Rhive 技术文档手册

智能分析组

January 29, 2013

1 Rhive 简介

Rhive是通过hive在R中实现分布式计算的功能，可以在R中使用HQL，而且允许非常方便的在R中使用Hive的功能,Rhive中通过hiveserve 和Rserver让不同的节点之间快速，方便的通信，hive的主要用户是面向那些熟悉数据建模，并且熟悉SQL的数据分析师，可以快速的对存储hdfs的海量数据进行建模和分析，Rhive的主要用户就是那些既熟悉R又熟悉SQL的用户，通过将功能强大，算法丰富的R通hadoop平台的工具结合起来可以快速实现分布式建模来处理数据。

2 Rhive中的几个组件及安装过程

2.1 需要的配套的软件版本

- Java 1.6
    - R 2.13.0
    - Rserve 0.6-0
    - rJava 0.9-0
    - Hadoop 0.20.x (x >= 1)
    - Hive 0.8.x (x >= 0)

Rhive是将R同hive结合起来的工具，hive是在hadoop平台上运行的，所以首先必须安装hadoop，由于hadoop的版本发布非常频繁，从事hadoop开发的社区异常活跃，造成hadoop版本比较混乱，hive对hadoop的版本要求比高，所以必须选择合适的hadoop版本，经过反复测试最终选择比较稳定的版本是

- Hadoop 0.20.1
- Hive 0.8.0

2.2 Hadoop的安装主要过程是

1.安装需要的软件：JAVA ， SSH
2.配置SSH免密钥登录
3.配置系统信息

核心的配置主要是这五个文件：core-site.xml ，hdfs-site.xml ，mapred-site.xml ,slaves ,masters . 另外还有环境变量的配置，并将配置好的文件发送到各节点相同的目录下

2.3 Hive的安装过程

1.配置环境变量
2.配置系统信息文件

默认的系统信息文件主要在conf/下的hive-default.xml 文件中，将其改为hive-site.xml，并修改其中的配置（可以使用外部数据库，一般使用mysql数据库存储metadata信息）Rhive的安装主要过程:

2.4 安装R和相关的包

需要在所有节启动tasktracker的点上安装R，一般是在所有节点上安装好R。还要在所有节点安装好rJava包和Rserve包，在使用Rhive之前必须先启动所有节点的Rserver服务。

2.5 注意事项

在安装过程中100%会遇到各种各样的问题，比如版本不兼容，SSH免密钥登录没有配置通，Rserve包安装不上，Rjava包安装不上，Rserve启动不了，R程序安装过程中编译出现错误.一下罗列一些需要注意的地方

(1)RHive 依赖于Rserve，因此在安装R的时候有些变化:

```
./configure --disable-nls --enable-R-shlib
make
make install
```

enable-R-shlib 是将R作为动态库进行安装，这样像Rserve依赖于R动态库的包就可以安装了，但缺点是会有20%左右的性能下降。

(2)SSH免密钥登录原理在Hadoop系统在工作过程中作为Master的主机必须可以免密钥登录到其他机器中去， Master（NameNode | JobTracker）作为客户端，要实现无密码公钥认证，连接到服务器Salve（DataNode | Tasktracker）上时，需要在Master上生成一个密钥对，包括一个公钥和一个私钥，而后将公钥复制到所有的Slave上。当Master通过SSH连接Salve时，Salve就会生成一个随机数并用Master的公钥对随机数进行加密，并发送给Master。Master收到加密数之后再用私钥解密，并将解密数回传给Slave，Slave确认解密数无误之后就允许Master进行连接了。这就是一个公钥认证过程，其间不需要用户手工输入密码。重要过程是将客户端Master复制到Slave上。在Master节点上执行以下命令:

ssh-keygen –t rsa –P

通过这条命令生成其无密码密钥对，询问其保存路径时采用其默认路径，生成的密钥对保存在”/home/hadoop/.ssh”目录下。接着在所有节点上做如下配置就是把Master的公复制到每个节点上，然后再把id_rsa.pub追加到授权的key里面去。

