Flink SQL CDC 上线！我们总结了 13 条生产实践经验

摘要：7月，Flink 1.11 新版发布，在生态及易用性上有大幅提升，其中 Table & SQL 开始支持 Change Data Capture（CDC）。CDC 被广泛使用在复制数据、更新缓存、微服务间同步数据、审计日志等场景，本文由社区由曾庆东同学分享，主要介绍 Flink SQL CDC 在生产环境的落地实践以及总结的实战经验，文章分为以下几部分：

1. 项目背景
2. 解决方案
3. 项目运行环境与现状
4. 具体实现
5. 踩过的坑和学到的经验
6. 总结

Tips：点击下方链接可查看社区直播的 Flink SQL CDC 相关视频～
https://flink-learning.org.cn/developers/flink-training-course3/

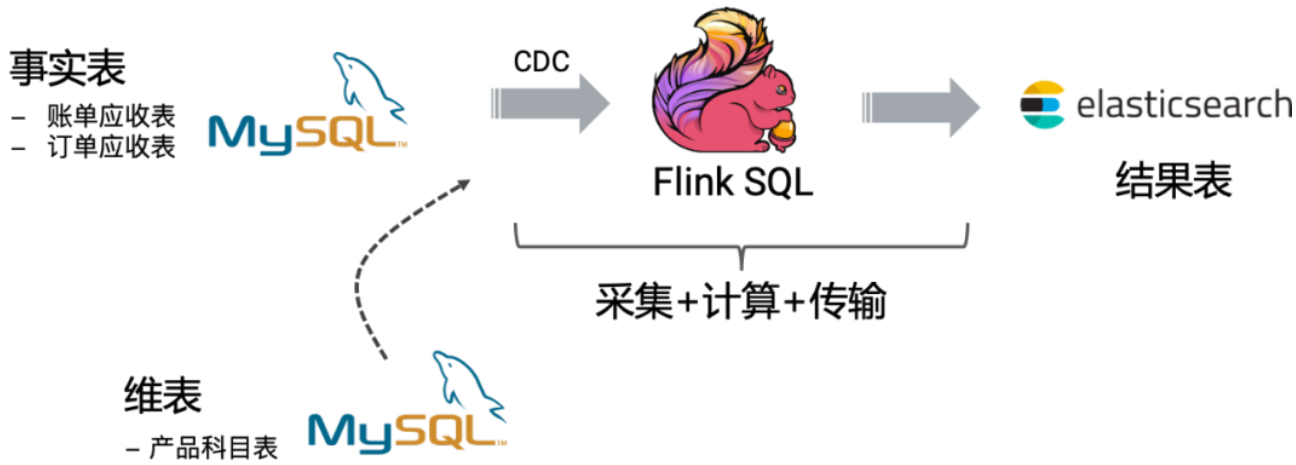
01 项目背景

本人目前参与的项目属于公司里面数据密集、计算密集的一个重要项目，需要提供高效且准确的 OLAP 服务，提供灵活且实时的报表。业务数据存储在 MySQL 中，通过主从复制同步到报表库。作为集团级公司，数据增长多而且快，出现了多个千万级、亿级的大表。为了实现各个维度的各种复杂的报表业务，有些千万级大表仍然需要进行 Join，计算规模非常惊人，经常不能及时响应请求。

随着数据量的日益增长和实时分析的需求越来越大，急需对系统进行流式计算、实时化改造。正是在这个背景下，开始了我们与 Flink SQL CDC 的故事。

02 解决方案

针对平台现在存在的问题，我们提出了把报表的数据实时化的方案。该方案主要通过 Flink SQL CDC + Elasticsearch 实现。Flink SQL 支持 CDC 模式的数据同步，将 MySQL 中的全增量数据实时地采集、预计算、并同步到 Elasticsearch 中，Elasticsearch 作为我们的实时报表和即席分析引擎。项目整体架构图如下所示：



实时报表实现具体思路是，使用 Flink CDC 读取全量数据，全量数据同步完成后，Flink CDC 会无缝切换至 MySQL 的 binlog 位点继续消费增量的变更数据，且保证不会多消费一条也不会少消费一条。读取到的账单和订单的全增量数据会与产品表做关联补全信息，并做一些预聚合，然后将聚合结果输出到 Elasticsearch，前端页面只需要到 Elasticsearch 通过精准匹配（terms）查找数据，或者再使用 agg 做高维聚合统计得到多个服务中心的报表数据。

