软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(4):731−752 [doi: 10.13328/j.cnki.jos.004569]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

大数据分析的分布式 MOLAP 技术*

宋 杰[1], 郭朝鹏[1], 王 智[1], 张一川[1], 于 戈[2], Jean-Marc PIERSON[3]

[1](东北大学 软件学院,辽宁 沈阳 110819)

[2](东北大学 信息科学与工程学院,辽宁 沈阳 110819)

[3](Laboratoire IRIT, Université Paul Sabatier, Toulouse F-31062, France)

通讯作者: 宋杰, E-mail: songjie@mail.neu.edu.cn, http://faculty.neu.edu.cn/songjie/

摘 要: 大数据的规模效应给数据存储、管理以及数据分析带来了极大的挑战,学界和业界广泛采用分布式文件系统和 MapReduce 编程模型来应对这一挑战.提出了大数据环境中一种基于 Hadoop 分布式文件系统(HDFS)和 MapReduce 编程模型的分布式 MOLAP 技术,称为 DOLAP(distributed OLAP).DOLAP 采用一种特殊的多维模型完成维和度量的映射;采用维编码和遍历算法实现维层次上的上卷下钻操作;采用数据分块和线性化算法将维和度量保存在分布式文件系统中;采用数据块选择算法优化 OLAP 的性能;采用 MapReduce 编程模型实现 OLAP 操作.描述了 DOLAP 在科学数据分析的应用案例,并与主流的非关系数据库系统进行性能对比.实验结果表明,尽管数据装载性能略显不足,但 DOLAP 的性能要优于基于 HBase,Hive,HadoopDB,OLAP4Cloud 等主流非关系数据库系统实现的 OLAP 性能.

关键词: 大数据;多维数据模型;OLAP;MapReduce

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 宋杰,郭朝鹏,王智,张一川,于戈,Pierson JM.大数据分析的分布式 MOLAP 技术.软件学报,2014,25(4):731−752. http://www.jos.org.cn/1000-9825/4569.htm

英文引用格式: Song J, Guo CP, Wang Z, Zhang YC, Yu G, Pierson JM. Distributed MOLAP technique for big data analysis. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):731−752 (in Chinese). http://www.jos.org.cn/1000-9825/4569.htm

Distributed MOLAP Technique for Big Data Analysis

SONG Jie[1], GUO Chao-Peng[1], WANG Zhi[1], ZHANG Yi-Chuan[1], YU Ge[2], Jean-Marc PIERSON[3]

[1](Software College, Northeastern University, Shenyang 110819, China)

[2](School of Information and Engineering, Northeastern University, Shenyang 110819, China)

[3](Laboratoire IRIT, Université Paul Sabatier, Toulouse F-31062, France)

Corresponding author: SONG Jie, E-mail: songjie@mail.neu.edu.cn, http://faculty.neu.edu.cn/songjie/

Abstract: To address the new challenges that big data has brought on data storage, management and analysis, distributed file systems and MapReduce programming model have been widely adopted in both industry and academia. This paper proposes a distributed MOLAP technique, named DOLAP (distributed OLAP), based on Hadoop distributed file system (HDFS) and MapReduce program model. DOLAP adopts the specified multidimensional model to map the dimensions and the measures. It comprises the dimension coding and traverse algorithm to achieve the roll up operation on dimension hierarchy, the partition and linearization algorithm to store dimensions and measures, the chunk selection strategy to optimize OLAP performance, and MapReduce to execute OLAP. In addition, the paper describes the application case of the scientific data analysis and compares DOLAP performance with other dominate non-relational data

* 基金项目: 国家自然科学基金(61202088); 中央高校基本科研业务费专项资金(N120817001); 中国博士后科学基金面上项目(2013M540232); 教育部博士点基金(20120042110028); 教育部-英特尔信息技术专项科研基金(MOE-INTEL-2012-06)

收稿时间: 2013-10-15; 修改时间: 2013-12-18; 定稿时间: 2014-01-27

management systems. Experimental results show that huge dominance in OLAP performance of the DOLAP technique over an acceptable performance lose in data loading.

Key words: big data; multi-dimensional data model; OLAP; MapReduce

近年来,随着大数据时代的到来以及互联网、传感器和科学数据分析等领域的快速发展,数据量近乎每年在成倍地增长[1].无论是在科学领域(生物学、地理学、天文学、气象学等),还是在工程领域(网络数据分析、市场数据分析等),都面临着数据雪崩的问题[2],大数据的规模效应给数据存储、管理以及数据分析带来了极大的挑战[3,4].OLAP(on-line analytical processing)联机分析处理是共享多维信息的、针对特定问题的联机数据访问和分析的快速软件技术[5],OLAP 按照其实现方式不同,可以分为 3 种类型,分别是 ROLAP,MOLAP 和 HOLAP[6].其中,ROLAP 采用关系表存储维信息和事实数据;MOLAP 则采用多维数据结构存储维信息和事实数据;而 HOLAP 称其为混合 OLAP,该方法结合了 ROLAP 和 MOLAP 技术[7].无论是何种 OLAP,都需要存储和计算平台的支持,尤其是在大数据环境下.

为了解决大数据所带来的诸多挑战,学界和业界涌现出许多新技术,如分布式文件系统[8]、NoSQL 数据库系统[9]、MapReduce 编程模型[10]以及相关的优化方法,这些技术都被广泛地运用到大数据分析中.MapReduce 编程模型是广为人知的可扩展、灵活且高效的分布式编程框架.Hadoop 是 MapReduce 的开源实现,可对海量数据进行可靠、高效、可扩展的并行处理.基于 Hadoop[11]的实现,涌现出大量的分布式数据管理系统,并广泛地运用在大数据管理和分析领域,如 Hive[12],HBase[13],HadoopDB[14]等.一方面,尽管这些数据管理系统均可支持 OALP,但其性能往往不尽如人意.例如,基于 HBase 的 OLAP 引擎 OLAP4cloud[15]框架属于一种基于云计算技术的 OLAP 实现,它采用列存储数据存储结构以及索引等技术优化 OLAP 的性能.但是,OLAP4cloud 并不提供维信息的管理,也无法直接支持上卷下钻操作,因此,OLAP4cloud 仅限于支持对度量数据的查询和简单的聚集操作.另一方面,这些数据库系统均未针对 OLAP 进行特殊的优化,我们之前的研究[16]表明,连接操作在 ROLAP 中是非常频繁且相当耗时的操作,当数据量或维数量增加时,连接操作会成为 OLAP 的瓶颈.MOLAP 可以避免数据集的连接操作,因此在性能方面有着天生的优势,但 MOLAP 需要集中式存储多维数据模型,且耗费大量空间,如何基于分布式文件系统和 MapReduce 模型实现 MOLAP 模型则是一个难题.据我们所知,在大数据分析领域,尚未有关于分布式 MOLAP 技术的权威报道,也鲜有成熟的基于 MapReduce 的 MOLAP 系统,该问题亟待解决.

本文研究大数据环境下基于 MapReduce 的分布式 MOLAP 技术,称为 DOLAP(distributed OLAP).DOLAP 采用一种特殊的多维模型完成维和度量的映射;采用维编码和遍历算法实现维层次上的上卷下钻操作;采用数据分块和线性化算法将维和度量保存在分布式文件系统中;采用数据块选择算法优化 OLAP 的性能;采用 MapReduce 编程模型实现 OLAP 操作.在 DOLAP 技术的基础上,我们基于 Hadoop 实现了一个 OLAP 系统 HaoLap (hadoop OLAP),设计了一系列测试用例,将 HaoLap 与 Hive,HadoopDB,HBase 和 oalp4cloud 等进行性能测试和比较.实验结果表明,HaoLap 的数据装载性能不具优势,但其 OLAP 性能优势明显,且性能与原始数据集规模以及查询复杂程度无关,尤其适合高维数据立方的 OLAP 操作.HaoLap 仅依赖分布式文件系统存储数据,不引入额外的存储代价,数据立方通过计算获得,对于立方的每个维,仅存储维级别名称和每个维级别中维值的个数,不同于传统 MOLAP 系统耗费大量空间存储数据立方.由于篇幅原因,本文略去了 HaoLap 系统的实现细节.

本文第 1 节介绍相关工作.第 2 节介绍维编码及事实存储,其中包括简化的多维数据模型的定义、维编码和维遍历算法、数据立方分块、存储以及寻址算法等,并给出数据立方建模和存储的应用案例 OceanCube.第 3 节重点介绍基于 MapReduce 的 OLAP 算法.第 4 节首先采用真实的科学数据集 OceanCube,通过 3 组实验分别测试和比较 DOLAP 在数值型数据上的数据装载、切块操作、上卷操作和存储代价,随后采用 SSB 基准测试比较 DOLAP 在枚举型数据上的 OLAP 性能,并总结实验结论.第 5 节总结全文并提出进一步工作.

1 相关工作

目前已有很多关于大数据或云计算环境下的 OLAP 优化方法研究,本节主要从以下两个方面介绍相关工

作:大数据环境中的 OLAP 优化技术和分布式的 OLAP 系统.大数据环境中,常用的 OLAP 优化方法有以下两种:利用预计算和浓缩数据立方的结果优化 OLAP 性能[17]和通过优化存储结构和算法来优化 OLAP 性能,其中,后者与本文最为相关.文献[18]提出了 OLAP 查询中的 SPAJG-OLAP 子集,在存储、查询、数据分布、网络传输和分布式缓存等方面研究海量数据大规模并行处理框架的优化策略和实现技术,实验证明,效果良好.该研究基于并行数据库技术优化 ROLAP 性能,通过对 OLAP 查询以及存储的优化达到加速 OLAP 的目的.文献[19]指出:在 Web 应用中,需要同时提供对海量数据的事务操作和决策分析,并介绍一种能够同时有效支持 OLTP 和 OLAP 的数据存储系统.通过建立索引、数据分块、预计算等方式,有效地提高了 OLAP 的性能.本文同样是通过优化查询以及数据存储来提高 OLAP 的执行效率,但不同的是,本文研究的优化方案基于 MapReduce.同时,针对 MapReduce 连接操作的低效性,本文研究提出了 DOLAP 技术,大大提高了 OLAP 的执行效率.

文献[20]指出,传统的 OLAP 分析无法很好地适用于大数据分析.该文根据短消息数据的特点,设计了一种基于 Hadoop 的有效存储格式,并使用 MapReduce 实现了 OLAP 操作,更好地适应了 SMS(short message service)业务中对短消息进行数据分析的需求.该文利用了特殊的数据结构来优化 OLAP 操作的性能,但该研究的适用面较窄.文献[21]基于 MapReduce,通过数据的筛选策略减少了数据在网络中不必要的传输,从而优化了复杂云环境中的 OLAP 性能.该研究重点关注云环境的复杂性以及网络延迟等因素对 OLAP 性能的影响,是大数据环境下 ROLAP 的一种优化策略,其优化方法与本文研究存在本质区别.

就分布式的 OLAP 系统而言,一些基于 Hadoop 的数据库系统,例如 Hive,HadoopDB,HBase,MongoDB,OLAP4cloud 等都支持 OLAP 分析.Hive 是一个基于 Hadoop 的数据仓库系统,为数据分析人员提供了类 SQL 接口,支持大数据分析.HadoopDB 将 MapReduce 和关系数据库技术结合起来,以管理和分析大数据;HBase 则是面向列存储的开源数据库系统.这些数据库将海量数据存储在分布式文件系统中,通过 MapReduce 完成上卷下钻等操作,属于分布式的 ROLAP 技术,其中,维表和事实表的连接运算是一个性能瓶颈.OLAP4cloud 是基于 HBase 的 OLAP 引擎,它将维表直接压缩到事实表中,并提供一种特殊的索引来加快寻址,采用数据立方预计算方法,属于一种近似 MOLAP 实现.与分布式的 MOLAP 技术最为相关的是文献[22],该文尝试使用多维数组存储海量数据,并将建立的存储模型运用到了基于 Hadoop 的数据分析工具 Pig[23]中,且通过实验证明了该存储模型在占用大量存储的同时提高了 OLAP 分析性能.为减少存储开销,本文使用多维数组存储维信息而非事实数据;除此之外,本文还提出了维相关的遍历算法以及数据筛选算法.

综上所述,尽管学界注意到大数据对 OLAP 分析提出的挑战,且已有部分研究分别从模型、存储、算法和预计算角度对传统 OLAP 进行了优化,但尚未有权威的大数据环境下分布式的 MOLAP 技术的研究报告.与此同时,Hadoop 也逐渐成为开源领域大数据分析的主流技术,但是基于 Hadoop 的分布式的 MOLAP 系统研究仍然尚未成熟.

