Table of Contents

Druid介绍

Druid的概念

Druid本来是一个开源的数据存储,为OLAP的事件查询而设计， 有兴趣的朋友可以关注下广东数果对于druid的翻译。 我们目前基于lucene，剔除lucene的词向量、行正向等非必要的信息，并做了多处优化，重新设计了一套Druid的索引，在保证其原有的查询功能的前提下，主要改动涉及以下方面：

- 重新设计了索引层，基于原始数据查询，可随时定制查询指标，不再需要提前指定指标
- 增强了维度支持，目前测试可支持上万维度
- 支持了多种数据类型（int/long/String/date/float/text等，后续会支持地理位置信息）
- 实现了按需加载，用户可以配置LRU按需加载策略，解决了原有的Druid在行数达到一定的情况下加载慢，甚至不可查等问题
- 支持了更多的查询，例如留存查询、漏斗查询及用户分群等
- 支持了更多查询相关的功能，如数字分组、日期分组查询
- 优化了查询聚合，避免虚函数的性能损耗
- 支持分词

为什么选择Druid框架

我们一直希望做一款针对万亿、上千维度的数据，做到能够实时查询，满足自由定义查询指标的产品。预研了Hbase类似的KV的存储结构、基于Dremal的列式存储方案， 甚至MPP的并行计算的方案，都无法满足我们。

14-15年期间，针对大数据的各种预聚合计算的方案进入开源的白热化阶段，我们发现Druid针对数十维度的场景下使用非常完美。 经过对其源码的详细深入和社区的交流，我们发现其框架扩展性、查询引擎设计的非常好，很多性能细节都考虑在内。例如：

- 堆外内存的复用，避免GC问题；
- 根据查询数据的粒度，以Sequence的方式构建小批量的数据，内存利用率更高；
- 查询有bySegment级别的缓存，可以做到大范围固定模式的查询；
- 多种query，最大化提升查询性能，例如topN、timeSeries等查询等等。

另外框架可灵活的扩展，也是我们考虑的一个很重要的元素，在我们重写了索引后，Druid社区针对高基数维度的查询上线了groupByV2，我们很快就完成了groupByV2也可见其框架非常灵活。

在我们看来，Druid的查询引擎很强大，但是索引层还是针对OLAP查询的场景，这就是我们选择Druid框架进行索引扩展的根本原因。 另外其充分考虑分布式的稳定性，HA策略，针对不同的机器设备情况和应用场景，灵活的配置最大化利用硬件性能来满足场景需要也是我们所看重的。

Druid服务的结构

虽然我们对Druid索引的重构，但是Druid的框架灵活性使得我们的改造并未影响其接口和服务层。所以，最终的产品形态和Druid的框架结构一致。 Druid的服务节点由五种类型组成：

- Overlord：用于管理实时任务的调度，新版本有支持Supervisor，可以监听实时任务的状态，挂掉的task可以自动拉起，需要zookeeper选举leader；
- Coordinator：协调数据段的管理，所有的数据段管理都由其统一调配，，需要zookeeper选举leader；
- MiddleManager：负责实时task任务的调起；
- Broker：分发查询任务及合并查询结果；
- History：查询每一个具体的历史数据段。

