

基于 MapReduce 的 Hadoop 大表导入编程模型

陈吉荣*, 乐嘉锦

(东华大学 计算机科学与技术学院, 上海 201620)

(* 通信作者电子邮箱 chenjirongdh@163.com)

摘要:针对 Sqoop 在导入大表时表现出的不稳定和效率较低两个主要问题,设计并实现了一种新的基于 MapReduce 的大表导入编程模型。该模型对于大表的切分算法是:将大表总的记录数对 mapper 数求步长,获得对应每个 split 的 SQL 查询语句的起始行和区间长度(等于步长),从而保证每个 mapper 的导入工作量完全相同。该模型的 map 方式是:进入 map 函数的键值对中的键是一个 split 所对应的 SQL 语句,将查询放在 map 函数中完成,从而使模型中的每个 mapper 只调用一次 map 函数。对比实验表明:两个记录数相同的大表,无论其记录区间如何分布,其导入时间基本相同,或者对同一表分别用不同的分割字段,导入时间也完全相同;而对于同一个大表,模型的导入效率比 Sqoop 有显著提高。

关键词:编程模型; Hadoop; MapReduce; Hadoop 分布式文件系统; Sqoop

中图分类号:TP311.1;TP311.5 **文献标志码:**A

Programming model based on MapReduce for importing big table into HDFS

CHEN Jirong*, LE Jiajin

(Computer Science and Technology School, Donghua University, Shanghai 201620, China)

Abstract: To solve the problems of instability and inefficiency when data from a relation database system are transferred into Hadoop Distributed File System (HDFS) using Sqoop, the authors proposed and implemented a new programming model based on MapReduce framework. The algorithm splitting a big table in this model was as follows: firstly a step was calculated by dividing the total lines by the mapper number, then a SQL statement corresponding to each split could be constructed with a start line index and a span range equal to the above step, so this approach could guarantee that each mapper task would issue identical SQL workload. In map phrase, a mapper would only call map function once, with the single key-value pair below: the key was the above SQL statement corresponding to a split, and the value was null. The comparison experiments show that, for two different big tables with the same number of records, the respective importing time was approximately identical regardless of the records distribution, while using two different splitting fields in one big table, the importing time was also the same. At the same time, when applying two different approaches to one big table, the importing efficiency using the model was largely promoted than that using Sqoop.

Key words: programming model; Hadoop; MapReduce; Hadoop Distributed File System (HDFS); Sqoop

0 引言

目前,对于大数据的解决方案,有商业解决方案和 Hadoop 生态系统^[1]解决方案两种。商业解决方案具有性能高、集成度高和使用方便等特点,但是从目前 Hadoop 生态系统的成员来看,其多样性、灵活性、扩展性对中小企业具有很大的吸引力,已经成为他们面对大数据时的首选解决方案。大数据除了具有数据量足够大、深度足够深和广度足够宽三个主要特点^[2-3]外,其数据来源丰富也是其显著特征,特别是传统的关系型数据管理系统(Relational Database Management System, RDBMS)是其数据来源的主要渠道之一,主要原因是:关系型数据库中的大数据通常不能直接利用 MapReduce 模型来分析(注: HadoopDB^[4]可以直接利用 RDBMS 中的大表);历史遗留数据大部分还是存放在 RDBMS 中;Hadoop 生态系统和传统的 RDBMS 是一种共生和竞争的关系^[5,6]。所以,基于 Hadoop 生态系统的大数据分析平台需要有工具来将

