在PC集群上的封闭立方体计算

游进国[1,2] 奚建清[1] 张平健[1] 刘艳霞[1]
(华南理工大学计算机科学与工程学院 广州 510641)[1]
(昆明理工大学信息工程与自动化学院 昆明 650093)[2]

摘 要 封闭立方体是联机分析处理中一种有效的数据立方体压缩技术,但封闭立方体的并行算法目前很少有相关文献研究。提出了一种简单而实用的解决方案,即基于MapReduce计算框架,在非共享内存的PC集群上对封闭立方体进行分布式的预计算和查询。相关实验表明,本方法能快速处理千万级的数据,具有较好的线性加速比,而且能够更大地压缩数据立方体存储空间。

关键词 联机分析处理,并行计算,封闭立方体,MapReduce技术
中图法分类号 TP311 **文献标识码** A

Closed Cubing on PC Clusters

YOU Jin-guo[1,2] XI Jian-qing[1] ZHANG Ping-jian[1] LIU Yan-xia[1]
(School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China)[1]
(School of Information Engineering and Automation, Kunming University of Science and Technololgy, Kunming 650093, China)[2]

Abstract The closed cube is a very efficient technology for the data cube compression in OLAP, but its parallel algorithm is little studied in the literature. This paper presented a solution that is easy to be implemented and applied. It parallelizes the closed cube construction and the query answering based on the MapReduce framework over low cost share nothing PC clusters. The experiments show that our approach can rapidly process huge data sets with at least 10 million rows and get pretty good linear speedup. Further, it compresses data cubes much greater.

Keywords OLAP, Parallel computation, Closed cube, MapReduce

1 引言

联机分析处理是决策支持系统中一项重要的技术,能将大量数据聚合到数据立方体[1]中,以做多视角的在线分析。然而数据量逐渐增大时,产生的数据立方体存储空间通常更为巨大,查询性能也将下降。针对该问题,当前研究者提出了许多串行和并行的算法。

在串行技术方面,近年来的研究主要是集中在数据压缩方面。文献[2]提出Dwarf Cube,该方法通过识别相同前缀和相同后缀来消除空间冗余。文献[3, 4]提出了更为有效的算法Quotient Cube,将度量相等,且具有上卷下钻语义的一组单元无损压缩成该组中的有且仅有一个的上界单元;其还提出基于自底向上算法框架(BUC)[5]的DFS算法计算出上界集。在查询时,可以由上界单元推导出组内其它的数据单元。文献[6, 7]称Quotient Cube为更直观的封闭立方体(closed cube),相应上界称为封闭元组或封闭单元。

并行技术方面主要是将串行算法在多个CPU上并行化。计算方式主要有两种策略:工作划分和数据划分。工作划分是指将要计算的立方体中的不同方体(即视图)进行分组,然后每组分配给一个CPU计算。工作划分方式中,文献[8]研究自底向上算法并行化处理;文献[9]结合自底向上以及自顶向下算法进行并行,但它们采用的BUC算法并没有进行有效的数据立方体压缩。数据划分是指将整个输入数据集划分为不同的数据子集,分配给相应的CPU计算。文献[10]采用数据分区方式对ROLAP进行了计算,主要基于较早提出的Pipesort算法做了并行化,也没有考虑立方体压缩。

本文提出了一种OLAP并行计算方案,以封闭立方体算法为核心,基于MapReduce计算框架[11]在PC集群下对封闭立方体的计算和查询进行并行化。封闭立方体能够达到很大的压缩率,本文提出将其分布式存储,能够进一步压缩存储空间。

2 基本概念

令r表示关系模式R上的关系,C是在r上预计算生成的数据立方体。数据单元c是封闭的,如果不存在其它数据单元c',满足c'上卷(泛化)到c并且c'和c的度量值相等。封闭立方体则是封闭单元的集合。

