基于 Flume 的 MySQL 数据自动收集系统

于金良[1],朱志祥[1],梁小江[2]
(1. 西安邮电大学,陕西 西安 710061;
2. 陕西省信息化工程研究院,陕西 西安 710061)

摘 要:针对分布式系统中、不同数据中心之间的数据收集,同时解决将数据由关系型数据库交换到非关系型数据库的问题,提出一种基于 Flume 的 MySQL 数据库数据自动收集系统。为了符合现实中的生产环境,该系统采用的是一种星型拓扑结构。系统可以自动查询给定的 MySQL 数据库表,自动检测表中的数据更新,实现自动增量传输,并对原始数据进行封装、解析,最终将数据存储到非关系型数据库 HBase 中。在测试中,系统中每台机器的平均传输速度可达到 1 111 kb/s,系统总的平均传输速度可以达到 3 333 kb/s,并且保证了数据的完整性,实现了可靠高效传输的目标。

关键词:Flume;MySQL 数据库;数据收集;HBase;JDBC
中图分类号:TP274.2 **文献标识码**:A **文章编号**:1673-629X(2016)12-0137-05
doi:10.3969/j.issn.1673-629X.2016.12.030

Automatic Collection System for MySQL Data Based on Flume

YU Jin-liang[1],ZHU Zhi-xiang[1],LIANG Xiao-jiang[2]
(1. Xi'an University of Posts and Telecommunications,Xi'an 710061,China;
2. Shaanxi Information Engineering Research Institute,Xi'an 710061,China)

Abstract:For data collecting in distributed systems or between different data centers,while addressing the issue of exchange data from a relational database to non-relational databases,an automatic collecting system for MySQL data based on Flume is put forword. In order to meet the real-world production environment,this system uses a star topology. It can automatically query a given MySQL database table,automatic delection of the data updating of the table,automatic incremental transfer,packaging and parsing to the original data,finally storing data into a database of HBase. In test,the average speed of transmission of every machine in the system can reach 1 111 kb/s,and the total speed of transmission can reach 3 333 kb/s,which ensure data integrity and achieve the goal of reliable and efficient transport.

Key words:Flume;MySQL database;data collecting;HBase;JDBC

0 引 言

近年来,随着信息技术的快速发展,带来的是各种信息、数据的爆发式增长,大数据时代应运而生。2008 年 8 月,首次提出大数据的概念。在大数据时代,TB、PB,甚至 EB 级的数据已经成为一种常态。为了应对大数据的存储、处理以及大型计算机成本高的难题,集群化的分布式系统快速发展并取代了单机服务系统[1]。各大公司和开源社区纷纷提出了自己的大数据解决方案,其中开源的 Hadoop 生态系统最为热门[2]。

Hadoop 是一个并行处理大规模数据的分布式计算和存储系统,可以将分布式系统部署在廉价机器上[3]。要使用 Hadoop 来存储、处理数据,首先要解决的问题是如何将数据收集到 Hadoop 平台上。而将原始关系型数据库中的数据导入到 Hadoop 中的非关系型数据库 HBase 中显得尤为重要。文中系统使用 Hadoop 的子项目 Flume 收集分布式系统中各个计算机中数据库的数据,并最终存储在非关系型数据库 HBase 中[4]。在 Hadoop 生态系统中,Sqoop 可以实现 Hadoop 集群与关系型数据之间的数据交换,但是由于它底层使用的是 MapReduce 计算框架,故依赖于 Hadoop 的集群环境,这是一大缺陷。而文中系统则可以脱离 Hadoop 环境,在构建大数据平台时更加灵活。

1 Flume 简介

Flume 最早是 Cloudera 开源的一个日志收集系统,设计它的初衷是在分布式系统中提供可靠而有效

收稿日期:2016-01-11 **修回日期**:2016-05-05 **网络出版时间**:2016-11-21
基金项目:2015 年工信部通信软科学研究项目(2015-R-19);2015 陕西省信息化技术研究项目(2015-002)
作者简介:于金良(1991-),男,硕士研究生,研究方向为大数据分析处理;朱志祥,教授,研究方向为计算机网络、信息化应用和网络安全。

