数据湖探索

Hudi SQL 语法参考

文档版本 01

发布日期 2024-12-25

华为技术有限公司

版权所有 © 华为技术有限公司 2024。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 Hudi 表概述......1
1.1 Hudi 表使用约束......1
1.2 Hudi 查询类型......2
1.3 Hudi 存储结构......5
2 DLI Hudi 元数据......6
3 DLI Hudi 开发规范......8
3.1 Hudi 开发规范概述......8
3.2 Hudi 数据表设计规范......8
3.2.1 Hudi 表模型设计规范......9
3.2.2 Hudi 表索引设计规范......10
3.2.3 Hudi 表分区设计规范......12
3.3 Hudi 数据表管理操作规范......13
3.3.1 Hudi 数据表 Compaction 规范......13
3.3.2 Hudi 数据表 Clean 规范......15
3.3.3 Hudi 数据表 Archive 规范......16
3.4 Spark on Hudi 开发规范......16
3.4.1 SparkSQL 建表参数规范......17
3.4.2 Spark 增量读取 Hudi 参数规范......17
3.4.3 Spark 异步任务执行表 compaction 参数设置规范......18
3.4.4 Spark 表数据维护规范......18
3.5 Bucket 调优示例......18
3.5.1 创建 Bucket 索引表调优......19
3.5.2 Hudi 表初始化......20
3.5.3 实时任务接入......21
3.5.4 离线 Compaction 配置......22
4 DLI 中使用 Hudi 开发作业......23
4.1 在 DLI 使用 Hudi 提交 Spark SQL 作业......23
4.2 在 DLI 使用 Hudi 提交 Spark Jar 作业......24
4.3 在 DLI 使用 Hudi 提交 Flink SQL 作业......27
4.4 使用 HetuEngine on Hudi......28
5 DLI Hudi SQL 语法参考......29
5.1 Hudi DDL 语法说明......29

5.1.1 CREATE TABLE 29
5.1.2 DROP TABLE 32
5.1.3 SHOW TABLE 33
5.1.4 TRUNCATE TABLE 34
5.2 Hudi DML 语法说明 34
5.2.1 CREATE TABLE AS SELECT 34
5.2.2 INSERT INTO 36
5.2.3 MERGE INTO 37
5.2.4 UPDATE 39
5.2.5 DELETE 40
5.2.6 COMPACTION 41
5.2.7 ARCHIVELOG 42
5.2.8 CLEAN 43
5.2.9 CLEANARCHIVE 43
5.3 Hudi CALL COMMAND 语法说明 44
5.3.1 CLEAN_FILE 44
5.3.2 SHOW_TIME_LINE 45
5.3.3 SHOW_HOODIE_PROPERTIES 47
5.3.4 ROLL_BACK 47
5.3.5 CLUSTERING 48
5.3.6 CLEANING 49
5.3.7 COMPACTION 50
5.3.8 SHOW_COMMIT_FILES 51
5.3.9 SHOW_FS_PATH_DETAIL 53
5.3.10 SHOW_LOG_FILE 54
5.3.11 SHOW_INVALID_PARQUET 55
5.4 Schema 演进语法说明 56
5.4.1 ALTER COLUMN 57
5.4.2 ADD COLUMNS 58
5.4.3 RENAME COLUMN 59
5.4.4 RENAME TABLE 60
5.4.5 SET 61
5.4.6 DROP COLUMN 62
5.5 配置 Hudi 数据列默认值 62
6 Spark datasource API 语法参考 64
6.1 API 语法说明 64
6.2 Hudi 锁配置说明 67
7 数据管理维护 69
7.1 Hudi Compaction 操作说明 69
7.2 Hudi Clean 操作说明 70
7.3 Hudi Archive 操作说明 71
7.4 Hudi Clustering 操作说明 72

8 Hudi 常见配置参数..75

1 Hudi 表概述

1.1 Hudi 表使用约束

Hudi 表类型

- **Copy On Write**

  写时复制表也简称COW表，使用parquet文件存储数据，内部的更新操作需要通过重写原始parquet文件完成。

  – 优点：读取时，只读取对应分区的一个数据文件即可，较为高效。

  – 缺点：数据写入的时候，需要复制一个先前的副本再在其基础上生成新的数据文件，这个过程比较耗时。且由于耗时，读请求读取到的数据相对就会滞后。

- **Merge On Read**

  读时合并表也简称MOR表，使用列格式parquet和行格式Avro两种方式混合存储数据。其中parquet格式文件用于存储基础数据，Avro格式文件（也可叫做log文件）用于存储增量数据。

  – 优点：由于写入数据先写delta log，且delta log较小，所以写入成本较低。

  – 缺点：需要定期合并整理compact，否则碎片文件较多。读取性能较差，因为需要将delta log和老数据文件合并。

表 1-1 两种表类型的 trade-off

Trade-off	CopyOnWrite	MergeOnRead
Data Latency（数据时延）	高	低
Query Latency（查询时延）	低	高
Update cost (I/O)（更新时（I/O）开销）	高（重写整个parquet）	低
Parquet File Size（Parquet文件大小）	小（更新时（I/O）开销大）	大（更新时开销小）

版权所有 © 华为技术有限公司

Trade-off	CopyOnWrite	MergeOnRead
Write Amplification（写放大）	高	低（取决于compaction策略）

Hudi 表使用约束与限制

- Hudi支持使用Spark SQL操作Hudi的DDL/DML的语法。但在使用DLI提供的元数据提交SparkSQL作业时，部分直接操作OBS路径的SQL语法暂不支持，详细说明请参考**DLI Hudi SQL语法参考**。
- 不支持在HetuEngine中写Hudi表，以及修改Hudi表结构，仅支持读Hudi表。
- 创建Hudi表时，必须且需要正确配置 primaryKey 和 preCombineField，否则存在数据最终表现与预期不一致的风险。
- **使用由DLI提供的元数据服务时，不支持创建DLI表，只支持创建OBS表，即必须通过LOCATION参数配置表路径。**
- 使用由LakeFormation提供的元数据服务时，创建内表和外表均支持。需要注意：在DROP内表时，数据也会被同步删除。
- 在提交Spark SQL或Flink SQL作业时，无需手动配置Hudi的hoodie.write.lock.provider 配置项，但在提交Spark jar作业时必须手动配置，请见 **Hudi锁配置说明** 一节。
- Hudi和队列计算引擎的版本对应关系为：

计算引擎	版本	Hudi版本
Spark	3.3.1	0.11.0
Flink	1.15	0.11.0
Hetu	2.1.0	0.11.0

如何判断队列支持的计算引擎版本：首先进入DLI的控制台界面，点击左侧菜单栏的”资源管理”-> ”队列管理”。在队列管理的界面筛选并选中需要查询的队列，随后点击窗口底部的窗格，展开隐藏的队列详情页面，在支持版本即可查看可用的计算引擎版本。对于SQL队列，无法切换版本，查看默认版本即可判断当前使用的计算引擎版本。

1.2 Hudi 查询类型

快照查询

快照查询（Snapshot Queries）可以读到最新的commit/compaction产生的快照。对于MOR表，还会在查询中合并最新的delta log文件的内容，使读取的数据近实时。

增量查询

增量查询（Incremental Queries）只会查询到给定的commit/compaction之后新增的数据。

 版权所有 © 华为技术有限公司

读优化查询

读优化查询（Read Optimized Queries）是针对MOR表进行的优化，只会读取最新的commit/compaction产生的快照（不包含delta log文件）。

表 **1-2** 实时查询和读优化查询的 trade-off

Trade-off	**实时查询**	**读优化查询**
Data Latency（数据时延）	低	高
Query Latency（查询时延）	只对于MOR表，高（合并parquet + delta log)	低 （读取parquet文件性能）

COW 表查询

- 实时视图读取（SparkSQL为例）：直接读取元数据服务里面存储的Hudi表即可，**${table_name}**表示表名称。

```
select (字段 or 聚合函数) from ${table_name};
```

- 实时视图读取（Spark jar作业为例）：

  Spark jar作业可以通过两种方式来读取Hudi表：Spark datasource API 或者通过SparkSession 提交 SQL。

  配置项 hoodie.datasource.query.type 需要配置为 snapshot（snapshot同时也是默认值，因此可以缺省）。

```
object HudiDemoScala {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .enableHiveSupport()
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .config("spark.sql.extensions", "org.apache.spark.sql.hudi.HoodieSparkSessionExtension")
      .appName("HudiIncrementalReadDemo")
      .getOrCreate();

    // 1. 通过Spark datasource API 读取
    val dataFrame = spark.read.format("hudi")
      .option("hoodie.datasource.query.type", "snapshot") // snapshot同时也作为默认值，因此可以缺省
本配置项
      .load("obs://bucket/to_your_table"); // 指定读取的hudi表路径，DLI仅支持使用OBS路径
    dataFrame.show(100);

    // 2. 通过SparkSession 提交 SQL，需要对接元数据服务。
    spark.sql("select * from ${table_name}").show(100);
  }
}
```

- 增量视图读取（以SparkSQL为例）：

  首先配置

```
hoodie.${table_name}.consume.mode=INCREMENTAL
hoodie.${table_name}.consume.start.timestamp=开始Commit时间
hoodie.${table_name}.consume.end.timestamp=结束Commit时间
```

  随后执行SQL

```
select (字段 or 聚合函数) from ${table_name} where `_hoodie_commit_time`>'开始Commit时间' and
`_hoodie_commit_time`<='结束Commit时间' //这个过滤条件必须带。
```

- 增量视图读取（Spark jar 作业为例）：

  配置项 hoodie.datasource.query.type 需要配置为 incremental。

版权所有 © 华为技术有限公司

```
object HudiDemoScala {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .enableHiveSupport()
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .config("spark.sql.extensions", "org.apache.spark.sql.hudi.HoodieSparkSessionExtension")
      .appName("HudiIncrementalReadDemo")
      .getOrCreate();

    val startTime = "20240531000000";
    val endTime = "20240531000000";
    spark.read.format("hudi")
      .option("hoodie.datasource.query.type", "incremental") // 指定查询类型为增量查询
      .option("hoodie.datasource.read.begin.instanttime", startTime)  // 指定初始增量拉取commit
      .option("hoodie.datasource.read.end.instanttime", endTime)  // 指定增量拉取结束commit
      .load("obs://bucket/to_your_table")  // 指定读取的hudi表路径
      .createTempView("hudi_incremental_temp_view");  // 注册为spark临时表
    // 结果必须根据startTime和endTime进行过滤，如果没有指定endTime，则只需要根据startTime进行过滤
    spark.sql("select * from hudi_incremental_temp_view where `_hoodie_commit_time`>'20240531000000' and `_hoodie_commit_time`<='20240531321456'")
      .show(100, false);
  }
}
```

- 读优化查询：COW表读优化查询等同于快照查询。

MOR 表查询

在Spark SQL作业中使用元数据服务，或者配置了HMS同步参数，在创建MOR表后，会额外同步创建：“表名_rt”和“表名_ro”两张表。查询后缀为rt的表等同于实时查询，查询后缀为ro的表代表读优化查询。例如：通过Spark SQL创建hudi表名为**${table_name}**， 同步元数据服务后，数据库中多出两张表分别为**${table_name}**_rt和**${table_name}**_ro。

- 实时视图读取（SparkSQL为例）：直接读取相同数据库中后缀为_rt的hudi表即可。
  ```
  select count(*) from ${table_name}_rt;
  ```
- 实时视图读取（Spark jar作业为例）：与COW表操作一致，请参考COW表相关操作。
- 增量视图读取（Spark SQL作业为例）：与COW表操作一致，请参考COW表相关操作。
- 增量视图读取（Spark jar作业为例）：与COW表操作一致，请参考COW表相关操作。
- 读优化视图读取（Spark jar作业为例）：

  配置项 hoodie.datasource.query.type 需要配置为 read_optimized。

```
object HudiDemoScala {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder
      .enableHiveSupport
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .config("spark.sql.extensions", "org.apache.spark.sql.hudi.HoodieSparkSessionExtension")
      .appName("HudiIncrementalReadDemo")
      .getOrCreate
    spark.read.format("hudi")
      .option("hoodie.datasource.query.type", "read_optimized") // 指定查询类型为读优化视图
      .load("obs://bucket/to_your_table") // 指定读取的hudi表路径
      .createTempView("hudi_read_optimized_temp_view")
    spark.sql("select * from hudi_read_optimized_temp_view").show(100)
  }
}
```

1.3 Hudi 存储结构

Hudi在写入数据时会根据设置的存储路径、表名、分区结构等属性生成Hudi表。

在DLI环境，Hudi表的数据文件存储在OBS上，因此可以通过查看OBS文件检查。

如下，展示了Hudi 多级分区COW表存储结构的示意。

```
hudi_table
├── .hoodie  // 元数据文件夹
│   ├── .aux
│   ├── .cleanData
│   ├── .schema
│   ├── .temp
│   ├── hoodie.properties  // Hudi表元数据存储文件
│   ├── 20231030232758468.commit.requested  // COW表写入记录元数据（代表计划）
│   ├── 20231030232758468.inflight  // COW表写入记录元数据（代表执行中）
│   ├── 20231030232758468.commit  // COW表写入记录元数据（代表完成）
│   └── ...
├── 2023  // 多级分区目录
│   ├── 10
│   │   ├── 30
│   │   │   ├── .hoodie_partition_metadata
│   │   │   ├── 425388c7-1a84-427e-a7b5-18e6745389b5_1-4-0_20231030232758468.parquet
│   │   │   ├── 425388c7-1a84-427e-a7b5-18e6745389b5_1-4-0_20231030524543412.parquet
│   │   │   ├── ...
│   │   │   └── 425388c7-1a84-427e-a7b5-18e6745389b5_1-4-0_20231030744231276.parquet
│   │   └── ...
└── ...
```

2 DLI Hudi 元数据

DLI Hudi 元数据说明

创建Hudi表时会在元数据仓创建表的相关元数据信息。

Hudi支持对接DLI元数据和Lakeformation元数据（仅Spark 3.3.1及以上版本支持对接Lakeformation元数据），对接方式与Spark一致。

- DLI元数据可在数据湖探索管理控制台的“数据管理 > 库表管理”中查看。
- Lakeformation元数据可在湖仓构建Lakeformation服务的管理控制台中查看。

相关操作

- DLI SQL队列对接DLI元数据方法：
  a. 在DLI管理控制台的SQL编辑器页面的“数据目录”中选择“dli”。
  b. 在“数据库”选项中选择要对接的DLI元数据中的数据库，即可对接到DLI元数据。
- DLI通用队列对接DLI元数据方法：
  请参考**使用Spark作业访问DLI元数据**。
- DLI SQL队列对接Lakeformation元数据方法：
  参考**DLI对接LakeFormation**。
- DLI通用队列对接Lakeformation元数据方法：
  参考**DLI对接LakeFormation**。
- DLI元数据权限管理
  可通过DLI SQL权限管理或者IAM鉴权管理DLI元数据的权限
  – DLI SQL权限管理：
    i. 在“数据湖探索 > 数据管理 > 库表管理”页面，搜索要授权的库/表。
    ii. 单击表操作列的“权限管理”，即可查看当前库/表授权信息或者新增授权。
    更多信息请参考**在DLI控制台管理数据库资源**。
  – IAM鉴权：
    参考**权限管理概述**章节中的“IAM鉴权使用场景”。
- Lakeformation元数据权限管理

参考**DLI对接LakeFormation**。

3 DLI Hudi 开发规范

3.1 Hudi 开发规范概述

范围

本节内容介绍DLI-Hudi组件进行湖仓一体、流批一体方案的设计与开发方面的规则，适用于Hudi开发场景的表的设计、管理与作业开发。

主要包括以下方面的规范：

- 数据表设计
- 资源配置
- 性能调优
- 常见故障处理
- 常用参数配置

术语约定

本规范采用以下的术语描述：

- **规则**：编程时强制必须遵守的原则。
- **建议**：编程时必须加以考虑的原则。
- **说明**：对此规则或建议进行的解释。
- **示例**：对此规则或建议从正、反两个方面给出。

适用范围

- 基于DLI-Hudi进行数据存储、数据加工作业的设计、开发、测试和维护。
- 该设计开发规范是基于Spark 3.3.1，Hudi 0.11.0版本。

3.2 Hudi 数据表设计规范

3.2.1 Hudi 表模型设计规范

规则

- Hudi表必须设置合理的主键。

  Hudi表提供了数据更新和幂等写入能力，该能力要求Hudi表必须设置主键，主键设置不合理会导致数据重复。主键可以为单一主键也可以为复合主键，两种主键类型均要求主键不能有null值和空值，可以参考以下示例设置主键：

  SparkSQL：

```
// 通过primaryKey指定主键，如果是复合主键需要用逗号分隔
create table hudi_table (
id1 int,
id2 int,
name string,
price double
) using hudi
options (
primaryKey = 'id1,id2',
preCombineField = 'price'
);
```

  SparkDatasource：

```
// 通过hoodie.datasource.write.recordkey.field指定主键
df.write.format("hudi").
option("hoodie.datasource.write.table.type", COPY_ON_WRITE).
option("hoodie.datasource.write.precombine.field", "price").
option("hoodie.datasource.write.recordkey.field", "id1,id2").
```

  FlinkSQL：

```
// 通过hoodie.datasource.write.recordkey.field指定主键
create table hudi_table(
id1 int,
id2 int,
name string,
price double
) partitioned by (name) with (
'connector' = 'hudi',
'hoodie.datasource.write.recordkey.field' = 'id1,id2',
'write.precombine.field' = 'price')
```

- Hudi表必须配置precombine字段。

  在数据同步过程中不可避免会出现数据重复写入、数据乱序问题，例如：异常数据恢复、写入程序异常重启等场景。通过设置合理precombine字段值可以保证数据的准确性，老数据不会覆盖新数据，也就是幂等写入能力。该字段可用选择的类型包括：业务表中更新时间戳、数据库的提交时间戳等。precombine字段不能有null值和空值，可以参考以下示例设置precombine字段：

  SparkSQL:

```
//通过preCombineField指定precombine字段
create table hudi_table (
id1 int,
id2 int,
name string,
price double
) using hudi
options (
primaryKey = 'id1,id2',
preCombineField = 'price'
);
```

