构建可视化交互大数据查询平台

Admaster 向磊

大数据的交互式查询

1. 交互式查询很重要
   - 我们不能要求所有人使用Linux命令行
   - 减轻开发人员压力
   - 需求即结果

2. 可选择余地很少
   - HUE, Qubole, phpHiveAdmin, Shib...

3. 难点
   - 作业的提交
   - 作业监控
   - 集群监控
   - 操作HDFS

为什么要开发交互式平台

一个最根本的原因是，不是所有人都会用Linux shell

不是所有人都能从shell操作Hadoop, Hive, Spark...，而且这样不安全

让专业的人做专业的事，让产品经理自己去写无聊的SQL吧

一个数据分析作业在运行以前，可以在界面里先进行测试

界面化交互式平台可以解决以上问题

别人开发的东西难用，不会维护

最重要的事情提高数据分析工作的开发效率，干什么都要快

各平台对比

	HUE	phpHiveAdmin	Qubole
开源	Yes	Yes	No
开发语言	Python/Django	php/CodeIgniter	Python/java SDK
部署难度	Easy	Medium	AWS
功能	多/强大	Hive	Hive
二次开发	Python+Java	容易	Only SDK
Spark支持	Y	N	N
集群作业监控	Y	N	?
HDFS管理	Y	Half	?
可维护性	较难	矛盾	找Qubole
架构	B/S+C/S	B/S	Unknown

典型案例

一个典型的例子是我在那家涨停N次以后停牌的公司做phpHiveAdmin的时候

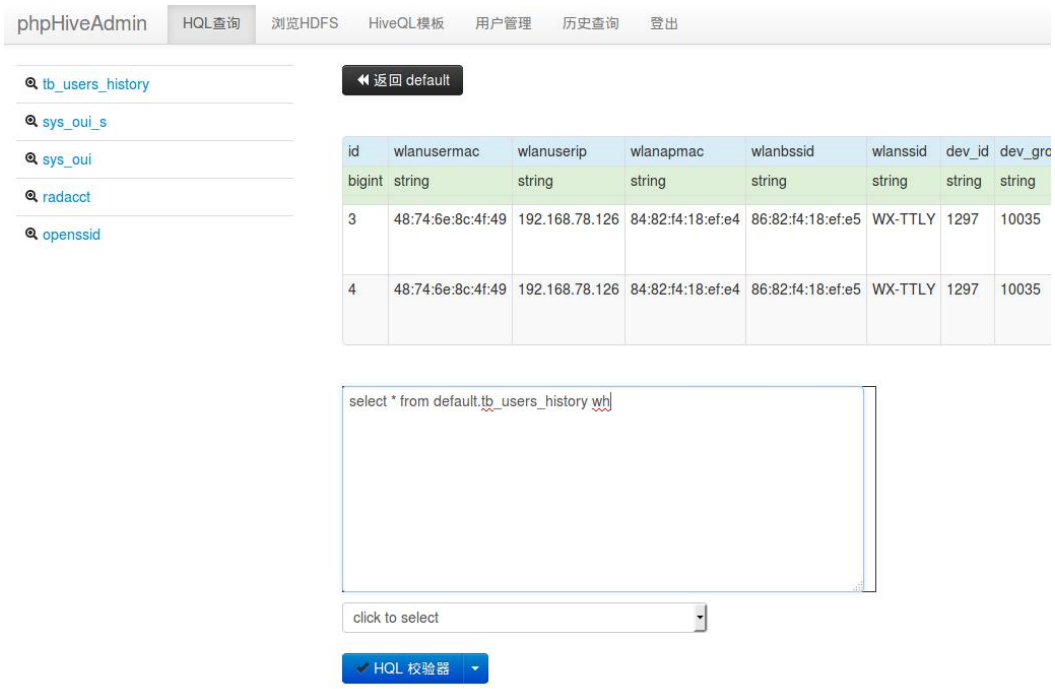
不要问我关于那家公司股票的问题，我没有的

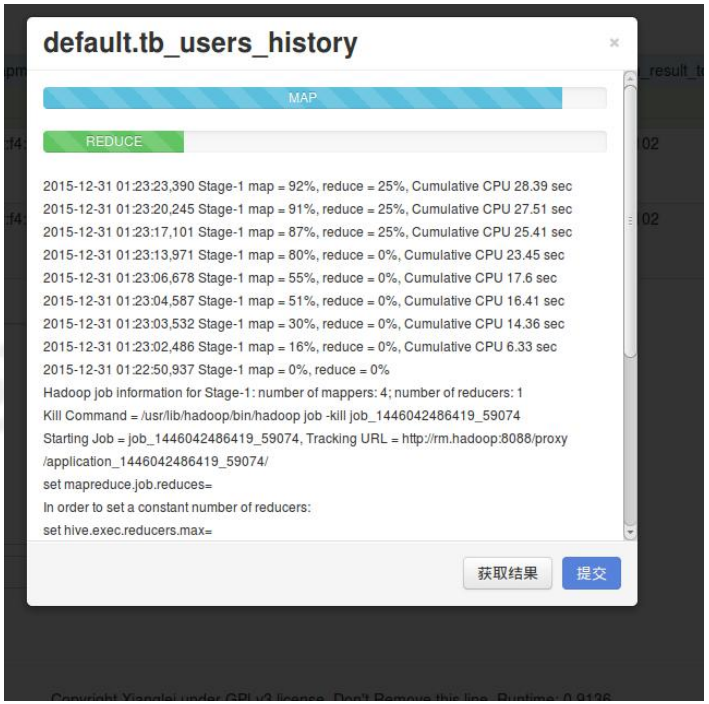
结果出乎意料

1. 极大的提高了在Hive上做数据分析的效率
2. 进而极大的提高了数据的产出效率
3. 开源后有N多国内外公司使用，并提交patch
4. 但是我仍然建议最好自己写一个
5. 这是一个公司从核心技术层面深入了解Hadoop和Spark的好机会
6. 将Spark/Hadoop/HBase/Hive整合为一个统一管理的可视化数据平台，大幅度降低分别管理和维护的成本。
7. 大幅度降低数据分析业务开发的周期和成本。

