

第一部分：Hbase 数据导入.....	1
Sqoop 批量导入.....	1
写程序 Mapredue 批量导入.....	1
HBase 自带 jar 包导入.....	1
第二部分：Hbase 二级索引方案.....	1
第三部分：基于 solr 二级索引方案.....	2
概述.....	2
方案原理.....	3
部署.....	3
第三部分：批量索引建立以及程序查询.....	10
Hbase 历史数据索引建立.....	10
编写程序查询.....	11
HBase 建立索引的源码分析（待写）.....	17
删除 solr 索引.....	17
第四部分：solr 调优.....	18
调优背景.....	18
Solr 调优项.....	18
第五部分：基于 Impala/hive SQL 语句快速查询/分析方案.....	21
简要介绍.....	21
部署.....	21

第一部分：Hbase 数据导入

Sqoop 批量导入

写程序 Mapredue 批量导入

HBase 自带 jar 包导入

第二部分：Hbase 二级索引方案

参考：<http://www.zuiyuemuai.com/wordpress/?p=1712>

里面讲了大多数公司使用的二级索引方案以及简单的原理

第三部分：基于 solr 二级索引方案

概述

在 Hbase 中,表的 RowKey 按照字典排序, Region 按照 RowKey 设置 split point 进行 shard, 通过这种方式实现的全局、分布式索引. 成为了其成功的最大的砝码。

然而单一的通过 RowKey 检索数据的方式,不再满足更多的需求, 查询成为 Hbase 的瓶颈, 人们更加希望像 Sql 一样快速检索数据, 可是, Hbase 之前定位的是大表的存储, 要进行这样的查询, 往往是要通过类似 Hive、Pig 等系统进行全表的 MapReduce 计算, 这种方式既浪费了机器的计算资源, 又因高延迟使得应用黯然失色。于是, 针对 HBase Secondary Indexing 的方案出现了。

Solr

Solr 是一个独立的企业级搜索应用服务器, 是 Apache Lucene 项目的开源企业搜索平台,

其主要功能包括全文检索、命中标示、分面搜索、动态聚类、数据库集成, 以及富文本 (如 Word、PDF) 的处理。Solr 是高度可扩展的, 并提供了分布式搜索和索引复制。Solr 4 还增加了 NoSQL 支持, 以及基于 Zookeeper 的分布式扩展功能 SolrCloud。SolrCloud 的说明可以参看: SolrCloud 分布式部署。它的主要特性包括: 高效、灵活的缓存功能, 垂直搜索功能, Solr 是一个高性能, 采用 Java5 开发, 基于 Lucene 的全文搜索服务器。同时对其进行了扩展, 提供了比 Lucene 更为丰富的查询语言, 同时实现了可配置、可扩展并对查询性能进行了优化, 并且提供了一个完善的功能管理界面, 是一款非常优秀的全文搜索引擎。

Solr 可以高亮显示搜索结果, 通过索引复制来提高可用, 性, 提供一套强大 Data Schema 来定义字段, 类型和设置文本分析, 提供基于 Web 的管理界面等。

