分布式大数据多函数依赖冲突检测

李卫榜 李战怀 姜 涛
(西北工业大学计算机学院 西安 710072)

摘 要 关系数据库数据质量的一个主要问题是存在数据不一致现象.为找出不一致数据,需要进行函数依赖冲突检测.集中式数据库中可以通过 SQL 技术检测不一致情况,而分布式环境下的函数依赖冲突检测更富有挑战性,特别是大数据背景下,这个问题尤为突出.分布式环境下的函数依赖冲突检测通常需要进行数据迁移,而且不同的数据迁移方法会对检测效率产生一定的影响.该文提出了一种基于等价类的分布式环境多个函数依赖冲突检测的方法,给出了冲突检测的响应时间代价模型.由于分布式环境函数依赖冲突检测问题的任务分配问题为 NP-难问题,多项式时间内难以得到最优解,该文将不一致性检测响应时间最小化问题转化为整数规划问题,并给出了近似最优解.针对集群规模和函数依赖个数大小不同的情况,分别给出了不同的任务分配策略,并在检测过程中实现了动态负载均衡,有效提高了负载均衡度和检测效率.在真实和人工数据集上的实验表明,相对于集中式检测方法以及基于 Hadoop 的 naïve 方法,该文提出的多函数依赖冲突检测方法检测效率有明显的提升,且在数据规模、节点个数和函数依赖个数等方面扩展性能良好.

关键词 函数依赖;冲突检测;不一致性;分布式数据;大数据
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.00144

Violations Detection of Multiple Functional Dependencies in Distributed Big Data

LI Wei-Bang LI Zhan-Huai JIANG Tao
(*College of Computer Science, Northwestern Polytechnical University, Xi'an* 710072)

Abstract One major problem of data quality in relational database is data inconsistency. To find out the inconsistent data in the relational database, we need to detect the functional dependency violations. It is easy to detect dependency violations in centralized databases via SQL-based techniques. However, it is far more challenging to check dependency violations in distributed databases, especially with big data. It is usually necessary to ship data from one site to another when detecting functional dependency violations from distributed data. Moreover, different data migration methods may have different impact on the detection efficiency. This paper proposes a novel equivalence class based multiple functional dependency violations detection approach in distributed big data, and provides a cost model of violations detection. Considering that the inconsistency detection problem is NP-hard, it is impossible to find an optimal solution in polynomial time, so we transform the problem of minimizing response time of inconsistency detection into an integer programming problem and provide an optimal solution for the allocation of detecting tasks. Against difference of cluster size and the number of functional dependencies, we propose different tasks allocation strategies, and achieve dynamic load balancing in the detection process, which can improve the detection efficiency and load balancing degree effectively. Experiments on

收稿日期:2015-11-24;在线出版日期:2016-05-17.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316203)、国家自然科学基金(61502390,61472321,61332006,61272121)、国家"八六三"高技术研究发展计划项目基金(2015AA015307)资助.**李卫榜**,男,1979 年生,博士研究生,主要研究方向为数据质量、大数据、海量数据计算. E-mail: wbli2003@163.com. **李战怀**,男,1961 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为数据库理论和技术、数据管理. **姜 涛**(通信作者),男,1983 年生,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为生物信息挖掘、数据管理. E-mail: jiangtao@mail.nwpu.edu.cn.

real-world and generated datasets demonstrate that compared with previous detection methods and naïve method based on Hadoop platform, our approach is more effective in efficiency and with good scalability on the number of nodes, on the size of datasets and on the number of functional dependencies.

Keywords　functional dependency; violations detection; inconsistency; distributed data; big data

1 引　言

数据管理最重要的问题之一是数据质量问题[1]，其中不一致性检测是数据质量的一个重要内容. 对于完整性约束函数依赖来说，由于数据管理不规范等问题，违反函数依赖的情况十分普遍，而且在数据融合等场合，也经常会遇到违反函数依赖的情况. 数据特别是重要数据对其质量有着极高的要求，违反函数依赖的数据如果不进行处理，可能会付出很大的代价.

为提高数据质量，通常需要对数据进行不一致性检测. 不一致性检测对于集中式数据来说较为容易，如函数依赖冲突检测可以使用一种基于 SQL 技术的检测方法[2]. 然而现实中的数据并不都是集中式分布的，可能被切分并分布在不同的机器上[3].

例 1.　考虑如下一个关系表 EMP(ID, ENO, ENAME, TITLE, SAL, PNO, PNAME, RESP, DUR)，每一个员工元组包含了 ID、工号、姓名、头衔、薪水、项目编号、项目名称、职责、持续时长等内容. 这里 ID 是关系表 EMP 的主键，EMP 的一个实例 D_0 如表 1 所示.

表 1　EMP 的一个实例 D_0

ID	ENO	ENAME	TITLE	SAL	PNO	PNAME	RESP	DUR
1	E1	M. Smith	Syst. Anal.	3500	P1	Database Develop	Manager	36
2	E1	M. Smith	Elec. Eng.	2600	P2	CAD/CAM	Engineer	24
3	E2	J. Jones	Programmer	2500	P2	CAD/CAM	Programmer	12
4	E2	J. Jones	Analyst	3300	P3	Instrumentation	Analyst	18
5	E2	J. Davis	Mech. Eng.	2700	P1	Database Develop	Engineer	24
6	E3	F. Lee	Programmer	2800	P3	Instrumentation	Programmer	18
7	E3	F. Lee	Analyst	3800	P2	CAD/CAM	Manager	12
8	E4	F. Lee	Mech. Eng.	2500	P1	Database Develop	Engineer	36
9	E5	B. Casey	Analyst	3300	P1	Database Develop	Manager	24
10	E5	D. Casey	Programmer	2800	P3	Instrumentation	Programmer	12

为了检测不一致性，在关系表 EMP 上定义了如下几个函数依赖(Functional Dependency, FD)作为数据质量规则：

φ_1: ENO→ENAME

φ_2: PNO→PNAME

φ_3: TITLE→SAL

φ_4: TITLE→RESP

φ_5: ENO, PNO→DUR

这里 φ_1 声明了工号唯一决定员工姓名，φ_2 声明了项目编号唯一决定项目名称，φ_3 表示头衔唯一确定薪水，φ_4 表示头衔唯一确定职责，φ_5 表示工号和项目编号联合确定持续时长.

为找出表 1 中存在的不一致数据，需要找出其中违反函数依赖集合 $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$ 的元组. D_0 中 id 为 i 的元组用 t_i 表示，则在 D_0 中有如下元组存在冲突：$t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}$. 其中元组 t_3, t_4 和 t_5 违反函数依赖 φ_1：元组 t_3, t_4 和 t_5 工号相同，员工姓名不同. 同理，元组 t_9 和 t_{10} 违反函数依赖 φ_1. 元组 t_3 和 t_6 违反函数依赖 φ_3：元组 t_3 和 t_6 头衔相同，薪水不同. 同理，元组 t_4 和 t_7 违反函数依赖 φ_3，元组 t_5 和 t_8 违反函数依赖 φ_3；元组 t_4, t_7 和 t_9 违反函数依赖 φ_4.

在集中式环境，数据分布在一个机器上，在进行函数依赖冲突检测时，可以使用已有的基于 SQL 的方法进行检测[1]. 在数据规模较大或者待检测函数依赖个数较多的情况下，传统的集中式函数依赖冲突检测方法还存在检测效率低下的问题.

如果数据不是集中分布，而是分布在不同的节点，则传统的检测方法无法直接使用. 一种 naïve 的方法是将所有待检测数据迁移到一个节点，在该节点使用集中式方法进行检测. 这种方法存在如下几个方面的问题：一是需要数据的迁移，在数据规模很

大的情况下,数据迁移耗时也比较可观;二是传统的方法每检测一个函数依赖,都需要将所有数据遍历一次,在函数依赖个数比较多的情况下,效率比较低下;三是所有的检测任务都在一个节点进行,负载严重的不均衡,效率较低.

以表 1 中的数据实例 D_0 为例,其数据被水平切分成 3 个部分,分别如表 2、表 3 和表 4 所示,其中分片 D_{H1}、D_{H2} 和 D_{H3} 分别分布在站点 S_1、S_2 和 S_3 上.为检测违反函数依赖 φ_1 的元组,需要进行数据迁移,数据迁移的原则是函数依赖左端属性值相同的元组迁移到同一个节点.迁移的方案有多种:(1)从站点 S_2 迁移元组 t_7 到 S_1,从站点 S_3 迁移元组 t_3 到 S_1,从站点 S_1 迁移元组 t_9 到 S_3;(2)从站点 S_1 迁移元组 t_5 到 S_2,从站点 S_3 迁移元组 t_4 到 S_2,从站点 S_1 迁移元组 t_9 到 S_3;(3)从站点 S_1 迁移元组 t_5 到 S_3,从站点 S_2 迁移元组 t_7 到 S_3,从站点 S_3 迁移元组 t_{10} 到 S_1;等等.从这个例子可以看出,与集中式环境不同,在分布式环境下,不一致数据冲突检测通常需要数据的迁移,因此集中式环境下的检测方法不适用于分布式环境下的不一致性检测.针对分布式环境大数据背景下多个函数依赖冲突检测的问题,本文提出了算法 $\text{MultiFDsDet}_{\text{DS}}$,基于 Hadoop 和 Hama 平台,实现分布式环境大数据多个函数依赖的并行冲突检测,有效提高检测的效率.

表 2 D_0 的一个水平划分的片段 D_{H1}

ID	ENO	ENAME	TITLE	SAL	PNO	PNAME	RESP	DUR
1	E1	M. Smith	Syst. Anal.	3500	P1	Database Develop	Manager	36
5	E2	J. Davis	Mech. Eng.	2700	P1	Database Develop	Engineer	24
8	E4	F. Lee	Mech. Eng.	2500	P1	Database Develop	Engineer	36
9	E5	B. Casey	Analyst	3300	P1	Database Develop	Manager	24

表 3 D_0 的一个水平划分的片段 D_{H2}

ID	ENO	ENAME	TITLE	SAL	PNO	PNAME	RESP	DUR
2	E1	M. Smith	Elec. Eng.	2600	P2	CAD/CAM	Engineer	24
3	E2	J. Jones	Programmer	2500	P2	CAD/CAM	Programmer	12
7	E3	F. Lee	Analyst	3800	P2	CAD/CAM	Manager	12

表 4 D_0 的一个水平划分的片段 D_{H3}

ID	ENO	ENAME	TITLE	SAL	PNO	PNAME	RESP	DUR
4	E2	J. Jones	Analyst	3300	P3	Instrumentation	Analyst	18
6	E3	F. Lee	Programmer	2800	P3	Instrumentation	Programmer	18
10	E5	D. Casey	Programmer	2800	P3	Instrumentation	Programmer	12

本文的主要贡献如下:

(1)提出了分布式环境下多个函数依赖不一致性检测响应时间代价模型.由于分布式环境不一致性检测最小化响应时间问题为 NP-难问题,无法在多项式时间得到最优解,本文基于该模型给出了近似最优并行检测方法.

(2)将等价类引入到函数依赖的冲突检测,给出了基于等价类的冲突检测的优化策略,有效提高检测效率.

(3)为避免检测过程中负载不均衡的情况,在检测过程中动态均衡负载,为减少负载迁移量,将该问题划归为二次规划问题,并采用拉格朗日算子法得到最优解.

(4)对本文提出的方法基于真实和人工数据集进行了对比实验验证.实验结果表明,本文提出的方法在数据扩展性、集群节点扩展性以及函数依赖规模扩展性方面均表现良好.

