基于任务合并的并行大数据清洗过程优化

杨东华[1),2)] 李宁宁[1)] 王宏志[1)] 李建中[1)] 高 宏[1)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](哈尔滨工业大学基础与交叉科学研究院 哈尔滨 150001)

摘 要 数据质量问题会对大数据的应用产生致命影响,因此需要对存在数据质量问题的大数据进行清洗. MapReduce 编程框架可以利用并行技术实现高可扩展性的大数据清洗,然而,由于缺乏有效的设计,在基于 MapReduce 的数据清洗过程中存在计算的冗余,导致性能降低.因此文中的目的是对并行数据清洗过程进行优化从而提高效率.通过研究,作者发现数据清洗中一些任务往往都运行在同一输入文件上或者利用同样的运算结果,基于该发现文中提出了一种新的优化技术——基于任务合并的优化技术.针对冗余计算和利用同一输入文件的简单计算进行合并,通过这种合并可以减少 MapReduce 的轮数从而减少系统运行的时间,最终达到系统优化的目标.文中针对数据清洗过程中多个复杂的模块进行了优化,具体来说分别对实体识别模块、不一致数据修复模块和缺失值填充模块进行了优化.实验结果表明,文中提出的策略可以有效提高数据清洗的效率.

关键词 大数据;多任务优化;海量数据;数据清洗;Hadoop;MapReduce

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.00097

The Optimization of the Big Data Cleaning Based on Task Merging

YANG Dong-Hua[1),2)] LI Ning-Ning[1)] WANG Hong-Zhi[1)] LI Jian-Zhong[1)] GAO Hong[1)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*Academy of Fundamental and Interdisciplinary Sciences, Harbin Institute of Technology, Harbin* 150001)

Abstract Data quality issues will result in lethal effects of big data applications, so it is needed to clean the big data with the problem of data quality. MapReduce programming framework can take advantage of parallel technology to achieve high scalability for large data cleaning. However, due to the lack of effective design, redundant computation exists in the cleaning process based on MapReduce, resulting in decreased performance. Therefore, the purpose of this paper is to optimize the parallel data cleaning process to improve efficiency. Through research, we found that some data cleaning tasks are often run on the same input file or using the same calculation results. Based on the discovery this paper presents a new optimization techniques — optimization techniques based task combinations. By merging redundant computation and several simple calculations for the same input file, we can reduce the number of rounds of MapReduce system thereby reducing the running time, and ultimately achieve system optimization. In this paper, some complex modules of data cleaning process have been optimized, respectively entity recognition module, inconsistent data recovery module, and the module of missing values filling. The experimental results show that the proposed strategy in this paper can effectively improve the efficiency of data cleaning.

Keywords big data; multi-task optimization; massive data; data cleaning; Hadoop; MapReduce

收稿日期:2015-01-18;在线出版日期:2015-05-11.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316200)、国家自然科学基金(61472099,60933001,61272046)、国家"八六三"高技术研究发展计划项目基金(2012AA011004)、国家博士后基金(20090450126,201003447)、国家博士后基金特别资助项目(2013T60372)、教育部博士点基金(20102302120054)和黑龙江省自然科学基金(F201317)资助.**杨东华**,男,1976 年生,博士,副教授,主要研究方向为大数据管理,主要包括大数据查询处理与分析. E-mail: yang.dh@hit.edu.cn. **李宁宁**,男,1991 年生,硕士研究生,主要研究方向为数据质量等. **王宏志**(通信作者),男,1978 年生,博士,副教授,主要研究方向为 XML 数据管理、数据质量等. E-mail: wangzh@hit.edu.cn. **李建中**,男,1950 年生,博士,教授,主要研究领域为大数据、数据库、无线传感器网络. **高 宏**,女,1966 年生,博士,教授,主要研究领域为大数据、数据库、物联网.

1 引 言

本节主要介绍研究背景及其意义、海量数据清洗系统、本文优化方法的主要思想、本文的贡献与结构.

1.1 研究背景及其意义

现今企业的成功和社会的进步,越来越依赖于数据和对其所做的分析.为了获得竞争优势即使是小企业也会投入时间和精力来收集和分析数据.很多大公司都部署了自己的云服务平台,国内比较著名的有百度云、阿里云、天翼云①等.

但是如果一味地将精力投入到对数据所做的分析而不关注数据本身,很可能产生灾难性的后果.统计表明,美国企业中 1%～30%的数据存在各类错误和误差[1],医疗数据库中 13.6%～81%的关键数据不完整或陈旧[2].根据市场研究公司 Gartner 的调查,全球财富 1000 强公司超过 25%的关键数据不正确或不准确[3].数据质量问题会使基于其的分析和研究毫无意义甚至还会产生灾难性的后果,在美国由于数据错误引起的医疗事故每年使 98000 名患者丧生[4].上述实例表明数据质量问题存在于社会生活的方方面面,数据清洗系统应运而生.

在海量数据处理领域,MapReduce 编程框架作为当下最流行的并行编程开发框架已被 Google、Amazon、Yahoo!、Facebook 以及国内的腾讯、阿里巴巴等大型互联网公司奉为至宝.将 Hadoop 用于海量数据处理主要有如下优势:易用性、高可扩展性、高容错性.上述优势使得基于 MapReduce 的海量数据清洗顺其自然的产生了.

海量数据上的数据分析往往需要相对高昂的硬件成本和时间成本,这就引起了人们对优化数据分析的兴趣.当前已经有不少人开始研究大数据上的数据清洗系统,有对整个数据清洗系统进行研究[5-7],也有对其中的数据一致性[8-10],实体识别如文献[11-14]等问题进行研究的.然而现在还没有人对基于 MapReduce 的数据清洗系统的优化进行研究.现在几乎所有的数据分析任务都可以用 MapReduce 编程框架来实现,但是在实现过程中往往会存在冗余的 MapReduce,基于 MapReduce 的海量数据清洗系统也不例外.本文提出的基于任务合并的优化方法着眼于系统中冗余的 MapReduce,从细节和流程的各个方面实施.

1.2 海量数据清洗系统

海量数据清洗系统如图 1 所示,它在 Hadoop 平台上实施,以一个灵活的结构来处理不同类型的数据质量问题,每种类型的数据质量问题都由一个或多个模块来处理,由哈尔滨工业大学海量数据计算与研究中心提供源代码.系统中的交互模块提供一个输入接口来输入需要清洗的文件以及清洗数据的要求.结果展示模块提供清洁数据的下载链接以及脏数据和清洗后的数据的对比情况.实体识别和真值发现模块用于消冗,其中实体识别把指向同一现实世界实体的元组聚类,而真值发现用来在冲突中寻找出真实值.不一致检测模块发现数据中违反依赖规则的部分并且尝试把数据修复到符合规则的状态.数值填充部分检测数据缺失部分并填充.用户可以选择合适的模块来处理所遇到的数据质量问题.

