



向 Kylin 添加实时 OLAP 能力

作者：Kyligence 大数据研发工程师 俞霄翔

时间：2019 年 3 月

Apache Kylin 在诞生之初，主要目的在于解决海量数据上进行交互式数据分析的需求，数据源主要来自于数据仓库（Hive），数据大都是历史的而非实时的。流式数据处理是一个大数据开发的新兴领域，它要求数据一旦进入系统即刻可被查询。直到 v2.6 版本，Apache Kylin 的主要能力仍然发挥在历史数据分析的领域，即便是 v1.6 引入的准实时数据摄入，依然有数分钟的延迟，难以满足对流式数据的实时查询需求。为了紧跟大数据开发的发展步伐，eBay 的 Kylin 开发团队([allenma](#), [mingmwang](#), [sanjulian](#), [wangshisan](#) 等)基于 Kylin 开发了 Real-time OLAP 的特性，实现了 Kylin 对 Kafka 流式数据的实时查询；此功能在 eBay 内部已经用于生产，并稳定运行超过一年时间，于 2018 年下半年开源，与社区共同改进和完善。

在这篇文章，我们将着重于介绍和分析 Apache Kylin 的 Real-time OLAP 功能、使用方式、基准测试等方面，在**什么是 Real-time OLAP**，我们将介绍架构、概念和功能；在**如何使用 Real-time OLAP**，我们将介绍 Receiver 集群的部署、启用和监控等方面；最后在**Real-time OLAP 常见问题**，我们将介绍一些常见问题的答案，重要配置项的含义，使用限制，以及未来的开发计划。

- **什么是 Real-time OLAP**
 - 流式数据处理和实时 OLAP
 - Real-time OLAP 的简介
 - Real-time OLAP 的概念和角色
 - Real-time OLAP 的架构
 - Real-time OLAP 的特性
 - Real-time OLAP 的 Local Segment Cache
 - Real-time OLAP 的元数据
 - Streaming Segment 的状态及其转化
 - Real-time OLAP 的构建流程分析
 - Real-time OLAP 的查询流程分析
 - Real-time OLAP 的 Rebalance 过程分析



- **如何使用 Real-time OLAP**
 - 部署 Coordinator 和 Receiver
 - 配置 Streaming Table
 - 增加和调整 Replica Set
 - 设计 Model 和 Cube
 - 启用和停止 Cube
 - 监控消费状况

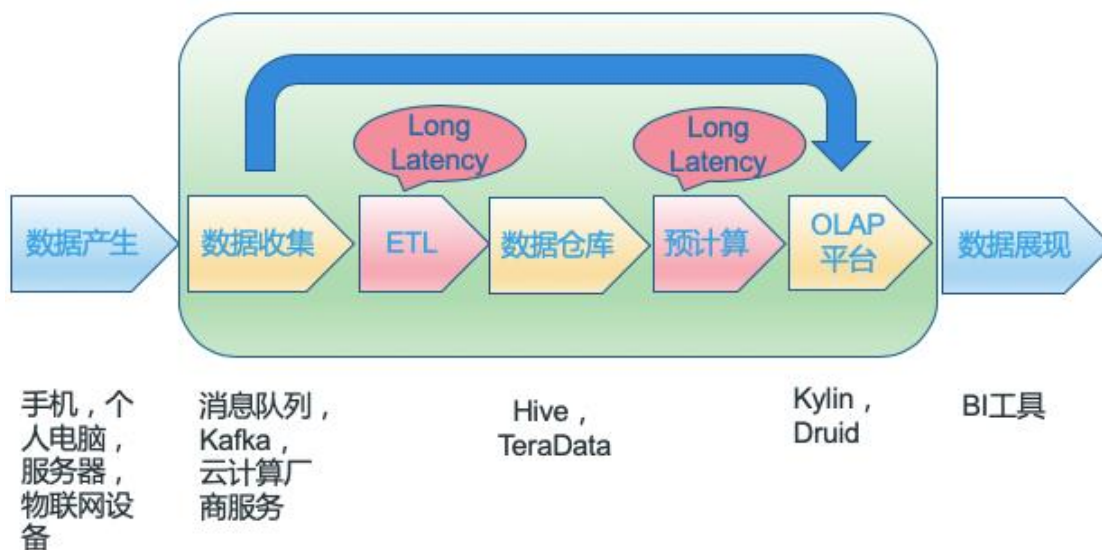
- **Real-time OLAP 的常见问题**
 - 在配置 Kafka 数据源时有一个“Lambda”勾选框，它是做什么的？
 - 除了 Base Cuboid，我还能在 Receiver 端构建其它 Cuboid 吗？
 - 我应该如何给我的 Receiver 集群进行扩容？Kafka Topic 增加 Partition 需要如何应对？
 - 基准测试结果如何？查询时长大概是多少？单个 Receiver 的数据摄入速度大概是多少？
 - 和 Kylin's NRT Streaming 相比较，哪个更加适合我的需求？
 - 目前 Real-time OLAP 有哪些主要限制？未来的开发计划有哪些？

什么是 Real-time OLAP for Kylin

流式数据处理和实时 OLAP

对于很多商业公司，用户消息被分析后可用于商业决策和制定更好的市场计划，若消息更早地进入数据分析平台，那么决策者可以更快地做出响应，从而减少时间和资金的浪费。利用流式数据处理，意味着更快的信息反馈，决策者因此可以进行更加频繁和灵活的计划调整。

企业数据源类型多样，包括服务器、手机等移动设备、物联网设备等，来自不同源头的消息往往通过不同的主题加以区分，汇聚到消息队列(Message Queue/Message Bus)以供数据分析使用。传统的数据分析工具使用批处理工具如 MapReduce 来进行数据分析，其数据延迟较大，通常为数小时到数天。从下图我们可以看出，主要的数据延迟来自于两个过程：从消息队列通过 ETL 流程抽取到数据仓库，和从数据仓库抽取数据进行预计算将结果保存为 Cube 数据。由于这两部分都是使用批处理程序进行计算，计算耗时较长，无法满足实时查询需求，于是我们想到要解决问题只能绕过这些过程，通过在数据收集和 OLAP 平台间架起一道桥梁，让数据直接进入 OLAP 平台。

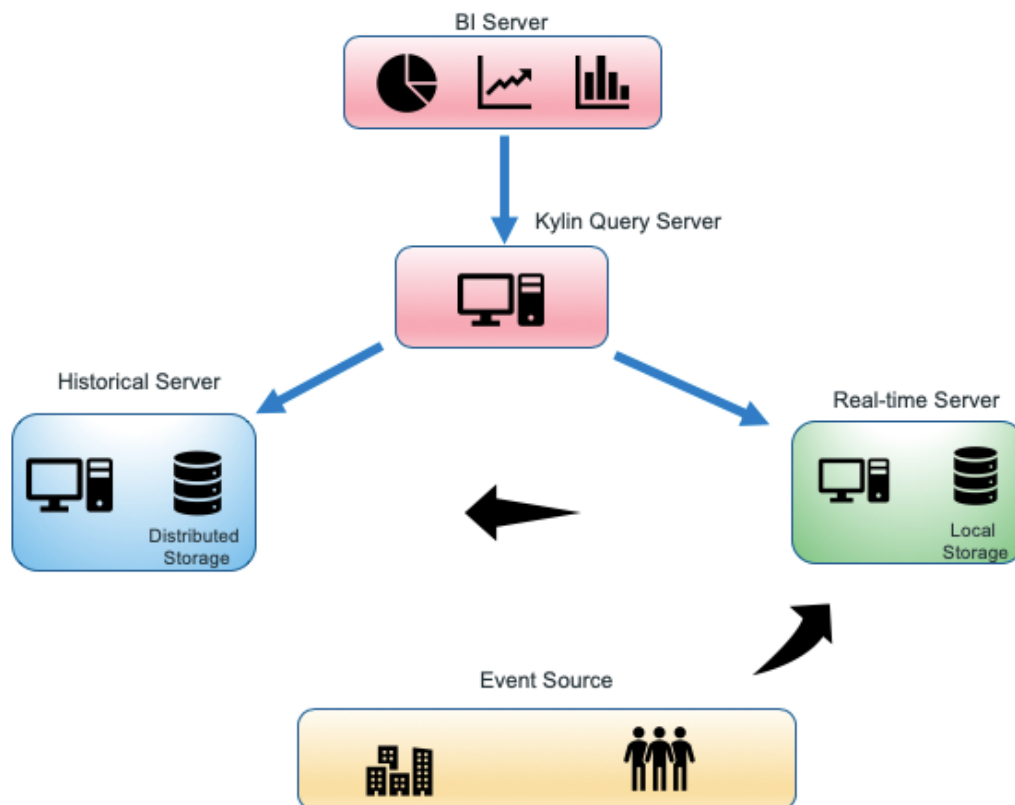


目前已经有一些成熟的实时 OLAP 方案，如 Druid，通过合并实时和历史部分的查询结果提供较低的数据延迟。目前 Kylin 在分析海量历史数据的方面达到了一定的水平，为了向实时 OLAP 领域迈进一步，Kylin 开发者们开发了 Real-time OLAP。