Key-Value Store Indexer

这个组件非常关键, 是 Hbase 到 Solr 生成索引的中间工具。

在 CDH 中的 Key-Value Indexer 使用的是 Lily HBase NRT Indexer 服务。

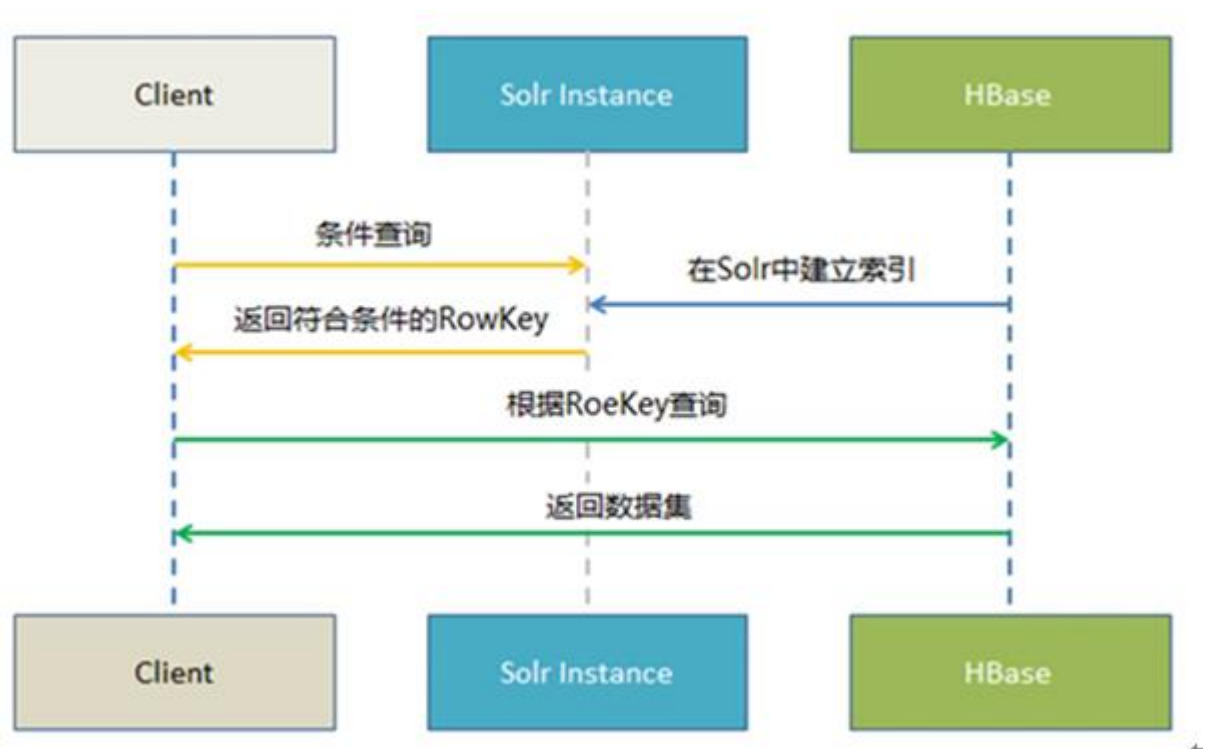
Lily HBase Indexer 是一款灵活的、可扩展的、高容错的、事务性的, 并且近实时的处理 HBase 列索引数据的分布式服务软件。它是 NGDATA 公司开发的 Lily 系统的一部分, 已开放源代码。Lily HBase Indexer 使用 SolrCloud 来存储 HBase 的索引数据, 当 HBase 执行写入、更新或删除操作时, Indexer 通过 HBase 的 replication 功能来把这些操作抽象成一系列的 Event 事件, 并用来保证写入 Solr 中的 HBase 索引数据的一致性。并且 Indexer 支持用户自定义的抽取, 转换规则来索引 HBase 列数据。Solr 搜索结果会包含用户自定义的 columnfamily:qualifier 字段结果, 这样应用程序就可以直接访问 HBase 的列数据。而且 Indexer 索引和搜索不会影响 HBase 运行的稳定性和 HBase 数据写入的吞吐量, 因为索引和搜索过程是完全分开并且异步

的。Lily HBase Indexer 在 CDH5 中运行必须依赖 HBase、SolrCloud 和 Zookeeper 服务。

方案原理

Hbase -----> Key Value Store ---> Solr -----> Web 前端实时查询展示

- 1) Hbase 提供海量数据存储
- 2) Solr 提供索引构建与查询
- 3) Key Value Store 提供自动化索引构建(从 Hbase 到 Solr)



部署

1, 前提

CDH 中部署 Solr 集群, 以及 Key-Value Store Indexer 集群

2, 开启 Hbase 的复制功能

复制

启用向次要区域副本进行复制 RegionServer Default Group ☐

hbase.region.replica.replication.enable

启用编制索引 HBase (服务范围) ☒

启用复制 HBase (服务范围) ☒

hbase.replication

复制来源比例 HBase (服务范围)

replication.source.ratio

3, Hbase 表需要开启 REPLICATION 复制功能

`create 'table',{NAME => 'cf', REPLICATION_SCOPE => 1}` #其中 1 表示开启 replication 功能, 0 表示不开启, 默认为 0

对于已经创建的表可以使用如下命令

```
disable 'table'
alter 'table',{NAME => 'wzx', REPLICATION_SCOPE => 1}
enable 'table'
```

4, 生成实体配置文件

```
solrctl instancedir --generate /opt/cdhsolr/waslog
```

5, 编辑生成好的 scheme.xml 文件

把 hbase 表中需要索引的列添加到 scheme.xml filed 节点, 其中的 name 属性值要与 Morphline.conf 文件中的 outputField 属性值对应

我这里添加 5 个列

源表列名: Time,User,Content,Frequency,Comment 列族: info
solr 中映射列名: HTime,HUser,HContent,HFrequency,HComment

```
<field name="HTime" type="string" indexed="true" stored="false"/>
<field name="HUser" type="string" indexed="true" stored="false"/>
<field name="HContent" type="string" indexed="true" stored="false"/>
<field name="HFrequency" type="string" indexed="true" stored="false"/>
<field name="HComment" type="string" indexed="true" stored="false"/>
```

```

<fields>
<field name="HTime" type="string" indexed="true" stored="false"/>
<field name="HUser" type="string" indexed="true" stored="false"/>
<field name="HContent" type="string" indexed="true" stored="false"/>
<field name="HFrequency" type="string" indexed="true" stored="false"/>
<field name="HComment" type="string" indexed="true" stored="false"/>

  <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />

  <!-- points to the root document of a block of nested documents. Required for nested
       document support, may be removed otherwise
  -->
  <field name="_root_" type="string" indexed="true" stored="false"/>