2 相关工作

函数依赖作为关系数据库中的一种完整性约束,反映了不同属性之间的一种制约关系,其概念最早由 Armstrong 提出[1].函数依赖冲突在现实世界中十分常见,特别是在 Web 数据抽取、数据融合等场合更为常见.当前针对分布式环境违反完整性约束进行检测的相关文献还比较少,Fan 等人针对条件函数依赖(CFD)在分布式数据上的不一致性问题进行了研究[4],给出了几种不一致性检测算法,并利用条件函数依赖的结构特点减少数据迁移或响应时间.条件函数依赖是对传统函数依赖的一种扩展[5],对函数依赖的左部(LHS)加以条件限制,然后在一定条件的基础上使用传统的函数依赖.然而条件函

数依赖和传统的函数依赖有着很大的不同，没有条件函数依赖的模式表(pattern table)，因此无法使用针对 CFD 的分布式数据不一致性检测方法检测传统函数依赖的冲突. Fan 还提出一种分布式环境 CFD 不一致性增量检测方法，针对数据规模动态变化的情况，检测 CFD 的不一致性[6].

文献[7-8]对分布式数据库中完整性约束的检测问题进行了相关的研究. 文献[7]研究了在仅访问局部关系的情况下对基本关系更新时全局不一致性检测的问题. 文献[9]中给出了完整性约束检测的本地条件，认为如果本地数据满足这些条件，则全局数据也同样满足，因此无需进行数据的迁移，这样可以有效减少通信代价. 文献[10]对数据融合中数据冲突的解析展开了相关研究，主要针对的是集中式环境下的冲突问题.

文献[11-12]对分布式数据源的异常检测问题进行了研究. 文献[11]提出了一种针对包含混合属性数据集的快速分布式异常检测策略. 这些方法主要针对的是异常检测，主要目标是从给定的数据源中找到异常情况. 与之不同，函数依赖冲突检测主要是检测函数依赖约束的违反情况. 文献[13-14]对函数依赖冲突的修复进行了相关研究. 为了对存在违反函数依赖情况的不一致数据进行修复，通常采用对属性值进行修改的方法. 与本文工作不同，这些文献主要关注的是函数依赖冲突的修复问题. 文献[15-16]研究了函数依赖发现的问题，主要是研究给定一个模式，如何发现隐含的约束关系.

3　预备知识

3.1　函数依赖

定义 1.　给定关系 R，$\mathrm{Attrs}(R)$ 为 R 上所有属性的集合，$X, Y \subseteq \mathrm{Attrs}(R)$. 函数依赖(Functional Dependency，FD)是定义在 R 上的一个约束，用 $X \rightarrow Y$ 表示. 函数依赖 $X \rightarrow Y$ 成立当且仅当对 $\forall t_i, t_j \in R$，如果 $t_i[X]=t_j[X]$，则必然有 $t_i[Y]=t_j[Y]$.

对于函数依赖 $X \rightarrow Y$，我们称 X 为其左部(Left Hand Side，LHS)，相应的 Y 为其右部(Right Hand Side，RHS). 如果函数依赖的 LHS 与 RHS 交集为空，称该函数依赖为非平凡函数依赖. 本文中仅考虑非平凡多个函数依赖的冲突检测问题.

3.2　等价类

定义 2.　等价类(Equivalence Class，EC). 假定 $X \subset \mathrm{Attr}(R)$ 为关系 R 上的一个属性集合，其中 $\mathrm{Attr}(R)$ 为关系 R 的所有属性集合，我们称元组 $t \in R$ 在属性集合 X 上的等价类为 $[t]_X=\{p \in R \mid$ 对所有 $A \in X$，有 $p[A]=t[A]\}$，这里 $t[A]$ 为元组 t 在属性 A 上的值.

例 2.　以表 1 中 EMP 的一个实例 D_0 为例，属性 ENO 在元组 1 和 2 上有着相同的属性值 E1，因此元组 1 和 2 构成等价类{1,2}，这里我们用元组的 ID 值代表元组. 同理，元组 3、4 和 5 在 ENO 上有着相同的属性值 E2，因此构成等价类{3,4,5}.

定义 3.　局部等价类(Local Equivalence Class，LEC). 假定 $X \subset \mathrm{Attr}(R)$ 为关系 R 上的一个属性集合，其中 $\mathrm{Attr}(R)$ 为关系 R 的所有属性集合，关系 R 的实例 D 被横向切分为 n 块，$D=D_1 \cup \cdots \cup D_n$，我们称元组 $t \in D_i$ 在属性集合 X 上的等价类为局部等价类，表示为 $[t]_{Xi}=\{p \in D_i \mid$ 对所有 $A \in X$，有 $p[A]=t[A]\}$，这里 $t[A]$ 为元组 t 在属性 A 上的值.

定义 4.　划分(Partition). 假定 $X \subset \mathrm{Attr}(R)$ 为关系 R 上的一个属性集合，定义 $\Pi_X=\{[t]_X \mid t \in R\}$ 为关系 R 在属性集合 X 上的一个划分.

根据等价类和划分的定义可以看出，划分 Π_X 由关系 R 在属性集合 X 上不相交的等价类组成，每一个等价类内的元组在属性集合 X 上有着相同的属性值，不同的等价类在 X 上的属性值不同，而等价类的并集与关系 R 等价.

例 3.　以表 1 中 EMP 的一个实例 D_0 为例，关系 EMP 在属性 ENO 上的划分为{{1,2},{3,4,5},{6,7},{8},{9,10}}，关系 EMP 在属性集合{ENO, ENAME}的划分为{{1,2},{3,4},{5},{6,7},{8},{9},{10}}.

引理 1.　给定关系 R，$\varphi: X \rightarrow Y$ 为关系 R 上的函数依赖，Π_X 为关系 R 在属性集合 X 上的一个划分，则有 Π_X 中所有等价类包含的元组个数之和等于关系 R 包含的总的元组个数，即 $\Sigma|[t]_X|=|R|$，其中 $[t]_X \in \Pi_X$.

证明.　假设 $\Sigma|[t]_X| \neq |R|$，则有 $\Sigma|[t]_X|>|R|$ 或者 $\Sigma|[t]_X|<|R|$. 根据定义 2，$[t]_X$ 中元组均为 R 中的元组，且不同等价类之间不存在交集，因此，$\Sigma|[t]_X| \leq |R|$，故 $\Sigma|[t]_X|>|R|$ 不成立. 又根据定义 4，划分 Π_X 中包含了所有的等价类 $[t]_X$，而等价类又是在 R 的所有元组上进行定义的，因此所有等价类中元组个数之和不少于 R 中元组个数，因此有

$\Sigma|[t]_X|\geq|R|$，故 $\Sigma|[t]_X|<|R|$ 不成立. 综上，假设 $\Sigma|[t]_X|\neq|R|$ 不成立，$\Sigma|[t]_X|=|R|$ 成立. 证毕.

定义 5. 局部划分(Local Partition). 假定 $X\subset \mathrm{Attr}(R)$ 为关系 R 上的一个属性集合，关系 R 的实例 D 被横向切分为 n 块，$D=D_1\cup\cdots\cup D_n$，定义 $\Pi_{Xi}=\{[t]_X|t\in D_i, i\in[1,n]\}$ 为关系 R 在属性集合 X 上的一个局部划分.

例 4. 以表 2 中 EMP 的一个实例 D_{H1} 为例，关系 EMP 在属性 RESP 上的划分对于关系 EMP 来说是一个局部划分，为$\{\{1,9\},\{5,8\}\}$.

定义 6. 剥离划分(Stripped Partition). 假定 $X\subset \mathrm{Attr}(R)$ 为关系 R 上的一个属性集合，定义 $\Pi'_X=\{[t]_X|t\in R, |[t]_X|>1\}$ 为关系 R 在属性集合 X 上的一个剥离划分.

例 5. 以表 1 中 EMP 的一个实例 D_0 为例，根据定义 4，关系 EMP 在属性集合{ENO，ENAME}的剥离划分为去除仅包含一个元组的等价类后的等价类集合，即为$\{\{1,2\},\{3,4\},\{6,7\}\}$.

3.3 函数依赖冲突

函数依赖作为完整性约束的一种，其冲突意味着违反了完整性约束. 函数依赖冲突在现实中十分常见，特别是在数据集成、数据融合以及 Web 数据抽取应用中.

定义 7. 函数依赖冲突. 给定关系 R 的一个实例 D，函数依赖 $\varphi: A\rightarrow B$，对于 D 中的每一个元组 t，如果存在一个元组 t' 满足 $t(A)=t'(A)$ 且 $t(B)\neq t'(B)$，说明元组 t 和 t' 违反了函数依赖 φ. 将 D 上违反 φ 的冲突表示为 $Viot(\varphi,D)$，因此有 $t\in Viot(\varphi,D)$.

假定 Σ 为定义在 D 上的函数依赖的集合，则 $Viot(\Sigma,D)$ 表示 D 上的所有违反 Σ 中函数依赖的冲突. 用符号 $Viot^{\Pi}(\varphi,D)$ 表示 $\Pi_A Viot(\varphi,D)$，这里 $\Pi_A Viot(\varphi,D)$ 是 $Viot(\varphi,D)$ 在属性 A 上的投影，R 中其他属性值用空值 null 表示. 与 $Viot(\varphi,D)$ 相比，$Viot^{\Pi}(\varphi,D)$ 包含数据更少.

前面提到，一个关系 R 的任一水平分片与 R 有着相同的模式，因此，如果函数依赖 φ 是定义在 R 的实例 D 上面的，则 φ 同样也是定义在 D 的各个分片上的. 这里将 D 的任一水平切片 D_i 上违反函数依赖 φ 的冲突定义为 $Viot(\varphi,D_i)$.

引理 2. 给定关系 R，$\varphi: X\rightarrow Y$ 为关系 R 上的函数依赖，$[t]_X$ 为在属性集合 X 上的等价类，$[t]_{X\cup Y}$，$[t]'_{X\cup Y}$ 为在属性集合 $X\cup Y$ 上的等价类，对任意两个元组 t_i 和 t_j，如果满足 $t_i,t_j\in[t]_X$，$t_i\in[t]_{X\cup Y}$，$t_j\in[t]'_{X\cup Y}$，且 $[t]_{X\cup Y}\neq[t]'_{X\cup Y}$，则元组 t_i 和 t_j 为违反函数依赖 φ 的冲突元组.

证明. 由 $t_i,t_j\in[t]_X$，可知元组 t_i 和 t_j 属于 X 属性集合上的同一个等价类，根据等价类的定义 2，可以得到 $t_i[X]=t_j[X]$. 同理，由 $t_i\in[t]_{X\cup Y}$，$t_j\in[t]'_{X\cup Y}$，且 $[t]_{X\cup Y}\neq[t]'_{X\cup Y}$，根据定义 2，可以得到 $t_i[X\cup Y]\neq t_j[X\cup Y]$，即元组 t_i 和 t_j 在属性集合 $X\cup Y$ 上的值不同. 而元组 t_i 和 t_j 在属性集合 $X\cup Y$ 上的值包含了在属性集合 X 上的值，且由前面可知元组 t_i 和 t_j 在属性集合 X 上的值相同，$t_i[X\cup Y]$ 和 $t_j[X\cup Y]$ 去掉相同的属性值 $t_i[X]$ 和 $t_j[X]$，剩余部分为 $t_i[Y]$ 和 $t_j[Y]$，这里必然有 $t_i[Y]\neq t_j[Y]$，否则假定 $t_i[Y]=t_j[Y]$，而前面已知 $t_i[X]=t_j[X]$，则必然有 $t_i[X\cup Y]=t_i[X]\cup t_i[Y]=t_j[X\cup Y]$，与已知矛盾，假设不成立，故 $t_i[Y]\neq t_j[Y]$ 成立. 由定义 7 中函数依赖冲突的界定，可以得到元组 t_i 和 t_j 为违反函数依赖 φ 的冲突元组. 证毕.

