SequoiaDB Spark Yarn部署及案例演示

1、 背景

由于MRv1在扩展性、可靠性、资源利用率和多框架等方面存在明显的不足，在Hadoop MRv2中引入了资源管理和调度系统YARN。YARN是 Hadoop MRv2计算机框架中构建的一个独立的、通用的资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。主要体现在以下几个方面：

（1）资源利用率大大提高。一种计算框架一个集群，往往会由于应用程序数量和资源需求的不均衡性，使得在某段时间有些计算框架集群资源紧张，而另外一些集群资源空闲。共享集群模式则通过多种框架共享资源，使得集群中的资源得到更加充分的利用；

（2）运维成本大大降低。共享集群模式使得少数管理员就可以完成多个框架的统一管理；

（3）共享集群的模式也让多种框架共享数据和硬件资源更为方便。

2、 产品介绍

巨杉数据库SequoiaDB是一款分布式非关系型文档数据库，可以被用来存取海量非关系型的数据，其底层主要基于分布式，高可用，高性能与动态数据类型设计，它兼顾了关系型数据库中众多的优秀设计：如索引、动态查询和更新等，同时以文档记录为基础更好地处理了动态灵活的数据类型。并且为了用户能够使用常见的分布式计算框架，SequoiaDB可以和常见分布式计算框架如Spark、Hadoop、HBase进行整合。本文主要讲解SequoiaDB与Spark、YARN的整合以及通过一个案例来演示MapReduce分析存储在SequoiaDB中的业务数据。

3、 环境搭建

3.1、 服务器分布

服务器	服务名称
192.168.1.46	
192.168.1.47	
192.168.1.48	NameNode、DataNode、

3.2、 软件配置

操作系统：RedHat6.5
JDK 版本：1.7.0_80 64 位
Scala 版本：
Hadoop 版本：2.7.2
Spark 版本：2.0
SequoiaDB 版本：2.0

3.3、 安装步骤

1、JDK 安装

```
tar  -xvf  jdk-7u45-linux-x64.tar.gz  －C /usr/local
cd  /usr/local
ln  -s  jdk1.7.0_45  jdk
```

配置环境变量

```
vim  ~/.bash_profile
export  JAVA_HOME=/usr/local/jdk
export  CLASS_PATH=$JAVA_HOME/lib:$JAVA_HOME/jre/lib
export  PATH=$PATH:$JAVA_HOME/bin
source  /etc/profile
```

2、Scala 安装

```
tar  -xvf  scala-2.11.8.tgz  －C /usr/local
cd  /usr/local
ln  -s  scala-2.11.8 scala
```

配置环境变量

```
vim  ~/.bash_profile
export SCALA_HOME=/usr/local/scala
export PATH=$PATH:$SCALA_HOME/bin
```

3、修改主机 hosts 文件配置
在每台主机上修改 host 文件

```
vim /etc/hosts
192.168.1.46 node01
192.168.1.47 node02
192.168.1.48  master
```

4、　SSH 免密钥登录
在 master 节点中执行 ssh-keygen 按回车键

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

将master节点中的授权文件authorized_keys传输到slave节点中

```
scp ~/.ssh/id_rsa.pub root@master:~/.ssh/
```

在slave节点中执行

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

在slave节点中验证SSH免密钥登录

```
ssh master
```

5、Hadoop集群安装

拷贝hadoop文件hadoop-2.7.2.tar.gz到/opt目录中

解压hadoop安装包

```
tar －xvf hadoop-2.7.2.tar.gz
mv hadoop-2.7.2 /opt/cloud/hadoop
```

创建hadoop数据存储及临时目录

```
mkdir －p /opt/hadoop/data
mkdir －p /opt/hadoop/tmp
```

配置Hadoop jdk环境变量

```
vim hadoop-env.sh
export JAVA_HOME=/usr/local/jdk
```

编辑core.xml文件

```
<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://master:9000</value>
   </property>
   <property>
      <name>hadoop.tmp.dir</name>
      <value>/opt/data/tmp</value>
   </property>
   <property>
      <name>io.file.buffer.size</name>
      <value>4096</value>
   </property>
</configuration>
```

编辑mapred-site.xml

```
<configuration>
```

```
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
   <property>
      <name>mapreduce.jobtracker.http.address</name>
      <value> master:50030</value>
   </property>
   <property>
      <name>mapreduce.jobhistory.address</name>
      <value> master:10020</value>
   </property>
   <property>
      <name>mapreduce.jobhistory.webapp.address</name>
      <value>master:19888</value>
   </property>
</configuration>
```