2 维编码及事实存储

2.1 数据模型

OLAP 采用的多维数据模型包括维和事实两部分,其关键操作是找到维和事实的映射关系.ROLAP 采用关系数据库以及星形模式或雪花模式,将维信息和事实数据分别存储于关系数据库表中,并使用外键完成维信息和事实数据的映射.但是 ROLAP 涉及到大量的连接操作,性能较低.MOLAP 采用多维数组存储维和事实,通过对维进行编码和对事实数据直接寻址的方式获得其映射关系,从而避免了连接操作的开销.但 MOLAP 需要以一种集中的方式维护维和事实的映射,由于一个维可以包含多个层次,每个层次可以包含若干个级别,维和事实又是一对多的关系,所以维模型具有复杂性.为了避免额外的存储和维护代价,DOLAP 首先对维进行了简化,以适应分布式环境,同时降低 OLAP 算法的复杂程度.本节重新定义维和事实数据,同时也定义了其他相关术语.

定义 1(维(dimension)). 在多维数据模型中,维将所有的数据项分类至一个无重叠的数据结构中,并且提供数据项的筛选、组织和标识方法.本文研究对维的定义进行了简化,简化后的维基于多维模型的维定义,并遵循以下 3 个约束:设 d 为维,则,

1) d 有且仅有 1 个维层次;
2) d 是 m 个维级别所组成的集合,记为$\{l_1,l_2,...,l_m\}$.设 $l_i(i\in[1,m])$为任意一个维级别,则 l_i 仅包含 1 个维属性,且包含 n_i 个维值;
3) 将 d 视作由各级别的维属性取值所组成的树形结构(维值树),则同一级别的兄弟节点包含有相同数目的子节点.

基于上述假设,维 d 由以下两部分组成:

1) 维模式(dimension schema)包含:① m 个维级别的有限集 $L(d)$,且这些维级别仅包含一个概念层次(concept hierarchy);② 在集合 $L(d)$上存在一个全序关系$\succ_d$,上卷(roll-up)操作是指沿着概念层次向上攀升.如果有关系 $l_j\succ_d l_i(i,j\in[1,m],i<j)$成立,那么认为 l_j 可以上卷到 l_i;
2) 维实例(dimension instance)包含:① 函数 m_d 可以获得维级别的维属性取值集合,对于 l_i 而言,仅包含 1 个维属性,该维属性包含 n_i 个取值,$|m_d(l_i)|=n_i$;② 上卷函数 $\rho_d^{l_j\to l_i}=m_d(l_j)\to m_d(l_i)$, 对每个关系 $l_j\succ_d l_i$: $\forall v^j\in m_d(l_j),\exists v^i\in m_d(l_i)$满足 $\rho_d^{l_j\to l_i}(v^j)=v^i$, 且$\forall v^i\in m_d(l_i),\exists v^j\in m_d(l_j)$满足 $\rho_d^{l_j\to l_i}(v^j)=v^i$;③ $\forall v_x^i,v_y^i\in m_d(l_i)$, $|\{v_x^{i+1}\mid\rho_d^{l_{i+1}\to l_i}(v_x^{i+1})=v_x^i\}|=|\{v_y^{i+1}\mid\rho_d^{l_{i+1}\to l_i}(v_y^{i+1})=v_y^i\}|$.

为简便起见,如果不加说明,本文中"维"均指符合定义 1 的维.

定义 2(度量(measure)). 度量 u 是一个独立变量,它们参照每个维的某一维值,并作为 OLAP 的分析对象.度量的粒度是度量参照的维值所在的维级别,最细粒度的度量参照每一维 d 中的最低维级别的某一维值.设 u 参照维集合 $D=\{d_1,d_2,\ldots,d_n\},\forall d\in D$,即集合 D 可以确定度量 u,记作 $D\to u$,则满足:$D\to u\Leftrightarrow\exists!v\in m_d(l_m^d)\wedge v\to u$($d$ 有 m 个维级别),其中,$v\to u$ 是指维值 v 可以确定度量 u.

定义 3(单元格(cell)). 在逻辑视图中,单元格是由若干不同的度量组成的原子单元,这些度量都参照相同的维值.对于维集合 D 而言,单元格可以表示为度量的集合,记作$\{u|D\to u\}$.

定义 4(数据立方(cube)). 根据定义 1~定义 3,数据立方是 OLAP 中的多维数据结构,简称立方.数据立方的维符合定义 1,且由若干单元格组成.

定义 5(块(chunk)). 块是数据立方的逻辑划分,一个数据立方可以根据维的取值分成多个块.

图 1 是由 3 个维(x,y,z)所组成的立方,图中较小的方格代表单元格,较大的方格代表块.在实际操作中,块中有可能包含一些空的单元格,即,该单元格中没有任何度量.在实际应用中,为了减少立方占用物理空间的大小,若单元格内没有任何度量,则在该块文件中不保存该单元格的记录.

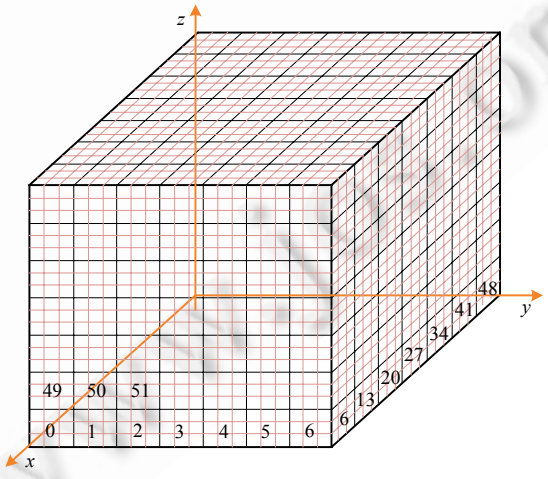


Fig.1 Example of data cube

图 1 数据立方示例

2.2 维算法

在 OLAP 操作中,对维的操作是非常频繁的.维的编码和遍历算法是 MOLAP 的关键技术,本节将对 DOLAP 技术中维的编码和遍历算法进行阐述.

2.2.1 维编码算法

维编码方法主要包括二进制编码和十进制编码.二进制编码也称作位图编码,通过编码的拼接可以包含维的级别信息,通过编码的移位实现维的遍历,但是二进制编码会造成很大程度上的稀疏[24];十进制编码是对每个维级别的维值依次使用十进制数编码,但是无法直接获得编码和维值的映射.在大数据环境中,为了避免稀疏,DOLAP 采用十进制的编码方法,同时提出了维的遍历算法来计算编码和维值间的映射关系.设 l 是维 d 中的某个维级别,对$\forall x\in[1,|m_d(l)|]$,$v_x\in m_d(l)$,v_x 的编码为 $code(v_x)$,则 $code(v_x)=x-1$.该编码方法如算法 1 所示.

算法 1. 维编码算法.

Input: *Dimension d*: A target dimension;

Function: *DimensionCoding*.

1. **FOR** i=1 **TO** $|L(d)|$;
2. **FOR** j=0 **TO** $|m_d(l_i)|-1$;
3. *Dimension value* of $v_j^i \in m_d(l_i)$
4. $v_j^i.code = j$;
5. **END FOR**
6. **END FOR**

在实际应用环境,绝大部分维是数值型的,例如高度、经度、价格、流水号等.数值型的维可以按照其值域进行划分,不同的划分步长可以确定不同的维级别,因此,数值型的维可以很容易地满足定义 1 的约束条件.但是还存在一部分非数值型的枚举型的维,例如日期、城市、部门等.为使枚举型维符合定义 1,可以使用一些空值填补维值树,使同一级别的兄弟节点包含有相同数目的子节点.图 2 展示了日期维的编码结果.在“月”级别上,每个月的天数是不同的,为了满足定义 1,设每个月有 31 天,所以在图 2 中的 2 月插入了“29 日”、“30 日”和“31 日”这 3 个空值.

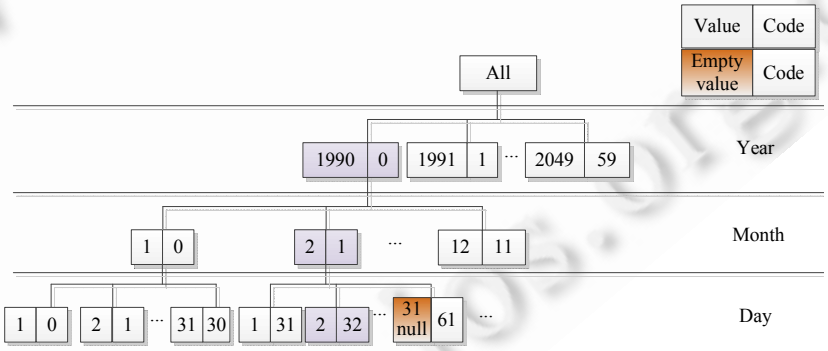


Fig.2 Example of date dimension coding

图 2 日期维编码示例

对于实际应用中更为复杂的维,采取化简、划分维层次的方法使其形成维值树,使用空值填补维值树的方法使其满足定义 1 的约束条件.例如,针对 TCP-H[25]数据集中的维模式,处理方法如下:① 通过取舍和合并的方法,化简 TCP-H 的雪花模式为星型模式,其结果为 SSB[26]数据集中的维模式;② 针对 *SUPPLIER* 维表,使用区域属性(*Nation*,*Region*,*City*)作为划分维层次的依据,得到维层次 *Nation-Region-City*;③ 在维 *Nation-Region-City* 的维值树中添加空值,使其满足定义 1 的约束.

2.2.2 维遍历算法

DOLAP 的维可以看作是一棵特殊的单根树,记作 T^d,其中,*ALL* 是 T^d 的根节点,记为第 0 级别.每个维级别中的维值可以看作维值树中的节点,同时,每一个兄弟节点都有相同数目的子节点,如图 2 所示.OLAP 操作中涉及到大量对维值树 T^d 的遍历操作.例如,沿着 T^d 攀升(即上卷)或沿着 T^d 下降(即下钻).设有关系$\forall i\in[1,m-1], l_{i+1}\succ_d l_i$, $v^i\in m_d(l_i), v^{i+1}\in m_d(l_{i+1})$,那么 v^i 和 v^{i+1} 之间的上卷关系 $\rho_d^{l_{i+1}\to l_i}(v^{i+1})=v^i$ 是 OLAP 中的关键操作.编码机制将能够表征这种上卷关系,我们可以通过编码运算实现 T^d 中的上卷操作.本节首先引入维级别规模的概念,并通过对节点 v^i 和 v^{i+1} 的编码的运算获得关系 $\rho_d^{l_{i+1}\to l_i}(v^{i+1})=v^i$.至于从 v_i 到 v_{i+1} 的下钻操作,可以等同地视作从 v^{i+1} 至 v^i 的上卷操作,此处不再赘述.

定义 6(维级别规模(dimension level size)). 设 d 是一个维,由 m 个维级别组成,第 i 个维级别记作 $l_i(i\in[1, m-1])$,维级别规模记作$|l_i|$,则$|l_{i+1}|=|\{v^{i+1}|\forall v^i\in m_d(l_i), \rho_d^{l_{i+1}\to l_i}(v^{i+1})=v^i\}|$,其中,$\{v^{i+1}|\forall v^i\in m_d(l_i), \rho_d^{l_{i+1}\to l_i}(v^{i+1})=v^i\}$是指维级别中符合条件的维属性取值的集合,$|\{v^{i+1}|\forall v^i\in m_d(l_i), \rho_d^{l_{i+1}\to l_i}(v^{i+1})=v^i\}|$是指该集合的大小.

根据定义 6,维级别规模是指在维值树中上层维级别中任意节点子节点的个数.

设$\langle v^1,v^2,\ldots,v^i,v^{i+1},\ldots,v^m\rangle$是 T^d 中自上而下的一条分析路径,v^i 在其兄弟节点间的位置记为 $order(v^i)$(兄弟节点拥有共同的父节点,且其位置计数从 0 开始,从左至右),则编码与位置的关系可表述为公式(1).对于图 2 中的路径$\langle 1990^1,2^2,2^3\rangle$:$code(2^3)=32, order(2^3)=1; code(2^2)=1, order(2^2)=1; code(1990^1)=0, order(1990^1)=0$.

$$code(v^i)=(\ldots((0+order(v^1)\times|l_2|+order(v^2))\times|l_3|+order(v^3))\ldots)\times|l_i|+order(v^i) \tag{1}$$

同理,当给出 $code(v^i)$时,$order(v^1)$到 $order(v^i)$的值可通过公式(2)计算得到:

$$\begin{aligned}
&temp_i=code(v^i)\\
&order(v^i)=temp_i\%|l_i|, temp_{i-1}=\left\lfloor\frac{temp_i}{|l_i|}\right\rfloor\\
&order(v^{i-1})=temp_{i-1}\%|l_{i-1}|, temp_{i-2}=\left\lfloor\frac{temp_{i-1}}{|l_{i-1}|}\right\rfloor\\
&\ldots\\
&order(v^1)=temp_1\%|l_1|
\end{aligned} \tag{2}$$

联合公式(1)和公式(2),对于给定的 $code(v^i)$,可以计算出其对应的所有父节点的编码,即 $code(v^{i-1})$到 $code(v^1)$的值,从而可以进行上卷操作.例如图 2 中,在“天”维级别中,已知 $code(2^3)=32$,其路径为$\langle 1990^1,2^2,2^3\rangle$,根据公式(2),已知 $code(2^3)=32, |l_3|=31, |l_2|=12, |l_1|=50$,则 $order(2^2)=1, order(1990^1)=0$;再根据公式(1),可以计算得到 $code(2^2)=1$, $code(1990^1)=0$.

