群如下：

> **说明**
> - 若未使用过 VPC、日志、存储这些组件，需要先进行创建。
> - VPC 及子网需要和下面的 MySQL、EMR 集群使用同一个，否则需要手动打通（如对等连接）。



创建 VPC 私有网络

私有网络是一块您在腾讯云上自定义的逻辑隔离网络空间，在构建 MySQL、EMR、TCHouse-C 集群等服务时选择的网络必须保持一致，网络才能互通，否则需要使用对等连接、VPN 等方式打通网络。

登录 私有网络 控制台，选择**私有网络 > +新建**，新建私有网络。

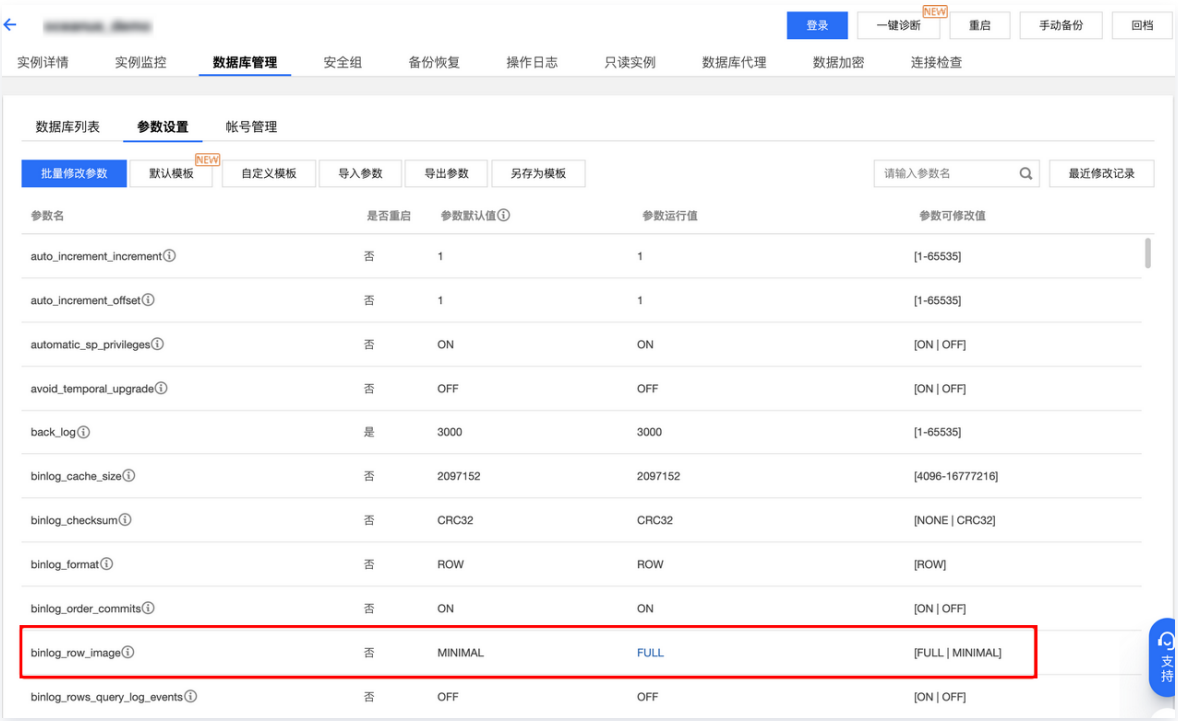
创建云数据库 MySQL 服务

云数据库 MySQL（TencentDB for MySQL）是腾讯云基于开源数据库 MySQL 专业打造的高性能分布式数据存储服务，让用户能够在云中更轻松地设置、操作和扩展关系数据库。

1. 登录 云数据库 TencentDB 控制台，选择**实例列表 > 新建**，新建实例。

2. **新建 MySQL 服务时，网络需要选择之前创建的。**

3. 创建完 MySQL 服务后，需要修改 binlog 参数，如图修改为 FULL（默认值为 MINIMAL）。



4. 修改完参数后，登录 MySQL 创建示例所需要的数据库和数据库表。

创建数据库 mysqltestdb

1. 登录 MySQL 创建示例所需要的数据库。

2. 打开 SQL 窗口或者单击可视化页面创建数据库及表。

新建数据库

```
create database mysqltestdb;
```

在新建的数据库上新建表 student：

```
create table `student` (
  `id` int(11) not null auto_increment comment '主键id',
  `name` varchar(10) collate utf8mb4_bin default '' comment '名字',
  `age` int(11) default null comment '年龄',
  `create_time` timestamp null default current_timestamp comment '数据创建时间',
  primary key (`id`)
) engine=innodb auto_increment=4 default charset=utf8mb4 collate=utf8mb4_bin
row_format=compact comment='学生表'
```