因何降低成本？

把N多数据分析作业以众包形式分解，以提高分析作业的开发效率





自己开发一个难吗？

关键在了解各种接口及对接口数据的获取

	HDFS	YARN/MR	Hive	Pig	HBase	Spark
Restful	Y	Y	N	N	Y	1.4 later
Thrift	N	N	Y	N	Y	SparkSQL
CLI	Y	Y	Y	Y	Y	Y

需要各接口的整合开发，首先以HDFS为例：

Hadoop提供基于HTTP/HTTPS的HDFS访问接口
- 同时支持对HDFS的读写操作
- 可以从程序或脚本中访问
- 可以用命令行工具如curl或wget来实现数据文件的上传下载
- 启用webhdfs或httpfs
- webHDFS为内置，但不支持HA
- httpfs为外置tomcat，但支持HA
- 题外话，集群对拷也可以用webhdfs方式

REST = REpresentational State Transfer

Restful HDFS

通过浏览器或CURL访问HDFS

curl -i -L "http://x.x.x.x:50070/webhdfs/v1/?OP=LISTSTATUS&user.name=hdfs"
HTTP GET组内OP参数可以列目录，文件状态.....
HTTP PUT组内OP参数可以创建文件夹，改名，设权限.....
HTTP POST组内OP参数可以APPEND, CONCAT, TRUNCATE
HTTP DELETE组内OP参数可以删除文件和快照.....


```
{"FileStatuses":{"FileStatus":[
{"accessTime":0,"blockSize":0,"childrenNum":6,"fileId":16389,"group":"hadoop","length":0,"modificationTime":1434635241699,"owner":"yarn","pathSuffix":"app-logs","permission":"777","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":3,"fileId":16403,"group":"hdfs","length":0,"modificationTime":1430373809715,"owner":"hdfs","pathSuffix":"apps","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":2,"fileId":145301,"group":"hdfs","length":0,"modificationTime":1435719592612,"owner":"hdfs","pathSuffix":"data","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":0,"fileId":437109,"group":"hdfs","length":0,"modificationTime":1434351238766,"owner":"flume","pathSuffix":"flume","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":1,"fileId":16395,"group":"hdfs","length":0,"modificationTime":1430245449189,"owner":"hdfs","pathSuffix":"hdp","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":1,"fileId":2268282,"group":"hdfs","length":0,"modificationTime":1447818808124,"owner":"hdfs","pathSuffix":"home","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":1,"fileId":16390,"group":"hdfs","length":0,"modificationTime":1430245405207,"owner":"mapred","pathSuffix":"mapred","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":2,"fileId":16392,"group":"hdfs","length":0,"modificationTime":1430245405307,"owner":"hdfs","pathSuffix":"mr-history","permission":"777","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":1,"fileId":16400,"group":"hdfs","length":0,"modificationTime":1437481112419,"owner":"hdfs","pathSuffix":"system","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":1,"fileId":2282779,"group":"hdfs","length":0,"modificationTime":1447926770271,"owner":"hdfs","pathSuffix":"tempdata1","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":8,"fileId":16386,"group":"hdfs","length":0,"modificationTime":1448362128914,"owner":"hdfs","pathSuffix":"tmp","permission":"777","replication":0,"storagePolicy":0,"type":"DIRECTORY"},
{"accessTime":0,"blockSize":0,"childrenNum":13,"fileId":16387,"group":"hdfs","length":0,"modificationTime":1436341722627,"owner":"hdfs","pathSuffix":"user","permission":"755","replication":0,"storagePolicy":0,"type":"DIRECTORY"}
]}}
```


curl -i -X PUT
"http://x.x.x.x:50070/webhdfs/v1/TokyoHot?op=MKDIRS&user.name=xianglei"

Doc: http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html

RESTful YARN/MR

通过curl访问YARN/MapReduce

curl -i -L "http://x.x.x.x:8088/ws/v1/cluster/info"

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Tue, 29 Dec 2015 13:38:59 GMT
Date: Tue, 29 Dec 2015 13:38:59 GMT
Pragma: no-cache
Expires: Tue, 29 Dec 2015 13:38:59 GMT
Date: Tue, 29 Dec 2015 13:38:59 GMT
Pragma: no-cache
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(6.1.26.hwx)