Real-time OLAP 的简介

在新的架构下，数据查询请求根据时间分区列（Timestamp Partition Column）分为两部分，历史数据的查询请求仍将发送给 HBase Region Server，最新时间段的查询请求将发送到实时计算节点，Query Server 需要将两者的结果整合后返回给查询客户端。

与此同时，实时计算节点会不断将本地数据上传到 HDFS，在满足一定条件时会通过 Hadoop 来构建 segment，从而完成实时数据向历史数据的转化，并且实现了降低实时计算节点压力的目的。



Real-time OLAP 的概念和角色

为实现 Real-time OLAP，Kylin 引入了一些新的概念，这里跟大家做一个初步的介绍。

1. Streaming Receiver

Streaming Receiver 的角色是 worker，每个 receiver 是一个 Java 进程，受 Coordinator 的管理，它的主要职责包含：

- 实时摄入数据；
- 在内存构建 cuboid，定时将保存在内存的 cuboid 数据 flush 到磁盘，形成 Fragment 文件；
- 定时 checkpoint 和合并 Fragment 文件；
- 接受对它所负责的 Partition 的查询请求；



- 当 segment 变为不可变后，上传到 HDFS 或者从本地删除（依据配置）；

2. Receiver Cluster

Streaming Receiver 组成的集合称为 Receiver 集群。

3. Streaming Coordinator

Streaming Coordinator 作为 Receiver 集群的 Master 节点，主要职责是管理 Receiver，包括将 Kafka topic 的 partition 分配/解除分配到指定的 Replica set、暂停或者恢复消费、收集和展示各项统计指标(例如 message per second)。当 kylin.server.mode 被设置为 all 或者 stream_coordinator，这个进程就成为一个 Streaming Coordinator。Coordinator 只处理元数据和集群调度，不摄入消息。

4. Coordinator Cluster

多个 Coordinator 可以同时存在，组成一个 Coordinator 集群。在多个 Coordinator 中，同一时刻只存在一个 Leader，只有 Leader 才可以响应请求，其余进程作为 standby/backup。

5. Replica Set

Replica Set 是一组 Streaming Receiver，它们动作一致。Replica Set 作为任务分配的最小单位，Replica Set 下的所有 Receiver 做相同的工作（即摄入相同的一组 partition），互为 backup。当集群中存在一些 Receiver 进程无法访问，但能保证每一个 Replica Set 至少存在一个健康的 Receiver，那么集群仍能正常工作并且返回合理的查询结果。

在一个 Replica Set 中，将存在一个 Leader Receiver 来做额外的工作，其余的 Receiver 作为 Follower。

6. Streaming Segment

当 Receiver 摄取一个新的消息，并且这个消息的时间分区列的时间戳不包含于现有的任意 Segment，那么 Receiver 会创建一个新的 Segment，这个 Segment 的初始状态为 **Active**，并且这个 Segment 的开始时间和结束时间的间隔等于 **Segment Window**，所有时间分区列的时间戳包含在这个 Segment 的开始时间和结束时间之间的消息，将由这个 Segment 负责。当时间达到结束时间，这个 Segment 并不会立即关闭，这是为了等待那些延迟的消息，但是等待时间并不是永久的，一旦满足以下条件后之一后，Segment 的状态将转为 **IMMUTABLE**，一旦处于 **IMMUTABLE**，延迟的消息默认将被丢弃：

- 在 **Segment Duration** 长度的时间段一直未收到属于本 Segment 时间段的消息
- 等待时间的总和超过了某一个固定的阈值

7. Retention Policy

当 Segment 转变为 **IMMUTABLE** 后，该 Segment 的本地数据如何处理将由这个配置项决定，这个配置项目前有两个选项：

- **FULL_BUILD** 当前 Receiver 进程是所属 Replica Set 的 Leader 时，会上传 Segment 的本地数据文件到 HDFS，并且当上传成功后，将 Segment 状



态置为 **REMOTE_PERSISTED**；若是所属 Replica Set 的 Follower 时，不做处理。

- **PURGE** 等待一定时间然后删除本地数据文件
若用户只对最近一段时间的数据分析结果感兴趣，可以考虑使用 **PURGE** 选项

8. Assignment & Assigner

Assignment 是一种 Map 型的数据结构，其 key 是 Replica Set 的 ID，value 是分配给该 Replica Set 的 Partition 列表（用 Java 语言来表示的话是：Map<Integer, List<Partition>>），下图是一个简单的例子。

```
1 @{
2     "2":@[
3         @{
4             "partition_id":3,
5             "partition_info":null
6         },
7         @{
8             "partition_id":0,
9             "partition_info":null
10        }
11    ],
12    "3":@[
13        @{
14            "partition_id":5,
15            "partition_info":null
16        },
17        @{
18            "partition_id":4,
19            "partition_info":null
20        }
21    ]
22 }
```

Assignment 十分重要，它表示了一个 Kafka Topic 的 Partition 分别是由哪些 Replica Set 去负责的，了解它对如何使用 Rebalance 相关的 API 和学习 Streaming 元数据十分重要。

基于当前集群资源和 Topic Partition 数量，如何进行合适的 partition 分配，有不同的策略，这些策略由 **Assigner** 负责。**Assigner** 目前有两种具体实现，分别是 **CubePartitionRoundRobinAssigner**，**DefaultAssigner**。

