**Spark-1.3.1与Hive整合实现查询分析**

[2015-05-14 21:42:08](http://shiyanjun.cn/archives/1113.html)    [Yanjun](http://shiyanjun.cn/archives/author/yanjun" \o "查看作者 Yanjun 的全部文章)

在大数据应用场景下，使用过Hive做查询统计分析的应该知道，计算的延迟性非常大，可能一个非常复杂的统计分析需求，需要运行1个小时以上，但是比之于使用MySQL之类关系数据库做分析，执行速度快很多很多。使用HiveQL写类似SQL的查询分析语句，最终经过Hive查询解析器，翻译成Hadoop平台上的MapReduce程序进行运行，这也是MapReduce计算引擎的特点带来的延迟问题：Map中间结果写文件。如果一个HiveQL语句非常复杂，会被翻译成多个MapReduce Job，那么就会有很多的Map输出中间结果数据到文件中，基本没有数据的共享。  
如果使用Spark计算平台，基于Spark RDD数据集模型计算，可以减少计算过程中产生中间结果数据写文件的开销，Spark会把数据直接放到内存中供后续操作共享数据，减少了读写磁盘I/O操作带来的延时。另外，如果基于Spark on YARN部署模式，可以充分利用数据在Hadoop集群DataNode节点的本地性（Locality）特点，减少数据传输的通信开销。

**软件准备**

我把使用的相关软件的版本在这里列出来，以便测试验证，如下所示：

* CentOS-6.6 (Final)
* JDK-1.7.0\_25
* Maven-3.2.1
* Hadoop-2.2.0
* Spark-1.3.1
* Hive-0.12.0
* MySQL-Server-5.5.8

另外还要搭建好Hadoop集群，以及安装配置好Hive客户端，能够在Hive上正确执行查询分析，安装过程不再累述，可以参考网上很多文档。由于我们使用最新版本的Spark-1.3.1，为了使用我们现有2.2.0版本的Hadoop平台，所以需要重新编译构建Spark程序，接下来会做详细说明。  
这里，给出使用的各个集群环境的结构拓扑，如下表所示：

|  |  |  |
| --- | --- | --- |
| **Source节点** | **服务名称** | **说明** |
| hadoop1 | Spark Master/Spark Driver | Spark集群 |
| hadoop2 | DataNode/NodeManager | Hadoop集群 |
| hadoop3 | DataNode/NodeManager | Hadoop集群 |
| hadoop4 | Hive | Hive客户端 |
| hadoop5 | Spark Worker | Spark集群 |
| hadoop6 | Spark Worker/NameNode/ResourceManager/Secondary NameNode | Spark集群/Hadoop集群 |
| 10.10.4.130 | MySQL | 用于存储Hive元数据 |

上述节点配置相同，因为是测试机，所以配置相对比较低。我们是分别将Spark集群和Hadoop集群的Worker和NodeManager/DataNode分开部署了，在使用Spark做计算的时候，就没有数据本地性（Locality）的特性，所以如果基于Spark on YARN的模式，可能会获得更好地计算性能的提升。

**Spark编译安装配置**

首先从官网下在Spark源码文件：

|  |  |
| --- | --- |
| 1 | cd ~/ |
| 2 | wget <http://mirror.bit.edu.cn/apache/spark/spark-1.3.1/spark-1.3.1.tgz> | |

|  |  |
| --- | --- |
| 3 | tar xvzf spark-1.3.1.tgz |
| 4 | mv spark-1.3.1 spark-1.3.1-bin-hadoop2.2 | |

我的环境是JDK 1.7，使用Maven构建，执行如下命令行：

|  |  |  |
| --- | --- | --- |
| 1 | export MAVEN\_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m" | |
| 2 | cd /home/spark/spark-1.3.1-bin-hadoop2.2/ |

|  |  |
| --- | --- |
| 3 | mvn -Pyarn -Dyarn.version=2.2.0 -Phadoop-2.2 -Dhadoop.version=2.2.0 -Phive -Phive-0.12.0 -Phive-thriftserver -DskipTests clean package |