例 6. 以表 1 中 EMP 的实例 D_0 为例，对于函数依赖 φ_1：ENO→ENAME，ENO 属性值为“E2”的等价类为 $\{3,4,5\}$，对应的基于属性集合{ENO，ENAME}的等价类为 $\{3,4\}$ 和 $\{5\}$. 不难看出，元组 t_3，t_4 和 t_5 属于等价类 $[t]_{\mathrm{ENO}}=\{3,4,5\}$，而 $t_3,t_4\in[t]_{\mathrm{ENO\cup ENAME}}=\{3,4\}$，$t_5\in[t]'_{\mathrm{ENO\cup ENAME}}=\{5\}$，根据引理 2，元组 t_3，t_4 和 t_5 为函数依赖 φ_1 的冲突元组.

3.4 函数依赖冲突检测

假定 D 为 R 的实例，D 被切分为 n 份，每一个切分 D_i 分布在不同的节点. 函数依赖 φ 是定义在 R 上的一个函数依赖，φ 的冲突检测问题是查找 $Viot^{\Pi}(\varphi,D)$. 给定函数依赖 φ，φ 可以在本地检测当且仅当满足 $Viot^{\Pi}(\varphi,D)=\bigcup_{i\in[1,n]}Viot^{\Pi}(\varphi,D_i)$.

具有最小响应时间的函数依赖不一致性检测问题，其输入是一个函数依赖的集合 Σ 和水平切分的关系 R 的实例 D，响应时间满足以下条件：(1) 集合 Σ 中的函数依赖冲突在数据迁移 U 之后可以本地检测；(2) 响应时间 RT 是最小的. 然而找到一个最小响应时间的函数依赖检测算法是不现实的，在水平切分情况下，具有最小响应时间的函数依赖不一致性检测问题是一个 NP-难问题[4].

4 分布式大数据多函数依赖冲突检测

函数依赖冲突检测是不一致性检测的重要内容. 在集中式环境下，函数依赖冲突检测主要取决于

数据规模和函数依赖的个数.在分布式大数据环境下,不仅需要进行数据的迁移,而且要考虑如何提高冲突检测的效率.根据文献[4],分布式环境下单个函数依赖的冲突检测问题已经是非平凡的,为 NP-难问题,由此可见,分布式环境下多个函数依赖的冲突检测问题同样为 NP-难问题.

4.1　方法 CenDet

方法 CenDet 在检测多个函数依赖冲突时,首先将分布式数据迁移到一个节点,然后利用集中式函数依赖冲突检测方法对多个函数依赖分别进行冲突检测,最终得到违反多个函数依赖的冲突元组.在具体的检测过程中,采用流水线方式,在完成一个函数依赖冲突检测后,进行下一个函数依赖的冲突检测,直到所有函数依赖冲突全部检测完毕为止.

方法 CenDet 在进行多个函数依赖冲突检测的时候,响应时间会比较长:由于检测每一个函数依赖冲突的时候都需要扫描一遍全部数据,而这种扫描是十分耗时的,在大数据背景下,方法 CenDet 的检测效率较低,响应时间较长.由于所有检测任务均在一个节点进行,因此方法 CenDet 的负载严重的不均衡.

函数依赖冲突检测耗时很大程度上取决于数据的规模,在数据规模特别大的情况下,对数据的每次扫描时间开销都很大,如果能够一次扫描全部数据而实现对多个函数依赖的冲突检测,则可以大大提高函数依赖的冲突检测效率.

4.2　方法 $\mathbf{MultiFDsDet_{DS}}$

方法 $\mathrm{MultiFDsDet_{DS}}$进行分布式大数据多函数依赖冲突检测时,包括以下几个步骤:(1)数据预处理(生成局部等价类);(2)任务分配;(3)数据迁移;(4)合并得到全局等价类;(5)冲突检测.

给定关系 R 的实例 D,待检测函数依赖集合$\Sigma=\{\varphi_1,\varphi_2,\cdots,\varphi_m\}$,假定 D 被水平切分为 n 份,分别为 $D_1,\cdots,D_n$,不同的分片分布在不同的节点,对 $\forall i\in[1,n]$,分片 D_i分布在节点 S_i上.为检测函数依赖集合 Σ 中函数依赖的冲突情况,在任一节点 $S_i,i\in[1,n]$,方法 $\mathrm{MultiFDsDet_{DS}}$首先进行数据的预处理,假定在节点 S_i数据预处理时间为 $PreDeal(D_i)$.数据预处理后,在各个节点得到各个待检测函数依赖的局部等价类.完成数据预处理后,方法 $\mathrm{MultiFDsDet_{DS}}$对函数依赖的冲突检测任务进行分配,对任一函数依赖 $\varphi_j,j\in[1,m]$,执行其冲突检测的节点为该函数依赖的执行节点.然后根据任务分配的结果进行数据迁移,将与每一个函数依赖相关的局部等价类数据迁移到该执行节点上.假定任一节点 S_i迁出的数据为 $D_{i\text{-out}}$,迁入的数据为 $\Delta D_{i\text{-in}}$,则总的数据迁移量为 $\Delta D=\sum\limits_{i\in[1,n]}\Delta D_{i\text{-out}}=\sum\limits_{i\in[1,n]}\Delta D_{i\text{-in}}$.所有节点数据迁移完成后,算法根据任务分配情况在各个节点进行局部等价类的合并,进而得到全局等价类.假定节点 S_i局部等价类合并耗时为 $merge(EC_i)$,EC_i为分配给节点 S_i的待检测函数依赖的局部等价类集合.得到全局等价类后,根据冲突检测任务分配的结果,算法在各个节点并行进行函数依赖冲突检测,假定数据迁移及局部等价类合并后节点 S_i的待检测数据为 D_i',冲突检测耗时为 $check(D_i')$.假定函数依赖集合 Σ 冲突检测总的响应时间代价模型为 $cost_{RT}(\Sigma)$,这里定义多函数依赖冲突检测响应时间代价模型如下:

$$cost_{RT}(\Sigma)=\max_{i\in[1,n]}PreDeal(D_i)+\frac{1}{bw}\sum_{i\in[1,n]}\Delta D_{i\text{-in}}+\max_{i\in[1,n]}merge(EC_i)+\max_{i\in[1,n]}check(D_i'),$$

式中 $PreDeal(D_i)$为各节点数据预处理(生成局部等价类)耗时,n 为节点个数,Σ 为待检测函数依赖集合,bw 为网络带宽,$merge(EC_i)$为各节点局部等价类合并得到全局等价类的耗时,$check(D_i')$为各节点函数依赖冲突检测耗时.

依据多函数依赖冲突检测响应时间代价模型,方法 $\mathrm{MultiFDsDet_{DS}}$检测多个函数依赖冲突的响应时间取决于四部分:各节点数据预处理耗时即生成局部等价类的耗时 $\max\limits_{i\in[1,n]}PreDeal(D_i)$,数据迁移耗时$\frac{1}{bw}\sum\limits_{i\in[1,n]}\Delta D_{i\text{-in}}$,各节点合并局部等价类得到全局等价类的耗时 $\max\limits_{i\in[1,n]}merge(EC_i)$,各节点执行函数依赖冲突检测的耗时 $\max\limits_{i\in[1,n]}check(D_i')$.为减少分布式环境多函数依赖冲突检测的响应时间,有必要在以上各个阶段考虑对其响应时间进行优化.

第 1 步是数据预处理,在各个节点生成局部等价类.假定待检测函数依赖集合为 Σ,对 Σ 中每个函数依赖 $\varphi_i,i\in[1,n]$,分别基于属性集合 LHS 和(LHS+RHS)在各个节点的数据上并行计算局部等价类$[t]_{\mathrm{LHS}}$和$[t]_{\mathrm{LHS\cup RHS}}$,得到形如〈key,value〉的键值对,其中 key 为元组在属性集合上的属性值组合,value 为有着相同属性值组合的局部等价类.

例 7.　以表 1 中 EMP 的一个实例 D_{H1} 为例,

对函数依赖 φ_3：TITLE→SAL，LHS 部分为属性集{TITLE}，(LHS+RHS)部分属性集合为{TITLE，SAL}. 在数据预处理阶段，在节点 S_1 上基于属性集合{TITLE}得到键值对〈“Syst. Anal.”，{1}〉，〈“Mech. Eng.”，{5，8}〉，〈“Analyst”，{9}〉，基于属性集合{TITLE，SAL}得到键值对〈“Syst. Anal._3500”，{1}〉，〈“Mech. Eng._2700”，{5}〉，〈“Mech. Eng._2500”，{8}〉，〈“Analyst_3300”，{9}〉. 同理，在节点 S_2 及 S_3 上进行数据的预处理.

在各节点并行预处理得到各个函数依赖的局部等价类后，接下来对局部等价类数据进行迁移，使得相同函数依赖的所有局部等价类数据迁移到同一节点，如图 1 所示.

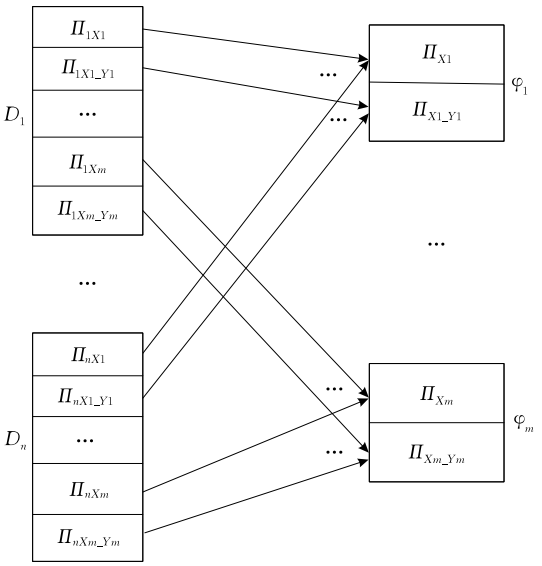


图 1 D_1，…，D_n 中局部等价类合并

算法 1 为数据预处理算法.

