Apache Hudi Clustering 数据布局功能介绍

背景

Apache Hudi将流处理带到大数据，相比传统批处理效率高一个数量级，提供了更新鲜的数据。在数据湖/仓库中，需要在摄取速度和查询性能之间进行权衡，数据摄取通常更喜欢小文件以改善并行性并使数据尽快可用于查询，但很多小文件会导致查询性能下降。在摄取过程中通常会根据时间在同一位置放置数据，但如果把查询频繁的数据放在一起时，查询引擎的性能会更好，大多数系统都倾向于支持独立的优化来提高性能，以解决未优化的数据布局的限制。本博客介绍了一种称为Clustering[RFC-19]的服务，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摄取速度之间进行权衡，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，我们引入了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。

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。
- 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。

调度Clustering

调度Clustering会有如下步骤

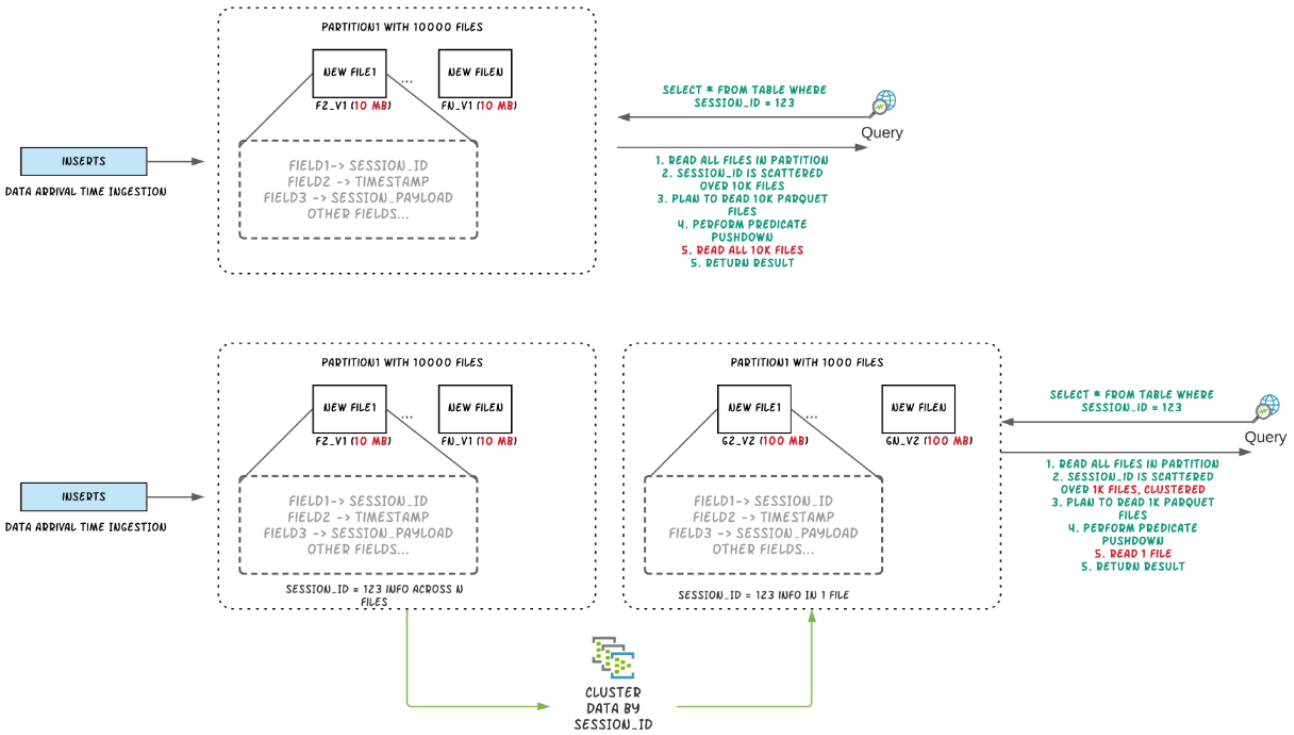
- 识别符合Clustering条件的文件：根据所选的Clustering策略，调度逻辑将识别符合Clustering条件的文件。
- 根据特定条件对符合Clustering条件的文件进行分组。每个组的数据大小应为targetFileSize的倍数。分组是计划中定义的"策略"的一部分。此外还有一个选项可以限制组大小，以改善并行性并避免混排大量数据。
- 最后将Clustering计划以avro元数据格式保存到时间线。