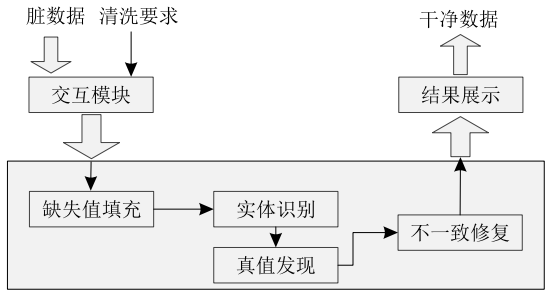

图 1 海量数据清洗系统功能模块结构图

1.3 本文优化方法概要和其他优化方法

本节首先给出基于任务合并的优化方法,然后对基于 Hadoop 平台的海量数据清洗系统进行优化.

一个实际的系统往往需要很多轮 MapReduce 来实现.有文献表明,对于比较复杂的单一型人物,拆分开来执行的话性能反而更好.但是根据 MapReduce 编程方式的特点,往往需要将一个问题分解成很多简单的任务,每一个任务由一轮 MapReduce 实现.在大多数情况下,这种"分解"是过度的,由此而产生冗余的 MapReduce.将可以合并的任务进行适当的合并,并且在不改变原系统的算法复杂度与迭代可终止性等前提下,满足可以减少原系统的 MapReduce 轮数和 IO 次数等条件进而达到优化的目的.

与本文研究方向相同的工作最杰出的优化方法

① Amazon EC2. http://aws.amazon.com/ec2/

有 MRShare[15-16]，后者是在前者的基础上发展而来的优化方法并且实现过程复杂，优化效率提升不是很明显. MRShare 把多个共享相同的 map 输入或输出的任务合并成一个任务，减少扫描文件的次数，从而达到优化的目的. 但当合并后的任务的较大的 map 输出的 sort 代价高于合并之前的多个独立的较小的 map 输出的代价时，就不会有任何优化效果.

本文提出的基于任务合并的优化技术针对冗余计算和利用同一输入文件的简单计算进行合并，通过这种合并可以减少 MapReduce 的轮数从而减少系统运行的时间. 通过对整个系统的框架与流程进行优化设计，有效地提高系统的效率.

1.4　本文的贡献

本文的主要贡献有：

(1) 提出一种基于 MapReduce 的应用系统的优化方法.

(2) 对海量数据清洗系统中计算较为复杂的 3 个模块进行讨论并提出优化方案.

(3) 对海量数据清洗系统的各个模块优化前后进行了大量的对比实验.

1.5　本文的结构

本文第 1 节介绍背景、主要内容和本文结构；第 2、3、4 节详细讨论优化方法与实施过程；第 5 节给出实验结果和分析；最后在第 6 节给出结论.

2　优化的缺失值填充

在实际的生产生活中，数据缺失是一种不可避免的现象，尤其是在数据收集工作日趋自动化的今天. 本模块是一种利用朴素贝叶斯分类的缺失值填充机制.

2.1　缺失值填充模块介绍

举一个例子，假设一个学校有 60% 的男生和 40% 的女生. 女生穿裤子的人数和穿裙子的人数相等，所有男生都穿裤子. 一个人在远处随机看到了一个穿裤子的学生，那么该学生的性别是什么？

如何解决这一问题呢？我们知道性别的取值只可能是男和女，那么可以根据公式 $P(C_i \mid X)=(P(X \mid C_i)\times P(C_i))/P(X)$ (C_i = 男、女，X = 穿裤子) 求出各个性别取值的概率，选取其中具有最大概率的性别取值就是我们给出的答案.

在这个例子中，未知的学生性别是缺失值，求解缺失值的过程中利用了朴素贝叶斯分类法. 下面通过图 2 和式(1)简要介绍本模块的实现.

图 2　缺失值填充模块流程图

$$P(C_i \mid X)=\frac{P(X \mid C_i)P(C_i)}{P(X)} \tag{1}$$

(1) 参数估计模块

整个缺失值填充系统是用贝叶斯分类的思想来计算出概率最大的取值作为填充值. 参数估计模块的任务是利用式(1)计算出所有的概率，其中 $P(X)$ 对所有取值为常数，所以只需要计算 $P(X \mid C_i)\times P(C_i)$ 即可. 在各个取值的先验概率未知的情况下，不妨假设其是等概率的，因此只需计算 $P(X \mid C_i)$ 即可. 对于具有多属性的数据集本文采用式(2)来计算 $P(X \mid C_i)$，

$$P(X \mid C_i)=P(x_1 \mid C_i)P(x_2 \mid C_i)\cdots P(x_n \mid C_i) \tag{2}$$

因此整个参数估计模块就是用来计算所有属性的每个取值的概率 $P(X \mid C_i)$. 概率论中认为，当样本空间足够大时概率≈频率，系统用统计不含缺失值的元组中各个属性取值出现的频率的方式来计算概率 $P(x_i \mid C_i)$.

(2) 连接模块

系统在填充模块会根据式(2)计算出含有缺失值的元组在它的依赖属性取值确定的所确定的各个待填充值的概率. 但是由于 MapReduce 函数在 map 阶段和 reduce 阶段一次只能处理一条记录，所以系统必须使依赖属性取值和其条件概率关联起来，这就是连接模块存在的必要性和需要解决的问题. 连接模块的输入数据为参数估计模块的输出数据和原待填充数据，输出数据是将含缺失数据的元组中依赖属性取值与该取值条件概率相关联的文件.

此模块的输入数据为原待填充数据和连接模块的输出，输出数据为经过填充之后的数据. 利用式(2)计算出每个 C_i(待填充属性可能的取值)对应的条件概率，选择其中 $P(C_i \mid X)$ 概率最大的 C_i 进行填充.

(3) 填充模块

填充模块由一轮 MapReduce 实现，首先将连接模块的输出结果和原始输入数据以偏移量为键值进行连接运算，map 阶段和连接模块类似，在此不再说明. reduce 阶段利用式(2)计算出每个 C_i(待填充属性可能的取值)对应的条件概率，选择其中 $P(C_i \mid X)$ 概率最大的 C_i 进行填充. 填充效果见图 3.

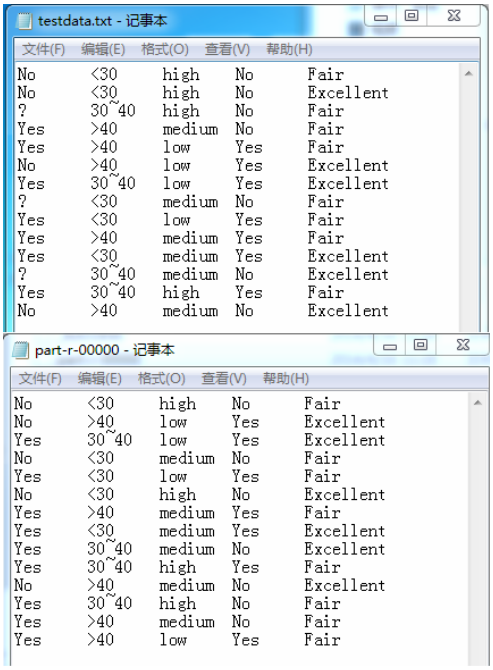

图 3 填充前后对比

以上是对缺失值填充模块的简要介绍，详细介绍参考文献[17]，本文仅对其离散类型的缺失值填充做考虑.