的大规模日志收集服务[5]。2011 年 10 月,Cloudera 完成了 Flume-728,对 Flume 进行了里程碑式的改动,重构了核心组件、核心配置以及代码结构,重构后的版本统称为 Flume-ng。同时 Cloldera 将 Flume 贡献给了 Apache基金会,成为了顶级项目 Hadoop 中的一个子项目。它的架构非常简单灵活,尤其是当前的 Flume-ng 版本,它只是一个纯粹的数据传输工具,将数据的读入和写出分为两个独立的部分,实现了二者的异步性。一个传输通道只是一个代理,各个代理之间又是相互独立的。每个代理包括 Source(数据源)、Channel(中间传输通道)、Sink(数据接收器)三个部分。

其中 Source 是将数据从数据源读入,封装为一个事件发送到 Channel;Channel 的作用是临时缓存这些事件,为了保证数据的可靠性,当事件被 Sink 接收时才将其删除,否则一直缓存;Sink 负责接收事件后将数据存储到指定的目的端,完成一次数据传输。

Flume 作为日志收集系统,支持多种数据源,如 Exec、Spooling、Kafka、NetCat 和用户自定义的源等;拥有多种接收器,如 HDFS、File Roll、HBase、Kafka、数据仓库 Hive[6] 和用户自定义的接收器等;包括多种 Channel,如 Memory、File 等,其中 File Channel 具有很高的故障恢复能力。

它的使用也非常简单,用户根据自己的需求编写配置文件,启动代理服务,即可完成数据收集。

因为 Flume 是一个开源项目,所以用户可以在原有的架构上自己定制,实现自己的数据收集系统。

2 收集系统

2.1 系统介绍

该系统是基于 Flume-ng 1.6.0 的 MySQL 数据库数据收集系统,收集 MySQL 中的数据,自动监测数据库中的数据更新,实现实时增量收集,最终将数据存储到非关系型数据库 HBase 中。可将原本非 Hadoop 集群中的数据导入到集群中,实现单机系统与分布式系统的数据交换。可在脱离 Hadoop 环境的前提下将数据导入到 Hadoop 集群,具有依赖小、量级轻的优点。

系统需要实现的目标:

(1)可以收集 MySQL 数据库中的原始数据;

(2)自动检测数据库中数据的更新,只收集变化的数据,实现数据的实时增量收集;

(3)将收集到的数据存储到非关系型数据库 HBase 中。

2.2 系统设计

在 Flume 的运行过程中根据设定会运行一个或者多个代理,由每一个代理完成数据的收集服务。每个代理都是一个进程,它们是相互独立的,因此可以实现同时对多个数据源进行并行处理,以达到从分布式系统中的不同计算机上收集数据的目的。代理在运行的过程中,并不依赖于 Hadoop 环境,这使得将非 Hadoop 集群上的数据交换到集群中变得可行[7]。系统架构如图 1 所示。

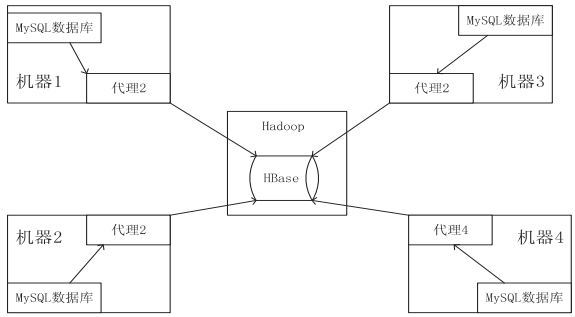

图 1 系统架构图

为安装有 MySQL 数据源的每台机器部署一个 Flume 的代理,将 Source 配置为每台机器的 MySQL 数据源,URL 指定为连接 MySQL 数据库的连接符,形如 jdbc:mysql://IP:port/databasename;Sink 配置为要将数据交换到 Hadoop 集群中的 HBase 数据库。启动每个机器上的代理,向集群发送数据。