编辑 hdfs-site.xml

```
<configuration>
   <property>
      <name>dfs.nameservices</name>
      <value>master</value>
   </property>
   <property>
      <name>dfs.namenode.secondary.http-address</name>
      <value> master:50090</value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>file:///opt/hadoop/data/name</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>file:///opt/hadoop/data</value>
   </property>
   <property>
      <name>dfs.replication</name>
      <value>3</value>
   </property>
   <property>
      <name>dfs.webhdfs.enabled</name>
      <value>true</value>
   </property>
</configuration>
```

编辑 yarn-site.xml

```
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
   <property>
      <name>yarn.resourcemanager.address</name>
      <value> master:8032</value>
   </property>
   <property>
      <name>yarn.resourcemanager.scheduler.address</name>
      <value> master:8030</value>
   </property>
   <property>
      <name>yarn.resourcemanager.resource-tracker.address</name>
      <value> master:8031</value>
   </property>
   <property>
      <name>yarn.resourcemanager.admin.address</name>
      <value> master:8033</value>
   </property>
   <property>
      <name>yarn.resourcemanager.webapp.address</name>
      <value>master:8088</value>
   </property>
   <property>
      <name>yarn.nodemanager.resource.memory-mb</name>
      <value>12288</value>
   </property>
  <property>
      <name>yarn.nodemanager.log-dirs</name>
      <value>/opt/hadoop/tmp/userlogs</value>
   </property>
</configuration>
```

启动 Hadoop
首次启动集群时，做如下操作
进入到/opt/cloud/hadoop/bin 目录中执行./hdfs namenode －format 格式化

hdfs 文件系统

进入到/opt/cloud/hadoop/sbin目录中执行./start-all.sh启动hadoop集群

6、安装Spark集群

拷贝Spark安装包到/opt目录中，解压

```
tar  －xvf spark-2.0.0-bin-hadoop2.7.tgz
mv spark-2.0.0-bin-hadoop2.7 /opt/cloud/spark
```

编辑spark-env.sh

```
vim spark-env.sh
JAVA_HOME="/usr/jdk1.7"
SPARK_DRIVER_MEMORY="1g"
SPARK_EXECUTOR_CORES=1
SPARK_EXECUTOR_MEMORY="512m"
SPARK_MASTER_PORT="7077"
SPARK_MASTER_WEBUI_PORT="8070"
SPARK_CLASSPATH="/opt/cloud/spark/jars/sequoiadb.jar:/opt/cloud/spark/jars/spark-sequoiadb_2.11-2.6.0.jar"
SPARK_MASTER_IP="node03"
SPARK_WORKER_MEMORY="712m"
SPARK_WORKER_CORES=1
SPARK_WORKER_INSTANCES=1
SPARK_WORKER_DIR="/opt/data/spark/work"
SPARK_LOCAL_DIRS="/opt/data/spark/tmp"
HADOOP_HOME="/opt/cloud/hadoop"
HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

编辑 slaves

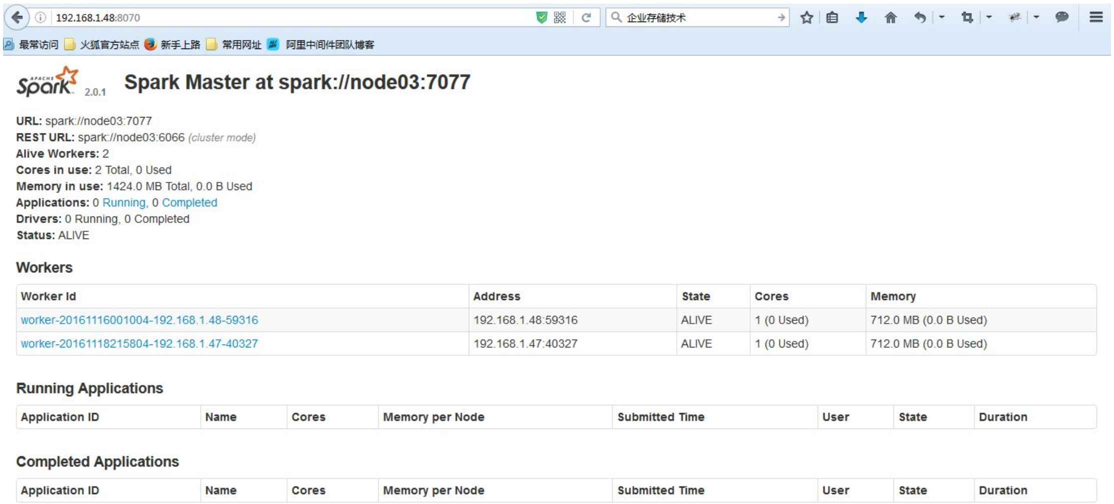
```
node02
node03
```

启动spark集群

进入到目录/opt/cloud/spark/sbin目录中

