尚硅谷大数据技术之 Presto

(作者：尚硅谷大数据研发部)

版本：V2.0

第 1 章 Presto 简介

1.1 Presto 概念

Presto 是一个开源的分布式 SQL 查询引擎，适用于交互式分析查询，数据量支持 GB 到 PB 字节。

Presto 的设计和编写完全是为了解决像 Facebook 这样规模的商业数据仓库的交互式分析和处理速度的问题。

注意：虽然 Presto 可以解析 SQL，但它不是一个标准的数据库。不是 MySQL、Oracle 的代替品，也不能用来处理在线事务（OLTP）。

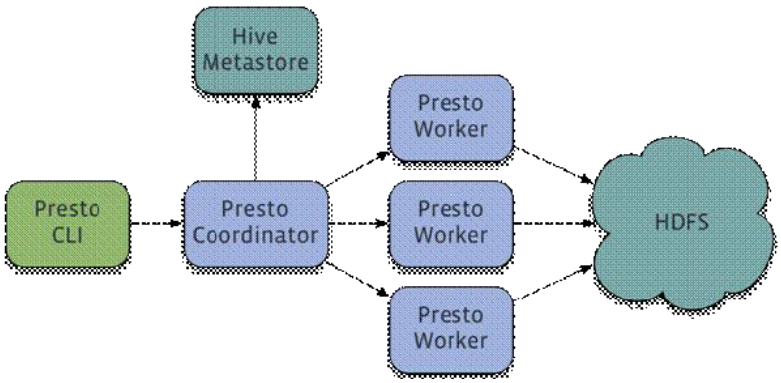
1.2 Presto 应用场景

Presto 支持在线数据查询，包括 Hive，关系数据库（MySQL、Oracle）以及专有数据存储。一条 Presto 查询可以将多个数据源的数据进行合并，可以跨越整个组织进行分析。

Presto 主要用来处理响应时间小于 1 秒到几分钟的场景。

1.3 Presto 架构

Presto 是一个运行在多台服务器上的分布式系统。完整安装包括一个 Coordinator 和多个 Worker。由客户端提交查询，从 Presto 命令行 CLI 提交到 Coordinator。Coordinator 进行解析，分析并执行查询计划，然后分发处理队列到 Worker。



更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

Presto 有两类服务器：Coordinator 和 Worker。

1）Coordinator

Coordinator 服务器是用来解析语句，执行计划分析和管理 Presto 的 Worker 结点。Presto 安装必须有一个 Coordinator 和多个 Worker。如果用于开发环境和测试，则一个 Presto 实例可以同时担任这两个角色。

Coordinator 跟踪每个 Work 的活动情况并协调查询语句的执行。Coordinator 为每个查询建立模型，模型包含多个 Stage，每个 Stage 再转为 Task 分发到不同的 Worker 上执行。

Coordinator 与 Worker、Client 通信是通过 REST API。

2）Worker

Worker 是负责执行任务和处理数据。Worker 从 Connector 获取数据。Worker 之间会交换中间数据。Coordinator 是负责从 Worker 获取结果并返回最终结果给 Client。

当 Worker 启动时，会广播自己去发现 Coordinator，并告知 Coordinator 它是可用，随时可以接受 Task。

Worker 与 Coordinator、Worker 通信是通过 REST API。

3）数据源

贯穿全文，你会看到一些术语：Connector、Catelog、Schema 和 Table。这些是 Presto 特定的数据源

（1）Connector

Connector 是适配器，用于 Presto 和数据源（如 Hive、RDBMS）的连接。你可以认为类似 JDBC 那样，但却是 Presto 的 SPI 的实现，使用标准的 API 来与不同的数据源交互。

Presto 有几个内建 Connector：JMX 的 Connector、System Connector（用于访问内建的 System table）、Hive 的 Connector、TPCH（用于 TPC-H 基准数据）。还有很多第三方的 Connector，所以 Presto 可以访问不同数据源的数据。

每个 Catalog 都有一个特定的 Connector。如果你使用 catelog 配置文件，你会发现每个文件都必须包含 connector.name 属性，用于指定 catelog 管理器（创建特定的 Connector 使用）。一个或多个 catelog 用同样的 connector 是访问同样的数据库。例如，你有两个 Hive 集群。你可以在一个 Presto 集群上配置两个 catelog，两个 catelog 都是用 Hive Connector，从而达到可以查询两个 Hive 集群。

（2）Catelog

一个 Catelog 包含 Schema 和 Connector。例如，你配置 JMX 的 catelog，通过 JXM Connector

访问 JXM 信息。当你执行一条 SQL 语句时，可以同时运行在多个 catelog。

Presto 处理 table 时，是通过表的完全限定（fully-qualified）名来找到 catelog。例如，一个表的权限定名是 hive.test_data.test，则 test 是表名，test_data 是 schema，hive 是 catelog。

