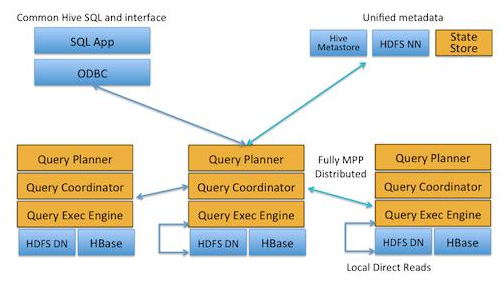
# Impala架构和工作原理

## 1. Impala架构

　　Impala是Cloudera在受到Google的Dremel启发下开发的实时交互SQL大数据查询工具（实时SQL查询引擎Impala），Impala没有再使用缓慢的Hive+MapReduce批处理，而是通过使用与商用并行关系数据库中类似的分布式查询引擎（由Query Planner、Query Coordinator和Query Exec Engine三部分组成），可以直接从HDFS或HBase中用SELECT、JOIN和统计函数查询数据，从而大大降低了延迟。



　　　　　　　　Impala架构如图所示

Impala主要由Impalad， State Store和CLI组成。

**Impalad**: 与DataNode运行在同一节点上，由Impalad进程表示，它接收客户端的查询请求（接收查询请求的Impalad为Coordinator，Coordinator通过JNI调用java前端解释SQL查询语句，生成查询计划树，再通过调度器把执行计划分发给具有相应数据的其它Impalad进行执行），读写数据，并行执行查询，并把结果通过网络流式的传送回给Coordinator，由Coordinator返回给客户端。同时Impalad也与State Store保持连接，用于确定哪个Impalad是健康和可以接受新的工作。

在Impalad中启动三个ThriftServer: beeswax\_server（连接客户端），hs2\_server（借用Hive元数据）， be\_server（Impalad内部使用）和一个ImpalaServer服务。

**Impala State Store**: 跟踪集群中的Impalad的健康状态及位置信息，由statestored进程表示，它通过创建多个线程来处理Impalad的注册订阅和与各Impalad保持心跳连接，各Impalad都会缓存一份State Store中的信息，当State Store离线后（Impalad发现State Store处于离线时，会进入recovery模式，反复注册，当State Store重新加入集群后，自动恢复正常，更新缓存数据）因为Impalad有State Store的缓存仍然可以工作，但会因为有些Impalad失效了，而已缓存数据无法更新，导致把执行计划分配给了失效的Impalad，导致查询失败。

**CLI**: 提供给用户查询使用的命令行工具（Impala Shell使用python实现），同时Impala还提供了Hue，JDBC， ODBC使用接口。

## Impala支持的文件格式

Impala可以对Hadoop中大多数格式的文件进行查询。它能通过create table和insert的方式将一部分格式的数据加载到table中，但值得注意的是，有一些格式的数据它是无法写入的（write to）。对于Impala无法写入的数据格式，我们只能通过Hive建表，通过Hive进行数据的写入，然后使用Impala来对这些保存好的数据执行查询操作。

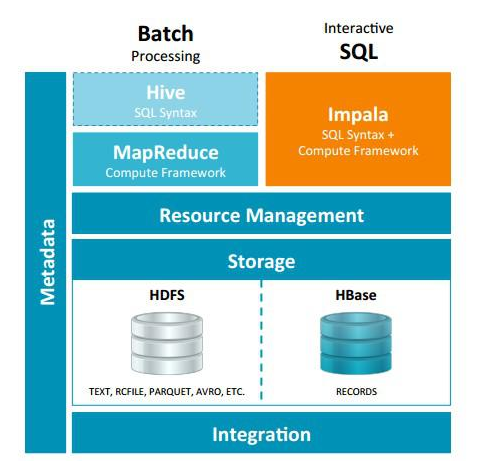
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文件类型** | **文件格式** | **压缩编码** | **能否CREATE ?** | **能否INSERT ?** |
| **Parquet** | 结构化 | Snappy  GZIP | 能 | 能 |
| **Text** | 非结构化 | LZO | 能。  如果建表时没有指定存储类型，默认采用未压缩的text，字段由ASCII编码的0x01字符串分割。 | 能。  如果使用了LZO压缩，则只能通过Hive建表和插入数据。 |
| **Avro** | 结构化 | Snappy  GZIP  Deflate  BZIP2 | 在Impala 1.4.0 或者更高的版本上支持，之前的版本只能通过Hive来建表。 | 不能。  只能通过LOAD DATA的方式将已经转换好格式的数据加载进去，或者使用Hive来插入数据。 |
| **RCFile** | 结构化 | Snappy  GZIP  Deflate  BZIP2 | 能 | 不能。  只能通过LOAD DATA的方式将已经转换好格式的数据加载进去，或者使用Hive来插入数据。 |
| **SequenceFile** | 结构化 | Snappy  GZIP  deflate  BZIP2 | 能 | 不能。  只能通过LOAD DATA的方式将已经转换好格式的数据加载进去，或者使用Hive来插入数据。 |