运行Clustering

- 读取Clustering计划，并获得clusteringGroups，其标记了需要进行Clustering的文件组。
- 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。
- 创建一个REPLACE提交，并更新HoodieReplaceCommitMetadata中的元数据。

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。

注意：现在对表进行Clustering时还不支持更新，将来会支持并发更新。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

Clustering配置

使用Spark可以轻松设置内联Clustering，参考如下示例

```
import org.apache.hudi.QuickstartUtils._
import scala.collection.JavaConversions._
import org.apache.spark.sql.SaveMode._
import org.apache.hudi.DataSourceReadOptions._
import org.apache.hudi.DataSourceWriteOptions._
import org.apache.hudi.config.HoodieWriteConfig._
val df =  //generate data frame
```

```
df.write.format("org.apache.hudi").
        options(getQuickstartWriteConfigs).
        option(PRECOMBINE_FIELD_OPT_KEY, "ts").
        option(RECORDKEY_FIELD_OPT_KEY, "uuid").
        option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
        option(TABLE_NAME, "tableName").
        option("hoodie.parquet.small.file.limit", "0").
        option("hoodie.clustering.inline", "true").
        option("hoodie.clustering.inline.max.commits", "4").
        option("hoodie.clustering.plan.strategy.target.file.max.bytes", "1073741824").
        option("hoodie.clustering.plan.strategy.small.file.limit", "629145600").
        option("hoodie.clustering.plan.strategy.sort.columns", "column1,column2"). //optional, if sorting is needed as part of rewriting data
        mode(Append).
        save("dfs://location");
```

对于设置更高级的异步Clustering管道，参考此处示例。

表查询性能

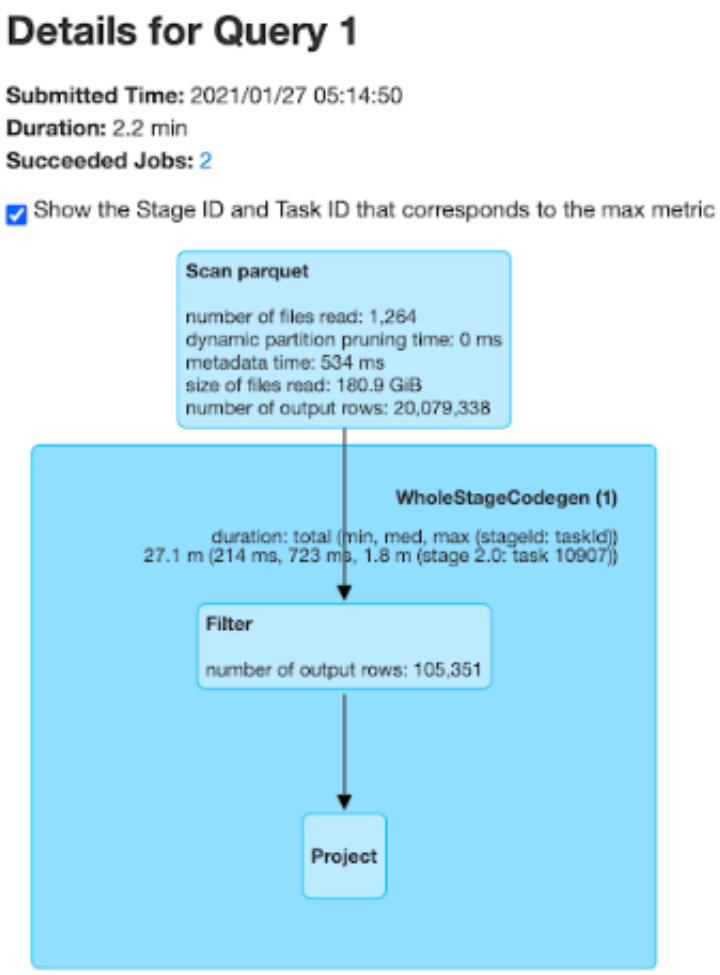
我们使用生产环境表的一个分区创建了一个数据集，该表具有约2000万条记录，约200GB，数据集具有多个session_id的行。用户始终使用会话谓词查询数据，单个会话的数据会分布在多个数据文件中，因为数据摄取会根据到达时间对数据进行分组。下面实验表明通过对会话进行Clustering可以改善数据局部性并将查询执行时间减少50%以上。