Catelog 的定义文件是在 Presto 的配置目录中。

（3）Schema

Schema 是用于组织 table。把 catelog 好 schema 结合在一起来包含一组的表。当通过 Presto 访问 hive 或 Mysq 时，一个 schema 会同时转为 hive 和 mysql 的同等概念。

（4）Table

Table 跟关系型的表定义一样，但数据和表的映射是交给 Connector。

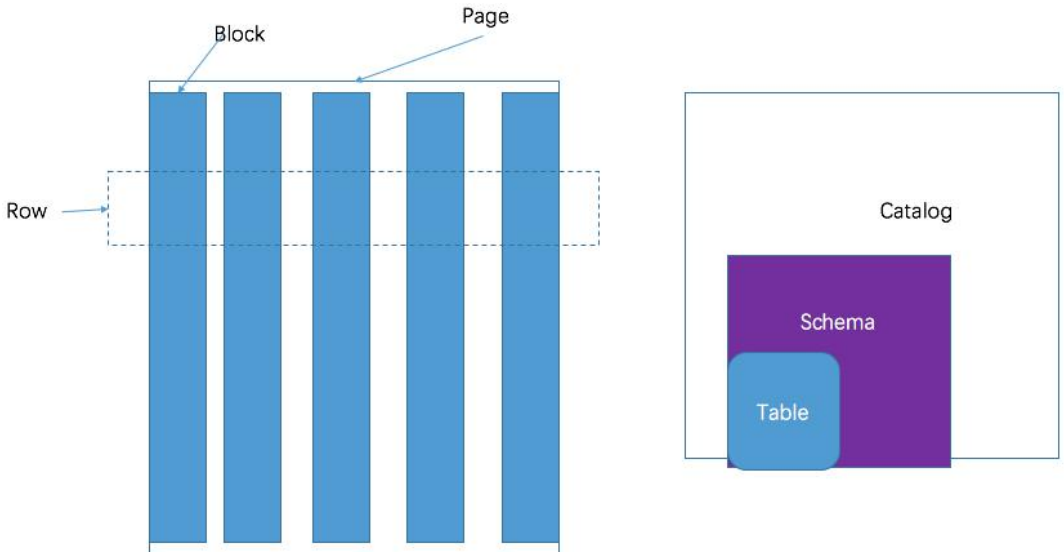
1.4 Presto 数据模型

1）Presto 采取三层表结构：

Catalog：对应某一类数据源，例如 Hive 的数据，或 MySql 的数据

Schema：对应 MySql 中的数据库

Table：对应 MySql 中的表



2）Presto 的存储单元包括：

Page：多行数据的集合，包含多个列的数据，内部仅提供逻辑行，实际以列式存储。

Block：一列数据，根据不同类型的数据，通常采取不同的编码方式，了解这些编码方式，有助于自己的存储系统对接 presto。

3）不同类型的 Block：

（1）Array 类型 Block，应用于固定宽度的类型，例如 int，long，double。block 由两部

分组成：

boolean valueIsNull[]表示每一行是否有值。

T values[] 每一行的具体值。

（2）可变宽度的 Block，应用于 String 类数据，由三部分信息组成

Slice：所有行的数据拼接起来的字符串。

int offsets[]：每一行数据的起始便宜位置。每一行的长度等于下一行的起始便宜减去当前行的起始便宜。

boolean valueIsNull[] 表示某一行是否有值。如果有某一行无值，那么这一行的便宜量等于上一行的偏移量。

（3）固定宽度的 String 类型的 block，所有行的数据拼接成一长串 Slice，每一行的长度固定。

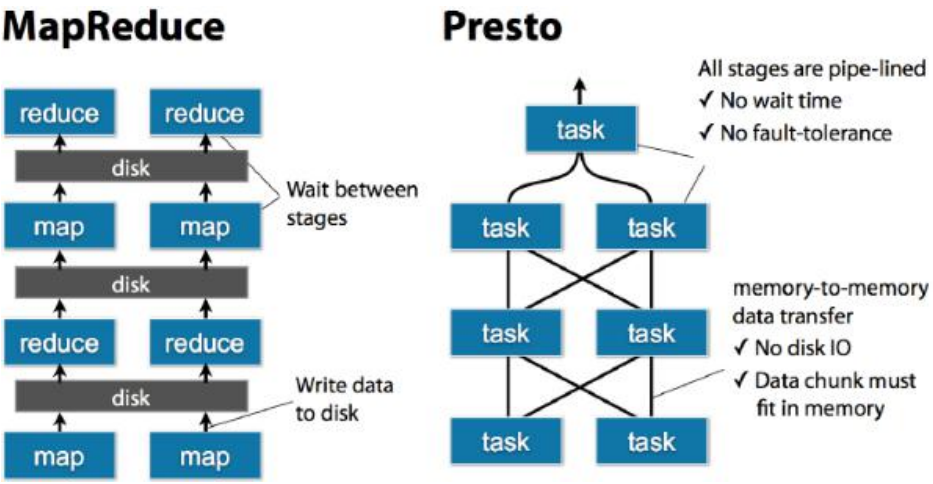
（4）字典 block：对于某些列，distinct 值较少，适合使用字典保存。主要有两部分组成：