2.3 数据立方分块

将数据立方划分为块的目的是在 OLAP 过程中对数据进行筛选,从而优化 OLAP 的性能.同时,块也可以作为数据立方的存储单元.本节首先定义维、数据立方以及块的规模和容量,而后讨论数据立方的分块策略.

定义 7(维规模(dimension size)). 维规模是指最底层维级别的维值个数.设 d 是一个维,包含 m 个维级别,记作 $l_1,l_2,\ldots,l_m$.记$|d|$为维 d 的规模,则$|d|=|m_d(l_m)|=\prod_{i=1}^{m}|l_i|$.

定义 8(立方规模(cube size)和立方容量(cube capacity)). 立方规模是由组成立方的维的规模构成的元组,立方容量则是立方内包含的单元格的个数.设立方由 n 维组成,其中第 i 个维记作 $d_i(i\in[1,n])$.立方规模为$|cube|=\langle|d_1|,|d_2|,\ldots,|d_n|\rangle$,立方容量为$|cube|^n=\prod_{i=1}^{n}|d_i|$.

定义 9(块规模(chunk size)和块容量(chunk capacity)). 块规模是一个由该块包含的每个维最底层维级别上维值的个数构成的元组,块容量是块包含的单元格的个数.设块由 n 维组成,其中第 i 个维记作 $d_i(i\in[1,n])$,且

d_i 被划分为 p_i 份.记块的规模为|*chunk*|,则|*chunk*|=$\langle\lambda_1,\lambda_2,\ldots,\lambda_n\rangle$,其中,$\lambda_i=\left\lceil\frac{|d_i|}{p_i}\right\rceil$,块容量为 $|chunk|^n=\prod_{i=1}^{n}\lambda_i$.

DOLAP 的性能与|*chunk*|^{n}的取值密切相关,|*chunk*|^{n}取值越小,并行性越好,实际参与运算的单元格数量越少,但此时调度代价变高.如何折中地确定|*chunk*|^{n}的取值,将变得十分关键.借鉴文献[27],本文提出了一种通过查询条件及其出现的概率来确定块容量的方法,但我们难以穷举所有查询条件及其出现概率,所以该方法采用随机抽样,抽取一些查询条件及其出现概率.除此之外,|*chunk*|^n 的取值还应该考虑算法的运行环境.DOLAP 利用 MapReduce 来实现,所以|*chunk*|^{n}的取值还需考虑 MapReduce 的一些特性,例如文件寻址时间、数据处理时间等.表 1 列出了相关符号的定义,其中,λ_i 为变量,T 和 N_a 是计算结果,其他为已知常量.

Table 1 Definition of notations

表 1 相关符号定义

符号	含义描述
T	一个 OLAP 操作消耗的平均时间
n	多维数组维的个数
λ_i	维 d_i 上块规模
N_a	平均一个查询命中的块个数
v	Map 任务的处理速度(个/秒)
t_0	文件寻址时间
t_1	Map 任务消耗的额外时间
q	所有可能的查询条件个数
a_{ij}	查询条件 j 在维 d_i 上的取值个数
ξ_j	查询条件 j 出现的概率

平均一个查询命中的块个数 N_a 可以通过每个查询条件出现的概率得到,如公式(3)所示:

$$N_a=\sum_{j=1}^{q}\left(\prod_{i=1}^{n}\left\lceil\frac{a_{ij}}{\lambda_i}\right\rceil\xi_j\right) \tag{3}$$

考虑 MapReduce 相关的一些影响因素之后,可以得到一个 OLAP 操作消耗的平均时间,如公式(4)所示:

$$T=\left[\sum_{j=1}^{q}\left(\prod_{i=1}^{n}\left\lceil\frac{a_{ij}}{\lambda_i}\right\rceil\xi_j\right)\right]\left(t_0+t_1+\frac{\prod_{i=1}^{n}\lambda_i}{v}\right) \tag{4}$$

通过公式(4),可以计算出 T 取最小值时的λ_i,也就得到了块规模.每个维上的块规模均可通过上述方法计算得到,从而可以得到块容量的最佳值.由公式(4)可以看出,'⌈ ⌉'操作会导致块容量之和大于立方容量,所以块中会存在空的单元格.

2.4 数据存储

传统 MOLAP 数据立方的存储代价很大,尤其是在高维数据立方或每个维包含大量维值的情况下.事实上,传统 MOLAP 依赖访问内存中的多维数组来加快 OLAP 操作,这种方式在大数据环境中显然难以实现.DOLAP 的"多维数组"是通过计算得到,无需存储,因此数据立方的存储代价非常小.DOLAP 对维进行简化,可以保证在同一级别上维的编码是连续的十进制数,同时,每一个兄弟节点都有相同个数的子节点,所以维信息仅需存储每个维级别规模,极大地降低了维的存储代价.设维 d 由 m 个维级别组成,记作$\{l_i|i\in[1,m]\}$,则 d 的物理存储可以表示为该维级别和维级别规模的序偶所组成的集合$\{\langle l_i,|l_i|\rangle|i\in[1,m]\}$,其中,$l_i$ 表示该维级别的名称.DOLAP 实现系统可以使用 XML 文件存储维信息,并保存于集群主节点.

逻辑上,"立方和其单元格"或是"立方和其块"的数据结构均可以类比为"多维数组和其元素"的数据结构.物理上,块是立方的存储单元,将块内的单元格线性化(linearization)后,块可以作为独立的文件进行存储.为了便于寻址,块和单元格都需要支持线性化和反线性化(reverse-linearization)运算,且该运算与多维数组的线性化和反线性化运算是一致的.设存在一个 n 维数组,其维规模记作$\langle A_1,A_2,\ldots,A_n\rangle$,多维数组中的元素 X 在多维空间中的

坐标记作$(X_1,X_2,\ldots,X_n)$,其线性化后的坐标记作 $index(X)$,则其线性化方法如公式(5)所示,反线性化方法如公式(6)所示:

$$index(X)=(\ldots((X_n\times A_{n-1}+X_{n-1})\times A_{n-2}+\ldots+X_3)\times A_2+X_2)\times A_1+X_1 \tag{5}$$

$$\begin{aligned}&temp_1=index\\&X_1=temp_1\%A_1,temp_2=\left\lfloor\frac{temp_1}{A_1}\right\rfloor\\&X_2=temp_2\%A_2,temp_3=\left\lfloor\frac{temp_2}{A_2}\right\rfloor\\&\ldots\\&X_n=temp_n\%A_n\end{aligned} \tag{6}$$

对于单元格而言,构成该立方的维记作 $d_1,d_2,...,d_n$.设 x 是立方中的一个单元格,x 在维 d_i 上对应的维值为 v^i,$code(v^i)=x_i$,则 x 的坐标为$(x_1,x_2,\ldots,x_n)$,公式(5)将$(X_1,X_2,\ldots,X_n)$替换为$(x_1,x_2,\ldots,x_n)$,$\langle A_1,A_2,\ldots,A_n\rangle$替换为$\langle|d_1|,|d_2|,\ldots,|d_n|\rangle$,公式(6)亦然,可得单元格的线性化和反线性化算法.在存储实现中,单元格存储为 Hadoop HDFS[28]中 MapFile[29]文件的一条记录,该记录包括该单元格线性化后的坐标及其包含的度量.

对于块而言,设 y 是一个块,它是立方的一个划分,$|y|=\langle\lambda_1,\lambda_2,\ldots,\lambda_n\rangle$,$y$ 的坐标为$(y_1,y_2,\ldots,y_n)$,y 内任意一个单元格的坐标为$(x_1,x_2,\ldots,x_n)$,则 $y_i=\left\lfloor\frac{x_i}{\lambda_i}\right\rfloor$,公式(5)中,将$(X_1,X_2,\ldots,X_n)$替换为$(y_1,y_2,\ldots,y_n)$,$\langle A_1,A_2,...,A_n\rangle$替换为$\langle\lambda_1,\lambda_2,\ldots,\lambda_n\rangle$,公式(6)亦然,可得单元格的线性化和反线性化算法.在存储实现中,一个块存储为 Hadoop HDFS 中一个 MapFile 文件,线性化后的块坐标作为块文件的文件名,以便进行块文件的寻址;块文件内每一条记录存储了该块内的一个单元格,块文件则存储于分布式文件系统 Hadoop HDFS 中.在块文件中按 *Key-Values* 对的形式存储所有的单元格数据,其中,*Values* 对应事实数据,*Key* 则是单元格(cell)坐标按公式(5)线性化后的索引值.该索引值是一个十进制正整数,在大数据环境中,数据立方规模会非常大,以 Java 语言为例,索引值会超出长整型(long)的最大取值范围 2^{64},因此,我们采用字符串表示的数字数据.此外,若采用 *Key-Values* 方式存储,*Key* 的存储开销可能会比对应的 *Values* 存储开销大很多,这样就浪费了大量存储.因此对于一个块文件,仅记录一个最小的单元格索引值,而 *Key* 存储的则是相对于该值的偏移量.

2.5 应用案例

本节描述一个 DOLAP 的真实应用案例 OceanCube.案例的数据模式来源于海洋科学中通过 CTD[30]收集的海水温度数据,数据集来源于真实的国家海洋数据[30],该数据集中的数据项均为连续型数值数据.OceanCube 数据集描述如下:3 个维分别为时间维、区域维和深度维,对应 *Time*,*Area* 和 *Depth*.*Time* 共有 5 个级别:Year,Season,Month,Day,Slot.其中,Slot 是指一天的 3 个时间段,其维值分别是“上午”、“下午”和“晚上”.*Area* 共有 7 个维级别:1°,1/2°,1/4°,1/8°,1/16°,1/32°和 1/64°.其中,1°是指由 1°经度和 1°纬度所组成的方形区域,地球表面可以划分为 360×180 个 1°方区.*Depth* 共有 3 个维级别:100m,50m,10m,其中,100m 指的是“海洋的深度以 100m 的间隔进行划分”,若海水有 1 000m 深,则可以划分为 10 层,对应 10 个维值.见表 2.

Table 2

表 2

各个维规模	数据立方规模和容量	最优的块规模及块容量
(1) $\lvert Time\rvert$=11160 (2) $\lvert Area\rvert$=320 (3) $\lvert Depth\rvert$=100	(1) $\lvert OceanCube\rvert=\langle 11160,320,100\rangle$ (2) $\lvert OceanCube\rvert^n$=357120000	(1) $\lvert OceanChunk\rvert=\langle 72,64,50\rangle$ (2) $\lvert OceanChunk\rvert^n$=230400

我们设每个月有 31 天,初始化数据立方为 OceanCube,其中包括 10 年、5°、1 000m 深度的海洋温度数据,维信息如下所示:

(1) $Time=\{\langle$Year,10$\rangle,\langle$Season,4$\rangle,\langle$Month,3$\rangle,\langle$Day,31$\rangle,\langle$Slot,3$\rangle\}$;

(2) *Area*={⟨1°,5⟩,⟨1/2°,2⟩,⟨1/4°,2⟩,⟨1/8°,2⟩,⟨1/16°,2⟩,⟨1/32°,2⟩,⟨1/64°,2⟩};

(3) *Depth*={⟨100m,10⟩,⟨50m,2⟩,⟨10m,5⟩}.

因此,OceanCube 一共产生 1 550 个块文件,每个块中将包含 230 400 个单元格,其中,每个单元仅有 1 个度量,即海水温度.

3 OLAP 算法

本节在数据模型的基础上介绍 OLAP 算法及其 MapReduce 实现,OLAP 包括上卷、下钻、切片、切块以及旋转操作.切片、切块可以视作范围查询;上卷、下钻可以视作范围查询和数据聚集;旋转可以视作创建立方的不同视图.一个 OLAP 操作的输入可以抽象为四元组表示,记作⟨*Target*,*Range*,*Aggregation*,*Result*⟩,其中,*Target* 代表待分析的数据立方(元数据);*Range* 代表立方中待分析数据的数据范围;*Aggregation* 指的是聚集函数,例如 *mean*(·),*sum*(·),maximum(·)以及 minimum(·);*Result* 则代表结果数据立方(元数据).当 *Target* 的最高维级别低于(高于)*Result* 的最高维级别时,表示完成上卷(下钻)操作.OLAP 操作的输入和输出均为数据立方,*Result* 是 *Target* 经过查询和聚集后新生成的立方.显然,*Result* 和 *Target* 的维规模可能有所不同.