  SparkDatasource:

```
//通过hoodie.datasource.write.precombine.field指定precombine字段
df.write.format("hudi").
```

```
option("hoodie.datasource.write.table.type", COPY_ON_WRITE).
option("hoodie.datasource.write.precombine.field", "price").
option("hoodie.datasource.write.recordkey.field", "id1,id2").
```

Flink:

```
//通过write.precombine.field指定precombine字段
create table hudi_table(
id1 int,
id2 int,
name string,
price double
) partitioned by (name) with (
'connector' = 'hudi',
'hoodie.datasource.write.recordkey.field' = 'id1,id2',
'write.precombine.field' = 'price')
```

- 流式计算采用MOR表。

  流式计算为低时延的实时计算，需要高性能的流式读写能力，在Hudi表中存在的MOR和COW两种模型中，MOR表的流式读写性能相对较好，因此在流式计算场景下采用MOR表模型。关于MOR表在读写性能的对比关系如下：

对比维度	MOR表	COW表
流式写	高	低
流式读	高	低
批量写	高	低
批量读	低	高

- 实时入湖，表模型采用MOR表。

  实时入湖一般的性能要求都在分钟内或者分钟级，结合Hudi两种表模型的对比，因此在实时入湖场景中需要选择MOR表模型。

- Hudi表名以及列名采用小写字母。

  多引擎读写同一张Hudi表时，为了规避引擎之间大小写的支持不同，统一采用小写字母。

建议

- Spark批处理场景，对写入时延要求不高的场景，采用COW表。

  COW表模型中，写入数据存在写放大问题，因此写入速度较慢；但COW具有非常好的读取性能力。而且批量计算对写入时延不是很敏感，因此可以采用COW表。

- Hudi表的写任务要开启Hive元数据同步功能。

  SparkSQL天然与Hive集成，无需考虑元数据问题。该条建议针对的是通过Spark Datasource API或者Flin写Hudi表的场景，通过这两种方式写Hudi时需要增加向Hive同步元数据的配置项；该配置的目的是将Hudi表的元数据统一托管到Hive元数据服务中，为后续的跨引擎操作数据以及数据管理提供便利。

3.2.2 Hudi 表索引设计规范

规则

- 禁止修改表索引类型。

  Hudi表的索引会决定数据存储方式，随意修改索引类型会导致表中已有的存量数据与新增数据之间出现数据重复和数据准确性问题。常见的索引类型如下：

  - 布隆索引：Spark引擎独有索引，采用bloomfiter机制，将布隆索引内容写入到Parquet文件的footer中。
  - Bucket索引：在写入数据过程中，通过主键进行Hash计算，将数据进行分桶写入；该索引写入速度最快，但是需要合理配置分桶数目；Flink、Spark均支持该索引写入。
  - 状态索引：Flink引擎独有索引，是将行记录的存储位置记录到状态后端的一种索引形式，在作业冷启动过程中会遍历所有数据存储文件生成索引信息。
- 用Flink状态索引，Flink写入后，不支持Spark继续写入。

  Flink在写Hudi的MOR表只会生成log文件，后续通过compaction操作，将log文件转为parquet文件。Spark在更新Hudi表时严重依赖parquet文件是否存在，如果当前Hudi表写的是log文件，采用Spark写入就会导致重复数据的产生。在批量初始化阶段 ，先采用Spark批量写入Hudi表，再用Flink基于Flink状态索引写入不会有问题，原因是Flink冷启动的时候会遍历所有的数据文件生成状态索引。
- 实时入湖场景中，Spark引擎采用Bucket索引，Flink引擎可以用Bucket索引或者状态索引。

  实时入湖都是需要分钟内或者分钟级的高性能入湖，索引的选择会影响到写Hudi表的性能。在性能方面各个索引的区别如下：
  - Bucket索引

    优点：写入过程中对主键进行hash分桶写入，性能比较高，不受表的数据量限制。Flink和Spark引擎都支持，Flink和Spark引擎可以实现交叉混写同一张表。

    缺点：Bucket个数不能动态调整，数据量波动和整表数据量持续上涨会导致单个Bucket数据量过大出现大数据文件。需要结合分区表来进行平衡改善。
  - Flink状态索引

    优点：主键的索引信息存在状态后端，数据更新只需要点查状态后端即可，速度较快；同时生成的数据文件大小稳定，不会产生小文件、超大文件问题。

    缺点：该索引为Flink特有索引。在表的总数据行数达到数亿级别，需要优化状态后端参数来保持写入的性能。使用该索引无法支持Flink和Spark交叉混写。
- 对于数据总量持续上涨的表，采用Bucket索引时，须使用时间分区，分区键采用数据创建时间。

  参照Flink状态索引的特点，Hudi表超过一定数据量后，Flink作业状态后端压力很大，需要优化状态后端参数才能维持性能；同时由于Flink冷启动的时候需要遍历全表数据，大数据量也会导致Flink作业启动缓慢。因此基于简化使用的角度，针对大数据量的表，可以通过采用Bucket索引来避免状态后端的复杂调优。

  如果Bucket索引+分区表的模式无法平衡Bueckt桶过大的问题，还是可以继续采用Flink状态索引，按照规范去优化对应的配置参数即可。

建议

- 基于Flink的流式写入的表，在数据量超过2亿条记录，采用Bucket索引，2亿以内可以采用Flink状态索引。

  参照Flink状态索引的特点，Hudi表超过一定数据量后，Flink作业状态后端压力很大，需要优化状态后端参数才能维持性能；同时由于Flink冷启动的时候需要遍历全表数据，大数据量也会导致Flink作业启动缓慢。因此基于简化使用的角度，针对大数据量的表，可以通过采用Bucket索引来避免状态后端的复杂调优。

如果Bucket索引+分区表的模式无法平衡Bueckt桶过大的问题，还是可以继续采用Flink状态索引，按照规范去优化对应的配置参数即可。

- 基于Bucket索引的表，按照单个Bucket 2GB数据量进行设计。

  为了规避单个Bucket过大，建议单个Bucket的数据量不要超过2GB（该2GB是指数据内容大小，不是指数据行数也不是parquet的数据文件大小），目的是将对应的桶的Parquet文件大小控制在256MB范围内（平衡读写内存消耗和HDFS存储有效利用），因此可以看出2GB的这个限制只是一个经验值，因为不同的业务数据经过列存压缩后大小是不一样的。

  为什么建议是2GB?

  - 2GB的数据存储成列存Parquet文件后，大概的数据文件大小是150MB ~ 256MB左右。不同业务数据会有出入。而HDFS单个数据块一般会是128MB，这样可以有效地利用存储空间。
  - 数据读写占用的内存空间都是原始数据大小（包括空值也是会占用内存的），2GB在大数据计算过程中，处于单task读写可接受范围之内。

  如果是单个Bucket的数据量超过了该值范围，可能会有什么影响?

  - 读写任务可能会出现OOM的问题，解决方法就是提升单个task的内存占比。
  - 读写性能下降，因为单个task的处理的数据量变大，导致处理耗时变大。

3.2.3 Hudi 表分区设计规范

规则

分区键不可以被更新：

Hudi具有主键唯一性机制，但在分区表的场景下通常只能保证分区内主键唯一，因此如果分区键的值发生变更后，会导致相同主键的行记录出现多条的情况。在以日期分区的场景，可采用数据的创建时间为分区字段，切记不要采用数据更新时间做分区。

说明

当指定Hudi的索引类型为Global索引类型时，Hudi支持跨分区进行数据更新，但Global索引性能较差一般不建议使用。

建议

- 事实表采用日期分区表，维度表采用非分区或者大颗粒度的日期分区

  是否采用分区表要根据表的总数据量、增量和使用方式来决定。从表的使用属性看事实表和维度表具有的特点：

  - 事实表：数据总量大，增量大，数据读取多以日期做切分，读取一定时间段的数据。
  - 维度表：总量相对小，增量小，多以更新操作为主，数据读取会是全表读取，或者按照对应业务ID过滤。

  基于以上考虑，维度表采用天分区会导致文件数过多，而且是全表读取，会导致所需要的文件读取Task过多，采用大颗粒度的日期分区，例如年分区，可以有效降低分区个数和文件数量；对于增量不是很大的维度表，也可以采用非分区表。如果维度表的总数据量很大或者增量也很大，可以考虑采用某个业务ID进行分区，在大部分数据处理逻辑中针对大维度表，会有一定的业务条件进行过滤来提升处理性能，这类表要结合一定的业务场景来进行优化，无法从单纯的日期分区进行优化。事实表读取方式都会按照时间段切分，近一年、近一个月或者近一天，读取的文件数相对稳定可控，所以事实表优先考虑日期分区表。

- 分区采用日期字段，分区表粒度，要基于数据更新范围确定，不要过大也不要过小。

  分区粒度可以采用年、月、日，分区粒度的目标是减少同时写入的文件桶数，尤其是在有数据量更新，且更新数据有一定时间范围规律的，比如：近一个月的数据更新占比最大，可以按照月份创建分区；近一天内的数据更新占比大，可以按照天进行分区。

  采用Bucket索引，写入是通过主键Hash打散的，数据会均匀的写入到分区下每个桶。因为各个分区的数据量是会有波动的，分区下桶的个数设计一般会按照最大分区数据量计算，这样会出现越细粒度的分区，桶的个数会冗余越多。例如：

  采用天级分区，平均的日增数据量是3GB，最多一天的日志是8GB，这个会采用Bucket桶数= 8GB/2GB = 4 来创建表；每天的更新数据占比较高，且主要分散到近一个月。这样会导致结果是，每天的数据会写入到全月的Bucket桶中，那就是4*30 = 120个桶。如果采用月分区，分区桶的个数= 3GB * 30 /2GB = 45个桶 ，这样写入的数据桶数减少到了45个桶。在有限的计算资源下，写入的桶数越少，性能越高。

3.3 Hudi 数据表管理操作规范

3.3.1 Hudi 数据表 Compaction 规范

mor表更新数据以行存log的形式写入，log读取时需要按主键合并，并且是行存的，导致log读取效率比parquet低很多。为了解决log读取的性能问题，Hudi通过compaction将log压缩成parquet文件，大幅提升读取性能。

规则

- 有数据持续写入的表，24小时内至少执行一次compaction。

  对于MOR表，不管是流式写入还是批量写入，需要保证每天至少完成1次Compaction操作。如果长时间不做compaction，Hudi表的log将会越来越大，这必将会出现以下问题：

  - Hudi表读取很慢，且需要很大的资源。 这是由于读MOR表涉及到log合并，大log合并需要消耗大量的资源并且速度很慢。
  - 长时间进行一次Compaction需要耗费很多资源才能完成，且容易出现OOM。
  - 阻塞Clean，如果没有Compaction操作来产生新版本的Parquet文件，那旧版本的文件就不能被Clean清理，增加存储压力。

- 提交Spark jar作业时，CPU与内存比例为1:4~1:8。

  Compaction作业是将存量的parquet文件内的数据与新增的log中的数据进行合并，需要消耗较高的内存资源，按照之前的表设计规范以及实际流量的波动结合考虑，建议Compaction作业CPU与内存的比例按照1:4~1:8配置，保证Compaction作业稳定运行。当Compaction出现OOM问题，可以通过调大内存占比解决。

建议

- 通过增加并发数提升Compaction性能。

  CPU和内存比例配置合理会保证Compaction作业是稳定的，实现单个Compaction task的稳定运行。但是Compaction整体的运行时长取决于本次Compaction处理文件数以及分配的cpu核数（并发能力），因此可以通过增加

Compaciton作业的CPU核的个数来提升Compaction性能（注意增加cpu也要保证CPU与内存的比例）。

- Hudi表采用异步Compaction。

  为了保证流式入库作业的稳定运行，就需要保证流式作业不在实时入库的过程中做其它任务，比如Flink写Hudi的同时会做Compaction。这看似是一个不错的方案，即完成了入库又完成Compaction。但是Compaction操作是非常消耗内存和IO的，它会给流式入库作业带来以下影响：

  - 增加端到端时延：Compaction会放大写入时延，因为Compaction比入库更耗时。
  - 作业不稳定：Compaction会给入库作业带来更多的不稳定性，Compaction OOM将会导致整个作业直接失败。

- 建议2~4小时进行一次compaction。

  Compaction是MOR表非常重要且必须执行的维护手段，对于实时任务来说，要求Compaction执行合并的过程必须和实时任务解耦，通过周期调度Spark任务来完成异步Compaction，这个方案的关键之处在于如何合理的设置这个周期，周期如果太短意味着Spark任务可能会空跑，周期如果太长可能会积压太多的Compaction Plan没有去执行而导致Spark任务耗时长并且也会导致下游的读作业时延高。对此场景，在这里给出以下建议：按照集群资源使用情况，可以每2小时或每4个小时去调度执行一次异步Compaction作业，这是一个基本的维护MOR表的方案。

- 采用Spark异步执行Compaction，不采用Flink进行Compaction。

  Flink写hudi建议的方案是Flink只负责写数据和生成Compaction计划。由单独的队列提交Spark SQL或Spark jar作业异步执行compaction、clean和archive。Compaction计划的生成是轻量级的对Flink写入作业影响可以忽略。

  上述方案落地的具体步骤参考如下：

  - **Flink只负责写数据和生成Compaction计划**

    Flink流任务建表语句/SQL hints中添加如下参数，控制Flink任务写Hudi时只会生成Compaction plan。

    ```
    'compaction.async.enabled' = 'false'      // 关闭Flink 执行Compaction任务
    'compaction.schedule.enabled' = 'true'    // 开启Compaction计划生成
    'compaction.delta_commits' = '5'          // MOR表默认5次checkpoint尝试生成compaction plan，该参数需要根据具体业务调整
    'clean.async.enabled' = 'false'           // 关闭Clean操作
    'hoodie.archive.automatic' = 'false'      // 关闭Archive操作
    ```

  - **Spark离线完成Compaction计划的执行，以及Clean和Archive操作**

    在调度平台（可以使用华为的DataArts）运行一个定时调度的离线任务来让Spark完成Hudi表的Compaction计划执行以及Clean和Archive操作。

    以SQL作业为例，在配置中添加：

    ```
    hoodie.archive.automatic = false;
    hoodie.clean.automatic = false;
    hoodie.compact.inline = true;
    hoodie.run.compact.only.inline=true;
    hoodie.cleaner.commits.retained = 500;  // clean保留timeline上最新的500个deltacommit对应的数据文件，之前的deltacommit所对应的旧版本文件会被清理。该值需要大于compaction.delta_commits设置的值，需要根据具体业务调整。
    hoodie.keep.max.commits = 700;  // timeline最多保留700个deltacommit
    hoodie.keep.min.commits = 501;  // timeline最少保留500个deltacommit。该值需要大于hoodie.cleaner.commits.retained设置的值，需要根据具体业务调整。
    ```

    随后保持上述配置**按顺序调度执行**以下SQL：

    ```
    run compaction on <database name>. <table name>;   // 执行Compaction计划
    run clean on <database name>. <table name>;        // 执行Clean操作
    run archivelog on <database name>.<table name>;    // 执行Archive操作
    ```

- 异步Compaction可以将多个表串行到一个作业，资源配置相近的表放到一组，该组作业的资源需求为最大消耗资源的表所需的资源

  对于在•Hudi表采用异步Compaction和•采用Spark异步执行Compaction，不...中提到的异步Compaction任务，这里给出以下开发建议：

  - 不需要对每张Hudi表都开发异步Compaction任务，这样会导致作业开发成本上升。
  - 异步Compaction任务可以通过提交Spark SQL作业来完成，也可以在Spark jar任务中处理多张表的compaction，clean，archive：

```
hoodie.clean.async = true;
hoodie.clean.automatic = false;
hoodie.compact.inline = true;
hoodie.run.compact.only.inline=true;
hoodie.cleaner.commits.retained = 500;
hoodie.keep.min.commits = 501;
hoodie.keep.max.commits = 700;
```

  需要**按顺序调度执行**以下SQL：

```
run compaction on <database name>. <table1>;
run clean on <database name>. <table1>;
run archivelog on <database name>.<table1>;
run compaction on <database name>.<table2>;
run clean on <database name>.<table2>;
run archivelog on <database name>.<table2>;
```

3.3.2 Hudi 数据表 Clean 规范

Clean也是Hudi表的维护操作之一，该操作对于MOR表和COW表都需要执行。Clean操作的目的是为了清理旧版本文件（Hudi不再使用的数据文件），这不但可以节省Hudi表List过程的时间，也可以缓解存储压力。

规则

Hudi表必须执行Clean。

对于Hudi的MOR、COW表，都需要开启Clean。

- Hudi表在写入数据时会自动判断是否需要执行Clean，因为Clean的开关默认打开(hoodie.clean.automatic默认为true)。
- Clean操作并不是每次写数据时都会触发，至少需要满足两个条件：
  a. Hudi表中需要有旧版本的文件。对于COW表来说，只要保证数据被更新过就一定存在旧版本的文件。对于MOR表来说，要保证数据被更新过并且做过Compaction才能有旧版本的文件。
  b. Hudi表满足hoodie.cleaner.commits.retained设置的阈值。如果是Flink写hudi，则至少提交的checkpoint要超过这个阈值；如果是批写Hudi，则批写次数要超过这个阈值。

建议

- MOR表下游采用批量读模式，采用clean的版本数为compaction版本数+1。

  MOR表一定要保证Compaction Plan能够被成功执行，Compaction Plan只是记录了Hudi表中哪些Log文件要和哪些Parquet文件合并，所以最重要的地方在于保证Compaction Plan在被执行的时候它需要合并的文件都存在。而Hudi表中只有Clean操作可以清理文件，所以建议Clean的触发阈值（hoodie.cleaner.commits.retained的值）至少要大于Compaction的触发阈值（对于Flink任务来说是compaction.delta_commits的值）。

- MOR表下游采用流式计算，历史版本保留小时级。

  如果MOR表的下游是流式计算，例如Flink流读，可以按照业务需要保留小时级的历史版本，这样的话近几个小时之内的增量数据可以通过log文件读出，如果保留时长过短，下游flink作业在重启或者异常中断阻塞的情况下，上游增量数据已经Clean掉了，flink需要从parquet文件读增量数据，性能会有下降；如果保留时间过长，会导致log里面的历史数据冗余存储。

