基于 Hive 的日志数据统计实战

一、Hive 简介

Hive 是一个基于 hadoop 的开源数据仓库工具，用于存储和处理海量结构化数据。 它把海量数据存储于 hadoop 文件系统，而不是数据库，但提供了一套类数据库的数据存储和处理机制，并采用 HQL （类 SQL ）语言对这些数据进行自动化管理和处理。我们可以把 hive 中海量结构化数据看成一个个的表，而实际上这些数据是分布式存储在 HDFS 中的。 Hive 经过对语句进行解析和转换，最终生成一系列基于 hadoop 的 map/reduce 任务，通过执行这些任务完成数据处理。

Hive 诞生于 facebook 的日志分析需求，面对海量的结构化数据， hive 以较低的成本完成了以往需要大规模数据库才能完成的任务，并且学习门槛相对较低，应用开发灵活而高效。

Hive 自 2009.4.29 发布第一个官方稳定版 0.3.0 至今，不过一年的时间，正在慢慢完善，网上能找到的相关资料相当少，尤其中文资料更少，本文结合业务对 hive 的应用做了一些探索，并把这些经验做一个总结，所谓前车之鉴，希望读者能少走一些弯路。

Hive 的官方 wiki 请参考这里 ：

http://wiki.apache.org/hadoop/Hive

官方主页在这里：

http://hadoop.apache.org/hive/

Hive-0.5.0 源码包和二进制发布包的下载地址

http://labs.renren.com/apache-mirror/hadoop/hive/hive-0.5.0/

二、部署

由于 Hive 是基于 hadoop 的工具，所以 hive 的部署需要一个正常运行的 hadoop 环境。以下介绍 hive 的简单部署和应用。

部署环境：

操作系统： Red Hat Enterprise Linux AS release 4 (Nahant Update 7)

Hadoop ： hadoop-0.20.2 ，正常运行

部署步骤如下：

1、下载最新版本发布包 hive-0.5.0-dev.tar.gz ，传到 hadoop 的 namenode 节点上，解压得到 hive 目录。假设路径为： /opt/hadoop/hive-0.5.0-bin

2、设置环境变量 HIVE_HOME ，指向 hive 根目录 /opt/hadoop/hive-0.5.0-bin 。由于 hadoop 已运行，检查环境变量 JAVA_HOME 和 HADOOP_HOME 是否正确有效。

3、切换到 $HIVE_HOME 目录， hive 配置默认即可，运行 bin/hive 即可启动 hive ，如果正常启动，将会出现" hive> "提示符。

4、在命令提示符中输入" show tables; "，如果正常运行，说明已部署成功，可供使用。

常见问题：

1、执行" show tables; "命令提示" FAILED: Error in metadata: java.lang.IllegalArgumentException: URI: does not have a scheme "，这是由于 hive 找不到存放元数据库的数据库而导致的，修改 conf/ hive-default.xml 配置文件中的 hive.metastore.local 为 true 即可。由于 hive 把结构化数据的元数据信息放在第三方数据库，此处设置为 true ， hive 将在本地创建 derby 数据库用于存放元数据。当然如果有需要也可以采用 mysql 等第三方数据库存放元数据，不过这时 hive.metastore.local 的配置值应为 false 。

2、如果你已有一套 nutch1.0 系统正在跑，而你不想单独再去部署一套 hadoop 环境，你可以直接使用 nutch1.0 自带的 hadoop 环境，但这样的部署会导致 hive 不能正常运行，提示找不到某些方法。这是由于 nutch1.0 使用了 commons-lang-2.1.jar 这个包，而 hive 需要的是 commons-lang-2.4.jar ，下载一个 2.4 版本的包替换掉 2.1 即可， nutch 和 hive 都能正常运行。

三、应用场景

本文主要讲述使用 hive 的实践，业务不是关键，简要介绍业务场景，本次的任务是对搜索日志数据进行统计分析。

集团搜索刚上线不久，日志量并不大 。这些日志分布在 5 台前端机，按小时保存，并以小时为周期定时将上一小时产生的数据同步到日志分析机，统计数据要求按小时更新。这些统计项，包括关键词搜索量 pv ，类别访问量，每秒访问量 tps 等等。

基于 hive ，我们将这些数据按天为单位建表，每天一个表，后台脚本根据时间戳将每小时同步过来的 5 台前端机的日志数据合并成一个日志文件，导入 hive 系统，每小时同步的日志数据被追加到当天数据表中，导入完成后，当天各项统计项将被重新计算并输出统计结果。