```
./start-all.sh
```

Spark成功启动后截图如下：



Spark Master at spark://node03:7077

URL: spark://node03:7077
REST URL: spark://node03:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 2 Total, 0 Used
Memory in use: 1424.0 MB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20161116001004-192.168.1.48-59316	192.168.1.48:59316	ALIVE	1 (0 Used)	712.0 MB (0.0 B Used)
worker-20161118215804-192.168.1.47-40327	192.168.1.47:40327	ALIVE	1 (0 Used)	712.0 MB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

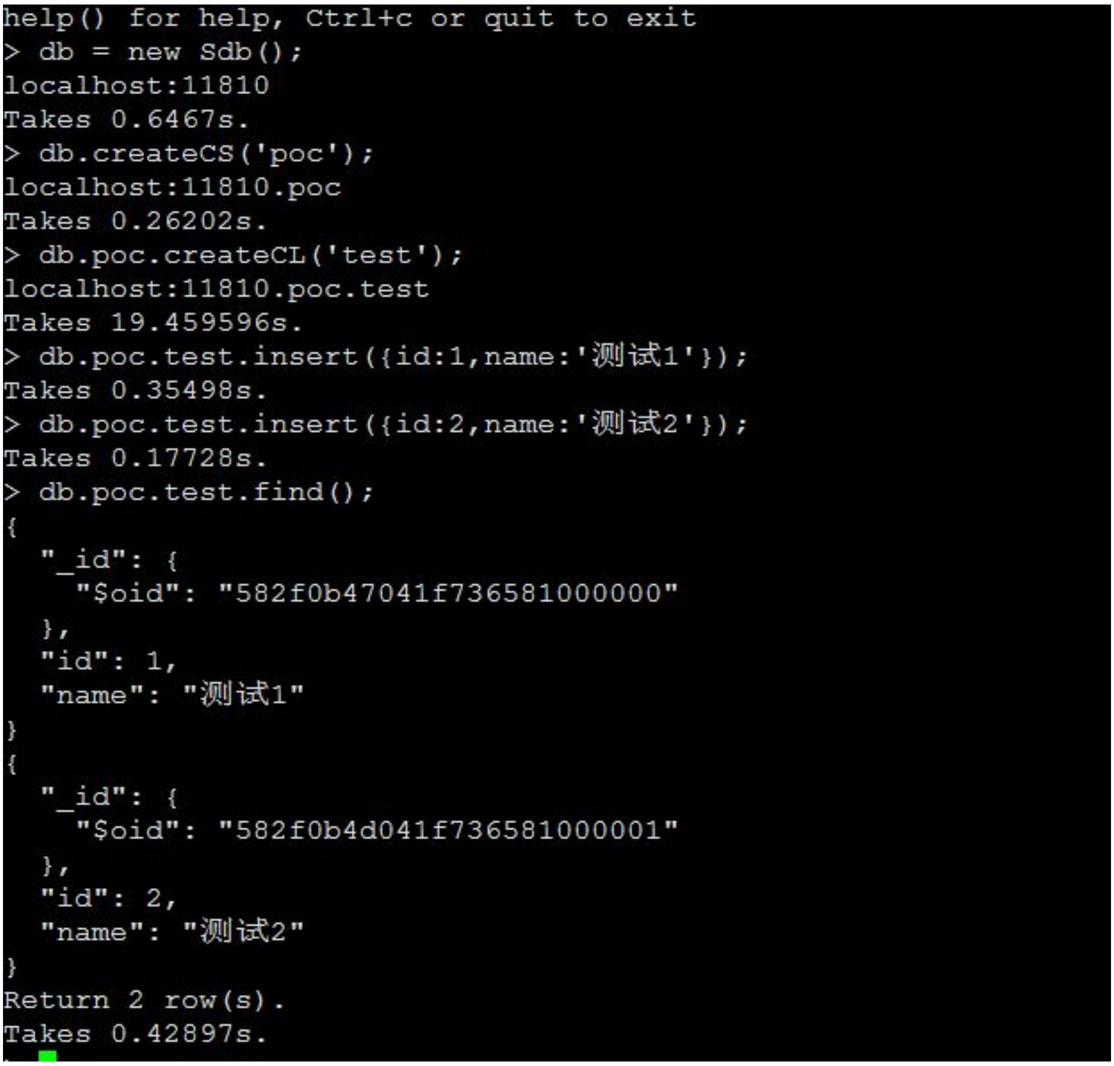
Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

7、Spark Yarn 连接 SequoiaDB

在 SequoiaDB 中创建集合空间、集合

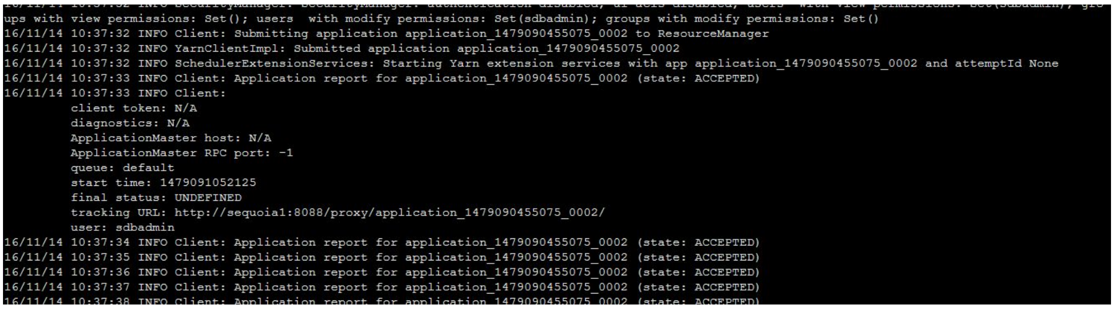
db.createCS('poc');

db.poc.createCL('test');