  具体可以按照下面的计算公式来保留2个小时的历史版本数据：

  版本数设置为3600*2/版本interval时间，版本interval时间来自于flink作业的checkpoint周期，或者上游批量写入的周期。

- COW表如果业务没有历史版本数据保留的特殊要求，保留版本数设置为1。

  COW表的每个版本都是表的全量数据，保留几个版本就会冗余多少个版本。因此如果业务无历史数据回溯的需求，保留版本数设置为1，也就是保留当前最新版本

- clean作业每天至少执行一次，可以2~4小时执行一次。

  Hudi的MOR表和COW表都需要保证每天至少1次Clean，MOR表的Clean可以参考2.2.1.6小节和Compaction放在一起异步去执行。COW的Clean可以在写数据时自动判断是否执行。

3.3.3 Hudi 数据表 Archive 规范

Archive（归档）是为了减轻Hudi读写元数据的压力，所有的元数据都存放在这个路径：Hudi表根目录/.hoodie目录，如果.hoodie目录下的文件数量超过10000就会发现Hudi表有非常明显的读写时延。

规则

Hudi表必须执行Archive。

对于Hudi的MOR类型和COW类型的表，都需要开启Archive。

- Hudi表在写入数据时会自动判断是否需要执行Archive，因为Archive的开关默认打开(hoodie.archive.automatic默认为true)。
- Archive操作并不是每次写数据时都会触发，至少需要满足以下两个条件：
  - Hudi表满足hoodie.keep.max.commits设置的阈值。如果是Flink写hudi至少提交的checkpoint要超过这个阈值；如果是Spark写hudi，写Hudi的次数要超过这个阈值。
  - Hudi表做过Clean，如果没有做过Clean就不会执行Archive。

建议

Archive作业每天至少执行一次，可以2~4小时执行一次。

Hudi的MOR表和COW表都需要保证每天至少1次Archive，MOR表的Archive可以参考2.2.1.6小节和Compaction放在一起异步去执行。COW的Archive可以在写数据时自动判断是否执行。

3.4 Spark on Hudi 开发规范

3.4.1 SparkSQL 建表参数规范

规则

- 建表必须指定primaryKey和preCombineField。

  Hudi表提供了数据更新的能力和幂等写入的能力，该能力要求数据记录必须设置主键用来识别重复数据和更新操作。不指定主键会导致表丢失数据更新能力，不指定preCombineField会导致主键重复。

参数名称	参数描述	输入值	说明
primaryKey	hudi主键	按需	必须指定，可以是复合主键但是必须全局唯一。
preCombineField	预合并键，相同主键的多条数据按该字段进行合并	按需	必须指定，相同主键的数据会按该字段合并，不能指定多个字段。

- 禁止建表时将hoodie.datasource.hive_sync.enable指定为false。

  指定为false将导致新写入的分区无法同步到Hive Metastore中。由于缺失新写入的分区信息，查询引擎读取该时会丢数。

- 禁止指定Hudi的索引类型为INMEMORY类型。

  该索引仅是为了测试使用。生产环境上使用该索引将导致数据重复。

建表示例

```
create table data_partition(id int, comb int, col0 int,yy int, mm int, dd int)
using hudi                              --指定hudi 数据源
partitioned by(yy,mm,dd)                --指定分区， 支持多级分区
location 'obs://bucket/path/data_partition'       --指定路径，使用DLI提供的元数据服务时只支持创建OBS表
options(
type='mor',                             --表类型 mor 或者 cow
primaryKey='id',                        --主键，可以是复合主键但是必须全局唯一
preCombineField='comb'                  --预合并字段，相同主键的数据会按该字段合并，当前不能指定多个字段
)
```

3.4.2 Spark 增量读取 Hudi 参数规范

规则

增量查询之前必须指定当前表的查询为增量查询模式，并且查询后重写设置表的查询模式

如果增量查询完，不重新将表查询模式设置回去，将影响后续的实时查询

示例

以SQL作业为例：

配置参数

```
hoodie.tableName.consume.mode=INCREMENTAL // 必须设置当前表读取为增量读取模式
hoodie.tableName.consume.start.timestamp=20201227153030 // 指定初始增量拉取commit
```

```
hoodie.tableName.consume.end.timestamp=20210308212318 // 指定增量拉取结束commit，如果不指定的话采用最新的commit
```

随后执行SQL

```
select * from tableName where `_hoodie_commit_time`>'20201227153030' and `_hoodie_commit_time`<='20210308212318'; // 结果必须根据start.timestamp和end.timestamp进行过滤，如果没有指定end.timestamp，则只需要根据start.timestamp进行过滤。
```

提交其他SQL时，需要清除上述配置参数，避免影响其他任务执行结果。

3.4.3 Spark 异步任务执行表 compaction 参数设置规范

- 写作业未停止情况下，禁止手动执行run schedule命令生成compaction计划。

  错误示例：

  ```
  run schedule on dsrTable
  ```

  如果还有别的任务在写这张表，执行该操作会导致数据丢失。

- 执行run compaction命令时，禁止将hoodie.run.compact.only.inline设置成false，该值需要设置成true。

  错误示例：

  配置参数

  ```
  hoodie.run.compact.only.inline=false
  ```

  随后执行SQL

  ```
  run compaction on dsrTable;
  ```

  如果还有别的任务在写这张表，执行上述操作会导致数据丢失。

  正确示例：异步Compaction

  ```
  hoodie.compact.inline = true
  hoodie.run.compact.only.inline=true
  ```

  执行SQL

  ```
  run compaction on dsrTable;
  ```

3.4.4 Spark 表数据维护规范

禁止通过Alter命令修改表关键属性信息：type/primaryKey/preCombineField/hoodie.index.type

错误示例，执行如下语句修改表关键属性：

```
alter table dsrTable set tblproperties('type'='xx');
alter table dsrTable set tblproperties('primaryKey'='xx');
alter table dsrTable set tblproperties('preCombineField'='xx');
alter table dsrTable set tblproperties('hoodie.index.type'='xx');
```

除Spark以外，其他引擎也可以修改Hudi表元数据，但是这种修改会导致整个Hudi表出现数据重复，甚至数据损坏；因此禁止修改上述属性。

3.5 Bucket 调优示例

3.5.1 创建 Bucket 索引表调优

创建 Bucket 索引表调优

Bucket索引常用设置参数：

- Spark：
```
hoodie.index.type=BUCKET
hoodie.bucket.index.num.buckets=5
```
- Flink
```
index.type=BUCKET
hoodie.bucket.index.num.buckets=5
```

判断使用分区表还是非分区表

根据表的使用场景一般将表分为事实表和维度表：

- 事实表通常整表数据规模较大，以新增数据为主，更新数据占比小，且更新数据大多落在近一段时间范围内（年或月或天），下游读取该表进行ETL计算时通常会使用时间范围进行裁剪（例如最近一天、一月、一年），这种表通常可以通过数据的创建时间来做分区以保证最佳读写性能。
- 维度表数据量一般整表数据规模较小，以更新数据为主，新增较少，表数据量比较稳定，且读取时通常需要全量读取做join之类的ETL计算，因此通常使用非分区表性能更好。
- **分区表的分区键不允许更新，否则会产生重复数据。**

例外场景：超大维度表和超小事实表

特殊情况如存在**持续大量新增数据的维度表**（表数据量在200G以上或日增长量超过60M）或**数据量非常小的事实表**（表数据量小于10G且未来三至五年增长后也不会超过10G）需要针对具体场景来进行例外处理：

- 持续大量新增数据的维度表
  - 方法一：预留桶数，如使用非分区表则需通过预估较长一段时间内的数据增量来预先增加桶数，**缺点是随着数据的增长，文件依然会持续膨胀；**
  - 方法二：大粒度分区（推荐），如果使用分区表则需要根据数据增长情况来计算，例如使用年分区，**这种方式相对麻烦些但是多年后表无需重新导入。**
  - 方法三：数据老化，按照业务逻辑分析大的维度表是否可以通过数据老化清理无效的维度数据从而降低数据规模。
- 数据量非常小的事实表

  这种可以在预估很长一段时间的数据增长量的前提下使用非分区表预留稍宽裕一些的桶数来提升读写性能。

确认表内桶数

Hudi表的桶数设置，关系到表的性能，需要格外引起注意。

以下几点，是设置桶数的关键信息，需要建表前确认。

- 非分区表
  - 单表数据总条数 = select count(1) from tablename（入湖时需提供）；
  - 单条数据大小 = 平均 1KB（华为建议通过select * from tablename limit 100，得出100条数据的大小，再除以100得到单条平均大小）

  - 单表数据量大小(G) = 单表数据总条数*单条数据大小/1024/1024
  - 非分区表桶数 = 单表数据量大小(G)/2G*2，再向上取整，如果小于4就设置桶数为4
- 分区表
  - 最近一个月最大数据量分区数据**总条数** = 入湖前咨询产品线
  - 单条数据大小 = 平均 1KB（华为建议通过select * from tablename limit 100，得出100条数据的大小，再除以100得到单条平均大小）
  - 单分区数据量大小(G) = 最近一个月最大数据量分区数据总条数*单条数据大小/1024/1024
  - 分区表桶数 = 单分区数据量大小(G)/2G，再后向上取整，最小设置1个桶

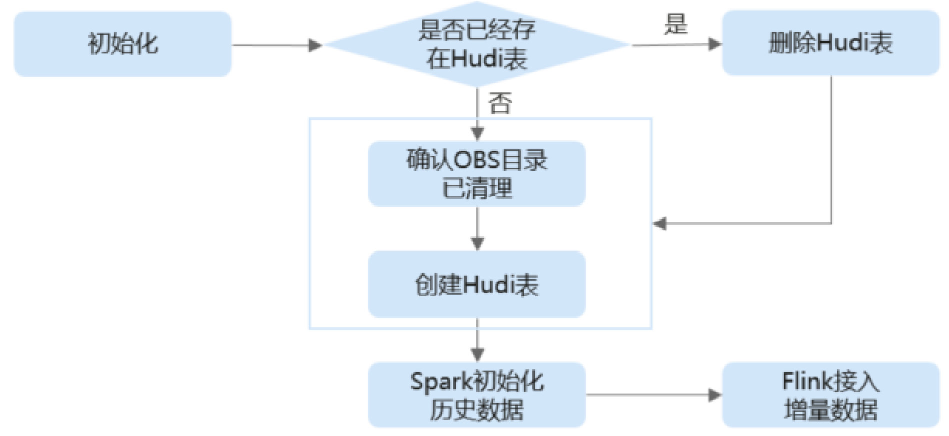
注意

- 需要使用的是表的总数据大小，而不是压缩以后的文件大小
- 桶的设置以偶数最佳，非分区表最小桶数请设置4个，分区表最小桶数请设置1个。

3.5.2 Hudi 表初始化

- 初始化导入存量数据通常由Spark作业来完成，由于初始化数据量通常较大，因此推荐使用API方式给充足资源来完成。
- 对于批量初始化后需要接Flink或Spark流作业实时写入的场景，一般建议通过对上有消息进行过滤，从一个指定的时间范围开始消费来控制数据的重复接入量（例如Spark初始化完成后，Flink消费Kafka时过滤掉2小时之前的数据），如果无法对kafka消息进行过滤，则可以考虑先实时接入生成offset，再truncate table，再历史导入，再开启实时。

图 3-1 初始化操作流程



版权所有 © 华为技术有限公司

说明

- 如果批量初始化前表里已经存在数据且没有truncate table，则会导致批量数据写成非常大的log文件，对后续compaction形成很大压力需要更多资源才能完成。
- Hudi表在Hive元数据中，应该会存在1张内部表（手动创建），2张外部表（写入数据后自动创建）。
- 2张外部表，表名_ro（用户只读合并后的parquet文件，即读优化视图表），_rt（读实时写入的最新版本数据，即实时视图表）。

3.5.3 实时任务接入

实时作业一般由Flink Sql或Sparkstreaming来完成，流式实时任务通常配置同步生成compaction计划，异步执行计划。

- Flink SQL作业中sink端Hudi表相关配置如下：

```
create table hudi_sink_table (
  // table columns...
) PARTITIONED BY (
  years,
  months,
  days
) with (
  'connector' = 'hudi',                                               //指定写入的是Hudi表
  'path' = 'obs://bucket/path/hudi_sink_table',                      //指定Hudi表的存储路径
  'table.type' = 'MERGE_ON_READ',                                    //Hudi表类型
  'hoodie.datasource.write.recordkey.field' = 'id',                  //主键
  'write.precombine.field' = 'vin',                                  //合并字段
  'write.tasks' = '10',                                              //flink写入并行度
  'hoodie.datasource.write.keygenerator.type' = 'COMPLEX',           //指定KeyGenerator，与Spark
创建的Hudi表类型一致
  'hoodie.datasource.write.hive_style_partitioning' = 'true',        //使用hive支持的分区格式
  'read.streaming.enabled' = 'true',                                 //开启流读
  'read.streaming.check-interval' = '60',                            //checkpoint间隔，单位为秒
  'index.type'='BUCKET',                                             //指定Hudi表索引类型为BUCKET
  'hoodie.bucket.index.num.buckets'='10',                            //指定bucket桶数
  'compaction.delta_commits' = '3',                                  //compaction生成的commit间隔
  'compaction.async.enabled' = 'false',                              //compaction异步执行关闭
  'compaction.schedule.enabled' = 'true',                            //compaction同步生成计划
  'clean.async.enabled' = 'false',                                   //异步clean关闭
  'hoodie.archive.automatic' = 'false',                              //自动archive关闭
  'hoodie.clean.automatic' = 'false',                                //自动clean关闭
  'hive_sync.enable' = 'true',                                       //自动同步元数据
  'hive_sync.mode' = 'jdbc',                                         //同步元数据方式为jdbc
  'hive_sync.jdbc_url' = '',                                         //同步元数据的jdbc url
  'hive_sync.db' = 'default',                                        //同步元数据的database
  'hive_sync.table' = 'hudi_sink_table',                             //同步元数据的tablename
  'hive_sync.support_timestamp' = 'true',                            //同步hive表支持timestamp格式
  'hive_sync.partition_extractor_class' = 'org.apache.hudi.hive.MultiPartKeysValueExtractor'            //同步
hive表的extractor类
);
```

- Spark streaming写入Hudi表常用的参数如下（参数意义与上面flink类似，不再做注释）：

```
hoodie.table.name=
hoodie.index.type=BUCKET
hoodie.bucket.index.num.buckets=3
hoodie.datasource.write.precombine.field=
hoodie.datasource.write.recordkey.field=
hoodie.datasource.write.partitionpath.field=
hoodie.datasource.write.table.type= MERGE_ON_READ
hoodie.datasource.write.hive_style_partitioning=true
hoodie.compact.inline=true
hoodie.schedule.compact.only.inline=true
hoodie.run.compact.only.inline=false
hoodie.clean.automatic=false
hoodie.clean.async=false
```

```
hoodie.archive.async=false
hoodie.archive.automatic=false
hoodie.compact.inline.max.delta.commits=50
hoodie.datasource.hive_sync.enable=true
hoodie.datasource.hive_sync.partition_fields=
hoodie.datasource.hive_sync.database=
hoodie.datasource.hive_sync.table=
hoodie.datasource.hive_sync.partition_extractor_class=org.apache.hudi.hive.MultiPartKeysValueExtracto
r
```

3.5.4 离线 Compaction 配置

对于MOR表的实时业务，通常设置在写入中同步生成compaction计划，因此需要额外通过DataArts或者脚本调度SparkSQL去执行已经产生的compaction计划。

- 执行参数

```
set hoodie.compact.inline = true;              //打开compaction操作
set hoodie.run.compact.only.inline = true;       //compaction只执行已生成的计划，不产生新计划
set hoodie.cleaner.commits.retained = 120;       // 清理保留120个commit
set hoodie.keep.max.commits = 140;              // 归档最大保留140个commit
set hoodie.keep.min.commits = 121;              // 归档最小保留121个commit
set hoodie.clean.async = false;                // 打开异步清理
set hoodie.clean.automatic = false;             // 关闭自动清理，防止compaction操作触发clean

run compaction on $tablename;                  // 执行compaction计划
run clean on $tablename;                     // 执行clean操作清理冗余版本
run archivelog on $tablename;                  // 执行archivelog合并清理元数据文件
```

> **注意**
>
> - 关于清理、归档参数的值不宜设置过大，会影响Hudi表的性能，通常建议：
>   - hoodie.cleaner.commits.retained = compaction所需要的commit数的2倍
>   - hoodie.keep.min.commits = hoodie.cleaner.commits.retained + 1
>   - hoodie.keep.max.commits = hoodie.keep.min.commits + 20
> - 执行compaction后再执行clean和archive，由于clean和archivelog对资源要求较小，为避免资源浪费，使用DataArts调度的话可以compaction作为一个任务，clean、archive作为一个任务分别配置不同的资源执行来节省资源使用。

- 执行资源
  - Compaction调度的间隔应小于Compaction计划生成的间隔，例如1小时左右生成一个Compaction计划的话，执行Compaction计划的调度任务应该至少半小时调度一次。
  - Compaction作业配置的资源，vcore数至少要大于等于单个分区的桶数，vcore数与内存的比例应为1：4即1个vcore配4G内存。

4 DLI 中使用 Hudi 开发作业

4.1 在 DLI 使用 Hudi 提交 Spark SQL 作业

登录DLI管理控制台，选择“SQL编辑器”首进入提交SQL作业的界面。提交SQL作业时需要选择支持Hudi的Spark SQL队列。

步骤1 创建一张Hudi表：

将如下的建表语句粘贴至DLI SQL编辑器的输入区域，修改 LOCATION，执行引擎选择Spark，配置队列，数据目录，数据库，随后点击右上角的执行按钮，提交作业。

注意：由DLI提供元数据服务时，暂不支持创建Hudi内表，即必须配置 LOCATION 指向 OBS 路径。

```
CREATE TABLE
  hudi_table (id int, comb long, name string, dt date) USING hudi PARTITIONED BY (dt) OPTIONS (
    type = 'cow',
    primaryKey = 'id',
    preCombineField = 'comb'
  ) LOCATION 'obs://bucket/path/hudi_table';
```

等待下方执行历史显示作业执行成功，代表建表成功，此时创建了一张Hudi的COW分区表。

可以执行 SHOW TABLES 检查建表是否成功：

```
SHOW TABLES;
```

步骤2 执行SQL写入刚才创建的Hudi表：

```
INSERT INTO hudi_table VALUES (1, 100, 'aaa', '2021-08-28'), (2, 200, 'bbb', '2021-08-28');
```

