Hive性能优化指南

caolei

Exported on 01/10/2020

Table of Contents

1 Hive是什么

关于Hive的定义，想必大家都都听过或看过一些。为了让之前没有接触过大数据的同学方便了解，这里也将给一个简单的定义：Hive 是一个基于 Hadoop 构建的开源数据仓库系统，我们使用它来查询和分析存储在 Hadoop 文件中的大型数据集。此外，通过使用 Hive，我们可以在 Hadoop 中处理结构化和半结构化数据。换句话说，Hive 是一个数据仓库基础设施，便于查询和管理驻留在分布式存储系统中的大型数据集。它提供了一种类 SQL 的查询语言 HiveQL（Hive Query Language）查询数据的方法。 此外，编译器在内部将 HiveQL 语句转换为 MapReduce、Tez、Spark 等作业。进一步提交给 Hadoop 框架执行。

2 Hive的存在价值

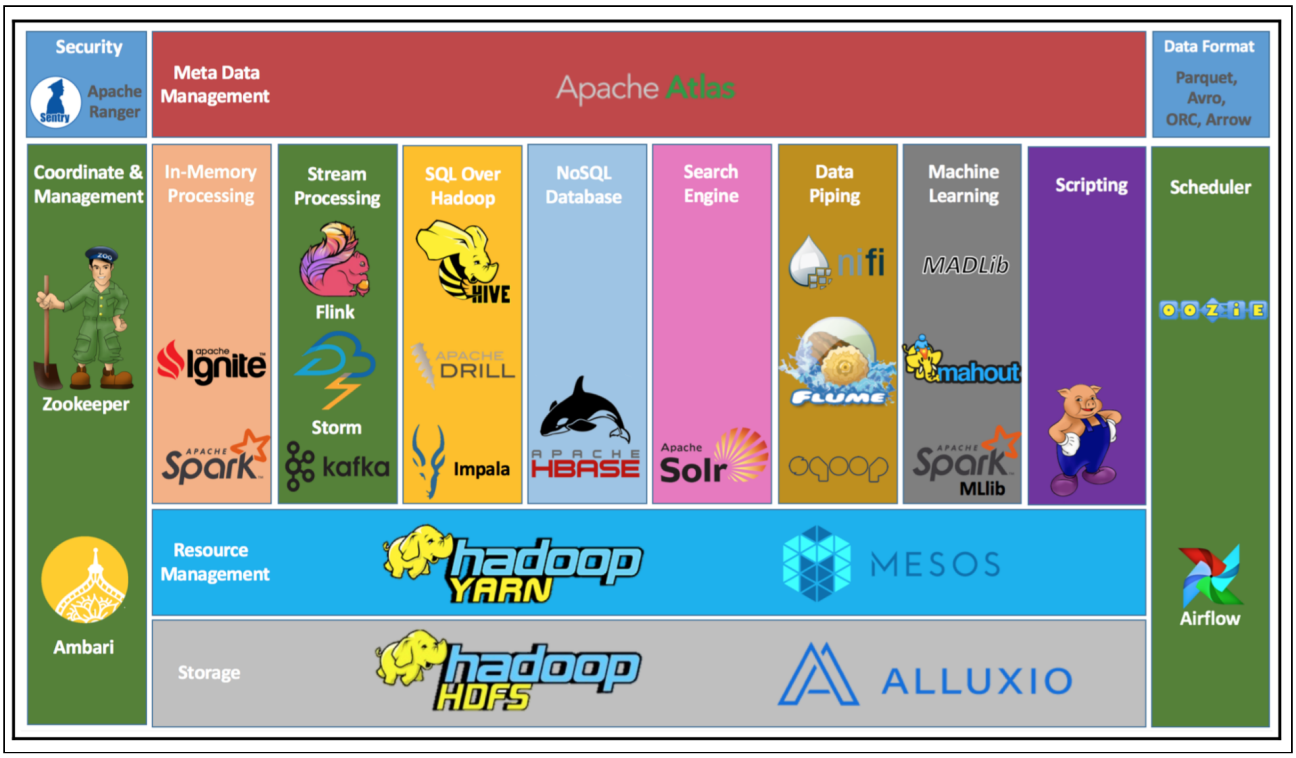
随着 Hadoop MapReduce 的出现，极大的简化大数据编程的难度，使得普通程序员也能从事开发大数据编程。但在生产活动中经常要对大数据计算分析是从事商务智能行业（BI）的工程师，他们通常使用 SQL 语言进行大数据统计及分析工作，而 Mapreduce 编程是有一定的门槛，如果每次都采用 MapReduce 开发计算分析任务，一是成本较高，二是效率太低，那么有没有更简单的办法，可以直接通过 SQL 在大数据平台下运行进行统计分析？有的，答案之一就是 Hive。