字典，可以是任意一种类型的 block(甚至可以嵌套一个字典 block)，block 中的每一行按照顺序排序编号。

int ids[]表示每一行数据对应的 value 在字典中的编号。在查找时，首先找到某一行的 id，然后到字典中获取真实的值。

1.5 Presto 优缺点

Presto 中 SQL 运行过程：MapReduce vs Presto



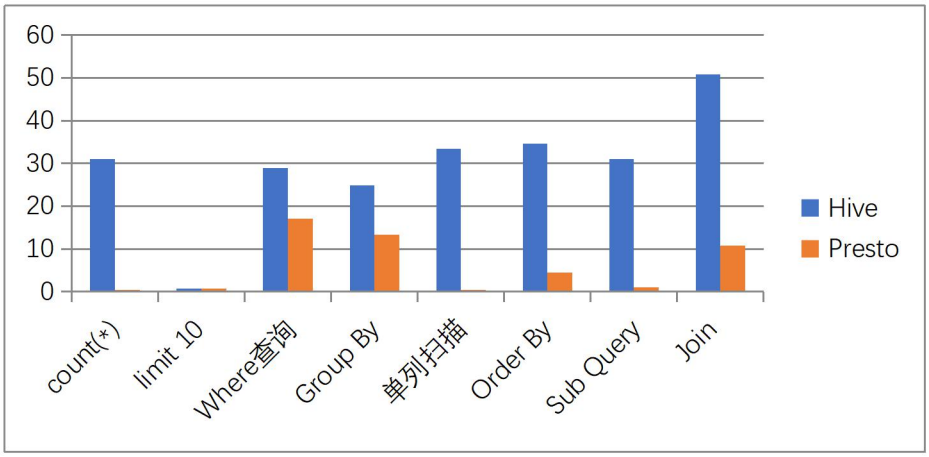
使用内存计算，减少与硬盘交互。

1.5.1 优点

1）Presto 与 Hive 对比，都能够处理 PB 级别的海量数据分析，但 Presto 是基于内存运算，减少没必要的硬盘 IO，所以更快。

2）能够连接多个数据源，跨数据源连表查，如从 Hive 查询大量网站访问记录，然后从 Mysql 中匹配出设备信息。

3）部署也比 Hive 简单，因为 Hive 是基于 HDFS 的，需要先部署 HDFS。



1.5.2 缺点

1）虽然能够处理 PB 级别的海量数据分析，但不是代表 Presto 把 PB 级别都放在内存中计算的。而是根据场景，如 count，avg 等聚合运算，是边读数据边计算，再清内存，再读数据再计算，这种耗的内存并不高。但是连表查，就可能产生大量的临时数据，因此速度会变慢，反而 Hive 此时会更擅长。

2）为了达到实时查询，可能会想到用它直连 MySql 来操作查询，这效率并不会提升，瓶颈依然在 MySql，此时还引入网络瓶颈，所以会比原本直接操作数据库要慢。

1.6 Presto、Impala 性能比较

https://blog.csdn.net/u012551524/article/details/79124532

第 2 章 Presto 安装部署

https://blog.csdn.net/u012551524/article/details/79013194

2.1 环境需求

Presto 的基本需求

- Linux or Mac OS X

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

- Java 8, 64-bit
- Python 2.4+

2.2 连接器

Presto 支持插接式连接器提供的数据。各连接器的设计需求会有所不同。

HADOOP / HIVE

Presto 支持从以下版本的 Hadoop 中读取 Hive 数据：

Apache Hadoop 1.x

Apache Hadoop 2.x

Cloudera CDH 4

Cloudera CDH 5

支持以下文件类型：Text, SequenceFile, RCFile, ORC

此外，需要有远程的 Hive 元数据。不支持本地或嵌入模式。Presto 不使用 MapReduce，只需要 HDFS。

2.3 安装 Presto 服务器

2.3.1 下载安装包

https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.189/presto-server-0.189.tar.gz

2.3.2 解压安装包

```
tar -zxvf presto-server-0.189.tar.gz -C /opt/cdh-5.3.6/
chown -R hadoop:hadoop /opt/cdh-5.3.6/presto-server-0.189/
```