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

这样就实现了从Master免密钥登录各个Slaves访问,接着就要做的是让各个Slaves节点实现免密钥访问Master主机，过程基本相同。

(3) 在安装Rjava过程中一般会遇到一个java环境变量问题，解决方法就是:
加入两个环境变量:

```
export JAVA_HOME$=/usr/lib/jvm/java-6-openjdk-amd64/jre
export LD_LIBRARY_PATH=/usr/lib/jvm/java-1.6.0-openjdk/jre/lib/amd64/server
```

具体的值可能因安装Java的位置而有所差异。除此之外，还可能出现很多问题，只要将异常在google中搜索都能找到相应的解决方案。
这里有几个比较好的网络资源:
关于Hadoop安装配置：http://www.cnblogs.com/xia520pi/archive/2012/05/16/2503949.html
关于Hive安装配置：http://www.cnblogs.com/tangtianfly/archive/2012/03/12/2391594.html
关于Rhive安装配置：http://www.bjt.name/2012/12/RHive-install/

3 Rhive的简介

Hive的设计目的是为了让精通SQL技能，但是Java编程技能相对较弱的分析师能够在facebook存放在HDFS中的大规模数据集上运行查询。Hive 是一个构建在Hadoop上的数据仓库框架，Hive是应Facebook每天产生的海量新兴网络数据进行管理和机器学习的需求和发展的。

3.1 hive的定义

Hive is a data warehouse infrastructure built on top of Hadoop.It provides tools to enable easy data ETL, a mechanism to put structures on the data, and the capability to querying and analysis of large data sets stored in Hadoop files. Hive defines a simple SQL-like query language, called QL, that enables users familiar with SQL to query the data. At the same time, this language also allows programmers who are familiar with the MapReduce fromwork to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language. Hive does not mandate read or written data be in the "Hive format"—there is no such thing. Hive works equally well on Thrift, control delimited, or your specialized data formats. Please see File Format and SerDe in Developer Guide for details.

基于hadoop搭建,是一种数据抽取，数据转换，数据清洗，数据载入ETL工具.为了分析存储在handoop中的大数据集定义了一种类似于SQL的查询语言QL，使用MapReduce作为其基本框架。

3.2 hive描述

Tools to enable easy data extract/transform/load (ETL)
A mechanism to impose structure on a variety of data formats
Access to files stored either directly in Apache HDFS (TM) or in other data storage systems such as Apache HBase (TM)
Query execution via MapReduce

3.3 Rhive

RHive 是一种通过HIVE高性能查询来扩展R计算能力的包。它可以在R环境中非常容易的调用HQL，也允许在Hive中使用R的对象和函数。理论上数据处理量可以无限扩展的Hive平台，搭配上数据挖掘的利器R环境，堪称是一个完美的大数据分析挖掘的工作环境, 就目前的使用看来Rhive最大的有点就是非常方便的从hdfs存储的结构化数据中查询需要的数据。

4 HiveQL

4.1 基本操作

(1) 创建表

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type
[COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)]
INTO num_buckets BUCKETS]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path]
```

CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用IF NOT EXIST 选项来忽略这个异常。EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION），Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。LIKE 允许用户复制现有的表结构，但是不复制数据。用户在建表的时候可以自定义SerDe 或者使用自带的SerDe。如果没有指定ROW FORMAT 或者ROW FORMAT DELIMITED，将会使用自带的SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的SerDe，Hive > 通过SerDe 确定表的具体的列的数据。如果文件数据是纯文本，可以使用STORED AS TEXTFILE。如果数据需要压缩，使用STORED AS SEQUENCE 。有分区的表可以在创建的时候使用PARTITIONED BY 语句。

一个表可以拥有一个或者多个分区，每一个分区单独存在一个目录下。而且，表和分区都可以对某个列进行CLUSTERED BY 操作，将若干个列放入一个桶（bucket）中。也可以利用SORT BY 对数据进行排序。这样可以为特定应用提高性能。表名和列名不区分大小写，SerDe 和属性名区分大小写。表和列的注释是字符串。

表名和列名不区分大小写，SerDe 和属性名区分大小写。表和列的注释是字符串。

(2) 载入与查询