Hive 主要用于数据查询，统计和分析，提高开发人员的工作效率。Hive 通过内置函数将 SQL 语句生成 DAG（有向无环图），再让 Mapreduce 计算处理。从而得到我们想要的统计结果。而且在处理具有挑战性的复杂分析处理和数据格式时，极大的简化了开发难度。

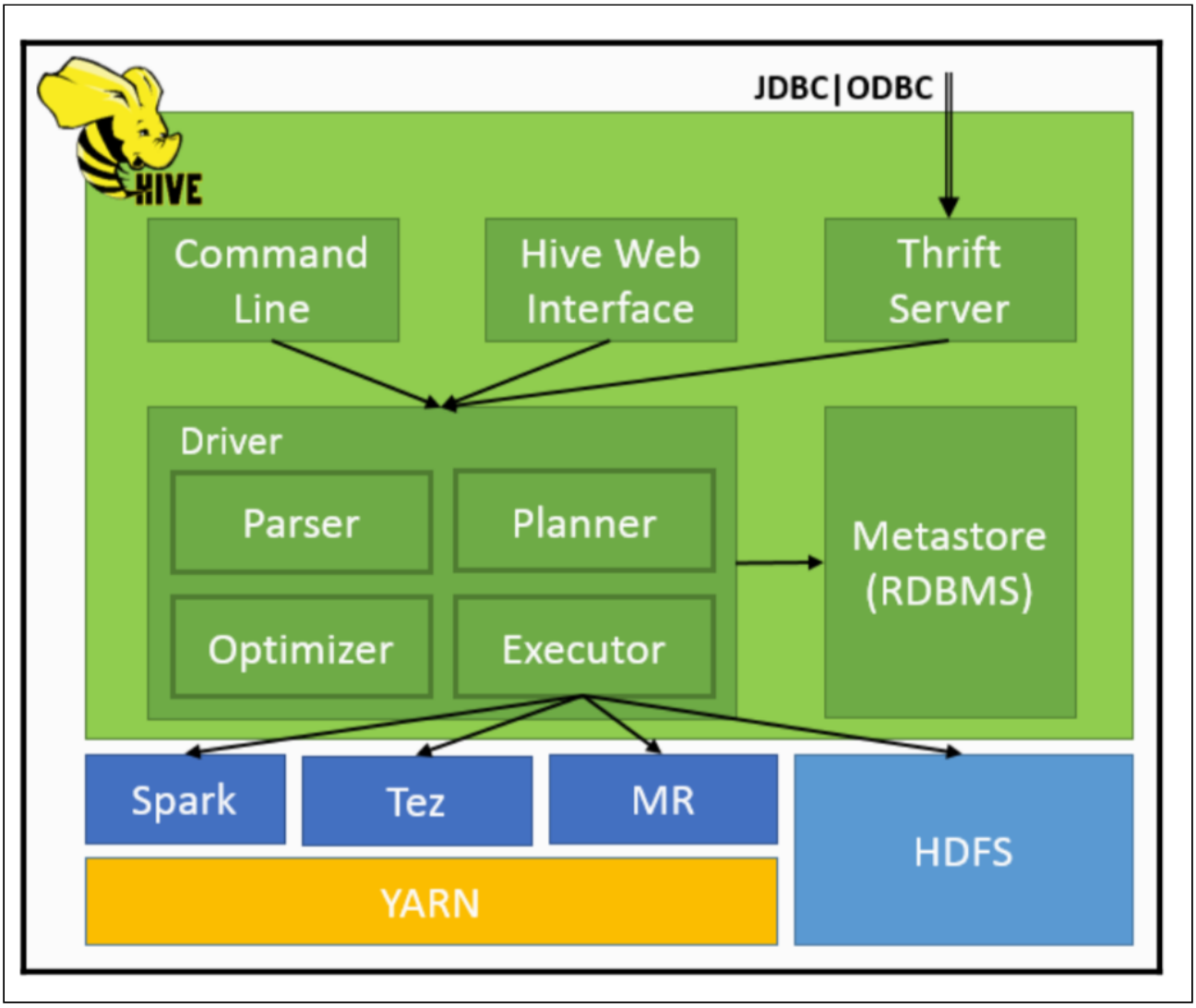
3 Hive架构

本文的中心是关于Hive性能优化，但是在开始之前，了解下Hive在Hadoop生态中的位置及Hive的基础架构有助于对后面性能优化部分的理解。

3.1 Hive在Hadoop生态的位置



3.2 Hive架构


JDBC|ODBC
HIVE
Command Line
Hive Web Interface
Thrift Server
Driver
Parser
Planner
Optimizer
Executor
Metastore (RDBMS)
Spark
Tez
MR
YARN
HDFS

4 性能优化

大数据生态系统中的各种组件的构建都会重点考量一个问题：性能，Hive也不例外。好的性能不仅能节省资源，而且还能得到用户的满意。关于Hive的性能考量，主要包含以下四个方面：

1. 性能工具
2. 设计优化
3. 数据优化
4. 任务优化

4.1 性能工具

如果要进行Hive性能调优，首先需要判断是否有性能问题，Hive提供了多种工具帮助用户检查和判断查询的性能情况，其中比较常用的工具包括EXPLAIN和ANALYZE语句。除此之外，查询执行日志包含了非常多的信息，对性能问题的定位与解决也非常有帮助。

4.1.1 EXPLAIN

Hive通过EXPLAIN命令来显示查询的执行计划。语法如下：

```
1 EXPLAIN [FORMATTED|EXTENDED|DEPENDENCY|AUTHORIZATION] hql_query
```