2.3.3 配置 Presto

在安装目录中创建一个 etc 目录。在这个 etc 目录中放入以下配置信息：

- 节点属性：每个节点的环境配置信息
- JVM 配置：JVM 的命令行选项
- 配置属性：Presto server 的配置信息
- Catalog 属性：configuration forConnectors（数据源）的配置信息

1）Node Properties

节点属性配置文件：etc/node.properties 包含针对于每个节点的特定的配置信息。一个节点就是在一台机器上安装的 Presto 实例。这份配置文件一般情况下是在 Presto 第一次安装的

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

时候，由部署系统创建的。一个 etc/node.properties 配置文件至少包含如下配置信息：

```
node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
node.data-dir=/var/presto/data
```

针对上面的配置信息描述如下：

node.environment：集群名称。所有在同一个集群中的 Presto 节点必须拥有相同的集群名称。

node.id：每个 Presto 节点的唯一标示。每个节点的 node.id 都必须是唯一的。在 Presto 进行重启或者升级过程中每个节点的 node.id 必须保持不变。如果在一个节点上安装多个 Presto 实例（例如：在同一台机器上安装多个 Presto 节点），那么每个 Presto 节点必须拥有唯一的 node.id。

node.data-dir： 数据存储目录的位置（操作系统上的路径）。Presto 将会把日期和数据存储在这个目录下。

2）JVM 配置

JVM 配置文件，etc/jvm.config，包含一系列在启动 JVM 的时候需要使用的命令行选项。这份配置文件的格式是：一系列的选项，每行配置一个单独的选项。由于这些选项不在 shell 命令中使用。 因此即使将每个选项通过空格或者其他的分隔符分开，java 程序也不会将这些选项分开，而是作为一个命令行选项处理。（就想下面例子中的 OnOutOfMemoryError 选项）。

一个典型的 etc/jvm.config 配置文件如下：

```
-server
-Xmx16G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:+ExitOnOutOfMemoryError
```

由于 OutOfMemoryError 将会导致 JVM 处于不一致状态，所以遇到这种错误的时候我们一般的处理措施就是将 dump headp 中的信息（用于 debugging），然后强制终止进程。

Presto 会将查询编译成字节码文件，因此 Presto 会生成很多 class，因此我们我们应该增大 Perm 区的大小（在 Perm 中主要存储 class）并且要允许 Jvm class unloading。

3）Config Properties

Presto 的配置文件：etc/config.properties 包含了 Presto server 的所有配置信息。每个 Presto server 既是一个 coordinator 也是一个 worker。但是在大型集群中，处于性能考虑，建议单独

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

用一台机器作为 coordinator。

一个 coordinator 的 etc/config.properties 应该至少包含以下信息：

```
coordinator=true
node-scheduler.include-coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri=http://example.net:8080
```

以下是最基本的 worker 配置：

```
coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery.uri=http://example.net:8080
```

但是如果你用一台机器进行测试，那么这一台机器将会即作为 coordinator，也作为 worker。配置文件将会如下所示：

```
coordinator=true
node-scheduler.include-coordinator=true
http-server.http.port=8080
query.max-memory=5GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri=http://example.net:8080
```

对配置项解释如下：

coordinator：指定是否运维 Presto 实例作为一个 coordinator(接收来自客户端的查询情切管理每个查询的执行过程)。

node-scheduler.include-coordinator：是否允许在 coordinator 服务中进行调度工作。对于大型的集群，在一个节点上的 Presto server 即作为 coordinator 又作为 worke 将会降低查询性能。因为如果一个服务器作为 worker 使用，那么大部分的资源都不会被 worker 占用，那么就不会有足够的资源进行关键任务调度、管理和监控查询执行。

http-server.http.port：指定 HTTP server 的端口。Presto 使用 HTTP 进行内部和外部的所有通讯。

task.max-memory=1GB：一个单独的任务使用的最大内存 (一个查询计划的某个执行部分会在一个特定的节点上执行)。这个配置参数限制的 GROUP BY 语句中的 Group 的数目、JOIN 关联中的右关联表的大小、ORDER BY 语句中的行数和一个窗口函数中处理的行数。该参数应该根据并发查询的数量和查询的复杂度进行调整。如果该参数设置的太低，很多查询将不能执行；但是如果设置的太高将会导致 JVM 把内存耗光。

discovery-server.enabled：Presto 通过 Discovery 服务来找到集群中所有的节点。为了能

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

够找到集群中所有的节点，每一个 Presto 实例都会在启动的时候将自己注册到 discovery 服务。Presto 为了简化部署，并且也不想再增加一个新的服务进程，Presto coordinator 可以运行一个内嵌在 coordinator 里面的 Discovery 服务。这个内嵌的 Discovery 服务和 Presto 共享 HTTP server 并且使用同样的端口。

discovery.uri：Discovery server 的 URI。由于启用了 Presto coordinator 内嵌的 Discovery 服务，因此这个 uri 就是 Presto coordinator 的 uri。修改 example.net:8080，根据你的实际环境设置该 URI。注意：这个 URI 一定不能以“/“结尾。