由关系r预计算生成的封闭立方体叫全局封闭立方体。

到稿日期:2008-08-12 返修日期:2008-11-26 本文受广东省国际科技合作计划项目(2007A050100026),广东省科技计划项目(2006B11301001),广东省工业科技攻关计划项目(2006B80407001)资助。

游进国(1977—),男,博士生,主要研究方向为数据仓库、并行计算等,E-mail:jgyou@126.com;奚建清(1962—),男,教授,博士生导师,主要研究方向为数据库、信息集成等;张平健(1966—),男,副教授,主要研究方向为数据仓库等;刘艳霞(1979—),女,博士生,主要研究方向为信息集成。

相应地，由关系 r 划分的一个子集预计算生成并保存在本地节点的封闭立方体叫本地封闭立方体。本地封闭立方体与全局封闭立方体大小关系如下：

定理 1 本地封闭立方体是全局封闭立方体的子集。

证明：在关系 r 上的全局封闭立方体的封闭单元可以通过关系 r 的子集求出，即子集上产生的所有封闭单元一定属于全局封闭立方体，因此子集上的本地封闭立方体一定包含于全局封闭立方体，即是全局封闭立方体的子集。

引理 1 本地封闭立方体并集小于全局封闭立方体。

证明：设关系 r 有 T 个不同元组，$T>1$，其被水平划分为 k 个子集，$1\leqslant k\leqslant T$，这些子集分别生成 k 个本地封闭立方体 L_1，L_2，…，L_i，…，L_k，令 $S=|\cup L_i|$；又设在 r 上生成的全局封闭立方体为 O，令 $S'=|O|$。由定理 1，可以得到对于所有 $1\leqslant i\leqslant k$，$L_i\subseteq O$，并进一步有 $\cup L_i\subseteq O$，$|\cup L_i|\leqslant|O|$，即 $S\leqslant S'$。当 $k=T$ 时，关系 r 被划分为 T 个子集，每个子集只包含一个元组，因此只分别产生一个封闭单元，此时 S 达到最小值 T，即 $T\leqslant S$。

例子 设关系 $r=\{(1\ 1\ 1),(1\ 2\ 2),(2\ 1\ 2)\}$，产生的全局封闭立方体 $C_c=\{(1\ 1\ 1),(1\ 2\ 2),(2\ 1\ 2),(1\ *\ *),(*\ *\ 2),(*\ 1\ *),(*\ *\ *)\}$，其大小 $|C_c|=7$。又设 r 被分区为 $\{(1\ 1\ 1),(1\ 2\ 2)\}$ 和 $\{(2\ 1\ 2)\}$，生成的本地封闭立方体大小分别是 $\{(1\ 1\ 1),(1\ 2\ 2),(1\ *\ *)\}$ 和 $\{(2\ 1\ 2)\}$，因此本地封闭立方体包含封闭单元的总数是 4，小于全局封闭立方体。

3 算法

3.1 系统结构

本系统由一个名字节点和多个数据节点组成。数据保存在多个数据节点上，并由名字节点管理和调度。具体而言，名字节点采用数据划分方式进行数据分块，分发数据块到数据节点，维护数据块在节点位置等信息，并进行分布式计算任务的调度；数据节点保存和读写数据块，对数据块执行 Map 计算任务以及 Reduce 计算任务。