Impala支持以下压缩编码：

* Snappy – 推荐的编码，因为它在压缩率和解压速度之间有很好的平衡性，Snappy压缩速度很快，但是不如GZIP那样能节约更多的存储空间。Impala不支持Snappy压缩的text file。
* GZIP – 压缩比很高能节约很多存储空间，Impala不支持GZIP压缩的text file。
* Deflate – Impala不支持GZIP压缩的text file。
* BZIP2 - Impala不支持BZIP2压缩的text file。
* LZO – 只用于text file，Impala可以查询LZO压缩的text格式数据表，但是不支持insert数据，只能通过Hive来完成数据的insert。

## 2. 与Hive的关系

        Impala与Hive都是构建在Hadoop之上的数据查询工具各有不同的侧重适应面，但从客户端使用来看Impala与Hive有很多的共同之处，如数据表元数据、ODBC/JDBC驱动、SQL语法、灵活的文件格式、存储资源池等。Impala与Hive在Hadoop中的关系如图 2所示。Hive适合于长时间的批处理查询分析，而Impala适合于实时交互式SQL查询，Impala给数据分析人员提供了快速实验、验证想法的大数据分析工具。可以先使用hive进行数据转换处理，之后使用Impala在Hive处理后的结果数据集上进行快速的数据分析。



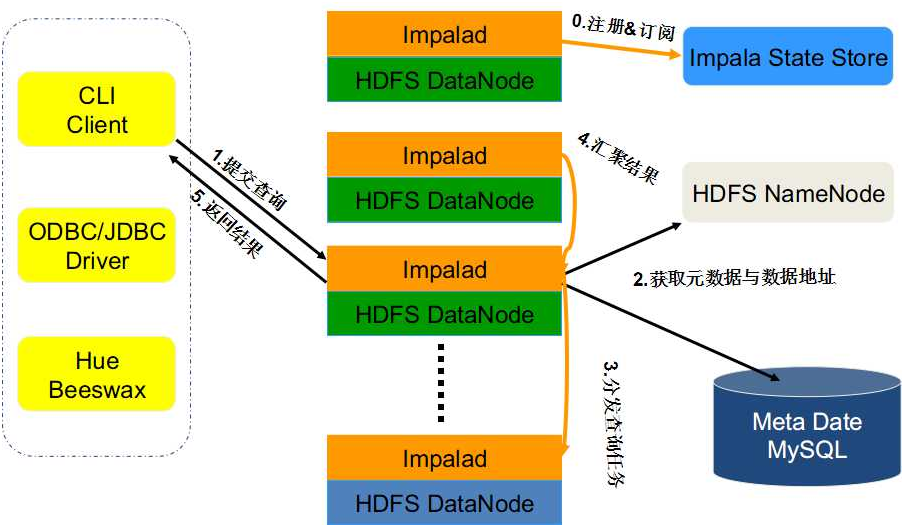
## 3. Impala的查询处理过程

　　Impalad分为Java前端与C++处理后端，接受客户端连接的Impalad即作为这次查询的Coordinator，Coordinator通过JNI调用Java前端对用户的查询SQL进行分析生成执行计划树，不同的操作对应不用的PlanNode, 如：SelectNode， ScanNode， SortNode， AggregationNode， HashJoinNode等等。