4）日志级别

日志配置文件：etc/log.properties。在这个配置文件中允许你根据不同的日志结构设置不同的日志级别。每个 logger 都有一个名字（通常是使用 logger 的类的全标示类名）. Loggers 通过名字中的“.“来表示层级和集成关系。 (像 java 里面的包). 如下面的 log 配置信息：

```
com.facebook.presto=INFO
```

5）Catalog Properties

Presto 通过 connectors 访问数据。这些 connectors 挂载在 catalogs 上。connector 可以提供一个 catalog 中所有的 schema 和表。例如：Hive connector 将每个 hive 的 database 都映射成为一个 schema，所以如果 hive connector 挂载到了名为 hive 的 catalog，并且在 hive 的 web 有一张名为 clicks 的表，那么在 Presto 中可以通过 hive.web.clicks 来访问这张表。

通过在 etc/catalog 目录下创建 catalog 属性文件来完成 catalogs 的注册。例如：可以先创建一个 etc/catalog/jmx.properties 文件，文件中的内容如下，完成在 jmxcatalog 上挂载一个 jmxconnector：

```
connector.name=jmx
```

查看 Connectors 的详细配置选项。

2.3.4 运行 Presto

在安装目录的 bin/launcher 文件，就是启动脚本。Presto 可以使用如下命令作为一个后台进程启动：

```
bin/launcher start
```

另外，也可以在前台运行，日志和相关输出将会写入 stdout/stderr（可以使用类似 daemontools 的工具捕捉这两个数据流）：

```
bin/launcher run
```

运行 bin/launcher–help，Presto 将会列出支持的命令和命令行选项。另外可以通过运行 bin/launcher–verbose 命令，来调试安装是否正确。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

启动完之后，日志将会写在 var/log 目录下，该目录下有如下文件：

launcher.log：这个日志文件由 launcher 创建，并且 server 的 stdout 和 stderr 都被重定向到了这个日志文件中。这份日志文件中只会有很少的信息，包括：

在 server 日志系统初始化的时候产生的日志和 JVM 产生的诊断和测试信息。

server.log：这个是 Presto 使用的主要日志文件。一般情况下，该文件中将会包括 server 初始化失败时产生的相关信息。这份文件会被自动轮转和压缩。

http-request.log： 这是 HTTP 请求的日志文件，包括 server 收到的每个 HTTP 请求信息，这份文件会被自动轮转和压缩。

2.4 安装 Presto 客户端

1）下载：

https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.189/presto-cli-0.189-executable.jar

2）上传 Linux 服务器上，重命名为 presto：

```
$mv presto-cli-0.189-executable.jar presto
$chmod a+x presto
```

3）执行以下命令：

```
$ ./presto --server localhost:8080 --catalog hive --schema default
```

2.5 配置 Presto 连接 Hive

1）编辑 hive-site.xml 文件，增加以下内容：

```
<property>
<name>hive.server2.thrift.port</name>
<value>10000</value>
</property>
<property>
<name>hive.server2.thrift.bind.host</name>
<value>chavin.king</value>
</property>
<property>
<name>hive.metastore.uris</name>
<value>thrift://chavin.king:9083</value>
</property>
```

2）启动 hiveserver2 和 hive 元数据服务：

```
bin/hive --service hiveserver2 &
bin/hive --service matestore &
```

3）配置 hive 插件，etc/catalog 目录下创建 hive.properties 文件，输入如下内容。

（1）hive 配置：

connector.name=hive-hadoop2 #这个连接器的选择要根据自身集群情况结合插件包的名

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

字来写

hive.metastore.uri=thrift://chavin.king:9083 #修改为 hive-metastore 服务所在的主机名称，这里我是安装在 master 节点

（2）HDFS Configuration：

如果 hive metastore 的引用文件存放在一个存在联邦的 HDFS 上，或者你是通过其他非标准的客户端来访问 HDFS 集群的，请添加以下配置信息来指向你的 HDFS 配置文件:

hive.config.resources=/etc/hadoop/conf/core-site.xml,/etc/hadoop/conf/hdfs-site.xml

大多数情况下，Presto 会在安装过程中自动完成 HDFS 客户端的配置。 如果确实需要特殊配置，只需要添加一些额外的配置文件，并且需要指定这些新加的配置文件。 建议将配置文件中的配置属性最小化。尽量少添加一些配置属性，因为过多的添加配置属性会引起其他问题。

（3）Configuration Properties

Property Name	Description	Example
hive.metastore.uri	The URI of the Hive Metastore to connect to using the Thrift protocol. This property is required.	thrift://192.0.2.3:9083
hive.config.resources	An optional comma-separated list of HDFS configuration files. These files must exist on the machines running Presto. Only specify this if absolutely necessary to access HDFS.	/etc/hdfs-site.xml
hive.storage-format	The default file format used when creating new tables	RCBINARY
hive.force-local-scheduling	Force splits to be scheduled on the same node as the Hadoop DataNode process serving the split data. This is	true

	useful for installations where Presto is collocated with every DataNode.	