```

每个字段的解释：

type 参数代表索引数据类型，我这里将 **type** 全部设置为 **string** 是为了避免异常类型的数据导致索引建立失败，正常情况下应该根据实际字段类型设置，比如整型字段设置为 **int**，更加有利于索引的建立和检索；

indexed 参数代表此字段是否建立索引，根据实际情况设置，建议不参与条件过滤的字段一律设置为 **false**；

stored 参数代表是否存储此字段的值，建议根据实际需求只将需要获取值的字段设置为 **true**，以免浪费存储，比如我们的场景只需要获取 **rowkey**，那么只需把 **rowkey** 字段设置为 **true** 即可，其他字段全部设置 **false**；

required 参数代表此字段是否必需，如果数据源某个字段可能存在空值，那么此属性必需设置为 **false**，不然 Solr 会抛出异常；

multiValued 参数代表此字段是否允许有多个值，通常都设置为 **false**，根据实际需求可设置为 **true**。

6，创建 **collection** 实例并配置文件上传到 **zookeeper**

```
solrctl instancedir --create batch /opt/cdhsolr/waslog/conf/
```

```

[zk: localhost:2181(CONNECTED) 3]
[zk: localhost:2181(CONNECTED) 3] ls /solr/configs
[managedTemplateSecure, predefinedTemplateSecure, predefinedTemplate, managedTemplate,
schemalessTemplate, batch, schemalessTemplateSecure]
[zk: localhost:2181(CONNECTED) 4] ls /solr/configs/batch
[admin-extra.menu-top.html, currency.xml, protowords.txt, mapping-FoldToASCII.txt, solrc
onfig.xml.secure, _schema_analysis_synonyms_english.json, _rest_managed.json, solrconfi
g.xml, _schema_analysis_stopwords_english.json, stopwords.txt, lang, spellings.txt, map
ping-ISOLatin1Accent.txt, admin-extra.html, xslt, scripts.conf, synonyms.txt, update-sc
ript.js, velocity, elevate.xml, admin-extra.menu-bottom.html, schema.xml, clustering]
[zk: localhost:2181(CONNECTED) 5]

```

7，上传到 **zookeeper** 之后，其他节点就可以从 **zookeeper** 下载配置文件。接下来创

建 collection

```
solrctl collection --create batch -s 15 -r 4 -m 50
```

这里如果报错，就减少-r 的值

8, 创建 Lily HBase Indexer 配置文件

以便和 solr 建立对应连接关系

```
Vim /opt/cdhsolr/morphline-hbase-mapper.xml

<?xml version="1.0" encoding="UTF-8"?>
<indexer                                table="batch_content"
mapper="com.ngdata.hbaseindexer.morphline.MorphlineResultToSolrMapper"
>
  <param name="morphlineFile" value="morphlines.conf"></param>
  <param name="morphlineId" value="wasMap"></param>
</indexer>
```

其中 morphlineId 的 value 是对应 Key-Value Store Indexer 中配置文件 Morphlines.conf 中 morphlines 属性 id 值

9,修改 Morphlines 文件

具体操作：进入 Key-Value Store Indexer 面板->配置->查看和编辑->属性-Morphline 文件

```
morphlines : [
{
id :wasMap
importCommands : ["org.kitesdk.**", "com.ngdata.**"]

commands : [
{
  extractHBaseCells {
    mappings : [
      {
        inputColumn : "info:time"
        outputField : "HTime"
        type : string
        source : value
      },
      {
```

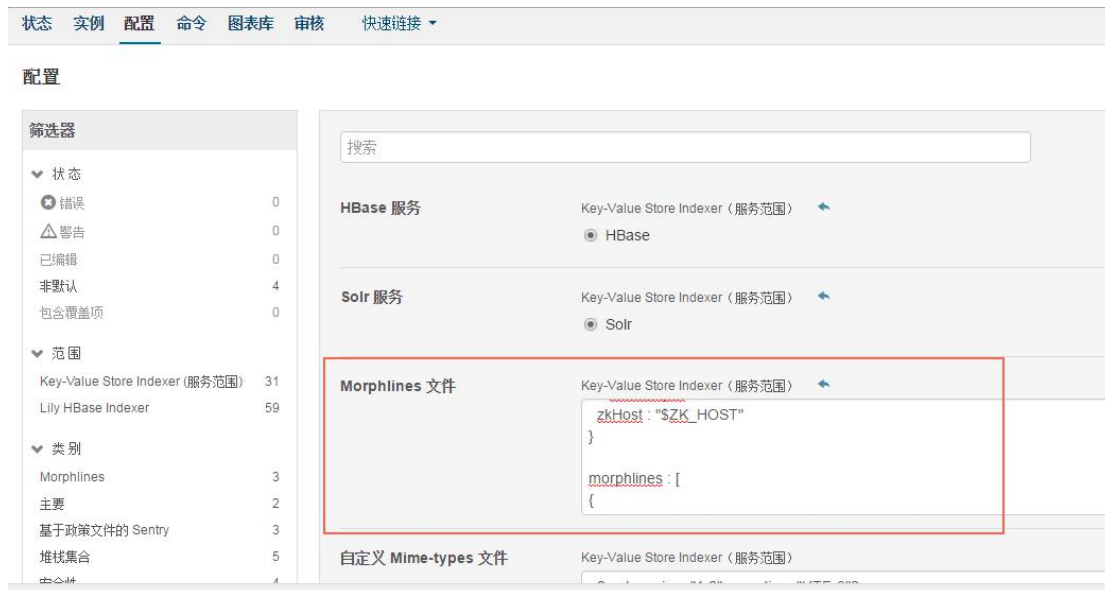
```

        inputColumn : "info:user"
        outputField : "HUser"
        type : string
        source : value
    },
    {
        inputColumn : "info:content"
        outputField : "HContent"
        type : string
        source : value
    },
    {
        inputColumn : "info:frequency"
        outputField : "HFrequency"
        type : string
        source : value
    },
    {
        inputColumn : "info:comment"
        outputField : "HComment"
        type : string
        source : value
    }
]
}
{ logDebug { format : "output record: {}", args : ["@{}"] } }
]
}
]