Druid的服务介绍不在此多介绍了，有兴趣可以了解下广东数果对于Druid设计的翻译。

Druid配置参数说明

1.通用配置

> 安装目录/conf/druid/_common/common.runtime.properties文件配置内容如下：

```
# 日志模式，logging表示单纯的文件日志，ingest表示上报消费的数据情况，用于数据质量监控
# 若使用ingest，必须指定druid.emitter.ingest.recipientBaseUrl，用于接收数据
druid.emitter=logging
#druid.emitter=composing
#druid.emitter.composing.emitters=["logging", "ingest"]
#druid.emitter.ingest.recipientBaseUrl=http://192.168.0.218:8080/api/v1/datapoints

#日志级别，默认info即可
druid.emitter.logging.logLevel=info
#是否开启启动时的包加载情况日志，默认即可
druid.startup.logging.logProperties=true

#Druid扩展依赖的目录，对应druid的`安装目录`/extensions
druid.extensions.directory=/opt/apps/druidio_sugo/extensions
druid.extensions.hadoopDependenciesDir=/opt/apps/druidio_sugo/hadoop-dependencies

#Druid可以定义不能呢个功能模块，默认使用postgresql数据库存储元数据，
#如果更换mysql，可将`postgresql-metadata-storage`更换为`mysql-metadata-storage`
druid.extensions.loadList=["druid-kafka-eight", "postgresql-metadata-storage", "druid
-hdfs-storage", "druid-lucene-extensions"]

#实时task运行完后log存储类型，如果需要hdfs统一保存可设置为`hdfs`
druid.indexer.logs.type=file
#实时task运行完后log上传文件位置
druid.indexer.logs.directory=/data1/druid/indexing-logs

#元数据保存数据库，如果是mysql数据库，`postgresql`改为`mysql`，并修改connectURI。
#但需要将druid.extensions.loadList中的`postgresql-metadata-storage`更换为`mysql-metadata
-storage`
druid.metadata.storage.type=postgresql
druid.metadata.storage.connector.connectURI=jdbc:postgresql://192.168.0.210:5432/drui
d_test
druid.metadata.storage.connector.password=druid
druid.metadata.storage.connector.user=druid

#设置管理服务的名称，默认即可
druid.selectors.coordinator.serviceName=druid/coordinator
druid.selectors.indexing.serviceName=druid/overlord

#设置需要监控的内容，druid包含多种监控，可根据需要配置
druid.monitoring.monitors=["com.metamx.metrics.JvmMonitor"]
```

```
# Druid的实时task运行完后，数据备份位置，可以设置为hdfs，对应hdfs路径
druid.storage.type=local
druid.storage.storageDirectory=/data1/druid/storage

# Druid依赖与zookeeper同步信息，需要设置zookeeper路径，
# druid.zk.paths.base表示同步数据信息的路径，
#druid.discovery.curator.path主要用于服务发现
druid.zk.paths.base=/druid_test
druid.discovery.curator.path=/druid_test/discovery
druid.zk.service.host=127.0.0.1:2181

#针对用户分群的场景，如果没有该场景可以忽略
druid.lookup.lru.cache.maxEntriesSize=50
#druid.lookup.lru.cache.expireAfterWrite=300
#druid.lookup.lru.cache.expireAfterAccess=300
```

2.log文件配置

> Druid相关的log文件位于目录：`安装目录`/conf/druid/_common，文件分别为：log4j2-default.xml和log4j2.xml

log4j2-default.xml为实时task任务的日志配置，与官方提供的log4j一致，未做调整。
log4j2.xml为Druid每个服务所使用的日志配置。