从整体架构中，可以看到，Flink SQL 及其 CDC 功能在我们的架构中扮演着核心角色。我们采用 Flink SQL CDC，而不是 Canal + Kafka 的传统架构，主要原因还是因为其依赖组件少，维护成本低，开箱即用，上手容易。具体来说 Flink SQL CDC 是一个集采集、计算、传输于一体的工具，其吸引我们的优点有：

① 减少维护的组件、简化实现链路；
② 减少端到端延迟；
③ 减轻维护成本和开发成本；
④ 支持 Exactly Once 的读取和计算（由于我们是账务系统，所以数据一致性非常重要）；
⑤ 数据不落地，减少存储成本；
⑥ 支持全量和增量流式读取；

有关 Flink SQL CDC 的介绍和教程，可以观看 Apache Flink 社区发布的相关视频：

https://www.bilibili.com/video/BV1zt4y1D7kt/

项目使用的是 flink-cdc-connectors 中提供的 mysql-cdc 组件。这是一个 Flink 数据源，支持对 MySQL 数据库的全量和增量读取。它在扫描全表前会先加一个全局读锁，然后获取此时的 binlog position，紧接着释放全局读锁。随后开始扫描全表，当全表快照读取完后，会从之前获取的 binlog position 获取增量的变更记录。因此这个读锁是非常轻量的，持锁时间非常短，不会对线上业务造成太大影响。更多信息可以参考 flink-cdc-connectors 项目官网：https://github.com/ververica/flink-cdc-connectors。

03 项目运行环境与现状

我们在生产环境搭建了 Hadoop + Flink + Elasticsearch 分布式环境，采用的 Flink on YARN 的 per-job 模式运行，使用 RocksDB 作为 state backend，HDFS 作为 checkpoint 持久化地址，并且做好了 HDFS 的容错，保证 checkpoint 数据不丢失。我们使用 SQL Client 提交作业，所有作业统一使用纯 SQL，没有写一行 Java 代码。

目前已上线了 3 个基于 Flink CDC 的作业，已稳定在线上运行了两个星期，并且业务产生的订单实收和账单实收数据能实时聚合输出到 Elasticsearch，输出的数据准确无误。现在也正在对其他报表采用 Flink SQL CDC 进行实时化改造，替换旧的业务系统，让系统数据更实时。

04 具体实现

① 进入 Flink/bin，使用 ./sql-client.sh embedded 启动 SQL CLI 客户端。

② 使用 DDL 创建 Flink Source 和 Sink 表。这里创建的表字段个数不一定要与 MySQL 的字段个数和顺序一致，只需要挑选 MySQL 表中业务需要的字段即可，并且字段类型保持一致。

```
-- 在Flink创建账单实收source表
CREATE TABLE bill_info (
  billCode STRING,
  serviceCode STRING,
  accountPeriod STRING,
  subjectName STRING ,
  subjectCode STRING,
  occurDate TIMESTAMP,
  amt  DECIMAL(11,2),
  status STRING,
  proc_time AS PROCTIME() --使用维表时需要指定该字段
) WITH (
  'connector' = 'mysql-cdc', -- 连接器
```

```
  'hostname' = '******',   --mysql地址
  'port' = '3307',  -- mysql端口
  'username' = '******',  --mysql用户名
  'password' = '******',  -- mysql密码
  'database-name' = 'cdc', --  数据库名称
  'table-name' = '***'
);

-- 在Flink创建订单实收source表
CREATE TABLE order_info (
  orderCode STRING,
  serviceCode STRING,
  accountPeriod STRING,
  subjectName STRING ,
  subjectCode STRING,
  occurDate TIMESTAMP,
  amt  DECIMAL(11, 2),
  status STRING,
  proc_time AS PROCTIME()  --使用维表时需要指定该字段
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = '******',
  'port' = '3307',
  'username' = '******',
  'password' = '******',
  'database-name' = 'cdc',
  'table-name' = '***',
);

-- 创建科目维表
CREATE TABLE subject_info (
  code VARCHAR(32) NOT NULL,
  name VARCHAR(64) NOT NULL,
  PRIMARY KEY (code) NOT ENFORCED  --指定主键
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://xxxx:xxxx/spd?useSSL=false&autoReconnect=true',
  'driver' = 'com.mysql.cj.jdbc.Driver',
  'table-name' = '***',
  'username' = '******',
  'password' = '******',
  'lookup.cache.max-rows' = '3000',
  'lookup.cache.ttl' = '10s',
  'lookup.max-retries' = '3'
);

-- 创建实收分布结果表，把结果写到 Elasticsearch
CREATE TABLE income_distribution (
  serviceCode STRING,
  accountPeriod STRING,
  subjectCode STRING,
  subjectName STRING,
  amt  DECIMAL(13,2),
  PRIMARY KEY (serviceCode, accountPeriod, subjectCode) NOT ENFORCED
) WITH (
  'connector' = 'elasticsearch-7',
```