RDBMS 中的大表高速的导入进 Hadoop 生态系统中,导入后的格式可为 Hadoop 分布式文件系统(Hadoop Distributed File System, HDFS)文件、Hive 表或 HBase 表。由于 Hive 表和 HBase 表的物理存储结构为 HDFS 文件,转换较为方便,所以一般只需要实现 HDFS 文件即可。虽然可以利用 Hadoop 提供的 DBInputFormat^[1]来导入 RDBMS 中的大表,但这是一种串行的方式,效率很低。目前, Sqoop^[7]提供并行导入功能的最常用工具(命令行的执行方式),但在实际的应用中,发现 Sqoop 作为导入大表的工具还存在性能问题和稳定性问题。性能问题指的是:当一个大表的行数达到 1 亿以上时,完成 job 的时间较长。稳定性问题指的是:同样行数的多个大表,其各自完成导入的时间差别很大;或者对同一大表用不同的分割字段。本文通过深入分析 Sqoop 导入原理,提出了一个新的基于 MapReduce 的编程模型,命名为大表导入器(Big Table Importer, BigTableImporter)编程模型,该模型能解决 Sqoop 目前存在的两个主要问题。

收稿日期:2013-03-16;修回日期:2013-04-20。 基金项目:国家核高基项目(2010ZX01042-001-003)。

作者简介:陈吉荣(1971-),男,安徽舒城人,讲师,博士,主要研究方向:Hadoop 生态系统的大数据平台;乐嘉锦(1951-),男,上海人,教授,博士生导师,主要研究方向:数据工程。

1 BigTableImporter 编程模型

1.1 编程模型的架构

BigTableImporter 的架构如图 1 所示。

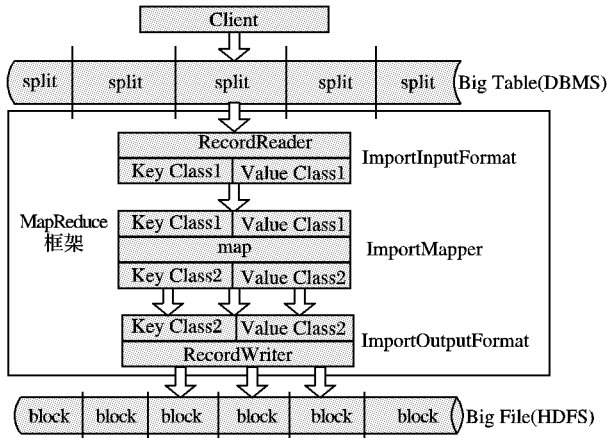


图1 BigTableImporter 的架构

在图 1 中,整个 BigTableImporter 编程模型的架构可以分为客户端(client)、数据源(source data)、导入方法(importing method)和目标数据(destination data)4 个部分(或者说 4 个层次)。

1) client 主要是给用户导入功能的接口,针对于大表的操作,常用的有 3 种方式:命令行、远程客户端的和插件。命令行方式如“BigTableImporter mysql://192.168.0.1:3306/hadoopguide--table student”形式;远程客户端类似于 SQLyog 这种数据库客户端,将导入操作加入到图形用户界面(Graphical User Interface, GUI)环境中;插件的方式类似于 MyEclipse 的 DBBrowser 数据库插件,也是将导入操作加入插件的 GUI 环境中。本文主要的工作是验证模型,采用了简单的命令行方式,在东华大学正在开发的大数据分析平台上,已经将该编程模型集成到远程客户端上(该远程客户端是基于 Squirrel SQL Client 开源项目的二次开发)。

2) source data 在模型中就是指传统的关系型数据库中的大表。

3) importing method 在模型中采用的是 MapReduce 框架。MapReduce 框架有三种形式,分别是:单一 Map 型(Map-Only) MapReduce、经典型(Classic) MapReduce 和迭代型(Iterative) MapReduce^[8]。Map-Only MapReduce 的特征是只有一次 Map 阶段,Classic MapReduce 的特征是分别有一次 Map 阶段和 Reduce 阶段,Iterative MapReduce 的特征是有多次的 Map 阶段和 Reduce 阶段。本文采用的是第一种形式。