4）presto 连接 hive schema，注意 presto 不能进行垮库 join 操作，测试结果如下：

```
$ ./presto --server localhost:8080 --catalog hive --schema chavin
presto:chavin> select * from emp;
```

empno | ename | job | mgr | hiredate | sal | comm | deptno

-------+--------+----------+------+-----------+--------+--------+--------

7369 | SMITH | CLERK | 7902 | 1980/12/17 | 800.0 | NULL | 20

7499 | ALLEN | SALESMAN | 7698 | 1981/2/20 | 1600.0 | 300.0 | 30

7521 | WARD | SALESMAN | 7698 | 1981/2/22 | 1250.0 | 500.0 | 30

7566 | JONES | MANAGER | 7839 | 1981/4/2 | 2975.0 | NULL | 20

7654 | MARTIN | SALESMAN | 7698 | 1981/9/28 | 1250.0 | 1400.0 | 30

7698 | BLAKE | MANAGER | 7839 | 1981/5/1 | 2850.0 | NULL | 30

7782 | CLARK | MANAGER | 7839 | 1981/6/9 | 2450.0 | NULL | 10

7788 | SCOTT | ANALYST | 7566 | 1987/4/19 | 3000.0 | NULL | 20

7839 | KING | PRESIDENT | NULL | 1981/11/17 | 5000.0 | NULL | 10

7844 | TURNER | SALESMAN | 7698 | 1981/9/8 | 1500.0 | 0.0 | 30

7876 | ADAMS | CLERK | 7788 | 1987/5/23 | 1100.0 | NULL | 20

7900 | JAMES | CLERK | 7698 | 1981/12/3 | 950.0 | NULL | 30

7902 | FORD | ANALYST | 7566 | 1981/12/3 | 3000.0 | NULL | 20

7934 | MILLER | CLERK | 7782 | 1982/1/23 | 1300.0 | NULL | 10

(14 rows)

Query 20170711_081802_00002_ydh8n, FINISHED, 1 node

Splits: 17 total, 17 done (100.00%)

0:05 [14 rows, 657B] [2 rows/s, 130B/s]

presto:chavin>

第 3 章 Presto 优化

3.1 数据存储

1）合理设置分区

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

与 Hive 类似，Presto 会根据元信息读取分区数据，合理的分区能减少 Presto 数据读取量，提升查询性能。

2）使用列式存储

Presto 对 ORC 文件读取做了特定优化，因此在 Hive 中创建 Presto 使用的表时，建议采用 ORC 格式存储。相对于 Parquet，Presto 对 ORC 支持更好。

3）使用压缩

数据压缩可以减少节点间数据传输对 IO 带宽压力，对于即席查询需要快速解压，建议采用 Snappy 压缩。

4）预先排序

对于已经排序的数据，在查询的数据过滤阶段，ORC 格式支持跳过读取不必要的数据。比如对于经常需要过滤的字段可以预先排序。

```
INSERT INTO table nation_orc partition(p) SELECT * FROM nation SORT
BY n_name;
```

如果需要过滤 n_name 字段，则性能将提升。

```
SELECT count(*) FROM nation_orc WHERE n_name='AUSTRALIA';
```

3.2 查询 SQL 优化

1）只选择使用必要的字段

由于采用列式存储，选择需要的字段可加快字段的读取、减少数据量。避免采用*读取所有字段。

```
[GOOD]: SELECT time,user,host FROM tbl
[BAD]:  SELECT * FROM tbl
```

2）过滤条件必须加上分区字段

对于有分区的表，where 语句中优先使用分区字段进行过滤。acct_day 是分区字段，visit_time 是具体访问时间。

```
[GOOD]: SELECT time,user,host FROM tbl where acct_day=20171101
[BAD]:  SELECT * FROM tbl where visit_time=20171101
```

3）Group By 语句优化

合理安排 Group by 语句中字段顺序对性能有一定提升。将 Group By 语句中字段按照每个字段 distinct 数据多少进行降序排列。

```
[GOOD]: SELECT GROUP BY uid, gender
[BAD]:  SELECT GROUP BY gender, uid
```