随后可以在编辑器下方的”执行历史（最近一天）”窗格中检查执行结果，或者点击左侧”作业管理”-> ”SQL作业”跳转到SQL作业记录中检查。

步骤3 在执行SQL的时候配置Hudi参数：

说明

DLI不支持直接使用SET语句配置参数

点击”设置”，随后在”参数设置”一栏可以配置键和值，Hudi的参数可以通过此处提交。配置在此处的参数会在提交SQL作业时被应用。

随后可以在左侧菜单点击”作业管理”->”SQL作业”，随后在列表中选中执行的作业，并点击下方窗格，唤出作业详情，在”参数设置”一栏中，可以检查参数配置情况。

步骤4 执行SQL查询刚才写入的内容：

```
select id,comb,name,dt from hudi_table where dt='2021-08-28';
```

可以在编辑器下方窗格查看查询结果。

步骤5 删除刚才创建的Hudi表：

如果创建的是外表，执行SQL删除表时仅删除Hudi表的元数据，数据仍然存在OBS桶中，需要手动清理。

```
DROP TABLE IF EXISTS hudi_table;
```

----结束

4.2 在 DLI 使用 Hudi 提交 Spark Jar 作业

提交Spark jar作业的场景需要手动配置由LakeFormation提供元数据服务的Hudi锁实现类，请参照 **Hudi锁配置说明**。

步骤1 登录DLI管理控制台，选择“作业管理 > Spark作业”，进入到Spark作业的界面。

提交Hudi相关的Spark jar作业需要选择Spark版本为3.3.1，且使用的通用队列需要支持Hudi。

步骤2 单击右上角的”创建作业”即可提交Spark jar的作业。

步骤3 编写并打包Spark jar的程序包：（以Maven项目为例）

创建或使用现有的maven java项目，在 pom.xml 中引入scala 2.12，spark 3.3.1 和 hudi 0.11.0 版本的依赖。由于DLI环境已提供所需依赖，因此scope可以配置为provided。

```
<dependencies>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>2.12.15</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>3.3.1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>3.3.1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.hudi</groupId>
    <artifactId>hudi-spark3-bundle_2.12</artifactId>
    <version>0.11.0</version>
    <scope>provided</scope>
  </dependency>
  <!-- ... -->
</dependencies>
```

配置 scala-maven-plugin，用于编译和打包。

```
<build>
  <plugins>
    <plugin>
      <groupId>net.alchim31.maven</groupId>
      <artifactId>scala-maven-plugin</artifactId>
      <version>3.3.1</version>
      <executions>
        <execution>
          <goals>
            <goal>compile</goal>
            <goal>testCompile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <!-- ... -->
  </plugins>
  <!-- ... -->
</build>
```

随后在main目录下创建scala目录，并新建一个包，随后在包目录下新建一个scala文件，在里面写入：

```
import org.apache.spark.sql.catalyst.expressions.GenericRowWithSchema
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.sql.types.{DataTypes, StructField, StructType}

import java.util.{ArrayList, List => JList}

object HudiScalaDemo {
  def main(args: Array[String]): Unit = {
    // 步骤1：获取/创建SparkSession实例
    val spark = SparkSession.builder
      .enableHiveSupport
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .config("spark.sql.extensions", "org.apache.spark.sql.hudi.HoodieSparkSessionExtension")
      .appName("spark_jar_hudi_demo")
      .getOrCreate

    // 步骤2：构造写入用的DataFrame数据
    val schema = StructType(Array(
      StructField("id", DataTypes.IntegerType),
      StructField("name", DataTypes.StringType),
      StructField("update_time", DataTypes.StringType),
      StructField("create_time", DataTypes.StringType)
    ))
    val data: JList[Row] = new ArrayList[Row]()
    data.add(new GenericRowWithSchema(Array(1, "Alice", "2024-08-05 09:00:00", "2024-08-01"), schema))
    data.add(new GenericRowWithSchema(Array(2, "Bob", "2024-08-05 09:00:00", "2024-08-02"), schema))
    data.add(new GenericRowWithSchema(Array(3, "Charlie", "2024-08-05 09:00:00", "2024-08-03"),
schema))
    val df = spark.createDataFrame(data, schema)

    // 步骤3：配置写入的表名和OBS路径
    val databaseName = "default"
    val tableName = "hudi_table"
    val basePath = "obs://bucket/path/hudi_table"

    // 步骤4：执行写入，同时会同步DLI元数据服务建表
    df.write.format("hudi")
      .option("hoodie.table.name", tableName)
      .option("hoodie.datasource.write.table.type", "COPY_ON_WRITE")
      .option("hoodie.datasource.write.recordkey.field", schema.fields(0).name)  // 主键，必须配置
      .option("hoodie.datasource.write.precombine.field", schema.fields(2).name)  // 预聚合键，必须配置，如
果不需要可以配置和主键相同的列
      .option("hoodie.datasource.write.partitionpath.field", schema.fields(3).name)  // 分区列，可以配置多个分
区，使用英语逗号分隔
      .option("hoodie.datasource.write.keygenerator.class", "org.apache.hudi.keygen.ComplexKeyGenerator")
      // 使用DLI元数据服务时，需要同步配置使用对应的Hudi锁
      .option("hoodie.write.lock.provider", "com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider")
```

```
      // 开启同步配置
      .option("hoodie.datasource.hive_sync.enable", "true")
      .option("hoodie.datasource.hive_sync.partition_fields", schema.fields(3).name)
      // 根据实际分区字段情况配置，非分区表请选择 org.apache.hudi.hive.NonPartitionedExtractor
      .option("hoodie.datasource.hive_sync.partition_extractor_class",
"org.apache.hudi.hive.MultiPartKeysValueExtractor")
      .option("hoodie.datasource.hive_sync.use_jdbc", "false")
      .option("hoodie.datasource.hive_sync.table", tableName)
      .option("hoodie.datasource.hive_sync.database", databaseName)
      .mode(SaveMode.Overwrite)
      .save(basePath)

    // 步骤5：使用SQL的方式查询刚才写入的表
    spark.sql(s"select id,name,update_time,create_time from ${databaseName}.${tableName} where
create_time='2024-08-01'")
      .show(100)
  }
}
```

随后执行maven打包命令，从target目录获取打包的jar文件并上传至OBS目录中。

```
mvn clean install
```

步骤4 提交Spark jar作业：

进入DLI界面，在左侧菜单点击”作业管理”->”Spark作业”，随后在右侧界面的右上角点击”创建作业”。

- 首先配置队列，Spark版本选择 3.3.1 及之后的版本。
- 可以选择配置作业名称，便于识别和筛选。
- 配置”应用程序”，路径指向上一步上传至OBS的Spark jar包。
- 配置委托。选择提交DLI作业所需的委托。自定义委托请参考**创建DLI自定义委托**。
- 配置”主类（--class）”，为上一步中所写的，包含需要执行的main函数的类的全名。
- 在”Spark参数（--conf）“处也可以配置Hudi参数，但是需要额外添加前缀“spark.hadoop.“，例如：
  ```
  spark.hadoop.hoodie.write.lock.provider=com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider
  ```
- 配置”访问元数据”为”是”，推荐使用元数据服务管理Hudi表，上一步中写入配置包含了同步元数据的配置项。

最后点击右上角的”执行”按钮即可提交作业。

步骤5 执行作业，检查日志：（注意：日志归档耗时较长，在作业执行完成后，日志可能需要等待1-5分钟才能归档。）

点击执行后会跳转到”Spark作业”界面，此处可以看到作业的执行状态。点击对应作业右侧的更多，可以在下拉菜单中跳转日志选单：

- 归档日志：跳转OBS界面，可以看到该作业的全部日志归档地址，包含提交日志，Driver日志和Executor日志，在此处可以下载日志。
- 提交日志：跳转到提交日志的聚合展示界面，可以查看任务提交中的日志信息。
- Driver日志：跳转到Driver日志的聚合展示界面，从上至下依次展示 spark.log，stderr.log以及stdout.log。

随后进入Driver日志，如果日志还未聚合，请等待几分钟后再次检查。可以在日志底部的stdout.log中查看到示例程序最后打印的select语句的结果。

----结束

4.3 在 DLI 使用 Hudi 提交 Flink SQL 作业

本节操作介绍在DLI使用Hudi提交Flink SQL作业的操作步骤。

具体语法说明请参考**Flink OpenSource SQL1.15语法概览**。

进入DLI控制台，随后单击左侧菜单的“作业管理 > Flink作业”，进入Flink作业的界面。

步骤1 创建Flink作业：点击界面右上角的”创建作业”按钮，在弹出窗口中配置作业名称，类型选择”Flink OpenSource SQL”

步骤2 写入Flink SQL （不使用Catalog的场景）：

这里的sink表通过创建临时表指向Hudi表路径来写入数据，同时在表参数中配置hive_sync相关参数，实时同步元数据至由DLI提供的元数据服务。（具体参数详见Flink参数一节）

请将作业中sink表的path参数修改为希望保存hudi表的obs路径。

```
-- 临时表作为source，通过datagen mock数据
create table
  orderSource (
    order_id STRING,
    order_name STRING,
    order_time TIMESTAMP(3)
  )
with
  ('connector' = 'datagen', 'rows-per-second' = '1');

-- 创建Hudi临时表作为sink，通过配置hms同步，将表同步至DLI元数据服务
CREATE TABLE
  hudi_table (
    order_id STRING PRIMARY KEY NOT ENFORCED,
    order_name STRING,
    order_time TIMESTAMP(3),
    order_date String
  ) PARTITIONED BY (order_date)
WITH
  (
    'connector' = 'hudi',
    'path' = 'obs://bucket/path/hudi_table',
    'table.type' = 'MERGE_ON_READ',
    'hoodie.datasource.write.recordkey.field' = 'order_id',
    'write.precombine.field' = 'order_time',
    'hive_sync.enable' = 'true',
    'hive_sync.mode' = 'hms',
    'hive_sync.table' = 'hudi_table',
    'hive_sync.db' = 'default'
  );

-- 执行insert，读取source表，写入sink
insert into
  hudi_table
select
  order_id,
  order_name,
  order_time,
  DATE_FORMAT (order_time, 'yyyyMMdd')
from
  orderSource;
```

步骤3 配置作业运行参数：

- 选择队列，并配置Flink版本至少为1.15。

- 配置权限足够的委托。
- 配置OBS桶。
- 开启Checkpoint，使用Hudi时必须开启Checkpoint。

步骤4 提交作业并检查Flink UI和日志：

直接点击界面右上角的”提交”，在跳转界面再次确认参数无误后，点击底部”立即启动”。完成提交后自动跳转至Flink作业界面，此处可以筛选刚才提交的Flink作业并检查执行状态。

点击作业的名称，可以跳转至作业界面，此处可以点击”提交日志”或”运行日志”，检查聚合的日志。也可以直接点击日志列表，选择JobManager或者TaskManager，并下载对应日志。

点击作业界面右上角的”更多”->”Flink UI”，即可跳转至该任务的Flink UI界面。

----结束

4.4 使用 HetuEngine on Hudi

HetuEngine是高性能的交互式SQL分析及数据虚拟化引擎，它与大数据生态无缝融合，实现海量数据秒级交互式查询，并支持跨源跨域统一访问，使能数据湖内、湖间、湖仓一站式SQL融合分析。

HetuEngine对Hudi仅支持select操作，即支持SELECT语法来查询Hudi表中的数据。

HetuEngine暂不支持查询Hudi的增量视图。

详细语法说明请参考《HetuEngine SQL语法参考》中“SELECT”语法说明。

5 DLI Hudi SQL 语法参考

5.1 Hudi DDL 语法说明

5.1.1 CREATE TABLE

命令功能

CREATE TABLE命令通过指定带有表属性的字段列表来创建Hudi Table。在使用由DLI提供的元数据服务时仅可创建外表，即需要通过LOCATION指定表路径。

命令格式

CREATE TABLE *[IF NOT EXISTS] [database_name.]table_name*

[(columnTypeList)]

USING hudi

[COMMENT table_comment]

[LOCATION location_path]

[OPTIONS (options_list)]

参数描述

表 5-1 CREATE TABLE 参数描述

参数	描述
database_name	Database名称，由字母、数字和下划线（_）组成。
table_name	Database中的表名，由字母、数字和下划线（_）组成。
columnTypeList	以逗号分隔的带数据类型的列表。列名由字母、数字和下划线（_）组成。

参数	描述
using	参数hudi，定义和创建Hudi table。
table_comment	表的描述信息。
location_path	OBS路径，指定该路径Hudi 表会创建为外表。
options_list	Hudi table属性列表。

表 5-2 CREATE TABLE Options 描述

参数	描述
primaryKey	主键名，多个字段用逗号分隔，该字段为必填字段。
type	表类型。'cow' 表示 COPY-ON-WRITE 表，'mor' 表示 MERGE-ON-READ 表。未指定type的话，默认值为 'cow'。
preCombineField	表的preCombine字段，写入前预聚合数据时，当主键相同，preCombine字段会用于比较，该字段为必填字段。
payloadClass	使用preCombineField字段进行数据过滤的逻辑，默认使用DefaultHoodieRecordPayload，同时也提供了多种预置Payload供用户使用，如OverwriteNonDefaultsWithLatestAvroPayload、OverwriteWithLatestAvroPayload及EmptyHoodieRecordPayload。
useCache	是否在Spark中缓存表的relation，无需用户配置。为支持SparkSQL中对COW表增量视图查询，默认将COW表中该值置为false。

示例

- **创建非分区表**

```
create table if not exists hudi_table0 (
id int,
name string,
price double
) using hudi
options (
type = 'cow',
primaryKey = 'id',
preCombineField = 'price'
);
```

- **创建分区表**

```
create table if not exists hudi_table_p0 (
id bigint,
name string,
ts bigint,
dt string,
hh string
) using hudi
options (
```

```
type = 'cow',
primaryKey = 'id',
preCombineField = 'ts'
)
partitioned by (dt, hh);
```

- **在指定路径下创建表**

```
create table if not exists h3(
id bigint,
name string,
price double
) using hudi

options (
primaryKey = 'id',
preCombineField = 'price'
)
location 'obs://bucket/path/to/hudi/h3';
```

- **创建表指定表属性**（支持该操作，但是不建议，因为属性写在建表语句里后续不方便修改）

```
create table if not exists h3(
id bigint,
name string,
price double
) using hudi
options (
primaryKey = 'id',
type = 'mor',
hoodie.cleaner.fileversions.retained = '20',
hoodie.keep.max.commits = '20'
);
```

注意事项

- Hudi当前不支持使用char、varchar、tinyint、smallint类型，建议使用string或int类型。
- Hudi当前只有int、bigint、float、double、decimal、string、date、timestamp、boolean、binary类型支持设置默认值。
- Hudi表必须指定primaryKey与preCombineField。
- 在指定路径下创建表时，如果路径下已存在Hudi表，则建表时不需要指定列，且不能修改表的原有属性。

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
CREATE_TABLE	无

- 细粒度权限：dli:database:createTable

LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

Table创建成功，创建的Hudi表可以进入DLI控制台，在左侧菜单栏选择”数据管理”->”库表管理”，随后筛选数据库并点击名称，进入表列表查询。

5.1.2 DROP TABLE

命令功能

DROP TABLE的功能是用来删除已存在的Table。

命令格式

DROP TABLE *[IF EXISTS] [db_name.]table_name;*

参数描述

表 **5-3** DROP TABLE 参数描述

参数	描述
db_name	Database名称。如果未指定，将选择当前database。
table_name	需要删除的Table名称。

注意事项

- 在该命令中，IF EXISTS和db_name是可选配置。
- 在使用本语句删除外表时，OBS目录的数据不会自动删除。
- 删除MOR表时，后缀_rt表和后缀_ro表不会自动删除，如需删除需要额外执行DROP语句。

示例

DROP TABLE IF EXISTS hudidb.h1;

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
DROP_TABLE	无

- 细粒度权限：dli:table:dropTable

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

Table将被删除。

5.1.3 SHOW TABLE

命令功能

SHOW TABLES命令用于显示所有在当前database中的table，或所有指定database的table。

命令格式

SHOW TABLES *[IN db_name];*

参数描述

表 **5-4** SHOW TABLES 参数描述

参数	描述
IN db_name	Database名称，仅当需要显示指定Database的所有Table时配置。

注意事项

IN db_Name为可选配置。

示例

```
SHOW TABLES IN hudidb;
```

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
LIST_TABLES，DISPLAY_ALL_TABLES，SELECT（只需其中之一即可）	无

- 细粒度权限：dli:database:displayAllTables

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以直接在提交任务界面查看任务结果，或者在SQL作业界面，点击对应任务右侧的”更多”->”查看结果”检查任务结果。

5.1.4 TRUNCATE TABLE

命令功能

该命令将会把表中的数据清空。

命令语法

TRUNCATE TABLE *tableIdentifier*

参数描述

表 5-5 TRUNCATE TABLE 参数描述

参数	描述
tableIdentifier	表名。

示例

```
truncate table h0_1;
```

系统响应

通过运行QUERY语句查看表中数据已被删除。

5.2 Hudi DML 语法说明

5.2.1 CREATE TABLE AS SELECT

命令功能

CREATE TABLE As SELECT命令通过指定带有表属性的字段列表来创建Hudi Table。在使用由DLI提供的元数据服务时仅可创建外表，即需要通过LOCATION指定表路径。

命令格式

CREATE TABLE *[IF NOT EXISTS] [database_name.]table_name*

USING hudi

[COMMENT table_comment]

[LOCATION location_path]

[OPTIONS (options_list)]

[AS query_statement]

参数描述

表 5-6 CREATE TABLE As SELECT 参数描述

参数	描述
database_name	Database名称，由字母、数字和下划线（_）组成。
table_name	Database中的表名，由字母、数字和下划线（_）组成。
using	参数hudi，定义和创建Hudi table。
table_comment	表的描述信息。
location_path	OBS路径，指定该路径Hudi表会创建为外表。
options_list	Hudi table属性列表。
query_statement	select查询表达式

示例

- 创建分区表
```
create table h2 using hudi
options (type = 'cow', primaryKey = 'id', preCombineField = 'dt')
partitioned by (dt)
as
select 1 as id, 'a1' as name, 10 as price, 1000 as dt;
```
- 创建非分区表
```
create table h3 using hudi
options (type = 'cow', primaryKey = 'id', preCombineField = 'dt')
as
select 1 as id, 'a1' as name, 10 as price, 1000 as dt;

从parquet表加载数据到hudi表
# 创建parquet表
create table parquet_mngd using parquet options(path=’obs://bucket/path/parquet_dataset/
*.parquet’);

# CTAS创建hudi表
create table hudi_tbl using hudi location 'obs://bucket/path/hudi_tbl/' options (
type = 'cow',
primaryKey = 'id',
preCombineField = 'ts'
)
partitioned by (datestr) as select * from parquet_mngd;
```

