Presto

# 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）。

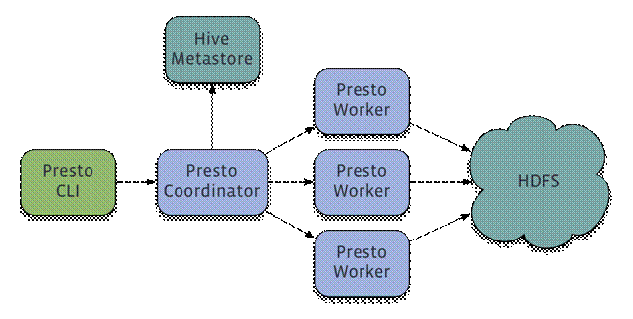
## 1.2 Presto应用场景

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。



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。

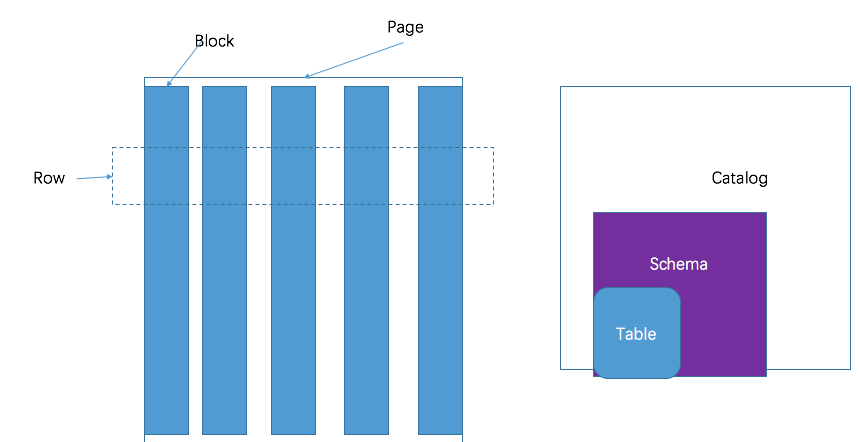
## 1.4 Presto数据模型

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。

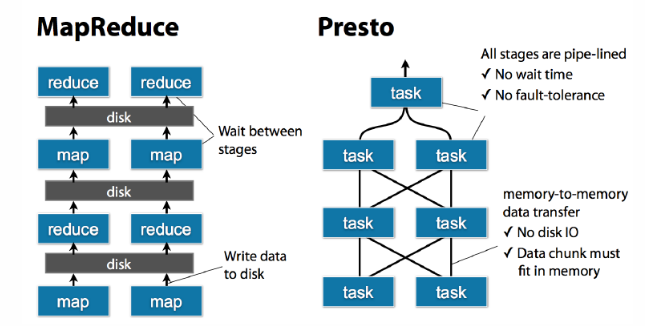
（4）字典block：对于某些列，distinct值较少，适合使用字典保存。主要有两部分组成：

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
* 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。

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 useful for installations where Presto is collocated with every DataNode. | true |

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

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。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

[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。

## 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。