- FORMATTED：以JSON格式返回查询计划
- EXTENDED：提供执行计划关于操作的额外的信息，如文件名
- DEPENDENCY：执行计划信息中包含依赖信息，如表、分区等
- AUTHORIZATION：执行计划信息中包含此查询需要授权的对象

Hive查询会被转换成（这是一个有向无环图）阶段序列。这些阶段可能是mapper/reducer阶段，或者做metastore或文件系统的操作，如移动和重命名的阶段。 EXPLAIN的输出包括三个部分：

- 查询的抽象语法树(AST[1]), Hive使用的是一个名为ANTLR[2]的解析器生成器为HQL生成语法树。ANTLR[3]简单概括就是牛X强大。
- 执行计划的不同阶段之间的依赖关系
- 每个阶段的详细描述

每个阶段的描述信息包括一个关于操作符(operators)的序列，这些操作符还会关联一些元数据信息。元数据可能包括FilterOperator的筛选表达式、SelectOperator的选择表达式或FileSinkOperator的输出文件名。

4.1.2 ANALYSE

类似于Oracle的分析表，Hive中也提供了分析表和分区的功能，通过自动和手动分析Hive表，将Hive表的一些统计信息存储到元数据中。表和分区的统计信息主要包括：行数、文件数、原始数据大小、所占存储大小、最后一次操作时间等；

1 https://en.wikipedia.org/wiki/Abstract_syntax_tree
2 http://www.antlr.org/
3 http://www.antlr.org/

4.1.2.1 列统计

针对表中的列数，特定列数据的直方图，有多种方式可以实现。作为查询优化的一种方法，统计输入给优化器的代价函数，然后优化器比较不同的计划，并从中获取较优的计划。统计有时能够满足用户的查询，从而让用户，快速获取结果（需执行存储的统计信息，而不需要触发长时间的执行计划）