之所以单独配置Druid每个服务的日志，是因为官方并未给出具体的配置，随着Druid服务的运行日志越来越大， 并且每分钟metric的日志和正常的log混合在一起，难以区分。log4j2.xml的文件内容如下：

```
<Configuration status="WARN">
    <Properties>
        <Property name="LOG_DIR">${sys:log.file.path}</Property>
    <Property name="LOG_NAME">${sys:log.file.type}</Property>
    </Properties>
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{ISO8601} %p [%t] %c - %m%n"/>
        </Console>
        <RollingRandomAccessFile name="DruidLog" fileName="${LOG_DIR}/${LOG_NAME}.log
"  filePattern="${LOG_DIR}/${LOG_NAME}.%d{yyyy-MM-dd}.log">
            <PatternLayout pattern="%date{yyyy-MM-dd HH:mm:ss.SSS} %level [%thread][%
file:%line] - %msg%n" />
            <Policies>
                <TimeBasedTriggeringPolicy interval="1" modulate="true" />
            </Policies>
            <!--日志保留7天-->
            <DefaultRolloverStrategy>
                <Delete basePath="${LOG_DIR}" maxDepth="1">
                    <IfFileName glob="${LOG_NAME}.*.log" />
                    <IfLastModified age="7d" />
                </Delete>
```

```
            </DefaultRolloverStrategy>
        </RollingRandomAccessFile>
            <RollingRandomAccessFile name="MetricLog" fileName="${LOG_DIR}/${LOG_NAME
}-metric.log" filePattern="${LOG_DIR}/${LOG_NAME}-metric.%d{yyyy-MM-dd}.log">
            <PatternLayout pattern="%date{yyyy-MM-dd HH:mm:ss.SSS} %level [%thread][%
file:%line] - %msg%n" />
            <Policies>
                <TimeBasedTriggeringPolicy interval="1" modulate="true" />
            </Policies>
            <!--日志保留7天-->
            <DefaultRolloverStrategy>
                <Delete basePath="${LOG_DIR}" maxDepth="1">
                    <IfFileName glob="${LOG_NAME}-metric.*.log" />
                    <IfLastModified age="7d" />
                </Delete>
            </DefaultRolloverStrategy>
        </RollingRandomAccessFile>
    </Appenders>
    <Loggers>
        <!--metric相关的日志另外输出-->
        <logger name="com.metamx.emitter.core.LoggingEmitter" level="info" additivity
="false">
                <appender-ref ref="MetricLog" />
        </logger>
        <Root level="info">
            <AppenderRef ref="DruidLog"/>
        </Root>
    </Loggers>
</Configuration>
```

Overlord配置说明

Overlord的主要功能是管理task任务，所有task都以http post的方式发送给Overlord。Overlord有本地(local)和集群(remote)两种模式，当使用本地模式时，Overlord还承担这进程调度的职责，Overlord会在本地启动进程，需要注意的是需要Middlemanager相关的配置，才能启动本地模式。生产环境需要使用集群模式，在集群模式中，Overlord只会分配task给相应的Middlemanager。

Overlord的配置文件包括两部分内容：jvm.config和runtime.properties。

JVM配置

> 安装目录/conf/druid/overlord/jvm.config配置内容如下：

```
-server
-Xms1g
-Xmx1g
-Duser.timezone=UTC
-Dfile.encoding=UTF-8
-Djava.io.tmpdir=var/tmp
-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
-Dlog.file.path=var/logs
-Dlog.file.type=overlord
-Ddruid.dir=/opt/apps/druidio_sugo
```

该节点不会消耗过多内存，一般情况2-4G内存足够。建议统一用UTC时区，java.io.tmpdir用于指定运行时临时文件目录，请确保有该目录的权限和存储足够存放服务的log。
需要注意的是由于我们使用了自定义的log4j2，在此需要指定log存储位置和节点类型，分别对应 `-Dlog.file.path` 和 `-Dlog.file.type` ， 另外需要指定项目的Druid的安装目录 `-Ddruid.dir` 。

Overlord节点属性配置

> 安装目录/conf/druid/overlord/runtime.properties配置内容如下：

```
# Overlord节点的服务名，一个集群多个为同一个服务名
druid.service=druid/overlord

# 服务启动的机器ip或域名，此处需要配置正确，避免在zookeeper引起错乱
druid.host=192.168.0.220

# Overlord服务的端口，默认即可
druid.port=8090

# 在Overlord启动一段时间后，才开始管理task的队列。进程刚启动时，需要一段时间来协调网络。
druid.indexer.queue.startDelay=PT30S

# Druid有两种模式：local和remote，分布式环境需要设置为remote
druid.indexer.runner.type=remote

# Druid的task信息存储有两种模式：local和metadata，local为内存存储，metadata为数据库存储，生产环境使用metadata
druid.indexer.storage.type=metadata
```