```
help() for help, Ctrl+c or quit to exit
> db = new Sdb();
localhost:11810
Takes 0.6467s.
> db.createCS('poc');
localhost:11810.poc
Takes 0.26202s.
> db.poc.createCL('test');
localhost:11810.poc.test
Takes 19.459596s.
> db.poc.test.insert({id:1,name:'测试1'});
Takes 0.35498s.
> db.poc.test.insert({id:2,name:'测试2'});
Takes 0.17728s.
> db.poc.test.find();
{
  "_id": {
    "$oid": "582f0b47041f736581000000"
  },
  "id": 1,
  "name": "测试1"
}
{
  "_id": {
    "$oid": "582f0b4d041f736581000001"
  },
  "id": 2,
  "name": "测试2"
}
Return 2 row(s).
Takes 0.42897s.
```

进入到 spark 安装目录 bin 中，执行./spark-sql –master yarn 启动 spark sql 交互界面



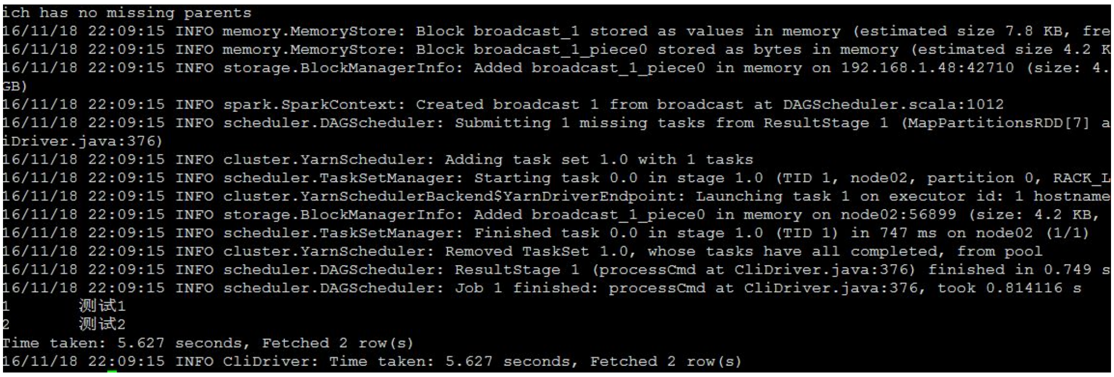
```
16/11/14 10:37:32 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(sdbadmin); gro
ups with view permissions: Set(); users  with modify permissions: Set(sdbadmin); groups with modify permissions: Set()
16/11/14 10:37:32 INFO Client: Submitting application application_1479090455075_0002 to ResourceManager
16/11/14 10:37:32 INFO YarnClientImpl: Submitted application application_1479090455075_0002
16/11/14 10:37:32 INFO SchedulerExtensionServices: Starting Yarn extension services with app application_1479090455075_0002 and attemptId None
16/11/14 10:37:33 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
16/11/14 10:37:33 INFO Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: N/A
         ApplicationMaster RPC port: -1
         queue: default
         start time: 1479091052125
         final status: UNDEFINED
         tracking URL: http://sequoia1:8088/proxy/application_1479090455075_0002/
         user: sdbadmin
16/11/14 10:37:34 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
16/11/14 10:37:35 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
16/11/14 10:37:36 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
16/11/14 10:37:37 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
16/11/14 10:37:38 INFO Client: Application report for application_1479090455075_0002 (state: ACCEPTED)
```

创建表，映射到上述 poc 集合空间中 test 集合