　　执行计划树的每个原子操作由一个PlanFragment表示，通常一条查询语句由多个Plan Fragment组成， Plan Fragment 0表示执行树的根，汇聚结果返回给用户，执行树的叶子结点一般是Scan操作，分布式并行执行。

      Java前端产生的执行计划树以Thrift数据格式返回给Impala C++后端（Coordinator）（执行计划分为多个阶段，每一个阶段叫做一个PlanFragment，每一个PlanFragment在执行时可以由多个Impalad实例并行执行(有些PlanFragment只能由一个Impalad实例执行,如聚合操作)，整个执行计划为一执行计划树），由Coordinator根据执行计划，数据存储信息（Impala通过libhdfs与HDFS进行交互。通过hdfsGetHosts方法获得文件数据块所在节点的位置信息），通过调度器（现在只有simple-scheduler, 使用round-robin算法）Coordinator::Exec对生成的执行计划树分配给相应的后端执行器Impalad执行（查询会使用LLVM进行代码生成，编译，执行。对于使用LLVM如何提高性能这里有说明），通过调用GetNext()方法获取计算结果，如果是insert语句，则将计算结果通过libhdfs写回HDFS当所有输入数据被消耗光，执行结束，之后注销此次查询服务。

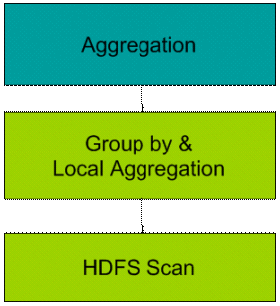
　　Impala的查询处理流程大概如图所示：



下面以一个SQL查询语句为例分析Impala的查询处理流程。如select sum(id), count(id), avg(id) from customer\_small  group by id； 以此语句生成的计划为：

PLAN FRAGMENT **0**   PARTITION: UNPARTITIONED   **4**:EXCHANGE      tuple ids: **1**PLAN FRAGMENT **1**   PARTITION: HASH\_PARTITIONED: <slot **1**>   STREAM DATA SINK     EXCHANGE ID: **4**     UNPARTITIONED   **3**:AGGREGATE   |  output: SUM(<slot **2**>), SUM(<slot **3**>)   |  group by: <slot **1**>   |  tuple ids: **1**   |     **2**:EXCHANGE      tuple ids: **1**PLAN FRAGMENT **2**   PARTITION: RANDOM   STREAM DATA SINK     EXCHANGE ID: **2**     HASH\_PARTITIONED: <slot **1**>   **1**:AGGREGATE   |  output: SUM(id), COUNT(id)   |  group by: id   |  tuple ids: **1**   |     **0**:SCAN HDFS      table=default.customer\_small #partitions=**1** size=193B      tuple ids: **0**

执行行计划树如图所示， 绿色的部分为可以分布式并行执行：



## 4. Impala相对于Hive所使用的优化技术

1、没有使用MapReduce进行并行计算，虽然MapReduce是非常好的并行计算框架，但它更多的面向批处理模式，而不是面向交互式的SQL执行。与MapReduce相比：Impala把整个查询分成一执行计划树，而不是一连串的MapReduce任务，在分发执行计划后，Impala使用拉式获取数据的方式获取结果，把结果数据组成按执行树流式传递汇集，减少的了把中间结果写入磁盘的步骤，再从磁盘读取数据的开销。Impala使用服务的方式避免每次执行查询都需要启动的开销，即相比Hive没了MapReduce启动时间。  
2、使用LLVM产生运行代码，针对特定查询生成特定代码，同时使用Inline的方式减少函数调用的开销，加快执行效率。  
3、充分利用可用的硬件指令（SSE4.2）。  
4、更好的IO调度，Impala知道数据块所在的磁盘位置能够更好的利用多磁盘的优势，同时Impala支持直接数据块读取和本地代码计算checksum。  
5、通过选择合适的数据存储格式可以得到最好的性能（Impala支持多种存储格式）。  
6、最大使用内存，中间结果不写磁盘，及时通过网络以stream的方式传递。