{"clusterInfo":{"id":1451059230551,"startedOn":1451059230551,"state":"STARTED","
haState":"ACTIVE","rmStateStoreName":"org.apache.hadoop.yarn.server.resourcemana
ger.recovery.ZKRMStateStore","resourceManagerVersion":"2.6.0.2.2.6.0-2800","reso
urceManagerBuildVersion":"2.6.0.2.2.6.0-2800 from acb70ecfae2c3c5ab46e24b0caebce
aec16fdcd0 by jenkins source checksum 65301b226e4b9e8135f2f93f9887ea63","resourc
eManagerVersionBuiltOn":"2015-05-18T20:28Z","hadoopVersion":"2.6.0.2.2.6.0-2800"
,"hadoopBuildVersion":"2.6.0.2.2.6.0-2800 from acb70ecfae2c3c5ab46e24b0caebceaec
16fdcd0 by jenkins source checksum a25c30f622eb057f47e2155f78dba5e","hadoopVersi
onBuiltOn":"2015-05-18T20:21Z"}}xianglei@xianglei-u303ub:~$
```

可以获取所有我们想要的作业相关信息，也可以提交作业，不过，首先你得把作业的jar包放进HDFS，然后再写一个json文件来提交。详情稍后

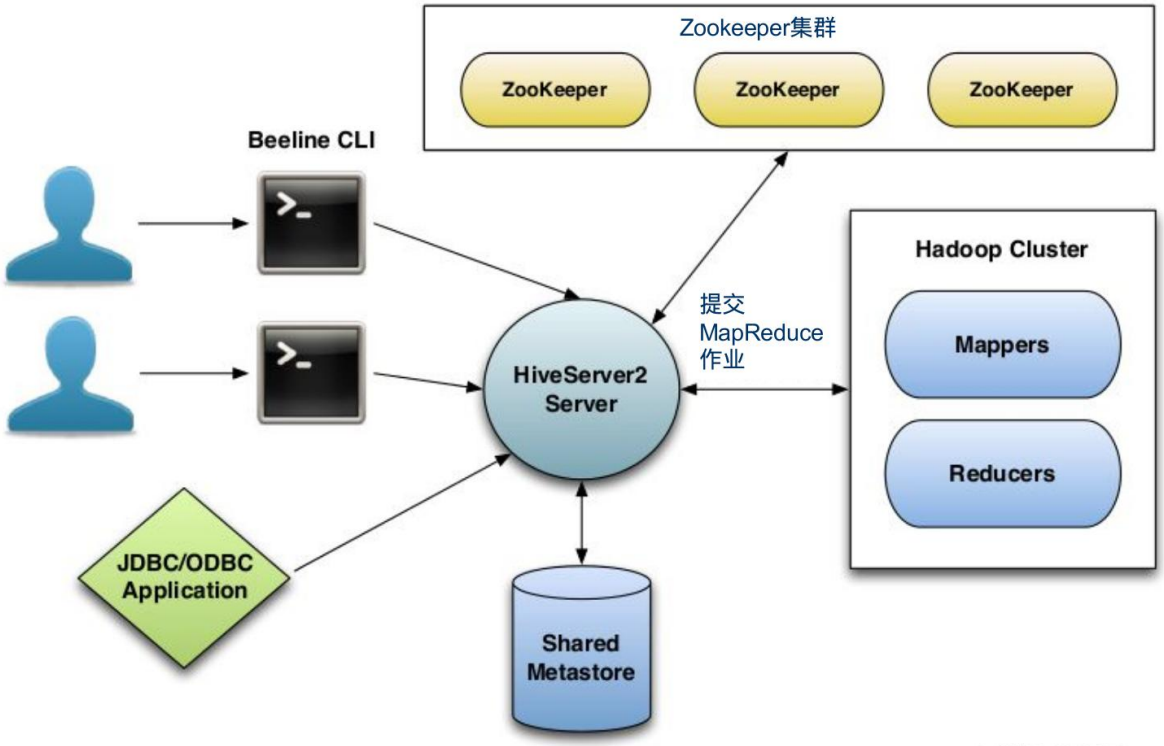
Doc: http://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerRest.html

关于Hive和HBase的Thrift

Hive, HBase, SparkSQL都可以通过thrift / thrift2来访问

- 实际上SparkSQL用的就是Hive的接口
- thrift2实际上是thrift接口的一个增强版，增加了安全性(cyrus-sasl authentication)，并发性(Zookeeper Quorum)...

Hive有个HiveMetastore的9083端口的什么鬼，还有个10000端口的HiveServer / HiveServer2的什么鬼。



Hive thrift/2能做什么？

可以获取数据库，表，字段定义
可以提交SQL作业
可以创建数据库，表，字段定义
可以修改库，表，字段定义
可以创建修改分区，桶，和其他一切操作

```
public $hive_host;
public $hive_port;
public $socket;
public $transport;
public $protocol;
public $hive;
```

```
public function __construct()
{
        parent::__construct();
        $GLOBALS['THRIFT_ROOT'] = __DIR__ . "/../../libs/";
        include_once $GLOBALS['THRIFT_ROOT'] . 'packages/hive_service/ThriftHive.php';
        include_once $GLOBALS['THRIFT_ROOT'] . 'transport/TSocket.php';
        include_once $GLOBALS['THRIFT_ROOT'] . 'protocol/TBinaryProtocol.php';

        $this->hive_host = $this->config->item('hive_host');
        $this->hive_port = $this->config->item('hive_port');
        $this->socket = new TSocket($this->hive_host, $this->hive_port);
        $this->socket->setSendTimeout(30000);
        $this->socket->setRecvTimeout(30000);
        $this->transport = new TBufferedTransport($this->socket);
        $this->protocol = new TBinaryProtocol($this->transport);
        $this->hive = new ThriftHiveClient($this->protocol);
}
```

```
public function drop_database($db_name, $del = FALSE)
{
        try
        {
                $this->transport->open();
                $this->hive->drop_database($db_name, $del);
                $this->transport->close();
        }
        catch (Exception $e)
        {
                echo 'Caught exception: '.  $e->getMessage(). "\n";
        }
}
```