4）Order by 时使用 Limit

Order by 需要扫描数据到单个 worker 节点进行排序，导致单个 worker 需要大量内存。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

如果是查询 Top N 或者 Bottom N，使用 limit 可减少排序计算和内存压力。

```
[GOOD]: SELECT * FROM tbl ORDER BY time LIMIT 100
[BAD]:  SELECT * FROM tbl ORDER BY time
```

5）使用近似聚合函数

Presto 有一些近似聚合函数，对于允许有少量误差的查询场景，使用这些函数对查询性能有大幅提升。比如使用 approx_distinct() 函数比 Count(distinct x)有大概 2.3%的误差。

```
SELECT approx_distinct(user_id) FROM access
```

6）用 regexp_like 代替多个 like 语句

Presto 查询优化器没有对多个 like 语句进行优化，使用 regexp_like 对性能有较大提升

```
[GOOD]
SELECT
  ...
FROM
  access
WHERE
  regexp_like(method, 'GET|POST|PUT|DELETE')

[BAD]
SELECT
  ...
FROM
  access
WHERE
  method LIKE '%GET%' OR
  method LIKE '%POST%' OR
  method LIKE '%PUT%' OR
  method LIKE '%DELETE%'
```

7）使用 Join 语句时将大表放在左边

Presto 中 join 的默认算法是 broadcast join，即将 join 左边的表分割到多个 worker，然后将 join 右边的表数据整个复制一份发送到每个 worker 进行计算。如果右边的表数据量太大，则可能会报内存溢出错误。

```
[GOOD] SELECT ... FROM large_table l join small_table s on l.id
= s.id
[BAD] SELECT ... FROM small_table s join large_table l on l.id =
s.id
```

8）使用 Rank 函数代替 row_number 函数来获取 Top N

在进行一些分组排序场景时，使用 rank 函数性能更好。

```
[GOOD]
SELECT checksum(rnk)
FROM (
  SELECT rank() OVER (PARTITION BY l_orderkey, l_partkey ORDER BY
l_shipdate DESC) AS rnk
  FROM lineitem
) t
WHERE rnk = 1
```

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

```
[BAD]
SELECT checksum(rnk)
FROM (
  SELECT row_number() OVER (PARTITION BY l_orderkey, l_partkey
ORDER BY l_shipdate DESC) AS rnk
  FROM lineitem
) t
WHERE rnk = 1
```

3.3 无缝替换 Hive 表

如果之前的 hive 表没有用到 ORC 和 snappy，那么怎么无缝替换而不影响线上的应用：

比如如下一个 hive 表：

```
CREATE TABLE bdc_dm.res_category(
channel_id1 int comment '1 级渠道 id',
province string COMMENT '省',
city string comment '市',
uv int comment 'uv'
)
comment 'example'
partitioned by (landing_date int COMMENT '日期:yyyymmdd')
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' COLLECTION ITEMS
TERMINATED BY ',' MAP KEYS TERMINATED BY ':' LINES TERMINATED BY
'\n';
```

建立对应的 orc 表

```
CREATE TABLE bdc_dm.res_category_orc(
channel_id1 int comment '1 级渠道 id',
province string COMMENT '省',
city string comment '市',
uv int comment 'uv'
)
comment 'example'
partitioned by (landing_date int COMMENT '日期:yyyymmdd')
row format delimited fields terminated by '\t'
stored as orc
TBLPROPERTIES ("orc.compress"="SNAPPY");
```

先将数据灌入 orc 表，然后更换表名

```
insert overwrite table bdc_dm.res_category_orc
partition(landing_date)
select * from bdc_dm.res_category where landing_date >= 20171001;

ALTER TABLE bdc_dm.res_category RENAME TO
bdc_dm.res_category_tmp;
ALTER TABLE bdc_dm.res_category_orc RENAME TO
bdc_dm.res_category;
```

其中 res_category_tmp 是一个备份表，若线上运行一段时间后没有出现问题，则可以删除该表。

3.4 注意事项

ORC 和 Parquet 都支持列式存储，但是 ORC 对 Presto 支持更好（Parquet 对 Impala 支持

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

更好）

对于列式存储而言，存储文件为二进制的，对于经常增删字段的表，建议不要使用列式存储（修改文件元数据代价大）。对比数据仓库，dwd 层建议不要使用 ORC，而 dm 层则建议使用。

第 5 章 Presto 上使用 SQL 遇到的坑

https://segmentfault.com/a/1190000013120454?utm_source=tag-newest

5.1 如何加快在 Presto 上的数据统计

很多的时候，在 Presto 上对数据库跨库查询，例如 Mysql 数据库。这个时候 Presto 的做法是从 MySQL 数据库端拉取最基本的数据，然后再去做进一步的处理，例如统计等聚合操作。