4) destination data 在模型中指的是 HDFS 中的大文件。需要注意的是,导入 job 完成后不是产生一个大文件,而是可能产生多个大文件,大文件的个数和 mapper 的个数相同(Sqoop 工具甚至产生空文件)。当然,最理想的情况是产生一个大文件,但是如果采用的是 Map-Only MapReduce 且 mapper 的个数不是 1,一定会产生多个大文件;如果采用的是 Classic MapReduce 且 reducer 的个数是 1,只会产生一个大文件,但是这种情况就是一种“伪并行”,原因是:Map 阶段并行,Reduce 阶段串行,且 Reduce 阶段需要在 Map 阶段完成后才开始归并。这种出现多个大文件的情况对后续操作的影响有限,因为这些大文件都放在一个 HDFS 目录下,该目录的名称就是大表的表名,后续的操作可以用这个 HDFS 目录为输

入参数。

编程模型中的 ImportOutputFormat 类是 Hadoop 中的 FileOutputFormat 类的简单封装(为了编程模型中类的命名统一),所以下面重点分析如何设计和实现编程模型中的 ImportInputFormat 类和 ImportMapper 类,包括实现的算法和思路。

1.2 ImportInputFormat 类的实现

ImportInputFormat 类的结构如图 2 所示。

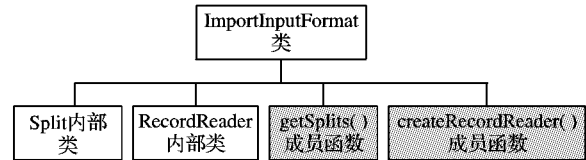


图2 ImportInputFormat 类的结构

Split 和 RecorderReader 是 ImportInputFormat 中的两个内部类, getSplits() 和 createRecordReader() 是 ImportInputFormat 中的两个成员函数。

一个 Split 对象逻辑上代表大表的一块区域。表示方法有很多,例如: <startRow, endRow>; <startRow, rowNumber>。

一个 RecorderReader 对象与一个 Split 相对应,主要提供了 createKey()、createValue() 和 nextKeyValue() 三个方法,需要完成的工作是将这个 Split 上的数据抽取并转换为 Map 需要的 <key, value> 键值对。

getSplits(): 返回多个 split,从而将一个表从逻辑上切分为多个 split,每个 split 按照“切分算法”包含大表中的相关记录。

createRecordReader(), 返回一个 RecordReader, 该 RecordReader 与一个 split 相对应。

1) getSplits() 和 Split 的具体实现。

对于分布式的编程模型^[9],最重要的是如何将任务尽量平均分解到各个子任务(各个子节点)中,MapReduce 编程模型也应当如此。如果任务分解不均匀,最终的执行时间应该是工作量最大的那个子任务的完成时间。Sqoop 是按照如下算法来分解导入任务:

假设表为 student,该表的主键为 id, mapper 的个数为 n;

step1 选定表的主键字段(如果没有主键,可以用其他字段),利用 SQL 的统计函数 min() 和 max(), 计算 id 字段在这个大表中的最小值和最大值;

step2 以 [min(id), max(id)] 整个连续区间作为 id 的取值范围, (max(id) - min(id)) / n 作为分隔这个连续区间的单位长度,计算每个 split 的上下边界(伪代码形式,注意最后一个 split 上下边界的设置,特别是下边界的设置):

```
int max = max(id);
int min = min(id);
int step = (max - min) / n;
int lowerbound = min;
int upperbound = lowerbound;
for( int i = 0; i < mapper - 1; i++) {
    split[i].lowerbound = lowerbound;
    upperbound = upperbound + step;
    split[i].upperbound = upperbound;
    lowerbound = upperbound;
}
split[mapper - 1].lowerbound = lowerbound;
split[mapper - 1].upperbound = max + 1;
/* 如果设置为 max(id), 在下面的 select 的查询结果中就会遗漏掉 id = max(id) 这条记录 */
```

对于每一个 split,其包含的大表区间记录可以这样获得:

```
select * from student where (id >= lowerbound) and (id < upperbound)
```