我们通常通过维来查询度量,所以,OLAP 操作的查询条件同样由维组成.*Range* 是一个多维二元组,指出了 *Target* 中待分析数据的数据范围.设组成立方 *Target* 的维集合为{$d_1,d_2,\ldots,d_n$},对于维 d_i,序偶⟨α_i,β_i⟩($\alpha_i<\beta_i$)代表在该维上所需数据的取值范围,则 *Range*={⟨α_i,β_i⟩|$i\in[1,n]$}.

为了便于表述,*Range* 记为如⟨{$\alpha_1,\alpha_2,\ldots,\alpha_n$},{$\beta_1,\beta_2,\ldots,\beta_n$}⟩的形式.

3.1 块选择算法

本文提出的 OLAP 操作包括 4 种基本算法:块选择、数据查询、改变维级别、数据聚集,很多现有研究都讨论了如何在 MapReduce 中实现查询和聚集操作,改变维级别参见第 2.2.2 节的维遍历算法,因此本节仅针对块选择进行讨论.块选择是指“确定包含满足查询条件的数据的所有块”的过程.利用块选择算法减少 OLAP 操作输入块的个数,缩小查询空间,可以极大地提高 OLAP 性能.设一个 OLAP 操作中给定 *Range* 为⟨{$\alpha_1,\alpha_2,\ldots,\alpha_n$},{$\beta_1,\beta_2,\ldots,\beta_n$}⟩,满足 *Range* 的块的坐标为($c_1,c_2,\ldots,c_n$),块规模为⟨$\lambda_1,\lambda_2,\ldots,\lambda_n$⟩,则$\forall i\in[1,n]$,$c_i$ 满足公式(7):

$$\left\lfloor\frac{\alpha_i}{\lambda_i}\right\rfloor\leqslant c_i\leqslant\left\lfloor\frac{\beta_i}{\lambda_i}\right\rfloor \tag{7}$$

满足公式(7)的块会作为 OLAP 操作的输入,而非输入全部数据块,从而缩小了 OLAP 操作的查询空间;而且块选择算法无需额外的查询,仅通过编码计算,算法代价很小.图 3 显示了块选择算法的示例.

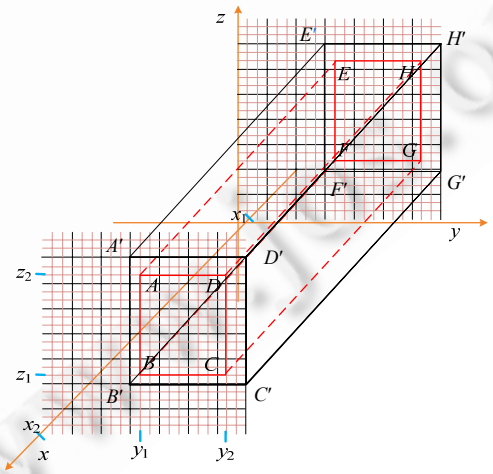


Fig.3 Example of chunk selection

图 3 块选择算法示例

设某 OLAP 操作中给定 *Range* 为$\langle\{x_1,y_1,z_1\},\{x_2,y_2,z_2\}\rangle$.图 3 中,$F$ 的坐标为(x_1,y_1,z_1),D 的坐标为(x_2,y_2,z_2),则待查询的数据范围是 $ABCD\text{-}EFGH$.通过公式(7)可得实际查询输入块的范围是 $A'B'C'D'\text{-}E'F'G'H'$.对比整个立方,通过块选择可以筛选掉一大部分与查询无关的数据,但是 $A'B'C'D'\text{-}E'F'G'H'$的边缘块中仍有少部分无关数据,这部分数据将在 OLAP 操作的查询阶段进行过滤.

3.2 基于MapReduce的算法实现

以上卷操作为例,基于 MapReduce 的 OLAP 算法由 4 部分组成:*InputFormatter*,*Mapper*,*Reducer* 和 *OutputFormatter*,分别对应上卷操作中的查询、改变维级别、聚集和输出结果集的 4 个步骤.上卷操作执行流程如图 4 所示.

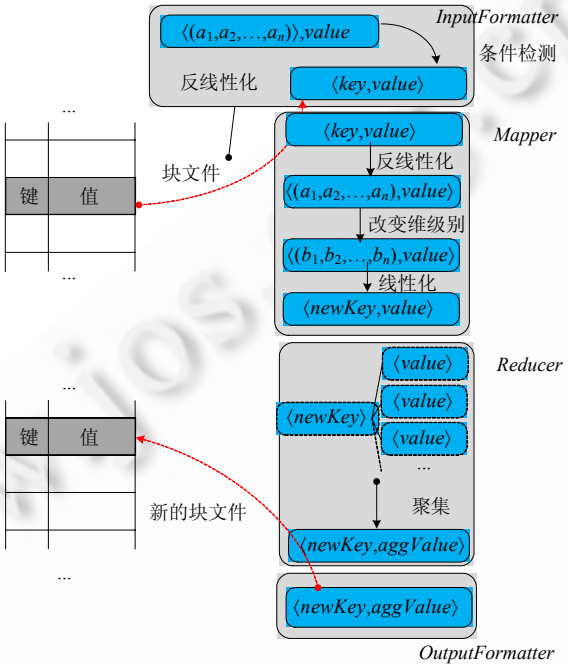


Fig.4 Process of MapReduce based OLAP algorithm

图 4 基于 MapReduce 的 OLAP 算法执行流程

在 MapReduce 任务执行前,OLAP 查询四元组将会被终端提交,系统验证四元组中信息的有效性,防止任务因查询定义无效而失败.随后利用块选择算法获取输入块列表,称为 *chunk-file-list*.*InputFormatter* 扫描 *chunk-file-list* 中块包含的单元格,将线性化的单元格坐标反线性化,按查询条件对数据进行过滤.如果某一单元格符合查询条件,则该单元格的数据将会传递给 *Mapper* 进行后续处理;若不符合查询条件,该单元格将会被抛弃.

Mapper 和 *Reducer* 的输入和输出都是由用户来定义格式的键值对形式,其中,*Mapper* 函数输入为$\langle Key_M^{In},Value_M^{In}\rangle$,输出为$\langle Key_M^{Out},Value_M^{Out}\rangle$;*Reducer* 函数的输入为$\langle Key_R^{In},Value_R^{In}\rangle$,输出为$\langle Key_R^{Out},Value_R^{Out}\rangle$.其中,$Key_M^{Out}$能够隐式地转换为 Key_R^{In},$Value_R^{In}$ 是保存 $Value_M^{Out}$ 的集合.上卷操作中,Key_M^{In} 是单元格的线性化坐标,$Value_M^{In}$ 则是由 *InputFormatter* 所读取的单元格内的度量.Key_M^{Out} 是更新维级别后单元格的线性化坐标,$Value_M^{Out}$ 则与 $Value_M^{In}$ 相同.Key_R^{In} 与 Key_M^{Out} 相同,$Value_R^{In}$ 是具有相同 Key_M^{Out} 的度量的集合.Key_M^{Out} 与 Key_R^{In} 相同,$Value_R^{Out}$ 是 $Value_R^{In}$ 的聚集值.

Mapper 首先将 Key_M^{In} 反线性化,根据客户端给出的 *Result* 信息,完成维层次的攀升,对单元格坐标进行更新.*Mapper* 将更新后的坐标线性化,作为 Key_M^{Out} 输出.对于单元格内的度量不进行聚合运算,所以在 *Mapper* 输出过

程中,$Value_M^{Out}=Value_M^{In}$.*Reducer* 根据客户端指定的 *Aggregation* 对 $Value_R^{In}$ 内所有的值进行聚集运算,并将最终的聚集值作为 $Value_R^{Out}$ 输出.在这个过程中,Key_R^{In} 保持不变,所以最终输出时,$Key_R^{Out}=Key_R^{In}$.*Mapper* 输出数据后,由于 *Partition* 的作用,属于同一个块的单元格总会被分配到同一 *Reducer* 中进行分组处理,所以,*Reducer* 的输出是每一个块文件内的所有单元格.

OutputFormatter 接收 *Reducer* 的输出、计算块的坐标、并对坐标进行线性化,最终生成物理存储文件保存到分布式文件系统中,文件名是该块的线性化坐标.当所有的块文件生成后,将新产生的数据立方信息写入到 *Result* 中并返回给客户端.

4 实验分析

为了验证本文所提出的 DOLAP 在大数据环境中的高效性,我们实现了一个基于 Hadoop 的 OLAP 系统 HaoLap(hadoop based OLAP).由于篇幅原因,本文略去 HaoLap 的实现细节.

本节比较了 HaoLap,Hive,HadoopDB,HBase 和 Olap4Cloud 的 OLAP 性能,HaoLap 采用本文提出的 DOLAP,Hive,HadoopDB 和 HBase 采用基于星形模式的 ROLAP,Olap4Cloud 采用的则是 MOLAP.除此之外,还对各个系统的数据装载过程和存储代价进行了对比.

本实验运行在真实的集群环境中,该集群中共包含 13 台 PC 机,包含 1 个 *NameNode*,12 个 *DataNode*.每台 PC 机的软、硬件配置如下:Inter Core i5 2.80GHz CPU,8GB 内存,1TB 硬盘,CentOS 6.3,Linux 2.6.32-279.el6.i686 操作系统.

在实验的初始阶段,我们拟将业界广泛关注的开源 NoSQL 数据库 MongoDB 纳入到实验中,但是最终没有采纳,原因如下:① MongoDB 是一个文档型 NoSQL 数据库,如果将维和事实存储于同一个文档中,将会造成该文档的结构过于复杂而难以实现 OLAP 操作;② MongoDB 并不原生地支持连接操作,我们可以采用星形模式将维和度量分别存储于 MongoDB 中,同时使用一些编程技巧实现连接操作,但是无法避免 MongoDB 连接操作时对中间数据进行不必要的分片过程,该分片过程相当耗时,会影响到性能对比.

HaoLap 设计之初是为了应用于国家海洋科学数据中连续的数值型维的区间查询和 OLAP 操作,如第 2.5 节中的应用案例所述,但同样也适用于离散的枚举型维的 OLAP 操作.因此,针对数值型维,本节采用真实的科学数据集,比较 HaoLap 和其他主流云数据库系统的性能,将涉及 4 组实验,分别是数据装载、切块操作、上卷操作和存储代价.每个实验都将涉及多组实验用例,并通过 3 个不同规模的数据集对比 5 个系统的性能;针对枚举型维,将采用 SSB 基准测试用例,比较 HaoLap 和其他系统的性能;最后总结实验结论.为表述简单,我们采用 SQL 描述实验用例,针对不同数据库系统,采用不同的方式实现这些用例,具体实现方法从略.

4.1 实验案例

本节采用第 2.5 节描述的案例 OceanCube 作为实验数据.在实验中使用了 3 个数据集(S_1,S_2,S_3),为了便于表述,使用 $Size(S_i)(1\leqslant i\leqslant 3)$表示数据集的规模,$Size(S_i)$的单位为数据条数.本文没有采用大数据研究中常用的 GB 为单位是因为:HaoLap,Hive,HadoopDB 和 HBase 的数据文件格式不同,导致文件大小差异较大.各个数据集相关参数见表 3.

Table 3 Size of OceanCube experimental set

表 3 OceanCube 实验数据集大小

维/维级别	数据集名称		
	S_1	S_2	S_3
Time/Year	3 年	30 年	30 年
Area/1°	1 个	1 个	5 个
Depth/100m	5 层	5 层	10 层
$Size(S_i)$	10^8 条	10^9 条	10^{10} 条

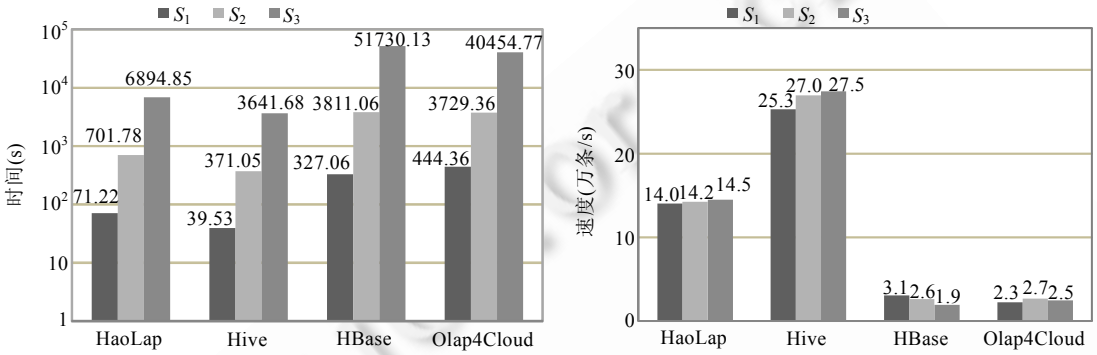
4.1.1 数据装载

由于 HadoopDB 的数据是手工载入的,所以在本实验中无法比较 HadoopDB 的装载性能[31].我们对 HaoLap,

Hive 和 HBase 进行单线程数据装载,若使用多线程装载,会因数据间的固有联系而产生数据不一致问题.对于 Olap4Cloud 而言,首先按照一定的文件格式生成所有的数据,再使用 MapReduce 载入数据,由于数据的生成是 ETL 阶段,在下文中,我们仅针对 MapReduce 过程所耗费的时间进行对比.