2.2 系统分析与优化

首先分析一下整个模块各个子模块之间的数据流和联系纽带. 参数估计子模块利用输入数据中的不包含缺失数据元组来计算以依赖属性的不同取值为条件的待填充属性的各种取值的条件概率①. 在计算填充值的过程中需要用到以各依赖属性的当前取值为条件的待填充属性的各种可能取值的条件概率②，②是①中的一组特定的值，②和①的联系纽带是依赖属性的取值. 而②和原始数据中的待填充元组之间的联系是该元组的偏移量. 因此在参数估计模块和填充模块之间增加了连接模块.

仔细观察系统各阶段数据流可以发现，在参数估计的 map 输入和 map 输出数据中均包含元组的偏移量，但是 reduce 输出数据中只有属性值和②. 这种情况使系统必须通过增加一个连接运算才能将②与待填充元组的偏移量结合在一起.

针对上述情况本文提出了一种将参数估计模块和连接模块的任务合并的优化方案，即在参数估计模块就将输出的条件概率与含有缺失值的元组偏移量关联起来. 其算法如下.

算法 1. 参数估计算法.

输入：含缺失值的数据文件、缺失值可能取值的文件

输出：条件概率

Map〈Object，Text，Text，Text〉

Input：*key*=offset，*value*=tuple

1. FOR each 〈*key*，*value*〉 DO
2. IF tuple contains missingvalue THEN
3. FOR each *value* in possiblevalue.txt DO
4. FOR each attribute in the tuple DO
5. *outkey* := possiblevalue，
   outvalue :=“#” offset+attribute ID+attribute

Reduce〈Text，Text，Text，Text〉

1. FOR each *value* in valuelist DO
2. IF *value* contain “#” THEN
3. append the “attribute ID+attribute” on likelihood
4. ELSE
5. calculate the conditional probability of each different “attribute ID+attribute”
6. IF “attribute ID+attribute” in the likelihood THEN
7. *outkey* := offset，*outvalue* := “#”+ the list of “possible value and the conditional probability”

其中 Tvalue 是每条元组在含有缺失值的列上的属性值，offset 是原记录在原文件中的偏移量，conditional probability 为当前属性值下的条件概率，possiblevalue.txt 是包含缺失值的可能取值的文本文件，attribute ID 为属性 ID 即属性的列号. 其 reduce 阶段流程可见于图 4.

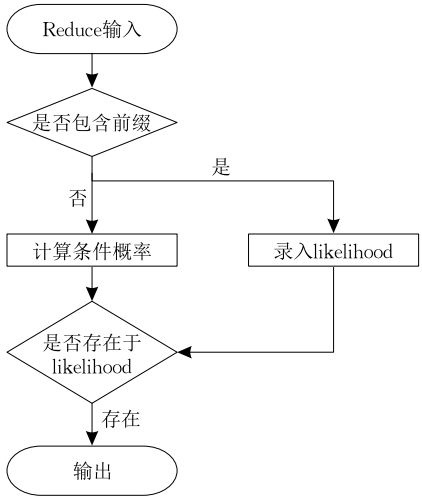

图 4 参数估计 Reduce 流程图

下面举例说明优化后的算法流程，表 1 为包含缺失值的数据，缺失值可能为 y 或 n. 前两条元组不含缺失值，故仅将其按属性拆分；第 3 条元组含有缺失值，我们在每种可能取值的情况下按属性拆分，Map 阶段输出结果见表 2. Reduce 阶段检查所有输入数据的前缀，若不包含缺失值则进入条件概率计算环节；若包含缺失值则将其录入 likelihood（用于判定条件概率是否需要输出）. 最后选择属性值存在于 likelihood 中的条件概率进行输出，输出结果见表 3.

表 1　含缺失值数据

Offset	A	B	C
0	*n*	*y*	*n*
120	*y*	*n*	*y*
178	?	*n*	*n*

表 2　参数估计 Map 输出

Key	*Value*
Tvalue/possiblevalue	Prefix+offset+attributeID+attribute
n	0,1,*y*
n	0,2,*n*
y	120,1,*n*
y	120,2,*y*
n	#178,1,*n*
y	#178,1,*n*
n	#178,2,*n*
y	#178,2,*n*

表 3　参数估计 Reduce 输出

Key	*Value*
#178	#y 0.5,n 0.5

优化后的参数估计子模块，Map 阶段的算法复杂度为 $O(n+ML)$，其中 n 为不包含缺失值的元组的数量，M 为包含缺失值的元组的数量，L 为缺失值可能取值的数目. 一般情况下 $M\leq n$，故 $O(n+ML)=O((1+L)n)$. 因为 L 为一个远小于 n 的常数，所以 Map 阶段的算法复杂度为 $O(n)$. Reduce 阶段的算法复杂度为 $O(n)$. 故整个参数估计子模块的算法复杂度为 $O(n)$.

优化前后，参数估计子模块的算法复杂度一直是 $O(n)$，填充子模块未做优化. 整个缺失值填充模块的 MapReduce 轮数和 IO 次数均由优化前的 3 变为 2，加速比为 3/2，优化效果明显.

3　优化的实体识别

实体识别，就是识别出同一实体的不同表现形式. 不同的数据来源对同一对象的表示形式往往有着不同的要求，并且在数据的存储和传递过程中均会产生不可避免的错误，因此产生了同一实体的不同表现形式. 关于 MapReduce 框架下的实体识别技术，现在已经有了相关研究工作[18-19]，但是他们只解决了异名实体识别问题，对同名问题没有进行研究；而我们的工作可以同时解决了异名和同名问题.

3.1　实体识别模块介绍

(1) 预处理. 系统读入海量数据文件并进行预处理，给每一条输入元组加上一个唯一的序号——实体 ID，方便后续处理.

(2) 初步聚类. 读取预处理模块生成的数据，按照相同属性值进行初步聚类，生成属性索引表.

(3) 实体识别. 对实体进行识别，对同一属性索引表中的实体对计算相似度并与阈值进行比较，大于阈值的相似对输出成相似对集合文件.

(4) 实体划分. 依据相似对集合文件生成图，通过对图的划分获得实体划分结果.

以上是对实体识别模块的简要介绍，详细介绍参考文献[20].