Coordinator配置说明

Coordinator负责管理所有的数据段，数据段的加载、删除、副本的管理都由其统一调度协调。事实上，Coordinator在数据段的管理方面并未直接与history通信，而是将history需要加载的段写到zookeeper相应history加载队列的路径，history一旦发现，则会开始加载提供查询服务。

Coordinator的配置文件包括两部分内容：jvm.config和runtime.properties。

JVM配置

> 安装目录 /conf/druid/coordinator/jvm.config配置内容如下：

```
-server
-Xms2g
-Xmx2g
-Duser.timezone=UTC
-Dfile.encoding=UTF-8
-Djava.io.tmpdir=var/tmp
-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
-Dlog.file.path=var/logs
-Dlog.file.type=coordinator
-Ddruid.dir=/opt/apps/druidio_sugo
```

该节点不会消耗过多内存，可根据数据段的数量适当调整内存，一般情况4-8G内存足够。 建议统一用UTC时区，java.io.tmpdir用于指定运行时临时文件目录，请确保有该目录的权限和存储足够存放服务的log。
需要注意的是由于我们使用了自定义的log4j2，在此需要指定log存储位置和节点类型，分别对应 `-Dlog.file.path` 和 `-Dlog.file.type` ， 另外需要指定项目的Druid的安装目录 `-Ddruid.dir` 。

Overlord节点属性配置

> 安装目录 /conf/druid/coordinator/runtime.properties配置内容如下：

```
# 服务名，一个集群多个为同一个服务名
druid.service=druid/coordinator

# 服务启动的机器ip或域名，此处需要配置正确，避免在zookeeper引起错乱
druid.host=192.168.0.220

# 服务的端口，默认即可
druid.port=8081

# Druid本身存在一个问题，当history节点加载大量数据段时，需要花费较多的时间，
# 若此处配置的时间较短，history节点未能加载完数据段，coordinator则会重新分配
# 这些数据段给其他history节点，这点非常不友好。广东数果增加了数据段动态加载的功能，
# 可以避免这种情况
druid.coordinator.startDelay=PT30S

# coordinator检测数据段有限性周期
druid.coordinator.period=PT30S
```

MiddleManager配置说明

MiddleManager相对来说非常轻量级，只是单纯的以JVM进程的方式调起需要执行的task。 但具体的task进程，特别是写索引的进程(我们一般可以叫做实时task)，不仅要写索引还要提供查询服务，需要消耗一定系统资源。 task运行的jvm参数可以通过修改 `druid.indexer.runner.javaOpts` 调整。 建议实时task一般写的数据段数据量控制在1-2千万左右，这是一个平衡值，过多单个数据段查询慢，过少会导致数据段过多，都会在一定程度上影响查询性能。 另外，每个task的内存2G，堆外内存2G，一台机器的内存 > task数*(2G+2G)。

MiddleManager的配置文件包括两部分内容：jvm.config和runtime.properties。

JVM配置

> 安装目录 /conf/druid/MiddleManager/jvm.config配置内容如下：

```
-server
-Xms64m
-Xmx64m
-Duser.timezone=UTC
-Dfile.encoding=UTF-8
-Djava.io.tmpdir=/home/druid/tmp
-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
-Dlog.file.path=/data2/druidio/logs/jvm
-Dlog.file.type=middleManager
-Ddruid.dir=/opt/apps/druidio_sugo
-Dlog.configurationFile=/opt/apps/druidio_sugo/conf/druid/_common/log4j2-default.xml
```

该节点不会消耗过多内存，一般情况64~128M内存足够。建议统一用UTC时区，java.io.tmpdir用于指定运行时临时文件目录，请确保有该目录的权限和存储足够存放服务的log。

需要注意的是由于我们使用了自定义的log4j2，在此需要指定log存储位置和节点类型，分别对应 `-Dlog.file.path` 和 `-Dlog.file.type` ， 另外需要指定项目的Druid的安装目录 `-Ddruid.dir` 。