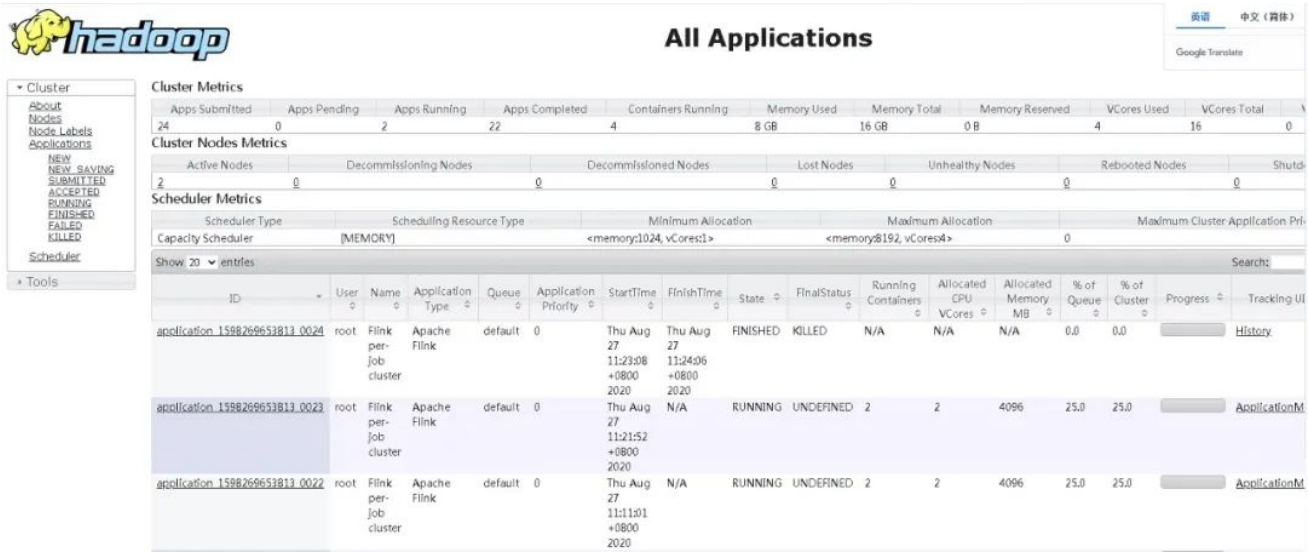
```
    'hosts' = 'http://xxxx:9200',
    'index' = 'income_distribution',
    'sink.bulk-flush.backoff.strategy' = 'EXPONENTIAL'
  );
```

以上的建表 DDL 分别创建了订单实收 source 表、账单实收 source 表、产品科目维表和 Elasticsearch 结果表。建表完成后，Flink 是不会马上去同步 MySQL 的数据，而是等到用户提交了一个 insert 作业后才会执行同步数据，并且 Flink 不会存储数据。我们的第一个作业是计算收入分布，数据来源于 bill_info 和 order_info 两张 MySQL 表，并且账单实收表和订单实收表都需要关联维表数据获取应收科目的最新中文名称，按照服务中心、账期、科目代码和科目名称进行分组计算实收金额的 sum 值，实收分布具体 DML 如下：

```
INSERT INTO income_distribution
SELECT t1.serviceCode, t1.accountPeriod, t1.subjectCode, t1.subjectName, SU
FROM (
  SELECT b.serviceCode, b.accountPeriod, b.subjectCode, s.name AS subjectNa
  FROM bill_info AS b
  JOIN subject_info FOR SYSTEM_TIME AS OF b.proc_time s ON b.subjectCode =
  GROUP BY b.serviceCode, b.accountPeriod, b.subjectCode, s.name
UNION ALL
  SELECT b.serviceCode, b.accountPeriod, b.subjectCode, s.name AS subjectNa
  FROM order_info AS b
  JOIN subject_info FOR SYSTEM_TIME AS OF b.proc_time s ON b.subjectCode =
  GROUP BY b.serviceCode, b.accountPeriod, b.subjectCode, s.name
) AS t1
GROUP BY t1.serviceCode, t1.accountPeriod, t1.subjectCode, t1.subjectName;
```

