第41卷 第6A期
2014年6月

计 算 机 科 学
Computer Science

Vol. 41 No. 6A
June 2014

基于快速排序和huffman树的物化视图增量保持算法

张银玲 武 彤 邓少勋
(贵州大学计算机科学与信息学院 贵阳 550025)

摘 要 物化视图是一种提高查询响应的有效方法,其核心问题在于如何与基表保持数据同步。目前已经有很多算法用于物化视图增量保持,各算法的效率不同,因此也出现了很多提高物化视图增量保持效率的算法。在构造最优Delta传播树的BinPartition算法的基础之上提出了一种基于快速排序和huffman树的物化视图增量保持算法,并通过实验证明了该算法可以提高物化视图增量保持的效率。

关键词 物化视图,增量保持,快速排序,huffman树
中图法分类号 TP301 **文献标识码** A

Materialized Views Incremental Algorithm Based on Quicksort and Huffman Tree

ZHANG Yin-ling WU Tong DENG Shao-xun
(College of Computer Science and Information, Guizhou University, Guiyang 550025, China)

Abstract Materialized view is an effective way to improve query response, its essential question is how to maintain the data synchronization with the base table. There are many algorithms maintaining materialized view incrementally and each have different time response, so there also appeared a lot of algorithms to improve the efficiency of maintaining materialized views incrementally. This paper proposed a materialized views incremental algorithm based on quicksort and huffman tree, and proved this algorithm can improve the efficiency of maintaining materialized views incrementally.

Keywords Materialized view, Incremental maintenance, Quicksort, Huffman tree

随着对查询性能要求的提高,物化视图成为提高查询响应时间的一种有效方法。物化视图是典型的利用空间来换取时间的策略,其核心问题是如何保持与基础表的数据同步一致。根据不同的需要,可以创建投影(projection)、选择-投影-连接(select-project-join, SPJ)和聚集(aggregation)等类型的物化视图。物化视图可以选择两种更新方式,一种是重新计算的全量更新,另一种是只更新最近一次时间的变化,即增量更新。不难理解,当基础表中的数据很大时,只更新变化的数据比重新计算的时间开销会少得多。但对于SPJ物化视图,由于其连接了多张基础表,实现快速的增量更新比较困难。目前,很多主流数据库软件都提供了物化视图功能,而在实际的应用中,对于SPJ型的物化视图进行快速更新时总是出现各种错误最终导致更新失败。因此,正确且高效地实现物化视图增量更新意义重大。

文献[1-10]提出了一些可行的增量更新策略和算法。文献[1,2]分别提出了两种用于表示物化视图变化的表达式。但文献[1]提出的表达式用于计算物化视图变化的连接项太多,导致计算物化视图变化的效率太低。而文献[2]中的表达式虽然比文献[1]表达式的连接项少,但当基础关系中存在大表时,由于多次连接大表导致其计算物化视图变化的代价也过高。文献[3]提出了一种Delta传播树策略,用Delta传播树描述物化视图增量保持的各种传播可能,通过选择不同的传播顺序可以减少对大表的访问,从而避免了多次连接大表的时间开销,降低了计算物化视图变化的代价。该文献也提出了一种动态规划算法,其能够从所有的Delta传播树中找到最优的传播树使得计算物化视图变化的费用最低,当n足够大时,该算法的时间复杂性接近于$O(2^n \times n)$,其中n为参与物化视图连接的基础关系个数。文献[4-6]在该算法的基础上进行了改进,提出了两种改进算法。其中文献[4,5]提出了一种改进的构造最优Delta传播树的二分贪心算法,该算法的时间复杂度为$O(n\log n)$,该算法在Delta传播树策略上提高了约10%的效率;文献[6]提出了一种流水线的并行算法,该算法可以将时间开销缩小到$1/k$(k为并行的CPU个数)。

本文就文献[5]中提出的算法进行分析,在该算法的基础上加入了快速排序的思想,提出一个基于快速排序和Huffman树的物化视图增量保持算法,并通过实验证明了该算法可以快速构造最优Delta传播树,从而提高物化视图增量保持的效率。

1 基本介绍

一个由SPJ表达式定义的物化视图,假设数据源为$R_1, R_2, \cdots, R_n$,n为数据源个数(即基础关系数目)。则物化视图V可以定义为:

本文受贵州省工业攻关项目:生产线质量控制系统的开发研究(黔科合GY字[2010]3061)资助。
张银玲(1988—),女,硕士,主要研究方向为数据库技术与应用系统;武 彤(1964—),女,教授,硕士生导师,主要研究方向为数据库、数据仓库、数据挖掘技术;邓少勋(1980—),男,讲师,主要研究方向为数据库技术、软件工程等。

$$V=\Pi_L\sigma_c(R_1\infty R_2\infty\cdots\infty R_n) \quad (1)$$

式中，Π 表示投影操作，L 代表投影属性，σ 表示选择操作，C 为选择的条件，R 代表关系。

该式表示物化视图是由 n 个关系在一定的条件下选择投影连接而成。为了能够实现增量更新物化视图的内容，必须要获取到每一个基础关系发生的变化（用 ΔR_i 表示），然后通过相应的计算后将变化的数据传播到物化视图中以实现物化视图的同步更新。文献[3]提出了一种 Delta 传播树策略，即将组成物化视图连接的各个基础关系进行不同的划分，然后用树的形式来描述不同划分情况时各基础关系的变化的传播情况，最后实现物化视图增量保持。该文献将这种传播树定义为 Delta 传播树，用 Delta 传播树可以描述物化视图增量保持的各种传播可能（通过不同的基础关系划分实现不同的 Delta 传播树），通过选择不同的传播顺序可以减少对大表的访问，从而避免了多次连接大表所耗费的时间开销，最终降低了计算物化视图变化的时间代价。为了在所有可能的 Delta 传播树中找到一种最高效的传播树，该文献也提出了一种动态规划算法，其能够找到最优的传播树使得计算物化视图增量保持的费用最低。但当 n 足够大时，该算法的时间复杂性接近于 $O(2^n\times n)$，显然，该时间复杂度太高。因此，出现了很多改进的物化视图增量保持算法[4-6]，以及其他类型的物化视图更新算法[7-10]，其目的都是希望可以减少物化视图增量保持的时间开销。

2 构造最优传播树的 BinPartition 算法

文献[5]在文献[3]的基础上提出了一种构造最优 Delta 传播树的 BinPartition 算法。该算法的思想由 huffman 树[11]的构造思想而来，其形式化描述如下：

(1)对给定的一组关系 $\{R_1,R_2,\cdots,R_n\}$（n 为关系的数目），以它们的变化 $\Delta(R_1),\Delta(R_2),\cdots,\Delta(R_n)$ 作为结点并将 $|R_1|,|R_2|,\cdots,|R_n|$（$|R_i|$ 表示关系的大小，即关系中的元组数）视为相应的权值，构造具有 n 棵二叉树的森林 $F=\{T_1,T_2,\cdots,T_n\}$，其中每棵树 T_i 只有一个权为 $|R_i|$ 的根结点 $\Delta(R_i)(1\leqslant i<n)$。

(2)重复以下步骤，直至 F 中剩下一棵树为止：

①在 F 中选取两棵根结点的权值最小的二叉树 T_i（设根为 $\Delta(R_s\infty\cdots\infty R_t)$）和 $T_j(\Delta(R_u\infty\cdots\infty R_v))$ 作为左、右子树构造一棵新的二叉树 T'。T' 的根为 $\Delta(R_s\infty\cdots\infty R_t\infty R_u\infty\cdots\infty R_v)$，权值为其左、右子树的根结点权值之和；

②从 F 中删去 T_i,T_j；

③将 T' 加入 F，返回步骤①。

最后得到的就是具有根结点最小计算费用的最优的Delta传播树。

该算法在构造最优 Delta 传播树上的时间复杂性与 huffman 树的构造算法是一样的，为 $O(n\log n)$，n 为参与运算的关系的个数。

下面以一个由 $n=4$ 个关系组成的连接 $\{R_1,R_2,R_3,R_4\}$ 来描述该算法的具体构造过程。假设各关系的权值分别为 $|R_1|=40,|R_2|=20,|R_3|=80,|R_4|=50$。用上述构造最优 Delta 传播树的 BinPartition 算法的构造过程如图 1 所示。