9. Checkpoint

Checkpoint 能够让 Receiver 重新启动后，从上次结束的安全点继续消费。

Checkpoint 使得 receiver 重启后，数据能够不丢失，同时尽可能减少数据的重复消



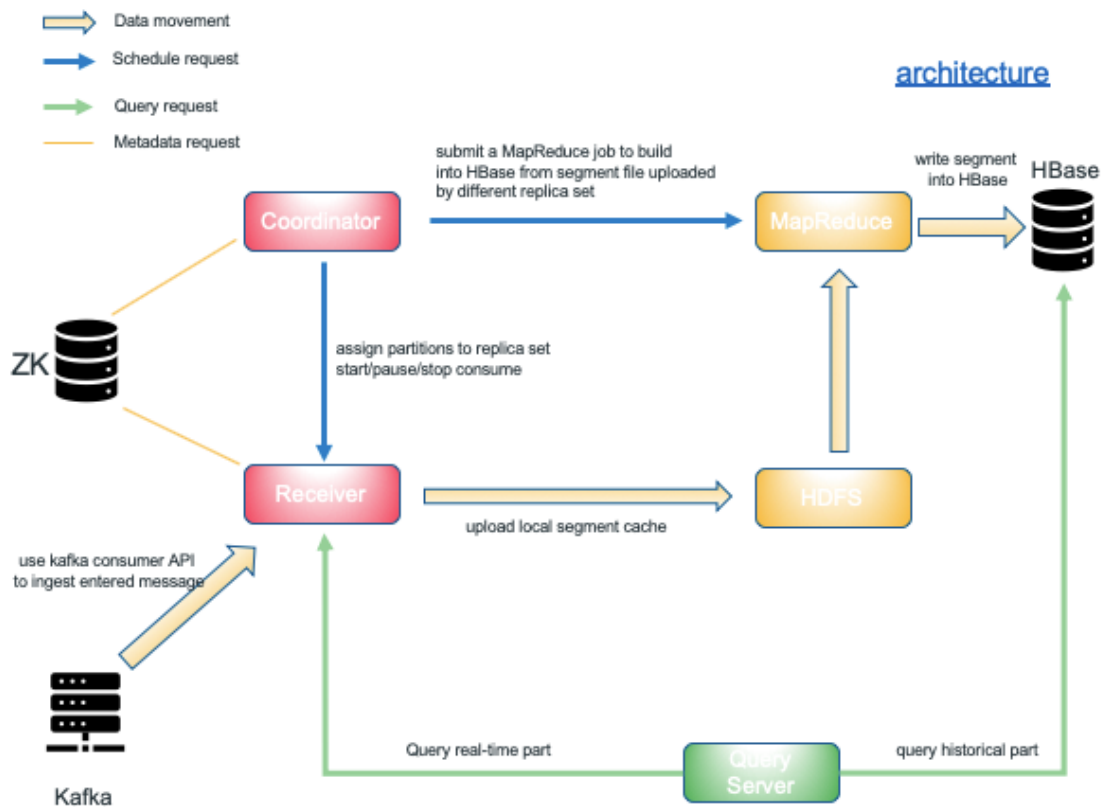
费。Checkpoint 主要分为两种，一种是 local checkpoint，一种是 remote checkpoint。其中 remote checkpoint 发生在将本地 segment 数据文件上传到 HDFS 时，将 offset 信息记录在元数据里；local checkpoint 是由 receiver 定时调度或者由事件触发，会将数据 flush 到本地，并在本地文件记录 offset。当 Receiver 启动 Kafka Consumer API 时，会尝试检查 local checkpoint 和 remote checkpoint，寻找最新的 offset 开始消费。

Real-time OLAP 的架构

数据流向方面，我们能看到数据的流向是从 Kafka 到 Receiver，再由 Receiver 上传到 HDFS，最后由 MapReduce 程序合并和重新加工 Segment 进入 HBase。

查询方面，查询请求由 Query Server 发出，根据查询条件中出现的时间分区列，分发请求到 Receiver 和 HBase Region Server 两端。

Topic Partition 的分配和 Rebalance、Segment 状态管理和作业提交由 Coordinator 负责。





Real-time OLAP 的特性

1. 数据一旦进入，将立刻在内存计算 cuboid，即刻可被查询（毫秒级数据延迟）；
2. 自动化的数据状态管理和作业调度；
3. 根据查询条件，查询结果可包含实时结果和历史结果；
4. 实时部分使用列式存储和倒排索引加速查询，降低查询延迟；
5. 一个新的分布式计算和存储集群被引入到 Apache Kylin；
6. Coordinator 和 Receiver 具有高可用性。

Real-time OLAP 的 Local Segment Cache

Receiver 端的 Segment 数据由两部分组成，MemoryStore 和 Fragment File。在 MemoryStore 内部是用 Map<String[], MeasureAggregator[]>存储聚合后的数据，其中 key 是维度值组成的字符串数组，value 是 MeasureAggregator 数组。

随着 Receiver 不断摄入消息，当 MemoryStore 的数据行数达到阈值会触发 flush 操作，将整个 MemoryStore flush 到磁盘形成一个 Fragment File。对 Fragment File 的读取使用了 **memory-mapped file** 来加速读取速度，可以参考[源代码](#)。另外，Fragment File 为了优化扫描和过滤性能也使用了多种方式来加速扫描，包括：

- 压缩
- 倒排索引
- 列式存储

最后，Fragment File 可以进行 Merge 来减少数据冗余，提升扫描性能。

Real-time OLAP 的元数据结构

Real-time OLAP 增加了集群管理和 Topic Partition 的分配关系、Replica Set 等新的元数据类型，这些元数据目前由保存在 Zookeeper 中，主要包括：

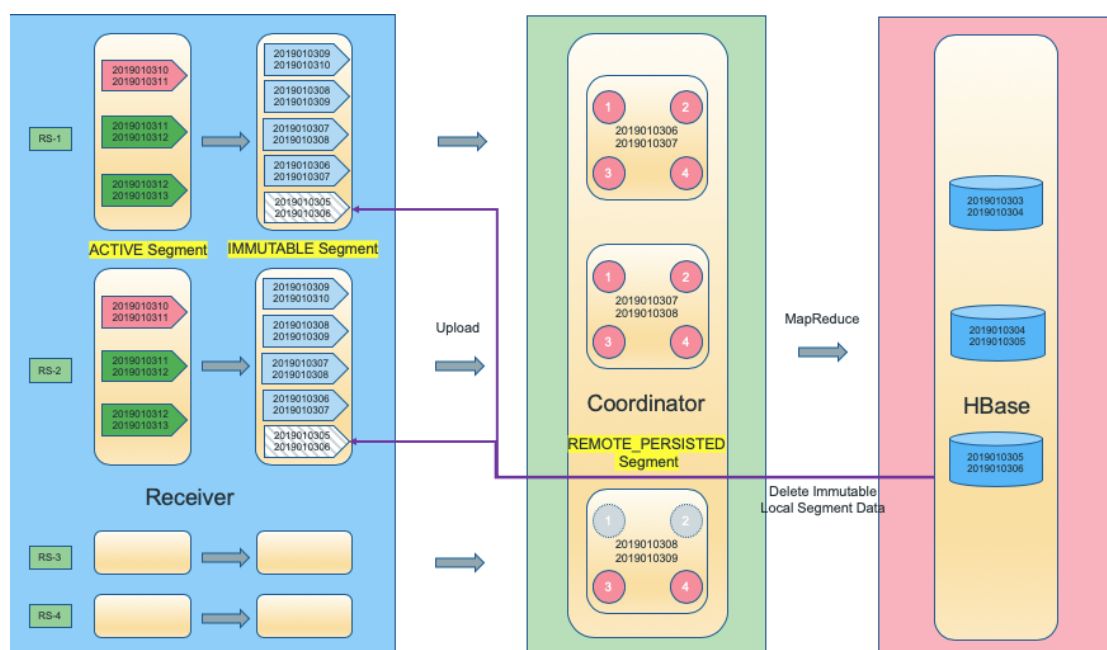
1. 当前的 Coordinator Leader 节点

2. Receiver 节点信息
3. Replica Set 信息和 Replica Set Leader 节点
4. Cube 的 Assignment 信息
5. Cube 下 Segment 的构建状态和上传完整度

Streaming Segment 的状态及其转化

我们可以把 Segment 的状态转变过程表示为下图，从左到右共有四个 **FIFO** 队列，队列的成员是 Segment，每个 Segment 会依次从左到右进入下个队列，这四个队列分别为

- **Active Segment**
在 Receiver 端，可以接受写入的 Segment
- **Immutable Segment**
在 Receiver 端，已经不接受写入，但是尚未上传到 HDFS 上的 Segment
- **RemotePersisted Segment**
在 Receiver 端，已经不接受写入，并且已经上传到 HDFS 上，但是尚未通过 MapReduce 构建入 HBase 的 Segment（只有 Replica Set Leader 上才有）
- **Ready Segment**
已经通过 MapReduce 构建进入 HBase 的 Segment，在 Receiver 被删除。





1. 左边的蓝色区域表示的 Receiver
2. 中间的绿色区域表示的 Coordinator
3. 右边的红色部分表示的是 HBase
4. 数据的流向是从左到右

处于 **Active** Segment 和 **Immutable** Segment 队列的 segment 由 Receiver 负责管理，所属的 Segment 图标是五角形，每个五角形表示的是 Segment 数据的一部分，因为一个 Replica Set 通常只负责 Topic 的一部分 Partition。从 **Active** Segment 到 **Immutable** Segment 的转化由 **Segment Duration** 和 **Segment Window** 控制。

当 Segment 状态变为 **Immutable** 后，**Retention Policy** 为 **FullBuild** 并且当前 Receiver 是 replica set 的 Leader 时，Segment 的本地列存格式数据文件会排队上传到 HDFS 上，上传成功后状态变为 **RemotePersisted**。