查询SQL如下

```
spark.sql("select  *  from table where session_id=123")
```

进行Clustering之前

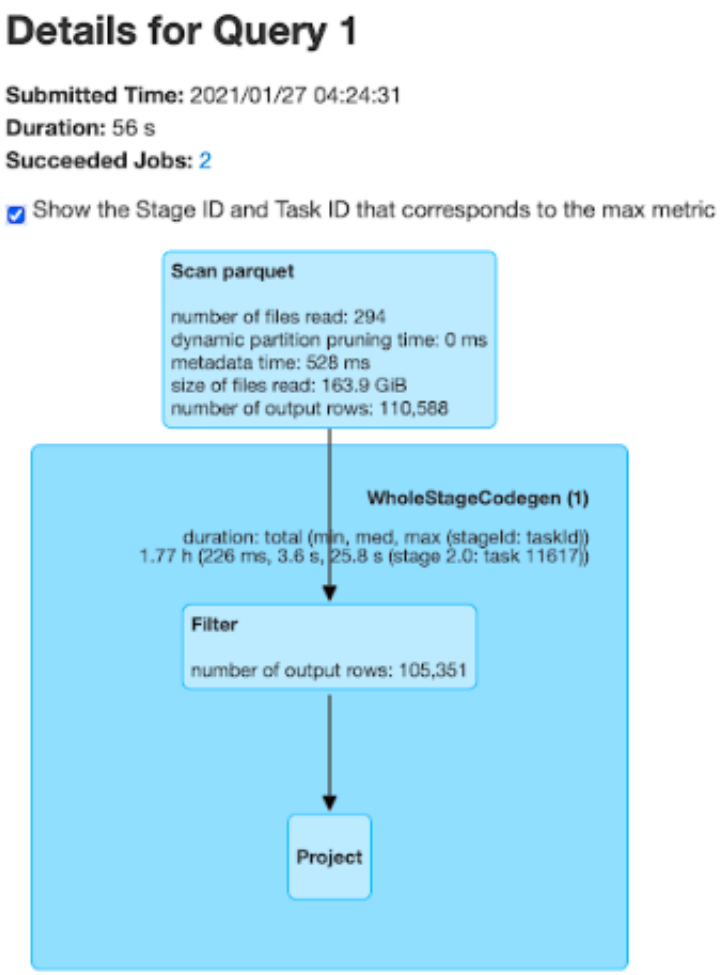
查询花费了2.2分钟。请注意查询计划的"扫描parquet"部分中的输出行数包括表中的所有2000W行。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

进行Clustering之后

查询计划与上面类似，但由于改进了数据局部性和谓词下推，Spark可以修剪很多行。进行Clustering后，相同的查询在扫描parquet文件时仅输出11万行（2000万行中的），这将查询时间从2.2分钟减少到不到一分钟。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

下表总结了使用Spark3运行的实验对查询性能的改进。

Table State	Query runtime	Num Records Processed	Num files on disk	Size of each file
Unclustered	130,673 ms	~20M	13642	~150 MB
Clustered	55,963 ms	~110K	294	~600 MB

Clustering后查询运行时间减少了60%，在其他样本数据集上也观察到了类似的结果，请参阅示例查询计划和RFC-19性能评估上的更多详细信息。

我们希望大型表能够大幅度提高速度，与上面的示例不同，查询运行时间几乎完全由实际I/O而不是查询计划决定。

总结

使用Clustering，我们可以通过以下方式提高查询性能：

- 利用空间填充曲线之类的概念来适应数据湖布局并减少查询读取的数据量。
- 将小文件合并成较大的文件以减少查询引擎需要扫描的文件总数。

Clustering使得大数据进行流处理，摄取可以写入小文件以满足流处理的延迟要求，可以在后台使用Clustering将这些小文件重写成较大的文件并减少文件数。

除此之外，Clustering框架还提供了根据特定要求异步重写数据的灵活性，我们预见到许多其他用例将采用带有自定义可插拔策略的Clustering框架来按需管理数据湖数据，如可以通过Clustering解决如下一些用例：

- 重写数据并加密数据。
- 从表中修剪未使用的列并减少存储空间。

本文转载自：查询时间降低60%！Apache Hudi数据布局黑科技了解下