为了区分task的配置需要指定 `-Dlog.configurationFile` ，否则task日志会出现问题。

MiddleManager节点属性配置

> 安装目录 /conf/druid/coordinator/runtime.properties配置内容如下：

```
# 服务名，一个集群多个为同一个服务名
druid.service=druid/middleManager

# 服务启动的机器ip或域名，此处需要配置正确，避免在zookeeper引起错乱
druid.host=192.168.0.212

# 服务的端口，默认即可
druid.port=8091

# MiddleManager最大启动的task进程数
druid.worker.capacity=12

# MiddleManager查询并发线程数和每个线程需要的堆外内存，配置时确保task的堆外内存 -XX:MaxDirectM
emorySize > numThreads*sizeBytes
druid.processing.numThreads=3
druid.processing.buffer.sizeBytes=524288000

# 每个task进程的jvm配置，MaxDirectMemorySize表示堆外内存
druid.indexer.runner.javaOpts=-server -Xmx2g -XX:MaxDirectMemorySize=2g -Duser.timezo
ne=UTC -Dfile.encoding=UTF-8 -Djava.util.logging.manager=org.apache.logging.log4j.jul
.LogManager -Djava.io.tmpdir=var/tmp

# task运行时临时目录
druid.indexer.task.baseTaskDir=var/druid/task
# hadoop任务所需要的hadoop版本，指定前确保版本所对应的目录已经存在于druid.extensions.hadoopDe
pendenciesDir
druid.indexer.task.defaultHadoopCoordinates=["org.apache.hadoop:hadoop-client:2.7.2"]
# hadoop task临时目录
druid.indexer.task.hadoopWorkingPath=var/druid/hadoop-tmp

# 用于处理http请求的线程数
druid.server.http.numThreads=25
```

Broker配置说明

Broker为Druid的查询节点，通过zookeeper监听得到每个数据段所在的history节点或者实时task，当有查询请求时，Broker节点会将查询分发到具体的history和实时task查询，然后汇总查询返回的结果。

Broker的配置文件包括两部分内容：jvm.config和runtime.properties。

JVM配置

> 安装目录 /conf/druid/Broker/jvm.config配置内容如下：

```
-server
-Xms20g
-Xmx20g
-XX:MaxDirectMemorySize=20g
-Duser.timezone=UTC
-Dfile.encoding=UTF-8
-Djava.io.tmpdir=var/tmp
-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
-Dlog.file.path=var/logs
-Dlog.file.type=broker
-Ddruid.dir=/opt/apps/druidio_sugo
```

建议统一用UTC时区，java.io.tmpdir用于指定运行时临时文件目录，请确保有该目录的权限和存储足够存放服务的log。

需要注意的是由于我们使用了自定义的log4j2，在此需要指定log存储位置和节点类型，分别对应 `-Dlog.file.path` 和 `-Dlog.file.type` ， 另外需要指定项目的Druid的安装目录 `-Ddruid.dir` 。