Flink SQL 的维表 JOIN 和双流 JOIN 写法上不太一样，对于维表，还需要在 Flink source table 上添加一个 proctime 字段 proc_time AS PROCTIME()，关联的时候使用 FOR SYSTEM_TIME AS OF 的 SQL 语法查询时态表，意思是关联查询最新版本的维表数据。关于维表 JOIN 的使用可参阅：https://ci.apache.org/projects/flink/flink-docs-release-1.11/zh/dev/table/streaming/joins.html。

③ 在 SQL Client 执行以上作业后，YARN 会创建一个 Flink 集群运行作业，并且用户可以在 Hadoop 上查看到执行作业的所有信息，并且能进入 Flink 的 Web UI 页面查看 Flink 作业详情，以下是 Hadoop 所有作业情况。



④ 作业提交后，Flink SQL CDC 会扫描指定的 MySQL 表，在这期间 Flink 也会进行 checkpoint，所以需要按照上文所述的配置 checkpoint 的重试策略和重试次数。当数据被读取进 Flink 后，Flink

会流式地进行作业逻辑的计算，实时统计出聚合结果输出到 Elasticsearch（sink 端）。相当于我们使用 Flink 在 MySQL 的表上维护了一个实时的物化视图，并将这个实时物化视图的结果存在了 Elasticsearch 中。在 Elasticsearch 中使用 GET /income_distribution/_search{ "query": {"match_all": {}}} 命令查看输出的实收分布结果，如下图：

通过图中的结果可以看出聚合结果被实时的计算出来，并写到了 Elasticsearch 中了。

05 踩过的坑和学到的经验

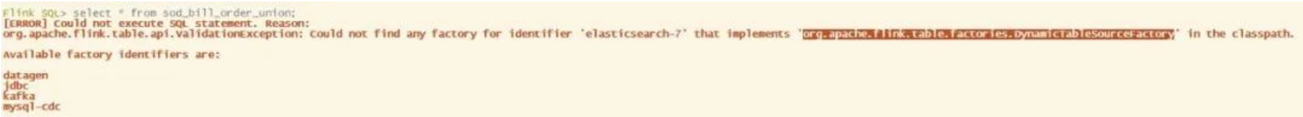
1. Flink 作业原来运行在 standalone session 模式下，提交多个 Flink 作业会导致作业失败报错。

- **原因：**因为 standalone session 模式下启动多个作业会导致多个作业的 Task 共享一个 JVM，可能会导致一些不稳定的问题。并且排查问题时，多个作业的日志混在一个 TaskManager 中，