算法 1. *PreDeal.*

输入：函数依赖集合 $\Sigma=\{\varphi_1,\cdots,\varphi_m\}$，$D=(D_1,\cdots,D_n)$

输出：$[t_{X1}]_{\text{LHS}}$，$[t_{X1\cup Y1}]_{\text{LHS}\cup\text{RHS}}$，…，$[t_{Xm}]_{\text{LHS}}$，$[t_{Xm\cup Ym}]_{\text{LHS}\cup\text{RHS}}$

1. Map(k_L，v_L) M_L←null，Map($k_{L\cup R}$，$v_{L\cup R}$)$M_{L\cup R}$←null；
2. FOR EACH $\varphi_j\in\Sigma$ DO
3. Map(k_L，v_L)m_L←null，Map($k_{L\cup R}$，$v_{L\cup R}$)$m_{L\cup R}$←null；
4. FOR EACH $t\in D_i$ DO//生成局部键值对
5. IF !$m_L.containsKey(t(X_j))$
6. $v_L.add(t(ID))$；
7. $m_L.put(t(X_j),v_L)$；
8. ELSE IF !$m_L.get(t(X_i)).containsKey(t(ID))$
9. $m_L.get(t(X_j)).add(t(ID))$；
10. IF !$m_{L\cup R}.containsKey(t(X_j))$
11. $v_{L\cup R}.add(t(ID))$；
12. $m_{L\cup R}.put(t(X_j),v_{L\cup R})$；
13. ELSE IF !$m_{L\cup R}.get(t(X_j)).containsKey(t(ID))$
14. $m_{L\cup R}.get(t(X_j)).add(t(ID))$；
15. END FOR；
16. IF $M_L.containsKey(m_L.getKey())$ //局部键值对合并
17. $M_L.get(m_L.getKey()).add(m_L.getValue())$；
18. ELSE $M_L.put(m_L.getKey(),m_L.getValue())$；
19. IF $M_{L\cup R}.containsKey(m_{L\cup R}.getKey())$
20. $M_{L\cup R}.get(m_{L\cup R}.getKey()).add(m_{L\cup R}.getValue())$；
21. ELSE $M_{L\cup R}.put(m_{L\cup R}.getKey(),m_{L\cup R}.getValue())$；
22. $[t_{Xj}]_{\text{LHS}}$←M_L；
23. $[t_{Xj\cup Yj}]_{\text{LHS}\cup\text{RHS}}$←$M_{L\cup R}$；
24. END FOR；
25. RETURN $[t_{X1}]_{\text{LHS}}$，$[t_{X1\cup Y1}]_{\text{LHS}\cup\text{RHS}}$，…，$[t_{Xm}]_{\text{LHS}}$，$[t_{Xm\cup Ym}]_{\text{LHS}\cup\text{RHS}}$.

算法 1 的输入为待检测分布式数据(D_1，…，D_n)及函数依赖集合 Σ，输出为每个待检测函数依赖关于 LHS 和(LHS+RHS)的局部等价类. 算法首先对每个待检测函数依赖的 LHS 和(LHS+RHS)按照元组分别生成局部键值对，然后对有着相同 key 值的局部键值对进行合并，最后得到待检测函数依赖的 LHS 和(LHS+RHS)局部等价类.

例 8. 表 1 中 EMP 的实例 D_0 被切分为 D_{H1}，D_{H2}，D_{H3}，对函数依赖 φ_3：TITLE→SAL，由例 7 中的局部等价类，根据图 1 中的合并规则，可以得到 φ_3 基于属性集合{TITLE}的划分 $\Pi_{\text{TITLE}}=\{\{1\},\{2\},\{3,6,10\},\{4,7,9\},\{5,8\}\}$ 以及基于属性集合{TITLE，SAL}的划分 $\Pi_{\text{TITLE\_SAL}}=\{\{1\},\{2\},\{3\},\{6,10\},\{4,9\},\{7\},\{5\},\{8\}\}$.

本文中需要检测的函数依赖个数为 m，节点个数为 n，对每个 $i=1,2,\cdots,m$ 和每个 $j=1,2,\cdots,n$，第 i 个函数依赖在第 j 个节点上检测耗时为 t_{cij}，第 i 个函数依赖在第 j 个节点上局部等价类数据合并耗时为 t_{mij}，依据多函数依赖冲突检测响应时间代价模型 $cost_{RT}(\Sigma)$，要想减少总的响应时间，应当最小化数据迁移耗时 $\frac{1}{bw}\sum_{i\in[1,n]}\Delta D_{i\text{-in}}$、局部等价类合并耗时 $\max_{i\in[1,n]}merge(EC_i)$、函数依赖冲突检测耗时 $\max_{i\in[1,n]}check(D'_i)$，也就是对待检测函数依赖进行任务分配后，数据迁移耗时、局部等价类合并耗时以及冲突检测耗时之和最小. 这里定义 x_{ij} 为第 j 个节点上处理第 i 个函数依赖，则当第 i 个函数依赖分配

给第 j 个节点时，$x_{ij}=1$，否则 $x_{ij}=0$. 该问题可以转化为如下的整数规划问题.

目标函数：$\min\ t_m+t_c+\frac{1}{bw}\sum_{i\in[1,n]}\Delta D_{i\text{-in}}$，

约束条件：

$$\forall j\in[1,n],\ t_c\geq\sum_{i=1}^{m}x_{ij}t_{cij},\ t_m\geq\sum_{i=1}^{m}x_{ij}t_{mij};$$

$$\forall j\in[1,n],\ \Delta D_{j\text{-in}}=\sum_{i=1}^{m}x_{ij}[(|\Pi_{Xi}|-|\Pi_{jXi}|)+(|\Pi_{Xi\_Yi}|-|\Pi_{jXi\_Yi}|)];\ \sum_{j=1}^{n}x_{ij}=1,\ 1\leq i\leq m;$$

$$x_{ij}\in\{0,1\},\ 1\leq i\leq m,\ 1\leq j\leq n.$$

上述整数规划中，t_m 为所有节点局部等价类合并耗时，t_c 为所有节点函数依赖冲突检测耗时，t_{mij} 为第 i 个函数依赖在第 j 个节点上局部等价类合并耗时，t_{cij} 为第 i 个函数依赖在第 j 个节点上冲突检测耗时，$|\Pi_{Xi}|$ 为第 i 个函数依赖 LHS 部分在全局数据上的划分，$|\Pi_{Xi\_Yi}|$ 为第 i 个函数依赖 LHS∪RHS 部分在全局数据上的划分，$|\Pi_{jXi}|$ 为第 i 个函数依赖 LHS 部分在第 j 个节点数据上的局部划分，$|\Pi_{jXi\_Yi}|$ 为第 i 个函数依赖 LHS∪RHS 部分在第 j 个节点数据上的局部划分.

上述函数依赖任务分配问题为多处理机任务分配问题，该问题为 NP-完全问题[4]，难以在多项式时间内得到问题的精确解，本文给出一种多函数依赖冲突检测任务分配的近似最优算法，如算法 2 所示.

算法 2.　*LoadAllocateByFD*.

输入：机器集合 $M=\{M_i\}$，$i\in[1,n]$，任务集合 $L=\{L_1,\cdots,L_m\}$，其中 $L_j=\{\Pi_{Xj},\Pi_{Xj\_Yj}\}$，$j\in[1,m]$

输出：$L(1),L(2),\cdots,L(n)$

1. FOR EACH $i\in[1,n]$ DO
2. $L_i\leftarrow\{\}$;
3. END FOR;
4. $Desc(|\Pi_{Xj}|)$;//依据 L 中等价类集合 Π_X 的势的大
　　//小对 L 中任务降序排列
5. 假定 $|\Pi'_{X1}|\geq|\Pi'_{X2}|\geq\cdots\geq|\Pi'_{Xm}|$;
6. FOR EACH $j\in[1,m]$ DO
7. 　IF 分配给机器 M_i 的任务包含等价类个数最少
8. 　　$L(i)\leftarrow L(i)\cup\{\Pi_{Xj},\Pi_{Xj\_Yj}\}$;
9. 　END IF;
10. END FOR.

算法 2 的输入为处理机集合 M 以及待分配任务集合 L，输出为任务分配结果. 算法首先依据任务集合中等价类 Π_X 势的大小对 L 中任务降序排列，在得到降序排列结果后，开始进行任务分配. 任务分配过程中，对当前要分配的任务，找出目前被分配的任务包含等价类个数最少的机器，将当前任务分配给该机器，直到所有任务都分配完成.

算法将一个待检测函数依赖基于 LHS 的划分 Π_{Xj} 及基于 LHS∪RHS 的划分 Π_{Xj_Yj} 作为一个整体的任务进行分配，根据等价类集合 Π_X 的势（等价类集合 Π_X 中包含元素的个数）的大小将多个划分的集合进行降序排列，然后从排序后的任务集合中依次取出任务集合进行分配，每次分配时选择当前任务负载最小的节点进行分配，在函数依赖个数 m 超过节点个数 n 的情况下，使用贪心算法进行任务分配的时候，通常很难得到最优解，而算法 *LoadAllocateByFD* 可以得到近似最优解.

引理 3.　给定关系 R，$\varphi:X\rightarrow Y$ 为关系 R 上的函数依赖，Π_X 为在属性集合 X 上的划分，Π_{XY} 为在属性集合 $X\cup Y$ 上的划分，则有 $|\Pi_X|\leq|\Pi_{XY}|$.

证明.　假定 $|\Pi_X|>|\Pi_{XY}|$，对每一个 $[t]_X\in\Pi_X$，如果 $\exists[t]_Y\in\Pi_Y$，使得 $[t]_X=[t]_Y$，则据定义 2，有 $[t]_X=[t]_Y=[t]_{XY}$，由定义 4，有 $\Pi_X=\Pi_{XY}$，因此 $|\Pi_X|=|\Pi_{XY}|$，与假设矛盾. 否则，对每一个 $[t]_X\in\Pi_X$，如果 $\nexists[t]_Y\in\Pi_Y$，使得 $[t]_X=[t]_Y$，则由定义 2，有 $[t]_X\neq[t]_{XY}$，根据定义 4，必然 $\exists[t]_Y,[t]'_Y\in\Pi_Y$，使得 $[t]_X=[t]_Y\cup[t]'_Y$. 同理存在 $\exists[t]_{XY},[t]'_{XY}\in\Pi_{XY}$，使得 $[t]_X=[t]_{XY}\cup[t]'_{XY}$. 又根据引理 1，$\Sigma|[t]_X|=\Sigma|[t]_{XY}|=|R|$，所以 Π_X 中等价类个数少于 Π_{XY} 中等价类个数，即 $|\Pi_X|<|\Pi_{XY}|$，与假设矛盾. 综上，假设不成立，$|\Pi_X|\leq|\Pi_{XY}|$ 成立.　　证毕.

由引理 3，对于待检测函数依赖 $\varphi:X\rightarrow Y$，满足 $|\Pi_X|\leq|\Pi_{XY}|$，即基于 LHS 属性集合的划分所包含的等价类个数不超过基于 LHS∪RHS 属性集合的划分所包含的等价类个数，因此在对 φ 进行冲突检测时，可以从 $|\Pi_X|$ 中等价类出发，在 $|\Pi_{XY}|$ 中寻找相应的冲突等价类.

基于等价类的函数依赖冲突检测问题可以视为多叉树生成问题，该多叉树包含 3 层，第 0 层为包含所有元组的根节点，第 1 层为包含所有基于 LHS 属性集合的等价类的中间节点，第 2 层为包含所有基于 LHS∪RHS 属性集合的等价类的叶子节点. 图 2 为函数依赖等价类生成树的一个示例.

其中节点 $[t_1]_X$ 为关系 R 在函数依赖 $\varphi:X\rightarrow Y$ 左端即属性集合 $\{X\}$ 上的一个等价类，而等价类 $[t_{1-1}]_{XY},[t_{1-2}]_{XY},\cdots,[t_{1-k}]_{XY}$ 为节点 $[t_1]_X$ 的子节点，同时也是等价类生成树的叶子节点. 函数依赖等价类生成树有着如下的性质.

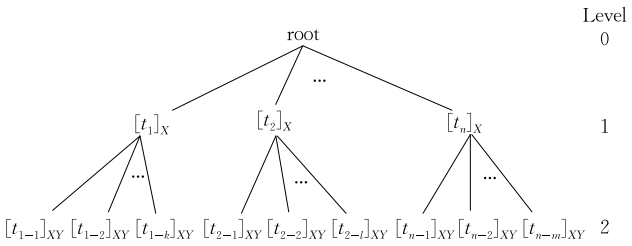


图 2 函数依赖等价类生成树示例

性质 1. 在函数依赖等价类生成树中,任一中间节点所包含元组在函数依赖 LHS 上的属性值与所有其子节点在函数依赖 LHS 上的属性值相同.

性质 2. 在函数依赖等价类生成树中,中间节点所包含元组的集合为其所有子节点所包含元组集合的并集.