```
CREATE TABLE `test` (`id` INT, `name` STRING)
USING com.sequoiadb.spark
OPTIONS (
  `collection` 'test',
  `host` 'node02:11810,node03:11810',
  `serialization.format` '1',
  `collectionspace` 'poc'
);
```

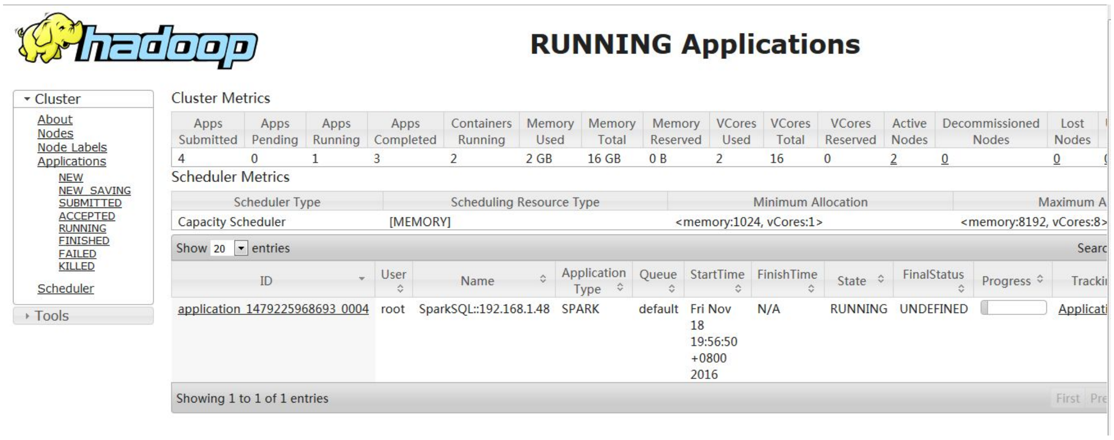
查询表 test 数据，执行：