增加了排查的难度。

- **解决方法：** 采用 YARN 的 per-job 模式启动多个作业，能有更好的隔离性。

2. SELECT elasticsearch table 报以下错误：

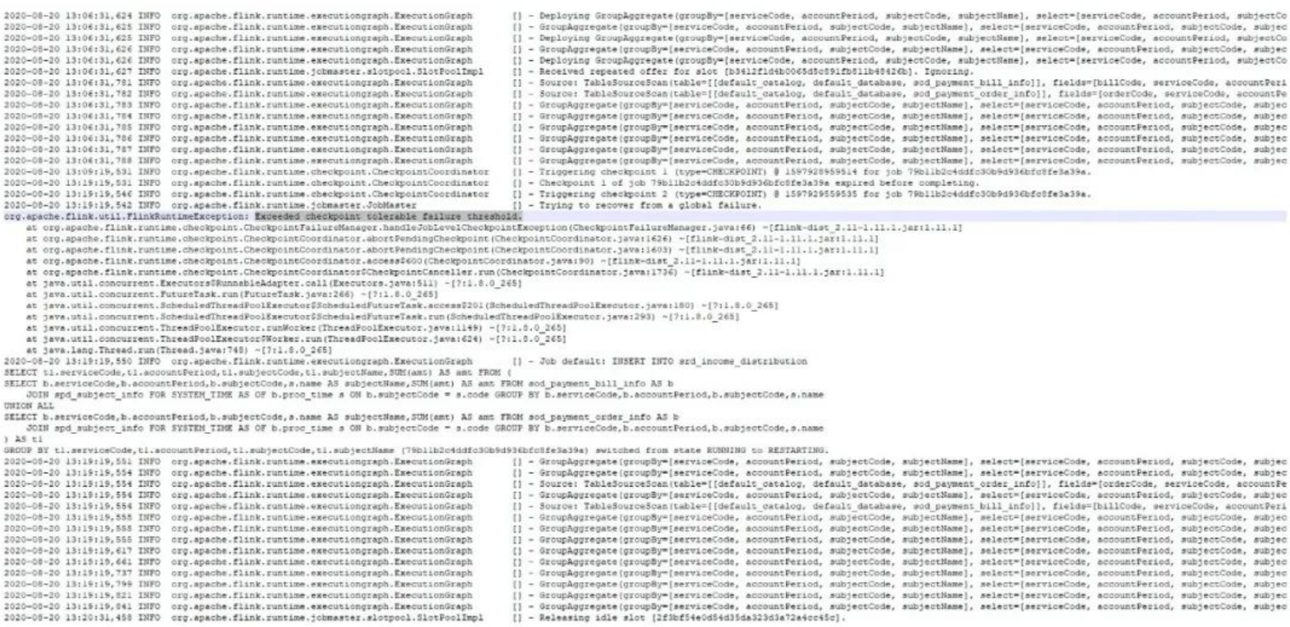


- **原因：** Elasticsearch connector 目前只支持了 sink，不支持 source 。所以不能 SELECT elasticsearch table。

3. 在 flink-conf.yaml 里修改默认并行度，但是在 Web UI 看到作业的并行度还是 1，并行度修改不生效。

- **解决办法：** 在使用 SQL Client 时 sql-client-defaults.yaml 中的并行度配置的优先级更高。在 sql-client-defaults.yaml 中修改并行度，或者删除 sql-client-defaults.yaml 中的并行度配置。更建议采用后者。

4. Flink 作业在扫描 MySQL 全量数据时，checkpoint 超时，出现作业 failover，如下图：



- **原因：** Flink CDC 在 scan 全表数据（我们的实收表有千万级数据）需要小时级的时间（受下游聚合反压影响），而在 scan 全表过程中是没有 offset 可以记录的（意味着没法做 checkpoint），但是 Flink 框架任何时候都会按照固定间隔时间做 checkpoint，所以此处 mysql-cdc source 做了比较取巧的方式，即在 scan 全表的过程中，会让执行中的 checkpoint

一直等待甚至超时。超时的 checkpoint 会被仍未认为是 failed checkpoint，默认配置下，这会触发 Flink 的 failover 机制，而默认的 failover 机制是不重启。所以会造成上面的现象。

- **解决办法**：在 flink-conf.yaml 配置 failed checkpoint 容忍次数，以及失败重启策略，如下：

```
execution.checkpointing.interval: 10min    # checkpoint间隔时间
execution.checkpointing.tolerable-failed-checkpoints: 100  # checkpoint 失败
restart-strategy: fixed-delay  # 重试策略
restart-strategy.fixed-delay.attempts: 2147483647    # 重试次数
```

目前 Flink 社区也有一个 issue（Flink-18578）来支持 source 主动拒绝 checkpoint 的机制，将来基于该机制，能比较优雅地解决这个问题。

5. Flink 怎么样开启 YARN 的 per-job 模式?

- **解决方法**：在 flink-conf.yaml 中配置 execution.target: yarn-per-job。

6. 进入 SQL Client 创建 table 后，在另外一个节点进入 SQL Client 查询不到 table。

- **原因**：因为 SQL Client 默认的 Catalog 是在 in-memory 的，不是持久化 Catalog，所以这属于正常现象，每次启动 Catalog 里面都是空的。

7. 作业在运行时 Elasticsearch 报如下错误：

> Caused by: org.apache.Flink.elasticsearch7.shaded.org.elasticsearch.ElasticsearchException: Elasticsearch exception [type=illegal_argument_exception, reason=mapper [amt] cannot be changed from type [long] to [float]]

- **原因**：数据库表的字段 amt 的类型是 decimal，DDL 创建输出到 es 的 amt 字段的类型也是 decimal，因为输出到 es 的第一条数据的amt如果是整数，比如是 10，输出到 es 的类型是 long 类型的，es client 会自动创建 es 的索引并且设置 amt 字段为 long 类型的格式，那么如果下一次输出到 es 的 amt 是非整数 10.1，那么输出到 es 的时候就会出现类型不匹配的错误。

- **解决方法**：手动生成 es 索引和 mapping 的信息，指定好 decimal 类型的数据格式是 saclefloat，但是在 DDL 处仍然可以保留该字段类型是 decimal。