```
create table records(id string ,name string , no int )
    load data local inpath 'input/sample.txt'
    overwrite into table records ;
```

运行这一命令行的目的告诉Hive把指定的本地文件放到它的仓库目录中。这只是一个简单的文件系统操作。这个操作并不解析文件或把它存储为内部数据库格式。这是因为Hive并不强行使用某种特定的文件格式。文件以原样逐字存储：Hive并不对文件进行修改。Hive默认文件系统是hdfs，如果要载入存储在hdfs中的文件，去掉关键字local即可。

load中的OVERWRITE关键字告诉Hive删除表所对应目录中已有的所有文件。如果省去这一关键字，Hive就简单地把新的文件加入目录（除非目录下正好有一个同名

的文件，否则替换掉原有的同名文件）。现在数据已经在Hive中，可以使用一个查询

```
select year ,max (no) from records where no!=100
and (quality=0 or quality =1 or quality =5 or quality =9 )
group by year ;
```

在Hive中每个查询就是一个MapReduce过程，这也是Hive最大的优势，将标准查询转化为MapReduce过程。

以上就是常用到的Hive的操作。

5 Hive服务

Hive的外壳环境是可以使用hive命令来运行的一项服务。可以在运行时使用−−service选项指明要使用那种服务。键入hive −−service −−help 可以或得可用的服务列表。下面介绍一些有用的服务。

cli Hive的命令行接口（外壳环境）。这是默认的服务。hiveserver 让Hive以提供Trift 服务的服务器形式运行，允许用不同的的语言编写的客户端进行访问。使用Thrift,JDBC,和ODBC连接器的客户端需要运行hiveserver服务来和Hive通信。通过设置HIVE_PORT环境变量来指明服务器监听的端口号（默认为10000）. hwi Hive的web接口。

jar 与hadoop jar等价的接口。这是运行类路径中同hadoop和Hive类的Java应用程序的简便方法。

metastore 默认情况下metastore 和Hive服务运行在同一个进程中。使用这个服务可以让metastore作为一个单独的（远程）进程运行。通过设置metastore_PORT 环境变量可以指定服务器监听的端口号。

6 meatastore

metastore 是Hive元数据的集中存放地。metastore包括两部分：服务(上面简单介绍)和后台数据的存储。默认情况下，metastore服务和Hive服务运行在同一个JVM中，它包含一个内嵌的以本地磁盘作为存储的Derby数据库实例。这称为“内嵌metastore”配置（embedded metastore configuration）

使用内嵌metastore是Hive入门最简单的方法。但是，只使用一个内嵌Derby数据库每次只能访问一个磁盘上的数据库文件，这也就是意味着每次只能为每个metastore打开一个Hive会话。如果试着启动第二个会话，在它试图连接metastore 会有以下错误信息:

’Failed to start datbase ’metastore_db’

如果要支持多会话（以及多用户），需要使用一个独立的数据库。这种配置称“本地metastore”,因为metastore服务仍然和Hive服务运行在同一个进程中，但连接的却是另一个进程中运行的数据库，在同一台机器上或者在远程机器上。

一般都用MySQL数据库作为独立的metastore，在这里javax.jdo.option.ConnectionURL设为jdbc:mysql://localhost/dbname?createDatabaseIf not Exist=true, 而javax.jdo.option.ConnectionDriv则设为com.mysql.jdbc.Driver.当然还需要独立设置用户名和密码。还需要将MySQL的JDBC驱动的JAR文件放在Hive的lib目录下。

7 数据类型

Hive支持原子和复杂数据类型。原子数据类型包括数值型，布尔型和字符串类型。复杂数据，原子数据类型包括数值型，布尔型，和字符串类型。复杂数据包括数组，映射和结构。

TINYINT 1个字节（8位）有符号整数，从-128到127
SMALLINT 2个字节（16位）有符号整数，从-32768到32767
INT 4个字节（32位）有符号整数，从-21474836482147483467
FLOAT 4个字节（32位）单精度浮点数
DOUBLE 8个字节（64位）双精度浮点数
BOOLEAN true /false
STRING 字符串

复杂数据 ARRAY 一组有序字段。字段的类型必须相同

MAP 一组无序的键/值对。键的类型必须是 map('a',1,'b',2)原子的；值的类型可以是任何类型的，同一个映射的键的类型必须相同，值的类型也必须相同 STRUCT 一组命名的字段。字段的类型可以不同struct（'a',1,1,0）

8 托管表和外部表

在传统数据库中，表的模式是在数据加载时强制确定的。如果在加载时发现数据不符合模式，则拒绝加载数据。因为数据是写入数据库时对照模式进行检查。这种模式也叫做写时模式。在Hive中创建表时，默认情况下Hive负责管理数据。这意味着Hive把数据移入它的"仓库目¿录"（waerhouse directory）．另一种选择是创建一个"外部表"（external table）。这会让Hive到仓库目录以外的位置访问数据。这两种表的区别表现在LOAD和DROP命令的语意上。