注意事项

为了更好的加载数据性能，CTAS使用bulk insert作为写入方式。

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
CREATE_TABLE	来源表：SELECT

- 细粒度权限：dli:table:createTable, dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

Table创建成功，创建的Hudi表可以进入DLI控制台，在左侧菜单栏选择”数据管理”->”库表管理”，随后筛选数据库并点击名称，进入表列表查询。

5.2.2 INSERT INTO

命令功能

INSERT命令用于将SELECT查询结果加载到Hudi表中。

命令格式

INSERT INTO *tableIndentifier select query;*

参数描述

表 **5-7** INSERT INTO 参数

参数	描述
tableIndentifier	需要执行INSERT命令的Hudi表的名称。
select query	查询语句。

注意事项

- 写入模式：Hudi对于设置了主键的表支持三种写入模式，用户可以设置参数hoodie.sql.insert.mode来指定Insert模式，默认为upsert。
```
hoodie.sql.insert.mode = upsert
```
  - strict模式，Insert 语句将保留 COW 表的主键唯一性约束，不允许重复记录。如果在插入过程中已经存在记录，则会为COW表执行HoodieDuplicateKeyException；对于MOR表，该模式与upsert模式行为一致。
  - non-strict模式，对主键表采用insert处理。
  - upsert模式，对于主键表的重复值进行更新操作。
- 在提交Spark SQL作业时，用户可以在设置中配置以下参数，切换bulk insert作为Insert语句的写入方式。
```
hoodie.sql.bulk.insert.enable = true
hoodie.sql.insert.mode = non-strict
```


版权所有 © 华为技术有限公司

- 也可以设置hoodie.datasource.write.operation的来控制insert语句的写入方式，可选包括bulk_insert、insert、upsert。（注意：会覆盖配置的hoodie.sql.insert.mode的结果）
  ```
  hoodie.datasource.write.operation = upsert
  ```

示例

```
insert into h0 select 1, 'a1', 20;

-- insert static partition
insert into h_p0 partition(dt = '2021-01-02') select 1, 'a1';

-- insert dynamic partition
insert into h_p0 select 1, 'a1', dt;

-- insert dynamic partition
insert into h_p1 select 1 as id, 'a1', '2021-01-03' as dt, '19' as hh;

-- insert overwrite table
insert overwrite table h0 select 1, 'a1', 20;

-- insert overwrite table with static partition
insert overwrite h_p0 partition(dt = '2021-01-02') select 1, 'a1';

-- insert overwrite table with dynamic partition
insert overwrite table h_p1 select 2 as id, 'a2', '2021-01-03' as dt, '19' as hh;
```

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	INSERT_INTO_TABLE

- 细粒度权限：dli:table:insertIntoTable

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以确认任务状态成功，运行QUERY语句查看表中写入的数据。

5.2.3 MERGE INTO

命令功能

通过MERGE INTO命令，根据一张表或子查询的连接条件对另外一张表进行查询，连接条件匹配上的进行UPDATE或DELETE，无法匹配的执行INSERT。这个语法仅需要一次全表扫描就完成了全部同步工作，执行效率要高于INSERT＋UPDATE。

命令格式

MERGE INTO *tableIdentifier AS target_alias*

USING *(sub_query | tableIdentifier) AS source_alias*

ON *<merge_condition>*

[WHEN MATCHED [AND <condition>] THEN <matched_action>]

[WHEN MATCHED [AND <condition>] THEN <matched_action>]

[WHEN NOT MATCHED [AND <condition>] THEN <not_matched_action>]

<merge_condition> =A equal bool condition

<matched_action> =

DELETE |

*UPDATE SET * |*

UPDATE SET column1 = expression1 [, column2 = expression2 ...]

<not_matched_action> =

*INSERT * |*

INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...])

参数描述

表 5-8 UPDATE 参数

参数	描述
tableIdentifier	在其中执行MergeInto操作的Hudi表的名称。
target_alias	目标表的别名。
sub_query	子查询。
source_alias	源表或源表达式的别名。
merge_condition	将源表或表达式和目标表关联起来的条件
condition	过滤条件，可选。
matched_action	当满足条件时进行Delete或Update操作
not_matched_action	当不满足条件时进行Insert操作

注意事项

1. merge-on condition当前只支持主键列。
2. 当前仅支持对COW表进行部分字段更新，且更新值必须包含预合并列，MOR表需要在Update语法中给出全部字段。

示例

- 部分字段更新

```
create table h0(id int, comb int, name string, price int) using hudi options(primaryKey = 'id',
preCombineField = 'comb') LOCATION 'obs://bucket/path/h0';
```

```
create table s0(id int, comb int, name string, price int) using hudi options(primaryKey = 'id',
preCombineField = 'comb') LOCATION 'obs://bucket/path/s0';
insert into h0 values(1, 1, 1, 1);
insert into s0 values(1, 1, 1, 1);
insert into s0 values(2, 2, 2, 2);
//写法1
merge into h0 using s0
on h0.id = s0.id
when matched then update set h0.id = s0.id, h0.comb = s0.comb, price = s0.price * 2;
//写法2
merge into h0 using s0
on h0.id = s0.id
when matched then update set id = s0.id,
name = h0.name,
comb = s0.comb + h0.comb,
price = s0.price + h0.price;
```

- 缺省字段更新和插入

```
create table h0(id int, comb int, name string, price int, flag boolean) using hudi
  options(primaryKey = 'id', preCombineField = 'comb') LOCATION 'obs://bucket/path/h0';
create table s0(id int, comb int, name string, price int, flag boolean) using hudi
  options(primaryKey = 'id', preCombineField = 'comb') LOCATION 'obs://bucket/path/s0';
insert into h0 values(1, 1, 1, 1, false);
insert into s0 values(1, 2, 1, 1, true);
insert into s0 values(2, 2, 2, 2, false);

merge into h0 as target
using (
select id, comb, name, price, flag from s0
) source
on target.id = source.id
when matched then update set *
when not matched then insert *;
```

- 多条件更新和删除

```
create table h0(id int, comb int, name string, price int, flag boolean) using hudi
  options(primaryKey = 'id', preCombineField = 'comb') LOCATION 'obs://bucket/path/h0';
create table s0(id int, comb int, name string, price int, flag boolean) using hudi
   options(primaryKey = 'id', preCombineField = 'comb') LOCATION 'obs://bucket/path/s0';
insert into h0 values(1, 1, 1, 1, false);
insert into h0 values(2, 2, 1, 1, false);
insert into s0 values(1, 1, 1, 1, true);
insert into s0 values(2, 2, 2, 2, false);
insert into s0 values(3, 3, 3, 3, false);

merge into h0
using (
select id, comb, name, price, flag from s0
) source
on h0.id = source.id
when matched and flag = false then update set id = source.id, comb = h0.comb + source.comb, price =
source.price * 2
when matched and flag = true then delete
when not matched then insert *;
```

系统响应

可以检查任务状态是否成功，查看任务日志确认有无异常。

5.2.4 UPDATE

命令功能

UPDATE命令根据列表达式和可选的过滤条件更新Hudi表。

命令格式

UPDATE *tableIdentifier SET column = EXPRESSION(,column = EXPRESSION) [WHERE boolExpression]*

参数描述

表 **5-9** UPDATE 参数

参数	描述
tableIdentifier	在其中执行更新操作的Hudi表的名称。
column	待更新的目标列。
EXPRESSION	需在目标表中更新的源表列值的表达式。
boolExpression	过滤条件表达式。

示例

```
update h0 set price = price + 20 where id = 1;
update h0 set price = price *2, name = 'a2' where id = 2;
```

系统响应

可以确认任务状态成功，运行QUERY语句查看表中数据已被更新。

5.2.5 DELETE

命令功能

DELETE命令从Hudi表中删除记录。

命令格式

DELETE from *tableIdentifier [WHERE boolExpression]*

参数描述

表 **5-10** DELETE 参数

参数	描述
tableIdentifier	在其中执行删除操作的Hudi表的名称。
boolExpression	删除项的过滤条件

示例

- 示例1：
```
delete from h0 where column1 = 'country';
```
- 示例2：
```
delete from h0 where column1 IN ('country1', 'country2');
```
- 示例3：
```
delete from h0 where column1 IN (select column11 from sourceTable2);
```
- 示例4：
```
delete from h0 where column1 IN (select column11 from sourceTable2 where column1 = 'xxx');
```
- 示例5：
```
delete from h0;
```

系统响应

可以确认任务状态成功，运行QUERY语句查看表中对应数据已被删除。

5.2.6 COMPACTION

命令功能

压缩(compaction)用于在 MergeOnRead表将基于行的log日志文件转化为parquet列式数据文件，用于加快记录的查找。

命令格式

SCHEDULE COMPACTION on *tableIdentifier* |tablelocation;

SHOW COMPACTION on *tableIdentifier* |tablelocation;

RUN COMPACTION on *tableIdentifier* |tablelocation [at instant-time];

参数描述

表 5-11 COMPACTION 参数

参数	描述
tableIdentifier	在其中执行删除操作的Hudi表的名称。
tablelocation	Hudi表的存储路径
instant-time	执行show compaction命令可以看到instant-time

示例

```
schedule compaction  on h1;
show compaction on h1;
run compaction on h1 at 20210915170758;

schedule compaction  on 'obs://bucket/path/h1';
run compaction on 'obs://bucket/path/h1';
```

 版权所有 © 华为技术有限公司

注意事项

- 使用API方式对SQL创建的Hudi表触发Compaction时需要添加参数**hoodie.payload.ordering.field**为**preCombineField**的值。
- 使用由DLI提供的元数据服务时，本命令不支持使用OBS路径。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.2.7 ARCHIVELOG

命令功能

用于根据配置对Timeline上的Instant进行归档，并从Timeline上将已归档的Instant删除，以减少Timeline的操作压力。

命令格式

RUN ARCHIVELOG ON tableIdentifier;

RUN ARCHIVELOG ON tablelocation;

参数描述

表 **5-12** 参数描述

参数	描述
tableIdentifier	Hudi表的名称
tablelocation	Hudi表的存储路径

示例

```
run archivelog on h1;
run archivelog on "obs://bucket/path/h1";
```

注意事项

- 首先需要执行clean命令，在clean命令清理了历史的数据文件后，Timeline上与清理的数据文件对应的Instant才允许归档。
- 不管是否进行compaction操作，至少会保留hoodie.compact.inline.max.delta.commits个Instant不会被归档，以此保证有足够的Instant去触发compation schedule。
- 使用由DLI提供的元数据服务时，本命令不支持使用OBS路径。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.2.8 CLEAN

命令功能

用于根据配置对Timeline上的Instant进行clean，删除老旧的历史版本文件，以减少hudi表的数据存储及读写压力。

命令格式

RUN CLEAN ON tableIdentifier;

RUN CLEAN ON tablelocation;

参数描述

表 5-13 参数描述

参数	描述
tableIdentifier	Hudi表的名称
tablelocation	Hudi表的存储路径

示例

```
run clean on h1;
run clean on "obs://bucket/path/h1";
```

注意事项

- 对表执行clean操作时需要表的owner才可以执行。
- 如果需要修改clean默认的参数，需要在执行SQL时，在设置中配置需要保留的commit数量等参数，参见**Hudi常见配置参数**。
- 使用由DLI提供的元数据服务时，本命令不支持使用OBS路径。

系统响应

可以检查任务状态是否成功，查看任务日志确认有无异常。

5.2.9 CLEANARCHIVE

命令功能

用于对Hudi表的归档文件进行清理，以减少Hudi表的数据存储及读写压力。

命令格式

- 按文件容量进行清理，需要配置参数：
  ```
  hoodie.archive.file.cleaner.policy = KEEP_ARCHIVED_FILES_BY_SIZE;
  hoodie.archive.file.cleaner.size.retained = 5368709120;
  ```
  提交SQL

```
run cleanarchive on tableIdentifier|tablelocation;
```

- 按保留时间进行清理，需要配置参数：

```
hoodie.archive.file.cleaner.policy = KEEP_ARCHIVED_FILES_BY_DAYS;
hoodie.archive.file.cleaner.days.retained = 30;
```

  提交SQL

```
run cleanarchive on tableIdentifier|tablelocation;
```

参数描述

表 5-14 参数描述

参数	描述
tableIdentifier	Hudi表的名称。
tablelocation	Hudi表的存储路径。
hoodie.archive.file.cleaner.policy	清理归档文件的策略：目前仅支持KEEP_ARCHIVED_FILES_BY_SIZE和KEEP_ARCHIVED_FILES_BY_DAYS两种策略，默认策略为KEEP_ARCHIVED_FILES_BY_DAYS。 ● KEEP_ARCHIVED_FILES_BY_SIZE策略可以设置归档文件占用的存储空间大小 ● KEEP_ARCHIVED_FILES_BY_DAYS策略可以清理超过某个时间点之外的归档文件
hoodie.archive.file.cleaner.size.retained	当清理策略为KEEP_ARCHIVED_FILES_BY_SIZE时，该参数可以设置保留多少字节大小的归档文件，默认值5368709120字节（5G）。
hoodie.archive.file.cleaner.days.retained	当清理策略为KEEP_ARCHIVED_FILES_BY_DAYS时，该参数可以设置保留多少天以内的归档文件，默认值30（天）。

注意事项

- 归档文件，没有备份，删除之后无法恢复。
- 使用由DLI提供的元数据服务时，本命令不支持使用OBS路径。

系统响应

可以检查任务状态是否成功，查看任务日志确认有无异常。

5.3 Hudi CALL COMMAND 语法说明

5.3.1 CLEAN_FILE

命令功能

用于清理Hudi表目录下的无效数据文件。

命令格式

call clean_file(table => '[table_name]', mode=>'[op_type]', backup_path=>'[backup_path]', start_instant_time=>'[start_time]', end_instant_time=>'[end_time]');

参数描述

表 5-15 参数描述

参数	描述
table_name	需要清理无效数据文件的Hudi表的表名，必选。
op_type	命令运行模式，可选，默认值为dry_run，取值：dry_run、repair、undo、query。 dry_run：显示需要清理的无效数据文件。 repair：显示并清理无效的数据文件。 undo：恢复已清理的数据文件 query：显示已执行清零操作的备份目录。
backup_path	运行模式为undo时有效，需要恢复数据文件的备份目录，必选。
start_time	运行模式为dry_run、repair时有效，产生无效数据文件的开始时间，可选，默认不限制开始时间。
end_time	运行模式为dry_run、repair时有效，产生无效数据文件的结束时间，可选，默认不限制结束时间。

示例

```
call clean_file(table => 'h1', mode=>'repair');
call clean_file(table => 'h1', mode=>'dry_run');
call clean_file(table => 'h1', mode=>'query');
call clean_file(table => 'h1', mode=>'undo', backup_path=>'obs://bucket/hudi/h1/.hoodie/.cleanbackup/
hoodie_repair_backup_20230527');
```

注意事项

命令只清理无效的parquet文件。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.2 SHOW_TIME_LINE

命令功能

查看当前生效或者被归档的Hudi time line以及某个指定instant time的详细内容。

命令格式

- 查看某个表生效的time line列表：

  call show_active_instant_list(table => '[table_name]');
- 查看某个表某个时间戳后的生效的time line列表：

  call show_active_instant_list(table => '[table_name]', instant => '[instant]');
- 查看某个表生效的某个instant信息：

  call show_active_instant_detail(table => '[table_name]', instant => '[instant]');
- 查看某个表已被归档的instant time line列表：

  call show_archived_instant_list(table => '[table_name]');
- 查看某个表某个时间戳后的已被归档的instant time line列表：

  call show_archived_instant_list(table => '[table_name]', instant => '[instant]');
- 查看某个表已被归档的instant信息：

  call show_archived_instant_detail(table => '[table_name]', instant => '[instant]');

参数描述

表 5-16 参数描述

参数	描述
table_name	需要查询的表的表名，支持database.tablename格式
instant	需要查询的instant time时间戳

示例

```
call show_active_instant_detail(table => 'hudi_table1', instant => '20230913144936897');
```

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	SELECT

- 细粒度权限：dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.3 SHOW_HOODIE_PROPERTIES

命令功能

查看指定hudi表的hoodie.properties文件中的配置。

命令格式

call show_hoodie_properties(table => '[table_name]');

参数描述

表 5-17 参数描述

参数	描述
table_name	需要查询的表的表名，支持database.tablename格式

示例

```
call show_hoodie_properties(table => "hudi_table5");
```

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	SELECT

- 细粒度权限：dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.4 ROLL_BACK

命令功能

用于回滚指定的commit。

命令格式

call rollback_to_instant(table => '[table_name]', instant_time => '[instant]');

参数描述

表 5-18 参数描述

参数	描述
table_name	需要回滚的Hudi表的表名，必选
instant	需要回滚的Hudi表的commit instant时间戳，必选

示例

```
call rollback_to_instant(table => 'h1', instant_time=>'20220915113127525');
```

注意事项

- 只能依次回滚最新的commit时间戳，可以通过SHOW_TIME_LINE命令检查最新的instant time。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.5 CLUSTERING

命令功能

对Hudi表进行clustering操作，具体作用可以参考**Hudi Clustering操作说明**章节。

命令格式

- 执行clustering：

  call run_clustering(table=>'[table]', path=>'[path]', predicate=>'[predicate]', order=>'[order]');

- 查看clustering计划：

  call show_clustering(table=>'[table]', path=>'[path]', limit=>[limit]);

参数描述

表 5-19 参数描述

参数	描述	是否必填
table	需要查询的表的表名，支持database.tablename格式	table，path须选填其中之一

参数	描述	是否必填
path	需要查询的表的路径	table，path须选填其中之一
predicate	需要定义的谓语句，筛选需要Clustering的分区	否
order	指定clustering的排序字段	否
limit	展示查询结果的条数	否

示例

```
call show_clustering(table => 'hudi_table1');

call run_clustering(table => 'hudi_table1', predicate => '(ts >= 1006L and ts < 1008L) or ts >= 1009L', order => 'ts');

call run_clustering(path => 'obs://bucket/path/hudi_test2', predicate => "dt = '2021-08-28'", order => 'id');
```