从中可以看出，经过了(a)—(d) 4 个步骤后，(d)即为最终生成的最优的 Delta 传播树，R_3 作为左子结点或右子结点的效果相同，不影响最终的结果。

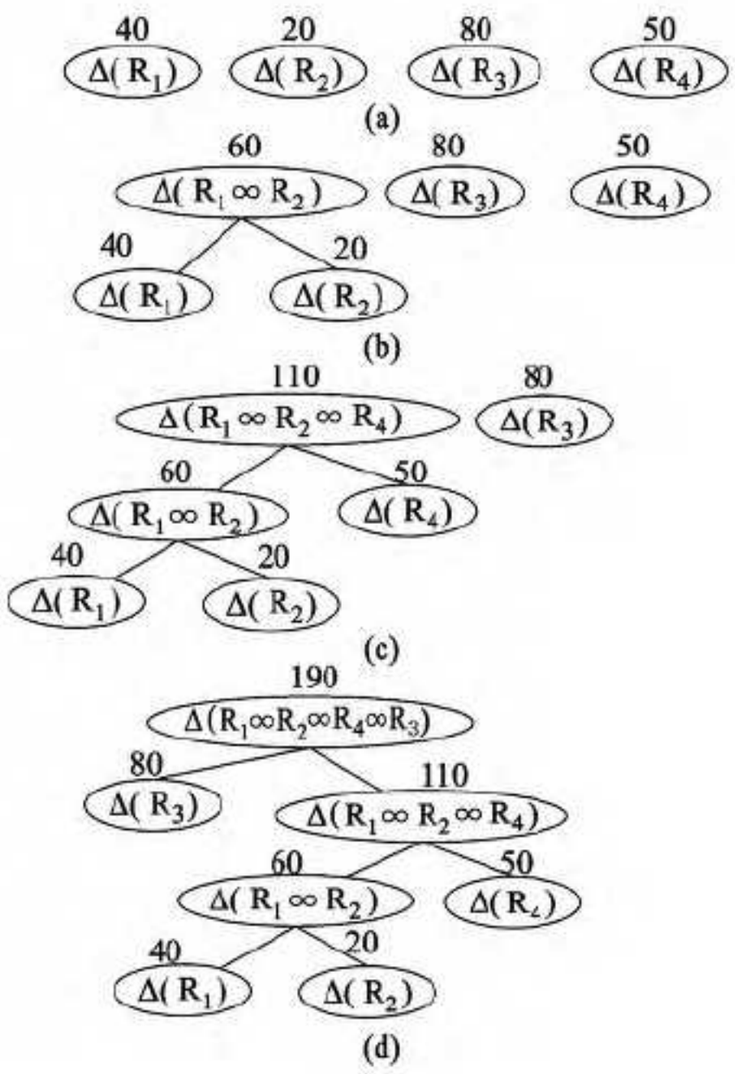

图 1 构造最优 Delta 传播树的 BinPartition 算法

3 基于快速排序和 huffman 树的物化视图增量保持算法

BinPartition 算法能够在 $O(n\log n)$ 的时间内构造出最优的 Delta 传播树。但该时间只是在构造树上所花费的时间开销。从其算法描述和构造最优 Delta 传播树的过程可以看出，要构造树，首先就必须要比较参与结点的权值大小，然后选择权值最小的两个结点进行树的构造，再循环地去比较、选择剩余结点中权值最小的结点进行树的构造。这个循环比较、选择的过程实际上花费了很多时间。那么，就可以通过提高步骤(2)①中的选择时间来对该算法进行改进，从而提高整体的效率。由此，我们在该算法中加入了排序的思想。

现有的排序算法主要有直接插入、直接选择、冒泡排序、快速排序、堆排序和归并排序等。快速排序是目前基于比较的内部排序中被认为是最好的方法之一，当待排序的关键字是随机分布时，快速排序的平均时间最短。因此，本文在 BinPartition 算法中加入快速排序的思想，提高选取左右子结点的速度。改进后的算法其形式化描述如下：

(1) 对给定的一组关系 $\{R_1,R_2,\cdots,R_n\}$（n 为关系的数目），以它们的变化 $\Delta(R_1),\Delta(R_2),\cdots,\Delta(R_n)$ 作为结点并将 $|R_1|,|R_2|,\cdots,|R_n|$（$|R_i|$ 表示关系的大小，即关系中的元组数）视为相应的权值，构造具有 n 棵二叉树的森林 $F=\{T_1,T_2,\cdots,T_n\}$，其中每棵树 T_i 只有一个权为 $|R_i|$ 的根结点 $\Delta(R_i)(1\leqslant i<n)$，此时它的左右子结点均为空。

(2)应用快速排序方法对二叉树森林 $F=\{T_1,T_2,\cdots,T_n\}$ 按照权值大小进行排序（从小到大或从大到小）。

(3)重复以下步骤，直至 F 中剩下一棵树为止：

①在 F 中选取两棵根结点的权值最小的二叉树 T_i（设根为 $\Delta(R_s)$）和 $T_j(\Delta(R_t))$ 作为左、右子树构造一棵新的二叉树 T'。T' 的根为 $\Delta(R_s\infty R_t)$，权值为其左、右子树的根结点权值之和；

②从 F 中删去 T_i,T_j；

③将 T' 加入 F 中适当的位置，保证 F 的顺序正确，返回步骤①。

同样以一个由 $n=4$ 个关系组成的连接 $\{R_1, R_2, R_3, R_4\}$ 来描述该算法的具体构造过程。假设各关系的权值分别为 $|R_1|=40, |R_2|=20, |R_3|=80, |R_4|=50$。基于快速排序和 huffman 树的物化视图增量保持算法构造最优 Delta 传播树的过程如图 2 所示。

