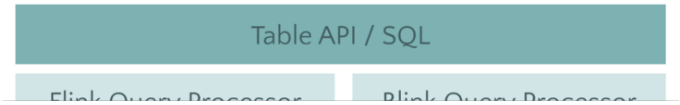
Flink Batch SQL 1.10 实践

Flink 作为流批统一的计算框架，在 1.10 中完成了大量 batch 相关的增强与改进。1.10 可以说是第一个成熟的生产可用的 Flink Batch SQL 版本，它一扫之前 Dataset 的羸弱，从功能和性能上都有大幅改进，以下我从架构、外部系统集成、实践三个方面进行阐述。

架构

Stack



首先来看下 stack，在新的 Blink planner 中，batch 也是架设在 Transformation 上的，这就意味着我们和 Dataset 完全没有关系了：

1. 我们可以尽可能的和 streaming 复用组件，复用代码，有同一套行为。
2. 如果想要 Table/SQL 的 toDataset 或者 fromDataset，那就完全没戏了。尽可能的在 Table 的层面来处理吧。
3. 后续我们正在考虑在 DataStream 上构建 BoundedStream，给 DataStream 带来批处理的功能。

网络模型

Batch 模式就是在中间结果落盘，这个模式和典型的 Batch 处理是一致的，比如 MapReduce/Spark/Tez。

Flink 以前的网络模型也分为 Batch 和 Pipeline 两种，但是 Batch 模式只是支持上下游隔断执行，也就是说资源用量可以不用同时满足上下游共同的并发。但是另外一个关键点是 Failover 没有对接好，1.9 和 1.10 在这方面进行了改进，支持了单点的 Failover。

建议在 Batch 时打开：

jobmanager.execution.failover-strategy = region

为了避免重启过于频繁导致 JobMaster 太忙了，可以把重启间隔提高：
restart-strategy.fixed-delay.delay = 30 s

Batch 模式的好处有：

- 容错好，可以单点恢复
- 调度好，不管多少资源都可以运行
- 性能差，中间数据需要落盘，强烈建议开启压缩：

  taskmanager.network.blocking-shuffle.compression.enabled = true

Batch 模式比较稳，适合传统 Batch 作业，大作业。

Pipeline 模式是 Flink 的传统模式，它完全和 Streaming 作业用的是同一套代码，其实社区里 Impala 和 Presto 也是类似的模式，纯走网络，需要处理反压，不落盘，它主要的优缺点是：

- 容错差，只能全局重来
- 调度差，你得保证有足够的资源
- 性能好，Pipeline 执行，完全复用 Stream，复用流控反压等功能。

有条件可以考虑开启 Pipeline 模式。

调度模型

Flink on Yarn 支持两种模式，Session 模式和 Per job 模式，现在已经在调度层次高度统一了。

1. Session 模式没有最大进程限制，当有 Job 需要资源时，它就会去 Yarn 申请新资源，当 Session 有空闲资源时，它就会给 Job 复用，所以它的模型和 PerJob 是基本一样的。
2. 唯一的不同只是：Session 模式可以跨作业复用进程。

另外，如果想要更好的复用进程，可以考虑加大 TaskManager 的超时释放：

resourcemanager.taskmanager-timeout = 900000

资源模型

先说说并发：

1. 对 Source 来说：目前 Hive 的 table 是根据 InputSplit 来定需要多少并发的，它之后能 Chain 起来的 Operators 自然都是和 source 相同的并发。
2. 对下游网络传输过后的 Operators(Tasks) 来说：除了一定需要单并发的 Task 来说，其它 Task 全部统一并发，由 table.exec.resource.default-parallelism 统一控制。

我们在 Blink 内部实现了基于统计信息来推断并发的功能，但是其实以上的策略在大部分场景就够用了。

Manage 内存

目前一个 TaskManager 里面含有多个 Slot，在 Batch 作业中，一个 Slot 里只能运行一个 Task (关闭 SlotShare)。

对内存来说，单个 TM 会把 Manage 内存切分成 Slot 粒度，如果 1 个 TM 中有 n 个 Slot，也就是 Task 能拿到 1/n 的 manage 内存。

我们在 1.10 做了重大的一个改进就是：Task 中 chain 起来的各个 operators 按照比例来瓜分内存，所以现在配置的算子内存都是一个比例值，实际拿到的还要根据 Slot 的内存来瓜分。