代理要完成的主要工作是:

(1)检测指定计算机上 MySQL 数据库表中的数据,实时检测表中数据的更新,并收集、预处理更新数据,预处理完成后记录当前数据行数,防止重复处理;

(2)将收集的数据通过 HBase 客户端插入到 HBase 数据库指定的数据表中。

在代理内部,按数据的流向又可分为 Source、Channel、Sink 三个组件,分别完成不同的任务。Source 组件的任务是查询 MySQL 数据库表中的数据,检查数据更新,预处理更新的数据,且记录处理数据的位置信息(即数据在表中所在的行数),将其保存在一个指定的文件中,并将数据封装为事件发送到 Channel 中;Channel 只负责缓存 Source 经过处理后发送来的事件,等待 Sink 抽取事件,抽取完成后,该事件自动删除;Sink 负责抽取 Channel 中的缓存事件,并进行解析,最终将解析完成的数据存入到 HBase 数据库中。该系统代理架构如图 2 所示。

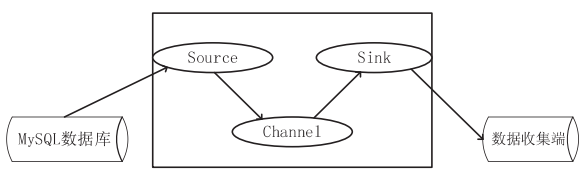

图 2 系统代理架构图

2.3 主要组件的设计与实现

代理内部各组件数据处理流程如图 3 所示。

Source 使用 JDBC 技术查询数据库表中的数据,执行查询时采用分页查询的方法,所谓分页查询就是在

数据量过大时分多次对数据库进行查询[8]。而 MySQL 数据库的分页查询是通过调用 LIMIT 函数实现的[9],在 SQL 语句的结尾处指定每次查询的行数。使查询一次数据库加载到内存里的数据量减小,从而在总数据量变大时减轻内存的压力。对数据库的查询是循环执行的,以达到实时监控数据库更新的目的。当检测到有数据时(第一次查询原始数据或者更新的数据),将这些数据从数据库读入到代理中,通过预处理将其封装为一个事件,然后更新已处理完成数据的位置信息(即数据库中数据的行数),并和数据库连接符拼接在一起保存到指定的记录文件中。在每次对此数据库表进行查询时先读取此文件,获得这个位置信息,查询时从此行开始,之前的数据不再查询,从而避免了重复查询处理,也解决了使用内存通道时不能自动故障恢复的缺陷[10]。当系统发生故障,导致代理宕机时,只需要重新启动代理,从文件中读取这个行数即可从上次处理的位置继续处理,使代理具有很好的可恢复性。下面是 Source 组件的部分源代码。

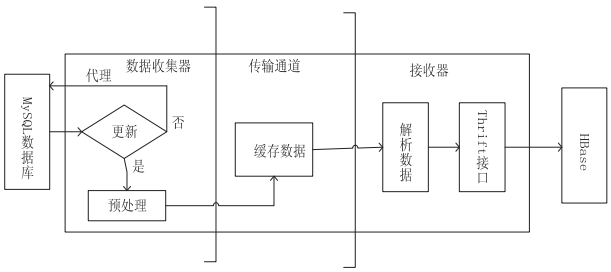

图 3　代理内部各组件数据处理流程图

```
public class MySQLSource extends AbstractSource implements Configurable, PollableSource[11] {
private JdbcHelper jdbcHelper; //通过 JDBC 查询数据库的辅助类
@ Override
public Status process( ) throwsEventDeliveryException {
try{
sqlSourceCounter. startProcess( );
//取得查询数据得到的结果
List<List<String>> result=jdbcHelper. executeQuery( );
//判断是否取得数据
if(! result. isEmpty( )) {
//将数据发到 ChannelcsvWriter. writeAll ( sqlSourceHelper. getAllRows( result) );
//处理完后,更新查询行数 sqlSourceCounter. incrementEventCount( result. size( ) );
//更新记录文件的数据即当前查询数据表中的行数
sqlSourceHelper. updateStatusFile( );
}
sqlSourceCounter. endProcess( result. size( ) );
if( result. size( ) <sqlSourceHelper. getMaxRows( ) )
{
Thread. sleep( sqlSourceHelper. getRunQueryDelay( ) );
}
//返回 READY,执行下一次查询
return Status. READY;
}
```