4.1.2.2 范围

Hive 现在支持的表和分区级别的统计， 这些统计会存在MetaStore中,统计项包括Number of Rows、Number of files、Size in Bytes、Number of Partitions等
比如分区：

4.2 设计优化

设计优化主要包括分区表设计、分桶表设计、索引设计、倾斜/临时表设计等。下面分别对这四种设计进行介绍。

4.2.1 分区表设计

分区表设计是提高查询性能的最有效方法之一，尤其是在体量较大的表上。带有分区筛选的查询只从指定的分区(子目录)加载数据，大大减少了加载的数据量，节省IO、CPU等资源。除了极少量的静态维度表可以不适用分区，数据仓库原则上都应该是分区表。

4.2.2 分桶表设计

与分区表类似，分桶表在HDFS中将数据组织到单独的文件中。如果Join操作的判断条件是分桶字段，那Join的性能将会被大大提升。选择更好的桶列可以使桶表Join执行得更好。

4.2.3 索引设计

使用索引是关系数据库中性能调优的一种常见的实践手段。目前Hive也支持在表/分区上创建索引。Hive中的索引提供一个基于键的数据视图，并为某些操作(如WHERE、GROUP BY和JOIN)提供更好的数据访问。使用索引总是比使用全表扫描性能更好。在HQL中创建索引的命令非常简单，如下所示,详细请参考Hive官方文档[4]。

```
1  CREATE INDEX index_name ON TABLE table_name (column_name) AS 'COMPACT' WITH DEFERRED REBUILD;
```

4.2.4 倾斜/临时表设计

除了使用常规的内部/外部或分区表之外，某些场景下我们还可以考虑使用倾斜或临时表，以获得更好的设计和性能。HQL支持创建一个特殊的表来组织倾斜的数据。倾斜表可以通过将这些倾斜的值自动分割到单独的文件或目录中来提高性能。结果，文件或分区文件夹的总数减少了。此外，查询可以快速有效地包含或忽略这些数据。下面是一个用来创建倾斜表的例子,详细请参考Hive官方文档[5]。

4 https://cwiki.apache.org/confluence/display/Hive/IndexDev
5 https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-SkewedTables

```
CREATE TABLE table_name (
        dept_no int,
        dept_name string
)
SKEWED BY (dept_no) ON (1000, 2000);
```

4.3 数据优化

数据优化主要包括对数据文件格式、压缩和存储方面的性能改进。

4.3.1 数据文件格式

Hive支持的文件格式比价多，主要有TEXTFILE, SEQUENCEFILE, AVRO, RCFILE, ORC, and PARQUET。用户不仅可以在创建表时指定数据文件格式，还可以后期修改数据文件格式。

```
CREATE TABLE ... STORE AS <file_format>:
```

```
ALTER TABLE ... [PARTITION partition_spec] SET FILEFORMAT
```

一旦创建了以文本格式存储的表，我们就可以直接将文本数据加载到其中。要将文本数据加载到具有其他文件格式的表中，我们可以先将数据加载到存储格式为文本的表中，然后使用INSERT OVERWRITE/ into table…从中选择数据，然后将数据插入具有其他文件格式的表中。

TEXT, SEQUENCE, and AVRO等面向行的文件存储格式虽然比较易于理解，但是缺陷也显而易见，如即使查询结果只包含一列，查询也必须读取完整的一行。为了解决这个问题，RCFILE、ORC、PARQUET 等行列混合文件存储方案诞生。

根据所使用的技术栈，如果Hive是用于定义或处理数据的主要工具，建议使用ORC格式。如果您在数据生态系统中使用多种工具，那么PARQUET在适应性方面是更好的选择。

4.3.2 数据压缩

Hive中的压缩技术可以通过适当地压缩中间和最终输出数据，大大减少mappers和reducers之间的数据传输量，因此，查询将具有更好的性能。要压缩多个MapReduce作业之间生成的中间文件，我们需要在命令行会话或hive-site.xml文件中设置以下属性(默认为false):

```
SET hive.exec.compress.intermediate=true;
```