Broker节点属性配置

> 安装目录 /conf/druid/broker/runtime.properties配置内容如下：

```
# 服务名，一个集群多个为同一个服务名
druid.service=druid/broker
# 服务启动的机器ip或域名，此处需要配置正确，避免在zookeeper引起错乱
druid.host=192.168.0.212
# 服务的端口，默认即可
druid.port=8082

#--------------------
# 缓存相关的配置
#--------------------
# 不需要缓存的查询，默认"groupBy", "select"
druid.broker.cache.unCacheable=["groupBy", "select"]
# 缓存类型，支持local、memcached和hybrid三种模式，
# 其中local表示堆内存，实际为LinkedHashMap
# hybrid为混合缓存，混合缓存支持两级缓存，一般情况local作为一级缓存，memcached作为二级缓存，需要注意的是二级缓存大于一级缓存
druid.cache.type=local
# 设置缓存大小，byte为单位
druid.cache.sizeInBytes=200000000
# useCache和populateCache需要配合使用，一个为是否读缓存，一个为是否写入缓存。
# 在使用的时候，一般情况两者要么都为false，要么都为true
# 在大集群的时候这两项需要false，尽量让history合并结果，然后传到broker
druid.broker.cache.useCache=false
druid.broker.cache.populateCache=false

# Broker以http方式发送查询请求，此处为http线程池
# 可以根据机器资源情况调整，一般30~50
druid.broker.http.numConnections=30

# 查询并发线程数和每个线程需要的堆外内存
# 一般情况看机器资源情况，druid.processing.buffer.sizeBytes大小在500M-2G，druid.processing.numThreads为cpu核心数-1。
# 在测试环境中，数据段小，查询简单druid.processing.buffer.sizeBytes设置为100-200M也没有问题，生产环境建议设置500M以上。
# 配置时确保task的堆外内存 -XX:MaxDirectMemorySize > (numThreads+numMergeBuffers)*sizeBytes
druid.processing.numThreads=11
druid.processing.buffer.sizeBytes=1073741824
# 当使用groupByV2时，需要设置该参数
#druid.processing.numMergeBuffers=3

# Broker处理查询请求线程池 一般情况20~50
druid.server.http.numThreads=25
```

> 关于内存的设置建议
> Broker节点需要合并所有段返回的结果，在段多的情况，消耗的内存相应也会多，另外查询场景也会影响内存。
> Druid的groupby查询需要堆外内存，topN查询大部分情况需要同时消耗堆内存和堆外内存，每个数据段返回的结果在druid中都是堆内存存储，而topN计算使用的是堆外内存，除此之外所有的查询都使用堆内存。
> 当查询大部分集中在使用groupby的时候，可以通过druid.processing.numThreads和druid.processing.buffer.sizeBytes控制查询内存消耗。
> 而当查询以堆内存的消耗为主或者各种查询都有时，需要注意适当调大堆内存，特别是段多的时候，需要合并所有段的返回结果。堆内存不建议设置过大，避免带来GC慢，消耗cpu等问题，一般建议堆内存20~30G。另外，Broker节点GC策略最好使用CMS。

History配置说明

History下载实时task上传的文件，并加载到内存提供查询服务。

History的配置文件包括两部分内容：jvm.config和runtime.properties。

JVM配置

> 安装目录 /conf/druid/Broker/jvm.config配置内容如下：

```
-server
-Xms4g
-Xmx4g
-XX:MaxDirectMemorySize=24g
-Duser.timezone=UTC
-Dfile.encoding=UTF-8
-Djava.io.tmpdir=var/tmp
-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
-Dlog.file.path=var/logs
-Dlog.file.type=historical
-Dcom.sun.management.jmxremote.port=2627
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
-Djava.rmi.server.hostname=192.168.0.212
-Ddruid.dir=/opt/apps/druidio_sugo
```

建议统一用UTC时区，java.io.tmpdir用于指定运行时临时文件目录，请确保有该目录的权限和存储足够存放服务的log。

需要注意的是由于我们使用了自定义的log4j2，在此需要指定log存储位置和节点类型，分别对应 `-Dlog.file.path` 和 `-Dlog.file.type` ， 另外需要指定项目的Druid的安装目录 `-Ddruid.dir` 。