性质 3. 在函数依赖等价类生成树中,所有中间节点所包含元组的并集等于关系 R 中所有元组集合.

性质 4. 在函数依赖等价类生成树中,对任一中间节点,如果其所包含的子节点个数大于 1,则该中间节点包含冲突元组,且其不同子节点中所包含的元组互为冲突元组.

根据上面的分析可以看出,基于等价类的函数依赖冲突检测本质上就是得到等价类生成树.而通常情况下,等价类生成树中会包含不存在冲突的中间节点和叶子节点,有必要对这些节点进行剪枝,通过剪去不存在冲突的中间节点和叶子节点,从而提高检测效率.

给定关系 R,$\varphi: X \rightarrow Y$ 为关系 R 上的函数依赖,$[t]_X$ 为在属性集合 X 上的等价类,$[t]_{X \cup Y}$ 为在属性集合 $X \cup Y$ 上的等价类,如果满足 $[t]_X = [t]_{X \cup Y}$,则 $[t]_X$ 或 $[t]_{X \cup Y}$ 中的元组关于函数依赖 φ 不存在冲突.对 $\forall t_i, t_j \in [t]_X, t_i \neq t_j$,由 $[t]_X = [t]_{X \cup Y}$ 可知 $t_i, t_j \in [t]_{X \cup Y}$.根据定义 2,由 $t_i, t_j \in [t]_{X \cup Y}$ 可知 $t_i[XY] = t_j[XY]$,由 $t_i, t_j \in [t]_X$ 可知 $t_i[X] = t_j[X]$.由 $t_i[XY] = t_j[XY]$ 和 $t_i[X] = t_j[X]$,将等式 $t_i[XY] = t_j[XY]$ 左右两端去掉相同部分 $t_i[X]$ 和 $t_j[X]$ 可以得到 $t_i[Y] = t_j[Y]$,根据函数依赖冲突的定义 7,元组 t_i, t_j 关于函数依赖 φ 不存在冲突.

可以看出,如果函数依赖等价类生成树中中间节点仅包含一个子节点,则该中间节点对应的等价类中不包含冲突元组.

剪枝策略 1. 给定关系 R,$\varphi: X \rightarrow Y$ 为关系 R 上的函数依赖,Π_X 为关系 R 在属性集合 X 上的一个划分,在检测函数依赖 φ 的冲突时,首先去除 Π_X 中包含单个元素的等价类,得到 Π_X 的剥离划分 Π'_X,然后去除 Π_{XY} 中所包含的同时在 Π_X 和 Π_{XY} 中为单个元素的等价类,得到 $\Pi''_{XY} = \Pi_{XY} \backslash (\Pi_X \backslash \Pi'_X)$,基于 Π'_X 和 Π''_{XY} 进行冲突检测.

剪枝策略 2. 对剪枝后得到的 Π'_X 和 Π''_{XY},去除其中包含的公共的等价类,得到 $\Pi''_X = \Pi'_X \backslash (\Pi'_X \cap \Pi''_{XY})$ 和 $\Pi'''_{XY} = \Pi''_{XY} \backslash (\Pi'_X \cap \Pi''_{XY})$,基于 Π''_X 和 Π'''_{XY} 进行冲突 2 检测.

经过剪枝策略 1 和 2 剪枝后,可以得到仅包含冲突元组的等价类,基于该等价类进行函数依赖冲突检测,可以有效减少冲突检测的比对空间,提高检测效率.

例 9. 以表 1 中 EMP 的一个实例 D_0 为例,对函数依赖 φ_1: ENO → ENAME,D_0 在属性集合 {ENO} 上的划分为 $\Pi_{\text{ENO}} = \{\{1,2\},\{3,4,5\},\{6,7\},\{8\},\{9,10\}\}$,在 {ENO, ENAME} 上的划分为 $\Pi_{\text{ENO\_ENAME}} = \{\{1,2\},\{3,4\},\{5\},\{6,7\},\{8\},\{9\},\{10\}\}$.图 3 是等价类 Π_{ENO} 和 $\Pi_{\text{ENO\_ENAME}}$ 生成树剪枝过程示意图.第 1 次剪枝,分别从 Π_{ENO} 和 $\Pi_{\text{ENO\_ENAME}}$ 中剪去了包含单个元组的等价类 {8},得到如图 3 中间的等价类生成树.第 2 次剪枝,分别从 Π_{ENO} 和 $\Pi_{\text{ENO\_ENAME}}$ 中剪去了包含单个元组的等价类 {1,2} 和 {6,7},得到如图 3 下部的等价类生成树.从图 3 不难看出,经过两次剪枝后得到的等价类生成树为一棵仅包含冲突元组的等价类生成树,也就是一棵冲突等价类生成树.

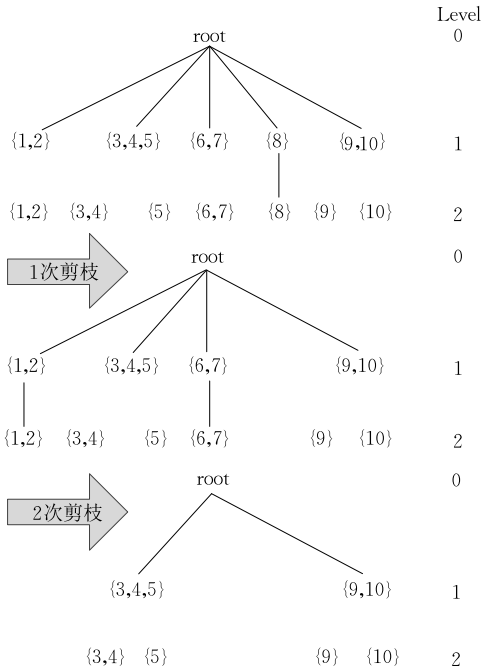


图 3 D_0 中函数依赖等价类生成树剪枝过程

经过上述剪枝策略剪枝后得到包含冲突元组的等价类，为进一步确定图 2 中第 1 层每一个冲突等价类的子集合，也就是构造冲突等价类生成树，需要将剪枝后第 1 层中的任一节点与第 2 层中的叶子节点进行匹配，以确定是否为父子关系. naïve 的匹配方法是对第 1 层中的任一节点，取第 2 层中所有未匹配上的叶子节点进行匹配，这种匹配方法由于要与第 2 层中所有未匹配上的叶子节点进行匹配，因此效率不高.

为提升匹配效率，首先对两次剪枝后得到的第 1 层的等价类集合 $\Pi''_X=\Pi'_X\backslash(\Pi'_X\cap\Pi''_{XY})$ 中的所有等价类根据其势的大小从小到大进行排序，得到一个升序的等价类集合 $\Pi''_X{}^{\uparrow}$. 然后从等价类势最小的一端逐个选取等价类进行匹配，在匹配时，先对 $\Pi'''_{XY}=\Pi''_{XY}\backslash(\Pi'_X\cap\Pi''_{XY})$ 中的等价类进行剪枝.

给定关系 R，$\varphi: X\rightarrow Y$ 为关系 R 上的函数依赖，Π_X 为在属性集合 X 上的划分，Π_{XY} 为在属性集合 $X\cup Y$ 上的划分，$[t]_X$ 为在属性集合 X 上的等价类，如果 $[t]_X$ 中存在冲突元组，则对 $\forall[t]_{X\cup Y}\in\Pi_{XY}$，如果 $[t]_{X\cup Y}\subseteq[t]_X$，即 $[t]_{X\cup Y}$ 为 $[t]_X$ 的子集合，则 $|[t]_X|>|[t]_{X\cup Y}|$. 等价类集合 $\Pi''_X{}^{\uparrow}$ 中的所有等价类，其子集合的势均小于等价类自身的势，因此对 $\Pi'''_{XY}=\Pi''_{XY}\backslash(\Pi'_X\cap\Pi''_{XY})$ 中的等价类进行剪枝时，对 $\forall[t]_X\in\Pi''_X{}^{\uparrow}$，如果 $[t]_{X\cup Y}\in\Pi'''_{XY}$ 且 $|[t]_{X\cup Y}|\geq|[t]_X|$，则可以将 $[t]_{X\cup Y}$ 从 Π'''_{XY} 中移除.

例 10.　以图 3 中 2 次剪枝后得到的等价类为例，其中 $\Pi''_{ENO}=\{\{3,4,5\},\{9,10\}\}$，$\Pi'''_{ENO\cup ENAME}=\{\{3,4\},\{5\},\{9\},\{10\}\}$. 首先对 Π''_{ENO} 中的等价类根据其势的大小从小到大排序，得到 $\Pi''_{ENO}{}^{\uparrow}=\{\{9,10\},\{3,4,5\}\}$. 然后选取 $\Pi''_{ENO}{}^{\uparrow}$ 中的第 1 个等价类 $\{9,10\}$，在从 $\Pi'''_{ENO\cup ENAME}$ 中寻找等价类 $\{9,10\}$ 的子集合时，存在冲突的等价类 $\{9,10\}$ 的子集合的势必然小于 $\{9,10\}$ 的势，即小于 2. 因此可以从 $\Pi'''_{ENO\cup ENAME}$ 中移除势不小于 2 的等价类，这里满足移除条件的等价类为 $\{3,4\}$，因此可以将其移除，而不影响匹配的结果. 表 5 对 Π''_{ENO} 中等价类排序前后最坏情况下的匹配及判断次数进行了比较.

表 5　等价类排序前后最坏匹配及判断次数比较

	等价类	最坏匹配次数	每次匹配最坏判断次数	合计
排序前	{3,4,5}	4	3	16
	{9,10}	2	2	
排序后	{3,4,5}	2	3	12
	{9,10}	3	2	

从表 5 可以看出，在最坏情况下，对 Π''_{ENO} 中等价类进行排序后总的匹配次数为 5 次，判断次数为 12 次，而不对 Π''_{ENO} 中等价类进行排序直接进行匹配和判断，则需要 6 次匹配和 16 次判断，相比较而言，对 Π''_{ENO} 中的等价类根据其势的大小从小到大排序可以明显减少匹配和判断的次数，提高检测效率.

在多个函数依赖不一致性检测时，假定待检测函数依赖个数为 m，待检测数据分布在 n 个节点，根据函数依赖个数及节点个数的大小情况不同采用不同的负载分配策略.

(1) $m\geq n$，即函数依赖个数大于等于节点个数，以函数依赖为最小单位进行检测任务的分配，可以保证所有节点均有负载；

(2) $m<n$，即函数依赖个数小于节点个数，此种情况将函数依赖的冲突等价类依据节点个数进行分组，将不同的等价类分组分配到不同的节点进行处理，保证所有节点都有负载，而且使得各节点间的负载尽可能的均衡.

对于函数依赖个数大于等于节点个数的情况，可以利用算法 2 中给出的任务分配策略进行任务分配，使得检测任务的分配尽可能的均衡. 对于函数依赖个数小于节点个数的情况，以函数依赖为单位进行任务的分配会出现有些节点负载为空的情况，检测效率不高，本文给出一种函数依赖个数小于节点个数情况下多函数依赖冲突检测任务分配的近似最优算法，具体见算法 3.