使用 JDBC 辅助查询数据库的部分代码:

```
public classJdbcHelper{
public List<List<String>>executeQuery( ) throws SQLException{
List<List<String>> rowsList=new ArrayList<List<String>>( );
//对数据库实行分页查询,减小内存的压力,提高查询效率
resultSet=connection. createStatement( ). executeQuery(
( sqlSourceHelper. getQuery ( ) ) +" LIMIT" +sqlSourceHelper. getCurrentIndex( ) +"," +sqlSourceHelper. getMaxRows( ) ) ; ) ;
//将查询的数据缓存到 rowsList 中
while( resultSet. next( ) ) {
List<String> list=newArrayList<String>( );
for( int i=0;i<columnNames. length;i++) { list. add( resultSet. getString( columnNames[ i] ) ) ;
}
rowsList. add( list) ;
}
//更新新的查询起始位置
sqlSourceHelper. setCurrentIndex ( sqlSourceHelper. getCurrentIndex( ) +rowsList. size( ) ) ;
//返回查询结果
returnrowsList;
}
}
```

Channel 的主要作用是缓存 Source 预处理的数据,即事件。只有当事件被 Sink 抽取后才会从当前 Channel 中删除。这种机制保证了数据传递的可靠性。系统采用内存通道,将数据直接缓存在内存中,优点是数据传输速度快,减少数据由读入端到写出端的时延。

Sink 主要负责从 Channel 抽取事件,由于事件包括一个事件头和事件体,事件体中存储的才是真正从 MySQL 中读取的数据,故先将事件体读出,再跟据在配置文件中设定好的规则解析这些数据,将数据分解为与原始字段和值一一对应的形式,最后把这些数据通过调用 HBase 的 Thrift 客户端接口插入到指定的 HBase 表中[12]。下面是插入到 HBase 中部分代码:

```
public class HBaseSink extends AbstractSink implements Configurable{
//开启 HBaseSink
public void start( ){
try{
//先建立通过 Thrift 连接到 Hbase 服务器[13]
client = privilegedExecutor. execute ( new PrivilegedExceptionAction<Hbase. Client>( ) {
```

```
@ Override
public Hbase. Client run( ) throws Exception {
socket = new TSocket( thriftQum, Integer. parseInt( port) ) ;
TProtocol protocol = new TBinaryProtocol( socket, true, true) ;
client = new Hbase. Client( protocol) ;
//打开 Thrift 连接
socket. open( ) ;
return client;
}
} ) ;
} catch( Exception e) {
//TODO Auto-generated catch block
e. printStackTrace( ) ;
sinkCounter. incrementConnectionFailedCount( ) ;
logger. error( "Could not connect hbase" +thriftQum, e) ;
}
//实现接口的方法
@ Override
public Status process( ) throws EventDeliveryException {
//初始化解析数据对象
serializer. initialize( event, columnFamily) ;
//拿到解析后的数据,并存放在 actions 中
actions. add( serializer. getActions( ) ) ;
//调用数据传递给插入 Hbase 的方法
putEventsAndCommit( actions, txn) ;
}
//真正将数据插入到 HBase 中的方法
private void putEventsAndCommit( final List < BatchMutation >
actions, Transactiontxn) throws Exception {
privilegedExecutor. execute ( new PrivilegedExceptionAction <
Void>( ) {
@ Override
public Void run( ) throws Exception {
Map<ByteBuffer, ByteBuffer> attributes = new HashMap<Byte-
Buffer, ByteBuffer>(0) ;
//将数据插入到 Hbase 中名为 tableName 的表中
client. mutateRows( wrap( tableName) , actions, attributes) ;
return null;
}
}
```