Student 表中插入数据

```
insert into mysqltestdb.student(id,name,age) values(1,“xiaomin”,20);
```

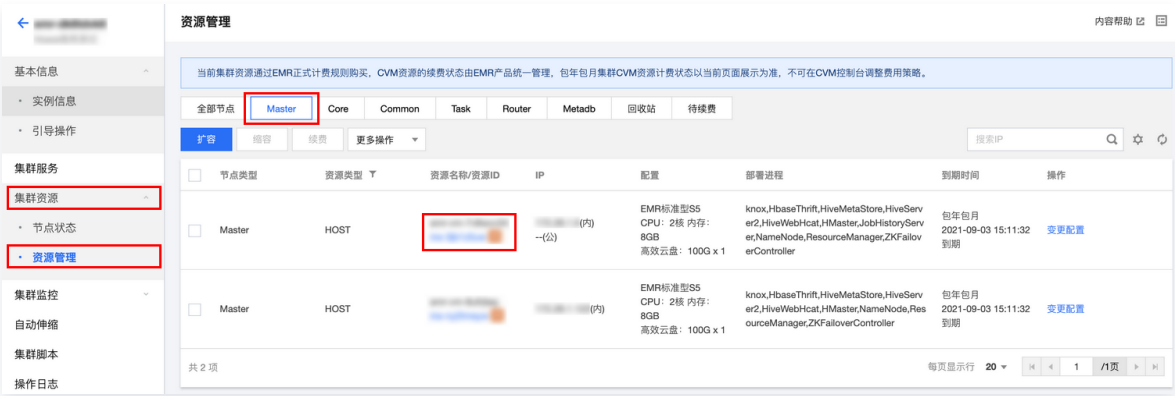
创建 EMR 集群

弹性 MapReduce 是云端托管的弹性开源泛 Hadoop 服务，支持 Spark、HBase、Presto、Flink、Druid 等大数据框架，本次示例主要需要使用 HBase 组件。

1. 登录 弹性 MapReduce 控制台，选择**集群列表 > 新建集群**，开始新建集群，具体可参考 创建 EMR 集群。新建集群时，需选择安装 HBase 组件。

如果是生产环境，服务器配置可根据实际情况选择。网络需要选择之前创建好的 VPC 网络，始终保持服务组件在同一 VPC 下。

2. 在集群列表中，单击新建的集群 ID/名称，进入集群详情页。选择**集群资源 > 资源管理**，即进入 HBase 的 Master 节点。



3. 进入 **云服务器控制台**，搜索 EMR **实例 ID**，然后单击**登录**进入服务器。

4. 创建 Hbase 表。

```
# 进入HBase命令
root@172~# hbase shell
```

进入 hbase shell，并新建表：

```
# 建表语句
 create ‘dim_hbase’, ‘cf’

# 插入数据
 put ‘dim_hbase’,’1’,’cf:name’,’MingDeSchool’
```

创建云数据仓库 TCHouse-C

新建集群

1. 登录 云数据仓库 TCHouse-C 控制台，选择**集群列表 > 新建集群**，新建集群。

2. 选择网络选择之前新建的 VPC 网络（依然保证各服务在同一网络）。

登录 TCHouse-C

在之前新建的 EMR 下选择一台云服务器单击**登录**，最好选择带有外网 IP 的节点。

安装 ClickHouse 客户端

在此机器上安装 ClickHouse 客户端，clickhouse-client 安装可参见 快速入门 。

登录客户端

登录客户端示例如下：

```
clickhouse-client -h用户自己的ClickHouse服务IP --port 9000
```

新建数据库：

```
create database testdb on cluster default_cluster;
```

新建表：

```
CREATE TABLE testdb.student_school on cluster default_cluster (
`id` Int32,
`name` Nullable(String),
`school_name` Nullable(String),
`Sign` Int8
) ENGINE = ReplicatedCollapsingMergeTree('/clickhouse/tables/{layer}-{shard}/testdb/
student_school, '{replica}', Sign) ORDER BY id;
```