下面列举了一些常用的数据压缩选项

压缩选项	对应的设置类	文件后缀	是否可分割
Deflate	`org.apache.hadoop.io.compress.DefaultCodec`	.deflate	N

Gzip	org.apache.hadoop.io.compress.GzipCodec	.gz	N
Bzip2	org.apache.hadoop.io.compress.BZip2Codec	.gz	Y
LZO	com.apache.compression.lzo.LzopCodec	.lzo	N
LZ4	org.apache.hadoop.io.compress.Lz4Codec	.lz4	N
Snappy	org.apache.hadoop.io.compress.SnappyCodec	.snappy	N

常用的压缩命令如下(可以在hive- site.xml或者命令行设置)

命令	描述
SET hive.intermediate.compression.codec= org.apache.hadoop.io.compress.SnappyCodec	中间结果压缩会为需要多个MapReduce作业的特定作业节省磁盘空间
SET hive.exec.compress.output=true	输出结果进行压缩
SET mapreduce.output.fileoutputformat.compress.codec= org.apache.hadoop.io.compress.SnappyCodec	当hive.exec.compress.output被设置成true后，属性mapreduce.output.fileoutputformat.compress.codec的值作为数据压缩格式

4.3.3 数据存储

数据根据使用频率可以简单分为冷热数据。如果我们能提高在热数据上的查询性能，其实整体的数据查询性能将有较大的改善。那么如何提高热数据的查询性能呢？其中一个方法就是增加数据的replication值，增大replication值在一定程度上会减少数据在集群节点之间的传输过程，进而提高性能。

```
1 hdfs dfs -setrep -R -w 4 /user/hive/warehouse/dwd.db/dwd_putong_profile_yay_private_question_a_d
```

任何事情都是有限度的，过多的冗余文件会使namenode的内存耗尽，特别是大量的小文件。Hadoop本身已经有一些解决方案，可以通过以下方式处理很多小文件问题:

- Hadoop Archive/HAR:对小文件进行打包归档
- 数据采用SEQUENCEFILE格式进行存储，一定程度上可以将小文件压缩成大文件
- HDFS Federation支持多个namenode来管理更多的文件
- 自己开发一个文件合并程序来合并HDFS中的小文件

针对Hive的数据存储优化，可以设置下列参数来避免创建更多的小文件

- hive.merge.mapfiles: 将在仅使用map的作业结束时合并小文件
- hive.merge.mapredfiles: 在MapReduce作业结束时合并小文件
- hive.merge.size.per。定义作业结束时合并文件的大小

- hive.merge.smallfiles.avgsize: 触发文件合并的阈值

4.4 作业优化

作业优化主要关注运行模式、JVM 重用、作业并行和Join查询优化等方面。其中还会介绍下执行引擎和优化器相关的概念。

4.4.1 运行模式

大多数的Hive查询是需要hadoop集群来处理大数据的，不过，有时Hive的输入数据量是非常小的。在这种情况下，为查询执行任务的时间消耗可能会比实际job的执行时间要多的多，因此hive0.7版本后Hive开始支持任务执行选择本地模式(local mode)，即任务提交到本地机器处理。如此一来，对数据量比较小的操作，就可以在本地执行，这样要比提交任务到集群执行效率要快很多。本地模式的配置命令如下：

```
1 set hive.exec.mode.local.auto=true;
```

需要注意是的执行本地模式需要满足几个条件：

- 作业的总输入大小低于：hive.exec.mode.local.auto.inputbytes.max，默认为128MB
- map任务的总数小于：hive.exec.mode.local.auto.tasks.max，默认为4
- 所需的reduce任务总数为1或0。

