Apache Druid 简介

Druid 是什么

Druid 是一个分布式的、支持实时多维 OLAP 分析的数据处理系统。它既支持高速的数据实时摄入处理，也支持实时且灵活的多维数据分析查询。因此 Druid 最常用的场景就是大数据背景下、灵活快速的多维 OLAP 分析。 另外，Druid 还有一个关键的特点：它支持根据时间戳对数据进行预聚合摄入和聚合分析，因此也有用户经常在有时序数据处理分析的场景中用到它。

为什么用 Druid

特性

1. 亚秒响应的交互式查询，支持较高并发。
2. 支持实时导入，导入即可被查询，支持高并发导入。
3. 采用分布式 shared-nothing 的架构，可以扩展到PB级。
4. 支持聚合函数，count 和 sum，以及使用 javascript 实现自定义 UDF。
5. 支持复杂的 Aggregator，近似查询的 Aggregator 例如 HyperLoglog 以及 Yahoo 开源的 DataSketches。
6. 支持Groupby，Select，Search查询。
7. 不支持大表之间的Join，但其 lookup 功能满足和维度表的 Join。

这里最关键是前两条。

不支持

1. 不支持精确去重
2. 不支持 Join（只能进行 semi-join）
3. 不支持根据主键的单条记录更新

如果不能接受这几点，则可以考虑放弃使用 Druid 了。

为什么快

数据的预聚合

Druid 可以按照给定的时间粒度和所有维度列，进行最细粒度的指标聚合运算，并加以保存为原始数据。

列式存储

对部分列进行查询时可以显著提高效率。

Bitmap 索引

利用位图对所有维度列构建索引，可以快速定位数据行。

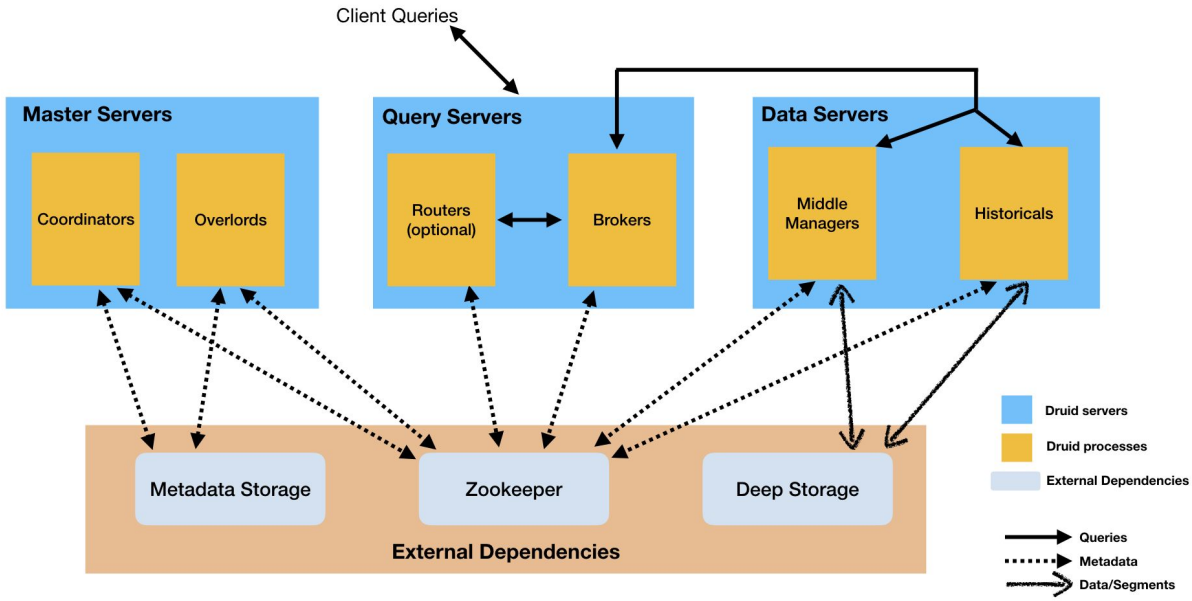
mmap

通过内存映射文件的方式加快对于 Segment 的访问。

查询结果的中间缓存

支持对于查询级别和 segment 级别的缓存。

架构



内部组件

Master

Runs Coordinator and Overlord processes, manages data availability and ingestion.