图 5 展示了数据装载实验的实验结果.从图 5(b)可以看出,HaoLap 的装载速度较为稳定,而 Hive 的装载速度是 4 个系统中最快的.这是由于 Hive 装载数据时将数据直接写入 HDFS,并在数据装载完成后修改其元数据实现的,而 HaoLap 在数据写入时还需计算每一个单元格的坐标信息.

与 HaoLap 和 Hive 不同的是,HBase 的载入速度随数据量的增大而显著降低.这是由于 HBase 需要额外地维护表的模式信息,如 *Column*,*ColumnFamily*,*Index* 等.Olap4Cloud 的装载速度与 HBase 相近,但在数据量增加后并没有出现装载速度的明显降低.这是因为 Olap4Cloud 的实现基于 HBase,但在本实验中其装载采用 MapReduce 并行方式,并非像 HBase 那样使用单线程装载.



(a) 数据装载时间(对数坐标) (b) 数据装载速度

Fig.5 Experimental results of data loading

图 5 数据装载实验结果

由于 Hive,HBase 和 Olap4Cloud 都提供了相应的 API 进行批量数据装载,且 HaoLap 需要对数据进行预处理,所以 HaoLap 在数据装载方面并没有先天的优势,我们目前考虑的优化方法包括根据数据的特点进行并行导入和优化立方的分块及存储算法.

4.1.2 切块操作

本组实验共包含 3 个切块操作用例 C_1,C_2,C_3(本文对切片和切块操作不加区别),每一个操作都将在 S_1,S_2,S_3 这 3 个数据集上分别执行,因此每个 OLAP 系统均执行 9 次实验.对于用例 C_j($1\leqslant j\leqslant 3$),有 j 个维参与 OLAP 操作,即,星形模式下(维表为 *Time*,*Area*,*Depth*,事实表为 *Temperature*)需要 j 次连接操作.

为了方便表述,在 S_i 上的用例 C_j 记作 S_iC_j($1\leqslant i\leqslant 3,1\leqslant j\leqslant 3$),其结果数据立方数据量记作 $ResultSize(S_iC_j)$ ($1\leqslant i\leqslant 3,1\leqslant j\leqslant 3$),其执行耗时记作 $Time_{[system]}(S_iC_j)$($1\leqslant i\leqslant 3,1\leqslant j\leqslant 3$,system∈{HaoLap,Hive,HadoopDB,HBase, Olap4Cloud}).由于各个用例的查询条件不同,除了 S_3C_1 和 S_3C_2 外,其他 $ResultSize(S_iC_j)$约为 350 万条. $ResultSize(S_3C_1)$为 3 500 万条,$ResultSize(S_3C_2)$为 700 万条.

图 6 用直方图(对数坐标轴)展示了本组实验结果.在同一用例下,HaoLap 的切块性能最优,Hive,Olap4Cloud 和 HadoopDB 次之,HBase 的切块操作性能最差.这种情况可以归结为以下原因:

① HBase,Hive 和 HadoopDB 采用 ROLAP 技术,由于 ROLAP 涉及大量连接操作,性能上没有优势;

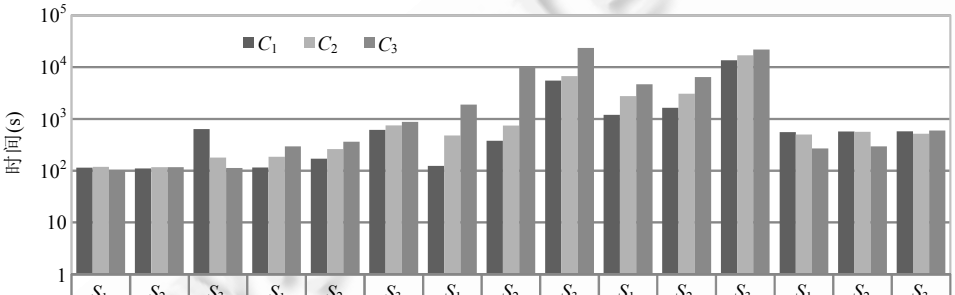
② Hive 和 HadoopDB 对连接操作进行优化,而 HBase 则没有.例如,Hive 可以利用分布式缓存进行 Map 端连接,HadoopDB 则利用索引加速连接操作;

③ Hive 和 HadoopDB 分别采用 HDFS 和 DBMS 作为存储,它们与本地文件系统高度集成,增强了本地文件读写,减少了网络数据传输;而 HBase 则独立于分布式文件系统,缺乏数据本地读写优化,大量的网络

I/O 操作降低了查询性能;

④ Olap4Cloud 在查询时,首先对查询条件进行解析并通过查询索引的方式,精确计算参与计算的数据地址,从而保证仅针对所需数据进行处理.

C_1	
SELECT	*Temperature.value*
FROM	*Temperature*, *Time*
WHERE	*Temperature.t_id=Time.t_id* **AND** *Time.year*=2000
C_2	
SELECT	*Temperature.value*
FROM	*Temperature*, *Time*, *Area*
WHERE	*Temperature.t_id=Time.t_id* **AND** *Temperature.a_id=Area.a_id* **AND** (*Time.year* Between 2000 And 2001) **AND** (*Area.bound* Between 0 And 0.5)
C_3	
SELECT	*Temperature.value*
FROM	*Temperature*, *Time*, *Area*, *Depth*
WHERE	*Temperature.t_id=Time.t_id* **AND** *Temperature.a_id=Area.a_id* **AND** *Temperature d_id=Depth.d_id* **AND** (*Time.year* Between 2000 And 2002) **AND** (*Area.bound* Between 0 And 0.75) **AND** (*Depth.value* Between 1 And 230)



	HaoLap			Hive			HadoopDB			HBase			Olap4Cloud		
	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
C_1	114	111	637	116	172	615	124	377	5 491	1 196	1 642	13 550	553	570	577
C_2	119	116	179	187	261	751	478	743	6 727	2 756	3 088	17 081	501	566	517
C_3	103	118	114	296	365	880	1 889	10 197	23 627	4 657	6 447	21 961	270	295	596

Fig.6 Execution time of dice operation (logarithmic coordinate)

图 6 切块操作的运行时间(对数坐标)

图 7(a)展示了 HaoLap 各个切块操作在不同数据集下的运行时间的变化趋势.

可以看出:除 $Time_{[HaoLap]}(S_3C_2)$运行时间稍长,$Time_{[HaoLap]}(S_3C_1)$运行时间明显增加外,$Time_{[HaoLap]}(S_iC_j)$基本上保持稳定.出现这种现象的原因是 $ResultSize(S_3C_1)$的数据量为 3 500 万条,$ResultSize(S_3C_2)$的数据量为 1 000 万条,其他结果的数据量均为 350 万条.由此可见,HaoLap 切块性能与结果集(或实际参与运算的数据集)大小相关,与原始数据集大小以及操作复杂程度(维的个数)无关.以 S_iC_3 为例:一方面,其性能与实际参与运算的数据量有关,而在 HaoLap 中,数据立方分块和块选择算法导致 S_iC_3 实际参与运算的数据集大小是相同的;另一方面,块选择算法是基于块文件的直接寻址而不是文件搜索算法,所以块选择算法与原始数据集大小无关.

图 7(b)~图 7(d)分别展示了 Hive,HadoopDB,HBase 的切块操作在不同数据集下运行时间的变化趋势.由于这些系统采用 ROLAP 技术,用例 $C_j(1\leqslant j\leqslant 3)$涉及的连接数量随着 j 的增大而增多,因此切块更耗时.且在不存在连接优化的情况下,表内数据量越大,连接操作越耗时.图 7(b)显示出 Hive 的切块性能符合 ROLAP 的普遍规律.$Time_{[Hive]}(S_iC_j)$随着 $Size(S_i)$以及所需连接表的数量而增加.Hive 仅通过分布式缓存优化 Map 端连接操作,但仅适用于有限的情况.同时,Hive 没有提供索引或分区等优化机制以加速连接和查询的速度,所以 Hive 的性能趋势符合 ROLAP 的普遍规律.

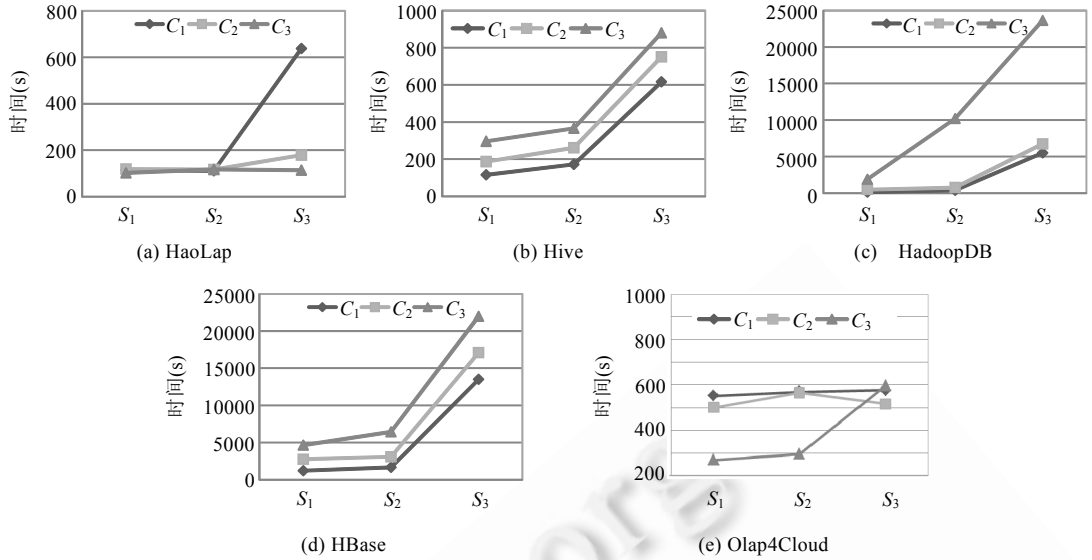


Fig.7 Tendency of dice performance when data set is changed

图 7 切块性能随数据集变化趋势

HadoopDB 与 Hive,HBase 的不同之处是其底层采用关系数据库存储数据.HadoopDB 将数据进行分片,建立索引并存储于 DBMS 中.在图 7(c)中,$Time_{[HadoopDB]}(S_2C_3)$,$Time_{[HadoopDB]}(S_3C_3)$与其他用例相比变化很大,导致 $Time_{[HadoopDB]}(S_iC_1)$和 $Time_{[HadoopDB]}(S_iC_2)$几乎重合.根据图 7(c)以及图 6 中的 HadoopDB 实验数据可得:$Time_{[HadoopDB]}(S_iC_1)$和 $Time_{[HadoopDB]}(S_iC_2)$曲线规律符合 ROLAP 的一般规律.HadoopDB 采用了索引优化查询,所以 S_1C_1 和 S_2C_1,S_1C_2 和 S_2C_2 这两组对比实验的查询时间很相近.但是,随着数据量的增加,索引优化效果明显降低,导致 S_3C_1 和 S_2C_1,S_3C_2 和 S_2C_2 这两组对比实验的执行时间相差很大;$Time_{[HadoopDB]}(S_2C_3)$和 $Time_{[HadoopDB]}(S_3C_3)$对比其他用例非常耗时,这是由于 HadoopDB 采用的分片方法加速了连接操作,同时避免了数据在节点间的传输,但是,当数据量显著增大、连接表的数量增多时,这种优化效果明显降低,所以,$Time_{[HadoopDB]}(S_iC_3)$曲线对比其他曲线要增长得更快.