3.2 算法概览

在 MapReduce 框架中，只要设计和实现适当 map 函数和 reduce 函数，就能实现封闭立方体的并行处理。

算法执行如图 1 所示，主要分为如下步骤。

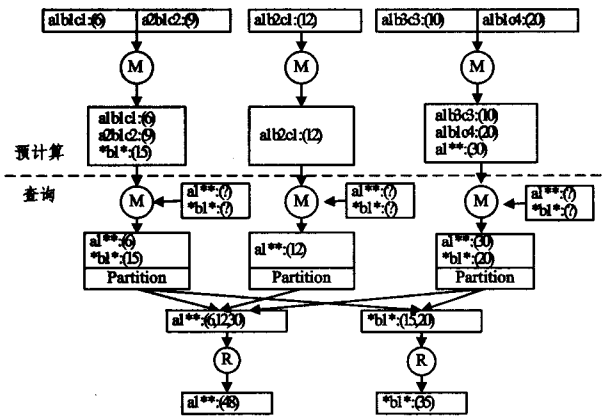


图 1 算法执行概览

(1)系统对要计算的大容量数据集进行分块，并分发到节点上；

(2)节点上的 Map 任务对其数据块进行预计算并输出一个本地封闭立方体；

(3)用户请求查询(如一系列的点查询)，系统将该查询提交到节点上的 Map 任务；

(4)Map 任务对该节点上的本地封闭立方体进行查询，并返回对应的本地结果集；

(5)这些本地结果集以点查询为键进行分区，形成一系列查询结果列表；

(6)对于一个查询结果列表，Reduce 任务将该列表中各个返回的度量值聚合为一个值交给用户。

3.3 分布式预计算

由引理 1，对于给定的输入数据集，在其上构造的全局封闭立方体大于所有本地封闭立方体并集的大小。因此，预计算算法对输入数据集切分的各个数据块构造封闭立方体后，保存在本地，并不将这些封闭立方体再合并成一个全局封闭立方体，这样能够节省存储空间。

预计算算法只包括 Map 任务，其算法 ParallelClosedCubing 设计如下：

Map 输入：(key：分块号 blockid，value：块数据 blockdata)；

Map 输出：(key：分块号 blockid，value：本地封闭立方体 closedCells)。

```
全局变量：string closedCells;
map(string blockid, string blockdata)
(1) cl = (ALL, …, ALL);
(2) call DFS(cl, blockdata, 0);
(3) emit(blockid, closedCells);

DFS(cl, data, d)
(1) Compute the closed cell cc of cl;
(2) closedCells += cc;
(3) if cc is examined previously, then return;
(4) for(i=d; i<dimsNum; i++)
(5)     if cc[i] == ALL
(6)         card=Partition(data, i); //partition data in ith dimension
(7)     for(j=0; j<card-1;j++)
(8)       let subdata=data's jth partition;
(9)       cc[i]=subdata[0][i];
(10)       DFS(cc, subdata, i);;
(11) return
```

系统首先将输入数据集进行分块，形成分块号 blockid，分块数据 blockdata 的键值对，并将数据块分发到空闲的节点上；节点上的 Map 任务随后启动，并调用 DFS 算法对输入的数据块进行预计算。DFS 算法自底向上，深度优先地递归计算出每一个封闭单元，并收集到变量 closedCells 中；DFS 结束后，以 blockid，closedCells 形成键值对作为本地封闭立方体输出。

由于一个节点获得数据块后，只在本地对该数据块做计算，并不与其它节点数据交换。并且输出的键是唯一的，随后 MapReduce 对键的合并操作并产生的节点间通讯开销很小。

3.4 分布式查询

本系统可以一次执行一个或多个查询。请求的查询以文件的形式保存，并发布到各个节点上，这样节点上的计算任务可以直接对其读取。分布式查询算法 ParallelQuery 包含 Map 任务和 Reduce 任务，其中 Map 任务的算法设计如下。

Map 输入：(key：分块号 blockid，value：本地封闭立方体

closedcells)；

Map 输出：(key：查询点 cell，value：查询点度量值 msr)。

map(string blockid, string closedcells)

(1) get request from the local file;

(2) for each cell in request

(3) msr = query(cell, closedcells); //scan the cell in the closedcells

(4) emit(cell, msr);

Reduce 任务的算法设计如下：

Reduce 输入：(key：查询点 cell，value：度量值列表 msrlist)；

Reduce 输出：(key：查询点 cell，value：查询点最终结果 result)。

reduce(string cell, float[] msrlist)

(1) result = 0;

(2) for each msr in msrlist

(3) result += msr;

(4) emit(cell, result);