注意事项

- table与path参数必须存在一个，否则无法判断需要执行clustering的表。
- 使用由DLI提供的元数据服务时，本命令仅支持配置table参数，不支持配置path参数。
- 如果需要对指定分区进行clustering，参考格式：predicate => "dt = '2023-08-28'"

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.6 CLEANING

命令功能

对Hudi表进行cleaning操作，具体作用可以参考**Hudi Clean操作说明**章节。

命令格式

call run_clean(table=>'[table]', clean_policy=>'[clean_policy]', retain_commits=>'[retain_commits]', hours_retained=> '[hours_retained]', file_versions_retained=> '[file_versions_retained]');

参数描述

表 5-20 参数描述

参数	描述	是否必填
table	需要查询表的表名，支持database.tablename格式	是
clean_policy	清理老版本数据文件的策略，默认KEEP_LATEST_COMMITS	否
retain_commits	仅对KEEP_LATEST_COMMITS策略有效	否
hours_retained	仅对KEEP_LATEST_BY_HOURS策略有效	否
file_version_retained	仅对KEEP_LATEST_FILE_VERSIONS策略有效	否

示例

```
call run_clean(table => 'hudi_table1');

call run_clean(table => 'hudi_table1', retain_commits => 2);

call run_clean(table => 'hudi_table1', clean_policy => 'KEEP_LATEST_FILE_VERSIONS', file_version_retained
=> 1);
```

注意事项

cleaning操作只有在满足触发条件后才会对分区的老版本数据文件进行清理，不满足触发条件虽然执行命令成功也不会执行清理。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.7 COMPACTION

命令功能

对Hudi表进行compaction操作，具体作用可以参考**Hudi Compaction操作说明**章节。

命令格式

call run_compaction(op => '[op]', table=>'[table]', path=>'[path]', timestamp=>'[timestamp]');

参数描述

表 5-21 参数描述

参数	描述	是否必填
op	生成compaction计划（op指定为“schedule”），或者执行已经生成的compaction计划（op指定为“run”）	是
table	需要查询表的表名，支持database.tablename格式	table，path须选填其中之一
path	需要查询表的路径	table，path须选填其中之一
timestamp	在op指定为“run”时，可以指定timestamp来执行该时间戳对应的compaction计划以及该时间戳之前未执行的compaction计划	否

示例

```
call run_compaction(table => 'hudi_table1', op => 'schedule');
call run_compaction(table => 'hudi_table1', op => 'run');
call run_compaction(table => 'hudi_table1', op => 'run', timestamp => 'xxx');
call run_compaction(path => 'obs://bucket/path/hudi_table1', op => 'run', timestamp => 'xxx');
```

注意事项

compaction操作仅支持MOR表。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.8 SHOW_COMMIT_FILES

命令功能

查看指定的instant一共更新或者插入了多个文件。

命令格式

call show_commit_files(table=>'[table]', instant_time=>'[instant_time]', limit=>[limit]);

参数描述

表 5-22 参数描述

参数	描述	是否必填
table	需要查询表的表名，支持database.tablename格式	是
instant_time	某次commit对应的时间戳	是
limit	限制返回结果的条数	否

示例

```
call show_commit_files(table=>'hudi_mor', instant_time=>'20230216144548249');
call show_commit_files(table=>'hudi_mor', instant_time=>'20230216144548249', limit=>1);
```

返回结果

参数	描述
action	instant_time对应的commit所属的action类型，如compaction、deltacommit、clean等
partition_path	指定的instant所更新或插入的文件位于哪个分区
file_id	指定的instant所更新或插入的文件的ID
previous_commit	指定的instant所更新或插入的文件的文件名中的时间戳
total_records_updated	该文件中多少个record被更新
total_records_written	该文件中新插入了多少个record
total_bytes_written	该文件新增多少bytes的数据
total_errors	指定的instant在更新或者插入过程中的报错
file_size	该文件的大小（bytes）

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	SELECT

- 细粒度权限：dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.9 SHOW_FS_PATH_DETAIL

命令功能

查看指定的FS路径的统计数据

命令格式

call show_fs_path_detail(path=>'[path]', is_sub=>'[is_sub]', sort=>'[sort]');

参数描述

表 5-23 参数描述

参数	描述	是否必填
path	需要查询的FS的路径	是
is_sub	默认false，false表示统计指定目录的信息，true表示统计指定目录的子目录的信息	否
sort	默认true，true表示根据storage_size排序结果，false表示根据文件数量排序结果	否

示例

```
call show_fs_path_detail(path=>'obs://bucket/path/hudi_mor/dt=2021-08-28', is_sub=>false, sort=>true);
```

返回结果

参数	描述
path_num	指定目录的子目录数量
file_num	指定目录的文件数量
storage_size	该目录的Size（bytes）
storage_size(unit)	该目录的Size（KB）
storage_path	指定目录的完整FS绝对路径

参数	描述
space_consumed	返回文件/目录在集群中占用的实际空间，即它考虑了为集群设置的复制因子
quota	名称配额（名称配额是对当前目录树中的文件和目录名称数量的硬性限制）
space_quota	空间配额（空间配额是对当前目录树中的文件所使用的字节数量的硬性限制）

注意事项

- 使用由DLI提供的元数据服务时，不支持本命令。

权限需求

由DLI提供的元数据服务

- SQL权限：不支持。

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.10 SHOW_LOG_FILE

命令功能

查看log文件的meta和record信息。

命令格式

- 查看meta：
  call show_logfile_metadata(table => '[table]', log_file_path_pattern => '[log_file_path_pattern]', limit => [limit])
- 查看record：
  call show_logfile_records(table => '[table]', log_file_path_pattern => '[log_file_path_pattern]', merge => '[merge]', limit => [limit])

参数描述

表 5-24 参数描述

参数	描述	是否必填
table	需要查询表的表名，支持database.tablename格式	是

参数	描述	是否必填
log_file_path_pattern	log file的路径，支持正则匹配	否
merge	执行show_logfile_records时，通过merge控制是否将多个log file中的record合并在一起返回	否
limit	限制返回结果的条数	否

示例

```
call show_logfile_metadata(table => 'hudi_mor', log_file_path_pattern => 'obs://bucket/path/hudi_mor/
dt=2021-08-28/.*?log.*?');
call show_logfile_records(table => 'hudi_mor', log_file_path_pattern => 'obs://bucket/path/hudi_mor/
dt=2021-08-28/.*?log.*?', merge => false, limit => 1);
```

注意事项

- 仅MOR表会用到此命令。

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	SELECT

- 细粒度权限：dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.3.11 SHOW_INVALID_PARQUET

命令功能

查看执行路径下损坏的parquet文件。

命令格式

call show_invalid_parquet(path => 'path')

参数描述

表 5-25 参数描述

参数	描述	是否必填
path	需要查询的FS路径	是

示例

```
call show_invalid_parquet(path => 'obs://path/hudi_table/dt=2021-08-28');
```

注意事项

- 使用由DLI提供的元数据服务时，不支持本命令。

权限需求

由DLI提供的元数据服务

- SQL权限：

database	table
无	SELECT

- 细粒度权限：dli:table:select

由LakeFormation提供的元数据服务，权限配置详见LakeFormation文档。

系统响应

可以检查任务状态是否成功，查看任务结果，查看任务日志确认有无异常。

5.4 Schema 演进语法说明

功能介绍

该能力用于支持Spark SQL对Hudi表的列进行Alter变更，使用该能力前必须开启Schema演进。

Schema 演进支持的范围

Schema演进支持范围：

- 支持列（包括嵌套列）相关的增、删、改、位置调整等操作。
- 不支持对分区列做演进。
- 不支持对Array类型的嵌套列进行增、删、列操作。

5.4.1 ALTER COLUMN

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ALTER TABLE ... ALTER COLUMN语法用于修改当前列属性包括列类型、列位置、列comment。

命令语法

ALTER TABLE *tableName* **ALTER**

[COLUMN] *col_old_name* **TYPE** *column_type*

[COMMENT] *col_comment*

[FIRST|AFTER] *column_name*

参数描述

表 **5-26** ALTER COLUMN 参数描述

参数	描述
tableName	表名。
col_old_name	待修改的列名。
column_type	目标列类型。
col_comment	列comment。
column_name	位置修改参照列，例如：AFTER column_name的语义是要将待修改列放到参照列column_name之后。

示例

- 列类型修改
  ```
  ALTER TABLE table1 ALTER COLUMN a.b.c TYPE bigint
  ```
  a.b.c 表示嵌套列全路径，嵌套列具体规则见**ADD COLUMNS**。

  当前类型修改支持：
  - int => long/float/double/string/decimal
  - long => float/double/string/decimal
  - float => double/String/decimal
  - double => String/Decimal
  - Decimal => Decimal/String

  - String => date/decimal
  - date => String
- 其他修改

```
ALTER TABLE table1 ALTER COLUMN a.b.c DROP NOT NULL
ALTER TABLE table1 ALTER COLUMN a.b.c COMMENT 'new comment'
ALTER TABLE table1 ALTER COLUMN a.b.c FIRST
ALTER TABLE table1 ALTER COLUMN a.b.c AFTER x
```

  a.b.c 表示嵌套列全路径，嵌套列具体规则见**ADD COLUMNS**。

系统响应

通过运行**DESCRIBE**命令，可显示修改的列。

5.4.2 ADD COLUMNS

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ADD COLUMNS命令用于为现有表添加新列。

命令语法

ALTER TABLE *tableName* **ADD COLUMNS** *(col_spec[, col_spec ...])*

参数描述

表 **5-27** ADD COLUMNS 参数描述

参数	描述
tableName	表名。

参数	描述
col_spec	可由[col_name][col_type][nullable][comment][col_position]五部分组成。 ● col_name：新增列名，必须指定。 给嵌套列添加新的子列需要指定子列的全名称： – 添加新列col1到STURCT类型嵌套列users struct<name: string, age: int>，新列名称需要指定为users.col1。 – 添加新列col1到MAP类型嵌套列memeber map<string, struct<n: string, a: int>>，新列名称需要指定为member.value.col1。 – 添加新列col2到ARRAY类型嵌套列arraylike array<struct<a1: string, a2: int>>，新列名称需要指定为arraylike.element.col2。 ● col_type：新增列类型，必须指定。 ● nullable：新增列是否可以为空，可以缺省。 ● comment：新增列comment，可以缺省。 ● col_position：列添加位置包括FIRST、AFTER origin_col两种，指定FIRST新增列将会被添加到表的第一列。AFTER origin_col新增列将会被加入到原始列origin_col之后，可以缺省。FIRST只能再嵌套列添加新的子列时使用，禁止top-level列使用FIRST，AFTER没有限制。

示例

```
alter table h0 add columns(ext0 string);
alter table h0 add columns(new_col int not null comment 'add new column' after col1);
alter table complex_table add columns(col_struct.col_name string comment 'add new column to a struct
col' after col_from_col_struct);
```

系统响应

通过运行**DESCRIBE**命令，可显示新添加的列。

5.4.3 RENAME COLUMN

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ALTER TABLE ... RENAME COLUMN语法用于修改列名称。

命令语法

ALTER TABLE *tableName* **RENAME COLUMN** *old_columnName* **TO** *new_columnName*

参数描述

表 5-28 参数描述

参数	描述
tableName	表名。
old_columnName	旧列名。
new_columnName	新列名。

示例

```
ALTER TABLE table1 RENAME COLUMN a.b.c TO x
```

a.b.c 表示嵌套列全路径，嵌套列具体规则见**ADD COLUMNS**。

说明

修改列名后自动同步到列comment中，comment的形式为：rename oldName to newName。

系统响应

通过运行**DESCRIBE**命令查看表列修改。

5.4.4 RENAME TABLE

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ALTER TABLE ... RENAME语法用于修改表名。

命令语法

ALTER TABLE *tableName* **RENAME TO** *newTableName*

参数描述

表 **5-29** RENAME 参数描述

参数	描述
tableName	表名。
newTableName	新表名。

示例

```
ALTER TABLE table1 RENAME TO table2
```

系统响应

通过运行**SHOW TABLES**查看新的表名。

5.4.5 SET

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ALTER TABLE ... SET|UNSET语法用于修改表属性。

命令语法

ALTER TABLE *tableName* **SET|UNSET** *tblproperties*

参数描述

表 **5-30** 参数描述

参数	描述
tableName	表名。
tblproperties	表属性。

示例

```
ALTER TABLE table SET TBLPROPERTIES ('table_property' = 'property_value')
ALTER TABLE table UNSET TBLPROPERTIES [IF EXISTS] ('comment', 'key')
```

系统响应

通过运行**DESCRIBE**命令查看表属性修改。

5.4.6 DROP COLUMN

功能开启

配置参数：

```
hoodie.schema.evolution.enable=true
```

命令功能

ALTER TABLE ... DROP COLUMN语法用于删除列。

命令语法

ALTER TABLE *tableName* **DROP COLUMN|COLUMNS** *cols*

参数描述

表 5-31 DROP COLUMN 参数描述

参数	描述
tableName	表名。
cols	待删除列，可以指定多个。

示例

```
ALTER TABLE table1 DROP COLUMN a.b.c
ALTER TABLE table1 DROP COLUMNS a.b.c, x, y
```

a.b.c 表示嵌套列全路径，嵌套列具体规则见**ADD COLUMNS**。

系统响应

通过运行**DESCRIBE**命令，可查看删除列。

5.5 配置 Hudi 数据列默认值

该特性允许用户在给表新增列时，设置列的默认值。查询历史数据时新增列返回默认值。

使用约束

- 新增列在设置默认值前，如果数据已经进行了重写，则查询历史数据不支持返回列的默认值，返回NULL。数据入库、更新、执行Compaction、Clustering都会导致部分或全部数据重写。
- 列的默认值设置要与列的类型一致，如不一致会进行类型强转，导致默认值精度丢失或者默认值为NULL。
- 历史数据的默认值与列第一次设置的默认值一致，多次修改列的默认值不会影响历史数据的查询结果。

- 设置默认值后rollback不能回滚默认值配置。
- Spark SQL暂不支持查看列默认值信息，可以通过执行**show create table** SQL查看。
- 不支持默认缺省列的写入方式，写入时必须**指定列名**。

支持范围

当前仅支持int、bigint、float、double、decimal、string、date、timestamp、boolean、binary类型，其他类型不支持。

表 5-32 引擎支持矩阵

引擎	DDL操作	写操作支持	读操作支持
SparkSQL	Y	Y	Y
Spark DataSource	N	N	Y
Flink	N	N	Y
HetuEngine	N	N	Y
Hive	N	N	Y

示例

SQL语法具体参考**DLI Hudi SQL语法参考**章节。

示例：

- 建表指定列默认值
```
create table if not exists h3(
id bigint,
name string,
price double default 12.34
) using hudi
options (
primaryKey = 'id',
type = 'mor',
preCombineField = 'name'
);
```
- 添加列指定列默认值
```
alter table h3 add columns(col1 string default 'col1_value');
alter table h3 add columns(col2 string default 'col2_value', col3 int default 1);
```
- 修改列默认值
```
alter table h3 alter column price set default 14.56;
```
- 插入数据使用列默认值，需要指定写入的列名，和插入的数据一一对应
```
insert into h3(id, name) values(1, 'aaa');
insert into h3(id, name, price) select 2, 'bbb', 12.5;
```

 版权所有 © 华为技术有限公司

6 Spark datasource API 语法参考

Spark jar作业提交方式请参考**在DLI使用Hudi提交Spark Jar作业**

6.1 API 语法说明

设置写入方式

Hudi通过hoodie.datasource.write.operation参数设置写入模式。

- insert： 该操作不需要通过索引去查询具体更新的文件分区，因此它的速度比upsert快。当不包含更新数据时建议使用该操作，如果存在更新数据使用该操作会出现重复数据。
- bulk_insert：该操作会对主键进行排序后直接以写普通parquet表的方式插入Hudi表，该操作性能是最高的，但是无法控制小文件，而upsert和insert操作可以很好的控制小文件。
- upsert： 默认操作类型。Hudi会根据主键进行判断即将插入的数据是否包含更新数据，如果包含则执行upsert，否则执行insert。

说明

- 由于insert时不会对主键进行排序，所以初始化数据集不建议使用insert，建议用bulk_insert。
- 确定数据都为新增数据时建议使用insert，当存在更新数据时建议使用upsert。

例：bulk_insert写COW无分区表

```
df.write.format("org.apache.hudi").
option("hoodie.datasource.write.table.type", COW_TABLE_TYPE_OPT_VAL).
option("hoodie.datasource.write.precombine.field", "update_time").
option("hoodie.datasource.write.recordkey.field", "id").
option("hoodie.datasource.write.partitionpath.field", "").
option("hoodie.datasource.write.operation", "bulk_insert").
option("hoodie.table.name", tableName).
option("hoodie.write.lock.provider", "com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider").
option("hoodie.datasource.write.keygenerator.class",
"org.apache.hudi.keygen.NonpartitionedKeyGenerator").
option("hoodie.datasource.hive_sync.enable", "true").
option("hoodie.datasource.hive_sync.partition_fields", "").
option("hoodie.datasource.hive_sync.partition_extractor_class",
"org.apache.hudi.hive.NonPartitionedExtractor").
option("hoodie.datasource.hive_sync.database", databaseName).
option("hoodie.datasource.hive_sync.table", tableName).
```

```
option("hoodie.datasource.hive_sync.use_jdbc", "false").
option("hoodie.bulkinsert.shuffle.parallelism", 4).
mode(SaveMode.Overwrite).
save(basePath)
```

设置分区

- 多级分区

配置项	说明
hoodie.datasource.write.partitionpath.field	配置为多个业务字段，用逗号分隔。
hoodie.datasource.hive_sync.partition_fields	和hoodie.datasource.write.partitionpath.field的分区字段保持一致。
hoodie.datasource.write.keygenerator.class	配置为org.apache.hudi.keygen.ComplexKeyGenerator。
hoodie.datasource.hive_sync.partition_extractor_class	配置为org.apache.hudi.hive.MultiPartKeysValueExtractor。