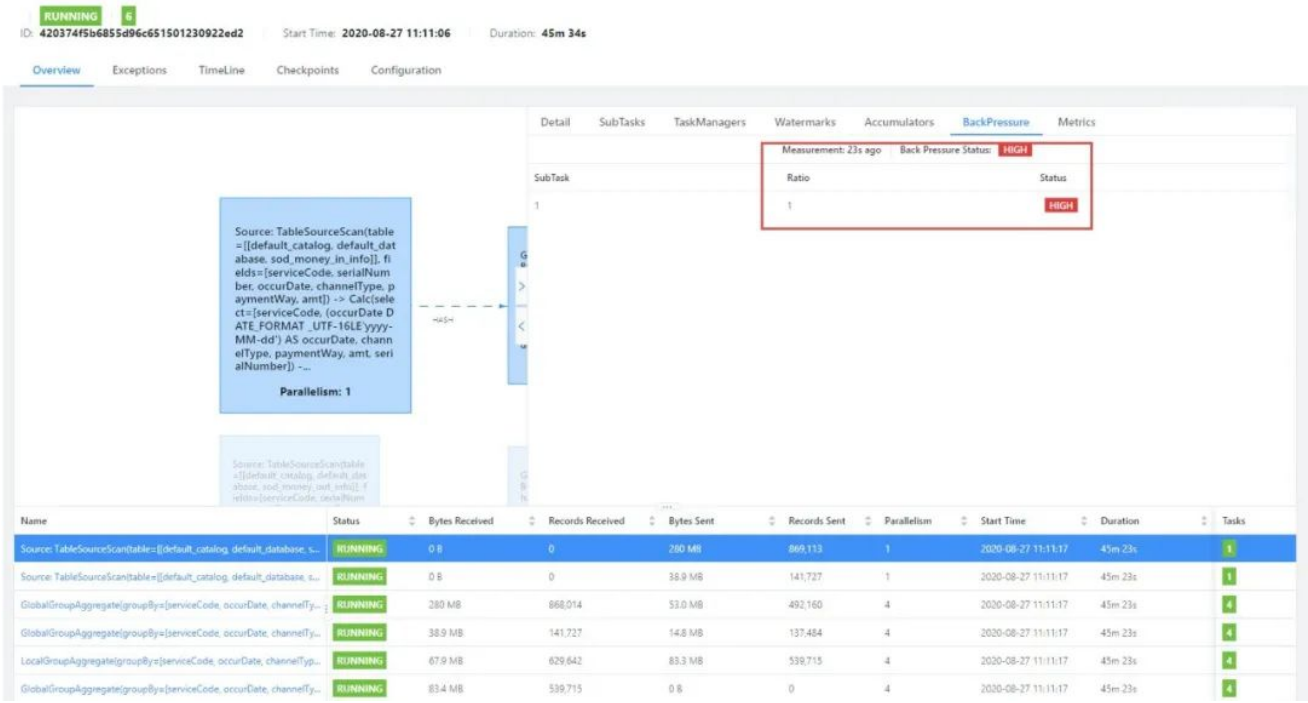
8. 作业在运行时 mysql cdc source 报如下错误：

```
at akka.dispatch.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107) [flink-dist_2.11-1.11.1.jar:1.11.1]
Caused by: org.apache.flink.util.SerializedThrowable: no viable alternative at input 'alter table std'
    at io.debezium.connector.mysql.AbstractReader.wrap(AbstractReader.java:230) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.connector.mysql.AbstractReader.failed(AbstractReader.java:207) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.connector.mysql.BinlogReader.handleEvent(BinlogReader.java:600) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at com.github.shyiko.mysql.binlog.BinaryLogClient.notifyEventListeners(BinaryLogClient.java:1130) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at com.github.shyiko.mysql.binlog.BinaryLogClient.listenForEventPackets(BinaryLogClient.java:978) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at com.github.shyiko.mysql.binlog.BinaryLogClient.connect(BinaryLogClient.java:581) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at com.github.shyiko.mysql.binlog.BinaryLogClient$7.run(BinaryLogClient.java:860) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at java.lang.Thread.run(Thread.java:748) ~[?:1.8.0_221]
Caused by: org.apache.flink.util.SerializedThrowable: no viable alternative at input 'alter table std'
    at io.debezium.antlr.ParsingErrorListener.syntaxError(ParsingErrorListener.java:40) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at org.antlr.v4.runtime.ProxyErrorListener.syntaxError(ProxyErrorListener.java:41) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at org.antlr.v4.runtime.Parser.notifyErrorListeners(Parser.java:544) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at org.antlr.v4.runtime.DefaultErrorStrategy.reportNoViableAlternative(DefaultErrorStrategy.java:310) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at org.antlr.v4.runtime.DefaultErrorStrategy.reportError(DefaultErrorStrategy.java:136) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.ddl.parser.mysql.generated.MySqlParser.sqlStatements(MySqlParser.java:1121) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.ddl.parser.mysql.generated.MySqlParser.root(MySqlParser.java:849) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.connector.mysql.antlr.MySqlAntlrDdlParser.parseTree(MySqlAntlrDdlParser.java:68) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
    at io.debezium.connector.mysql.antlr.MySqlAntlrDdlParser.parseTree(MySqlAntlrDdlParser.java:41) ~[flink-sql-connector-mysql-cdc-1.0.0.jar:1.0.0]
```