用户请求查询，系统将该查询分发到集群中的每一个节点；节点接收到后，启动 Map 任务，扫描本地封闭立方体执行查询，并将每一个数据单元及其对应的度量值组成键值对输出。系统以数据单元为键进行排序分区，将属于同一数据单元的度量值放在一起形成一个中间键值对。随后 Reduce 任务启动，将数据单元对应的度量进行合并为一个全局的结果。

尽管本算法需要查询各个节点，但查询立方体是在本地节点进行的，返回结果集很小，节点间通讯开销较小。

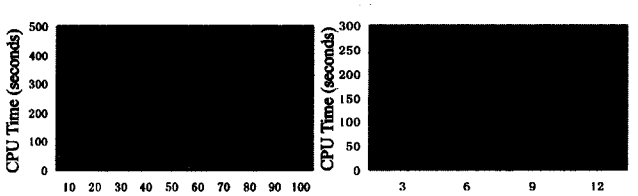
4 实验

系统采用了 C++语言实现分布式预计算和查询中的 Map 以及 Reduce 任务，并基于实现 MapReduce 等功能的开源软件 Hadoop[12]进行任务并行化。

实验环境是 18 台普通 PC 机组成的集群系统。其中每台机器硬件配置是：3.0 GHz Intel Pentium 4 CPU，1GB RAM 和 30 GB 7200 RPM IDE 可用硬盘空间，软件配置是：Linux 操作系统(内核版本是 2.6.21，gcc 版本是 4.1.2)。一台机器作为名字节点，其它机器作为数据节点。测试数据是人工合成的。

4.1 预计算性能

数据集的元组数从 1000 万递增到 1 亿条时，在 17 个节点的集群上的预计算耗用时间也是逐步递增。同时系统也具有较快的性能，例如在 17 个节点上计算 6000 万的数据时，平均时间才 268s，见图 2(a)。当 2000 万元组的数据集的维数由 3 维递增到 9 维时，预计算时间指数增长，见图 2(b)，这是因为尽管数据集的元组数保持不变，但每增加一个维，需要扫描的分区(或 group-bys)个数也指数增长，也就耗用更长的时间。

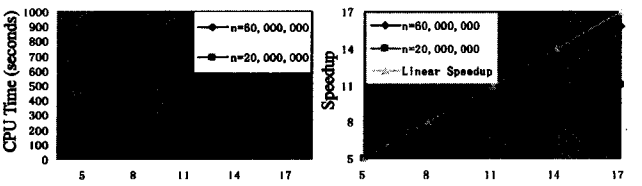


(a) 不同元组数下的预计算时间 (d=5, p=17)

(b) 不同维数下的预计算时间 (n=20,000,000, p=17)

图 2

在另外一组实验中，我们变化集群中的节点数，分别对 5 维、2000 万元组和 5 维、6000 万元组的数据集进行了预计算。在图 3(a)中，当节点数从 5 递增到了 17，2000 万元组的数据集预计算时间下降得比较平缓(降幅达 121s)，而 6000 万元组的数据集预计算时间则下降得很快(降幅达 588s)。这是因为在节点数较少时，更大的数据集造成更多的因外排序引起的磁盘 I/O 次数以及更多的节点间通讯开销；而当节点数较多时，数据在节点间分配比较合理，外排序较少，节点间的通讯开销大大减少，因此时间也减少许多。小数据集无论节点数多还是少，磁盘 I/O 和通讯开销都较少，因此时间变化不大。相应地，在图 3(b)中，6000 万元组的数据集预计算加速比比 2000 万元组的数据更趋向于线性加速比。



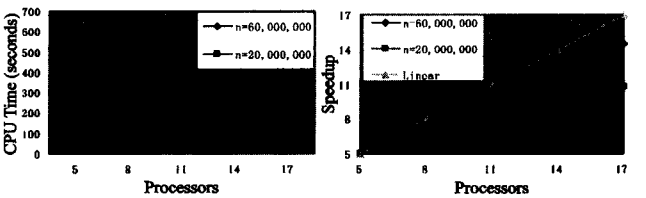
(a) 不同节点数下的预计算时间 (d=5)