```
Select * from test;
```



```
ich has no missing parents
16/11/18 22:09:15 INFO memory.MemoryStore: Block broadcast_1 stored as values in memory (estimated size 7.8 KB, fre
16/11/18 22:09:15 INFO memory.MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 4.2 K
16/11/18 22:09:15 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on 192.168.1.48:42710 (size: 4.
GB)
16/11/18 22:09:15 INFO spark.SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:1012
16/11/18 22:09:15 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[7] a
iDriver.java:376)
16/11/18 22:09:15 INFO cluster.YarnScheduler: Adding task set 1.0 with 1 tasks
16/11/18 22:09:15 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, node02, partition 0, RACK_L
16/11/18 22:09:15 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Launching task 1 on executor id: 1 hostname
16/11/18 22:09:15 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on node02:56899 (size: 4.2 KB,
16/11/18 22:09:15 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 747 ms on node02 (1/1)
16/11/18 22:09:15 INFO cluster.YarnScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
16/11/18 22:09:15 INFO scheduler.DAGScheduler: ResultStage 1 (processCmd at CliDriver.java:376) finished in 0.749 s
16/11/18 22:09:15 INFO scheduler.DAGScheduler: Job 1 finished: processCmd at CliDriver.java:376, took 0.814116 s
1       测试1
2       测试2
Time taken: 5.627 seconds, Fetched 2 row(s)
16/11/18 22:09:15 INFO CliDriver: Time taken: 5.627 seconds, Fetched 2 row(s)
```

进入到 yarn 管理页面查看 spark 任务



RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes
4	0	1	3	2	2 GB	16 GB	0 B	2	16	0	2	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum A
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries   Searc

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracki
application_1479225968693_0004	root	SparkSQL::192.168.1.48	SPARK	default	Fri Nov 18 19:56:50 +0800 2016	N/A	RUNNING	UNDEFINED		Applicati

Showing 1 to 1 of 1 entries   First Pre

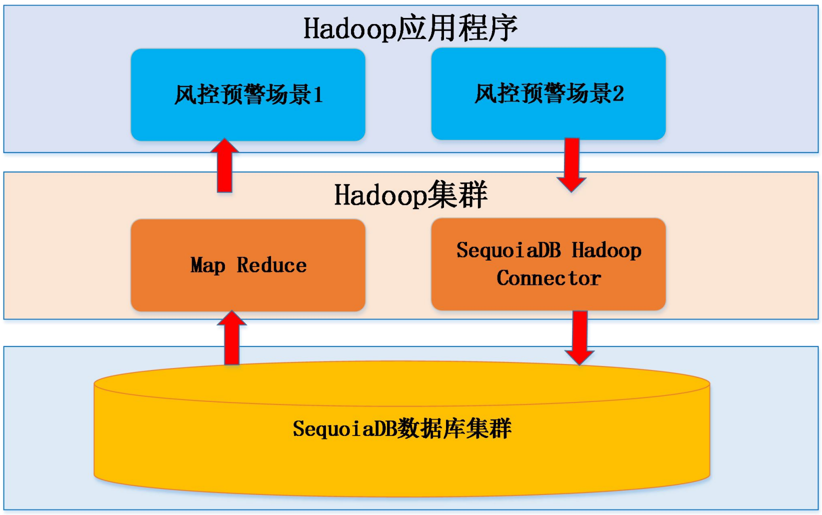
5、 案例演示

为了配合司法部门的执法和银行内部的风险监管，部分商业银行对于存取款业务定制了相关预警方案，本案例以个人存取款业务高频交易来讲述 MapReduce 如何分析 SequoiaDB 中的个人交易明细数据。

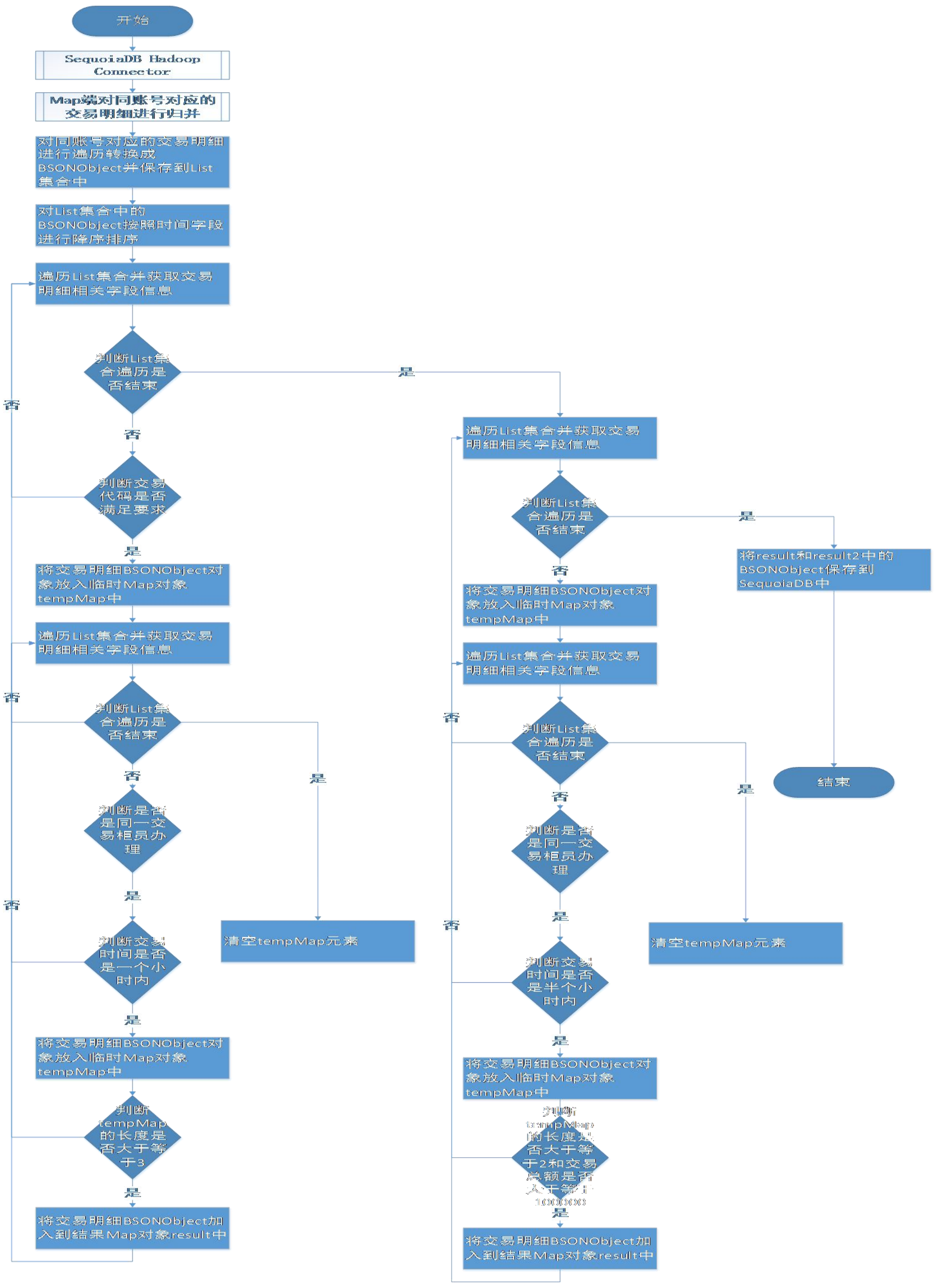
具体场景为：分析同一实体柜员办理，1 小时内同一账户连续 3 笔以上支取类金额的交易账户及明细。

本演示案例采用 Hadoop Map Reduce 实现，开发语言为 Java 语言。整个测试程序分为两个部分 Map 算法和 Reduce 算法。演示程序中 Map 算法负责将同一个账号的所有对应交易明细归并在一起并输出给 Reduce 端，Reduce 端根据 Map 算法的结果运算具体的业务场景，最后将运算结果写入到 SequoiaDB 中。

具体架构如下：



Reduce 端具体算法流程如下：


开始
SequoiaDB Hadoop Connector
Map端对同账号对应的交易明细进行归并
对同账号对应的交易明细进行遍历转换成BSONObject并保存到List集合中
对List集合中的BSONObject按照时间字段进行降序排序
遍历List集合并获取交易明细相关字段信息
判断List集合遍历是否结束
是
否
判断交易代码是否满足要求
是
将交易明细BSONObject对象放入临时Map对象tempMap中
遍历List集合并获取交易明细相关字段信息
判断List集合遍历是否结束
否
是
判断是否是同一交易柜员办理
是
判断交易时间是否是一个小时内
是
将交易明细BSONObject对象放入临时Map对象tempMap中
判断tempMap的长度是否大于等于3
是
将交易明细BSONObject加入到结果Map对象result中
清空tempMap元素
否
否
遍历List集合并获取交易明细相关字段信息
判断List集合遍历是否结束
是
否
将result和result2中的BSONObject保存到SequoiaDB中
将交易明细BSONObject对象放入临时Map对象tempMap中
遍历List集合并获取交易明细相关字段信息
判断List集合遍历是否结束
否
是
判断是否是同一交易柜员办理
是
判断交易时间是否是半个小时内
是
将交易明细BSONObject对象放入临时Map对象tempMap中
判断tempMap的长度是否大于等于2和交易总额是否大于等于1000000
是
将交易明细BSONObject加入到结果Map对象result中
清空tempMap元素
否
否
结束

Map 端算法代码如下：