算法 3.　*LoadAllocateByEC.*

输入：机器集合 $M=\{M_i\}, i\in[1,n]$，函数依赖集合 $\Sigma=\{\varphi_1,\cdots,\varphi_m\}$

输出：$FL(1),FL(2),\cdots,FL(m)$

```
1.  FOR EACH i∈[1,m] DO
2.      FL(i)←{};
3.  END FOR;
4.  FOR EACH i∈[1,m] DO
5.      FOR EACH j∈[1,n] DO
6.          IF |FL(i)|<⌊n/m⌋ && i<m
7.              FL(i)←FL(i)∪M_j;
8.          M←M\M_j;
9.      ELSE IF i==m //最后剩余的节点分配给最后
                     //一个 FD
10.         FL(i)←M;
11.     END FOR;
12. END FOR.
```

算法 3 的输入为处理机集合 M 以及待分配函数依赖集合 Σ，输出为任务分配结果. 为使得待检测

函数依赖所分配的执行节点个数尽可能的均衡,对前 $m-1$ 个待检测 FD,算法首先将执行节点个数根据待检测 FD 个数进行均分,每个待检测 FD 分配 $\lfloor n/m \rfloor$ 个执行节点,其中 m 为待检测函数依赖总数,n 为节点总数,最后得到的第 m 个待检测 FD 所分配的执行节点个数为前面 $m-1$ 个待检测 FD 分配后剩余的节点个数. 基于以上的分配策略,算法 *LoadAllocateByEC* 可以得到近似最优解.

对于待检测的函数依赖,假定其检测任务需要分配到不同节点并行进行不一致性检测. 在各个节点并行计算局部等价类$[t]_{\mathrm{LHS}}$和$[t]_{\mathrm{LHS}\cup\mathrm{RHS}}$得到形如〈key,value〉的键值对后,根据 $t[\mathrm{LHS}]$值对得到的键值对进行分组,分组基于散列函数进行,输入即散列函数的关键字为 $t[\mathrm{LHS}]$值,散列函数的输出由负载该函数依赖的节点个数确定. 对于各节点数据预处理得到的键值对,根据散列函数值的分布情况将有着相同散列值的键值对散列到同一执行节点,不同执行节点上的键值对有着不同的散列值. 这样可以实现有着相同散列值的键值对在同一个执行节点处理. 键值对散列的示意图如图 4 所示.

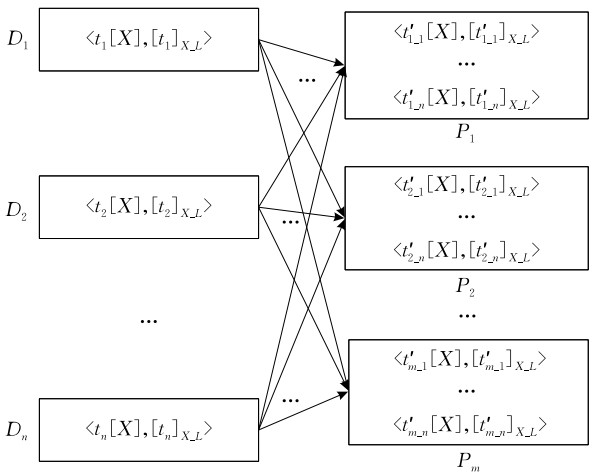


图 4 函数依赖局部等价类〈键,值〉对散列示意图

图 4 中 $t_i[X]$为 D_i 中元组在属性集合 X 上的属性值,$[t_i]_{X_L}$为 D_i 中元组在属性值 $t_i[X]$上的局部等价类,$\langle t'_{m_j}[X],[t'_{m_j}]_{X_L}\rangle$为 D_j 中键值对集合根据哈希函数散列后散列值为 $m-1$ 的键值对集合,所有散列值为 $m-1$ 的键值对被分配到数据分块 P_m 上. 这种散列策略使得有着相同 $t_i[X]$的键值对散列到相同的执行节点. 根据引理 2,只有 $t_i[X]$相同的等价类中元组才可能存在冲突,因此图 4 中的局部等价类分配策略使得所有存在潜在冲突的元组分配到相同的节点,保证了冲突检测的准确性.

基于 Hadoop 平台进行数据处理时,由于数据的特点及分布情况不同,可能会出现默认的分区函数 *Partitioner*()得到的数据分块大小差异较大的情况,这会导致在 Reduce 阶段负载分配的不均衡,一些 Reducer 节点负载过大,总耗时取决于耗时最长通常也是负载最大节点的耗时,因此处理效率不高. 本文针对 Hadoop 内核进行修改,在 Map 阶段,根据 partition 函数将待检测数据划分成数量多于 reducer 个数的 partition,采用改进的负载均衡贪心算法对得到的 partition 进行合并,合并后的数据分块数量与执行节点个数即 Reducer 个数相同,将合并后的数据分块分配到 Reducer 上,保证每个 Reducer 处理一个合并后的数据分块,具体算法如算法 4 所示.

算法 4. *PartitionAllocate*.

输入:节点集合 $M=\{M_i\}, i\in[1,n]$,数据分块集合 $P=\{P_1,\cdots,P_m\}$

输出:$A(1),A(2),\cdots,A(n)$

1. FOR EACH $i\in[1,n]$ DO
2. $A(i)\leftarrow\{\}$;
3. END FOR;
4. $Desc(|P_j|)$;//依据 P 中数据分块的大小对数据块降序排列
5. 假定$|P'_1|\geq|P'_2|\geq\cdots\geq|P'_m|$;
6. FOR EACH $j\in[1,m]$ DO
7. IF M_i是达到最小值 $\min|A(i)|$的节点
8. $A(i)\leftarrow A(i)\cup P'_j$;
9. END IF;
10. END FOR.

算法 4 的输入为节点集合 M 和待分配的数据分块集合 P,输出为数据分块的分配结果,$A(i)$表示分配给节点 M_i 的数据分块集合. 算法根据待分配数据分块集合中数据块大小对所有数据分块降序排列,然后对排序后的数据分块,找出当前分配的数据规模最小的节点,将当前数据分块分配给该节点,直到所有数据块都分配完成.

假定对于函数依赖 $\varphi: X\rightarrow Y$,执行节点个数为 k,根据算法 4 中的任务分配策略,每个执行节点可以得到部分冲突元组,所有 k 个执行节点所得到的关于 φ 的部分冲突元组的并集即为 φ 在全局数据上的冲突元组集合,具体如图 5 所示.

根据函数依赖检测任务分配策略,如果函数依赖个数少于节点个数,对每一个待检测函数依赖,有着相同 LHS 属性值的键值对被散列到相同节点,而根据引理 2,潜在冲突的元组其散列值相同,因此

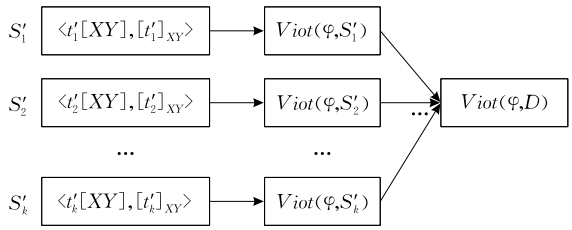


图 5　各执行节点部分冲突元组得到全部冲突元组

被散列到同一个节点. 对函数依赖 $\varphi: X \rightarrow Y$，在各个执行节点，对散列到该节点的所有键值对，根据键中包含的 LHS 部分(这里为 X)属性值是否相同，对所有键值对进行划分，将有着相同 LHS 属性值的键值对划分到同一个组，得到该 LHS 部分属性值的等价类. 然后根据 RHS 部分属性值是否相同，将有着相同 RHS 部分属性值的键值对划分到同一个组，得到关于 RHS 部分属性值的等价类. 根据引理 2，对每一个基于 LHS 部分属性值的等价类，如果其对应的 LHS∪RHS 属性值等价类个数大于 1，则说明其对应的不同 LHS∪RHS 属性值等价类之间违反函数依赖 φ.

例 11.　以表 1 中 EMP 的一个实例 D_0 为例，对函数依赖 φ_1：ENO→ENAME，假定节点个数为 2，分别为 S_1 和 S_2，在散列后分配到节点 S_1 上的在属性集合{ENO}的键值对合并后为〈“E1”，{1，2}〉，〈“E2”，{3，4，5}〉，在属性集合{ENO，ENAME}的键值对合并后为〈“E1_M. Smith”，{1，2}〉，〈“E2_J. Jones”，{3，4}〉，〈“E2_J. Davis”，{5}〉. 键值对〈“E2_J. Jones”，{3，4}〉，〈“E2_J. Davis”，{5}〉的 LHS 属性值相同，RHS 属性值不同，根据引理 2，等价类{3，4}和{5}为冲突等价类. 在节点 S_2，同理. 对于不存在冲突元组，该等价类可以被剪枝掉. 冲突等价类子树合并后可以构建出冲突等价类生成树，具体生成过程如图 6 所示.

在函数依赖冲突检测过程中，由于不同节点任务分配情况以及存在冲突情况的不同，可能导致检测过程中负载不均衡情况的出现. 为提高检测效率，本文考虑在检测过程中进行动态负载均衡. 可以将动态负载均衡时节点间负载迁移情况用有向图 $G(V,E)$ 的形式表示，V 为图的顶点集合，每个顶点代表一个节点，E 为有向边的集合，每条边代表节点间负载的迁移关系. 假定节点个数为 m 个，节点 S_i 的当前负载用 l_i 表示，当前所有节点的平均负载为 $\bar{l}=\sum_{i=1}^{m} l_i/m$，则 $l_i-\bar{l}$ 为节点 S_i 需要迁移的负载量，

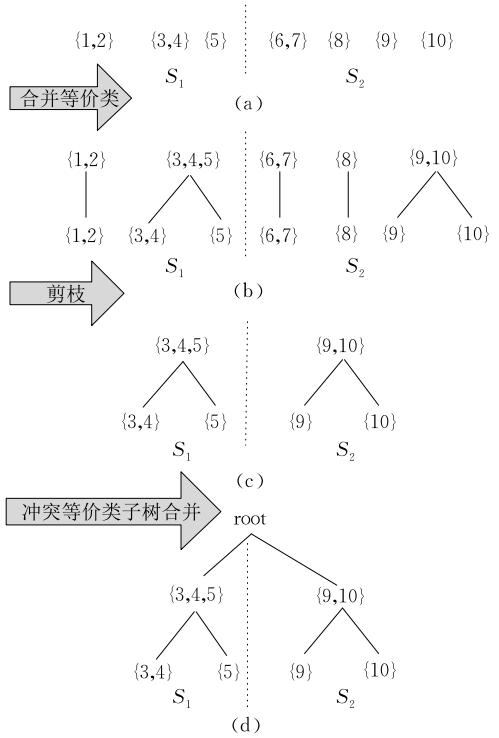


图 6　D_0 中函数依赖冲突等价类生成树生成过程

该值>0，说明负载超出平均负载，反之说明负载低于平均负载. 边 e_{ij} 的权重表示从节点 S_i 到 S_j 的负载迁移量，用 ε_{ij} 表示. 对 $\forall i \in [1,m]$ 有 $\sum_{\{j|e_{ij}\in E\}} \varepsilon_{ij} = l_i - \bar{l}$[2]，可得如下线性方程组：

$$\begin{cases} \sum\limits_{\{j|e_{ij}\in E\}} \varepsilon_{1j} = l_1 - \bar{l} \\ \cdots \\ \sum\limits_{\{j|e_{mj}\in E\}} \varepsilon_{mj} = l_m - \bar{l} \end{cases} \tag{1}$$

由于负载总迁入量和总迁出量相同，在式(1)的 m 个等式中，如果其中任意 $m-1$ 个成立，则剩余的那个等式必然成立. 假定式(1)中线性方程组的系数矩阵为 $\boldsymbol{A}$，则 $r(\boldsymbol{A}) \leq m-1$，根据线性方程组解的性质可知该线性方程组有无穷多解. 为减少总的负载迁移量，有必要找出负载迁移量最小的解.