RemotePersisted Segment 队列由 Coordinator 负责管理，所属的 Segment 图标是圆角正方形，里面的每一个红色圆表示对应编号的 Replica Set 已经把 Segment 数据文件上传到 HDFS 上了，每个灰色圆表示该 Replica Set 尚未完成 segment 数据上传。存在灰色圆的圆角正方形表示，该 Segment 因为在 HDFS 上的数据尚不完整，不能立刻进行 MapReduce 构建。

当一个 Segment 的所有数据文件都上传完毕后，该 Segment 会被提交 MapReduce 作业构建成 HBase Segment。

一旦 MapReduce 作业成功，Segment 顺利进入 HBase 后，Coordinator 会通知相关的 Receiver 删除本地数据，并且清除 Segment 暂存在 HDFS 上的数据，将 HBase Segment 的状态设置为 **READY**。

若 Receiver 所属 Cube 下状态是 **Immutable** 和 **RemotePersisted** 的 Segment 数量过多，会抑制消费行为，直到 **Immutable/ RemotePersisted** 数量低于阈值。

Real-time OLAP 的构建流程分析

取决于 Retention Policy 的设置，当设置为 Full Build 时，Immutable 的 segment 会被上传到 HDFS，等到全部 Replica Set 上传完毕时，Coordinator 会进行 build cube，通过 MapReduce 将上传到 HDFS 的 segment cache 转换为 HFile 保存到 HBase；当设置为 Purge 后，本地的 segment 将被定期删除，没有上传和构建动作。

构建步骤

1. Merge dictionary
2. Convert to base cuboid from columnar record



3. Using InMem Cubing build segment

Real-time OLAP 的查询流程分析

对用户提交的 SQL 语句，根据时间分区列来确定是否需要查询历史和实时这两部分

1. 对历史部分的查询来自于所有状态为 Ready 的 HBase Segment，并记录最新 HBase Segment 的 End time ；
2. 对实时部分的查询来自于 Receiver 集群，时间条件限制到最新 HBase Segment 的 End time 以后的 Segment。

同一 Replica Set 下的 Receiver 可能存在摄入速率不一致的情况，为了避免查询结果不一致，对于单个 Cube 查询将尽可能查询固定的 Receiver，除非首选的 Receiver 不可访问，才会查询其他 Receiver。

Real-time OLAP 的 Rebalance/Reassign 流程分析

简介

为了应对 Kafka 单位时间消息量猛增，或者某些 Replica Set 下的 Receiver 需要下线等情况，需要重新调整 Cube 的 Replica Set 和 Partition 的分配关系，使得整个 Receiver 集群处于负载均衡的状态。此时需要通过重分配 (Reassign) 操作来达成目的。

通过 Reassign 操作，我们可以将移除的 Replica Set 所属的 Receiver 的摄入操作停止，将其所负责的 Partition 移交给新的 Replica Set 去负责。这个过程是由 Coordinator 向 Receiver 发送 REST API 来完成的，过程中并不需要数据移动，整个过程的耗时一般取决于 Reassign 涉及的节点数，一般在数秒内完成。

需要注意的是，被移除的 Replica Set 所属的 Receiver 在 Reassign 完成后，并不能立刻结束进程，这是因为数据不能保证已经完成上传到 HDFS，并且查询也需要这些移除了的 Replica Set。

步骤

Reassign 是从 **CurrentAssignment** 到 **NewAssignment** 的过程，整个 Reassign 操作是一个分布式事务，整体可以分为四个步骤，步骤失败会进行自动回滚操作。

- **StopAndSync** 停止消费和同步 Offset



- 停止 **CurrentAssignment** 的全部 Receiver 的消费，每个 Receiver 向 Coordinator 汇报 Offset
- Coordinator 合并每个 Partition 的 Offset，保留最大者，通知 Receiver 消费到指定 Offset 并结束消费行为
- **AssignAndStart**
 - AssignNew 向 **NewAssignment** 的全部 Receiver 发送 assign 请求，更新它们的 Assignment
 - StartNew 向 **NewAssignment** 的全部 Receiver 发送 startConsumer 请求，要求他们根据上一步的 Assignment 启动 Consumer 进行信息消费
- **ImmutableRemoved** 向移除的 Replica Set 所属的 Receiver 发送 ImmutableCube 请求，要求它们强制将 Cube 下的全部 Active Segment 转变为 Immutable Segment
- **Update MetaData** 更新元数据，将 NewAssignment + RemovedAssignment 记录到元数据中（被移除的 ReplicaSet 在 ReAssign 完成后仍将接受查询请求）

若一个 Replica Set 在 Reassign 过程中未改变分配关系，则会跳过全部步骤。

Reassign 完成后向 NewAssignment 添加了 Remove Replica Set, List 为空

```
private CubeAssignment reassignCubeImpl(String cubeName, CubeAssignment
preAssignments,
    CubeAssignment newAssignments) {
    logger.info("start cube reBalance, cube:{}, previous assignments:{}, new
assignments:{}", cubeName,
        preAssignments, newAssignments);
    if (newAssignments.equals(preAssignments)) {
        logger.info("the new assignment is the same as the previous assignment, do
nothing for this reassignment");
        return newAssignments;
    }
    CubeInstance cubeInstance =
CubeManager.getInstance(KylinConfig.getInstanceFromEnv()).getCube(cubeName);
    doReassign(cubeInstance, preAssignments, newAssignments);
    MapDifference<Integer, List<Partition>> assignDiff =
Maps.difference(preAssignments.getAssignments(),
```



```
newAssignments.getAssignments());

// add empty partitions to the removed replica sets, means that there's still data in
the replica set, but no new data will be consumed.

Map<Integer, List<Partition>> removedAssign = assignDiff.entriesOnlyOnLeft();

for (Integer removedReplicaSet : removedAssign.keySet()) {

    newAssignments.addAssignment(removedReplicaSet, Lists.<Partition>
newArrayList());

}

streamMetadataStore.saveNewCubeAssignment(newAssignments);

AssignmentsCache.getInstance().clearCubeCache(cubeName);

return newAssignments;

}
```

当被移除的 Replica Set 的 Segment 全部上传至 HDFS 并完成构建进入 HBase 后，会清除这些空 List 的 Assignment

```
// for the assignment that doesn't have partitions, check if there is local segments exist

if (assignments.getPartitionsByReplicaSetID(replicaSetID).isEmpty()) {

    logger.info(

        "no partition is assign to the replicaSet:{}, check whether there are local
segments on the rs.",

        replicaSetID);

    Node leader = rs.getLeader();

    try {

        ReceiverCubeStats receiverCubeStats =
receiverAdminClient.getReceiverCubeStats(leader, cubeName);

        Set<String> segments = receiverCubeStats.getSegmentStatsMap().keySet();

        if (segments.isEmpty()) {

            logger.info("no local segments exist for replicaSet:{}, cube:{}, update
assignments.",

                replicaSetID, cubeName);

        }

    }

}
```

```

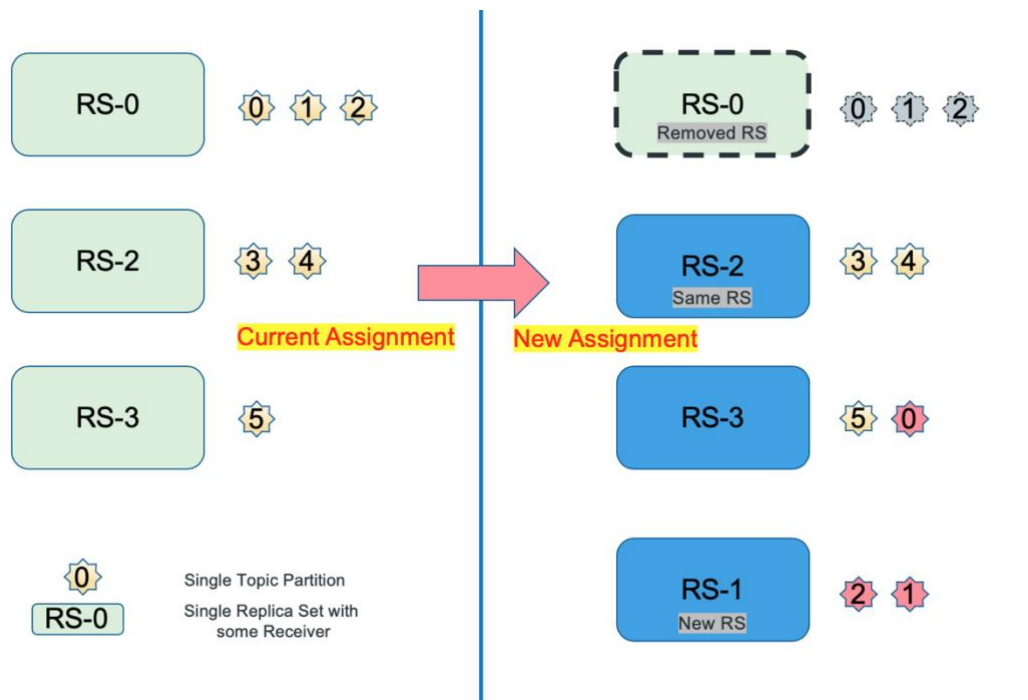
    assignments.removeAssignment(replicaSetID);

    streamMetadataStore.saveNewCubeAssignment(assignments);
}
} catch (IOException e) {
    logger.error("error when get receiver cube stats from:" + leader, e);
}
}
}