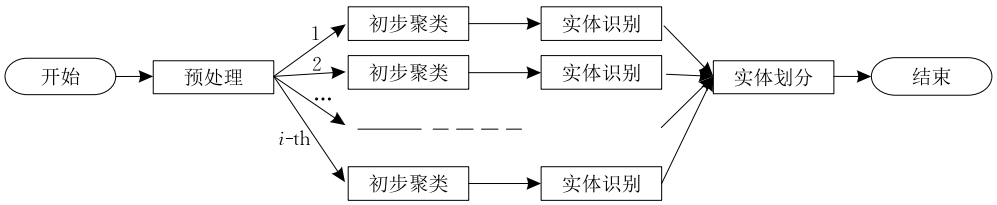

图 5　实体识别模块

3.2　系统分析与优化

通过研究发现初步聚类模块和实体识别模块，对预处理结果重复利用了 N 次（N 为待处理数据每条元组包含的属性个数），而且后续的实体识别模块也是在单一属性上处理的. 如果将预处理模块和实体识别模块看作一个整体（系统实际应用中也是这样的），那么就是对输入数据文件扫描多次，并且只能利用输入数据中的一部分，系统对输入数据的利用率很低. 此外系统每次分配任务都需要消耗额外的资源. 我们需要将分开处理各个属性的初步聚类和实体识别合并成一次能处理所有属性进而只运行一轮就能处理每条元组所有属性的解决方案.

为此本文针对实体识别模块提出的优化思想是：在初步聚类子模块一次处理所有属性，生成所有属性值的属性索引表. 这样就能将原来按属性分开处理的预处理和实体识别合并起来.

下面给出具体的优化方案和算法.

(1) 初步聚类子模块，map 阶段不是仅仅输出第 i 个属性值，而是将所有属性值都输出. 但是为了区分初步聚类产生的结果——属性索引表集合中的

实体 ID 是来自不同的属性，系统在 map 输出数据的 *key* 上做了一些改动，在原 *key* 前加上了一个前缀，由“属性值”变成“属性序号 $ 属性值”. 因为 MapReduce 是按照 *key* 进行分类的，所以只有同一属性的具有相同属性值的实体才会进入同一属性索引表. reduce 阶段将属性序号作为实体 ID 的前缀加在实体 ID 中. 以下是优化后的初步聚类算法.

算法 2. 初步聚类算法.

输入：关系表

输出：属性索引表

Map〈LongWritable，Text，Text，Text〉

Input：*key*=实体 ID，*value*=元组

1. FOR each 〈*key*，*value*〉 DO
2. FOR each 属性 DO
3. *outkey* := 属性 ID+“$”+属性值，
   outvalue := 实体 ID

Reduce〈Text，Text，NullWritable，Text〉

1. FOR each *value* in valuelist DO
2. 实体 ID := 属性 ID+“$”+实体 ID
3. *outkey* := null，*outvalue* := all the 实体 ID in valuelist

其中实体 ID 是每个元组的编号，在预处理阶段为每一个元组（一行数据）设定唯一的实体 ID. 属性 ID 是属性在元组中的顺序.

下面举例说明优化后的算法流程，待识别数据如表 4 所示. Map 阶段将所有元组的按属性拆分后输出，结果如表 5 所示. 属性值相同的实体会进入同一个 reduce，并输出成属性索引表，如表 6 所示.

表 4 实体识别数据

实体 ID	会议	年份	作者
1	ismar	2003	Michael Wagner
2	presence	2004	Michael Wagner
3	icde	2003	Ajay Gupta

表 5 初步聚类 map 输出

属性 ID+“$”+属性值	实体 ID
1$ismar	1
2$2003	1
3$ michael wagner	1
1$presence	2
2$2004	2
3$ michael wagner	2
1$icde	3
2$2003	3
3$ajay gupta	3

表 6 属性索引表

1$1
2$1，2$3
3$1，3$2
1$2
2$2
1$3
3$3

优化后的初步聚类模块，map 阶段的算法复杂度为 $O(n(x+x^2))$，其中 x 为属性个数，实际应用中 x 为一个很小的常数值，故其计算复杂度为 $O(n)$. reduce 阶段除了在附加的读取属性序号外没有任何改动，其计算复杂度为 $O(n)$. 综上整个初步聚类模块的算法复杂度为 $O(n)$.

(2) 实体识别子模块，因为整个实体识别模块是在初步聚类生成的属性索引表中进行的，而初步聚类模块的改动保证了同一属性的具有相同属性值的实体 ID 聚集在同一个属性索引表中，所以这一模块的算法不需要修改. 除了在第 3 个 MapReduce 的 reduce 阶段去除实体 ID 中包含的前缀外，没有任何更改. 这样做的目的是为了使同一实体的相似度能够在第 4 个 MapReduce 中汇合.

整个实体识别模块的算法没有经过改动，所以其算法的时间复杂度仍保持 $O(n)$.

由于本文只对系统 Hadoop 上运行部分进行优化处理，所以将实体划分模块视作常量. 在时间复杂度方面，从上一小节对实体识别子系统的介绍和本小节前面的一部分的优化方案中算法的计算复杂度可以看出，优化前后没有改变各个模块以及各个模块内部的各个 MapReduce 的计算复杂度. 优化前的 MapReduce 轮数为 $1+N(1+4)=5N+1$，优化之后的 MapReduce 轮数为 $1+1+4=6$，加速比为 $(5N+1)/6$. 正常情况下 N 大于 1，所以加速比大于 1，并且 N 越大加速效果越明显. IO 次数也由先前的 $5N+1$ 次变为 6 次，IO 次数减少使得系统用于 IO 的时间减少. 另外由于 MapReduce 的轮数减少，系统用于任务调度的时间和资源也相应减少.

综上，从理论上讲，通过本文的提供的优化方案能产生明显的优化效果.

4 优化的不一致数据修复

在实际的数据库系统及相关应用中由于种种原因，其中包含的数据违反最初定义的完整性约束，所以存在大量不一致数据. 本系统利用数据依赖理论中的条件函数依赖原理，定义完整性约束，利用完整性约束进行不一致数据修复. 本文的重点在于提高不一致数据修复模块的性能和效率，使之一致. 至于如何保证这样的修改过程是正确的，是由条件函数依赖理论所决定的，本文相关的理论证明和推导详见文献[9-10].

4.1　不一致数据修复模块介绍

不一致数据修复模块步骤的简要介绍如下：

(1)系统读入待修复的海量数据文件和 cfds 文件并进行预处理，将数据格式更改成符合系统要求的格式并对 cfds 进行初步检测，方便后续处理.

(2)对预处理结果中的数据文件进行检测与修复，得到初次修复结果.

(3)对初次修复结果进行检测，判断修复工作是否引入了新的不一致. 若引入了新的不一致则返回步骤(1)，否则进入步骤(3). 当然为了避免系统陷入死循环，系统为检测与修复的次数设置了一个上限.

(4)对修复结果进行后处理，将数据格式更改成数据的原始格式，使得修复结果能正常被其他系统使用.