```

inputColumn:Hbase 的 CLOUMN

outputField:Solr 的 Schema.XML 配置的 fields



10,注册 Lily HBase Indexer configuration 和 Lily Hbase Indexer Service

也就是索引

```
hbase-indexer add-indexer \  
  --name batch_content \  
  --indexer-conf /opt/cdhsolr/morphline-hbase-mapper.xml \  
  --connection-param  
solr.zk=cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/solr \  
  --connection-param solr.collection=batch \  
  --zookeeper  
cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181
```

删除索引

```
hbase-indexer delete-indexer \  
  --name cloudIndexer \  
  --zookeeper  
cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181
```

验证索引器是否成功创建

```
hbase-indexer list-indexers
```



```

batch content
+ Lifecycle state: ACTIVE
+ Incremental indexing state: SUBSCRIBE_AND_CONSUME
+ Batch indexing state: INACTIVE
+ SEP subscription ID: Indexer_batch_content
+ SEP subscription timestamp: 2016-07-28T10:15:37.249+08:00
+ Connection type: solr
+ Connection params:
+ solr.collection = batch
+ solr.zk = cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/s
olr
+ Indexer config:
  266 bytes, use -dump to see content
+ Indexer component factory: com.ngdata.hbaseindexer.conf.DefaultIndexerComponentFact
ory
+ Additional batch index CLI arguments:
  (none)
+ Default additional batch index CLI arguments:
  (none)
+ Processes
+ 1 running processes
+ 0 failed processes

```

这几个地方要正常，哪个 collection 和上面创建的要一样，如果不正常

11, 测试 put 数据查看结果

当写入数据后，稍过几秒我们可以在相对于的 solr 中查询到该插入的数据，表明配置已经成功

在 hbase 里面插入数据。然后界面访问 <http://192.168.100.101:8983/>

The screenshot shows the Apache Solr Admin UI. On the left is a sidebar with navigation links: Dashboard, Logging, Cloud, Core Admin, Java Properties, Thread Dump, and a dropdown menu for 'batch_shard11...'. The main area is divided into two panels. The left panel, titled 'Request-Handler (qt)', shows a search query: `q=id:000anyan000-Tl8xp1`. The right panel shows the JSON response from the query. The response includes a 'responseHeader' with status 0, QTime 269, and indent true. The 'response' section shows 'numFound': 1, 'start': 0, 'maxScore': 15.321125, and a single document with 'id': '000anyan000-Tl8xp1' and '_version_': '1541085701611716600'.

12, 扩展命令

Schema.xml 新增索引字段

执行以下命令更新配置

```
solrctl instancedir --update waslog /opt/cdhsolr /waslog
```

```
solrctl collection --reload waslog
```

查看 collection 命令: `solrctl collection -list`

第三部分：批量索引建立以及程序查询

Hbase 历史数据索引建立

在 CDH 中 Hbase-indexer 提供了 MapReduce 来批量构建索引的方式

```
/opt/cloudera/parcels/CDH-5.7.0-1.cdh5.7.0.p0.45/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.7.0-job.jar
```

构建命令

```
hadoop --config /etc/hadoop/conf \
  jar
/opt/cloudera/parcels/CDH-5.7.0-1.cdh5.7.0.p0.45/lib/hbase-solr/tools/hbase-indexer-mr-1.5-cdh5.7.0-job.jar \
  --conf /etc/hbase/conf/hbase-site.xml \
  -D 'mapred.child.java.opts=-Xmx2048m' \
  --zk-host
'cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/solr' \
  --collection batch \
  --hbase-indexer-file /opt/cdhsolr/morphline-hbase-mapper.xml \
  --go-live --go-live-threads 10 \
  --morphline-file morphlines.conf \
  --output-dir hdfs://jimmy/solr/int \
  --overwrite-output-dir
```

如果 solr 和 MR 没有做过优化, hbase 中的数据量又很大的话会超时错误, solr 会直接挂掉了, 当然也可以小批量的创建, 指定 startkey 和 endkey

```
hadoop --config /etc/hadoop/conf \
  jar
/opt/cloudera/parcels/CDH-5.7.0-1.cdh5.7.0.p0.45/lib/hbase-solr/tools/hbase
```

```
-indexer-mr-1.5-cdh5.7.0-job.jar \  
  --conf /etc/hbase/conf/hbase-site.xml \  
  -D 'mapred.child.java.opts=-Xmx2048m' \  
  --zk-host  
'cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:  
2181/solr' \  
  --collection batch \  
  --hbase-indexer-file /opt/cdhsolr/morphline-hbase-mapper.xml \  
  --go-live --go-live-threads 10 \  
  --morphline-file morphlines.conf \  
  --output-dir hdfs://jimmy/solr/int \  
  --hbase-start-row 000area-565eI1 \  
  --hbase-end-row 000hui-QbXjQb \  
  --overwrite-output-dir
```