上面这种方式存在较大的问题。通过多个 mapper 的方式就是为了提高导入大表的速度,整个 job 的完成时间是那个最慢 mapper 的执行时间,产生某个 mapper 的执行时间最长的主要原因是:这个 mapper 承担了比其他 mapper 更多的导入数据量。按照上述分配 split 的算法,虽然每个 split 的长度完全相同(可能除了最后一个),但是一个 split 中的值在大表中不一定有对应的记录,甚至绝大多数都没有记录;或者一个 split 中的值在大表中对应多条记录(如果某个大表没有 prime key,那么用作切分的字段就不可能是 prime key)。这种分配 split 的算法是造成 Sqoop 不稳定的主要原因,当然这种情况对大表的导入效率也会产生重要影响。

针对 Sqoop 切分算法存在的问题,本文从大表的“行数”这个角度来设计算法,这种方式对于所有的表都能从逻辑上均匀切分。算法表示如下:

假设表为 student,用作切分的字段为 id(可以选择任意字段),mapper 的个数为 n;

step1 利用 SQL 的统计函数 count(),计算这个大表的行数;

step2 一个 split 用两个变量表示: startRow、rowNumber。startRow 表示这个 split 在大表中的开始位置,rowNumber 表示这个大表跨越的行数。伪代码形式如下(注意最后一个 split 的 rowNumber 设置):

```
int allLines = count(id);
int step = allLines/n;
int startRow = 0;
int rowNumber = step;
for( int i=0; i < n-1; i++) {
    split[i]. startRow = startRow;
    split[i]. rowNumber = step;
    startRow = startRow + step;
}
split[mapper-1]. startRow = startRow;
split[mapper-1]. rowNumber = count(id); /* 设置为大表的总
行数,可以避免漏掉一些记录,因为 step 是结果取整获得的 */
```

对于每一个 split,其包含的大表区间的记录可以这样获得:

```
select * from student limit startRow, rowNumber
```

2) createRecordReader() 和 RecorderReader 的具体实现。

在 Sqoop 中,当通过 createRecordReader() 创建一个 RecorderReader 对象后,实际上这个对象已获得了针对当前大表的一个 SQL 查询的所有信息: url; 所有字段名; 查询条件。url 是 java 程序通过 JDBC 连接数据库管理系统的必要条件,例如对于 MySQL 数据库,url 的常用形式是: mysql://192.168.0.1:3306/hadoopguide; 查询条件就是 split 的上下边界,例如(id >= lowerbound) and (id < upperbound)。

下面重点分析一下 RecorderReader 如何将查询范围内的每行转化成 Map 能处理的键值对,其算法可以表示如下:

step1 预处理,准备 SQL 查询需要的 url、字段名、查询条件。

step2 执行查询 executeQuery(),返回结果集 ResultSet。

step3 循环执行,判断结果集当前位置是否为空;不为空,则取出一行记录,转换为(LongWritable, SqoopRecord)键值对,其中 LongWritable 这个 key 为空,SqoopRecord 就是该行

记录对应的反序列化对象;该键值对作为参数被送进 map 函数处理;移动结果集的当前指针,判断下一行是否为空。用伪代码形式表示 step3 如下:

```
LongWritable key = currentRecordReader.createKey();
SqoopRecord value = currentRecordReader.createValue();
mapper.map(key, value, output, reporter);
while (currentRecordReader.nextKeyValue(key, value)) {
    LongWritable key = currentRecordReader.createKey();
    SqoopRecord value = currentRecordReader.createValue();
    mapper.map(key, value, output, reporter);
}
```

通过上面的分析可以发现得出这样的结论:如果一个 split 包含 1 百万行的记录,这个 split 对应的 mapper 就需要调用 1 百万次的 map 函数。如果能减少 map 函数的调用次数,同时不将数据库的每条记录反序列化为一个 java 对象,将会显著加快每个 split 的导入速度,最终整个作业的运行速度将会加快。