以上是对实体识别模块的简要介绍，详细介绍见参考文献[21].

4.2　系统分析与优化

不一致数据修复子系统的 4 个模块中除第 1 个模块的CFD一致性检测子模块外都是在 MapReduce 编程框架上实现的，在本文的研究范围内.

本模块的一个重要缺点是没有掌握 MapReduce 编程“分解与合并”的精髓，将本来仅需要一个 Map 或者一轮 MapReduce 便可完成的任务拆分成一轮或多轮 MapReduce，由此使系统效率下降. 为此我们在不改变系统算法复杂度的条件下进行任务合并.

(1)预处理模块

预处理模块的脏数据预处理子模块功能很简单，就是给输入数据建立索引，实施过程中没有涉及到数据的分解与合并，可以通过一个 map 函数实现. 算法如下.

算法 3.　预处理算法.

输入：脏数据文件

输出：预处理结果

Map〈Object，Text，NullWritable，Text〉

Input：*key*=offset，*value*=tuple

1. FOR each 〈*key*，*value*〉 DO

2.　　*outkey* := null，*outvalue* := *key*+*value*；

显而易见，本模块的算法复杂度为 $O(n)$.

(2)不一致数据检测与修复模块

不一致数据的检测与修复模块中常量违反检测与修复模块通过一轮 MapReduce 实现，map 阶段将元组重新分发了 M 份(M 为输入元组发生常量违反的次数)，尽管 M 实际值一般不大，对 reduce 阶段的计算复杂度几乎没有影响. 但是 M 的存在会使中间数据量扩大 M 倍，对系统通信造成很大负担. 更重要的是在系统计算出建议修复值的同时就可以将其修复，那么就没有必要将找到建议修复值的过程和修复过程分开. 为此本文提出的优化方案是利用一个 map 函数实现常量违反检测与修复子模块.

将常量违反与修复子模块通过一个 map 函数实现之后，经过常量违反修复的数据直接进入变量违反修复环节. 两者输入数据的格式是相同的，假如原始数据中不存在常量违反，那么两者输入文件就是完全相同的. 基于上述观点，本文提出了将常量违反修复与变量违反修复合并的优化方案. 在这个优化方案中常量违反修复位于原变量违反修复的第 1 轮 MapReduce 的前端，让常量违反的结果在 Map 函数内部直接应用于变量违反. 算法如下.

算法 4.　不一致数据监测与修复第 1 轮 MapReduce 中 Map 算法.

输入：预处理结果

输出：变量违反第 1 轮 MapReduce 的 Map 输出

Input：*key*=offset，*value*=tuple

Map〈Object，Text，Text，Text〉

1. FOR each CFD $\alpha=(R:X\rightarrow Y,t_p)$ DO

2.　　IF tuple[X]$\asymp t_p[X]$ and tuple[Y]$\not\asymp t_p[Y]$ THEN

3.　　tuple[Y]$\asymp t_p[Y]$

4. FOR each CFD $\alpha=(R:X\rightarrow Y,t_p)$ DO

5.　　IF THEN

6.　　*outkey* := ($\mathrm{cfd}_{\mathrm{index}}$，$\mathrm{pt}_{\mathrm{index}}$，$\mathrm{attr}_{\mathrm{index}}$，1)，

　　outvalue := (offset，tuple)；

7. FOR each tuple that not match CFD with variables DO

8.　　*outkey* := ($\mathrm{cfd}_{\mathrm{index}}$，$\mathrm{pt}_{\mathrm{index}}$，$\mathrm{attr}_{\mathrm{index}}$，0)，

　　outvalue := (offset，tuple)；

其中，offset 为元组索引值，tuple 表示该条元组，fix_tag 为修复标志，用来区分是否发生违反需要修复，0 表示发生违反需要修复，1 表示不需要修复. $\mathrm{cfd}_{\mathrm{index}}$是 tuple 违反的 CFD 的序号，$\mathrm{pt}_{\mathrm{index}}$是该 tuple 违反的 CFD 的模式表中的模式元组序号，$\mathrm{attr}_{\mathrm{index}}$ 标志该 tuple 的不一致数据项的属性序号，fixvalue 即为该属性值应修复的结果.

算法 4 的计算复杂度为 $O(n)$.

算法的流程图如图 6 所示.

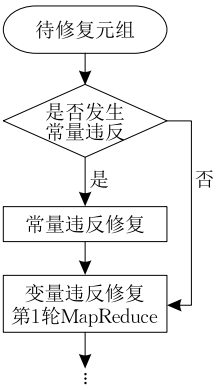

图 6 算法 4 流程

下面举例说明优化后的算法流程,待修复数据如表 7 所示. 为了便于说明,本例中仅使用 1 条 cfd 和一个 t_p,分别为 $\varnothing_1$:([CC,AC,PN]→[STR,CT,ST])和 T_1:01,908,_,_,MH,_. Map 阶段将每一条输入的待修复元组与模式元组 t_p 作比较,进行常量违反修复,然后再进行变量违反修复. 第 1 条元组没有发生常量违反,遂进入变量违反修复环节,其 map 输出为表 8 中第 1 行. 第 2 条元组发生常量违反,经常量违反修复进入变量违反修复环节,其 map 输出为表 8 中第 2 行. 下同.

表 7 待修复数据

offset	CC	AC	PN	NM	STR	CT	ZIP
109	01	908	11	Mike	Tree Ave	MH	07974
150	01	908	11	Rick	Tree Ave Str.	NYC	07974
190	01	908	11	David	Tree Ave	MH	03333
228	01	908	66	JoJo	Tree Ave	MH	07974

表 8 算法 3 Map 输出结果

key	*value*
0,0,2,1	109#01,908,11,Mike,Tree Ave,MH,07974
0,0,2,1	150#01,908,11,Rick,Tree Ave Str,MH,07974
0,0,2,1	190#01,908,11,David,Tree Ave,MH,03333
0,0,2,1	228#01,908,66,JoJo,Tree Ave,MH,07974

在时间复杂度方面,从预处理小节和本小节优化方案中算法的计算复杂度可以看出,优化前后没有改变各个模块以及各个模块内部的各个 MapReduce 的计算复杂度. 在 MapReduce 轮数和 IO 次数方面,系统的 MapReduce 轮数由优化前的 1+1+2+1+1+1=7 变成优化后的 1+2+1+1=5. 仅从 MapReduce 轮数来看系统的加速比为 7/5. 此外系统的优化工作还使得预处理模块的 MapReduce 变成了 map,这也会相应地减少系统的运行时间. 随着 MapReduce 轮数的减少,系统的 IO 次数也相应地减少,这也使得系统的 IO 负担减小.

综上所述,通过本文提供的优化方案,不一致修复子系统会获得理想的优化效果.