也可以做相应的优化，我这里共 2 千万条数据左右，更改的 solr 配置如下，MR 的优化忘记了，改良不少默认值

```
SOLR_MAX_CONNECTOR_THREAD=100000  
  
CATALINA_OPTS="-Djava.net.preferIPv4Stack=true  
-Dsolr.hdfs.blockcache.enabled=true  
-Dsolr.hdfs.blockcache.direct.memory.allocation=true  
-Dsolr.hdfs.blockcache.blocksperbank=16384  
-Dsolr.hdfs.blockcache.slab.count=1 -Xms2147483648 -Xmx2147483648  
-XX:MaxDirectMemorySize=2147483648 -XX:+UseParNewGC  
-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70  
-XX:+CMSParallelRemarkEnabled  
-XX:OnOutOfMemoryError={{AGENT_COMMON_DIR}}/killparent.sh"
```

主要是增加 Solrd 监视程序超时时间以及 java 相关堆内存

编写程序查询

Ok，开始编写程序来体检久违的秒级查询

基于 Solr 的 HBase 多条件查询原理很简单，将 HBase 表中涉及条件过滤的字段和 rowkey 在 Solr 中建立索引，通过 Solr 的多条件查询快速获得符合过滤条件的 rowkey 值，拿到这些 rowkey 之后在 HBASE 中通过指定 rowkey 进行查询。

需要的 jar 包如下：

Hbase jar：直接把 hbase 包里面的 lib 目录加进来就好了

Solr jar：直接把/opt/cloudera/parcels/CDH/lib/solr/webapps/solr/WEB-INF/lib 目录导进去

```

package Base;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.exceptions.DeserializationException;
import org.apache.hadoop.hbase.filter.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.impl.CloudSolrServer;
import org.apache.solr.client.solrj.impl.XMLResponseParser;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.apache.solr.common.SolrDocumentList;

import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;

public class SolrConnection {

    public static void main(String[] args) throws SolrServerException, IOException,
DeserializationException {

        //从 solr 中查询出符合条件的 rowkey, 基于内容查询
        //List<Get> b =
SolrQueryF("cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/solr",
"HContent:大家", "batch", 0, 100);

        //从 solr 中查询出符合条件的 rowkey, 基于 userid 查询
        List<Get> b =
SolrQueryF("cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/solr",
"id:0000sf-*", "batch", 0, 100);

        //定义需要获取的列名数组
        String[] ColcomArray = {"content", "time", "comment", "frequency"};

        //通过上面 solr 查询出来的 rowkey 去 hbase 中查询所需要的列的值, 会返回一个 list
        AtomicReference<List<List<Serializable>>> s =
HbaseQueryRowkey("cmagent1", b, "batch_content", ColcomArray, "info");

```

```

System.out.println("===== "+
s);

    //下面的方法太慢，放弃使用
    //AtomicReference<List<List<Serializable>>> a =
HbaseQueryUserId("cmagent1", "batch_content", ColcomArray, "info", "0000sf");

//System.out.println("=====
"+a);

}

/**
 * zkHost      zookeeper 地址
 * QueryContent 查询内容    格式: 列名: 内容    HContent:大家
 * SolrCollection 索引的 collection 名称
 * BeginNUM     开始行
 * EndNUM       结束行
 *
 * */
public static List<Get> SolrQueryF(String zkHost, String QueryContent, String
SolrCollection, Integer BeginNUM, Integer EndNUM) throws SolrServerException {
    //String zkHost =
"cmagent1:2181,cmagent2:2181,cmagent6:2181,cmagent3:2181,cmagent4:2181/solr";

    CloudSolrServer server = new CloudSolrServer(zkHost);
    server.connect();
    server.setParser(new XMLResponseParser());

    SolrQuery parameters = new SolrQuery();

    /*
    parameters.set("q", "HContent:大家");
    parameters.setStart(0);
    parameters.setRows(10);
    parameters.set("collection", "batch");*/

    parameters.set("q", QueryContent);
    parameters.setStart(BeginNUM);
    parameters.setRows(EndNUM);
    parameters.set("qt", "/select");

```

```

        parameters.set("collection", SolrCollection);

        QueryResponse response = server.query(parameters);

        //SolrDocumentList list;
        SolrDocumentList docs;
        List<Get> list = new ArrayList<>();
        Get get;
        docs = response.getResults();
        System.out.println(docs.size());

        for (SolrDocument doc : docs) {
            get = new Get(Bytes.toBytes((String) doc.getFieldValue("id")));
            list.add(get);
        }
        // System.out.println("list======" + list);

        return list;
    }

    /**
     *
     *
     * zkHosts      zookeeper 地址
     * list          上面 solr 查询出来的 rowkey (List<Get>)
     * TABLE_NAME   表名
     * ColcomArray    需要获取数据的列名
     * ColcomFamliy   列族
     *
     *
     * */
    public static AtomicReference<List<List<Serializable>>> HbaseQueryRowkey(String zkHosts,
List<Get> list, String TABLE_NAME, String[] ColcomArray, String ColcomFamliy) throws IOException
    {
        Configuration conf = HBaseConfiguration.create();

        conf.set("hbase.zookeeper.quorum", zkHosts);
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        System.setProperty("hadoop.home.dir", "C:\\src\\hadoop-2.6.4");

        Connection conn = ConnectionFactory.createConnection(conf);
        Table table = conn.getTable(TableName.valueOf(TABLE_NAME));