举个例子：

```
SELECT count(id) FROM table_1 WHERE condition=1;
```

上面的 SQL 语句会分为 3 个步骤进行：

（1）Presto 发起到 Mysql 数据库进行查询

```
SELECT id FROM table_1 WHERE condition=1;
```

（2）对结果进行 count 计算

（3）返回结果

所以说，对于 Presto 来说，其跨库查询的瓶颈是在数据拉取这个步骤。若要提高数据统计的速度，可考虑把 Mysql 中相关的数据表定期转移到 HDFS 中，并转存为高效的列式存储格式 ORC。

所以定时归档是一个很好的选择，这里还要注意，在归档的时候我们要选择一个归档字段，如果是按日归档，我们可以用日期作为这个字段的值，采用 yyyyMMdd 的形式，例如 20180123.

一般创建归档数据库的 SQL 语句如下：

```
CREATE TABLE IF NOT EXISTS table_1 (
id INTEGER,
........
partition_date INTEGER
)WITH ( format = 'ORC', partitioned_by =
ARRAY['partition_date'] );
```

查看创建的库结构：

```
SHOW CREATE TABLE table_1; /*Only Presto*/
```

带有分区的表创建完成之后，每天只要更新分区字段 partition_date 就可以了，聪明的

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

Presto 就能将数据放置到规划好的分区了。

如果要查看一个数据表的分区字段是什么，可以下面的语句：

```
SHOW PARTITIONS FROM table_1 /*Only Presto*/
```

5.2 查询条件中尽量带上分区字段进行过滤

如果数据被规当到 HDFS 中，并带有分区字段。在每次查询归档表的时候，要带上分区字段作为过滤条件，这样可以加快查询速度。因为有了分区字段作为查询条件，就能帮助 Presto 避免全区扫描，减少 Presto 需要扫描的 HDFS 的文件数。

5.3 多多使用 WITH 语句

使用 Presto 分析统计数据时，可考虑把多次查询合并为一次查询，用 Presto 提供的子查询完成。

这点和我们熟知的 MySQL 的使用不是很一样。

例如：

```
WITH subquery_1 AS (
    SELECT a1, a2, a3
    FROM Table_1
    WHERE a3 between 20180101 and 20180131
),              /*子查询 subquery_1,注意：多个子查询需要用逗号分隔*/
subquery_2 AS (
    SELECT b1, b2, b3
    FROM Table_2
    WHERE b3 between 20180101 and 20180131
)               /*最后一个子查询后不要带逗号，不然会报错。*/
SELECT
    subquery_1.a1, subquery_1.a2,
    subquery_2.b1, subquery_2.b2
FROM subquery_1
    JOIN subquery_2
    ON subquery_1.a3 = subquery_2.b3;
```

5.4 利用子查询，减少读表的次数，尤其是大数据量的表

具体做法是，将使用频繁的表作为一个子查询抽离出来，避免多次 read。

5.5 只查询需要的字段

一定要避免在查询中使用 SELECT *这样的语句，换位思考，如果让你去查询数据是不是告诉你的越具体，工作效率越高呢。

对于我们的数据库而言也是这样，任务越明确，工作效率越高。

对于要查询全部字段的需求也是这样，没有偷懒的捷径，把它们都写出来。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

5.6 Join 查询优化

Join 左边尽量放小数据量的表，而且最好是重复关联键少的表

5.7 字段名引用

Presto 中的字段名引用使用双引号分割，这个要区别于 MySQL 的反引号`。

当然，你可以不加这个双引号。

5.8 时间函数

对于 timestamp，需要进行比较的时候，需要添加 timestamp 关键字，而 MySQL 中对 timestamp 可以直接进行比较。

```
/*MySQL 的写法*/
SELECT t FROM a WHERE t > '2017-01-01 00:00:00';

/*Presto 中的写法*/
SELECT t FROM a WHERE t > timestamp '2017-01-01 00:00:00';
```

5.9 MD5 函数的使用

Presto 中 MD5 函数传入的是 binary 类型，返回的也是 binary 类型，要对字符串进行 MD5 操作时，需要转换.

```
SELECT to_hex(md5(to_utf8('1212')));
```

5.10 不支持 INSERT OVERWRITE 语法

Presto 中不支持 insert overwrite 语法，只能先 delete，然后 insert into。

5.11 ORC 格式

Presto 中对 ORC 文件格式进行了针对性优化，但在 impala 中目前不支持 ORC 格式的表，hive 中支持 ORC 格式的表，所以想用列式存储的时候可以优先考虑 ORC 格式。

5.12 PARQUET 格式

Presto 目前支持 parquet 格式，支持查询，但不支持 insert。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网