以上需求若直接基于 hadoop 开发，需要自行管理数据，针对多个统计需求开发不同的 map/reduce 运算任务，对合并、排序等多项操作进行定制，并检测任务运行状态，工作量并不小。但使用 hive ，从导入到分析、排序、去重、结果输出，这些操作都可以运用 hql 语句来解决，一条语句经过处理被解析成几个任务来运行，即使是关键词访问量增量这种需要同时访问多天数据的较为复杂的需求也能通过表关联这样的语句自动完成，节省了大量工作量。

四、Hive 实战

初次使用 hive ，应该说上手还是挺快的。 Hive 提供的类 SQL 语句与 mysql 语句极为相似，语法上有大量相同的地方，这给我们上手带来了很大的方便，但是要得心应手地写好这些语句，还需要对 hive 有较好的了解，才能结合 hive 特色写出精妙的语句。

关于 hive 语言的详细语法可参考官方 wiki 的语言手册：
http://wiki.apache.org/hadoop/Hive/LanguageManual

虽然语法风格为我们提供了便利，但初次使用遇到的问题还是不少的，下面针对业务场景谈谈我们遇到的问题，和对 hive 功能的定制。

1、 分隔符问题

首先遇到的是日志数据的分隔符问题，我们的日志数据的大致格式如下：

2010-05-24 00:00:02@$_$@QQ2010@$_$@all@$_$@NOKIA_1681C@$_$@1@$_$@10@$_$@@$_$@-1@$_$@10@$_$@application@$_$@1

从格式可见其分隔符是“ @$_$@ ”，这是为了尽可能防止日志正文出现与分隔符相同的字符而导致数据混淆。本来 hive 支持在建表的时候指定自定义分隔符的，但经过多次测试发现只支持单个字符的自定义分隔符，像“ @$_$@ ”这样的分隔符是不能被支持的，但是我们可以通过对分隔符的定制解决这个问题， hive 的内部分隔符是“ \001 ”，只要把分隔符替换成“\001 ”即可。

经过探索我们发现有两条途径解决这个问题。

a)自定义 outputformat 和 inputformat 。

Hive 的 outputformat/inputformat 与 hadoop 的 outputformat/inputformat 相当类似， inputformat 负责把输入数据进行格式化，然后提供给 hive ， outputformat 负责把 hive 输出的数据重新格式化成目标格式再输出到文件，这种对格式进行定制的方式较为底层，对其进行定制也相对简单，重写 InputFormat 中 RecordReader 类中的 next 方法即可，示例代码如下：

```
public boolean next(LongWritable key, BytesWritable value)

throws IOException {

while ( reader .next(key, text ) ) {

String strReplace =
text .toString().toLowerCase().replace( "@$_$@" , "\001" );

Text txtReplace = new Text();

txtReplace.set(strReplace );

value.set(txtReplace.getBytes(), 0, txtReplace.getLength());

return true ;

}
```

```
return false ;

}

重写 HiveIgnoreKeyTextOutputFormat 中 RecordWriter 中的
write 方法，示例代码如下：

public void write (Writable w) throws IOException {

String strReplace = ((Text)w).toString().replace( "\001" ,
"@$_$@" );

Text txtReplace = new Text();

txtReplace.set(strReplace);

byte [] output = txtReplace.getBytes();

bytesWritable .set(output, 0, output. length );

writer .write( bytesWritable );

}
```

自定义 outputformat/inputformat 后，在建表时需要指定 outputformat/inputformat ，如下示例：

stored as INPUTFORMAT 'com.aspire.search.loganalysis.hive.SearchLogInputFormat' OUTPUTFORMAT 'com.aspire.search.loganalysis.hive.SearchLogOutputFormat'

b) 通过 SerDe(serialize/deserialize) ，在数据序列化和反序列化时格式化数据。

这种方式稍微复杂一点，对数据的控制能力也要弱一些，它使用正则表达式来匹配和处理数据，性能也会有所影响。但它的优点是可以自定义表属性信息 SERDEPROPERTIES ，在 SerDe 中通过这些属性信息可以有更多的定制行为。

2、 数据导入导出

a) 多版本日志格式的兼容

由于 hive 的应用场景主要是处理冷数据（只读不写），因此它只支持批量导入和导出数据，并不支持单条数据的写入或更新，所以如果要导入的数据存在某些不太规范的行，则需要我们定制一些扩展功能对其进行处理。

我们需要处理的日志数据存在多个版本，各个版本每个字段的数据内容存在一些差异，可能版本 A 日志数据的第二个列是搜索关键字，但版本 B 的第二列却是搜索的终端类型，如果这两个版本的日志直接导入 hive 中，很明显数据将会混乱，统计结果也不会正确。我们的任务是要使多个版本的日志数据能在 hive 数据仓库中共存，且表的 input/output 操作能够最终映射到正确的日志版本的正确字段。