5 实验结果

整个系统在 Ubuntu 12.04.1 操作系统中的 Hadoop 1.2.1 平台上,用 java 语言实现,软件开发环境为 Eclipse. 实验运行的集群采用 12 个节点,1 个 tasktraker(namenode),11 个 jobtracker(datanode). 集群由 12 台机器组成,硬件环境为 Intel i7 3770 处理器,主频为 3.4GHz,内存 8GB,1TB 硬盘空间.

5.1 实体识别优化实验

为了使系统的优化效果更有说服力,所有实验数据是来自 DBLP 的真实数据集. 针对系统的特点,实验分别从扩展性、集群的并行化程度和数据的属性个数 3 个方面验证系统的优化效果.

DBLP 的数据规模并不大,看似不需要在 Hadoop 上实现. 但大家公认的数据源往往数据量比较小,使用 DBLP 数据集的意义不是因为其规模而是在于 DBLP 数据集是真实数据,这样做可以增加本文实验结果的可信度.

5.1.1 扩展性实验

本实验考虑数据集合大小对优化效果的影响,实验采用由真实数据集 DBLP 数据集,选择其中的 title, author and co-author, journal or url 这 3 个属性,选择大小分别为 13.2MB、32.3MB、64.9MB、97.1MB、128.9MB 的数据作为实验数据. 实验中各属性的权值分别为 0.9、0.05、0.05,实验结果如图 7 所示.

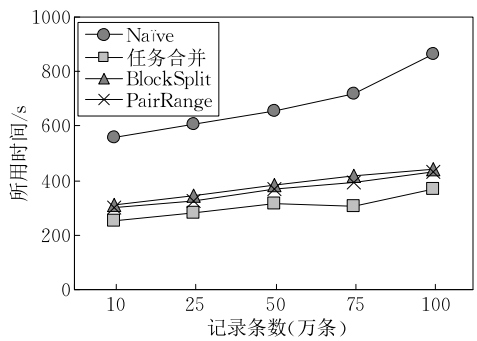

图 7 数据集合大小对优化结果的影响

实验中对比了基本的实现(Naïve)、文献[19]中的 BlockSplit 和 PairRange 与基于本文中任务合并的优化方法优化后的实体识别 4 种方法在不同数据集合大小下的运行结果.

随着数据集合的增大，优化前后系统的运行时间都在增加，但是优化前和优化后系统运行时间的比值(加速比)均在 2.3 左右. 这是因为本次实验所使用的数据具有 3 个属性，按照 2.2 节中对优化效果的分析，应具有的理论加速比为(5×3+1)/6=2.67 与实验结果一致. 由于基于 BlockSplit 和 PairRange 方法的实体识别实现过程的运行时间比基于任务合并的优化方法复杂，故其运行时间均比本文提出的优化方法对应的运行时间长. 本实验说明优化设计方案有良好的扩展性.

5.1.2　集群的并行化程度对优化效果的影响

实验考虑集群中 Reduce 个数对优化效果的影响，实验采用由真实数据集 DBLP 数据集，选择其中的 title, author and co-author, journal or url 这 3 个属性，选择大小分别为 128.9MB 含有 100000 条记录的数据作为实验数据. 实验中各属性的权值分别为 0.9、0.05、0.05，设置 Reduce 个数为 2、4、6、8、10，实验结果如图 8 所示，在不同的并行化程度下优化效果明显.

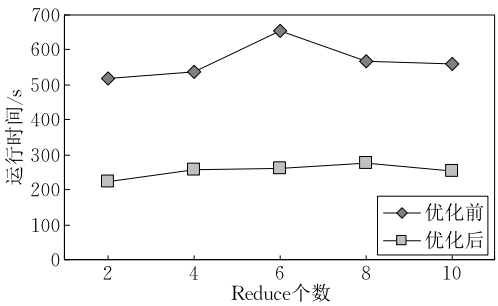

图 8　Reduce 个数对优化效果的影响

从图 8 中可以看出系统运行时间随并行化程度的增强而变长，这不符合大家普遍认为的“并行化程度越高，系统运行时间越短”的常识. 产生这一现象的原因是，实验数据集数据量较小，增加系统的并行度给系统带来的开销要大于由此带来的好处. 无论如何，系统在不同的并行度下达到了大约 2.3 的加速比，优化效果符合预期.

5.1.3　数据集的属性个数对优化效果的影响

针对优化方案的主要思想：充分利用输入记录中的所有属性，设计了本实验. 实验研究输入元组中的属性个数对优化效果的影响. 实验结果如图 9 所示，在处理同样大小(记录条数)的记录时，随着记录中包含的属性的增多，优化效果越来越好. 从上述数据可以看到：当处理的元组包含一条属性时，系统的优化效果最差，比优化前运行效率还要低；但是随着属性的增加优化效果越来越好. 本文的优化工作针对的是系统在处理多属性时不能充分利用输入数据，并且通过循环处理每一个属性增加了 MapReduce 轮数；但是处理单属性元组时优化后的方案产生的中间数据量多于优化前，并且处理过程变得更加复杂，因此产生了上述实验现象.

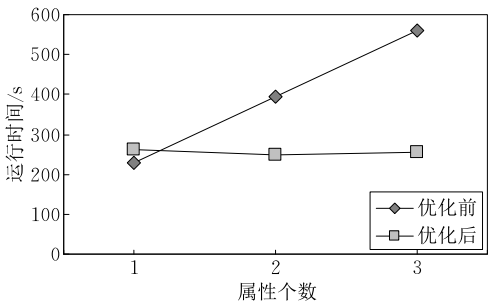

图 9　属性个数对优化结果的影响

5.2　不一致数据数据修复优化实验

为了验证系统在真实生产环境中的工作状态，实验采用来自真实数据集 Adult 的数据和由 TPC-H 生成的数据集. 进行了在 Adult 数据集上的加速比验证实验、在人工数据集上的扩展性和并行性验证实验.

5.2.1　加速比实验

实验采用条件函数依赖总共包含 2 条 cfd，共有 5 条模式元组. 根据 cfd 及其模式元组为来自 Adult 数据集中的无缺失值元组注入错误，使其违反一条或者多条约束. 实验条件和实验结果如表 9 所示，通过实验证明系统在真实数据集上的加速效果明显，符合优化方案设计预期. 此次实验加速比为 1.39，理论加速比大于 1.4，优化效果符合预期.

表 9　Adult 数据集上的加速比验证实验

项目	数据
数据源	Adult
数据记录条数	45000
cfd 数量	2
t_p 数量	5
reduce 个数	2
优化前运行时间	238
优化后运行时间	171
加速比	1.39

5.2.2　扩展性实验

为验证优化工作在不同大小的数据集上同样有明显的优化效果设计了本实验. 实验利用了由 TPC-H 生成的 lineitem.tbl 表中的 5 个属性生成的数据集，CFDs 由一条 cfd 包含 2 条 t_p 构成. 实验结果如图 9 所示，可见随着数据集合的增大优化效果会变好，说明优化设计方案有良好的扩展性.