```

```

        Result[] res = table.get(list);

        AtomicReference<List<List<Serializable>>> QueryResult = new AtomicReference<> (new
        ArrayList<List<Serializable>>());

        for (Result rs : res) {

            List tmp = new ArrayList<>();

            for (String aColcomArray : ColcomArray) {
                byte[] bt = rs.getValue(ColcomFamliy.getBytes(), aColcomArray.getBytes());
                if (bt != null && bt.length > 0) {
                    System.out.println("bt=====" + new String(bt));
                    tmp.add(new String(bt));
                } else {
                    tmp.add("无数据");
                }
            }

            System.out.println("tmp ===== " + tmp + " ===== rs == " + rs);
            QueryResult.get().add(tmp);
        }
        table.close();
        return QueryResult;
    }

    protected static AtomicReference<List<List<Serializable>>> HbaseQueryUserId(String
    zkHosts, String TABLE_NAME, String[] ColcomArray, String ColcomFamliy, String UserId) throws
    IOException, DeserializationException {
        Configuration conf = HBaseConfiguration.create();

        conf.set("hbase.zookeeper.quorum", zkHosts);
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        System.setProperty("hadoop.home.dir", "C:\\src\\hadoop-2.6.4");

        Connection conn = ConnectionFactory.createConnection(conf);
        Table table = conn.getTable(TableName.valueOf(TABLE_NAME));
        //Result[] res = table.get(list);

        Scan scan = new Scan();
        Filter filter = new RowFilter(CompareFilter.CompareOp.EQUAL,
            new RegexStringComparator(UserId+"-*"));
        Filter fileter1 = new RowFilter(CompareFilter.CompareOp.EQUAL, new
        SubstringComparator(UserId));
    
```

```

        if (UserId != null) { // 设置扫描的范围
            scan.setStartRow(Bytes.toBytes(UserId));
            scan.setFilter(fileter1);
        }
/*
        if (UserId != null) { // 设置扫描的范围
            scan.setStopRow(Bytes.toBytes(stopRow));
            //scan.setFilter(Filter.parseFrom(Bytes.toBytes(UserId+"-")));
        }*/

        ResultScanner res = table.getScanner(scan);

        AtomicReference<List<List<Serializable>>> QueryResult = new AtomicReference<>(new
        ArrayList<List<Serializable>>());

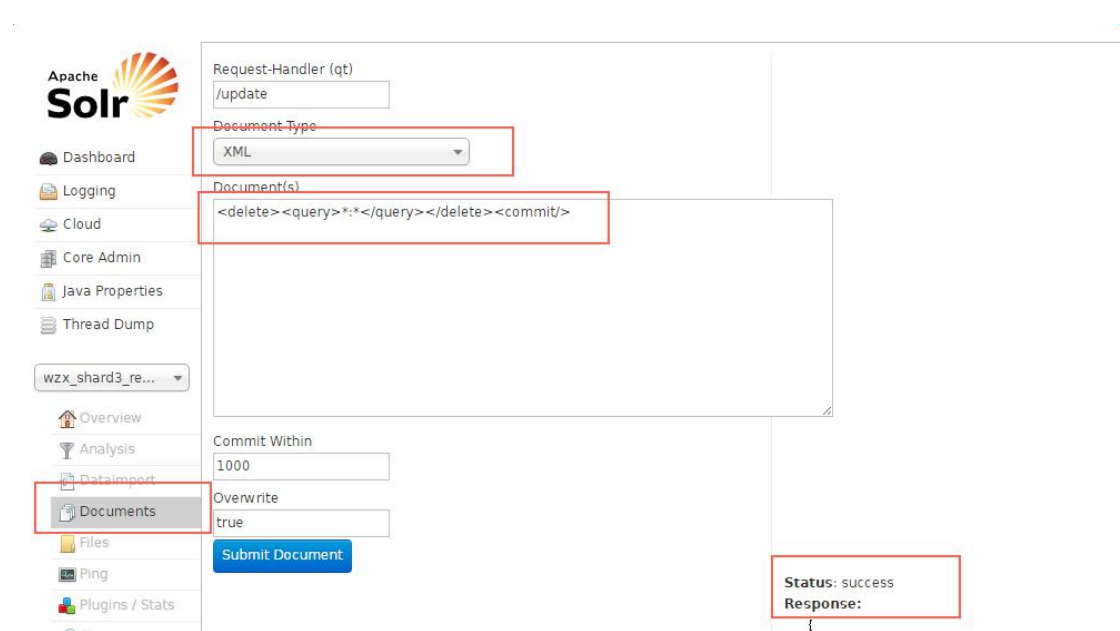
        for (Result rs : res) {

            List tmp = new ArrayList<>();

            for (String aColcomArray : ColcomArray) {
                byte[] bt = rs.getValue(ColcomFamliy.getBytes(), aColcomArray.getBytes());
                if (bt != null && bt.length > 0) {
                    //System.out.println("bt===== "+new String(bt));
                    tmp.add(new String(bt));
                } else {
                    tmp.add("无数据");
                }
            }
            System.out.println("tmp:" + tmp);
            QueryResult.get().add(tmp);
        }
        table.close();
        return QueryResult;
    }
}