下面是 Sink 解析数据的部分代码:

```
public class MyHbaseEventSerializer implements HbaseEvent-
Serializer {
//配置文件中的一些属性:
//分隔数据的字符,默认为空格
public static final String SPLIT_CONFIG = "splitChar" ;
public static final String SPLIT_DEFAULT = "" ;
//Hbase 表中字段的名字
public static final String COL_NAME_CONFIG = "colNames" ;
//数据的编码方式,默认为 UTF-8
public static final String CHARSET_CONFIG = "charset" ;
public static final String CHARSET_DEFAULT = "UTF-8" ;
//插入 Hbase 表中的列族
protected byte[ ] cf;
public void configure( Context context) {
//获得配置文件中的属性值
String splitChar = context. getString ( SPLIT _ CONFIG, SPLIT _
DEFAULT) ;
inputPattern = Pattern. compile( splitChar) ;
charset = Charset. forName( context. getString( CHARSET_CON-
FIG, CHARSET_DEFAULT) ) ;
String colNameStr = context. getString ( COL_NAME_CONFIG,
COLUMN_NAME_DEFAULT) ;
String[ ] columnNames = colNameStr. split( "," ) ;
for( String s: columnNames) {
colNames. add( s. getBytes( charset) ) ;
}
}
//从 Channel 中缓存的事件对象中获得 MySQL 中的数据
@ Override
public void initialize( Event event, byte[ ] columnFamily) {
//从事件(包括 header 和数据)中得到其中的数据
this. payload = event. getBody( ) ;
//获取列族
this. cf = columnFamily;
}
public BatchMutation getActions( ) throws FlumeException {
BatchMutation bm = null;
List<Mutation> mutations = new ArrayList<Mutation>( ) ;
//按约定的规则分隔数据
String[ ] data = inputPattern. split ( new String ( payload, char-
set) ) ;
//以系统当前时间的纳秒数为行键
String rowKey = String. valueOf( System. nanoTime( ) ) ;
Map<String, String> fieldName = null;
try{
fieldName = new HashMap<String, String>( ) ;
for( int i = 0; i<colNames. size( ) ; i++) {
String column = cf+":" +colNames. get( i) ;
//将数据插入到 HBase 中
mutations. add( new Mutation( false, wrap( column) , wrap( data
[i]) , false) ) ;
}
bm = new BatchMutation( wrap( rowKey) , mutations) ;
} catch ( Exception e) {throw newFlumeException( "Could not
get row key!" , e) ;
}
//返回要插入的数据
return bm;
}
```

Flume 配置文件中信息:

```
#配置 Sources
```

```
agent. sources = s
agent. sources. s. type = org. xy. flume. source. MySQLSource
agent. sources. s. connection. url = jdbc: mysql://IP: port/database
agent. sources. s. user = username
agent. sources. s. password = password
agent. sources. s. table = tablename
agent. sources. s. jdbc. connection. driver_class = com. mysql. jdbc. Driver
agent. sources. s. custom. query = SELECT * FROM tablename
agent. sources. s. channels = c
#配置 Channel
agent. channels = c
agent. channels. c. type = memory
agent. channels. c. capacity = 10 000
agent. channels. c. transactionCapacity = 1 000
#配置 sinks
agent. sinks. k. type = hbase
agent. sinks. k. channel = c
agent. sinks. k. table = hbaseTablename
agent. sinks. k. columnFamily = columnFamilyName
agent. sinks. k. serializer = org. apache. flume. sink. hbase. MyHbaseEventSerializer
agent. sinks. k. serializer. colNames = columns
```

2.4　测试与结果分析

测试系统使用的环境由四台机器组成的集群,其中安装 MySQL 数据库作为数据源的四台相同配置,HBase 集群中的 HBase Master 是一台服务器。测试中使用的 HBase 版本为 0.98,Flume 版本为 1.6.0。