由图 7(d)可知,HBase 对于 OLAP 没有任何优化策略,其性能符合 ROLAP 的一般规律.但是,HBase 在相同用例下的切块操作执行时间对比其他系统要长很多.图 8(a)展示在用例 S_1C_1 中,HBase 的 *Mapper* 和 *Reducer* 数量分别是 Hive 的 2.8 倍和 13 倍;图 8(b)则说明,用例 S_1C_1 中,*Mapper* 和 *Reducer* 的执行时间 HBase 是 Hive 的 20 倍和 2.3 倍.HBase 的任务执行性能比 Hive 要低很多.这种现象表明,HBase 并不适合执行多表的连接以及切块操作.

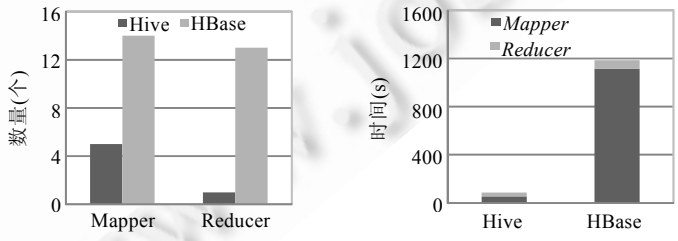
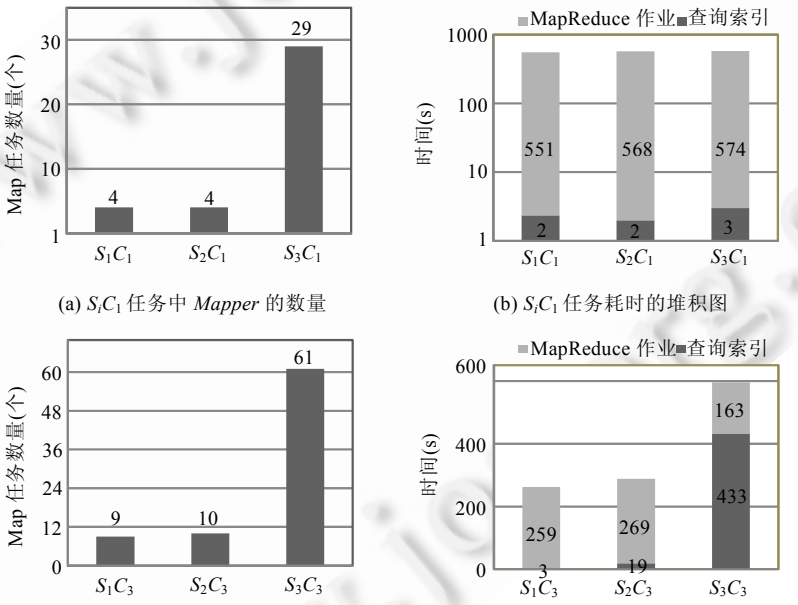


Fig.8 Performance comparison of Hive and HBase on S_1C_1 testcases

图 8 Hive 和 HBase 对于用例实验数据对比

图 7(e)展示了 Olap4Cloud 的切块操作在不同数据集下运行时间的变化趋势.从图中可以看出,Olap4Cloud 的整体切块性能较前 4 者都有很大不同.首先它并没有表现出一个固有趋势.其中,C_1 和 C_2 任务在各个节点上运行时间较为稳定,但是 C_3 在不同的数据集下,运行时间差异非常明显,特别是 S_1C_3 和 S_2C_3 与 S_3C_3 相差较大.为了对这一现象进行分析,我们考虑 Olap4Cloud 查询中的两个关键因素:查询索引时间和 MapReduce 任务时间.我们知道,MapReduce 处理时间与 Map,Reduce 任务的个数、数据量以及算法复杂度相关.在本实验中,查询方法是相同的,所以我们忽略算法复杂度所带来的影响.由于 Olap4Cloud 的索引技术,除 S_3C_1 和 S_3C_2 外的任务参与计算的数据量也是相同的.同时,由于 Olap4Cloud 基于 HBase,所以我们设定 *Reducer* 的个数固定为 12 个,即集群中 *datanode* 的个数.S_iC_1 与 S_iC_2 的曲线的趋势是相同的,我们仅针对 S_iC_1 进行研究.

图 9(a)和图 9(b)中展示了 S_iC_1 中各个用例的详细信息.参与 S_3C_1 的数据量为 S_1C_1 和 S_2C_1 的 3 倍,但 S_3C_1 所启动的 *Mapper* 数量则是 S_1C_1 和 S_2C_1 的 7.25 倍;同时,由于 C_1 仅针对一个维进行操作,所以使得查询索引的时间很快,各个操作所使用的时间差不多.这就导致了 S_iC_1 各个用例运行时间较为稳定的现象.图 9(a)和图 9(b)中展示了 S_iC_3 中各个用例的详细信息.对于 C_3 操作而言,在各个数据集下参与计算的数据量是相同的.但是各个数据集中维的取值不同,导致其索引文件的大小不相同.维的取值越多,索引文件越大.同时,在对 HBase 中 *HTable* 进行扫描的过程中,由于数据集的构成不同,从而使得在 MapReduce 任务中所启动的 *Mapper* 任务个数有所不同,其中,S_1C_3 和 S_2C_3 中 *Mapper* 分别为 9 个、10 个,但是 S_3C_3 中 *Mapper* 数量近乎是两者的 6 倍.所以,S_3C_3 任务在 MapReduce 任务中所使用的时间比 S_1C_3 和 S_2C_3 要少.但是,S_3 数据集所含数据量分别是 S_2 的 10 倍,S_1 的 100 倍,从而导致最终生成的索引文件更加庞大,所以 S_3C_3 中扫描索引的时间较长.同时,Olap4Cloud 执行查询时,首先在主节点中单线程地扫描整个索引表,然后再执行 MapReduce 任务,所以导致扫描索引的时间成为限制整个任务的瓶颈.



(a) S_iC_1 任务中 *Mapper* 的数量　(b) S_iC_1 任务耗时的堆积图

(c) S_iC_3 任务中 *Mapper* 的数量　(d) S_iC_3 任务耗时的堆积图

Fig.9　Detail information about test cases of Olap4Cloud

图 9　Olap4Cloud 中用例的详细信息

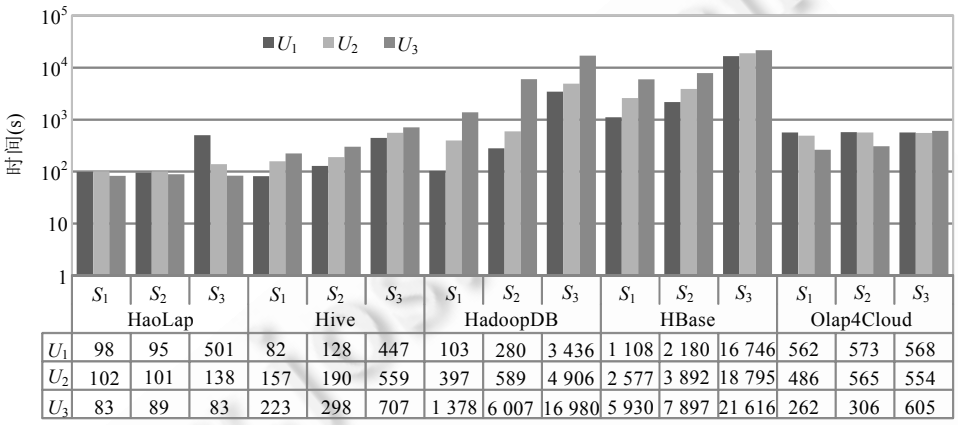
Olap4Cloud 将维表压缩如事实表的方式避免了连接操作,同时,使用索引的方式确定参与计算的实际数据量,缩短了 OLAP 运行时间.但是,当数据量过大、同时操作较为复杂涉及维较多时,容易产生系统瓶颈问题.从总体上看,DOLAP 的切块性能要优于 Olap4Cloud.

由本组实验可以得出:DOLAP的切块操作性能优势非常明显,且操作的性能与原始数据集的规模以及参与计算的维数无关.

4.1.3 上卷操作

本组实验共包含3个上卷操作用例 U_1,U_2,U_3,每个操作都将在 S_1,S_2,S_3 这3个数据集上分别执行.对比切块用例,U_j 和 C_j 的查询条件是完全相同的.U_j 比 C_j 包含维攀升和对度量的聚集操作,U_j 将度量从 *Time* 维的 Slot 级别聚集到 Month 级别.为了表述方便,实验用例记作 $S_iU_j(1\leqslant i\leqslant 3,1\leqslant j\leqslant 3)$,其结果数据的数据量记作 $ResultSize(S_iU_j)(1\leqslant i\leqslant 3,1\leqslant j\leqslant 3)$,用例的执行时间记作 $Time_{[system]}(S_iU_j)(1\leqslant i\leqslant 3,1\leqslant j\leqslant 3,system\in\{HaoLap,Hive,HadoopDB,HBase,Olap4Cloud\})$.由于聚集操作,$ResultSize(S_iU_j)$约为 $ResultSize(S_iC_j)$的1%.

各个用例的运行时间如图10所示,图11展示了在各个系统中上卷操作的性能趋势.



	HaoLap			Hive			HadoopDB			HBase			Olap4Cloud		
	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
U_1	98	95	501	82	128	447	103	280	3 436	1 108	2 180	16 746	562	573	568
U_2	102	101	138	157	190	559	397	589	4 906	2 577	3 892	18 795	486	565	554
U_3	83	89	83	223	298	707	1 378	6 007	16 980	5 930	7 897	21 616	262	306	605

Fig.10 Execution time of roll-up operation (logarithmic coordinate)

图10 上卷操作的运行时间(对数坐标)

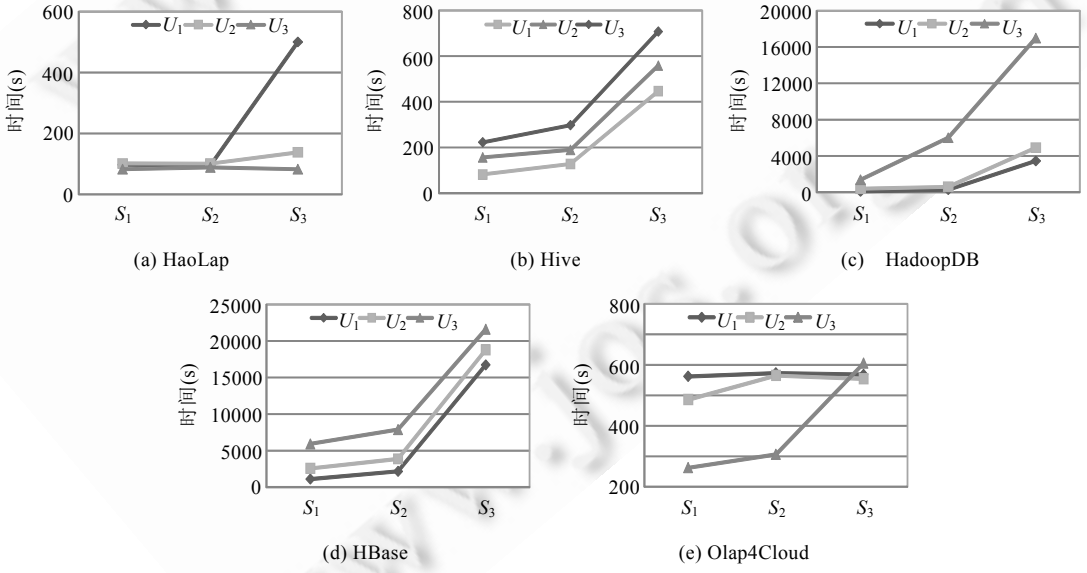


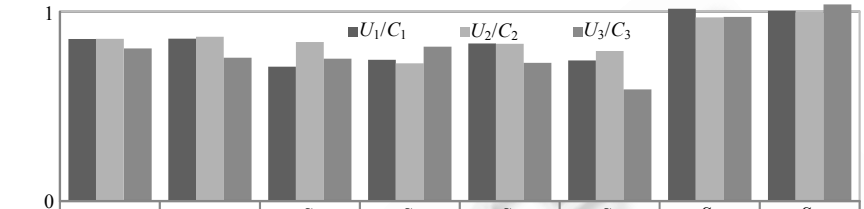
Fig.11 Tendency of roll-up performance when data set is changed

图11 上卷操作性能随数据集变化趋势

由图11可知,HaoLap的上卷性能优于其他系统,Hive,Olap4Cloud和HadoopDB次之,HBase的上卷性能最

差.图 11 的曲线趋势与图 7 非常接近,这是由于维攀升是通过维遍历算法计算来实现的,代价很小,而聚集操作并没有主导上卷性能.