```

测试结果



第四部分：solr 调优

调优背景

我们有一个项目列共达到七千多,在对历史数据约三千万条进行建索引时,怎么都建不成功,要吗是 MR 就报错,要吗就是 Live merge 的时候报错, solr 直接挂掉

Solr 调优项


1, Solrd 监视程序超时

Solrd 监视程序超时 Solr (服务范围) 1 小时

2, 堆栈收集频率 stacks_collection_frequency

堆栈收集频率 Solr Server Default Group stacks_collection_frequency 20 分钟

3, Solr Server 记录阈值

Solr Server 记录阈值 Solr Server Default Group 

☐ TRACE

☒ DEBUG

☐ INFO

☐ WARN

☐ ERROR


☐ FATAL

4, HDFS 块缓存碎片数 solr.hdfs.blockcache.slab.count

HDFS 块缓存碎片数 Solr Server Default Group 

solr.hdfs.blockcache.slab.count

5, Solr Server 的 Java 堆栈大小 (字节)

Solr Server 的 Java 堆栈大小 (字节) Solr Server Default Group 

此值内存足够可以配置大点

6, Solr 服务器的 Java 直接内存大小 (字节)

Solr 服务器的 Java 直接内存大小 (字节) Solr Server Default Group 

此值内存足够可以配置大点

调完以上参数, 前面遇到的问题就解决了, 索引建立成功, 当然这里也调理一些 hbase 的参数, 记不清调理哪些了。。。

约三千万条数据, 列共七千多, 建立索引的时间差不多三个小时, 有待继续调优, 如果硬件性能好点, 或许会快点

Dashboard

Logging

Cloud

Core Admin

Java Properties

Thread Dump

www_shard1_r...

Overview

Analysis

Dataimport

Documents

Files

Pin

Request-Handler (qt)

/select

common

q

,

fq

sort

start, rows

010

fl

df

Raw Query Parameters

key1=val1&key2=val2

http://dnode2:8983/solr/www_shard1_replica2/select?q=%3A*&wt=json&in

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 82,
    "params": {
      "indent": "true",
      "q": "*,*",
      "_: "1471231419141",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 29995544,
    "start": 0,
    "maxScore": 1,
    "docs": [
      {
        "id": "07984832103",
        "_version_": 1542457821835034600
      },
      {
        "id": "07984852826",
        "version": 1542457821843423200
      }
    ]
  }
}
```

Dashboard

Logging

Cloud

Core Admin

Java Properties

Thread Dump

www_shard1_r...

Overview

Analysis

Dataimport

Documents

Request-Handler (qt)

/select

common

q

U005001:3224

fq

sort

start, rows

010

fl

df

http://dnode2:8983/solr/www_shard1_replica2/select?q=U00

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 36,
    "params": {
      "indent": "true",
      "q": "U005001:3224",
      "_: "1471231461457",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1408,
    "start": 0,
    "maxScore": 10.99763,
    "docs": [
      {
        "id": "08236019286",
        "_version_": 1542457823028314000
      }
    ]
  }
}
```

第五部分：基于 Impala/hive SQL 语句快速查询/分析方案

简要介绍

我们知道，HBase 是一个基于列的 NoSQL 数据库，它可以实现的数据的灵活存储。它本身是一个大表，在一些应用中，通过设计 RowKey，可以实现对海量数据的快速存储和访问。但是，对于复杂的查询统计类需求，如果直接基于 HBase API 来实现，性能非常差，或者，可以通过实现 MapReduce 程序来进行查询分析，这也继承了 MapReduce 所具备的延迟性。实现 Impala 与 HBase 整合，我们能够获得的好处有如下几个：

- 可以使用我们熟悉的 SQL，像操作传统关系型数据库一样，很容易给出复杂查询、统计分析的 SQL 设计
- Impala 查询统计分析，比原生的 MapReduce 以及 Hive 的执行速度快很多

Impala 与 HBase 整合，需要将 HBase 的 RowKey 和列映射到 Impala 的 Table 字段中。Impala 使用 Hive 的 Metastore 来存储元数据信息，与 Hive 类似，在于 HBase 进行整合时，也是通过外部表（EXTERNAL）的方式来实现。

部署

1，环境部署

Hive, hbase, impala 环境部署，这个就不讲，cloudera 里面部署很方便

2，hbase 创建表

这里直接使用上面 batch_content 表，表结构

源表列名：Time,User,Content,Frequency,Comment 列族：info

3，在 hive 中创建外部表

```
CREATE EXTERNAL TABLE sho.batch(  
    id string,  
    time string,  
    user string,  
    content string,  
    frequency string,
```

```

comment string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe'
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, info:time, info:user, info:content,
info:frequency, info:comment")
TBLPROPERTIES("hbase.table.name" = "batch_content");

```

4, 同步元数据到 impala

Impala 和 hive 共用元数据 metedata 的, 在 hive 中创建完后默认未配置情况下不会自动同步到 impala 中, 可以手动进行同步

```
INVALIDATE METADATA;
```