假定 $\boldsymbol{x}$ 为式(1)中等式左端关于 ε_{ij} 的向量，$\boldsymbol{b}$ 为等式右端的列向量，则可将最小化负载迁移量问题归结为如下的二次规划问题[17]：

$$\begin{aligned} &\text{目标函数：} \quad \min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} \\ &\text{s. t.} \qquad\quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{aligned} \tag{2}$$

其中：$\boldsymbol{x}^{\mathrm{T}}$ 为向量 $\boldsymbol{x}$ 的转置，$\boldsymbol{A}$ 为 $|V|\times|E|$ 的矩阵，$|V|$ 为图 G 中顶点个数，$|E|$ 为边个数. $\boldsymbol{A}_{|V|\times|E|}=$

$[a_{ij}]_{|V|\times|E|}$，$a_{ij}=\begin{cases}1, & i\text{ 为边 }e_{ij}\text{ 的始点}\\ -1, & i\text{ 为边 }e_{ij}\text{ 的终点}.\\ 0, & \text{其他}\end{cases}$

上述优化问题为有等式约束的二次规划问题，可采用拉格朗日乘子法求解. 可构造拉格朗日函数如下 $\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}-\boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$.

令 $\nabla_x\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})=0$，得到 $\boldsymbol{x}-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda}=0$，即 $\boldsymbol{x}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda}$. 将 x 值代入式(2)中，得 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda}=\boldsymbol{b}$. 令 $\boldsymbol{L}=\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$，可得

$$\boldsymbol{L}\boldsymbol{\lambda}=\boldsymbol{b} \tag{3}$$

上述最小化负载迁移量问题转化为式(3)的线性方程组求解问题，求解式(3)得到任一顶点 S_i 的拉格朗日乘子 λ_i，则 $\lambda_i-\lambda_j$ 为从节点 S_i 到 S_j 的负载迁移量.

分布式环境多函数依赖冲突检测方法 Multi-FDsDet$_{DS}$ 如算法 5 所示.

算法 5. MultiFDsDet$_{DS}$.

输入：函数依赖集合 $\Sigma=\{\varphi_1,\cdots,\varphi_m\}$，$D=(D_1,\cdots,D_n)$，$M=(M_1,\cdots,M_n)$

输出：$Viot^{\Pi}(\varphi_1,D),Viot^{\Pi}(\varphi_2,D),\cdots,Viot^{\Pi}(\varphi_m,D)$

```
1.  L←PreDeal(Σ,D);//数据预处理
2.  IF m≥n //函数依赖个数大于等于节点个数
3.    LoadAllocateByFD(M,L);
4.    FOR EACH i∈[1,n] DO
5.      FOR EACH {Π_Xj,Π_Xj_Yj}∈L(i) DO
6.        {Π'_Xj,Π'_Xj_Yj}←prune(Π_Xj,Π_Xj_Yj);
7.        Viot^Π(φ_j,D)←check(Π'_Xj,Π'_Xj_Yj);
8.      END FOR;
9.    END FOR;
10. ELSE //函数依赖个数小于节点个数
11.   LoadAllocateByEC(M,Σ);
12.   FOR EACH j∈[1,m] DO
13.     FOR EACH S_k∈L(j) DO
14.       散列{Π_Xj,Π_Xj_Yj}对应键值对到 S_k;
15.       D''_k←prune(D'_k);
16.       Viot^Π(φ_j,D)← check(D''_k);
17.     END FOR;
18.   END FOR;
19. RETURN Viot^Π(φ_1,D),…,Viot^Π(φ_m,D).
```

算法 5 的输入为待检测函数依赖集合 Σ、分布式数据 D 以及处理机集合 M，输出为违反函数依赖的元组集合. 算法首先在各个节点对局部数据进行预处理，然后根据函数依赖个数与节点个数的大小情况分别处理. 如果函数依赖个数大于等于节点个数，则调用 *LoadAllocateByFD*() 进行任务分配，然后在每个处理机上根据当前处理机任务分配的情况对待检测函数依赖的等价类生成树进行剪枝，剪枝后检测冲突元组，最终得到所有违反函数依赖的元组；否则调用 *LoadAllocateByEC*() 进行任务分配，然后对每个函数依赖，散列所有局部划分对应键值对到执行节点，在执行节点对散列后的键值对合并，进一步对等价类生成树剪枝，最后进行不一致性检测，得到违反函数依赖的元组集合.

5 实验结果与分析

本文使用真实数据集和人工生成的数据集对本文提出的算法进行了实验验证，测试了算法基于节点规模、数据规模和待检测函数依赖规模的扩展性.

5.1 实验设置

(1) 实验环境

本实验中使用了通过局域网连接构成的由 10 台服务器组成的集群，每一台服务器配置如下：CPU 为英特尔 Xeon 2，16 GB 内存，Ubuntu 10.4 操作系统. 算法由 Java 编写，运行于 Hadoop 及 Hama 并行平台.

(2) 实验数据

实验过程中使用两种数据集，其中一个为 Bureau of Transportation Statistics 提供的数据集①，是一个关于航班信息的真实数据集，简称“AOTS”，包含了 64 个属性，如 FlightDate、Carrier、AirlineID、FlightNum、OriginCityName、DestCityName、TailNum 等. 数据集包含了 15 亿条元组，大小为 30 GB. 本文使用该数据集生成一个包含 8000 万条元组的数据库实例 $aots_8$，一个包含 1.2 亿条元组的数据库实例 $aots_{12}$. 另外一种数据集是一个人工生成的文中前面提到的 EMP 表的数据集，简称“EMP”，包含了 2 亿条元组. 利用数据集 EMP 生成一个包含 8000 万条元组的数据库实例 emp_8，一个包含 1.2 亿条元组的数据库实例 emp_{12}.

(3) 函数依赖

对于真实数据集 AOTS，找出一组反映真实约束关系的函数依赖，函数依赖个数为 10 个，每个包含了 2～6 个属性，为验证算法的效果，对部分数据人为加入了噪声. 对于数据集 EMP，定义了 10 个函数依赖，为了验证实验效果，在生成数据集时增加了部分噪声.

① Bureau of Transportation Statistics. http://apps.bts.gov/xml/ontimesummarystatistics/src/index.xml, 2015, 2, 5

5.2　实验结果与分析

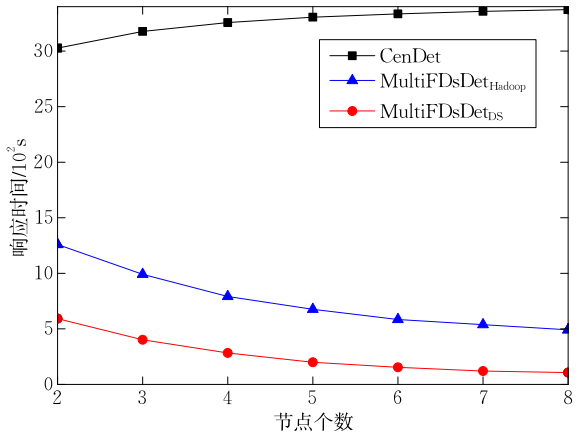
本文设计了 3 组实验分别对集中式函数依赖冲突检测方法 CenDet、基于 Hadoop 的 naïve 方法 $\text{MultiFDsDet}_{\text{Hadoop}}$ 以及本文提出的多个函数依赖冲突检测方法 MultiFDsDet_{DS} 进行验证. 为验证方法的节点规模扩展性、数据规模扩展性以及函数依赖规模扩展性，对节点规模($|S|$)、数据集规模($|D|$)以及函数依赖规模($|\Sigma|$)在不同情况下进行验证.

首先评估以上 3 种检测方法基于节点的扩展性，对每一个数据集选取 4 个函数依赖进行检测.

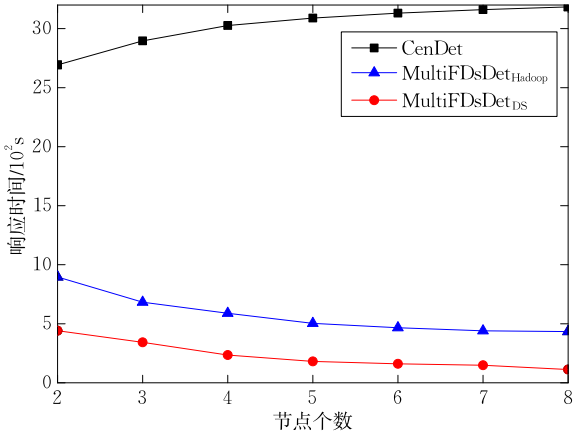
实验 1.　改变数据的分片个数. 为验证算法节点规模的扩展性，在数据集规模不变的情况下逐渐增加集群节点规模，基于数据集 $aots_8$ 和 emp_8，对方法的响应时间进行测试. 图 7(a)和图 7(b)反映了方法 CenDet、$\text{MultiFDsDet}_{\text{Hadoop}}$ 和 $\text{MultiFDsDet}_{\text{DS}}$ 在不同节点个数下响应时间情况. 不难看出，在数据规模固定的情况下，随着节点规模的增加，方法 $\text{MultiFDsDet}_{\text{Hadoop}}$ 和 $\text{MultiFDsDet}_{\text{DS}}$ 响应时间逐渐减少. 而集中式方法 CetDet 的响应时间则呈增加的趋势，原因是数据规模的增加导致数据迁移量增大，数据迁移耗时增加导致总的响应时间增大. 而在同等条件下，方法 $\text{MultiFDsDet}_{\text{DS}}$ 的响应时间明显小于 $\text{MultiFDsDet}_{\text{Hadoop}}$，说明与 $\text{MultiFDsDet}_{\text{Hadoop}}$ 相比，方法 $\text{MultiFDsDet}_{\text{DS}}$ 的节点扩展性更好，检测效率更高.

其次评估 3 种检测方法基于数据的扩展性，对每一个数据集选取 4 个函数依赖进行检测.

实验 2.　改变数据的规模. 为评价方法的数据规模扩展性，在保持节点规模(8 个)固定的情况下对数据规模进行扩展，从 2000 万条元组开始，每次增加 2000 万条，直到 12 000 万条，基于 $aots_{12}$ 和 emp_{12} 数据集分别进行验证. 图 8(a)和图 8(b)反映了方法 CenDet、$\text{MultiFDsDet}_{\text{Hadoop}}$ 和 $\text{MultiFDsDet}_{\text{DS}}$ 在数据规模扩展时响应时间的变化. 从实验结果不难看出，随着数据规模的扩大，各方法的响应时间呈增加趋势，而分布式方法 $\text{MultiFDsDet}_{\text{Hadoop}}$ 和

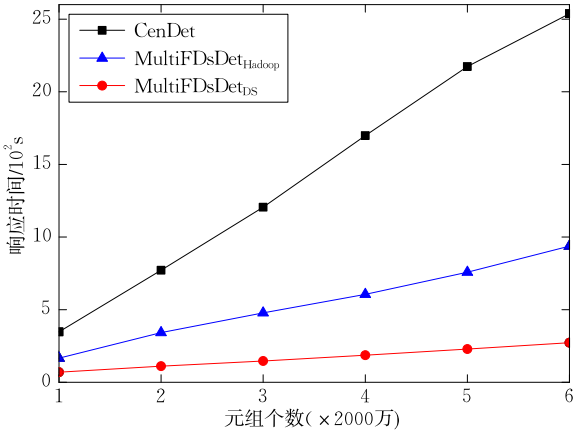


(a) 节点的扩展性($aots_8$)