为了最大限度地接近实际生产环境,使用的测试数据为 1 700 000 余条,总大小为 320 M,分别在每个数据库中存一份。三个 Flume 代理运行在服务器上,从三台机器的数据库中收集数据,测试结果见表 1。

表 1　测试数据

代理	时间/s	速度/(kb/s)	平均速度/(kb/s)	总速度/(kb/s)
1	316	1 012		
2	318	1 006	1 007	3 021
3	319	1 003		

由表 1 得到测试每个代理收集数据的平均时间为 318 s,平均速度为 1 007 kb/s,系统总速度为 3 021 kb/s,且不会丢失数据,在网络畅通的情况下数据可靠性强。后期研究发现在通过 JDBC 查询数据库时,对于大量数据,使用分页查询可以提高查询效率,并且解决了随着数据库数据量增大而引起的 Java 虚拟机内存溢出的问题。改变分页大小,测试系统得到测试结果如图 4 所示。

由图 4 可以看出,当分页为 10 000(这个值与具体的数据总量有关)时,传输的速度最快。此系统可在 5 min 左右将 510 000 000 多万条(大约 9 600 MB)的数据收集完毕,并且没有数据丢失,兼顾了数据传输的效率和可靠性,达到了预期的目标。

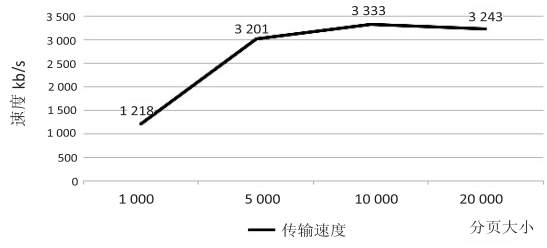

图 4　传输速度随分页大小变化的曲线

3　结束语

系统通过使用开源的工具 Flume-ng 1.6.0 实现了 MySQL 数据库数据收集系统,主要实现了其中的 Source 和 Sink 组件,完成了基于 JDBC 的自动查询、检测数据库更新和解析数据写入 HBase 的功能。丰富了 Flume 的功能,使得 Flume 可以在脱离 Hadoop 环境下进行数据传输。测试结果表明,系统可以高效可靠地收集数据库中的数据。

参考文献:

[1] 孙韩林. 一种基于云计算的网络流量分析系统结构[J]. 西安邮电大学学报,2013,18(4):75-79.

[2] 李　芬,朱志祥,刘盛辉. 大数据发展现状及面临的问题[J]. 西安邮电大学学报,2013,18(5):100-103.

[3] Apache Hadoop[EB/OL]. 2015. http://hadoop. apache. org/.

[4] 詹　玲,马　骏,陈伯江,等. 分布式 I/O 日志收集系统的设计与实现[J]. 计算机工程与应用,2010,46(36):88-90.

[5] Apache Flume[EB/OL]. 2015. http://flume. apache. org/.

[6] 王春梅. 基于数据仓库的数据挖掘技术[J]. 西安邮电学院学报,2006,11(5):99-102.

[7] 王　博,陈莉君. Hadoop 远程过程调用机制的分析和应用[J]. 西安邮电学院学报,2012,17(6):74-77.

[8] 孙　辉. MySQL 查询优化的研究和改进[D]. 武汉:华中科技大学,2007.

[9] 李现艳,赵书俊,初元萍. 基于 MySQL 的数据库服务器性能测试[J]. 核电子学与探测技术,2011,31(1):48-52.

[10] 谢晓燕,张静雯. 一种基于 Linux 集群技术的负载均衡方法[J]. 西安邮电大学学报,2014,19(3):64-68.

[11] Apache HBase[EB/OL]. 2015. http://hbase. apache. org/.

[12] 杨寒冰,赵　龙,贾金原. HBase 数据库迁移工具的设计与实现[J]. 计算机科学与探索,2013,7(3):236-246.

[13] Carstoiu D,Lepadatu E,Gaspar M. Hbase-non SQL database, performances evaluation[J]. International Journal of Advancements in Computing Technology,2010,2(5):42-52.