这里我们不关心这部分繁琐的工作，只关心技术实现的关键点，这个功能该在哪里实现才能让 hive 认得这些不同格式的数据呢？经过多方尝试，在中间任何环节做这个版本适配都将导致复杂化，最终这个工作还是在 inputformat/outputformat 中完成最为优雅，毕竟 inputformat 是源头， outputformat 是最终归宿。具体来说，是在前面提到的 inputformat 的 next 方法中和在 outputformat 的 write 方法中完成这个适配工作。

b) Hive 操作本地数据

一开始，总是把本地数据先传到 HDFS ，再由 hive 操作 hdfs 上的数据，然后再把数据从 HDFS 上传回本地数据。后来发现大可不必如此， hive 语句都提供了“ local ”关键字，支持直接从本地导入数据到 hive ，也能从 hive 直接导出数据到本地，不过其内部计算时当然是用 HDFS 上的数据，只是自动为我们完成导入导出而已。

3、 数据处理

日志数据的统计处理在这里反倒没有什么特别之处，就是一些 SQL 语句而已，也没有什么高深的技巧，不过还是列举一些语句示例，以示 hive 处理数据的方便之处，并展示 hive 的一些用法。

a) 为 hive 添加用户定制功能，自定义功能都位于 hive_contrib.jar 包中

```
add jar /opt/hadoop/hive-0.5.0-bin/lib/hive_contrib.jar;
```

b) 统计每个关键词的搜索量，并按搜索量降序排列，然后把结果存入表 keyword_20100603 中

```
create table keyword_20100603 as select keyword,count(keyword) as count from searchlog_20100603 group by keyword order by count desc;
```

c) 统计每类用户终端的搜索量，并按搜索量降序排列，然后把结果存入表 device_20100603 中

create table device_20100603 as select device,count(device) as count from searchlog_20100603 group by device order by count desc;

d) 创建表 time_20100603 ，使用自定义的 INPUTFORMAT 和 OUTPUTFORMAT ，并指定表数据的真实存放位置在 '/LogAnalysis/results/time_20100603' （ HDFS 路径），而不是放在 hive 自己的数据目录中

create external table if not exists time_20100603(time string, count int) stored as INPUTFORMAT 'com.aspire.search.loganalysis.hive.XmlResultInputFormat' OUTPUTFORMAT 'com.aspire.search.loganalysis.hive.XmlResultOutputFormat' LOCATION '/LogAnalysis/results/time_20100603';

e) 统计每秒访问量 TPS ，按访问量降序排列，并把结果输出到表 time_20100603 中，这个表我们在上面刚刚定义过，其真实位置在 '/LogAnalysis/results/time_20100603' ，并且由于 XmlResultOutputFormat 的格式化，文件内容是 XML 格式。

insert overwrite table time_20100603 select time,count(time) as count from searchlog_20100603 group by time order by count desc;

f) 计算每个搜索请求响应时间的最大值，最小值和平均值

insert overwrite table response_20100603 select max(responsetime) as max,min(responsetime) as min,avg(responsetime) as avg from searchlog_20100603;

g)创建一个表用于存放今天与昨天的关键词搜索量和增量及其增量比率，表数据位于 '/LogAnalysis/results/keyword_20100604_20100603' ，内容将是 XML 格式。

create external table if not exists keyword_20100604_20100603(keyword string, count int, increment int, incrementrate double) stored as INPUTFORMAT 'com.aspire.search.loganalysis.hive.XmlResultInputFormat' OUTPUTFORMAT 'com.aspire.search.loganalysis.hive.XmlResultOutputFormat' LOCATION '/LogAnalysis/results/keyword_20100604_20100603';

h)设置表的属性，以便 XmlResultInputFormat 和 XmlResultOutputFormat 能根据 output.resulttype 的不同内容输出不同格式的 XML 文件。

alter table keyword_20100604_20100603 set tblproperties ('output.resulttype'='keyword');

i) 关联今天关键词统计结果表（ keyword_20100604 ）与昨天关键词统计结果表（ keyword_20100603 ），统计今天与昨天同时出现的关键词的搜索次数，今天相对昨天的增量和增量比率，并按增量比率降序排列，结果输出到刚刚定义的 keyword_20100604_20100603 表中，其数据文件内容将为 XML 格式。

insert overwrite table keyword_20100604_20100603 select cur.keyword, cur.count, cur.count-yes.count as increment, (cur.count-yes.count)/yes.count as incrementrate from keyword_20100604 cur join keyword_20100603 yes on (cur.keyword = yes.keyword) order by incrementrate desc;

4、用户自定义函数 UDF

部分统计结果需要以 CSV 的格式输出，对于这类文件体全是有效内容的文件，不需要像 XML 一样包含 version ， encoding 等信息的文件头，最适合用 UDF(user define function) 了。