例：创建分区为p1/p2/p3的多级分区COW表

```
df.write.format("org.apache.hudi").
option("hoodie.datasource.write.table.type", COW_TABLE_TYPE_OPT_VAL).
option("hoodie.datasource.write.precombine.field", "update_time").
option("hoodie.datasource.write.recordkey.field", "id").
option("hoodie.datasource.write.partitionpath.field", "year,month,day").
option("hoodie.datasource.write.operation", "bulk_insert").
option("hoodie.table.name", tableName).
option("hoodie.write.lock.provider", "com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider").
option("hoodie.datasource.write.keygenerator.class",
"org.apache.hudi.keygen.ComplexKeyGenerator").
option("hoodie.datasource.hive_sync.enable", "true").
option("hoodie.datasource.hive_sync.partition_fields", "year,month,day").
option("hoodie.datasource.hive_sync.partition_extractor_class",
"org.apache.hudi.hive.MultiPartKeysValueExtractor").
option("hoodie.datasource.hive_sync.database", databaseName).
option("hoodie.datasource.hive_sync.table", tableName).
option("hoodie.datasource.hive_sync.use_jdbc", "false").
mode(SaveMode.Overwrite).
save(basePath)
```

- 单分区

配置项	说明
hoodie.datasource.write.partitionpath.field	配置为一个业务字段。
hoodie.datasource.hive_sync.partition_fields	和hoodie.datasource.write.partitionpath.field分区字段保持一致。

版权所有 © 华为技术有限公司

配置项	说明
hoodie.datasource.write.keygenerator.class	默认可以配置为 org.apache.hudi.keygen.SimpleKeyGenerator 和 org.apache.hudi.keygen.ComplexKeyGenerator
hoodie.datasource.hive_sync.partition_extractor_class	配置为 org.apache.hudi.hive.MultiPartKeysValueExtractor。

例：创建分区为p1的单分区的MOR表

```
df.write.format("org.apache.hudi").
option("hoodie.datasource.write.table.type", MOR_TABLE_TYPE_OPT_VAL).
option("hoodie.datasource.write.precombine.field", "update_time").
option("hoodie.datasource.write.recordkey.field", "id").
option("hoodie.datasource.write.partitionpath.field", "create_time").
option("hoodie.datasource.write.operation", "bulk_insert").
option("hoodie.table.name", tableName).
option("hoodie.write.lock.provider", "com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider").
option("hoodie.datasource.write.keygenerator.class",
"org.apache.hudi.keygen.ComplexKeyGenerator").
option("hoodie.datasource.hive_sync.enable", "true").
option("hoodie.datasource.hive_sync.partition_fields", "create_time").
option("hoodie.datasource.hive_sync.partition_extractor_class",
"org.apache.hudi.hive.MultiPartKeysValueExtractor").
option("hoodie.datasource.hive_sync.database", databaseName).
option("hoodie.datasource.hive_sync.table", tableName).
option("hoodie.datasource.hive_sync.use_jdbc", "false").
mode(SaveMode.Overwrite).
save(basePath)
```

- 无分区

配置项	说明
hoodie.datasource.write.partitionpath.field	配置为空字符串。
hoodie.datasource.hive_sync.partition_fields	配置为空字符串。
hoodie.datasource.write.keygenerator.class	配置为 org.apache.hudi.keygen.NonpartitionedKeyGenerator。
hoodie.datasource.hive_sync.partition_extractor_class	配置为 org.apache.hudi.hive.NonPartitionedExtractor。

例：创建无分区COW表

```
df.write.format("org.apache.hudi").
option("hoodie.datasource.write.table.type", COW_TABLE_TYPE_OPT_VAL).
option("hoodie.datasource.write.precombine.field", "update_time").
option("hoodie.datasource.write.recordkey.field", "id").
option("hoodie.datasource.write.partitionpath.field", "").
```


版权所有 © 华为技术有限公司

```
option("hoodie.datasource.write.operation", "bulk_insert").
option("hoodie.table.name", tableName).
option("hoodie.write.lock.provider", "com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider").
option("hoodie.datasource.write.keygenerator.class",
"org.apache.hudi.keygen.NonpartitionedKeyGenerator").
option("hoodie.datasource.hive_sync.enable", "true").
option("hoodie.datasource.hive_sync.partition_fields", "").
option("hoodie.datasource.hive_sync.partition_extractor_class",
"org.apache.hudi.hive.NonPartitionedExtractor").
option("hoodie.datasource.hive_sync.database", databaseName).
option("hoodie.datasource.hive_sync.table", tableName).
option("hoodie.datasource.hive_sync.use_jdbc", "false").
mode(SaveMode.Overwrite).
save(basePath)
```

- 时间日期分区

配置项	说明
hoodie.datasource.write.partitionpath.field	配置为date类型字段，格式为yyyy/mm/dd。
hoodie.datasource.hive_sync.partition_fields	和hoodie.datasource.write.partitionpath.field分区字段保持一致。
hoodie.datasource.write.keygenerator.class	默认配置为org.apache.hudi.keygen.SimpleKeyGenerator，也可以配置为org.apache.hudi.keygen.ComplexKeyGenerator。
hoodie.datasource.hive_sync.partition_extractor_class	配置org.apache.hudi.hive.SlashEncodedDayPartitionValueExtractor。

说明

SlashEncodedDayPartitionValueExtractor存在以下约束：要求写入的日期格式为yyyy/mm/dd。

6.2 Hudi 锁配置说明

提交Spark jar作业时需要手动配置Hudi锁。

当使用DLI托管的元数据服务时，必须配置Hudi锁开启，且配置使用DLI提供的Hudi锁实现类：

配置项	配置值
hoodie.write.lock.provider	com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider

同样在使用LakeFormation服务提供的元数据服务时，需要配置并使用LakeFormation提供的Hudi锁实现类。

配置项	配置值
hoodie.write.lock.provider	org.apache.hudi.lakeformation.LakeCatMetastoreBasedLockProvider

注意

- 关闭Hudi锁，或者使用其他的锁实现类时，存在数据丢失/异常的风险。
- 在任何情况下，DLI不对因关闭Hudi锁，或者使用与元数据服务不匹配的锁实现类，而直接或间接导致的任何形式的损失或损害承担责任，包括但不限于商业利润损失、业务中断、数据丢失或其他财务损失。

版权所有 © 华为技术有限公司

7 数据管理维护

7.1 Hudi Compaction 操作说明

什么是 Compaction

Compaction用于合并mor表Base和Log文件，Compaction包含两个过程Schedule和Run。Schedule过程会在TimeLine里生成一个Compaction Plan，这个Compaction Plan会记录哪些parquet文件将会与哪些log文件进行合并，但是仅仅是一个Plan，没有去合并。Run过程会将TimeLine里的所有Compaction Plan一个一个去执行，一直到全部都执行完。

对于Merge-On-Read表，数据使用列式Parquet文件和行式Avro文件存储，更新被记录到增量文件，然后进行同步/异步compaction生成新版本的列式文件。Merge-On-Read表可减少数据摄入延迟，因而进行不阻塞摄入的异步Compaction很有意义。

如何执行 Compaction

1. 仅执行Schedule
   - Spark SQL（设置如下参数，写数据时触发）

     ```
     hoodie.compact.inline=true
     hoodie.schedule.compact.only.inline=true
     hoodie.run.compact.only.inline=false
     hoodie.compact.inline.max.delta.commits=5 // 默认值为5，根据业务场景指定
     ```

     随后执行任意写入SQL时，在满足条件后（同一个file slice下存在5个 delta log文件），会触发compaction。
   - Spark SQL（设置如下参数，手动触发1次）

     ```
     hoodie.compact.inline=true
     hoodie.schedule.compact.only.inline=true
     hoodie.run.compact.only.inline=false
     hoodie.compact.inline.max.delta.commits=5 // 默认值为5，根据业务场景指定
     ```

     随后手动执行SQL：

     ```
     schedule compaction on ${table_name}
     ```
   - SparkDataSource（option里设置如下参数，写数据时触发）

     hoodie.compact.inline=true

     hoodie.schedule.compact.only.inline=true

     hoodie.run.compact.only.inline=false

hoodie.compact.inline.max.delta.commits=5 // 默认值为5，根据业务场景指定

- Flink（with属性里设置如下参数，写数据时触发）

  compaction.async.enabled=false

  compaction.schedule.enabled=true

  compaction.delta_commits=5 // 默认值为5，根据业务场景指定

2. 仅执行Run
   - Spark SQL（设置如下参数，手动触发1次）
     ```
     hoodie.compact.inline=true
     hoodie.schedule.compact.only.inline=false
     hoodie.run.compact.only.inline=true
     ```
     随后执行如下SQL
     ```
     run compaction on ${table_name}
     ```
3. Schedule+Run同时执行

   如果TimeLine中没有Compaction Plan，就尝试生成一个Compaction Plan去执行。
   - Spark SQL（设置如下参数，随后执行任意写入SQL时，在满足条件时触发）
     ```
     hoodie.compact.inline=true
     hoodie.schedule.compact.only.inline=false
     hoodie.run.compact.only.inline=false
     hoodie.compact.inline.max.delta.commits=5 // 默认值为5，根据业务场景指定
     ```
   - SparkDataSource（option里设置如下参数，写数据时触发）

     hoodie.compact.inline=true

     hoodie.schedule.compact.only.inline=false

     hoodie.run.compact.only.inline=false

     hoodie.compact.inline.max.delta.commits=5 // 默认值为5，根据业务场景指
   - Flink（with属性里设置如下参数，写数据时触发）

     compaction.async.enabled=true

     compaction.schedule.enabled=false

     compaction.delta_commits=5 // 默认值为5，根据业务场景指定
4. 推荐方案
   - Spark/Flink流任务仅执行Schedule，然后另起一个Spark SQL任务定时仅执行Run。
   - Spark批任务可以直接同时执行Schedule + Run。

   说明

   为了保证入湖的最高效率，推荐使用同步产生compaction调度计划，异步执行compaction调度计划。

7.2 Hudi Clean 操作说明

什么是 Clean

Cleaning用于清理Hudi表不再需要的**老版本数据文件 (parquet文件或者log文件)**，减轻存储压力，提升list操作效率。

如何执行 Clean

1. 写完数据后clean
   - Spark SQL（设置如下参数，随后执行任意写入SQL时，在满足条件时触发）

     ```
     hoodie.clean.automatic=true
     hoodie.cleaner.commits.retained=10 // 默认值为10，根据业务场景指定
     ```

   - SparkDataSource（option里设置如下参数，写数据时触发）

     hoodie.clean.automatic=true

     hoodie.cleaner.commits.retained=10 // 默认值为10，根据业务场景指定

   - Flink（with属性里设置如下参数，写数据时触发）

     clean.async.enabled=true

     clean.retain_commits=10 // 默认值为10，根据业务场景指定

2. 手动触发1次clean
   - Spark SQL（set设置如下参数，手动触发1次）

   ```
   hoodie.clean.automatic=true
   hoodie.cleaner.commits.retained=10 // 默认值为10，根据业务场景指定
   ```

   随后执行SQL，当Timeline中有10个以上的Instant记录时，会触发clean

   ```
   run clean on ${table_name}
   ```

7.3 Hudi Archive 操作说明

什么是 Archive

Archive用户清理Hudi表的元数据文件（位于.hoodie目录下，格式为 ${时间戳}.${操作类型}.${操作状态}，比如20240622143023546.deltacommit.request）。对Hudi表进行的每次操作都会产生元数据文件，而元数据文件过多会导致性能问题，所以元数据文件数量最好控制在1000以内。

如何执行 Archive

1. 写完数据后archive
   - Spark SQL（set设置如下参数，写数据时触发）

     ```
     hoodie.archive.automatic=true
     hoodie.keep.max.commits=30 // 默认值为30，根据业务场景指定
     hoodie.keep.min.commits=20 // 默认值为20，根据业务场景指定
     ```

   - SparkDataSource（option里设置如下参数，写数据时触发）

     hoodie.archive.automatic=true

     hoodie.keep.max.commits=30 // 默认值为30，根据业务场景指定

     hoodie.keep.min.commits=20 // 默认值为20，根据业务场景指定

   - Flink（with属性里设置如下参数，写数据时触发）

     hoodie.archive.automatic=true

     archive.max_commits=30 // 默认值为30，根据业务场景指定

     archive.min_commits=20 // 默认值为20，根据业务场景指定

2. 手动触发1次archive
   - Spark SQL（设置如下参数，手动触发1次）

     ```
     hoodie.archive.automatic=true
     hoodie.keep.max.commits=30 // 默认值为30，根据业务场景指定
     hoodie.keep.min.commits=20 // 默认值为20，根据业务场景指定
     ```

随后执行SQL，当执行过clean，Timeline中存在数据文件已经被清理的Instant，且总Instant数量超过30时，会触发archive。

```
run archivelog on ${table_name}
```

7.4 Hudi Clustering 操作说明

什么是 Clustering

即数据布局，该服务可重新组织数据以提高查询性能，也不会影响摄取速度。

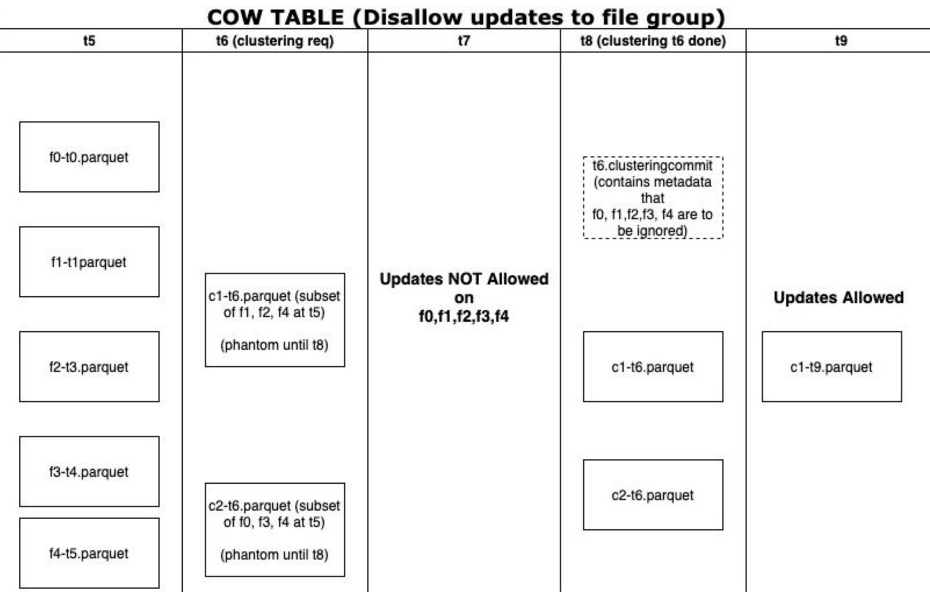
Clustering 架构

Hudi通过其写入客户端API提供了不同的操作，如insert/upsert/bulk_insert来将数据写入Hudi表。为了能够在文件大小和入湖速度之间进行权衡，Hudi提供了一个hoodie.parquet.small.file.limit配置来设置最小文件大小。用户可以将该配置设置为“0”，以强制新数据写入新的文件组，或设置为更高的值以确保新数据被“填充”到现有小的文件组中，直到达到指定大小为止，但其会增加摄取延迟。

为能够支持快速摄取的同时不影响查询性能，引入了Clustering服务来重写数据以优化Hudi数据湖文件的布局。

Clustering服务可以异步或同步运行，Clustering会添加了一种新的REPLACE操作类型，该操作类型将在Hudi元数据时间轴中标记Clustering操作。

Clustering服务基于Hudi的MVCC设计，允许继续插入新数据，而Clustering操作在后台运行以重新格式化数据布局，从而确保并发读写者之间的快照隔离。



总体而言Clustering分为两个部分：

- 调度Clustering：使用可插拔的Clustering策略创建Clustering计划。
  a. 识别符合Clustering条件的文件：根据所选的Clustering策略，调度逻辑将识别符合Clustering条件的文件。
  b. 根据特定条件对符合Clustering条件的文件进行分组。每个组的数据大小应为targetFileSize的倍数。分组是计划中定义的"策略"的一部分。此外还有一个选项可以限制组大小，以改善并行性并避免混排大量数据。

c. 将Clustering计划以avro元数据格式保存到时间线。

- 执行Clustering：使用执行策略处理计划以创建新文件并替换旧文件。
  a. 读取Clustering计划，并获得ClusteringGroups，其标记了需要进行Clustering的文件组。
  b. 对于每个组使用strategyParams实例化适当的策略类（例如：sortColumns），然后应用该策略重写数据。
  c. 创建一个REPLACE提交，并更新HoodieReplaceCommitMetadata中的元数据。

如何执行 Clustering

1. Spark SQL（设置如下参数，写数据时触发）

```
hoodie.clustering.inline=true  // 默认值 false，即默认为关闭状态
hoodie.clustering.inline.max.commits=4 // 默认值为4，根据业务场景指定
hoodie.clustering.plan.strategy.max.bytes.per.group=2147483648 // 默认值为2G，根据业务场景指定。一般不需要指定，因为正常每个file group下的数据量不会超过2G
hoodie.clustering.plan.strategy.max.num.groups=30 // 默认值为30，根据业务场景指定。一般通过调整这个参数来调整每次Clustering计划合并的数据量（max.bytes.per.group * max.num.groups）。
hoodie.clustering.plan.strategy.partition.regex.pattern=${正则表达式} // 无默认值，不指定该参数的时候Clustering会对所有分区下的数据进行重组。
hoodie.clustering.plan.strategy.small.file.limit=314572800 // 默认值是300M，根据业务场景指定。每个分区下，小于300M的文件会被筛选出来做Clustering。
hoodie.clustering.plan.strategy.sort.columns=${排序列1,......,排序列n} // 无默认值，根据业务场景指定。指定为查询业务经常使用且不包含null的列。
hoodie.clustering.plan.strategy.target.file.max.bytes=1073741824 // 默认值为1G，根据业务场景指定。用来设置Clustering产生的文件的最大size。
```

2. SparkDataSource（option里设置如下参数，写数据时触发）

   option("hoodie.clustering.inline", "true").

   option("hoodie.clustering.inline.max.commits", 4).

   option("hoodie.clustering.plan.strategy.max.bytes.per.group", 2147483648).

   option("hoodie.clustering.plan.strategy.max.num.groups", 30).

   option("hoodie.clustering.plan.strategy.partition.regex.pattern", "${正则表达式}").

   option("hoodie.clustering.plan.strategy.small.file.limit", 314572800).

   option("hoodie.clustering.plan.strategy.sort.columns", "${排序列1,......,排序列n}").

   option("hoodie.clustering.plan.strategy.target.file.max.bytes", 1073741824).