- **原因**：因为数据库中别的表做了字段修改，CDC source 同步到了 ALTER DDL 语句，但是解析失败抛出的异常。

- **解决方法**：在 flink-cdc-connectors 最新版本中已经修复该问题（跳过了无法解析的 DDL）。升级 connector jar 包到最新版本 1.1.0：flink-sql-connector-mysql-cdc-1.1.0.jar，替换 flink/lib 下的旧包。

9. 扫描全表阶段慢，在 Web UI 出现如下现象：



- **原因**：扫描全表阶段慢不一定是 cdc source 的问题，可能是下游节点处理太慢反压了。

- **解决方法**：通过 Web UI 的反压工具排查发现，瓶颈主要在聚合节点上。通过在 sql-client-

defaults.yaml 文件配上 MiniBatch 相关参数和开启 distinct 优化（我们的聚合中有 count distinct），作业的 scan 效率得到了很大的提升，从原先的 10 小时，提升到了 1 小时。关于性能调优的参数可以参阅：https://ci.apache.org/projects/flink/flink-docs-release-1.11/zh/dev/table/tuning/streaming_aggregation_optimization.html。

```
configuration:
  table.exec.mini-batch.enabled: true
  table.exec.mini-batch.allow-latency: 2s
  table.exec.mini-batch.size: 5000
  table.optimizer.distinct-agg.split.enabled: true
```

10. CDC source 扫描 MySQL 表期间，发现无法往该表 insert 数据。

- **原因**：由于使用的 MySQL 用户未授权 RELOAD 权限，导致无法获取全局读锁（FLUSH TABLES WITH READ LOCK），CDC source 就会退化成表级读锁，而使用表级读锁需要等到全表 scan 完，才能释放锁，所以会发现持锁时间过长的现象，影响其他业务写入数据。

- **解决方法**：给使用的 MySQL 用户授予 RELOAD 权限即可。所需的权限列表详见文档：https://github.com/ververica/flink-cdc-connectors/wiki/mysql-cdc-connector#setup-mysql-server。如果出于某些原因无法授予 RELOAD 权限，也可以显式配上 'debezium.snapshot.locking.mode' = 'none'来避免所有锁的获取，但要注意只有当快照期间表的 schema 不会变更才安全。

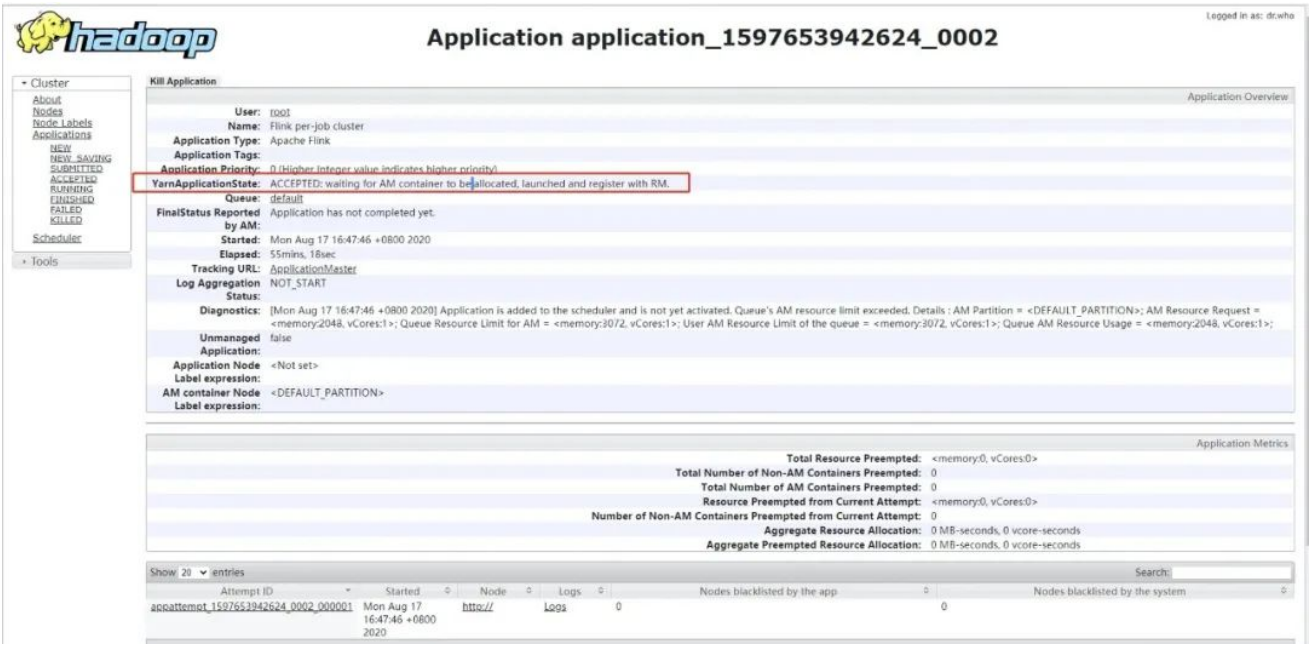
11. 多个作业共用同一张 source table 时，没有修改 server id 导致读取出来的数据有丢失。