```
public function create_database($db_name, $db_desc = '')
{
        $sql = "CREATE DATABASE IF NOT EXISTS ".$db_name." COMMENT '".$db_desc."'";
        try
        {
                $this->transport->open();
                $this->hive->execute($sql);
                $this->transport->close();
                return $sql;
        }
        catch (Exception $e)
        {
                echo 'Caught exception: '.  $e->getMessage(). "\n";
        }
}
```

Hive的thrift/2应该做什么？

除了提交SQL作业之外的所有事情

之前不是说thrift可以用来提交HQL查询吗？

Why Not?

Thrift/2无法返回作业的状态和进度，一个MR可能会延续几分钟或几小时，即便是你用Spark引擎跑Hive，一个大任务也会花费很长时间，坐在那什么也不干，只等着屏幕刷新是很无趣也是不可被接受的。特别是你还不知道它多长时间才会返回结果。
另外，thrift也不支持并发查询，thrift2虽然支持并发，但需要配置Zookeeper。

How to do？

我们在开发交互查询时，可以用Thrift来管理Hive/SparkSQL的元数据，但查询可以用别的方式。

使用CLI方式提交也许会更好

https://github.com/xianglei/phpHiveAdmin/blob/master/application/models/hive_model.php
774 - 880行

原理：在网页中单独开启一个php子进程，将SQL提交到CLI接口，然后通过子进程获取stdout和stderr的输出。然后将获取内容倒排后以JSON数组形式发给网页，网页只要不断的刷新就可以了。

其实更好的办法也许是用WebSocket，但是我对前端没那么熟。

One Notice：

由于你的webserver使用的用户也许没有配置HADOOP相关的环境变量，所以你需要把命令行里加上输出HADOOP_HOME,JAVA_HOME和HIVE_HOME的三个环境变量的export。

```
$descriptorspec = array(
       0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
       1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
       2 => array("pipe", "w") // stderr is a file to write to
);

$pipes= array();

$process = proc_open($command, $descriptorspec, $pipes);

$output= "";

if (!is_resource($process))
{
       return false;
}

#close child's input imidiately
fclose($pipes[0]);

stream_set_blocking($pipes[1],0);
stream_set_blocking($pipes[2],0);

$todo= array($pipes[1],$pipes[2]);
try
{
       $fp = fopen($file_name, "w");
       #fwrite($fp,$time_stamp."\n\n");
       while( true )
       {
              $read= array();
              #if( !feof($pipes[1]) ) $read[]= $pipes[1];
              if( !feof($pipes[$type]) )
                     $read[]= $pipes[$type];// get system stderr on real time

              if (!$read)
              {
                     break;
              }

              $ready= stream_select($read, $write=NULL, $ex= NULL, 2);

              if ($ready === false)
              {
                     break; #should never happen - something died
              }

              foreach ($read as $r)
              {
                     $s= fread($r,128);
                     $output .= $s;
                     fwrite($fp,$s);
              }

       }

       fclose($fp);
}
```

MR/Pig/Mahout/Spark

1. 都可以使用CLI或Restful方式来提交并运行作业，作业状态通过读取stdout和stderr或HTTP GET返回前端。

2. 把作业结果保存在文件中，可以保存在本地或保存在HDFS，在一个数据库里记录每个作业的提交，这样，你即使关闭浏览器，也可以从历史信息中找到你之前提交的作业结果。

```
public function cli_query($sql, $finger_print)
{
        $this->load->model('utilities_model', 'utils');
        $LANG = " export LANG=" . $this->config->item('lang_set') . "; ";
        $JAVA_HOME = " export JAVA_HOME=" . $this->config->item('java_home') . "; ";
        $HADOOP_HOME = " export HADOOP_HOME=" . $this->config->item('hadoop_home') . "; ";
        $HIVE_HOME = " export HIVE_HOME=" . $this->config->item('hive_home'). "; ";
```

浏览HDFS | HiveQL模板 | 用户管理 | 历史查询 | 登出

	文件名	文件内容	文件大小
☐	admin_2015-06-18-15-29-36_7ec66eba5728ae7c7b645e94cc84b2f6f7cf66fe.log	USE ruian_mac;select count(distinct user_mac) from ruian_mac.ruian_mac_20150618 where from_unixtime(cast(log_time as int),'HH') >= "14"	136 Bytes
☐	admin_2015-06-18-15-27-33_1e6f14d398c81e116a106f076464963364190b03.log	USE ruian_mac;select from_unixtime(cast(log_time as int),'HH') , count(distinct user_mac) from ruian_mac.ruian_mac_20150618 where from_unixtime(cast(log_time as int),'HH') >= "14" group by log_time	198 Bytes
☐	admin_2015-06-18-15-24-50_a650fd2ac35642e9ec000c29f59ae3b55df8dba1.log	USE ruian_mac;select from_unixtime(cast(log_time as int),'HH') as ttime, count(distinct user_mac) as cnt from ruian_mac.ruian_mac_20150618 where from_unixtime(cast(log_time as int),'HH') >= "14" group by ttime order by cnt desc	228 Bytes
☐	admin_2015-06-18-15-21-03_ee24c9130a1ac5c9f530b623028f0ba809bb69b9.log	USE ruian_mac;select count(*) from ruian_mac.ruian_mac_20150618 where from_unixtime(cast(log_time as int),'HH') >= "14"	119 Bytes
☐	admin_2015-06-18-15-18-33_b3455841e1dea589ef9c030e6dd5e34d532140e6.log	USE ruian_mac;select count(*) from ruian_mac.ruian_mac_20150618 where from_unixtime(cast(log_time as int),'HH') >= "14"	119 Bytes
☐	admin_2015-06-18-15-15-43_372652cbdbed212dd83ec6968bca1b23744f7476.log	USE ruian_mac;select count(*) from ruian_mac.ruian_mac_20150618 where from_unixtime(log_time,"HH") >= "14"	106 Bytes
☐	admin_2015-06-18-15-11-09_79991cd78459ef61c3c335649ac00d71cb82732f.log	USE ruian_mac;select count(*) from ruian_mac.ruian_mac_20150618 where from_unixtime(log_time,"HH") >= "14"	106 Bytes
☐	lvxuan_2015-06-18-13-21-03_e84c517acb2d124ef0b32b12ef37d3c37d65735c.log	USE ruian_mac;select * from ruian_mac.ruian_mac_20150618 group by user_mac	77 Bytes

ve -f "