通过计算 $Time_{[system]}(S_iU_j)$与 $Time_{[system]}(S_iC_j)(1\leqslant i\leqslant 3,1\leqslant j\leqslant 2,system\in\{HaoLap,Hive,HadoopDB,Olap4Cloud\})$的比值来对上卷操作与切块操作进行对比分析,该比值记作 $Radio^{ij}_{[system]}$.如果 $Radio^{ij}_{[system]}=1$,表明在该系统上卷操作与切块操作的性能相当;如果 $Radio^{ij}_{[system]}<1$,表明该系统中 S_iU_j 的执行速度时间比 S_iC_j 要短;反之亦然.由于结果数据集大小不同以及 HBase 整体性能较差,我们仅计算并分析 $1\leqslant i\leqslant 3,1\leqslant j\leqslant 2,system\in\{HaoLap,Hive,HadoopDB,Olap4Cloud\}$的 $Radio^{ij}_{[system]}$.实验结果如图 12 所示.



	S_1	S_2	S_1	S_2	S_1	S_2	S_1	S_2
	HaoLap		Hive		HadoopDB		Olap4Cloud	
U_1/C_1	0.85	0.86	0.71	0.75	0.83	0.74	1.01	1.00
U_2/C_2	0.86	0.87	0.84	0.73	0.83	0.79	0.97	1.00
U_3/C_3	0.81	0.76	0.75	0.81	0.73	0.59	0.97	1.04

Fig.12 Performance comparison of roll-up and dice

图 12 上卷操作与切块操作对比

如图 12 所示, $Radio^{ij}_{[system]}<1$,HaoLap,Hive 和 HadoopDB 的上卷操作性能比切块操作性能要高.原因是:上卷操作中的聚集运算导致上卷结果数据集是切块结果数据集的 1%,从而大大减少了数据输出时的 I/O 操作.但是 Olap4Cloud 的切块和上卷操作性能差不多,这是由于 Olap4Cloud 是基于 HBase 的,其数据写入性能本身较低,从而 I/O 操作无法成为整个操作性能中的主要因素.通过对比得出结论,额外的维攀升和聚集操作并没有影响 DOLAP 的性能.

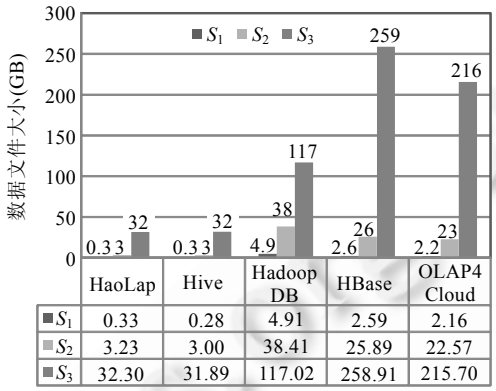
4.1.4 存储代价

本实验从数据存储和多维模型存储两个方面来度量分析 HaoLap 的存储代价.我们首先比较了上述 S_i-在 HaoLap,Hive,HadoopDB,HBase 和 Olap4Cloud 的存储数据集大小;随后介绍数据立方(cube)存储方法,并通过理论和实验分析证明:即使在高维数据集下,数据立方的存储代价仍然很小.

图 13 对比了各数据集在 HaoLap,Hive,HadoopDB,HBase 和 Olap4Cloud 系统中的存储代价.在实际实验中,备份数量采用 Hadoop 默认值为 3,图 13 中所列出的数值均不含数据备份的大小.以数据集 S_3 为例,HBase 和 OLAP4Cloud 的存储开销约是 HaoLap 和 Hive 的 8 倍,是 HadoopDB 的 2 倍.HaoLap 和 Hive 所占用磁盘空间相差无几,且小于其他系统,这是由于 HaoLap 和 Hive 均直接采用分布式文件系统 HDFS 中的原生文件格式存储数据,存储结构简单,并没有引入额外元数据而仅存储数据本身.由于 OLAP4Cloud 是基于 HBase 的 OLAP 系统,所以 OLAP4Cloud 和 HBase 的存储代价相似,且高于其他系统.HadoopDB 采用关系数据库系统 PostgreSQL 存储数据,首先采用哈希算法将事实数据分割为多个子数据集,并将子数据集分别导入到不同的数据节点的 PostgreSQL 数据库中,同时创建索引;为优化连接操作,HadoopDB 要求将维数据复制到每一个数据节点中.上述操作均导致额外的存储开销,造成 HadoopDB 的存储代价较高.此外,由于 S_1,S_2,S_3 数据集大小相差 10 倍,对应的 HaoLap 的数据文件大小也相差近 10 倍,说明存储代价和数据集大小呈良好的线性关系.

HaoLap 的数据立方存储代价非常小且可以忽略.为适应大数据环境,本文提出的 DOLAP 与传统 MOLAP 的本质不同在于:前者不需要存储庞大的“多维数组”,而是通过计算获得该“多维数组”.根据第 2.2 节中的描述,DOLAP 对维进行简化,同一级别上维值的编码是连续的,维值树的每一个兄弟节点都有相同个数的子节点,

所以对于每一个维,在 DOLAP 中仅需存储维级别名称和维级别规模,极大地降低了维的存储代价.例如在 HaoLap 的实现中,对于 *Time* 维,仅在主节点采用 XML 文件存储{⟨Year,10⟩,⟨Season,4⟩,⟨Month,3⟩,⟨Day,31⟩,⟨Slot,3⟩}这一条数据,并在运行时加载至内存.HaoLap 通过维编码和遍历算法确定单元格(cell)位置,并通过数据立方分块以及线性化算法完成单元格和物理存储位置(MapFile)的映射,通过 MapReduce 完成数据查询和聚集.因此,可以认为 HaoLap 的数据立方是零实例化的.实验结果表明,即使是高维数据立方,如将 *Time* 维存储 1 000 次,内存中数据对象也仅有 50.3KB.上述分析和实验证明,HaoLap 的数据立方存储代价可以忽略.



	HaoLap	Hive	Hadoop DB	HBase	OLAP4 Cloud
S_1	0.33	0.28	4.91	2.59	2.16
S_2	3.23	3.00	38.41	25.89	22.57
S_3	32.30	31.89	117.02	258.91	215.70

Fig.13 Storage comparison of HaoLap, Hive, HadoopDB, HBase and Olap4Cloud

图 13 数据集在 HaoLap,Hive,HadoopDB,HBase 和 Olap4Cloud 的存储代价对比

4.2 基准测试

为证明 HaoLap 系统以及本文提出的 DLOAP 算法能够适用于枚举型维的数据集,我们采用了 SSB 基准测试用例.如图 14 所示的 SSB 标准[26]是 TPC-H 标准的星型模型,目前被学术界所广泛采用.它将模式清晰地分解为 4 个维表和 1 个事实表,消除了 TPC-H 中 LIENITEM 和 ORDER 表的巨大连接代价,也消除了雪花状模型带来的复杂查询执行计划,从而使其更加适合于大规模并行计算环境下的简单数据分布[18].

由图 14 可以看出,与第 2.5 节描述的案例 OceanCube 中的连续数值型维属性不同,SSB 中维表属性多为枚举型数据,如 CUSTOMER 表的 CITY 属性以及 SUPPLIER 表中的 NAME 属性,且每个维上的维值远大于 OceanCube,属于高维数据.我们采用的分析用例 *D* 的 SQL 描述如下:

	D
SELECT	*C_NATION*, *S_NATION*, *D_YEAR*, *SUM*(*LO_REVENUE*) **AS** *REVENUE*
FROM	*CUSTOMER*, *LINEORDER*, *SUPPLIER*, *DATE*
WHERE	*LO_CUSTKEY*=*C_CUSTKEY* **AND**
	LO_SUPPKEY=*S_SUPPKEY* **AND**
	LO_ORDERDATE=*D_DATEKEY* **AND**
	C_REGION='*ASIA*' **AND** *S_REGION*='*ASIA*' **AND**
	D_YEAR>=1992 **AND** *D_YEAR*<=1997 **AND**
GROUP BY	*C_NATION*, *S_NATION*, *D_YEAR*
ORDER BY	*D_YEAR ASC*, *REVENUE DESC*

为了对比第 4.1 节的实验结果,我们同样选择了 S_1,S_2,S_3 这 3 个测试数据集作为测试数据,按 SSB 的数据生成规则,SF(scale factor)分别对应为 10,30 和 60,数据集大小见表 4.对比表 3 中 OceanCube 数据集可知,SSB 数据集为高维数据集,维数据量大于 OceanCube;而维和事实数据并非一一映射,这就导致在相同维数据的情况下,事实数据量小于 OceanCube.另外,表 3 所给出的数据为生成数据文件的大小,由于各个系统实现不同,会导致在各个系统内数据文件的大小会有明显差异.

基准测试实验比较了 HaoLap,Hive,HadoopDB 和 Olap4Cloud 在 S_1,S_2,S_3 这 3 个测试数据集上执行用例 *D* 的性能.根据第 4.1 节的分析,HBase 的连接性能较其他系统差距显著,因此基准测试略去了上述实验.

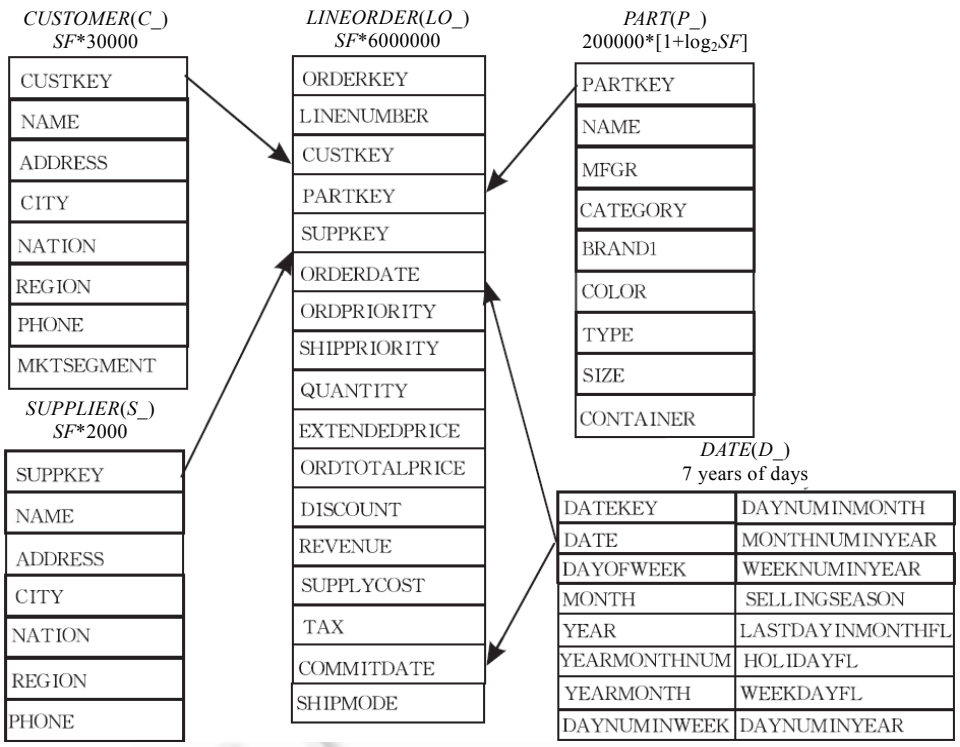


Fig.14 Schema of SSB

图 14 SSB 数据集的模式

Table 4 Size of SSB experimental set

表 4 SSB 实验数据集大小

表名	数据集名称		
	S_1 (*SF*=10)	S_2 (*SF*=30)	S_3 (*SF*=60)
CUSTOMER	300 000 条	900 000 条	1 800 000 条
SUPPLIER	20 000 条	60 000 条	120 000 条
PART	800 000 条	1 000 000 条	1 200 000 条
DATE	2 556 条	2 556 条	2 556 条
LINEORDER	5.6GB	17.2GB	34.8GB

从图 15 所示的基准测试实验结果可以看出:即使采用由离散型数据组成的高维数据立方,HaoLap 的分析性能仍然明显优于其他 3 个系统.具体分析如下:

① 整体上,HaoLap 的分析性能略优于 Hive,且明显优于 HadoopDB 和 Olap4Cloud;

② HaoLap 在图 15 中的分析性能规律和图 6 以及图 10 中呈现的规律是一致的,连接聚集查询性能与数据量和查询语句的复杂性无关,仅取决于选中的数据集大小.用例 D 对查询结果进行了聚集操作,因此,S_1,S_2,S_3 的结果数据集大小是一致的,但查询命中率不同,因此性能不同;

③ 对比 HaoLap 和 Hive 的性能可以看出,HaoLap 的性能优势在 SSB 数据集中没有其在 OceanCube 数据集中那样明显.这是因为在高维数据集下,HaoLap 的单元格索引值已经超出了 Java 语言长整型(long)的数据范围(2^{64}),因此,HaoLap 采用字符串表示大整数且进行数学运算,这种运算的性能开销难以忽略;

④ 基于 DOLAP 的 HaoLap 的分析性能明显要优于基于 MOLAP 的 Olap4Cloud,原因在于 Olap4Cloud 依赖索引在数据立方中可以精确地定位数据存储位置,这样做减少了数据搜索范围,但在高维数据立方

下,索引查询和维护的开销很大.实验中,Olap4Cloud 查询非常不稳定,经常出现无法响应的情况,为了得到完整结果,我们对 Olap4Cloud 的索引作了一些手工优化.而 HaoLap 则通过维编码和线性化算法初步确定数据存储位置,块选择算法选中的是包含目标数据的最小数据块集合,并没有精确定位到数据本身,数据查询则采用基于 MapReduce 的并行算法扫描数据块,避免了维护和查询索引的代价.