3. 手动触发1次Clustering
   - Spark SQL（设置如下参数，手动触发1次）

```
hoodie.clustering.inline=true
hoodie.clustering.inline.max.commits=4 // 默认值为4，根据业务场景指定
hoodie.clustering.plan.strategy.max.bytes.per.group=2147483648 // 默认值为2G，根据业务场景指定。一般不需要指定，因为正常每个file group下的数据量不会超过2G
hoodie.clustering.plan.strategy.max.num.groups=30 // 默认值为30，根据业务场景指定。一般通过调整这个参数来调整每次Clustering计划合并的数据量（max.bytes.per.group * max.num.groups）。
hoodie.clustering.plan.strategy.partition.regex.pattern=${正则表达式} // 无默认值，不指定该参数的时候Clustering会对所有分区下的数据进行重组。
hoodie.clustering.plan.strategy.small.file.limit=314572800 // 默认值是300M，根据业务场景指定。每个分区下，小于300M的文件会被筛选出来做Clustering。
hoodie.clustering.plan.strategy.sort.columns=${排序列1,......,排序列n} // 无默认值，根据业务场景指定。指定为查询业务经常使用且不包含null的列。
hoodie.clustering.plan.strategy.target.file.max.bytes=1073741824 // 默认值为1G，根据业务场景指定。用来设置Clustering产生的文件的最大size。
```

随后执行SQL

```
call run_clustering(table => '${表名}')
```

注意

1. Clustering的排序列不允许值存在null，这是Spark RDD的限制。
2. 当target.file.max.bytes的值较大时，启动Clustering执行需要提高--executor-memory，否则会导致executor内存溢出。
3. Clean不支持清理Clustering失败后的残留文件。
4. Clustering后产生的新文件大小不等，这可能引起数据倾斜。
5. Clustering不支持和Upsert（写操作更新待Clustering的文件）并发，如果Clustering处于inflight状态，该FileGroup下的文件不支持被更新。
6. 如果存在未完成的Clustering计划，后续写入触发生成Compaction调度计划时会报错失败，需要及时执行Clustering计划。

 版权所有 © 华为技术有限公司

8 Hudi 常见配置参数

本章节介绍Hudi重要配置的详细信息，更多配置请参考hudi官网：https://hudi.apache.org/cn/docs/0.11.0/configurations/。

- 提交DLI Spark SQL作业时，在SQL编辑器界面右上角的”设置”->”参数设置”中可以配置Hudi参数。
- 提交DLI Spark jar作业时，Hudi参数可以通过Spark datasource API的option来配置。

  或者，在提交作业时配置到"Spark参数(--conf)" 中，注意，此处配置的参数，键需要添加前缀 ”spark.hadoop.”，例如”spark.hadoop.hoodie.compact.inline=true”

写入操作配置

表 8-1 写入操作重要配置项

参数	描述	默认值
hoodie.datasource.write.table.name	指定写入的hudi表名。	无

 版权所有 © 华为技术有限公司

参数	描述	默认值
hoodie.datasource.write.operation	写hudi表指定的操作类型，当前支持upsert、delete、insert、bulk_insert等方式。 ● upsert：更新插入混合操作 ● delete：删除操作 ● insert：插入操作 ● bulk_insert： 用于初始建表导入数据， 注意初始建表禁止使用upsert、insert方式 ● insert_overwrite：对静态分区执行insert overwrite ● insert_overwrite_table：动态分区执行insert overwrite，该操作并不会立刻删除全表做overwrite，会逻辑上重写hudi表的元数据，无用数据后续由hudi的clean机制清理。效率比bulk_insert + overwrite 高	upsert
hoodie.datasource.write.table.type	指定hudi表类型，一旦这个表类型被指定，后续禁止修改该参数，可选值MERGE_ON_READ。	COPY_ON_WRITE
hoodie.datasource.write.precombine.field	该值用于在写之前对具有相同的key的行进行合并去重。	指定为具体的表字段
hoodie.datasource.write.payload.class	在更新过程中，该类用于提供方法将要更新的记录和更新的记录做合并，该实现可插拔，如要实现自己的合并逻辑，可自行编写。	org.apache.hudi.common.model.DefaultHoodieRecordPayload
hoodie.datasource.write.recordkey.field	用于指定hudi的主键，hudi表要求有唯一主键。	指定为具体的表字段
hoodie.datasource.write.partitionpath.field	用于指定分区键，该值配合hoodie.datasource.write.keygenerator.class使用可以满足不同的分区场景。	无
hoodie.datasource.write.hive_style_partitioning	用于指定分区方式是否和hive保持一致，建议该值设置为true。	true

参数	描述	默认值
hoodie.datasource.write.keygenerator.class	配合hoodie.datasource.write.partitionpath.field，hoodie.datasource.write.recordkey.field产生主键和分区方式。 **说明** 写入设置KeyGenerator与表保存的参数值不一致时将提示需要保持一致。	org.apache.hudi.keygen.ComplexKeyGenerator

同步 Hive 表配置

由DLI提供的元数据服务是一种Hive Metastore服务（HMS），因此下列参数与同步元数据服务相关。

表 8-2 同步 Hive 表参数配置

参数	描述	默认值
hoodie.datasource.hive_sync.enable	是否同步hudi表信息到Hive。当使用DLI提供的元数据服务时，配置该参数代表同步至DLI的元数据中。 **注意** 建议该值设置为true，统一使用元数据服务管理hudi表。	false
hoodie.datasource.hive_sync.database	要同步给hive的数据库名。	default
hoodie.datasource.hive_sync.table	要同步给hive的表名，建议这个值和hoodie.datasource.write.table.name保证一致。	unknown
hoodie.datasource.hive_sync.partition_fields	用于决定hive分区列。	""
hoodie.datasource.hive_sync.partition_extractor_class	用于提取hudi分区列值，将其转换成hive分区列。	org.apache.hudi.hive.SlashEncodedDayPartitionValueExtractor
hoodie.datasource.hive_sync.support_timestamp	当hudi表存在timestamp类型字段时，需指定此参数为true，以实现同步timestamp类型到hive元数据中。该值默认为false，默认将timestamp类型同步为bigInt，默认情况可能导致使用sql查询包含timestamp类型字段的hudi表出现错误。	true

 版权所有 © 华为技术有限公司

参数	描述	默认值
hoodie.datasource.hive_sync.username	使用jdbc方式同步Hive时，指定的用户名。	hive
hoodie.datasource.hive_sync.password	使用jdbc方式同步Hive时，指定的密码。	hive
hoodie.datasource.hive_sync.jdbcurl	连接hive jdbc指定的连接。	""
hoodie.datasource.hive_sync.use_jdbc	是否使用Hive jdbc方式连接Hive同步Hudi表信息。建议该值设置为false，设置为false后jdbc连接相关配置无效。	true

index 相关配置

表 8-3 index 相关参数配置

参数	描述	默认值
hoodie.index.class	用户自定义索引的全路径名，索引类必须为HoodieIndex的子类，当指定该配置时，其会优先于hoodie.index.type配置。	""
hoodie.index.type	使用的索引类型，默认为布隆过滤器。可能的选项是[BLOOM | GLOBAL_BLOOM | SIMPLE | GLOBAL_SIMPLE] 。 布隆过滤器消除了对外部系统的依赖，并存储在Parquet数据文件的页脚中。	BLOOM
hoodie.index.bloom.num_entries	存储在布隆过滤器中的条目数。假设maxParquetFileSize为128MB，averageRecordSize为1024B，因此，一个文件中的记录总数约为130K。 默认值（60000）大约是此近似值的一半。 **注意** 将此值设置得太低，将产生很多误报，并且索引查找将必须扫描比其所需的更多的文件；如果将其设置得非常高，将线性增加每个数据文件的大小（每50000个条目大约4KB）。	60000

版权所有 © 华为技术有限公司

参数	描述	默认值
hoodie.index.bloom.fpp	根据条目数允许的错误率。 用于计算应为布隆过滤器分配多少位以及哈希函数的数量。通常将此值设置得很低（默认值：0.000000001），在磁盘空间上进行权衡以降低误报率。	0.000000001
hoodie.bloom.index.parallelism	索引查找的并行度，其中涉及Spark Shuffle。 默认情况下，根据输入的工作负载特征自动计算的。	0
hoodie.bloom.index.prune.by.ranges	为true时，从文件框定信息，可以加快索引查找的速度。 如果键具有单调递增的前缀，例如时间戳，则特别有用。	true
hoodie.bloom.index.use.caching	为true时，将通过减少用于计算并行度或受影响分区的IO来缓存输入的RDD以加快索引查找。	true
hoodie.bloom.index.use.treebased.filter	为true时，启用基于间隔树的文件过滤优化。与暴力模式相比，此模式可根据键范围加快文件过滤速度。	true
hoodie.bloom.index.bucketized.checking	为true时，启用了桶式布隆过滤。这减少了在基于排序的布隆索引查找中看到的偏差。	true
hoodie.bloom.index.keys.per.bucket	仅在启用bloomIndexBucketizedChecking并且索引类型为bloom的情况下适用。 此配置控制“存储桶”的大小，该大小可跟踪对单个文件进行的记录键检查的次数，并且是分配给执行布隆过滤器查找的每个分区的工作单位。 较高的值将分摊将布隆过滤器读取到内存的固定成本。	10000000
hoodie.bloom.index.update.partition.path	仅在索引类型为GLOBAL_BLOOM时适用。 为true时，当对一个已有记录执行包含分区路径的更新操作时，将会导致把新记录插入到新分区，而把原有记录从旧分区里删除。为false时，只对旧分区的原有记录进行更新。	true

存储配置

表 8-4 存储参数配置

参数	描述	默认值
hoodie.parquet.max.file.size	Hudi写阶段生成的parquet文件的目标大小。对于DFS，这需要与基础文件系统块大小保持一致，以实现最佳性能。	120 * 1024 * 1024 byte
hoodie.parquet.block.size	parquet页面大小，页面是parquet文件中的读取单位，在一个块内，页面被分别压缩。	120 * 1024 * 1024 byte
hoodie.parquet.compression.ratio	当Hudi尝试调整新parquet文件的大小时，预期对parquet数据进行压缩的比例。 如果bulk_insert生成的文件小于预期大小，请增加此值。	0.1
hoodie.parquet.compression.codec	parquet压缩编解码方式名称，默认值为gzip。可能的选项是[gzip | snappy | uncompressed | lzo]	snappy
hoodie.logfile.max.size	LogFile的最大值。这是在将日志文件移到下一个版本之前允许的最大值。	1GB
hoodie.logfile.data.block.max.size	LogFile数据块的最大值。这是允许将单个数据块附加到日志文件的最大值。 这有助于确保附加到日志文件的数据被分解为可调整大小的块，以防止发生OOM错误。此大小应大于JVM内存。	256MB
hoodie.logfile.to.parquet.compression.ratio	随着记录从日志文件移动到parquet，预期会进行额外压缩的比例。 用于merge_on_read存储，以将插入内容发送到日志文件中并控制压缩parquet文件的大小。	0.35

compaction&cleaning 配置

表 8-5 compaction&cleaning 参数配置

参数	描述	默认值
hoodie.clean.automatic	是否执行自动clean。	true
hoodie.cleaner.policy	要使用的清理政策。Hudi将删除旧版本的parquet文件以回收空间。 任何引用此版本文件的查询和计算都将失败。建议确保数据保留的时间超过最大查询执行时间。	KEEP_LATEST_COMMITS
hoodie.cleaner.commits.retained	保留的提交数。因此，数据将保留为num_of_commits * time_between_commits（计划的），这也直接转化为逐步提取此数据集的数量。	10
hoodie.keep.max.commits	触发归档操作的commit数阈值。	30
hoodie.keep.min.commits	归档操作保留的commit数。	20
hoodie.commits.archival.batch	这控制着批量读取并一起归档的提交即时的数量。	10
hoodie.parquet.small.file.limit	该值应小于maxFileSize，如果将其设置为0，会关闭此功能。由于批处理中分区中插入记录的数量众多，总会出现小文件。Hudi提供了一个选项，可以通过将对该分区中的插入作为对现有小文件的更新来解决小文件的问题。此处的大小是被视为“小文件大小”的最小文件大小。	104857600 byte
hoodie.copyonwrite.insert.split.size	插入写入并行度。为单个分区的总共插入次数。写出100MB的文件，至少1KB大小的记录，意味着每个文件有100K记录。默认值是超额配置为500K。 为了改善插入延迟，请对其进行调整以匹配单个文件中的记录数。将此值设置为较小的值将导致文件变小（尤其是当compactionSmallFileSize为0时）。	500000

参数	描述	默认值
hoodie.copyonwrite.insert.auto.split	Hudi是否应该基于最后24个提交的元数据动态计算insertSplitSize，默认关闭。	true
hoodie.copyonwrite.record.size.estimate	平均记录大小。如果指定，Hudi将使用它，并且不会基于最后24个提交的元数据动态地计算。 没有默认值设置。这对于计算插入并行度以及将插入打包到小文件中至关重要。	1024
hoodie.compact.inline	当设置为true时，紧接在插入或插入更新或批量插入的提交或增量提交操作之后由摄取本身触发压缩。	true
hoodie.compact.inline.max.delta.commits	触发内联压缩之前要保留的最大增量提交数。	5
hoodie.compaction.lazy.block.read	当CompactedLogScanner合并所有日志文件时，此配置有助于选择是否应延迟读取日志块。选择true以使用I/O密集型延迟块读取（低内存使用），或者为false来使用内存密集型立即块读取（高内存使用）。	true
hoodie.compaction.reverse.log.read	HoodieLogFormatReader会从pos=0到pos=file_length向前读取日志文件。如果此配置设置为true，则Reader会从pos=file_length到pos=0反向读取日志文件。	false
hoodie.cleaner.parallelism	如果清理变慢，请增加此值。	200
hoodie.compaction.strategy	用来决定在每次压缩运行期间选择要压缩的文件组的压缩策略。默认情况下，Hudi选择具有累积最多未合并数据的日志文件。	org.apache.hudi.table.action.compact.strategy. LogFileSizeBasedCompactionStrategy
hoodie.compaction.target.io	LogFileSizeBasedCompactionStrategy的压缩运行期间要花费的MB量。当压缩以内联模式运行时，此值有助于限制摄取延迟。	500 * 1024 MB
hoodie.compaction.daybased.target.partitions	由org.apache.hudi.io.compact.strategy.DayBasedCompactionStrategy使用，表示在压缩运行期间要压缩的最新分区数。	10

参数	描述	默认值
hoodie.compaction.payload.class	这需要与插入/插入更新过程中使用的类相同。就像写入一样，压缩也使用记录有效负载类将日志中的记录彼此合并，再次与基本文件合并，并生成压缩后要写入的最终记录。	org.apache.hudi.common.model.Defaulthoodierecordpayload
hoodie.schedule.compact.only.inline	在写入操作时，是否只生成压缩计划。在hoodie.compact.inline=true时有效。	false
hoodie.run.compact.only.inline	通过Sql执行run compaction命令时，是否只执行压缩操作，压缩计划不存在时直接退出。	false

单表并发控制配置

表 8-6 单表并发控制参数配置

参数	描述	默认值
hoodie.write.lock.provider	指定lock provider，在元数据由DLI托管的场景，推荐配置com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider	Spark SQL作业和Flink SQL作业会根据元数据服务切换对应的实现类，由DLI托管元数据的场景为com.huawei.luxor.hudi.util.DliCatalogBasedLockProvider
hoodie.write.lock.hivemetastore.database	HMS服务中的database	无
hoodie.write.lock.hivemetastore.table	HMS服务中的table name	无
hoodie.write.lock.client.num_retries	重试次数	10
hoodie.write.lock.client.wait_time_ms_between_retry	重试间隔	10000
hoodie.write.lock.conflict.resolution.strategy	lock provider类，必须是ConflictResolutionStrategy的子类	org.apache.hudi.client.transaction.SimpleConcurrentFileWritesConflictResolutionStrategy

Clustering 配置

说明

Clustering中有两个策略分别是hoodie.clustering.plan.strategy.class和hoodie.clustering.execution.strategy.class。一般情况下指定plan.strategy为SparkRecentDaysClusteringPlanStrategy或者SparkSizeBasedClusteringPlanStrategy时，execution.strategy不需要指定。但当plan.strategy为SparkSingleFileSortPlanStrategy时，需要指定execution.strategy为SparkSingleFileSortExecutionStrategy。

表 8-7 Clustering 参数配置

参数	描述	默认值
hoodie.clustering.inline	是否同步执行clustering	false
hoodie.clustering.inline.max.commits	触发clustering的commit数	4
hoodie.clustering.async.enabled	是否启用异步执行clustering	false
hoodie.clustering.async.max.commits	异步执行时触发clustering的commit数	4
hoodie.clustering.plan.strategy.target.file.max.bytes	指定clustering后每个文件大小最大值	1024 * 1024 * 1024 byte
hoodie.clustering.plan.strategy.small.file.limit	小于该大小的文件会被clustering	300 * 1024 * 1024 byte
hoodie.clustering.plan.strategy.sort.columns	clustering用以排序的列	无
hoodie.layout.optimize.strategy	Clustering执行策略，可选linear、z-order、hilbert 三种排序方式	linear
hoodie.layout.optimize.enable	使用z-order、hilbert时需要开启	false
hoodie.clustering.plan.strategy.class	筛选FileGroup进行clustering的策略类，默认筛选小于hoodie.clustering.plan.strategy.small.file.limit阈值的文件	org.apache.hudi.client.clustering.plan.strategy.SparkSizeBasedClusteringPlanStrategy
hoodie.clustering.execution.strategy.class	执行clustering的策略类（RunClusteringStrategy的子类），用以定义群集计划的执行方式。 默认类们按指定的列对计划中的文件组进行排序，同时满足配置的目标文件大小	org.apache.hudi.client.clustering.run.strategy.SparkSortAndSizeExecutionStrategy

参数	描述	默认值
hoodie.clustering.plan.strategy.max.num.groups	设置执行clustering时最多选择多少个FileGroup，该值越大并发度越大	30
hoodie.clustering.plan.strategy.max.bytes.per.group	设置执行clustering时每个FileGroup最多有多少数据参与clustering	2 * 1024 * 1024 * 1024 byte

版权所有 © 华为技术有限公司