```
        }
        catch (Exception $e)
        {
                echo 'Caught exception: '.  $e->getMessage(). "\n";
        }
}
```

我们了解了这最基本的三种接口

RESTful

- 基于HTTP/HTTPS的json/xml接口

Thrift

- 基于thrift的接口，事实上，Hive的JDBC/ODBC也不过就是对 thrift 的再次封装

CLI

- 最普通的命令行接口，简单，却非常实用。另外，可以并发查询。

把这些接口都整合在一起

下面以Python为例，说明开发步骤。

Otherside: 为何不用Java写？

1. 我太懒，Java代码太啰嗦，人生苦短，我用Python
2. 我对Java属于会看不会写，理由参看上一条
3. RESTful的JSON可直接转换Python的dict类型处理, dict可直接转换JSON

构建可视化交互查询的步骤

1. 通过Thrift获取Hive/SparkSQL的库表定义，并以JSON形式返回前端

```
class GetDatabaseRestHandler(BaseHandler):
    @tornado.web.authenticated
    def get(self):
        config = Config.get_config()
        db = self.get_argument('db')
        if config['ServerPort']:
            try:
                transport = TSocket.TSocket(config['hive_server_ip'], config['ServerPort'])
                transport = TTransport.TBufferedTransport(transport)
                protocol = TBinaryProtocol.TBinaryProtocol(transport)

                client = ThriftHive.Client(protocol)
                transport.open()

                dbi = client.get_database(db)

                transport.close()
                '''
                Database is an instance,
                 dbi.name str
                 dbi.parameters dict
                 privileges ?
                 ownerType int
                 ownerName str
                 locationUri str
                 description str
                '''
                dbj = {}
                dbj['name'] = dbi.name
                dbj['parameters'] = dbi.parameters
                dbj['privileges'] = dbi.privileges
                dbj['ownerType'] = dbi.ownerType
                dbj['ownerName'] = dbi.ownerName
                dbj['locationUri'] = dbi.locationUri
                dbj['description'] = dbi.description
                #print dbi
                self.write(json.dumps(dbj).encode('unicode_escape'))
            except Thrift.TException, tx:
                self.write('%s' % (tx.message))
        elif config['Server2Port']:
            return
        else:
            self.write('Not set Hive thrift port')
        return
```

构建可视化交互查询的步骤

2. 在网页上搞个输入框用来输入SQL和提交查询的按钮
3. 把HQL或pig脚本写入到文本文件，既是提交查询所需，也是罪证
4. 提交查询，然后写入到stdout和stderr输出临时文件中

```
    def write_hive_log(self, hql):
        JAVA_HOME=self.config['JAVA_HOME']
        HADOOP_HOME=self.config['HADOOP_HOME']
        HIVE_HOME=self.config['HIVE_HOME']
        SPARK_SQL_HOME=self.config['SPARK_SQL_HOME']
        logger = Logger()

        try:
            hql_file = open(self.hql_filename, 'a')
            hql = 'set hive.cli.print.header=true;' + 'use '+self.db_name+';' + hql
            hql_file.write(hql)
            hql_file.close()
            if self.config['spark_sql'] == '1':
                cmd = 'export JAVA_HOME=' + JAVA_HOME + '; export HADOOP_HOME=' + HADOOP_HOME + '; export HIVE_HOME=' + HIVE_HOME + '; ' + 'export SPARK_HOME=' + SPARK_SQL_HOME +
'; ' + SPARK_SQL_HOME + '/bin/spark-sql -f ' + self.hql_filename + ' 2>' + self.log_filename + ' 1>' + self.tmp_filename + ' &'
            else:
                cmd = 'export JAVA_HOME=' + JAVA_HOME + '; export HADOOP_HOME=' + HADOOP_HOME + '; export HIVE_HOME=' + HIVE_HOME + '; ' + HIVE_HOME + '/bin/hive -f ' +
self.hql_filename + ' 2>' + self.log_filename + ' 1>' + self.tmp_filename + ' &'
            #debug command below
            #cmd = '/usr/bin/hive 2>' + self.log_filename + ' 1>' + self.tmp_filename + ' &'
            print cmd
            os.system(cmd)

            logger.info('Executor : write_hive_log : ' +self.log_filename)

            return self.log_filename
        except IOError, e:
            logger.error('Executor : write_hive_log : ' + e.replace('\n', ''))
            return e
```

构建可视化交互查询的步骤

5. 读取stdout和stderr的输出临时文件，并且返回给前端。