```

图解

如下图所示，在 Reassign 发生以前，该 Cube 由三个 Replica Set 负责消费，分别为 RS-0、RS-2、RS-3，并且各个 Replica Set 消费压力不均衡。



现在因为某些原因，将 RS-0 下线，并添加了 RS-1。需要将 RS-0 负责的 Partition 交由其它 Replica Set 负责消费，这里将 partition-0 移交 RS-3 负责，partition-1 和 partition-2 移交 RS-1 负责。

RS-0 在 Reassign 结束后不再进行消费数据，我们称之为 **Removed RS**。

RS-2 在 Reassign 过程中完全没有改变分配关系，我们称之为 **Same RS**。

RS-1 在 Reassign 过程新增的 RS，我们称之为 **New RS**。

- **CurrentAssignment**

```

{

```



```
"rs-0":[
  {
    "partition_id":0
  },
  {
    "partition_id":1
  },
  {
    "partition_id":2
  }
],
"rs-2":[
  {
    "partition_id":3
  },
  {
    "partition_id":4
  }
],
"rs-3":[
  {
    "partition_id":5
  }
]
]
```

- **NewAssignment**

```
{
  "rs-1":[
    {
```



```
        "partition_id":1
    },
    {
        "partition_id":2
    }
],
"rs-2":[
    {
        "partition_id":3
    },
    {
        "partition_id":4
    }
],
"rs-3":[
    {
        "partition_id":5
    },
    {
        "partition_id":0
    }
]
]
```

具体步骤如下

- StopAndSync
在这一步，我们停止 **CurrentAssignment** 包含的全部 Rs 的消费行为，但是跳过 **Same RS**，即 RS-2。
- AssignAndStart
在这一步，我们向 **NewAssignment** 包含的全部 Rs 发出 Assign 和 StartConsumer 请求，同样跳过 **Same RS**，即 RS-2。



- ImmutableRemoved
在这一步，我们向 **Removed RS** 发送强制 Immutable 请求，**Removed RS** 会尽快上传它所负责的那部分数据到 HDFS。
- Update MetaData
在这一步，Coordinator 会更新元数据，将 **NewAssignment** 加上 **Removed RS** 作为真正的 **NewAssignment** 写入 Metadata。

在 **Removed RS** 完成上传并且相关 Segment 成功构建进入 HBase 之前，查询请求都要向 RS-0 到 RS-3 发送。

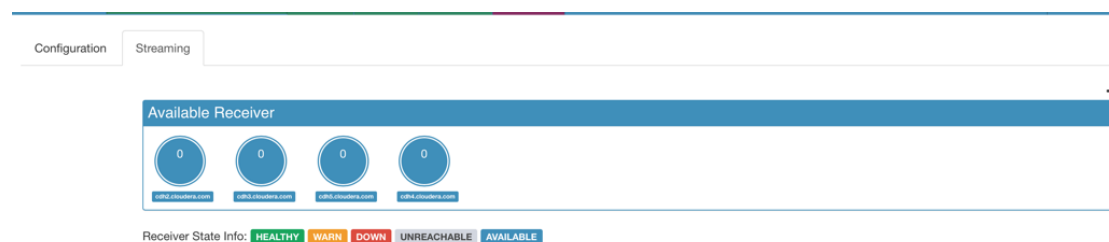


如何使用 Real-time OLAP

部署 Coordinator 和 Receiver

1. 下载 Kylin 的二进制包或者自己获取源代码编译打包
2. 部署 Coordinator
当 `kylin.server.mode` 设置为 `all` 或者 `stream_coordinator` 时，当前的 `kylin` 进程就具有 Coordinator 角色，启动命令仍为 `sh bin/kylin.sh start`。如要保证 Coordinator 不出现单点问题，可以启动多个 Coordinator。启动成功 Coordinator 会注册自身到元数据。
3. 部署 Receiver
启动 Receiver 的启动命令：`sh bin/kylin.sh streaming start`，启动成功 Receiver 会注册自身地址到元数据。Receiver 进程的数量，需要根据消息的产生速率，物理节点的配置，和你对 HA 的需要来确定。你可以在你的环境进行测试来确定适合的 Receiver 进程的数量，或者参考[我的基准测试](#)。
4. 若 Coordinator 和 Receiver 的端口不使用默认端口（Coordinator 的默认端口 7070，Receiver 的默认端口是 9090），需要在 `kylin.properties` 额外配置 `kylin.stream.node=HOST_NAME:PORT`。

若一切顺利的话，可以在 Kylin 的 **System** 页面下的 Streaming Tab 看到待分配的 Receiver，如图所示可见有四个已注册并且处于待分配状态的 Receiver。



配置 Streaming Table



1. 在 Data Source 下选择第四个选项“Add Streaming Table V2”

Kylin test_full_build Insight Model Monitor System

Models Data Source

Tables

No Result.

No tables
Click here to load your table

2. 输入 Topic 的名称和 Broker 信息

Kafka Config

Topic* skyrim1

Bootstrap Servers*	Host	Port	Action
	cdh1.cloudera.com	9092	-
	cdh2.cloudera.com	9092	-
	cdh3.cloudera.com	9092	-
	cdh4.cloudera.com	9092	-
	cdh5.cloudera.com	9092	-
	cdh6.cloudera.com	9092	-
	cdh7.cloudera.com	9092	-
	cdh8.cloudera.com	9092	-

Next →



3. 在下一个页面粘贴 Message 的 Sample，设置表名，点击完成

Streaming Table x
Map Streaming Schema to Table

Please input topic record sample, or automatically get it from

JSON

```
1 [{"province": "Summerset_Isles", "lid": "e80160bc-f25a-3566-bf9b-a16e91ef6ee4", "uid": 1, "level": 10, "coins": 6.9, "classes": "Necromancer", "experience": 424, "alliance": "Ebonheart_Pact", "race": "Dunmer", "dt": 1546385688791, "hold": "Auridon", "reputation": 59.06304}]
```

Table Name* Lambda

TSColumn*

Column	Type	Mapping	Comment
province	varchar(2000)		
lid	varchar(2000)		
uid	int		
level	int		
coins	decimal		
classes	varchar(2000)		
experience	int		
alliance	varchar(2000)		
race	varchar(2000)		
dt	timestamp		
hold	varchar(2000)		
reputation	decimal		
year_start	date		derived time dimension
quarter_start	date		derived time dimension

增加和调整 Replica Set

1. 为了添加新的 Replica Set，回到 Streaming Tab 下，点击加号

Kylin Insight Model Monitor System

Configuration Streaming

create replica set +

Available Receiver

0

10.1.0.51

Receiver State Info: HEALTHY WARN DOWN UNREACHABLE AVAILABLE

unallocated receiver



- 为新的 Replica Set 添加 Receiver，然后单击 Save

Create Replica Set

Nodes:

10.1.0.51:9090 ✕

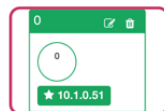
Add more than one receiver node will enable HA

Save

Cancel

- 回到 Streaming Tab，观察到新的 Replica Set 已经被成功创建（ID 为 0），这样我们就可以为 Cube 分配 Replica Set 去摄入数据了

Configuration Streaming



You will get a replica set which id is 0.
This rs contain only one node(10.1.0.51).