History节点属性配置

> 安装目录 /conf/druid/history/runtime.properties配置内容如下：

```
# 服务名，一个集群多个为同一个服务名
druid.service=druid/historical
# 服务启动的机器ip或域名，此处需要配置正确，避免在zookeeper引起错乱
druid.host=192.168.0.212
# 服务的端口，默认即可
druid.port=8083

# 必须指定数据段的类型为lucene，否则无法使用索引
druid.historical.segment.type=lucene

# 查询并发线程数和每个线程需要的堆外内存
# 一般情况看机器资源情况，druid.processing.buffer.sizeBytes大小在500M-2G，druid.processing.numThreads为cpu核心数-1。
# 在测试环境中，数据段小，查询简单druid.processing.buffer.sizeBytes设置为100-200M也没有问题，生产环境建议设置500M以上。
# 配置时确保task的堆外内存 -XX:MaxDirectMemorySize > numThreads*sizeBytes
druid.processing.numThreads=11
druid.processing.buffer.sizeBytes=1073741824

# history节点希望分配的存储，这个值并不会强制限制history的存储大小，为coordinator在分配segment时提供参考。
druid.server.maxSize=300000000000
# history下载的数据段存储的位置，如果有多块硬盘，可以配置多个path，便于充分利用磁盘性能
druid.segmentCache.locations=[{"path":"var/druid/segment-cache","maxSize":300000000000}]

# 处理查询请求线程池 一般情况20~50
druid.server.http.numThreads=25
```

> 关于内存的设置建议
> history节点利用堆外内存存储查询中间结果以及数据段的内存映射，堆外内存越大，就越能降低使用磁盘分页的可能性，查询的速度就越快。
> 对外内存可以通过druid.processing.numThreads和druid.processing.buffer.sizeBytes控制查询内存消耗。而堆内存一般建议 250M*(processing.numThreads)。
> 另外，Broker节点GC策略最好使用CMS。

Druid安装建议

计算机的三个核心资源为cpu、内存和存储。根据这三方面，针对Druid各种服务做以下建议：

存储

Druid的History节点相对来说比较占用存储，Middlemanager只需要少量存储，存储数据的临时写入，一旦实时task任务完成后，数据将会上传到分布式文件系统，移交查询给History后，本地临时写入的文件将会自动清除。其他节点基本不需要存储，对存储也没有要求，此处只介绍History节点和Middlemanager的存储建议。

为了提升实时task数据写入和查询速度，可以有可能尽量使用SSD盘，加快数据写入速度。另外，我们扩展了数据写入，可以指定多块盘同时写数据，也可以通过加硬盘和增加task数提升数据写入速度。但是，实时task并不是特别消耗存储，所以Middlemanager节点不需要太大的存储。

History需要比较大的存储，所有的索引数据History都需要从本地加载，另外如果机器资源充足，还是有必要做replicat的，可以提升查询并发。如果有可能使用SSD盘，能够大幅度提升查询性能。建议使用多块硬盘，也能提升数据查询性能。

建议History和Middlemanager节点使用独立的系统盘，避免因为频繁读盘，导致系统运行慢，而引起恶性循环。

内存

Overlord和Coordinator为管理节点，不需要消耗太多内存，一般2~4G足够，也不需要使用堆外内存。

Middlemanager节点，只是单纯的调度进程，使用的资源非常少，一般64~128M足够。但是需要注意Middlemanager中配置的实时task的内存，实时task需要参看自身所需要写入的数据段数，一般情况设置堆内存2~3G，堆外2~3G，这些要根据具体情况调。

Broker节点合并History的查询结果需要较多的堆内存，建议堆内存设置到20~30G。同时Broker处理groupby查询海上花需要消耗堆外内存，一般情况建议druid.processing.buffer.sizeBytes设置到500M以上，然后结合系统资源情况，设置druid.processing.numThreads，充分利用系统的资源。druid.processing.numThreads也不用设置过大，一般建议(cpu数-1)。

History节点需要堆外内存存储中间结果和对索引数据做内存映射，一般来说，堆外内存越大，数据段基本都可以基于内存映射，不需要硬盘分页，查询速度越快。相对来说，堆内存不是特别消耗，一般建议堆内存设置为：250mb * (processing.numThreads) 。