5, 查询测试

Hive 中查询:

```

hive>
>
> show databases;
OK
default
gdpi
sho
Time taken: 0.047 seconds, Fetched: 3 row(s)
hive>
>
> use sho;
OK
Time taken: 0.082 seconds
hive> show tables;
OK
batch
Time taken: 0.064 seconds, Fetched: 1 row(s)
hive> desc batch;
OK
id                string                from deserializer
time              string                from deserializer
user              string                from deserializer
content           string                from deserializer
frequency         string                from deserializer
comment           string                from deserializer
Time taken: 0.258 seconds, Fetched: 6 row(s)
hive> select * from batch limit 1;
OK
0000pppp-I0Bscx 20160501      NULL    [挤眼]水长城~go~ 北京·九渡河镇 0      0
Time taken: 0.715 seconds, Fetched: 1 row(s)
hive>

```

select * from batch where content like '%九渡河镇%';

```

2016-07-29 23:03:14,677 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 65.62 sec
2016-07-29 23:04:14,708 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 78.81 sec
2016-07-29 23:05:15,586 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 93.42 sec
2016-07-29 23:06:01,980 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 105.97 sec
MapReduce Total cumulative CPU time: 1 minutes 45 seconds 970 msec
Ended Job = job_1469512880806_0029
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 105.97 sec HDFS Read: 4366 HDFS Write: 3203 SUCCESS
Total MapReduce CPU Time Spent: 1 minutes 45 seconds 970 msec
OK
0000pppp-I0Bscx 20160501 NULL 【挤眼】水长城~go~ 北京·九渡河镇 0 0
1014120490-0ZClcc 20160501 NULL 分享图片 北京·九渡河镇 0 0
1691581772-4d8Y8u 20160521 NULL 520。521 北京·九渡河镇 0 0
2385200630-s7Yc9m 20160521 NULL 屎成傻狗.....【微笑】【微笑】 北京·九渡河镇 1 0
2576589753-PWRlIi 20160502 NULL 水长城 北京·九渡河镇 0 0
3228892930-3MvIKw 20160430 NULL 雅阁山房开业啦 地址:北京市延庆区大庄科乡龙泉峪村 预约电话:13701115943 北京·
九渡河镇 0 0
3228892930-bItWD1 20160504 NULL 雅阁山房对面长城 北京·九渡河镇 1 0
3536867383-6mGjcq 20160519 NULL 今天公司团建徒步穿越加露营,晚餐烧烤再加篝火晚会,累了一天总算进帐篷了,真的
被自己的脚臭到了。。。【doge】 北京·九渡河镇 0 0
3662433671-4F2Q3j 20160430 NULL 在城里待久了来农村散散心【美甲】 北京·九渡河镇 4 0
5300156726-8cWdmD 20160510 NULL Morning~ 北京·九渡河镇 1 0
Newyaoyao-fnLrF5 20160504 NULL 老黄带我们怒闯黄花城【酷】【酷】 北京·九渡河镇 0 0
TNqihun-YPz501 20160515 NULL 我愿为你摘下天上的星星 北京·九渡河镇 0 0
elaine8023soulmate-RebLrT 20160429 NULL 山间鱼塘边一直在僵持的一人一猫。【睡】【睡】【睡】 北京·九渡河镇 0 0
msd729-2rpGJq 20160430 NULL 一趟小游 黄昏时候的夕阳。 北京·九渡河镇 0 0
ranzhuo-iqq2nD 20160522 NULL 微博小视频比微信的清楚太多了【汗】 北京·九渡河镇 秒拍视频 . 24 2
yuzhenkun-GcCkJU 20160519 NULL 指责雷洋的网民分两类,一是坚定相信警方通告的雷洋嫖娼,认为其死有余辜,这类网
友是混淆了德与法的概念,并非恶意。另一类是故意的以道德审判掩盖警方执法中的失误,完全无视即使作为犯罪嫌疑人也有法定权力,纯粹
为权力洗地。第一类不是坏人,第二种无疑是坏人。 北京·九渡河镇 0 0
Time taken: 428.68 seconds, Fetched: 16 row(s)
hive>

```

Impala 中查询:

```

+-----+
| _impala_builtins | System database for Impala builtin functions |
| default          | Default Hive database                         |
| gdpi             |                                               |
| sho              |                                               |
+-----+
Fetched 4 row(s) in 0.28s
[cmagent2:21000] > use sho
> ;
Query: use sho
[cmagent2:21000] > show tables;
Query: show tables
+-----+
| name |
+-----+
| batch |
+-----+
Fetched 1 row(s) in 0.01s
[cmagent2:21000] > desc batch;
Query: describe batch
+-----+
| name | type | comment |
+-----+
| id | string | |
| comment | string | |
| content | string | |
| frequency | string | |
| time | string | |
| user | string | |
+-----+
Fetched 6 row(s) in 0.13s
[cmagent2:21000] >

```

select * from batch where content like '%九渡河镇%';

默认配置 MEM_LIMIT 只有 256M, 执行这条命令会提示 Memory limit exceeded 内存不够, 所以改为 3G 了