数据清洗和运算加工

数据准备

按照上面的操作创建表，并向 MySQL 和 HBase 表中插入数据。

创建 Flink SQL 作业

在流计算 Oceanus 控制台创建 SQL 作业，选择响应的内置 Connector。

Source 端

MySQL-CDC Source：

```
--学生信息作为cdc源表
CREATE TABLE `student` (
  `id` INT NOT NULL,
  `name` varchar,
  `age` INT,
  proc_time AS PROCTIME(),
  PRIMARY KEY (`ID`) NOT ENFORCED
) WITH (
  'connector' = 'mysql-cdc',
  -- 必须为 'mysql-cdc'
  'hostname' = 'YoursIp',
  -- 数据库的 IP
  'port' = '3306',
  -- 数据库的访问端口
  'username' = '用户名',
  -- 数据库访问的用户名（需要提供 SHOW DATABASES, REPLICATION SLAVE, REPLICATION
CLIENT, SELECT, RELOAD 权限）
  'password' = 'YoursPassword,
  -- 数据库访问的密码
  'database-name' = 'mysqltestdb',
  -- 需要同步的数据库
  'table-name' = 'student' -- 需要同步的数据表名
);
```

HBase 维表：

```
--示例使用school学校信息作为维表
CREATE TABLE dim_hbase (
  rowkey STRING,
  cf ROW <school_name STRING>,  -- 如果有多个列簇，写法 cf Row<age INT,name String>
  PRIMARY KEY (rowkey) NOT ENFORCED
) WITH (
  'connector' = 'hbase-1.4',
  'table-name' = 'dim_hbase',
  'zookeeper.quorum' = '用户自己的hbase服务器zookeeper地址，多个用逗号隔开'
);
```

Sink 端

创建到 TCHouse-C 的创建表语句。

```
--关联后存入clickhouse表
CREATE TABLE `student_school` (
  stu_id INT,
  stu_name STRING,
```

```
  school_name STRING,
  PRIMARY KEY (`id`) NOT ENFORCED
) WITH (
  -- 指定数据库连接参数
  'connector' = 'clickhouse',
  'url' = 'clickhouse://yourIP:8123',
  -- 如果TCHouse-C集群未配置账号密码可以不指定
  --'username' = 'root',
  --'password' = 'root',
  'database-name' = 'testdb',
  'table-name' = ' student_school ',
  'table.collapsing.field' = 'Sign'
);
```

进行逻辑运算

此示例中，只进行了简单的 Join 没有进行复杂的运算。详细运算逻辑可参见 Oceanus 运算符和内置函数 或者 Flink 官网 Flink SQL 开发 。

```
INSERT INTO
  student_school
SELECT
  student.id as stu_id,
  student.name as stu_name,
  dim_hbase.cf.school_name
FROM
  student
  JOIN dim_hbase for SYSTEM_TIME as of student.proc_time
    ON CAST(student.id AS STRING) = dim_hbase.rowkey;
```

结果验证

在数据库中查询数据是否正确。

```
select * from testdb.student_school;
```

ClickHouse 内核使用建议与规范

最近更新时间：2023-09-15 15:03:41

使用 ClickHouse 时需注意以下规范：

写入规范

1. **攒批写入：**ClickHouse 必须攒批写入，至少 1000 条/批，建议 5k – 10w 一批写入ClickHouse，每一次写入都会在底层生成 1 个或者多个 part 存储目录，后台任务自动合并小 part 到一个大 part ，如果写入频次过高会出现 part 过多，merge 速度跟不上导致写入失败报错：
   `Too many parts(301). Merges are processing significantly slower than inserts` 。
2. **减少分布式表直接写入：**为了提高写入和查询性能，应尽可能直接写入本地表，而不是分布式表。写分布式表最终也会转发给本地表，但是分布式表存在写放大以及异步落盘消耗 IO 的问题，写入性能较差。
3. **约束数据一致性：**ClickHouse 不支持数据写入的事务保证，因此需要通过外部导入数据模块来控制数据的幂等性。例如，如果某个批次的数据导入异常，可以删除对应的分区数据或清理导入的数据，然后重新导入该分区或批次的数据。也可以使用去重引擎（replacingMergetree）来保证最终一致性。
4. **大规模数据写入：**如果需要进行大规模数据写入，建议提前拆分数据，并按节点均匀地写入 ClickHouse 的各个节点。如果存在特定的分布规则，可以在业务侧进行 hash 计算。
5. **一次只写入一个分区数据：**为了避免写入性能下降和目录数量过多的问题，应该一次只写入一个分区的数据。如果一批写入数据跨多个分区，会导致底层产生多个 part 文件，消耗更多的 merge 性能，并且不利于幂等控制。