CPU

Broker、History和Middlemanager节点所在的机器，尽量使用较好的cpu，建议cpu核心数在12以上。

服务器搭配建议

Overlord和Coordinator用于管理集群的元数据，属于辅助功能，可以部署在同一台服务器，所需的服务器配置不用很高，建议：

- 4 vCPUs
- 12 GB RAM
- 100 GB HDD

History和Middlemanager可以部署在一台服务器，他们都是数据服务节点，这两个服务对cpu、内存和硬盘都有要求，如果使用SSD会更好，使用一般硬盘的话，建议多加几块盘，一般情况建议：

- 12 vCPUs
- 64 GB RAM
- 12*500G HDD

Broker主要合并查询结果，它的所有操作基本上都是基于内存的，对cpu和内存要求高，可以与查询的UI层部署在同一台机器，建议：

- 12 vCPUs
- 64 GB RAM
- 100 GB HDD

另外，Druid对zookeeper的依赖比较重，建议使用单独的zookeeper集群。

Supervisor例子

Druid在0.9.0之后提供了Supervisor的功能，对于挂掉的task可以重新拉起，目前其只实现了kafka的对接，我们也扩展了与metaq的对接。目前，我们支持kafka、metaq消息的不丢不重，且数据延迟也能正常接入。一个Supervisor的定义：

```
{
  "type": "lucene_supervisor",                          #`lucene_supervisor`为我们扩展的su
pervisor类型，
  "dataSchema": {
    "dataSource": "test",                               #数据源名
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "url",                                #数据解析方式，url为我们扩展的解析，
支持url解析，另外还支持json/csv/tsv/正则等
        "timestampSpec": {
          "column": "EventDateTime",                    #基于时序的存储，需要指定时间字段
          "format": "millis"                            #format为日期的格式，其中millis=毫秒
数，posix=秒数，另外我们扩展了此处，可以指定所需的format和时区
        },
        "dimensionsSpec": {
          "dimensions": [                               #指定需要索引的字段，我们扩展了此处，
支持多种数据类型，且可以支持多值
            {
              "name": "IP",
              "type": "string"
            },
            {
              "name": "Province",
              "type": "string"
            },
            {
              "name": "Value",
              "type": "float"
            },
            {
              "name": "age",
              "type": "int"
            },
            {
              "name": "length",
              "type": "long"
            },
            {
              "name": "time",
              "type": "date",
              "format": "yyyy-MM-dd HH:mm",
```

```
              "timeZone": "+08:00"
            }
          ],
          "dimensionExclusions": [],
          "spatialDimensions": []
        }
      }
    },
    "metricsSpec": [],
    "granularitySpec": {
      "type": "uniform",
      "segmentGranularity": "DAY",                    #每个数据段的粒度，可以根据实际情况设置
，可以设置为HOUR/DAY
      "queryGranularity": "NONE"
    }
  },
  "tuningConfig": {
    "type": "kafka",                                  #指定数据来源
    "maxRowsPerSegment": 20000000,                    #每个段最大行数，到达后将生成新的段
    "basePersistDirectory": "/opt/apps/druidio_sugo/var/tmp/taskStorage",      #task临
时目录
    "buildV9Directly": true
  },
  "ioConfig": {
    "topic": "test",                                   #kafka对应的topic
    "consumerProperties": {
      "bootstrap.servers": "192.168.0.217:9092,192.168.0.215:9092"     #kafka的broker
位置
    },
    "taskCount": 2,                                   #启动的task数
    "replicas": 1,                                    #task的replicas数
    "taskDuration": "P1D",                            #task执行的时间，一般情况根据数据情况
可以设置为 P1D  p1H或P2H等
    "useEarliestOffset": "true"                       #第一次消费是否从最早的位置开始消费
  }
}
```

curl -X POST -H 'Content-Type: application/json' -d @supervisor-spec.json http://overlord:port/druid/indexer/v1/supervisor

即可启动supervisor，相应的任务可以在http://overlord:port 页面看到具体的执行和日志。