Available Receiver

There are no available receivers.

Receiver State Info: HEALTHY WARN DOWN UNREACHABLE AVAILABLE

若部署了多个 Receiver，可以为每一个 Replica Set 选择多个 Receiver，这样可以保证在单个 Receiver 不可用的情况下，不会导致无法查询。如下图是四个 Replica Set，每个 Replica Set 拥有两个 Receiver。

Streaming



Available Receiver

There are no available receivers.

Receiver State Info: HEALTHY WARN DOWN UNREACHABLE AVAILABLE

- 若需要向现有的 Replica Set 增加或者减少 Receiver，可以点击 Replica Set 右上角的编辑图标；若想删除 Replica Set，可以点击垃圾桶图标



设计 Model 和 Cube

在模型和 Cube 的设计过程需要注意以下：

1. 设计模型过程没有差别，需要注意 Streaming Table 不能和维度表进行 Join。
2. 在设计 Cube 过程中，需要注意不能使用 TopN，也不能对整型以外的字段使用 COUNT_DISTINCT (Bitmap)。
3. 在 Streaming Properties，可以设置一些和 Streaming 功能相关的配置项。
 - 选择 **fullBuild**

Kylin Properties

kylin.stream.index.maxrows	1000000	-
kylin.stream.build.additional.cuboids	true	-
+ Property		

Streaming Properties

kylin.cube.algorithm	INMEM
kylin.stream.cube.window	3600
kylin.stream.cube.duration	3600
kylin.stream.index.checkpoint.intervals	300
kylin.stream.segment.retention.policy	fullBuild purge

- 选择 **purge**



Kylin Properties

kylin.stream.index.maxrows	1000000	-
kylin.stream.build.additional.cuboids	true	-
+ Property		

Streaming Properties

kylin.cube.algorithm	INMEM
kylin.stream.cube.window	3600
kylin.stream.cube.duration	3600
kylin.stream.index.checkpoint.intervals	300
kylin.stream.segment.retention.policy	fullBuild purge
kylin.stream.segment.retention.policy.purge.retentionTimeInSec	86400

这里对重要的参数进行说明

- kylin.cube.algorithm
上传到 HDFS 的 Segment 文件以何种算法构建进入 HBase，目前只有一个算法，即 InMemory 算法。
- kylin.stream.cube.window
决定一个 segment 的时间跨度，默认值是一小时
- kylin.stream.cube.duration
决定一个 segment 在达到结束时间之后，还会等待迟到的事件多久
- kylin.stream.index.checkpoint.intervals
在 Receiver 端定期进行 checkpoint 的间隔
- kylin.stream.segment.retention.policy
当 Segment 的状态变为 IMMUTABLE，Receiver 应该如何处理这个 segment。可选值为 fullBuild 和 purge
- kylin.stream.segment.retention.policy.purge.retentionTimeInSec
当 kylin.stream.segment.retention.policy 为 purge，决定了当 segment 变为 IMMUTABLE 后在 Receiver 端保留多久
- kylin.stream.index.maxrows
MemoryStore 保存的最大行数



- kylin.stream.build.additional.cuboids
除了 base cuboid，是否在 Receiver 端构建额外的 cuboid，选择是的话会在 Receiver 端构建 **Mandatory Cuboids**

启用和停止 Cube

- 当前面的步骤都完成后，现在需要启用 Cube。Assigner 会依据一定的规则将 Partition 分配给 Replica Set。

start cosume kafka message

Name	Status	Cube Size	Source Records	Last Build Time	Owner	Create Time	Actions	Admins
skyrim_3_cube	DISABLED	0.00 KB	0		ADMIN	2018-12-25 19:19:28 GMT+8	Drop Edit Merge Enable Purge Clone	Action

- 当 Cube 成功自动分配并启动自动消费后，可以在观察 Cube 的状态变为 READY，并且可以在 Streaming Tab 下观察到摄入速率。

Receiver 地址下方加粗的速度为最近一分钟的平均摄入速度

Name	Status	Cube Size	Source Records	Last Build Time	Owner	Create Time	Actions	Admins
cube2	READY	0.00 KB	0		ADMIN	2019-01-08 11:07:32 GMT+8	Action	Action

Replica Set ID	cdh2.cloudera.com:9091	cdh2.cloudera.com:9093
0	380 msg/s 5' 380msg/s 15' 380msg/s avg 370msg/s consume 2762 ingest 2762	380 msg/s 5' 380msg/s 15' 380msg/s avg 369msg/s consume 2762 ingest 2762
1	358 msg/s 5' 358msg/s 15' 358msg/s avg 361msg/s consume 2768 ingest 2768	381 msg/s 5' 381msg/s 15' 381msg/s avg 366msg/s consume 2595 ingest 2595
2	244 msg/s 5' 244msg/s 15' 244msg/s avg 249msg/s consume 1854 ingest 1854	244 msg/s 5' 244msg/s 15' 244msg/s avg 249msg/s consume 1854 ingest 1854



当 Receiver 目前不可达时，所有数据为 N/A，地址颜色为红色

NewUserLogCube	READY	0.00 KB	0		ADMIN	2019-02-14 20:26:09 GMT+8	Action ▾	Action ▾	
Grid	SQL	JSON(Cube)	Notification	Storage	Planner	Streaming			
Replica Set ID: 0	cdh2.cloudera.com:9091								
	5'	15'	N/A		avg	consume	ingest		
	N/A	N/A			N/A	N/A	N/A		

3. 在任意时间可以选择 **Disable** Cube，**Disable** 会移除除了存储在 HBase 以外的全部数据，包括上传到 HDFS 路径上的数据文件，Cube 在本地 Segment 数据文件和 Checkpoint 文件，以及 Zookeeper 上的关于这个 Cube 的元数据。下一次 **Enable** 时，会根据条件选择起始 Offset：
 - 没有状态为 **READY** 的 HBase Segment，从 LATEST offset 开始摄入数据
 - 有状态为 **READY** 的 HBase Segment，将从最新的 HBase Segment 的 **stream_source_checkpoint** 开始消费。（其中 **stream_source_checkpoint** 是保存在 HBase 元数据的一段 JSON 字符串，记录了一个 segment 对每一个 Partition 消费的截止的 Offset）
[related code](#)

监控消费状况

点击某一个 Receiver，可以看到这个 Receiver 对这个 Cube 的消费统计数据

1. Cube 层级
 - Latest Event Time
最新一条摄入消息的时间分区列的时间戳
 - Latest Event Ingest Time
最新一条消息摄入时的时间戳
 - Segments
在这个 Receiver 上存在的全部 Segment
 - Partitions
在这个 Receiver 上进行摄取的 Partition



cdh2.cloudera.com:9091



2. Segment 层级统计



- segment name
segment 的名称
- segment state
segment 当前的状态
- fragment number
Fragment 文件的数量
- row number in memory
Memory Store 上存储维度组合的行数
- latest event latency
最后一条消息的延迟（摄入消息的时间和消息时间列的时间差值）
- segment create time
segment 创建的时间点

The screenshot shows the Apache Kylin web interface. A table lists segments, with one selected. A popup window titled "Segment Info" displays the following details:

Name	Status	Build Time	Owner
FullBuildUserLogCube	READY	2019-02-20 14:46:44 GM	ADMIN

cdh2.cloudera.com:9091

Summary statistics for the selected segment:

Latest Event Time 2019-02-20 08:09:18
Latest Event Ingest Time 2019-02-20 08:00:00
Segments 
Partitions 