```
CREATE TABLE MANAGED_TABLE (DUMY STRING);
LOAD DATA INPATH '/usr/tom/data.txt' INTO TABLE MANAGED_TABLE;
```

由于加载操作就是文件系统中的文件移动，因此它的执行速度很快。如果丢弃一个托管表，可使用

```
DROP TABLE MANAGED_TABLE
```

然后这个表（包括它的元数据和数据）会被删除。还有一个表叫外部表：对于外部表，这两个操作的意义就不一样了。由自己来控制数据的创建和删除。外部表数据的位置需要在创建表的时候指明。丢弃外部表时，Hive只删除元数据，不碰数据。

9 Rhive

以下是几个使用Rhive包的例子

```
library(RHive)
rhive.init();
rhive.connect("192.168.3.146");
tables<-rhive.list.tables();
Desc<-rhive.desc.table('gener') ;
mode<-rhive.basic.mode("gener","col1") ;
range<-rhive.basic.range("gener","col1");
```

```
rhive.hdfs.connect();
rhive.hdfs.ls();
du<-rhive.hdfs.du();
print("The du of the hdfs")
du
gener<-rhive.query("select * from ia");
querstr<-"create table records (id int , name string , value double)";
# Result<-rhive.query(querstr);
D<-rhive.query('show tables ')
rhive.hdfs.du();
#rhive.query("dfs -ls ");
system("hive -f script.sql")
rhive.close()
#system("ls  ");
system("hive -f script.sql");
#system("Rscript  sys.r")
```

上面用到了Rhive的一些基本用发，其中使用最多的就是检索数据。

这里

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35
41	42	43	44	45

利用Rhive从hdfs中读取数据并做一个简单的回归分析

```
#!/usr/local/bin/Rscript
library(rJava)
library(Rserve)
library(RHive)
rhive.init();
rhive.connect();
d <- rhive.query('select * from ia')
#rhive.query('load data inpath \'input/data.txt\' overwrite into table gene
dataframe <-d;
tableinformation<-rhive.query('show  tables ');
tableinformation
class(dataframe);
summary(dataframe);
gener<-rhive.query('select * from gener');
colnames(gener)
class(gener)
str(gener)
#summary(gener)
t1<-as.numeric(as.character(gener[,1]))
gener<-gener[,-1];
gener[,1]<-as.numeric(as.character(gener[,1]))
gener[,2]<-as.numeric(as.character(gener[,2]))
gener[,3]<-as.numeric(as.character(gener[,3]))
gener
model<-lm(col2~col3,data=gener);
summary(model);
model
```

```
#gener
rhive.close()
```