## 5. Impala与Hive的异同

**数据存储**：使用相同的存储数据池都支持把数据存储于HDFS, HBase。

**元数据**：两者使用相同的元数据。

**SQL解释处理**：比较相似都是通过词法分析生成执行计划。

**执行计划**：  
Hive: 依赖于MapReduce执行框架，执行计划分成map->shuffle->reduce->map->shuffle->reduce…的模型。如果一个Query会被编译成多轮MapReduce，则会有更多的写中间结果。由于MapReduce执行框架本身的特点，过多的中间过程会增加整个Query的执行时间。  
Impala: 把执行计划表现为一棵完整的执行计划树，可以更自然地分发执行计划到各个Impalad执行查询，而不用像Hive那样把它组合成管道型的map->reduce模式，以此保证Impala有更好的并发性和避免不必要的中间sort与shuffle。

**数据流**：  
Hive: 采用推的方式，每一个计算节点计算完成后将数据主动推给后续节点。  
Impala: 采用拉的方式，后续节点通过getNext主动向前面节点要数据，以此方式数据可以流式的返回给客户端，且只要有1条数据被处理完，就可以立即展现出来，而不用等到全部处理完成，更符合SQL交互式查询使用。

**内存使用：**  
Hive: 在执行过程中如果内存放不下所有数据，则会使用外存，以保证Query能顺序执行完。每一轮MapReduce结束，中间结果也会写入HDFS中，同样由于MapReduce执行架构的特性，shuffle过程也会有写本地磁盘的操作。  
Impala: 在遇到内存放不下数据时，当前版本1.0.1是直接返回错误，而不会利用外存，以后版本应该会进行改进。这使用得Impala目前处理Query会受到一定的限制，最好还是与Hive配合使用。Impala在多个阶段之间利用网络传输数据，在执行过程不会有写磁盘的操作（insert除外）。

**调度：**  
Hive: 任务调度依赖于Hadoop的调度策略。  
Impala: 调度由自己完成，目前只有一种调度器simple-schedule，它会尽量满足数据的局部性，扫描数据的进程尽量靠近数据本身所在的物理机器。调度器目前还比较简单，在SimpleScheduler::GetBackend中可以看到，现在还没有考虑负载，网络IO状况等因素进行调度。但目前Impala已经有对执行过程的性能统计分析，应该以后版本会利用这些统计信息进行调度吧。

**容错：**  
Hive: 依赖于Hadoop的容错能力。  
Impala: 在查询过程中，没有容错逻辑，如果在执行过程中发生故障，则直接返回错误（这与Impala的设计有关，因为Impala定位于实时查询，一次查询失败，再查一次就好了，再查一次的成本很低）。但从整体来看，Impala是能很好的容错，所有的Impalad是对等的结构，用户可以向任何一个Impalad提交查询，如果一个Impalad失效，其上正在运行的所有Query都将失败，但用户可以重新提交查询由其它Impalad代替执行，不会影响服务。对于State Store目前只有一个，但当State Store失效，也不会影响服务，每个Impalad都缓存了State Store的信息，只是不能再更新集群状态，有可能会把执行任务分配给已经失效的Impalad执行，导致本次Query失败。

**适用面：**  
Hive: 复杂的批处理查询任务，数据转换任务。  
Impala：实时数据分析，因为不支持UDF，能处理的问题域有一定的限制，与Hive配合使用,对Hive的结果数据集进行实时分析。

## 6. Impala的优缺点

**优点：**  
支持SQL查询，快速查询大数据。  
可以对已有数据进行查询，减少数据的加载，转换。  
多种存储格式可以选择（Parquet, Text, Avro, RCFile, SequeenceFile）。  
可以与Hive配合使用。  
**缺点：**  
不支持用户定义函数UDF。  
不支持text域的全文搜索。  
不支持Transforms。  
 不支持查询期的容错。  
对内存要求高。

## ****7.Impala shell****

 impala-shell  (进入impala)

connect hostname或者ip即可 （连接impala服务器）

show tables;  （其他命令和Hive一样）