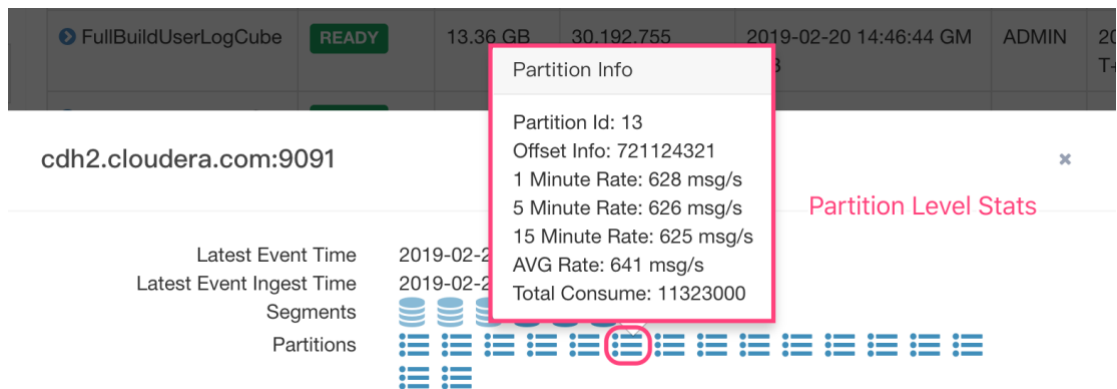
segment level stat

3. Partition 层级统计

- Partition ID
Topic Partition 的 ID

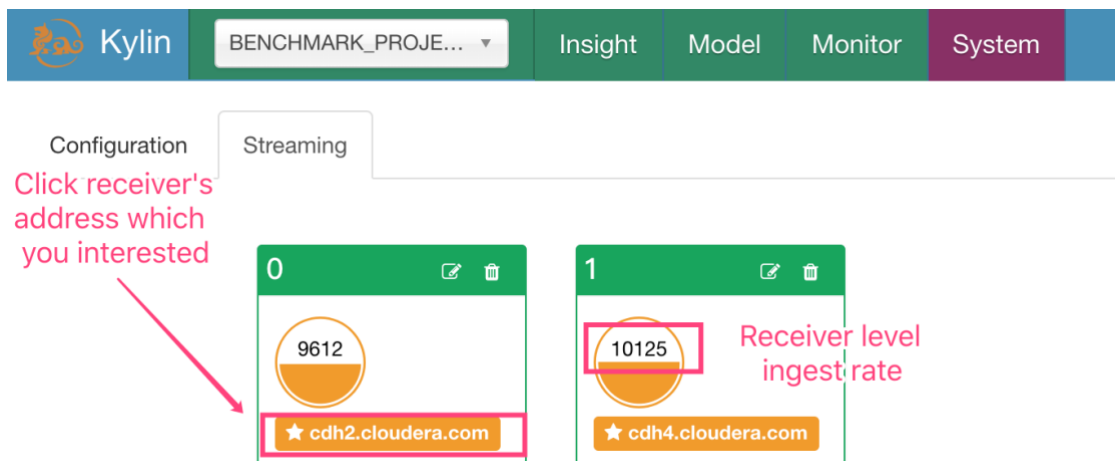


- Offset Info
当前 Partition 消费的最新 Offset
- 1/5/15 Minute Rate
最近 1/5/15 分钟的平均消费速率
- AVG Rate
从上次 assign 开始的平均消费速率
- Total Consume
从上次 assign 开始的的总消费消息数量



4. Receiver 层级统计

- 数字表示 Receiver 层级的消费速率
- 颜色表示 Receiver 的状态

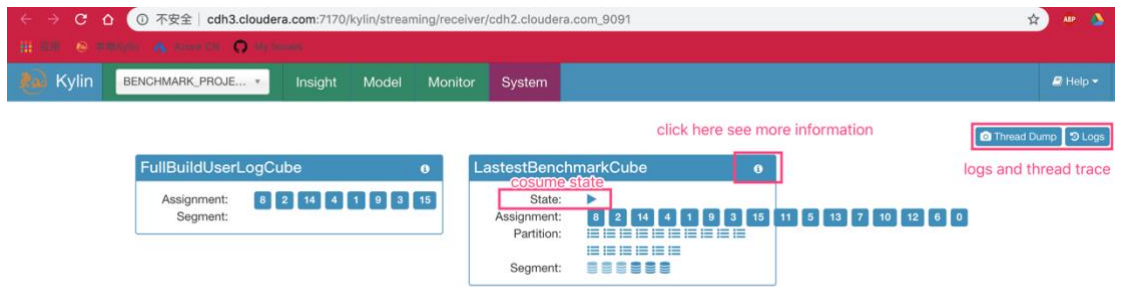


点击 Receiver 地址进入下一页面

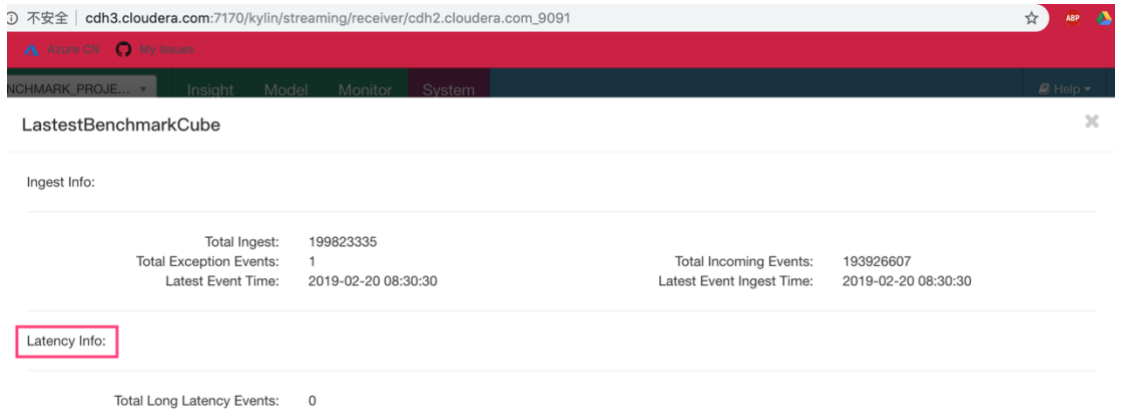
- 该 Receiver 所负责的全部 Cube
- 每个 Cube 的消费状态（正在消费或者暂停消费）



- Receiver 的日志和栈帧 dump



- Total Exception Events
解析错误消息数量
- Total Long Latency Events
因为消息延迟而被丢弃的消息数量





Real-time OLAP 的常见问题

- 在配置 Kafka 数据源时有一个“Lambda”勾选框，它是做什么的？
- 除了 base cuboid，我还能在 receiver 端构建其它 cuboid 吗？
- 我应该如何给我的 Receiver 集群进行扩容？Kafka Topic 增加 Partition 需要如何应对？
- 基准测试结果如何？查询时长大概是多少？单个 Receiver 的数据摄入速度大概是多少？
- 和 Kylin's NRT Streaming 相比较，哪个更加适合我的需求？
- 目前 Real-time OLAP 的主要限制有哪些？未来的开发计划有哪些？

Question-1

在配置 Kafka Streaming Table 时有一个 Lambda 勾选框，它是做什么的？

背景

在用户的一些场景下，实时部分数据需要保证数据查询延迟低，历史部分需要保证结果符合业务要求，因为 Kafka 消息需要经过 ETL 工具进行数据清洗筛选才是完全符合业务上要求的数据。基于这样的目的，在 Receiver 端摄入数据的同时，用户可以通过 ETL 作业从 Kafka 抽取数据写入 Hive 的同名表，一旦数据数据进入 Hive，再通过 Kylin 提供的 Build Cube API 来构建 Segment，查询结果的实时部分来自于 Receiver，历史部分来自于从 Hive 构建的 Segment，就能同时满足两个要求了。

步骤

1. 在 Hive 的 default namespace 下创建一张表 TABLE_1，要求 STREAMING_TABLE 的字段是下面创建 Streaming Table 的超集。
2. 在 Kylin 的 Web UI，创建 streaming table，主要表名和上面创建在 Hive 的表名相同，并且勾选 **Lambda** 选项，点击完成时 Kylin 会检查上面提到的要求，符合则会创建成功。
3. 进行 Model 和 Cube 设计过程，**Retention Policy** 可以设置为 **purge** 或者 **fullBuild**：若设置为 **fullBuild**，则可以将从 Hive 构建的 Segment 覆盖从 Receiver 构建的 Segment；若设置为 **purge**，则是将从 Hive 构建的 Segment 直接作为历史部分。



4. 在 Kylin 的 Web UI, 选择 Enable Cube ; 并且定时调度 ETL 作业和 Kylin 的 Build Rest API。

Question-2

除了 Base Cuboid, 我还能在 Receiver 端构建其它 Cuboid 吗 ?