本文是用下面这个方法实现的:

一般情况下“一个 RecorderReader 对象将会产生多个键值对”,而这里采用的是“一个 RecorderReader 对象产生一个键值对”,键的值是一个字符串类型,格式为“url;sql”。例如:“mysql://192.168.0.1:3306/hadoopguide;select id, name, sex, birthday, address, content from student limit startRow, rowNumber”,url 为“mysql://192.168.0.1:3306/hadoopguide”,sql 为“select id, name, sex, birthday, address, content from student limit startRow, rowNumber”,两个串之间用分割符隔开。这种方式保证了一个 split 只产生一个 map 键值对,减少了频繁调用 map 函数造成的系统开销,包括 Java 虚拟机(Java Virtual Machine,JVM)垃圾回收器的负担^[10],既增加了导入的速度又提高了可靠性。其算法可以表示如下:

step1 预处理,准备 SQL 查询需要的 url、字段名、查询条件。

step2 并不执行 executeQuery(),而是直接设置 ResultSet 为 null。

利用了“ResultSet 为 null 能保证 step3 执行一次”的程序设计技巧。

```
step3
/** ** createKey() **
String myKey = new StringBuilder();
myKey.append(url).append(";").append(sql);
return new Text(myKey);
Text key = currentRecordReader.createKey();
/** ** createValue() **
return null;
BigTableImporterRecord value = currentRecordReader.createValue();
mapper.map(key, value, output, reporter);
/* 这段代码不会执行,因为在 step2 中的设置保证了 while 条件
第一次执行就为 false
while (currentRecordReader.nextKeyValue(key, value)) {
    LongWritable key = currentRecordReader.createKey();
    BigTableImporterRecord value = currentRecordReader.createKey();
    mapper.map(key, value, output, reporter);
}
*/
```

1.3 ImportMapper 类的实现

在 ImportMapper 类的重点是 map 成员函数,其原型如下:

```
public void map (Text key, BigTableImporterRecord val, Context
context) throws IOException, InterruptedException {}
```


基于上文的设计,每个 split 只有一个键值对进入 Mapper,也就是说只需要调用一次 map 成员函数。基本算法如下:

- 1) 将 key 分解为 url 串和 SQL 查询串(通过“;”分割符);
- 2) 将 SQL 查询串分解,获得字段个数 n;(通过“,”分割符);
- 3) 通过 JDBC 方式执行本次 SQL 查询,获得 ResultSet 结果集,该结果集包含符合本次查询条件的所有记录;
- 4) 循环 ResultSet,利用 n 合并相关字段,写入 Context。

```

StringBulder OneRecord = new StringBulder();
while ( results.next()){
    for(int i = 1; i < fieldCount; i++)
        OneRecord.append( results.getString(i) ).append( "," );
    OneRecord.append( results.getString( fieldCount ) ).append( "\n" );
    outkey.set( OneRecord.toString() );
    context.write( outkey, NullWritable.get() );
    OneRecord.delete( 0, OneRecord.length() );
}
//数据清空,可以再次使用

```

这段代码为了减少 JVM 垃圾回收器的负担,只构建了一个 StringBuilder 对象,用完后将数据清空,可以再次使用而不必构建新的对象,代码是: OneRecord.delete(0, OneRecord.length())。context 的 write 不需要优化,因为按照 MapReduce 框架,context 是向一个缓冲区写入键值对,当缓冲区满的时候,才自动写入本地文件系统。

2 对比实验

实验主要是将 BigTableImporter 与 Sqoop 的导入功能进行对比,主要是对比效率和稳定性两个方面。

2.1 建立实验环境

实验环境包括硬件环境、软件环境和数据环境,分为 2 个步骤:

Step1 通过 Apache Ambari 工具在实验室局域网环境下搭建一个 Hadoop 集群环境,10 台 PC,其中 1 台为 master 节点,其余 9 台为 slave 节点。安装了 Hadoop 生态系统中的 HDFS、MapReduce 和 Sqoop。注意,Sqoop 只是为了对比实验,BigTableImporter 并不需要 Sqoop 相关类的支持。

Step2 在另一台 PC 上安装 MySQL 数据库服务器及其客户端 SQLyog,利用 SQLyog 建立建立测试数据库 hadoopguide,并在 hadoopguide 中建立测试大表 student 并注入 1 亿行测试数据:

```

//create table hadoopguide. student
//除了 prime key 字段 id,其余字段均赋缺省值
create table student(
id          int          primary key,
name        varchar(30)  DEFAULT 'chenjirong',
sex         varchar(30)  DEFAULT 'man',
birthday    varchar(30)  DEFAULT '1971-11-09',
address     varchar(30)  DEFAULT 'dong hua university',
content     varchar(30)  DEFAULT 'this is a test for my paper'
)
//建立了 1 个 store procedure: myinsert( linesum int)
DELIMITER $$
DROP PROCEDURE IF EXISTS 'hadoopguide'. 'myinsert' $$
CREATE PROCEDURE 'hadoopguide'. 'myinsert' ( linesum int)
BEGIN
    declare count int;

```

```

set count = 1;
while count <= linesum do
    insert into student( id) values( count);
    set count = count + 1;
end while;
END $$
DELIMITER;
//注入 1 亿行数据
call myinsert( 100000000 );

```

2.2 对比测试

分别用 Sqoop 和 BigTableImporter 在相同的测试环境下对比测试,测试方法如下:

```

Sqoop mysql://192.168.0.1:3306/hadoopguide -- student -m 3
BigTableImporter mysql://192.168.0.1:3306/hadoopguide -- student
-m 3

```

再依次将 m 的值分别取为 4, 5, 6 进行测试,可以发现,基于本文编程模型开发的工具比 Sqoop 的导入性能有很大的提高,具体的性能对比如图 3 所示。

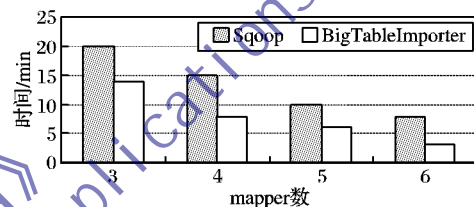


图 3 BigTableImporter 与 Sqoop 分别导入大表的性能对比

图 3 中:

- 1) 时间是实际时间(秒)取整后的数字(分钟)。
- 2) 当 mapper 数增加时,系统用于 mapper 的自身开销响应也会增加,所以并不能看到线性下降的结果。
- 3) 特别要申明的是,由于 mapper 个数和产生的 HDFS 文件个数是相等的,所以一般 mapper 的个数不能设置太大,因为这涉及到后续处理的问题,主要是 Hadoop 比较适合存储和处理大文件(一般将 mapper 的个数设置为 3)。

关于稳定性的测试,可以参照性能的测试,需要改变的是:建立一个没有 prime key 的表;写一个 store procedure,往表中注入大量连续相同的行。基于这样的大表即可进行稳定性的对比测试,限于篇幅,本文没有列出详细的实验步骤。

3 结语

基于 Hadoop 生态系统构建大数据分析平台是目前的一个应用趋势,也有一些典型案例,如 Yahoo、IBM 和淘宝等。但是生态系统中的一些管理成员的性能和功能还是有很大的提升空间,例如本文提到的大表导入工具 Sqoop。鉴于此,本文针对数据库大表的导入需求,设计并实现了一个新的导入编程 BigTableImporter。虽然二者均利用了 MapReduce 架构,但是在核心类的实现算法和设计思想上是完全不同的。本文的分析和实验均证明:BigTableImporter 的性能比 Sqoop 有很大的提高,且 BigTableImporter 已经完全解决了 Sqoop 不稳定的问题。