编译构建完成以后，可以看到如下内容：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | /home/spark/spark-1.3.1-bin-hadoop2.2/assembly/target/scala-2.10/spark-assembly-1.3.1-hadoop2.2.0.jar | | |
| 2 | | /home/spark/spark-1.3.1-bin-hadoop2.2/lib\_managed/\*.jar |

如果网络状况不好，可能无法构建成功。  
另外，也可以使用sbt构建，执行如下命令：

|  |  |
| --- | --- |
| 1 | cd /home/spark/spark-1.3.1-bin-hadoop2.2/ |
| 2 | build/sbt -Pyarn -Phadoop-2.2 -Dhadoop.version=2.2.0 -Phive -Phive-0.12.0 -Phive-thriftserver assembly | |

如果失败，多试几次可能会以成功。  
使用Maven构建与使用sbt构建，都要耗费很长时间，而且最终生成的文件可能会有所不同。

下面，我们配置Spark集群，首先在Spark Master节点上配置，修改配置文件conf/slaves，将Worker节点主机名加入进去，一行一个，内容如下所示：

|  |  |
| --- | --- |
| 1 | hadoop5 |
| 2 | hadoop6 |

修改Spark环境变量配置文件conf/spark-env.sh，增加如下配置行：

|  |  |
| --- | --- |
| 1 | SPARK\_MASTER\_IP=hadoop1 |

修改配置文件spark-defaults.conf，内容如下所示：

|  |  |  |
| --- | --- | --- |
| 1 | | spark.eventLog.enabled           true |
| 2 | spark.eventLog.dir              <hdfs://hadoop6:8020/spark/logs/events> | | |

登录到Hive安装的节点，将Hive的配置文件拷贝到Spark安装目录下的conf目录下面，执行如下命令：

|  |  |
| --- | --- |
| 1 | scp /usr/local/hive/conf/hive-site.xml spark@hadoop1:/home/spark/spark-1.3.1-bin-hadoop2.2/conf/ |

最后分发Spark安装文件到Spark Worker节点上：

|  |  |
| --- | --- |
| 1 | sudo scp -r /home/spark/spark-1.3.1-bin-hadoop2.2 spark@hadoop5:/home/spark/spark-1.3.1-bin-hadoop2.2/ |
| 2 | sudo scp -r /home/spark/spark-1.3.1-bin-hadoop2.2 spark@hadoop6:/home/spark/spark-1.3.1-bin-hadoop2.2/ |

为了方便启动Spark集群，可以配置Spark Master到Workers的ssh免密码登录，然后只需要在Master中执行如下脚本即可：

|  |  |
| --- | --- |
| 1 | sbin/start-all.sh |

可以查看Spark各个节点的服务启动情况，也可以通过Spark UI链接进入页面查看<http://hadoop1:8080/>，默认是8080端口，如果8080端口已经被占用，Spark会自动选择端口号数字加1，如<http://hadoop1:8081/>。

**Spark+Hive整合**

我们知道，在使用Hive进行查询的时候，到底层MapReduce计算层会将HiveQL翻译成MapReduce程序，在Hadoop平台上执行计算，这使得计算的延迟比较大。我们整合Spark和Hive，就是通过Spark平台来计算Hive查询，也就是Hive不再使用它默认的MapReduce计算引擎，Spark会直接读取Hive的元数据存储，将Hive数据转换成Spark RDD数据，通过Spark提供的计算操作来实现（Transformation和Action）。  
我们首先在Hive中创建一个数据库event\_db，执行如下命令：

|  |  |
| --- | --- |
| 1 | CREATE DATABASE event\_db; |

在创建一个Hive外部表user\_event，执行DDL脚本：

|  |  |  |
| --- | --- | --- |
| 01 | CREATE EXTERNAL TABLE event\_db.user\_event( | |
| 02 | appid string, |