4.4.2 JVM 重用

默认情况下，Hadoop为每个map或reduce作业启动一个新的JVM，并行运行map或reduce任务。当在map或reduce作业是只运行几秒钟的轻量级作业时，JVM启动过程可能会带来很大的开销。Hadoop有一个重用JVM的选项，通过共享JVM来串行运行mapper/reducer，而不是并行运行。JVM重用适用于映射或减少同一作业中的任务。来自不同作业的任务总是在单独的JVM中运行。为了支持重用，我们可以使用以下属性为JVM重用设置单个作业的最大任务数。它的默认值是1。如果设置为-1，则没有限制:

```
1 SET mapreduce.job.jvm.numtasks=5;
```

4.4.3 作业并行

Hive查询通常被转换成许多按默认顺序执行的阶段。这些阶段并不总是相互依赖的。相反，它们可以并行运行，以减少整个作业的运行时间。我们可以通过以下设置启用此功能，并设置并行运行作业的预期数量。并行执行将提高集群利用率。如果集群的利用率已经非常高，那么并行执行对总体性能没有多大帮助。

```
1 SET hive.exec.parallel=true;
2 SET hive.exec.parallel.thread.number=16;
```

4.4.4 Join查询优化

Hive提供的多种类型的Join，每种Join类型的优化不太一样。

4.4.4.1 Reduce端Join

其实我们经常用的就是这种Join, 对于这种Join类型，我们需要确保数据量大的表位于Join的最右边。

4.4.4.2 Map join

当其中一个用于Join的表小到足以完全装入内存时，使用Map Join的速度会更快，缺点就是受表大小的限制。设置Map Join的相关命令如下：

```
SET hive.auto.convert.join=true;
SET hive.mapjoin.smalltable.filesize=600000000;
SET hive.auto.convert.join.noconditionaltask=true;
SET hive.auto.convert.join.noconditionaltask.size=10000000;
```

启用Join类型自动转换后，Hive将自动检查较小的表文件大小是否大于 **Hive .mapjoin.smalltable** 指定的值。如果文件大小小于此阈值，则尝试将Join的类型转换为Map Join。

4.4.4.3 Bucket Map Join

和Map Join类似，只不过用于分桶表而已。需要注意的是：在 Bucket Map Join中，所有连接表都必须是桶表和桶列上的连接。此外，较大表中的桶号必须是较小表中的桶号的倍数。

4.4.4.4 Skew Join

在处理分布高度不均匀的数据时，数据倾斜可能以这样一种方式发生，即少数计算节点必须处理大量计算。如果发生数据倾斜，则通知Hive进行适当的优化:

```
SET hive.optimize.skewjoin=true;   --If there is data skew in join, set it to true. Default is false.
SET hive.skewjoin.key=100000;      --This is the default value. If the number of key is bigger than --this, the new keys will send to the other unused reducers.
```

4.4.5 任务执行引擎

Hive支持在不同的引擎上运行作业。引擎的选择也会影响整体性能。然而，与其他设置相比，这是一个更大的变化。可选的执行引擎包括mr、spark、tez

```
SET hive.execution.engine=<engine>; -- <engine> = mr|tez|spark
```

4.4.6 优化器

与关系数据库类似，Hive在提交最终执行之前会生成并优化每个查询的逻辑和物理执行计划。目前Hive中存在两种主要的优化器：即向量化优化(Vectorization optimization)和基于成本的优化(Cost-based optimization)。

4.4.6.1 Vectorization optimization

应用场景主要是在同时处理大量数据时，而不是逐行处理数据，详细内容参考https://cwiki.apache.org/confluence/display/Hive/Vectorized+Query+Execution，开启的配置如下

```
SET hive.vectorized.execution.enabled=true;
```

4.4.6.2 Cost-based optimization(CBO)

CBO是通过Apache Calcite[6]实现的。Hive CBO通过检查查询成本(由ANALYZE语句或metastore本身收集)来生成高效的执行计划，最终减少查询执行时间和资源利用率。要使用CBO，设置以下属性:

```
SET hive.cbo.enable=true; -- default true after v0.14.0
SET hive.compute.query.using.stats=true; -- default false
SET hive.stats.fetch.column.stats=true; -- default false
SET hive.stats.fetch.partition.stats=true; -- default true
```

6 https://calcite.apache.org/