这样做的一个重要好处是：

1. 不管当前 Slot 有多少内存，作业能都 run 起来，这大大提高了开箱即用。

2. 不管当前 Slot 有多少内存，Operators 都会把内存瓜分干净，不会存在浪费的可能。

当然，为了运行的效率，我们一般建议单个 Slot 的 manage 内存应该大于 500MB。

另一个事情，在 1.10 后，我们去除了 OnHeap 的 manage 内存，所以只有 off-heap 的 manage 内存。

外部系统集成

Hive

强烈推荐 Hive Catalog + Hive，这也是目前批处理最成熟的架构。在 1.10 中，除了对以前功能的完善以外，其它做了几件事：

1. 多版本支持，支持 Hive 1.X 2.X 3.X
2. 完善了分区的支持，包括分区读，动态/静态分区写，分区统计信息的支持。
3. 集成 Hive 内置函数，可以通过以下方式来 load:

   a)TableEnvironment.loadModule(“hiveModule”,newHiveModule(“hiveVersion”))
4. 优化了 ORC 的性能读，使用向量化的读取方式，但是目前只支持 Hive 2+ 版本，且要求列没有复杂类型。有没有进行过优化差距在 5 倍量级。

兼容 Streaming Connectors

得益于流批统一的架构，目前的流 Connectors 也能在 batch 上使用，比如 HBase 的 Lookup 和 Sink、JDBC 的 Lookup 和 Sink、Elasticsearch 的 Sink，都可以在 Batch 无缝对接使用起来。

实践

SQL-CLI

在 1.10 中，SQL-CLI 也做了大量的改动，比如把 SQL-CLI 做了 stateful，里面也支持了 DDL，还支持了大量的 DDL 命令，给 SQL-CLI 暴露了很多 TableEnvironment 的能力，这让用户可以方便得多。后续，我们也需要对接 JDBC 的客户端，让用户可以更好的对接外部工具。但是 SQL-CLI 仍然待继续改进，比如目前仍然只支持 Session 模式，不支持 Per Job 模式。

编程方式

```
TableEnvironmenttEnv =
TableEnvironment.create(EnvironmentSettings
  .newInstance()
  .useBlinkPlanner()
  .inBatchMode()
  .build());
```

老的 BatchTableEnv 因为绑定了 Dataset，而且区分 Java 和 Scala，是不干净的设计方式，所以 Blink planner 只支持新的 TableEnv。

TableEnv 注册的 source, sink, connector, functions，都是 temporary 的，重启之后即失效了。如果需要持久化的 object，考虑使用 HiveCatalog。

```
tEnv.registerCatalog(“hive”,hiveCatalog);
tEnv.useCatalog(“hive”);
```

可以通过 tEnv.sqlQuery 来执行 DML，这样可以获得一个 Table，我们也通过 collect 来获得小量的数据：

```
Table table = tEnv.sqlQuery(“SELECTCOUNT(*)
FROM MyTable”);
List<Row>results =
TableUtils.collectToList(table);
System.out.println(results);
```

可以通过 tEnv.sqlUpdate 来执行 DDL，但是目前并不支持创建 hive 的 table，只能创建 Flink 类型的 table：

```
tEnv.sqlUpdate(
   "CREATE TABLE myResult (" +
```

```
      "  cnt BIGINT"
      ") WITH (" +
      "  'connector.type'='jdbc',"
         ……
      ")");
```

可以通过 tEnv.sqlUpdate 来执行 insert 语句，Insert 到临时表或者 Catalog 表中，比如 insert 到上面创建的临时 JDBC 表中：

```
tEnv.sqlUpdate("INSERTINTO myResult SELECT
COUNT(*) FROM MyTable");
tEnv.execute("MyJob");
```

当结果表是 Hive 表时，可以使用 Overwrite 语法，也可以使用静态 Partition 的语法，这需要打开 Hive 的方言：

```
tEnv.getConfig().setSqlDialect(SqlDialect.HIVE
);
```

结语

目前 Flink batch SQL 仍然在高速发展中，但是 1.10 已经是一个可用的版本了，它在功能上、性能上都有很大的提升，后续还有很多有意思的 features，等待着大家一起去挖掘。