|  |  |
| --- | --- |
| 03 | event\_code string, |

|  |  |
| --- | --- |
| 35 | ip bigint, |
| 36 | area\_code int, | |

|  |  |  |
| --- | --- | --- |
| 37 | create\_time string) | |
| 38 | PARTITIONED BY ( |

|  |  |  |
| --- | --- | --- |
| 39 | create\_date string) | |
| 40 | ROW FORMAT DELIMITED |

|  |  |  |
| --- | --- | --- |
| 41 | FIELDS TERMINATED BY '\t' | |
| 42 | STORED AS INPUTFORMAT |

|  |  |  |
| --- | --- | --- |
| 43 | 'org.apache.hadoop.mapred.TextInputFormat' | |
| 44 | OUTPUTFORMAT |

|  |  |  |
| --- | --- | --- |
| 45 | 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' | |
| 46 | LOCATION |

|  |  |
| --- | --- |
| 47 | '<hdfs://hadoop6:8020/hive/event_db/user_event>'; |

我选择了一天的用户事件数据（大概有5G左右，13824560条记录），将数据加载到Hive的分区中，执行如下LOAD命令行：

|  |  |
| --- | --- |
| 1 | LOAD DATA LOCAL INPATH'/home/shirdrn/data/user\_event\_20150511.log' OVERWRITE INTO TABLEevent\_db.user\_event PARTITION (create\_date='2015-05-11'); |

* Standalone模式

我们可以通过指定SPARK\_CLASSPATH变量，将需要访问Hive的元数据存储MySQL的驱动包加入进去，然后直接启动Spark SQL Shell即可。这里，使用Spark默认的集群管理模式Standalone，启动Shell时需要指定master选项为Spark Master服务连接：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | SPARK\_CLASSPATH="$SPARK\_CLASSPATH:/home/spark/spark-1.3.0-bin-hadoop2.2/lib\_managed/jars/mysql-connector-java-5.1.34.jar" | | |
| 2 | | bin/spark-sql --master <spark://hadoop1:7077> |

这样我们可以直接在Spark SQL Shell上输入Hive查询语句就可以执行查询分析计算。  
另外，还可以通过Spark Shell进行操作，不过需要了解Spark SQL支持的Scala API，启动Spark Shell，执行如下命令：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | SPARK\_CLASSPATH="$SPARK\_CLASSPATH:/home/spark/spark-1.3.0-bin-hadoop2.2/lib\_managed/jars/mysql-connector-java-5.1.34.jar" | | |
| 2 | | bin/spark-shell --master <spark://hadoop1:7077> |

然后，创建一个org.apache.spark.sql.hive.HiveContext对象，用来执行Hive查询：

|  |  |  |
| --- | --- | --- |
| 1 | | scala> val sqlContext = neworg.apache.spark.sql.hive.HiveContext(sc) |
| 2 | sqlContext: org.apache.spark.sql.hive.HiveContext =org.apache.spark.sql.hive.HiveContext@6dcc664b | |

接着可以执行查询：

|  |  |
| --- | --- |
| 1 | scala> sqlContext.sql("SELECT area\_code,event\_code,COUNT(udid) AS user\_cnt FROM event\_db.user\_event WHERE create\_date='2015-05-11' GROUP BY area\_code,event\_code LIMIT 10").collect().foreach(println) |

可以看到查询结果。

* yarn-client模式

如果基于YARN模式运行（与Hive整合只支持yarn-client模式，不支持yarn-cluster），需要指定Hadoop集群的环境变量（在当前Driver节点上必须有Hadoop的安装文件），如下所示：

|  |  |
| --- | --- |
| 1 | export HADOOP\_HOME=/usr/local/hadoop-2.2.0 |
| 2 | export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop | |

然后启动Spark SQL Shell，执行如下命令：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | SPARK\_CLASSPATH="$SPARK\_CLASSPATH:/home/spark/spark-1.3.0-bin-hadoop2.2/lib\_managed/jars/mysql-connector-java-5.1.34.jar" | | |
| 2 | | bin/spark-sql --master yarn-client |