- Coordinator：负责集群 Segment 的管理和发布，并确保 Segment 在 Historical 集群中的负载均衡。
- Overlord：负责接受任务、协调任务的分配、创建任务锁以及收集、返回任务运行状态给客户端；在 Coordinator 节点配置 asOverlord，让 Coordinator 具备 Overlord 功能，这样可以减少一个组件的部署和运维。

Query

Runs Broker and optional Router processes, handles queries from external clients.

- Router 节点：可选节点，在 Broker 集群之上的 API 网关，有了 Router 节点 Broker 不再是单点服务了，提高了并发查询的能力。

- Broker：负责从客户端接收查询请求，并将查询请求转发给 Historical 节点和 MiddleManager 节点。Broker 节点需要感知 Segment 信息在集群上的分布。

Data

Runs Historical and MiddleManager processes, executes ingestion workloads and stores all queryable data.

- Middle Manager：主要是负责数据索引，生成索引文件，并把索引文件先发布到一个共享的存储系统里，我们选择了大家普遍采用的 HDFS 系统；
- Historical：主要负责加载索引文件，同时提供历史数据的查询服务；

外部依赖

- Metastore Storage：用于存储 Druid 的各种元数据信息，属于 Druid 的外部依赖组件，生产环境中可用 MySQL。
- Zookeeper：分布式协调服务，用于节点管理和事件监控。
- Deep Storage：用于存储 Segment 文件供 Historical 节点下载。Deep Storage 不属于 Druid 内部组件，用户可根据系统规模来自定义配置。单节点可用本地磁盘，分布式可用 HDFS。

存储

Druid 数据存储在“datasources”中，类似于传统RDBMS中的表。每个 datasource 按时间划分，并可选择进一步按其他属性划分。每个时间范围称为“chunk”（例如，如果您的 datasource 按天分区，则为一天）。在 chunk 内，数据被划分为一个或多个“segments”。每个 segment 都是单个文件，通常包含多达几百万行数据。

数据结构

DataSource

DataSource 是一个逻辑概念，表示 Druid 的基本数据结构，可以理解为关系型数据库中的表。它包含时间、维度和指标三列。

- 时间（TimeStamp）：表明每行数据的时间值，默认使用 UTC 时间格式且精确到毫秒级别。这个列是数据聚合与范围查询的重要维度。
- 维度（Dimension）：标识数据行的各个类别信息。
- 指标（Metric）：用于聚合计算的列，这些指标列通常是一些数字，主要操作包括 Count、Sum 和 Mean 等。

Segment

Segment 是 Druid 中数据的实际物理存储格式，Druid 正是通过 Segment 实现了对数据的横纵向切割（Slice and Dice）操作：

- **横向**：通过参数 segmentGranularity 的设置，将不同时间范围内的数据存储在不同的 Segment 数据块中。这样在指定时间范围内查询时，可以不用扫全表。

- **纵向**：即列式存储，对每个列进行切分并压缩，且利用 Bitmap 构建索引从而优化数据访问。

实际存储中，它由下面 4 个部分组成：

1. **Datasource name**：对应的 Datasource；
2. **Time interval**：该Segment段的时间间隔，使用ISO-8601格式，表示存的是那个时间段内的聚合数据；
3. **Version number**：版本号，用于区分多次加载同一数据对应的Segment；
4. **Partition number**：分区编号，在每个时间间隔内，根据数据量的大小，一个 Segment 内部可能会有多个partition。通过对数据分区，可以获得并行性。比如从 Kafka 摄入数据，则分区数等于 Kafka 中对应分区数。

因此 Segment 的命名为：

```
<dataSource>_<intervalStart>_<intervalEnd>_<version>_<partitionNum>
```

由于篇幅有限，后续文章将会陆续介绍环境部署、各个组件的功能、数据的摄入、数据的查询、冷热分离、运维监控、与 Superset 集成等。