```
static class TMapper extends Mapper<Object, BSONWritable,Text,BSONWritable>{
        @Override
        protected void map(Object key, BSONWritable value, Context context)
                throws IOException, InterruptedException {
                        BSONObject obj = value.getBson();
            String acct_no=(String) obj.get("ACCT_NO");
            context.write(new Text(acct_no), value);
        }
    }
```

Reduce 端算法代码如下：

```
static class TReducer extends Reducer<Text,BSONWritable,Nu
llWritable,NullWritable>{
        private static String pattern = "yyyy-MM-dd HH:mm:ss
";
        private DateFormat df = new SimpleDateFormat(patter
n);
        private static int tradeNum1 = 3;
        private static int tradeTime1 = 3600;

        private static int tradeNum2 = 2;
        private static int tradeTime2 = 1800;
        private static int tradeAll = 100000;

        private Sequoiadb sdb = null;
        private CollectionSpace cs = null;
        private DBCollection cl_1 = null;
        private DBCollection cl_2 = null;
        private static String CS_NAME="";
        private static String CL_NAME_1="";
        private static String CL_NAME_2="";

        public TReducer(){
            if (null == sdb) {
                sdb = ConnectionPool.getInstance().getConnecti
on();
            }

            if (sdb.isCollectionSpaceExist(CS_NAME)) {
                cs = sdb.getCollectionSpace(CS_NAME);
            } else {
                throw new BaseException("集合空间" + CS_NAME + "
不存在！");
            }

            if (null == cs) {
                throw new BaseException("集合空间不能为null！");
```

```
         } else {
            this.cl_1 = cs.getCollection(CL_NAME_1);

         }
         if (null == cs) {

            throw new BaseException("集合空间不能为null！");

         } else {
            this.cl_2 = cs.getCollection(CL_NAME_2);

         }
      }

       @Override
       protected void reduce(Text key, Iterable<BSONWritab
le> values,
             Context context)
             throws IOException, InterruptedException{
          Iterator<BSONWritable> iterator=values.iterator
();
          long sum=0;
          List<BSONWritable> oldList = new ArrayList<BSONW
ritable>();

          while(iterator.hasNext()){
            BSONWritable bsonWritable = iterator.next();
            oldList.add(bsonWritable);
          }