(b) 相应的加速比

图 3

4.2 查询性能

在一定节点数下，对 5 维、2000 万元组和 5 维、6000 万元组的数据集预计算完成后，相应的查询也被测试。我们随机生成 1000 个点查询，并提交到系统。如图 4 所示，查询回答性能较好。例如在 17 个节点上对 6000 万元组的数据集产生的封闭立方体做 1000 个点查询，总共耗用时间 203s，平均每个数据单元的查询时间约 0.2s。当节点数递增时，6000 万元组数据集产生的封闭立方体查询时间比 2000 万元组数据集产生的封闭立方体查询时间下降得更快，也更趋向于线性加速比。这是因为查询引起的磁盘 I/O 和通讯开销逐步减少，而小数据集引起的变化并不大。



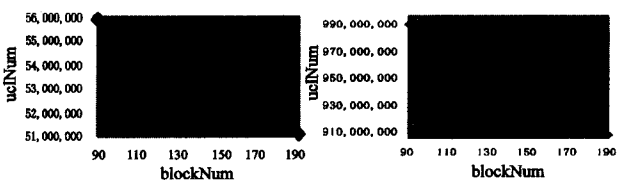
(a) 不同节点数下的查询时间 (同时执行 1000 个点查询)

(b) 相应的加速比

图 4

4.3 数据分块的影响

进一步地，我们还通过调整给定输入数据集的数据分块数，考察了封闭立方体的存储空间的变化。在图 5 中，对 5 维、2000万元组的数据集分块数从90递增到190，产生的所



(a) 不同数据分块数下的封闭单元个数

(b) 不同数据分块数下的封闭立方体文件大小

图 5

(下转第 161 页)

建模工具 ModelMaker 为设计和运行平台，通过调用它的 ToolsApi 接口模块控制活动图的一切元素。

本文以一个简单的从自动售货机买物品的片段为例子试验，对得到的所有场景排序得到场景的优先级，如图 1 所示。

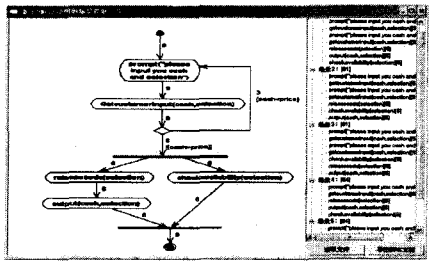


图 1

从这个实验中我们可以找到优先级高的测试场景，同时在场景中也可以看到各个消息的优先级。

结束语 本文通过对 UML 活动图测试场景进行优先级判断，从而利用这些优先级来指导我们的测试工作，当看到优先级高的测试场景就应该花大力气着重测试，这样就更能发现系统的关键问题，保障系统的健壮性，而本文的进一步工作是确定测试的充分性，测试需要到达什么级别的优先级我们就认为系统测试可以接受了。

参 考 文 献

[1] 李秋英，刘斌，阮镰. 灰盒测试方法在软件可靠性测试中的应用[J]. 航空学报，2002，23(5)：455-458

[2] 谢棠棠，张为群. 一种基于 UML 模型的系统测试方法[J]. 西南师范大学学报：自然科学版，2005，30(2)：258-263

[3] 谢棠棠. 基于 UML 模型的测试场景生成研究与工具实现[D]. 重庆：西南师范大学，2005

[4] Wang Linzhang，Yuan Jiesong，Yu Xiaofeng，et al. Generating Test Cases from UML Activity Diagrams Based on Gray-Box Method[C]//11thAsia-pacific Software Engineering Conference (APSEC'04). Washington：IEEE Computer Socrety，2004：284-291

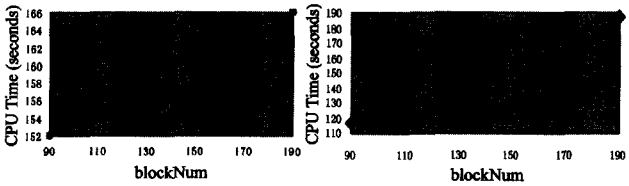
[5] Schmuller J. UML 基础、案例与应用(第三版)[M]. 李虎，赵龙刚，等译. 北京：人民邮电出版社，2004