查询规范

单表查询

1. 高频过滤和点查询字段使用索引加速。
2. 避免使用 `select *` 语句，应该明确需要查询的字段，只查询必要的字段。ClickHouse 底层是列式存储，查询的耗时与查询的字段大小和数量成线性关系。
3. 当查询千万以上的数据集时，建议使用 `where` 条件和 `limit` 语句来配合 `order by` 查询，以提高查询效率。
4. select {tablename} final 能够实现查询（read on merge），但是会减慢查询速度，需要有针对性使用。
5. 尽量按分区过滤裁剪，通过指定分区字段可以减少底层数据库扫描的文件数量，提高查询性能。
6. 谨慎使用 delete 和 update 的 mutation 操作。ClickHouse 的 update 和 delete 是异步进行的，并且会重写 `where` 条件过滤出的数据 part ，是非常重的操作，可能会消耗较多系统资源。此外，update 和 delete 是按照 part 逐个执行，不会保证整体执行的原子性。
7. 如果对唯一性要求不高，可以采用近似去重 uniqCombined 来优化去重逻辑，从而提高十倍的查询性能。如果查询允许有误差，可以使用 uniqCombined 替代，否则应该继续使用 distinct 语法。使用 distinct 会对查询性能有一定影响。

多表关联

1. 为了避免 Join 操作和 shuffle ，应尽量使用 Flat 大宽表结构代替多表 Join 。

2. 控制 Join 的表数量，尽量保持在3个及以下。
3. 如果查询字段出自单表，可考虑将 `Join` 改为 `in` 查询 ，CK 中的 `in` 查询支持单字段和 tuple ，例如 `SELECT name FROM tab_a WHERE id IN （SELECT id FROM tab_b WHERE name = 'xx '）`。
4. 多表 Join 最好改为两表 Join 和子查询形式。
5. 两表 Join 需大表 Join 小表（数据量控制在百万 - 千万行级别），小表 Join 小表，不允许大表 Join 大表；Join 时大表需在左边，小表在右边。
6. 建议使用列裁剪、分区裁剪，Join 前进行条件过滤，尽可能降低 Join 数据量。
7. 若 Join 时右表为子查询或者分布式表且数据不大，可以采用 GLOBAL JOIN 避免读放大，需要注意的是，GLOBAL JOIN 会触发数据在节点之间传播，占用部分网络流量。如果数据量较大，同样会带来性能损失。

建表规范

1. 高可用集群不可创建非 Replicated 表，非高可用集群不可创建 Replicated 表。
2. 如果对数据最终一致性有强要求，需要使用 ReplacingMergeTree 或者 CollapsingMergeTree 引擎，并定期进行 optimize 或使用 select {tablename} final 实现最终去重。
3. 在规划分区时，应该合理规划分区个数，并尽可能利用分区。一张表分区数不建议超过 1000 个，可以在查询时有效帮助进行数据过滤，使用得当可以提升数倍查询性能，通常按天分区是比较普遍的做法。分区也不建议过多，因为 ClickHouse 不同分区的数据不会合并，容易导致 part 过多，从而导致查询和重启变得很慢。
4. 建表时尽可能提前规划好表字段，并尽量避免删改字段。删改字段会重写整个表的全量数据，对于大表会消耗大量资源，执行时间可能很长。此外，删改字段期间也容易阻塞其他 DDL 语句，影响表的 merge 操作。如果中途出错，有概率会导致不可预知的数据一致性问题。
5. 禁止修改索引列，对索引列的修改会导致现有索引失效，触发重建索引，期间查询数据不准确。
6. 约束 COS 上存储数据的量，尽可能避免对冷分区进行写入和 mutation 操作。COS 单个桶大约只有1GB的带宽，远低于多节点的本地盘和云盘性能，且网络延迟比较高。如果 COS 上存储过多数据，会严重影响查询效率。针对 COS 分区的写入时，会触发 COS 分区进行 merge ，merge 效率也会降低甚至会影响本地盘的数据操作。