从图 9 可以看出优化前系统运行时间随数据集增加呈线性增加,优化后系统运行时间随数据集的增加也呈线性增加,但前者的斜率更大.此外,系统优化前后的加速比从 1.59 一直上升到 2.20.一方面,优化前各个模块的计算任务相当,但是优化工作大大减轻了除不一致数据检测与修复模块之外各模块的负载;另一方面,实验中为了突出数据集的大小对系统运行时间的影响,仅设置 Reduce 个数为 2,故随着数据集的增大优化前的系统率先满负荷运行.从而出现了图 10 中对比鲜明的实验结果.

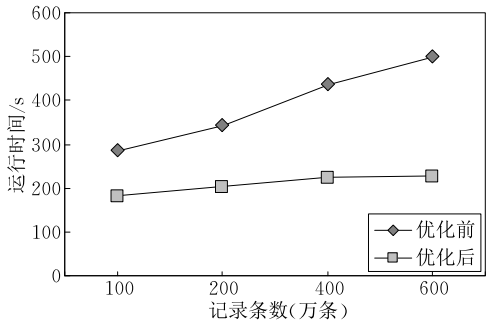

图 10 扩展性实验

5.2.3 并行性实验

为验证并行程度对优化效果的影响,设计了本实验.实验利用了由 TPC-H 生成的 lineitem.tbl 表中的 5 个属性生成的数据集,CFDs 由一条 cfd 包含 2 条 t_p 构成,详见图 11.

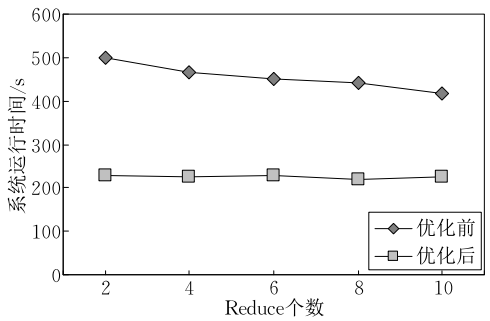

图 11 并行性验证实验

从图 10 可以看出系统在并行度较低(Reduce 个数为 2)的情况下加速比最高达到 2.20,之后随着系统并行化程度增高优化效果变差,加速比降低.并且优化前的系统随着系统的并行度的提高使得运行时间缩短,而优化后的系统基本保持不变.这是因为优化之前的系统处理数据的能力减弱,很容易满负载运行,只能通过增加系统的并行化程度提高数据处理的效率,故随着并行度增加系统运行时间减小;而优化后的系统吞吐量大,与处理和前者相同的数据一直都没有处于满负载状态运行,故增加系统的并行度带来的好处不明显.

5.3 缺失值填充优化实验

为了验证系统在真实生产环境中的工作状态,本文的实验采用来自真实数据集 Adult(www.archive.ics.uci.edu/ml/datasets.html)的数据和由 TPC-H 生成的数据集.进行了在 Adult 数据集上的缺失率对优化效果的验证实验、在人工数据集上的扩张性实验和并行性验证实验.

5.3.1 缺失率对优化效果的影响

实验主要研究不同的数据缺失率对优化结果的影响,通过将完整数据集按一定的比例(缺失率)随机置空数据生成实验所需的各种缺失率的数据.本文选取其中 9 个离散属性,缺失属性有 7 种取值,实验结果见图 12.在图中所示缺失率下,加速比稳定在 1.5 左右,与本模块的理论加速比 3/2 相吻合.

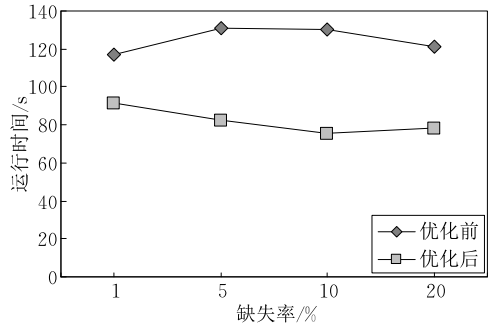

图 12 缺失率对优化结果的影响

5.3.2 扩展性验证实验

本实验利用由 TPC-H 生成的数据表 lineitem.tbl 选取其中 5 个属性,分别选择不同的记录条数生成实验所需的数据.图 13 所示实验结果表明,无论优化前后,系统的运行时间均随数据集的增大而增大,但是加速比均保持在 1.5 左右,与本模块的理论加速比相吻合.

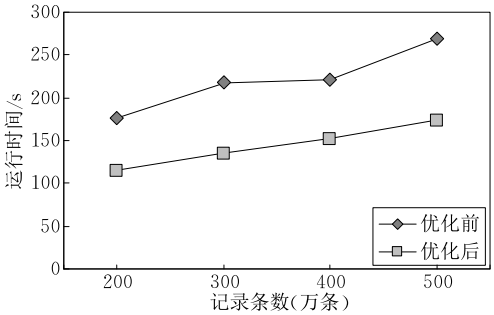

图 13 扩展性验证实验

5.3.3 并行性验证实验

为了验证系统在不同并行化程度下的优化效果,设计了本实验.实验利用由 TPC-H 生成的 lineitem.tbl

数据表，数据表共包含 5 个属性，3 000 000 条元组. 随机置空数据表第 1 列 5%的数据(缺失率 5%)，记录在不同的并行化程度下系统的优化效果. 实验结果见图 14.

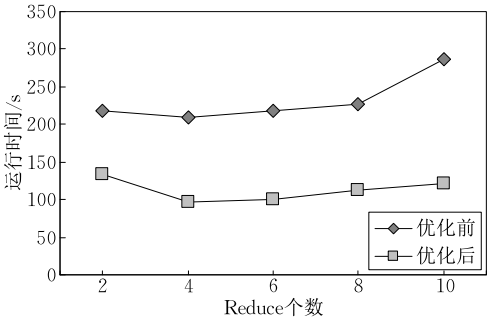

图 14　不同 Reduce 个数对优化结果的影响

在给定数据集上，系统优化前后的运行效率都未随着并行化程度的提高而变好. 这是因为对于特定大小的数据集最适宜的 Reduce 数目是确定的，一味地提高并行化程度只会给系统带来更多地任务分配的开销. 无论如何，在不同的并行化程度下优化效果明显.

6　结　论

虽然整个行业对 Hadoop 的研究和使用已经有了相当的积累，但是由于使用者对 MapReduce 编程框架理解不够深刻，所以利用 MapReduce 设计的软件系统大都效率低下. 为此本文提出了一种针对 MapReduce 编程框架设计的系统的优化方法，并通过了在海量数据清洗系统上的实施. 本文提出的优化方法仅需对原系统解决问题的思路稍作改动，几乎不影响其算法复杂性，通过减少 MapReduce 轮数和 IO 次数达到优化的目的. 优化方法简单，实用性强. 未来的工作包括将这种思想利用到更多基于 MapReduce 的系统中，对实验结果进行更为深入的分析以发现本文提供的优化方法的不足从而提出更好的优化方法.