下一步需要完善的地方是:多个 mapper(无 reducer)在 HDFS 中产生多个文件(多个 reducer 也存在这种现象,这也是 MapReduce 用于高性能计算的障碍之一),由于这个问题是 MapReduce 框架固有和普遍的问题(Hadoop 自带的 WordCount 例子和其他基于 MapReduce 的一些应用均暴露出

(下转第 2561 页)

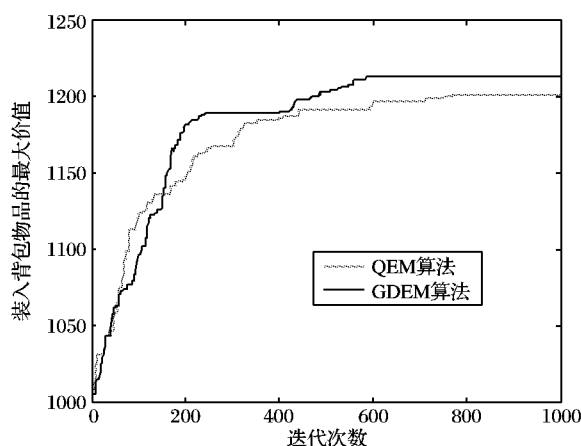


图4 两种算法求解实例3的过程(问题规模为250)

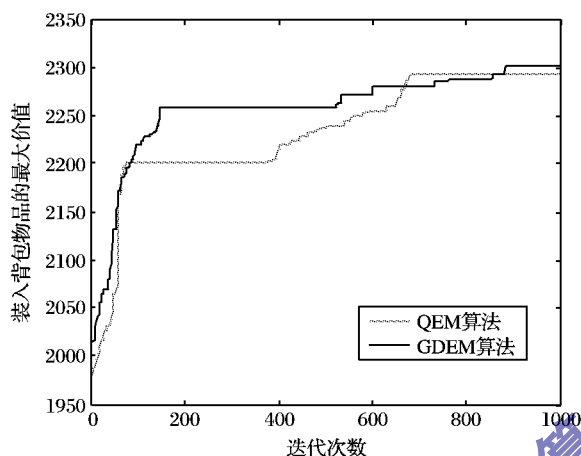


图5 两种算法求解实例3的过程(问题规模为500)

参考文献:

- [1] BIRBIL S I, FANG S C. An electromagnetism-like mechanism for global optimization[J]. *Journal of Global Optimization*, 2003, 25(3): 263-282.
- [2] BIEBIL S I. Stochastic global optimization techniques[D]. Raleigh: North Carolina State University, Department of Industrial Engineering, 2002.