- **原因**：MySQL binlog 数据同步的原理是，CDC source 会伪装成 MySQL 集群的一个 slave（使用指定的 server id 作为唯一 id），然后从 MySQL 拉取 binlog 数据。如果一个 MySQL 集群中有多个 slave 有同样的 id，就会导致拉取数据错乱的问题。

- **解决方法**：默认会随机生成一个 server id，容易有碰撞的风险。所以建议使用动态参数（table hint）在 query 中覆盖 server id。如下所示：

```
SELECT *
FROM bill_info /*+ OPTIONS('server-id'='123456') */ ;
```

12. 在启动作业时，YARN 接收了任务，但作业一直未启动：



- **原因：**Queue Resource Limit for AM 超过了限制资源限制。默认的最大内存是 30G (集群内存) * 0.1 = 3G，而每个 JM 申请 2G 内存，当提交第二个任务时，资源就不够了。

- **解决方法：**调大 AM 的 resource limit，在 capacity-scheduler.xml 配置 yarn.scheduler.capacity.maximum-am-resource-percent，代表AM的占总资源的百分比，默认为0.1，改成0.3（根据服务器的性能灵活配置）。

13. AM 进程起不来，一直被 kill 掉。

- **原因：**386.9 MB of 1 GB physical memory used; 2.1 GB of 2.1 GB virtual memory use。默认物理内存是 1GB，动态申请到了 1GB，其中使用了386.9 MB。物理内存 x 2.1=虚拟内存，1GBx2.1≈2.1GB ，2.1GB 虚拟内存已经耗尽，当虚拟内存不够时候，AM 的 container 就会自杀。

- **解决方法：**两个解决方案，或调整 yarn.nodemanager.vmem-pmem-ratio 值大点，或 yarn.nodemanager.vmem-check-enabled=false，关闭虚拟内存检查。参考：https://blog.csdn.net/lzxlfly/article/details/89175452。

06 总结

为了提升了实时报表服务的可用性和实时性，一开始我们采用了 Canal+Kafka+Flink 的方案，可是发现需要写比较多的 Java 代码，而且还需要处理好 DataStream 和 Table 的转换以及 binlong 位置的获取，开发难度相对较大。另外，需要维护 Kafka 和 Canal 这两个组件的稳定运行，对于我们小团队来说成本也不小。由于我们公司已经有基于 Flink 的任务在线上运行，因此采用 Flink SQL CDC 就成了顺理成章的事情。基于 Flink SQL CDC 的方案只需要编写 SQL ，不用写一行 Java 代码就能完成实时链路的打通和实时报表的计算，对于我们来说非常的简单易用，而且在线上运行的稳定性和性能表现也让我们满意。

我们正在公司内大力推广 Flink SQL CDC 的使用，也正在着手改造其他几个实时链路的任务。非常感谢开源社区能为我们提供如此强大的工具，也希望 Flink CDC 越来越强大，支持更多的数据库和功能。也再次感谢云邪老师对于我们项目上线的大力支持！

作者介绍：

曾庆东，金地物业中级开发工程师，负责聚合营业平台实时计算开发及运维工作，从事过大数据开发，目前专注于 Apache Flink 实时计算，喜欢开源技术，喜欢分享。