参　考　文　献

[1] Redman T C. The impact of poor data quality on the typical enterprise. Communications of the ACM, 1998, 41(2): 79-82

[2] Miller Jr D W, Yeast J D, Evans R L. Missing prenatal records at a birth center: A communication problem quantified //Proceedings of the AMIA Annual Symposium. Washington, USA, 2005: 535-539

[3] Swartz N. Gartner warns firms of 'dirty data'. Information Management Journal, 2007, 41(3): 6

[4] Kohn L T, Corrigan J M, Donaldson M S. To Err Is Human: Building a Safer Health System. Washington, USA: National Academies Press, 2000

[5] Dallachiesa M, et al. NADEEF: A commodity data cleaning system//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 541-552

[6] Batini C, Cappiello C, Francalanci C, et al. Methodologies for data quality assessment and improvement. ACM Computing Surveys, 2009, 41(3): 16

[7] Hellerstein J M. Quantitative data cleaning for large databases. United Nations Economic Commission for Europe (UNECE), Geneva, Switzerland: Technical Report, 2008

[8] Beskales G, et al. On the relative trust between inconsistent data and inaccurate constraints//Proceedings of the International Conference on Data Engineering. Brisbane, Australia, 2013: 541-552

[9] Fan Wenfei, Li Jianzhong, Ma Shuai, et al. Interaction between record matching and data repairing. Journal of Data and Information Quality, 2014, 4(4): 16

[10] Fan Wenfei, Geerts F, Tang Nan, Yu Wenyuan. Inferring data currency and consistency for conflict resolution// Proceedings of the International Conference on Data Engineering. Brisbane, Australia, 2013: 470-481

[11] Shen W, Li Xin, Doan A. Constraint-based entity matching //Proceedings of the Association for the Advancement of Artificial Intelligence. Pittsburgh, USA, 2005: 862-867

[12] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning//Proceedings of the International Conference on Data Engineering. Atlanta, USA, 2006: 5-5

[13] Bayardo R J, Ma Yiming, Srikant R. Scaling up all pairs similarity search//Proceedings of the 16th International Conference on World Wide Web. Banff, Canada, 2007: 131-140

[14] Shen W, et al. Source-aware entity matching: A compositional approach//Proceedings of the International Conference on Data Engineering. Istanbul, Turkey, 2007: 196-205

[15] Nykiel T, et al. MRShare: sharing across multiple queries in MapReduce. Proceedings of the VLDB Endowment, 2010, 3(1-2): 494-505

[16] Wang Guoping, Chan Chee-Yong. Multi-query optimization in MapReduce framework. Proceedings of the VLDB Endowment, 2013, 7(3): 145-156

[17] Jin Lian, Wang Hong-Zhi, Huang Shen-Bin, Gao Hong. Missing value imputation in big data based on Map-Reduce. Journal of Computer Research and Development, 2013, 50(Supplement): 312-321(in Chinese)
(金连，王宏志，黄沈斌，高宏. 基于 Map-Reduce 的大数据缺失值填充算法. 计算机研究与发展，2013，50(增刊)：312-321)

[18] Kolb L, Thor A, Rahm E. Load balancing for MapReduce-

based entity resolution//Proceedings of the International Conference on Data Engineering. Washington, USA, 2012: 618-629

[19] Kolb L, Thor A, Rahm E. Don't match twice: Redundancy-free similarity computation with MapReduce//Proceedings of the 2nd Workshop on Data Analytics in the Cloud. New York, USA, 2013: 1-5

[20] Huo Ran, Wang Hong-Zhi, Zhu Rong, et al. Map-Reduce based entity identification in big data. Journal of Computer Research and Development, 2013, 50(S2): 170-179(in Chinese)
(霍然, 王宏志, 朱鎔等. 基于 Map-Reduce 的大数据实体识别算法. 计算机研究与发展, 2013, 50(S2): 170-179)

[21] Men Xue-Ying, Zhang An-Zhen, Wang Hong-Zhi, et al. Hadoop-based inconsistence detection and reparation algorithms for big data. Journal of Frontiers of Computer Science & Technology, 2014. doi: 10.3778/j.issn.1673-9418.1411043 (in Chinese)
(门雪莹, 张安珍, 王宏志等. 大数据上基于 Hadoop 的不一致数据检测与修复算法. 计算机科学与探索, 2014. doi: 10.3778/j.issn.1673-9418.1411043)

YANG Dong-Hua, born in 1976, Ph. D., associate professor. His research interests include massive data management, query processing and cloud computing.

LI Ning-Ning, born in 1991, M. S. candidate. His research interests include data quality and data management.

WANG Hong-Zhi, born in 1978, Ph. D., associate professor. His research interests include XML data management, data quality, etc.

LI Jian-Zhong, born in 1950, Ph. D., professor. His research interests include big data, database and wireless sensor network.

GAO Hong, born in 1966, Ph. D., professor. Her research interests include big data, database and Internet of Things.

Background

In recent years, with the growth of the information, dirty data such as erroneous, duplicate, uncertain or inconsistent exists in many database systems. Dirty data greatly reduces the quality of the data and brings serious losses to the enterprises and communities. Therefore, new techniques are in demand to process dirty data to reduce its harm.

Analysis on huge amounts of data often requires a relatively high hardware and time cost, which draws people's attention on the optimization of data analysis. Currently an increasing number of people began to study the data cleaning system on big data in the area of data cleaning system, consistency of data, entity recognition and so on. But, at present, no one has turned his interest on the optimization of the data cleaning system which based on the Map-Reduce framework. Now almost all of the data analysis tasks can use Map-Reduce programming framework to implement in which a lot of redundant Map-Reduce appear. There is no exception in implementing of data cleaning system. The optimization method in this paper based on merging tasks with redundancy Map-Reduce focus on the details and entirety.

In the direction of Map-Reduce program optimization, there has been someone did a good job. But most of these studies have not been applied to the production system for some reasons. We believe that it is not only worth the effort but a crucial endeavor. Our method is simple and easy to use.

This work is supported in part by the National Basic Research Program (973 Program) of China under Grant No. 2012CB316200, the National Natural Science Foundation of China under Grant Nos. 61472099, 60933001 and 61272046, the National High Technology Research and Development Program (863 Program) of China under Grant No. 2012AA011004, the Doctoral Fund of Ministry of Education of China under Grant No. 20102302120054, the China Postdoctoral Science Foundation Nos. 20090450126, 201003447, 2013T60372 and the Natural Science Foundation of Heilongjiang Province of China No. F201317.

Our group focuses on the research of big data and data quality for many years. Many papers have been published in conferences and transactions, such as SIGMOD, VLDB, ICDE, KDD, INFOCOM, TKDE and VLDB Journal. Our papers have been cited by other researchers over 3000 times.