          //对values进行排序，排序字段为TRN_TIME（交易时间）

          Collections.sort(oldList, new Comparator<BSONWri
table>() {
            @Override
            public int compare(BSONWritable o1, BSONWritab
le o2) {
               String trn_time1 = (String)o1.getBson().get
("TRN_TIME");
               String trn_time2 = (String)o2.getBson().get
("TRN_TIME");
               return trn_time2.compareTo(trn_time1);
            }
         });

          Map<String,BSONWritable> result = new HashMap<St
```

```
ring,BSONWritable>();
          if(oldList != null && oldList.size() > 0){
            //记录同一账户满足条件的笔数
            Map<String,BSONWritable> tempMap = new HashMap
<String,BSONWritable>();
            for(int i=0;i<oldList.size()-1;i++){
               BSONWritable bSONWritable1 = oldList.get
(i);
               //交易代码
               String trn_cd = (String)bSONWritable1.getBs
on().get("TRN_CD");
               if(trn_cd.equals("000045") || trn_cd.equals
("001045")
                     || trn_cd.equals("021031") || trn_cd.
equals("020031")
                     || trn_cd.equals("001060") || trn_cd.
equals("000060")){
                  //交易柜员
                  String tran_teller_no1 = (String)bSONWri
table1.getBson().get("TRAN_TELLER_NO");
                  //流水号
                  String jrnl_no = (String)bSONWritable1.g
etBson().get("JRNL_NO");
                  //交易日期
                  String trn_date1 = (String)bSONWritable1.
getBson().get("TRN_DATE");
                  //交易时间
                  String trn_time1 = (String)bSONWritable1.
getBson().get("TRN_TIME");
                  Date bigDate = null;
                  try {
                     bigDate = df.parse(trn_date1+" "+trn_
time1);
                  } catch (ParseException e) {
                     e.printStackTrace();
                  }
                  tempMap.put(jrnl_no,bSONWritable1);
                  for(int j=i+1;j<oldList.size();j++){
                    BSONWritable bSONWritable2 = oldList.get(j);
                    //交易代码
                    String trn_cd1 = (String)bSONWritable2.getBson().get("TRN_CD");
                    if(trn_cd1.equals("000045") || trn_cd1.equals("001045")
                            || trn_cd1.equals("021031") || trn_cd1.equals("020031")
                            || trn_cd1.equals("001060") || trn_cd1.equals("000060")){
                        //交易柜员
                        String tran_teller_no2 = (String)bSONWritable2.getBson().get("TRAN_TELLER_NO");
                        //流水号
                        String jrnl_no2 = (String)bSONWritable2.getBson().get("JRNL_NO");
                        //交易日期
                        String trn_date2 = (String)bSONWritable2.getBson().get("TRN_DATE");
                        //交易时间
                        String trn_time2 = (String)bSONWritable2.getBson().get("TRN_TIME");
                        Date smallDate = null;
                        try {
                            smallDate = df.parse(trn_date1+" "+trn_time1);
                        } catch (ParseException e) {
                            e.printStackTrace();
                        }
                        //判断是否是同一实体{交易柜员}办理
                        if(!tran_teller_no1.equals(tran_teller_no2)){
                            continue;
                        }
                        //判断{交易日期}{交易时间}是否是[1小时]内
                        if((bigDate.getTime()-smallDate.getTime())/1000 > tradeTime1){
                           break;
                        }
                        tempMap.put(jrnl_no2,bSONWritable2);

                     }else{ //end if TRN_CD1.equals("000045")

                        continue;
                     }
                  }//end for

                  if(tempMap.size() >= tradeNum1){
                     result.putAll(tempMap);
                     tempMap.clear();
                  }
               }else{
                  continue;
               }//end if ||
            }//end for
          }

          Map<String,BSONWritable> result2 = new HashMap<String,BSONWritable>();

          List<BSONObject> cl_1_list = new ArrayList<BSONObject>();
          //结果写入sdb
          Iterator iter1 = result.keySet().iterator();
          while(iter1.hasNext()){
            String keyValue = (String)iter1.next();
            BSONWritable resultValue = result.get(keyValue);

            cl_1_list.add(resultValue.getBson());
            cl_1.insert(resultValue.getBson());
          }
          cl_1.bulkInsert(cl_1_list, DBCollection.FLG_INSERT_CONTONDUP);
          cl_1_list = null;
          List<BSONObject> cl_2_list = new ArrayList<BSONObject>();
          context.write(null,null);
```

```
        }
    }
```