[6] Binder R V. 面向对象系统的测试[M]. 华庆一，王斌君，陈莉，译. 北京：人民邮电出版社，2001

(上接第 155 页)

维有本地封闭立方体所包含的封闭单元数从 5600 万递减到 5100 万，相应的文件大小从 994MB 递减到 912MB。由该图趋势可以大致估计，当分块数为 1 时，产生的全局封闭立方要大得多，这进一步验证了引理 1。

然而，尽管封闭立方体存储空间会随着分块数增加而减小，相应的预计算时间和查询时间会增加，其中查询时间增幅较大。如图 6 所示，当数据分块数从 90 递增到 190 时，预计算时间增幅 14s，查询时间增幅 71s。因此，用户可以根据实际应用需求，通过调整数据分块的个数，在空间和时间上选择一个最佳点。



(a) 不同数据分块数下的预计算时间 (p=11)　(b) 不同数据分块数下的查询时间 (p=11)

图 6

结束语 企业和单位积累大量的数据亟待高性能的联机分析和处理，本文提出了一种在 PC 集群下的分布式处理的解决方案，即基于 MapReduce 框架的封闭立方体分布式计算和查询。实验表明，通过融合 MapReduce 框架和封闭立方体这两种技术的优势，实现的系统具有更大的数据压缩率，能够短时间内处理千万级数据，具有较好的扩展性，并且能通过 MapReduce 框架自动进行系统容错、负载平衡等处理。

在下一步工作中，我们将考虑怎样使用 MapReduce 框架实现数据立方体的增量维护，并研究和实现一个在 MapReduce 框架之上的分布式数据仓库原型系统。

参 考 文 献

[1] Gray J，Chaudhuri S，Bosworth A，et al. Data Cube：A Relational Aggregation Operator Generalizing Group-By，Cross-Tab，and Sub-Totals [J]. Data Mining and Knowledge Discovery，1997，1(1)：29-53

[2] Sismanis Y，Deligiannakis A，Roussopoulos N，et al. Dwarf：Shrinking the PetaCube [C]//SIGMOD. 2002：464-475

[3] Lakshmanan L V S，Pei J，Han J W. Quotient Cubes：How to Summarize the Semantics of a Data Cube [C]// VLDB. 2002：778-789

[4] Lakshmanan L V S，Pei J，Zhao Y. QCTrees：An Efficient Summary Structure for Semantic OLAP [C] // SIGMOD. 2003：64-75

[5] Beyer K，Ramakrishnan R. Bottom-Up Computation of Sparse and Iceberg CUBEs [C] // SIGMOD. 1999

[6] Xin D，Shao Z，Han J W，et al. C-Cubing：Efficient Computation of Closed Cubes by Aggregation-based Checking [C]// ICDE. 2006：4

[7] 李盛恩，王珊. 封闭数据立方体技术研究 [J]. 软件学报，2004，15(8)：1165-1171

[8] Ng R T，Wagner A，Yin Y. Iceberg-cube Computation with PC Clusters [C]//SIGMOD. 2001，30(2)：25-36

[9] Frank D，Todd E，Andrew R. The cgmCUBE project：Optimizing parallel data cube generation for ROLAP [J]. Distributed and Parallel Databases，2006，19(1)：29-62

[10] Chen Y，Dehne F，Eavis T. Parallel ROLAP Data Cube Construction on Shared-nothing Multiprocessors [J]. Distributed and Parallel Databases，2004，15(3)：219-236

[11] Dean J，Ghemawat S. MapReduce：Simplified Data Processing on Large Clusters[C]// 6th Symposium on Operating Systems Design and Implementation. 2004

[12] Apache. Hadoop[EB/OL]. http：//lucene.apache.org/hadoop/，2007