```
def read_hive_log(self, log_filename):
    logger = Logger()
    try:
        log_file = open(log_filename, 'r')
        f = log_file.readlines()
        log_file.close()

        logger.info('Executor : read_hive_log : ' + log_filename)

        return f #return is a list
    except IOError, e:
        logger.error('Executor : read_hive_log : IOError' + log_filename)
        return e
```

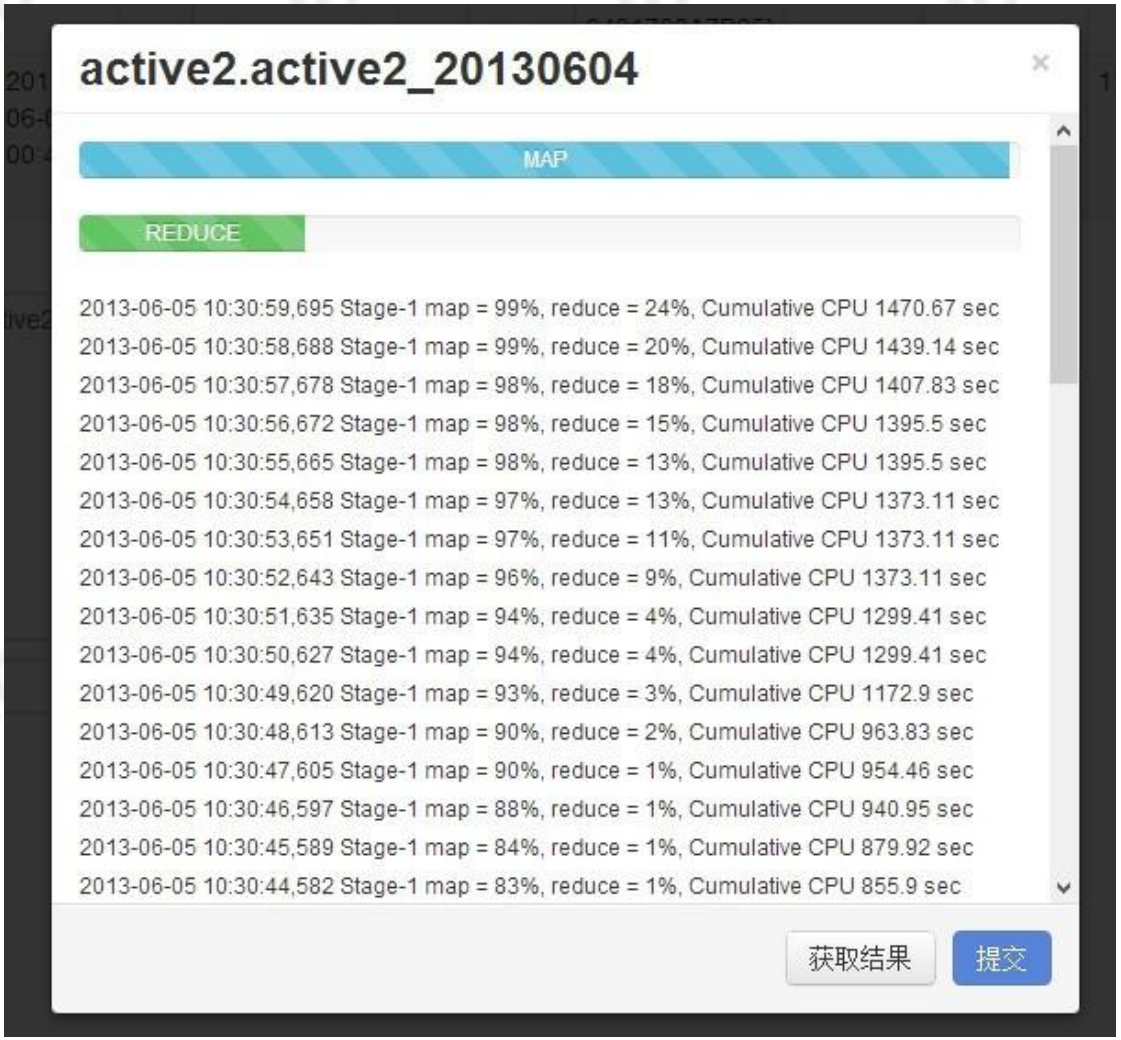
构建可视化交互查询的步骤

6. 用正则方式获取mr进度或其他内容，返回给前端做进度条用

```
def get(self, *args, **kwargs):
    logger = Logger()
    #log_filename = self.get_argument('log_filename')
    username = self.get_secure_cookie('username')
    finger_print = self.get_argument('finger_print')
    db_name = self.get_argument('db_name')

    config = Config.get_config()
    conn = sqlite3.connect(config['database'])
    cursor = conn.cursor()
    sql = 'select date_time from  hive_history_jobs where finger_print = ?'
    cursor.execute(sql,(finger_print,))
    row = cursor.fetchone()
    date = row[0]

    run = Executor(username, finger_print, db_name)
    file_basename = username + '_' + date + '_' + finger_print
    log_filename = config['log_dir'] + file_basename + '.log'
    log = {}
    try:
        log_reverse = run.read_hive_log(log_filename)[::-1]# or log.reverse()
        if config['Tez'] == '0' and config['spark_sql'] == '0':
            m_start = log_reverse[0].find('map = ') + 6
            m_end = log_reverse[0].find('%')
            m_per = log_reverse[0][m_start: m_end]

            r_start = log_reverse[0].find('reduce = ') + 9
            r_end = log_reverse[0].rfind('%')
            r_per = log_reverse[0][r_start: r_end]

            if r_per.isdigit() is not True:
                r_per = 100
            if m_per.isdigit() is not True:
                m_per = 100
            log['map_per'] = m_per
            log['reduce_per'] = r_per
            log['log'] = log_reverse
        else:
            log['map_per'] = '0'
            log['reduce_per'] = '0'
            log['log'] = log_reverse
        logger.info('QueryExecutorHandler : get : ' + run.log_filename)
    except IOError, e:
        log = e.split('\n')
        logger.error('QueryExecutorHandler : get : ' + e.replace('\n', '') + run.log_filename + sql)
    self.write(json.dumps(log, ensure_ascii=False))
```

构建可视化交互查询的步骤

7. 前端JS渲染从后端来的stdout和stderr数据



MR/Pig/Mahout/Spark/HBase

整合HDFS和YARN/MR RESTful API，那么数据分析师就可以自己选择HDFS目录和数据创建自己分析所需要的表。甚至可以自己从本地或服务器上传数据做分析。对于没有REST和Thrift的生态组件，我们可以采用CLI方式。

当然，再加上HCatalog，Pig也就可以使用Hive的元数据定义。