* 查询结果耗时比较

我们使用Hive，以及上面提到的两种模式分别执行如下HiveQL查询统计语句：

|  |  |
| --- | --- |
| 1 | SELECT area\_code,event\_code,COUNT(DISTINCT udid) AS user\_cnt FROMevent\_db.user\_event WHERE create\_date='2015-05-11' AND(create\_time BETWEEN '2015-05-11 17:00:00' AND '2015-05-11 23:30:00') GROUP BY area\_code,event\_code ORDER BY user\_cnt DESCLIMIT 10 |

可以看到查询结果，结果如下所示：

|  |  |  |
| --- | --- | --- |
| 01 | 156000000     100003     8290 | |
| 02 | 110000     100003     7832 |

|  |  |
| --- | --- |
| 03 | 440100     100003     4956 |
| 04 | 110000     100010     3850 |

|  |  |
| --- | --- |
| 05 | 440300     100003     3709 |
| 06 | 320100     100003     3683 |

|  |  |
| --- | --- |
| 07 | 410100     100003     3669 |
| 08 | 110000     101014     3479 |

|  |  |
| --- | --- |
| 09 | 110000     200004     3455 |
| 10 | 110000     100011     3423 |

对比耗时，如下表所示：

|  |  |
| --- | --- |
| **运行模式** | **花费时间（秒）** |
| Hive | 189.695 |
| Spark Standalone | 82.895 |
| Spark yarn-client | 104.259 |

可见，无论是Spark Standalone模式还是Spark yarn-client模式，耗时都比直接执行Hive查询要少得多。我们执行Spark计算，2个Worker节点上各用了一个Executor，每个Executor使用512M内存，如果增加Executor个数，或者调大内存，应该比上面运行耗时更少，例如，启动Spark SQL Shell并指定相关参数：

|  |  |
| --- | --- |
| 1 | bin/spark-sql --master <spark://hadoop1:7077> --driver-memory 1G --driver-cores 2 --executor-memory 4G |

或者：

|  |  |
| --- | --- |
| 1 | bin/spark-sql --master yarn-client --driver-memory 1G --driver-cores 2 --executor-cores 4 --num-executors 8 --executor-memory 4G |

**总结**

根据上面我们实践的整合Spark+Hive，在执行复杂统计分析时，完全可以使用Spark SQL来替代Hive，至少会提高几倍的速度，对于一些基于Hive统计应用，可能每天晚上要执行6个小时以上的统计计算，导致第二天结果数据都无法出来，如果统计需求再次增加，可能时间还会更长。除了对Hive查询语句进行优化之外，应该说优化空间不大，所以这个时候可以考虑使用Spark平台来实现统计分析，而且，Spark集群可以线性扩展，对于一些调优也更容易一些。  
另外，Spark的发展超级迅猛，新版本频繁发布，而且在后期的版本中还会在性能方面进行大幅改进。Tungsten项目将是Spark自诞生以来内核级别的最大改动，以大幅度提升Spark应用程序的内存和CPU利用率为目标，旨在最大程度上压榨新时代硬件性能。Tungsten项目包括了3个方面的努力：

* Memory Management和Binary Processing：利用应用的语义（Application Semantics）来更明确地管理内存，同时消除JVM对象模型和垃圾回收开销。
* Cache-aware Computation（缓存友好的计算）：使用算法和数据结构来实现内存分级结构（Memory Hierarchy）。
* 代码生成（Code Generation）：使用代码生成来利用新型编译器和CPU。

Tungsten将大幅度提升Spark的核心引擎，在Spark 1.4版本，会包括Dataframe API中聚合操作的内存管理，以及定制化序列化器。在Spark 1.5版本中，会有部分项目（基于DataFrame模型）包括二进制内存管理的扩展和Cache-aware数据结构。