- [3] 单玉乐, 曾建潮, 谭瑛. 一种改进的无局部搜索的类电磁机制算法[J]. *太原科技大学学报*, 2010, 31(6): 437-440.
- [4] 李如琦, 李芝荣, 凌武能, 等. 基于类电磁机制算法的配电网重构[J]. *电力系统保护与控制*, 2012, 40(14): 116-120.
- [5] 张智晟, 龚文杰, 段晓燕, 等. 类电磁机制算法在水电站厂内经济运行中的应用研究[J]. *电工电能新技术*, 2011, 30(4): 17-20.
- [6] 张科, 曹平. 复杂边坡非圆弧滑动面求解的类电磁机制算法[J]. *中南大学学报: 自然科学版*, 2011, 42(10): 3125-3130.
- [7] 孙禄, 张春江, 高亮, 等. 基于离散类电磁机制算法的装配序列规划[J]. *机械科学与技术*, 2012, 31(3): 353-358.
- [8] NIKBAKHS A, MOHSEN G A, REZA T. A discrete binary version of the electromagnetism-like heuristic for solving traveling salesman problem[C]// *Advanced Intelligent Computing Theories and Applications: with Aspects of Artificial Intelligence*. Berlin: Springer-Verlag, 2008: 123-130.
- [9] CHOU Y, CHANG C, CHIU C, et al. Classical and quantum-inspired electromagnetism-like mechanism for solving 0/1 knapsack problems[C]// *Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics*. Piscataway, NJ: IEEE Press, 2010: 3211-3218.
- [10] GAREY M R, JOHNSON D S. Computers and intractability: a guide to the theory of NP-completeness[M]. San Francisco: W. H. Freeman, 1979.
- [11] 吕晓峰, 张勇亮, 马羚. 一种求解0-1背包问题的改进遗传算法[J]. *计算机工程与应用*, 2011, 47(34): 44-46.
- [12] 何小锋, 马良. 求解0-1背包问题的量子蚁群算法[J]. *计算机工程与应用*, 2011, 47(16): 29-31.
- [13] 库向阳, 朱命昊, 赵亚敏. 求解0/1背包问题的改进人工鱼群算法研究[J]. *计算机工程与应用*, 2011, 47(21): 43-46.
- [14] 马丰宁, 谢龙, 郑重. 求解背包问题的基因属性保留遗传算法[J]. *天津大学学报*, 2010, 43(11): 1020-1024.
- [15] 汪定伟, 王俊伟, 王洪峰, 等. 智能优化方法[M]. 北京: 高等教育出版社, 2007.
- [16] 徐青鹤, 刘士荣, 吕强. 基于蚁群混沌行为的离散粒子群算法及其应用[J]. *计算机科学*, 2010, 37(5): 178-180.

(上接第2489页)

这个问题),所以要想解决这个问题,必须要修改Hadoop核心,解决思路是:每个mapper向带有编号的HDFS block写内容,最后通过“汇总”这些HDFS block编号,利用Hadoop元数据^[12]在逻辑上再形成一个HDFS大文件。

参考文献:

- [1] WHITE T. Hadoop: the definitive guide[M]. 影印版.3版.南京:东南大学出版社,2011.
- [2] ZIKOPOLOS P C, EATON C, DEROOS D, et al. Understanding big data: analytics for enterprise class hadoop and streaming data[M]. New York: McGraw-Hill, 2012.
- [3] HIDALGO C A. How to transform big data into knowledge[EB/OL]. [2013-02-10]. <http://enterprise.huawei.com/en/about/e-journal/ict/detail/hw-195167.htm>.
- [4] ABOUZEID A, BAJDA-PAWLIKOWSKI K, ABADI D J, et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads[C]// *Proceedings of the 35th International Conference on Very Large Data Bases*. New York: ACM Press, 2009: 733-743.
- [5] 覃雄派, 王会举, 杜小勇, 等. 大数据分析——RRDBMS与MapReduce的竞争与共生[J]. *软件学报*, 2012, 23(1): 32-45.
- [6] 王珊, 王会举, 覃雄派, 等. 架构大数据: 挑战、现状与展望[J]. *计算机学报*, 2011, 34(10): 1741-1752.
- [7] Apache Sqoop Project. Sqoop User Guide (V1.4.3)[EB/OL]. [2013-02-10]. <http://sqoop.apache.org/docs/1.4.3/SqoopUserGuide.html>.
- [8] HWANG K, FOX G, DONGARRA J. Distributed and cloud computing: from parallel processing to the Internet of things[M]. 北京: 机械工业出版社, 2012.
- [9] TANENBAUM A S, VAN-STEEN M. Distributed systems principles and paradigms[M]. 北京: 清华大学出版社, 2012.
- [10] 周志明. 深入理解Java虚拟机[M]. 北京: 机械工业出版社, 2011.
- [11] Apache Ambari Project. Ambari user guide[EB/OL]. [2013-02-10]. <http://incubator.apache.org/ambari/1.2.2/installing-hadoop-using-ambari/content/index.html>.
- [12] 文艾, 王磊. Hadoop分布式文件系统深度实践[M]. 北京: 清华大学出版社, 2012.