UDF 函数可直接应用于 select 语句，对查询结构做格式化处理之后，再输出内容。自定义 UDF 需要继承 org.apache.hadoop.hive.ql.exec.UDF ，并实现 evaluate 函数， Evaluate 函数支持重载，还支持可变参数。我们实现了一个支持可变字符串参数的 UDF ，支持把 select 得出的任意个数的不同类型数据转换为字符串后，按 CSV 格式输出，由于代码较简单，这里给出源码示例：

```
public String evaluate(String... strs) {

StringBuilder sb = new StringBuilder();

for ( int i = 0; i < strs. length ; i++) {

sb.append(ConvertCSVField(strs[i])).append( ',' );

}
```

```
sb.deleteCharAt(sb.length()-1);

return sb.toString();

}
```

需要注意的是，要使用 UDF 功能，除了实现自定义 UDF 外，还需要加入包含 UDF 的包，示例：

```
add jar /opt/hadoop/hive-0.5.0-bin/lib/hive_contrib.jar;
```

然后创建临时方法，示例：

```
CREATE TEMPORARY FUNCTION Result2CSv AS
'com.aspire.search.loganalysis.hive. Result2CSv';
```

使用完毕还要 drop 方法，示例：

```
DROP TEMPORARY FUNCTION Result2CSv;
```

5、输出 XML 格式的统计结果

前面看到部分日志统计结果输出到一个表中，借助 XmlResultInputFormat 和 XmlResultOutputFormat 格式化成 XML 文件，考虑到创建这个表只是为了得到 XML 格式的输出数据，我们只需实现 XmlResultOutputFormat 即可，如果还要支持 select 查询，则我们还需要实现 XmlResultInputFormat ，这里我们只介绍 XmlResultOutputFormat 。

前面介绍过，定制 XmlResultOutputFormat 我们只需重写 write 即可，这个方法将会把 hive 的以 '\001' 分隔的多字段数据格式化为我们需要的 XML 格式，被简化的示例代码如下：

```
public void write(Writable w) throws IOException {

String[] strFields = ((Text) w).toString().split( "\001" );

StringBuffer sbXml = new StringBuffer();
```

```
if ( strResultType .equals( "keyword" )) {

sbXml.append( "<record><keyword>" ).append(strFields[0]).append(

"</keyword><count>" ).append(strFields[1]).append( "</coun
t><increment>" ).append(strFields[2]).append(

"</increment><rate>" ).append(strFields[3]).append(

"</rate></result>" );

}

Text txtXml = new Text();

byte [] strBytes = sbXml.toString().getBytes( "utf-8" );

txtXml.set(strBytes, 0, strBytes. length );

byte [] output = txtXml.getBytes();

bytesWritable .set(output, 0, output. length );

writer .write( bytesWritable );

}
```

其中的 strResultType .equals("keyword") 指定关键词统计结果，这个属性来自以下语句对结果类型的指定，通过这个属性我们还可以用同一个 outputformat 输出多种类型的结果。

```
alter table keyword_20100604_20100603 set tblproperties
('output.resulttype'='keyword');
```

仔细看看 write 函数的实现便可发现，其实这里只输出了 XML 文件的正文，而 XML 的文件头和结束标签在哪里输出呢？所幸我们采用的是基于 outputformat 的实现，我们可以在构造函数输出 version ， encoding 等文件头信息，在 close() 方法中输出结束标签。

这也是我们为什么不使用 UDF 来输出结果的原因，自定义 UDF 函数不能输出文件头和文件尾，对于 XML 格式的数据无法输出完整格式，只能输出 CSV 这类所有行都是有效数据的文件。

五、总结

Hive 是一个可扩展性极强的数据仓库工具，借助于 hadoop 分布式存储计算平台和 hive 对 SQL 语句的理解能力，我们所要做的大部分工作就是输入和输出数据的适配，恰恰这两部分 IO 格式是千变万化的，我们只需要定制我们自己的输入输出适配器， hive 将为我们透明化存储和处理这些数据，大大简化我们的工作。本文的重心也正在于此，这部分工作相信每一个做数据分析的朋友都会面对的，希望对您有益。

本文介绍了一次相当简单的基于 hive 的日志统计实战，对 hive 的运用还处于一个相对较浅的层面，目前尚能满足需求。对于一些较复杂的数据分析任务，以上所介绍的经验很可能是不够用的，甚至是 hive 做不到的， hive 还有很多进阶功能，限于篇幅本文未能涉及，待日后结合具体任务再详细阐述。

本文转载自：http://xuxingyin.javaeye.com/blog/764309