```
    '''
    send a hive cli execution with stderr redirect to log_filename and result to tmp_filename
    return nothing
    '''
    def write_pig_log(self, pig):
        JAVA_HOME=self.config['JAVA_HOME']
        HADOOP_HOME=self.config['HADOOP_HOME']
        PIG_HOME = self.config['PIG_HOME']
        logger = Logger()

        try:
            pig_file = open(self.pig_filename, 'a')
            pig_file.write(pig)
            pig_file.close()
            cmd = 'export JAVA_HOME=' + JAVA_HOME + '; export HADOOP_HOME=' + HADOOP_HOME + '; export PIG_HOME=' + PIG_HOME + '; ' + PIG_HOME + '/bin/pig -f ' + self.pig_filename
+ ' 2>' + self.log_filename + ' 1>' + self.tmp_filename + ' &'
            #debug command below
            #cmd = '/usr/bin/hive 2>' + self.log_filename + ' 1>' + self.tmp_filename + ' &'
            print cmd
            os.system(cmd)

            logger.info('Executor : write_pig_log : ' +self.log_filename)

            return self.log_filename
        except IOError, e:
            logger.error('Executor : write_pig_log : ' + e.replace('\n', ''))
            return e
    '''
    read from log_filename that redirected by write_pig_log
    '''
    def read_pig_log(self, log_filename):
        logger = Logger()
        try:
            log_file = open(log_filename, 'r')
            f = log_file.readlines()
            log_file.close()

            logger.info('Executor : read_pig_log : ' + log_filename)

            return f #return is a list
        except IOError, e:
            logger.error('Executor : read_pig_log : IOError' + log_filename)
            return e
```

MR提交的特别之处

我们可以通过YARN的RESTful接口来提交作业，需要以下步骤

1. 通过HDFS RESTful把jar包传到HDFS
2. 提交一个HTTP POST获取app-id

curl -v -X POST 'http://localhost:8088/ws/v1/cluster/apps/new-application'

```
{
application-id: application_1409421698529_0012",
"maximum-resource-capability":{"memory":"8192","vCores":"32"}
}
```

3. 获取ID后编写一个JSON并POST提交给RESTful接口

curl -v -X POST -d @example-submit-app.json -H "Content-type: application/json"
'http://x.x.x.x:8088/ws/v1/cluster/apps'

```
{
  "application-id":"application_1404203615263_0001",
  "application-name":"test",
  "am-container-spec":
  {
    "local-resources":
    {
      "entry":
      [
        {
          "key":"AppMaster.jar",
          "value":
          {
            "resource":"hdfs://hdfs-namenode:9000/user/testuser/DistributedShell/demo-app/AppMaster.jar",
            "type":"FILE",
            "visibility":"APPLICATION",
            "size": "43004",
            "timestamp": "1405452071209"
          }
        }
      ]
    },
    "commands":
    {
      "command":"{{JAVA_HOME}}/bin/java -Xmx10m org.apache.hadoop.yarn.applications.distributedshell.ApplicationMaster --container_memory 10 --container_vcores 1 --num_containers 1
    },
    "environment":
    {
      "entry":
      [
        {
          "key": "DISTRIBUTEDSHELLSCRIPTTIMESTAMP",
          "value": "1405459400754"
        },
        {
          "key": "CLASSPATH",
          "value": "{{CLASSPATH}}<CPS>./*<CPS>{{HADOOP_CONF_DIR}}<CPS>{{HADOOP_COMMON_HOME}}/share/hadoop/common/*<CPS>{{HADOOP_COMMON_HOME}}/share/hadoop/common/lib/*<CPS>{{HADOO
        },
        {
          "key": "DISTRIBUTEDSHELLSCRIPTLEN",
          "value": "6"
        },
        {
          "key": "DISTRIBUTEDSHELLSCRIPTLOCATION",
          "value": "hdfs://hdfs-namenode:9000/user/testuser/demo-app/shellCommands"
        }
      ]
    }
  },
  "unmanaged-AM":"false",
  "max-app-attempts":"2",
  "resource":
  {
    "memory":"1024",
    "vCores":"1"
  },
  "application-type":"YARN",
  "keep-containers-across-application-attempts":"false"
}
```

MR提交的特别之处

然后你拿到ID号，还可以查看任务状态

curl 'http://localhost:8088/ws/v1/cluster/apps/application_1409421698529_0012/state'

或者蛋疼一下，再杀掉这个任务

curl -v -X PUT -d '{"state": "KILLED"}' "http://x.x.x.x:8088/ws/v1/cluster/apps/application_1409421698529_0012'

关于Spark作业RESTful提交与监控

监控
curl -i -L "http://x.x.x.x:4040/api/v1/applications/[app-id]/storage/rdd"

历史作业信息
curl -i -L "http://x.x.x.x:18080/api/v1/applications"

如果是 on YARN
curl -i -L "http://x.x.x.x:4040/api/v1/applications/[app-id]/[attempt-id]"

Doc: http://spark.apache.org/docs/latest/monitoring.html

Spark作业提交
ooyala公司有个叫spark-jobserver的项目

先sbt打包并upload 作业到spark
curl --data-binary @job-server-tests/target/scala-2.10/job-server-tests-$VER.jar localhost:8090/jars/test

然后
curl -d "input.string = a b c a b see"
'localhost:8090/jobs?appName=test&classPath=spark.jobserver.WordCountExample'

Spark返回:

```
{
  "status": "STARTED",
  "result": {
    "jobId": "5453779a-f004-45fc-a11d-a39dae0f9bf4",
    "context": "b7ea0eb5-spark.jobserver.WordCountExample"
  }
}
```

Doc: https://github.com/spark-jobserver/spark-jobserver

HBase

对于HBase来说界面化就更简单了，既支持restful，也支持thrift和cli，不再赘述

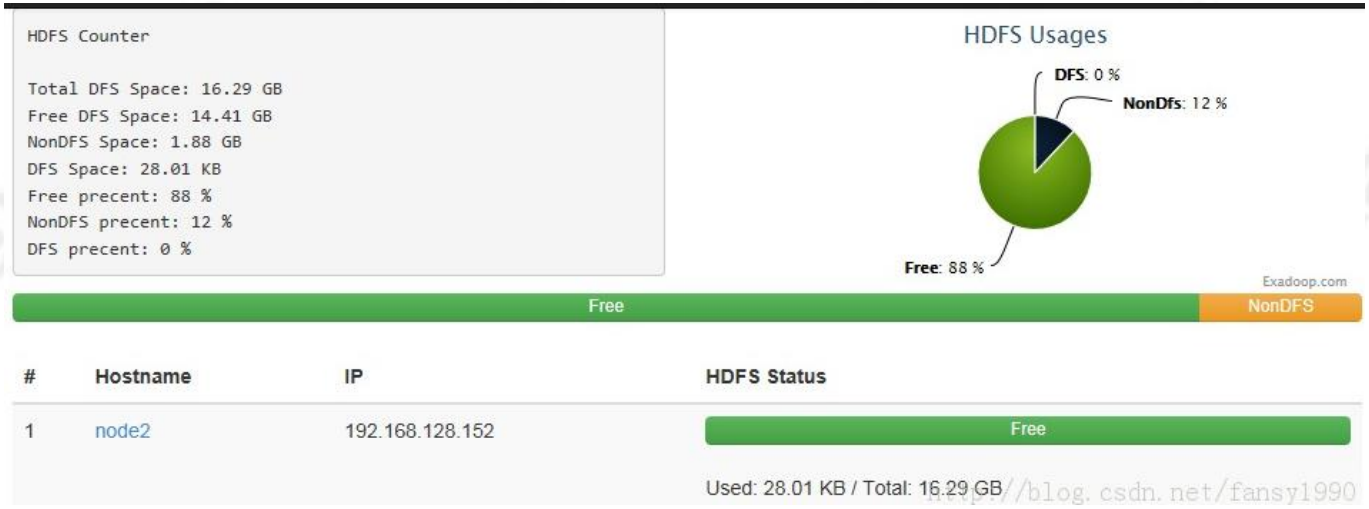
关于集群本身的监控

请使用HDFS/YARN/HBase/Spark的jmx接口和qry参数选择需要监控的内容。


```java
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response) {
  try {
    if (!isInstrumentationAccessAllowed(request, response)) {
      return;
    }
    JsonGenerator jg = null;
    PrintWriter writer = null;
    try {
      writer = response.getWriter();

      response.setContentType("application/json; charset=utf8");
      response.setHeader(ACCESS_CONTROL_ALLOW_METHODS, "GET");
      response.setHeader(ACCESS_CONTROL_ALLOW_ORIGIN, "*");

      jg = jsonFactory.createJsonGenerator(writer);
      jg.disable(JsonGenerator.Feature.AUTO_CLOSE_TARGET);
      jg.useDefaultPrettyPrinter();
      jg.writeStartObject();

      // query per mbean attribute
      String getmethod = request.getParameter("get");
      if (getmethod != null) {
        String[] splitStrings = getmethod.split("\\:\\:");
        if (splitStrings.length != 2) {
          jg.writeStringField("result", "ERROR");
          jg.writeStringField("message", "query format is not as expected.");
          jg.flush();
          response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
          return;
        }
        listBeans(jg, new ObjectName(splitStrings[0]), splitStrings[1],
            response);
        return;
      }

      // query per mbean
      String qry = request.getParameter("qry");
      if (qry == null) {
        qry = "*:*";
      }
      listBeans(jg, new ObjectName(qry), null, response);
    } finally {
      if (jg != null) {
        jg.close();
      }
      if (writer != null) {
        writer.close();
      }
    }
```

Doc: http://slaytanic.blog.51cto.com/2057708/1179108

关于集群本身的监控


HDFS Counter
Total DFS Space: 16.29 GB
Free DFS Space: 14.41 GB
NonDFS Space: 1.88 GB
DFS Space: 28.01 KB
Free precent: 88 %
NonDFS precent: 12 %
DFS precent: 0 %
HDFS Usages
DFS: 0 %
NonDfs: 12 %
Free: 88 %
Exadoop.com
Free
NonDFS
Hostname IP HDFS Status
1 node2 192.168.128.152
Free
Used: 28.01 KB / Total: 16.29 GB
http://blog.csdn.net/fansy1990


EasyHadoop 管理中心
Map/Reduce
Map
Reduce

开发这样一个平台需要多少人

全栈式？一个就够了

或者

一个牛逼前端加一个牛逼后端

平台开发的核心是要知道有这些接口可以使用。

平台最好做成前后端分离的方式，方便扩展新的接口。

Spark和Hadoop还留有一些隐藏的接口没有写入文档，需要看源码来发现或简化接口的调用。

Conclusion & Thanks a lot

It's very simple to build an AD-HOC query/management system on Hadoop & Spark.

My recently opensource project plan:

pyHUI - Hadoop User Interface based on python-tornado

Contact me @:

https://github.com/xianglei

or

mailto:horseman@163.com

C' ya