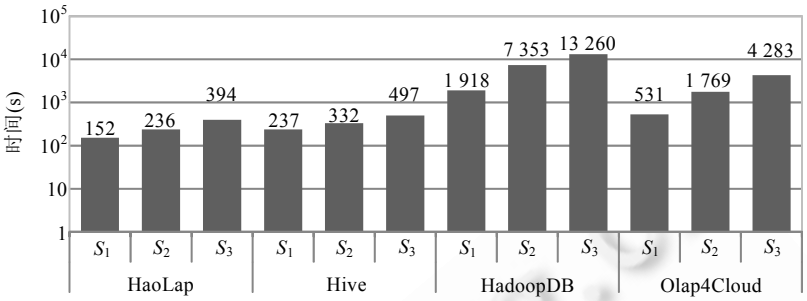


Fig.15 Performance comparison of SSB query

图 15 在 SSB 数据集中查询的性能对比

4.3 实验结论

HaoLap 是 DOLAP 在大数据环境中的实现系统,本节比较了 HaoLap,Hive,HadoopDB,HBase,Olap4Cloud 的数据装载、切块操作、上卷操作性能以及存储代价;实验数据采用了真实的由连续数值型维组成的真实数据集 OceanCube 以及由离散枚举型维组成的高维 SSB 数据集.实验结果表明:尽管 HaoLap 在数据装载方面没有优势,但其 OLAP 性能明显高于其他 4 个系统,且性能不取决于原始数据集的规模以及操作的复杂程度,存储代价和 Hive 相当并低于其他 3 个系统.产生这种现象的主要原因有以下两点:DOLAP 技术采用了简化的维模型和编码机制,使得维和度量的映射、维层次的遍历都更加高效;DOLAP 技术使用分块策略存储度量并在 OLAP 执行过程中使用块选择算法,尽量缩小数据查询范围.此外,本实验还分析了 Hive,HadoopDB,HBase,Olap4Cloud 的 OLAP 性能特征和成因.

5 结论与进一步工作

本文提出了一种用于大数据分析的 DOLAP 技术.该技术包括以下 6 个方面:① 采用特殊的多维模型来组织维和度量;② 维编码和遍历算法实现了维值树上的上卷下钻操作;③ 简化的维存储方法,且数据立方分块算法实现了维和度量的映射关系以及度量的分布式存储;④ 线性化和反线性化算法实现了块和单元格的高效寻址算法;⑤ 通过块选择算法来优化 OLAP 性能;⑥ 提出基于 MapReduce 的 OLAP 算法实现.与此同时,本文还实现了基于 Hadoop 的 DOLAP 实现系统 HaoLap,并比较了 HaoLap,Hive,HadoopDB,HBase,Olap4Cloud 的装载性能和 OLAP 性能.实验结果表明:尽管 HaoLap 在数据装载方面没有优势,但其 OLAP 性能明显高于其他 4 个系统,且性能与原始数据集的规模以及操作的复杂程度无关,存储代价和 Hive 相当并低于其他 3 个系统.同时,本文还分析了其他系统的 OLAP 性能,总结了相关规律.

本文的进一步工作将着重于以下几个方面:① 讨论数据块是否应被压缩、如何被压缩、如何在压缩数据立方上执行 OLAP 操作,以及压缩对 OLAP 性能将带来哪些影响;② 优化 HaoLap 数据装载的性能.

References:

[1] Gray J, Liu DT, Nieto-Santisteban M, Szalay A, DeWitt DJ, Heber G. Scientific data management in the coming decade. ACM SIGMOD Record, 2005,34:34−41. [doi: 10.1145/1107499.1107503]

[2] Miller HJ. The data avalanche is here. Shouldn't we be digging? Journal of Regional Science, 2010,50(1):181−201. [doi: 10.1111/j.1467-9787.2009.00641.x]

[3] Wang S, Wang HJ, Qin XP, Zhou X. Architecting big data: Challenges, studies and forecasts. Chinese Journal of Computers, 2011, 34(10):1741−1752 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01741]

[4] Meng XF, Ci X. Big data management: Concepts, techniques and challenges. Journal of Computer Research and Development, 2013,50(1):146−169 (in Chinese with English abstract).

[5] Shim JP, Warkentin M, Courtney JF, Power DJ, Sharda R, Carlsson C. Past, present, and future of decision support technology. Decision Support Systems, 2002,33:111−126. [doi: 10.1016/S0167-9236(01)00139-7]

[6] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. ACM Sigmod Record, 1997,26:65−74. [doi: 10.1145/248603.248616]

[7] Luk WS, Li C. A partial pre-aggregation scheme for HOLAP engines. In: Proc. of the 6th Int'l Conf. on Data Warehousing and Knowledge Discovery (DaWaK 2004). Berlin: Springer-Verlag, 2004. 129−137. [doi: 10.1007/978-3-540-30076-2_13]

[8] Bolosky WJ, Douceur JR, Ely D, Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. ACM SIGMETRICS Performance Evaluation Review, 2000,28(1):34−43. [doi: 10.1145/345063.339345]

[9] Shen DR, Yu G, Wang XT, Nie TZ, Kou Y. Survey on NoSQL for management of big data. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1786−1803 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4416.htm [doi: 10.3724/SP.J.1001.2013.04416]

[10] Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51:107−113. [doi: 10.1145/1327452.1327492]

[11] Hadoop home page. http://hadoop.apache.org

[12] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive: A warehousing solution over a map-reduce framework. Proc. of the VLDB Endowment, 2009,2(2):1626−1629.

[13] Vora MN. Hadoop-HBase for large-scale data. In: Proc. of the 2011 Int'l Conf. on Computer Science and Network Technology. Piscataway: IEEE, 2011. 24−26.

[14] Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A. HadoopDB: An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. Proc. of the VLDB Endowment, 2009,2(1):922−933.

[15] Olap4cloud home page. http://code.google.com/p/olap4cloud/

[16] Song J, Li TT, Zhu ZL, Bao YB, Yu G. Benchmarking and analyzing the energy consumption of cloud data management system. Chinese Journal of Computers, 2013,36(7):1485−1499 (in Chinese with English abstract).

[17] You JG, Xi JQ, Zhang PJ, Chen H. A parallel algorithm for closed cube computation. Computer and Information Science, 2008,8: 95−99. [doi: 10.1109/ICIS.2008.63]

[18] Zhang YS, Jiao M, Wang ZW, Wang S, Zhou X. One-Size-Fits-All OLAP technique for big data analysis. Chinese Journal of Computers, 2011,34(10):1936−1946 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01936]

[19] Cao Y, Chen C, Guo F, Jiang DW, Lin YT, Ooi BC, Vo HT, Wu S, Xu QQ. ES2: A cloud data storage system for supporting both OLTP and OLAP. In: Proc. of the Int'l Conf. on Data Engineering (ICDE). 2011. 291−302. [doi: 10.1109/ICDE.2011.5767881]

[20] Tian X. Large-Scale SMS messages mining based on map-reduce. Computational Intelligence and Design, 2008,1:7−12. [doi: 10.1109/ISCID.2008.9]

[21] Han H, Lee YC, Choi S, Yeom HY, Zomaya AY. Cloud-Aware processing of MapReduce-based OLAP applications. In: Javadi B, ed. Proc. of the 11th Australasian Symp. on Parallel and Distributed Computing. Darlinghurst: Australian Computer Society, 2013. 31−38.

[22] D'Orazio L, Bimonte S. Multidimensional arrays for warehousing data on clouds. In: Hameurlain A, ed. Proc. of the Data Management in Grid and Peer-to-Peer Systems. Berlin, Heidelberg: Springer-Verlag, 2010. 26−37.

[23] Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig latin: A not-so-foreign language for data processing. In: Lakshmanan LVS, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: Association for Computing Machinery, 2008. 1099−1110.

[24] Hu KF, Dong YS, Xu LZ, Yang KH. A novel aggregation algorithm for online analytical processing queries evaluation based on dimension hierachical encoding. Journal of Computer Research and Development, 2004,41(4):608−614 (in Chinese with English abstract).

[25] TPC-H homepage. http://www.tpc.org/tpch/

[26] O'Neil P, O'Neil B, Chen XD, Stephen R. The star schema benchmark and augmented fact table indexing. In: Proc. of the 1st TPC Technology Conf. on Performance Evaluation and Benchmarking (TPCTC 2009). Berlin: Springer-Verlag, 2009. 237–252. [doi: 10.1007/978-3-642-10424-4_17]

[27] Sarawagi S, Stonebraker M. Efficient organization of large multidimensional arrays. In: Proc. of the Int'l Conf. on Data Engineering. IEEE, 1994. 328–336. http://dl.acm.org/citation.cfm?id=645479.655138&coll=DL&dl=GUIDE&CFID=419816811&CFTOKEN=55823079

[28] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of the 2010 IEEE 26th Symp. on Mass Storage Systems and Technologies (MSST 2010). IEEE, 2010. 1–10. [doi: 10.1109/MSST.2010.5496972]

[29] Yang L, Shi ZZ. An efficient data mining framework on Hadoop using Java persistence API. In: Proc. of the 2010 IEEE 10th Int'l Conf. on Computer and Information Technology (CIT 2010). IEEE Computer Society, 2010. 203–209. [doi: 10.1109/CIT.2010.71]

[30] China oceanic information network. http://mds.coi.gov.cn/jcsj.asp

[31] Qin XP, Wang HJ, Du XY, Wang S. Big data analysis—Competition and symbiosis of RDBMS and MapReduce. Ruan Jian Xue Bao/Journal of Software, 2012,23(1):32–45 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4091.htm [doi: 10.3724/SP.J.1001.2012.04091]

附中文参考文献:

[3] 王珊,王会举,覃雄派,周烜.架构大数据:挑战、现状与展望.计算机学报,2011,34(10):1741−1752. [doi: 10.3724/SP.J.1016.2011.01741]

[4] 孟小峰,慈祥.大数据管理:概念、技术与挑战.计算机研究与发展,2013,50(1):146−169.

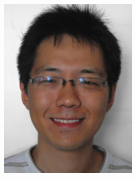
[9] 申德荣,于戈,王习特,聂铁铮,寇月.支持大数据管理的 NoSQL 系统研究综述.软件学报,2013,24(8):1786−1803. http://www.jos.org.cn/1000-9825/4416.htm [doi: 10.3724/SP.J.1001.2013.04416]

[16] 宋杰,李甜甜,朱志良,鲍玉斌,于戈.云数据管理系统能耗基准测试与分析.计算机学报,2013,36(7):1485−1499.

[18] 张延松,焦敏,王占伟,王珊,周烜.海量数据分析的 One-size-fits-all OLAP 技术.计算机学报,2011,34(10):1936−1946. [doi: 10.3724/SP.J.1016.2011.01936]

[24] 胡孔法,董逸生,徐立臻,杨科华.一种基于维层次编码的 OLAP 聚集查询算法.计算机研究与发展,2004,41(4):608−614.

[31] 覃雄派,王会举,杜小勇,王珊.大数据分析——RDBMS 与 MapReduce 的竞争与共生.软件学报,2012,23(1):32−45. http://www.jos.org.cn/1000-9825/4091.htm [doi: 10.3724/SP.J.1001.2012.04091]



宋杰(1980－),男,安徽淮北人,博士,副教授,CCF 高级会员,主要研究领域为云计算,高能效计算,海量数据计算.
E-mail: songjie@mail.neu.edu.cn

郭朝鹏(1990－),男,硕士生,主要研究领域为海量数据计算.
E-mail: gcp543706787@gmail.com

王智(1991－),男,硕士生,主要研究领域为高能效计算.
E-mail: neoh.jackson@gmail.com

张一川(1981－),男,博士,讲师,CCF 会员,主要研究领域为云计算,迭代计算.
E-mail: zhangyc@mail.neu.edu.cn

于戈(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为先进数据库系统,分布与并行式系统,大数据管理,云计算系统.
E-mail: yuge@mail.neu.edu.cn

Jean-Marc PIERSON(1970－),男,博士,教授,博士生导师,主要研究领域为分布式系统,高能效计算,大数据管理,云计算.
E-mail: pierson@irit.fr