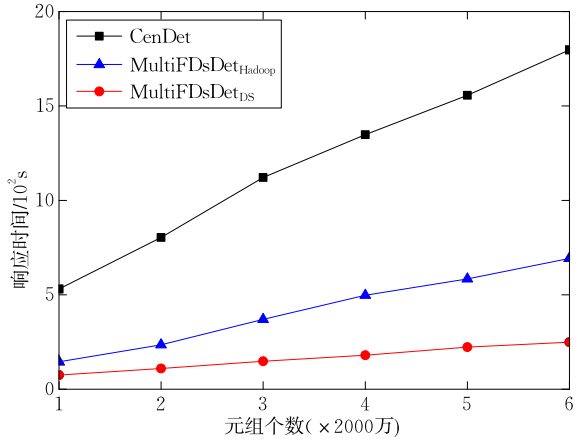


(b) 节点的扩展性(emp_8)

图 7　节点的扩展性



(a) 数据的扩展性($aots_{12}$)



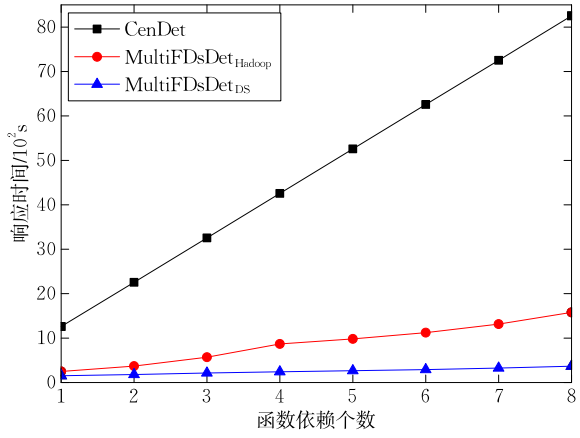
(b) 数据的扩展性(emp_{12})

图 8　数据的扩展性

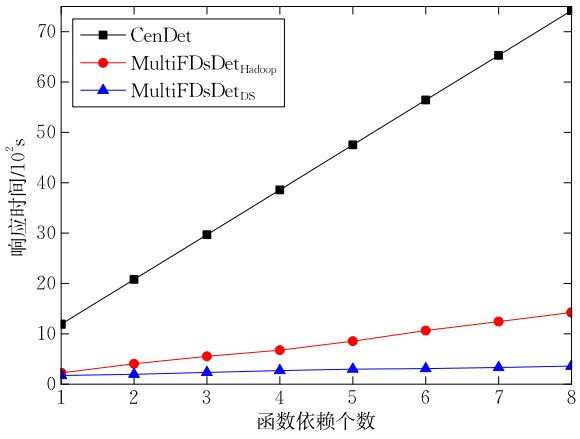
$MultiFDsDet_{DS}$比集中式方法 CenDet 的响应时间少很多，说明方法 $MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$的检测效率明显高于集中式方法 CenDet. 而且从图 8(a)和图 8(b)可以看出，在不同数据集的情况下，随着数据规模的不断增大，分布式方法 $MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$ 与 CetDet 相比，在响应时间方面优势更为明显. 而方法 $MultiFDsDet_{DS}$与 $MultiFDsDet_{Hadoop}$ 相比，在检测效率方面有着明显的提升，而且随着数据规模的增加，这种优势也更加突出，由此可见，本文提出的 $MultiFDsDet_{DS}$方法在数据扩展性方面明显优于分布式方法 $MultiFDsDet_{Hadoop}$和集中式方法 CenDet.

最后评估 3 种检测方法基于函数依赖个数的扩展性，集群的节点个数$|S|$为 8.

实验 3. 改变待检测函数依赖的个数. 为评价方法在不同数据规模情况下的扩展性，本文在数据规模固定的情况下，增加待检测函数依赖个数$|\Sigma|$从 1 个到 8 个，分别基于数据集 $aots_{12}$ 和 emp_{12}，对方法的响应时间进行测试. 图 9(a)和图 9(b)反映了



(a) 函数依赖规模$|\Sigma|$的扩展性($aots_{12}$)



(b) 函数依赖规模$|\Sigma|$的扩展性(emp_{12})

图 9 函数依赖规模$|\Sigma|$的扩展性

方法 CenDet、$MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$ 在函数依赖规模扩展时的响应时间情况. 从图 9(a)和图 9(b)不难看出，随着函数依赖规模的扩展，各方法的响应时间呈增加趋势. 在函数依赖个数相同的情况下，分布式并行方法 $MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$比集中式方法 CenDet 的检测耗时少很多，说明方法 $MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$在同等条件下的检测效率明显高于集中式方法 CenDet. 在不同数据集的情况下，随着函数依赖个数的不断增大，分布式方法 $MultiFDsDet_{Hadoop}$ 和 $MultiFDsDet_{DS}$与 CetDet 相比，在响应时间方面优势更为明显. 而方法 $MultiFDsDet_{DS}$ 与 $MultiFDsDet_{Hadoop}$相比，检测效率更高，而且随着函数依赖个数的增加，这种优势也更加明显. 从本组实验可以看出，本文提出的 $MultiFDsDet_{DS}$方法在函数依赖规模扩展性方面明显优于分布式方法 $MultiFDsDet_{Hadoop}$ 和集中式方法 CenDet.

前面对本文提出的方法在效率方面基于真实和人工数据集进行了验证. 由于本文提出的方法在算法设计时保证所有可能存在相互冲突的元组都被重分布到同一个执行节点进行检测，而且本文基于等价类进行函数依赖冲突检测，论文前面部分通过关于等价类的相关证明以及函数依赖冲突等价类生成树的构建使得函数依赖存在不一致性的元组可以全部被检测出来，因此在实验部分没有必要也不再对本文提出的方法的正确性进行验证.

6 结 论

本文研究了分布式大数据背景下多个函数依赖冲突检测的问题. 针对分布式环境多函数依赖冲突检测存在的问题，提出了一种基于等价类的分布式并行多函数依赖冲突检测算法. 论文主要关注的是分布式大数据背景下多个函数依赖冲突检测的响应时间，给出了分布式大数据背景下多个函数依赖冲突检测的响应时间代价模型，分析了影响多个函数依赖冲突检测响应时间的因素. 为提高分布式环境多个函数依赖冲突检测的并行度和检测效率，将检测任务分配问题划归为整数规划问题，并给出了多函数依赖冲突检测任务分配的近似最优算法. 针对集群规模和待检测函数依赖个数的不同情况，给出两种检测任务分配策略. 为避免检测过程中负载不均衡的情况，在检测过程中动态均衡负载，为减少负载迁移量，将该问题划归为二次规划问题，并采用拉

格朗日算子法得到最优解.基于真实数据集和人工数据集对论文提出的多函数依赖冲突检测算法进行了验证,实验结果表明算法在数据规模、节点个数以及函数依赖个数方面扩展性良好,而且在提高检测效率方面优势明显.本文主要研究的是水平切分的数据分布式环境下多函数依赖冲突检测问题,下一步考虑数据垂直切分的情况以及其他完整性约束在分布式环境下的不一致性检测问题.

参 考 文 献

[1] Armstrong W W. Dependency structures of data base relationships//Proceedings of the International Conference on Intelligent Information. Amsterdam, North-Holland, 1974: 580-583

[2] Abiteboul S, Hull R, Vianu V. Foundations of Databases. Menlo Park, USA: Addison-Wesley, 1995

[3] Garey M, Johnson D. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, USA: W. H. Freeman and Company, 1979

[4] Fan W, Geerts F, Ma S, Muller H. Detecting inconsistencies in distributed data//Proceedings of the IEEE 26th International Conference on Data Engineering. Long Beach, USA, 2010: 64-75

[5] Fan W, Geerts F, Jia X, Kementsietsidis A. Conditional functional dependencies for capturing data inconsistencies. ACM Transactions on Database Systems, 2008, 33(2): 1-48

[6] Fan W, Li J, Tang N, Yu W. Incremental detection of inconsistencies in distributed data//Proceedings of the IEEE 28th International Conference on Data Engineering. Washington, USA, 2012: 318-329

[7] Huyn N. Maintaining global integrity constraints in distributed databases. Constraints, 1997, 2(3): 377-399

[8] Chhabra P, Scott C, Kolaczyk E D, Crovella M. Distributed spatial anomaly detection//Proceedings of the International Conference on Computer Communications. Phoenix, USA, 2008: 1705-1713

[9] Gupta A, Widom J. Local verification of global integrity constraints in distributed databases//Proceedings of the ACM Special Interest Group on Management of Data. New York, USA, 1993: 49-58

[10] Dong X, Naumann F. Data fusion—Resolving data conflicts for integration. Proceedings of the VLDB Endowment, 2009, 2(2): 1654-1655

[11] Agrawal S, Deb S, Naidu K V M, Rastogi R. Efficient detection of distributed constraint violations//Proceedings of the 23rd IEEE International Conference on Data Engineering. Istanbul, Turkey, 2007: 1320-1324

[12] Koufakou A, Georgiopoulos M. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. Data Mining and Knowledge Discovery, 2010, 20(2): 259-289

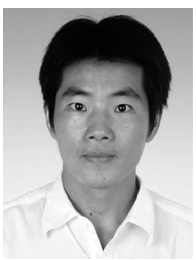
[13] Bohannon P, Fan W, Flaster M. A cost-based model and effective heuristic for repairing constraints by value modification //Proceedings of the ACM Special Interest Group on Management of Data. New York, USA, 2005: 143-154

[14] Lopatenko A, Bravo L. Efficient approximation algorithms for repairing inconsistent databases//Proceedings of the 23rd IEEE International Conference on Data Engineering. Istanbul, Turkey, 2007: 216-225

[15] Huhtala Y, Karkkainen J, Porkka P, Toivonen H. Tane: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal, 1999, 42(2): 100-111

[16] Novelli N, Cicchetti R. FUN: An efficient algorithm for mining functional and embedded dependencies//Proceedings of the 8th International Conference on Digital Telecommunications. London, UK, 2001: 189-203

[17] Hu Y F, Blake R J, Emerson D R. An optimal migration algorithm for dynamic load balancing. Concurrency and Computation Practice and Experience, 1998, 10(6): 467-483



LI Wei-Bang, born in 1979, Ph. D. candidate. His research interests include data quality, big data and massive data computation.

LI Zhan-Huai, born in 1961, Ph. D., professor. His main research interests include database theory and technology and data management.

JIANG Tao, born in 1983, Ph. D. candidate. His main research interests include biological information mining and data management.

Background

One of the most important issues of data management is data quality, and an important part of data quality is inconsistency detection. It is easy to detect dependency violations in centralized databases. However, it is far more challenging

to check dependency violations in distributed databases, especially with big data.

Nowadays the study of detecting violations of integrity constraints for the distributed environment is still relatively small. W. Fan proposed several algorithms for detecting violations of Conditional Functional Dependencies in Distributed Data. These algorithms are based on pattern table and not fit for Functional Dependencies and big data. This paper proposes a novel equivalence class based multiple functional dependency violations detection approach in distributed big data. The main contributions of this article are as follows: (1) Proposing a cost model of violations detection in distributed big data, and propose the approximate optimal parallel detection algorithm. (2) Introducing equivalence classes into the violations detection of multiple Functional Dependencies and propose optimization strategy based on equivalence classes. (3) Balancing load dynamically in the detection process and classify the question as quadratic programming problems, and obtain the optimal solution by Lagrange Multiplier. (4) Experiments on real-world and generated datasets demonstrate that our approach is more effective in efficiency and with good scalability on the number of nodes, on the size of datasets and on the number of functional dependencies.

This research is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2012CB316203, the National Natural Science Foundation of China under Grant Nos. 61502390, 61472321, 61332006, 61272121, and the National High Technology Research and Development Program (863 Program) of China under Grant No. 2015AA015307.