背景

默认情况下, Receiver 端只构建 Base Cuboid, 但是从 Base Cuboid 查询扫描范围大故查询时长一般较长, 所以可以通过在 Receiver 端构建除了 Base Cuboid 以外的 Cuboid 来加速查询。

步骤

- 在 kylin.properties 设置 kylin.stream.build.additional.cuboids 为 true ; 或者也可以选择 在 Cube 级别设置
- 在 Cube Design 步骤选择增加所需的 **Mandatory Cuboids**
- Assign 后 Receiver 端会构建 Base Cuboid + Mandatory Cuboid, 查询将尽可能利用到 Mandatory Cuboid

Question-3

我应该如何给我的 Receiver 集群进行扩容 ? Kafka Topic 增加 Partition 需要如何应对 ?

1. Web UI

- 在 Cube 下点击 "Assignment", 点击 "Edit"

Last Build Time ↕	Owner ↕	Create Time ▼	Actions
2019-02-15 18:08:44 GMT+8	ADMIN	2019-02-15 16:35:24 GMT+8	Action ▼
2019-02-15 18:10:44 GMT+8	ADMIN	2019-02-14 20:26:09 GMT+8	Disable Assignment Merge Clone
	ADMIN	2019-02-09 21:56:35 GMT+8	
	ADMIN	2019-02-03 17:07:01 GMT+8	

- 编辑 Assignment, 然后点击 "Save" 提交 Reassign 操作, 将由 Coordinator 来根据传入的 Assignment 进行重分配操作

Cube Assignment

Replica Set ID	Partition	
0	1 x 2 x 5 x 14 x 13 x 4 x 7 x 1 x 10 x 9 x 12 x 3 x 6 x 15 x 0 x	-
1	8 x	-
		+

Save Close

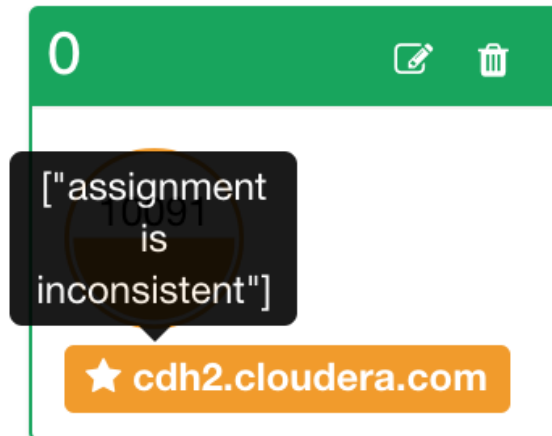
2. Rest API

使用 Reassign API 进行 Reassign 操作

注意

- 若 Reassign 过程某一些 Replica Set 被移除, Receiver 集群会出现一段时间的不一致状态。

Streaming





在 Reassign 过程被移除的 Replica Set 仍会出现在 Cube 的 Streaming Tab 下面，并且各项指标显示为 N/A，如下图 cdh2.cloudera.com:9091 就是被移除的 Receiver。

Name ↕	Status ↕	Cube Size ↕	Source Records ↕	Last Build Time ↕	Owner ↕	Create Time ↕	Actions	Admins
FullBuildUserLogCube	READY	0.00 KB	0		ADMIN	2019-02-18 16:00:31 GMT +8	Action ▾	Action ▾

Grid	SQL	JSON(Cube)	Notification	Storage	Planner	Streaming
cdh2.cloudera.com:9091						
Replica Set ID: 0	5' N/A	15' N/A	N/A	avg N/A	consume N/A	ingest 213969

点击查看 Segment 可以发现被分配的 Partition 为空，剩余的 Segment 为 **Immutable** 或者 **Remoted Persisted**

cdh2.cloudera.com:9091

Latest Event Time 2019-02-20 03:15:59
Latest Event Ingest Time 2019-02-20 03:15:59
Segments 
Partitions

Question-4

Real-time OLAP 和 Kylin's NRT Streaming 相比较，哪个更加适合我的需求？

Real-time OLAP

- 实时部分基于列存和倒排索引，延迟为毫秒级别
- 额外部署一个分布式计算和存储集群，学习/维护/管理压力大
- 不支持某些度量（不支持 TOPN，精确去重只支持整数类型）和 JOIN 维度表

NRT

- 数据延迟数分钟以上
- Kylin 仅仅作为客户端，借助于 MapReduce 稳定性，以较低的代价达到较好的稳定性
- 不存在度量不支持和无法 Join 维度表的问题



- 为了保证数据延迟低，需要构建很多个小的 MapReduce Job，增加了 Hadoop 系统压力

Question-5

Real-time OLAP 的基准测试结果如何？查询时长大概是多少？单个 Receiver 的数据摄入速度大概是多少？

- 最大摄入速度
58k msg per second （在一个 8vCPU 的虚拟机）
- 其余
[Brenchmark Result](#)

Question-6

目前 Real-time OLAP 的主要限制有哪些？未来的开发计划有哪些？

限制

- 不支持 Join 维度表
- 不支持部分度量，如字符串类型的精确去重

未来计划

- 基于 Kubernetes 部署和管理 Receiver
- 支持 Star Schema

Question-7

突然消费速率为 0，通过 jstack 查看 CUBENAME_channel 线程退出；查看 streaming_receiver.out，发现未捕获的 OOM 导致线程退出



解决办法：增加给 Receiver 进程分配的-xmx，或者调小 MemoryStore 的行数，然后重启 Receiver 进程。

```
Exception in thread "ClimateCubeFullbuild_channel" java.lang.OutOfMemoryError: GC
overhead limit exceeded

    at java.util.LinkedList.linkLast(LinkedList.java:142)
    at java.util.LinkedList.add(LinkedList.java:338)
    at
org.apache.kylin.stream.core.storage.columnar.ColumnarMemoryStorePersister.transform
ToColumnar(ColumnarMemoryStorePersister.java:195)
    at
org.apache.kylin.stream.core.storage.columnar.ColumnarMemoryStorePersister.persistDa
taFragment(ColumnarMemoryStorePersister.java:127)
    at
org.apache.kylin.stream.core.storage.columnar.ColumnarMemoryStorePersister.persist(Co
lumnarMemoryStorePersister.java:106)
    at
org.apache.kylin.stream.core.storage.columnar.ColumnarSegmentStore.persist(Columnar
SegmentStore.java:170)
    at
org.apache.kylin.stream.core.storage.columnar.ColumnarSegmentStore.checkpoint(Colum
narSegmentStore.java:148)
    at
org.apache.kylin.stream.core.storage.StreamingSegmentManager.checkpoint(StreamingS
egmentManager.java:575)
    at
org.apache.kylin.stream.core.storage.StreamingSegmentManager.addEvent(StreamingSeg
mentManager.java:194)
    at
org.apache.kylin.stream.core.consumer.StreamingConsumerChannel.run(StreamingConsu
merChannel.java:103)
    at java.lang.Thread.run(Thread.java:748)
```

注：如您在使用 Real-time OLAP 功能的过程中遇到问题，请发送问题至 Apache Kylin 邮件列表：user@kylin.apache.org。（该邮件列表需要订阅才能使用。如未订阅该邮件列表，请先发送邮件至 user-subscribe@kylin.apache.org，并回复确认完成订阅）。