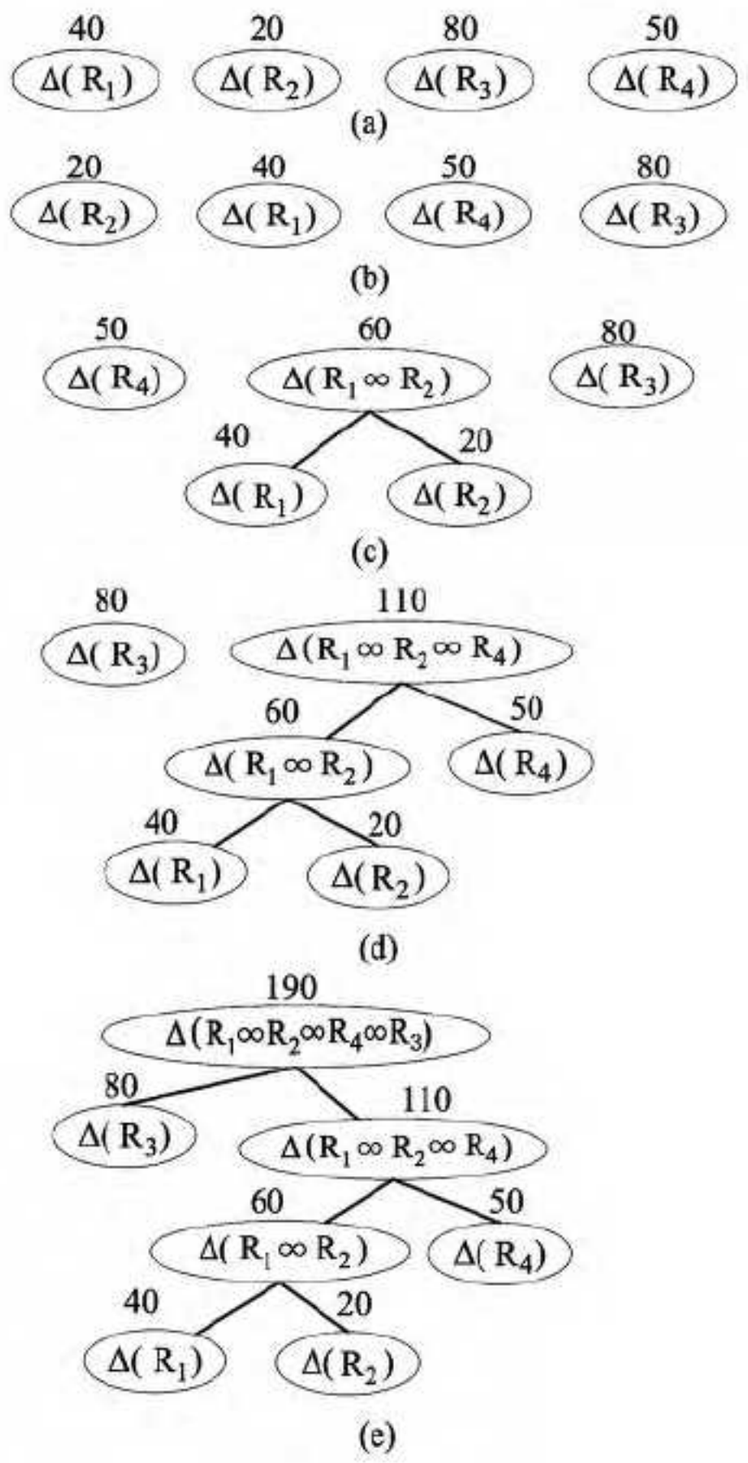

图 2　改进的构造最优 Delta 传播树算法

可以看出，经过了(a)—(e) 5 个步骤后，(e)即为最终生成的最优的 Delta 传播树，R_3 作为左子结点或右子结点的效果相同，不影响最终的结果。对比两个算法的构造过程可以看出，其最终构造出的结果完全相同。改进的算法中虽然多了一个排序的步骤(b)，但之前的算法由于每次都要从剩余的结点中重新去计算、比较，然后选择最小的值，将花费更多的时间。下面的实验也验证了这一点。

4　实验对比

我们首先通过实验 1 证明加入快速排序的 huffman 树构造算法的效率比普通的 huffman 树构造算法高，然后通过实验 2 证明基于快速排序和 huffman 树的物化视图增量保持算法比构造最优 Delta 传播树的 BinPartition 算法效率高。

进行对比测试的基本实验环境为：数据库系统版本：Oracle Database 11g Release 2 Standard Edition for Microsoft Windows x64；操作系统：Microsoft Windows Server 2003 Enterprise x64 Edition；处理器：Inter(R) Core(TM) i3-3220 CPU @3030GHz 3.29GHz，内存：4G，硬盘：500G；开发平台：Microsoft Visual Studio 2012。

实验 1　验证加入快速排序的 huffman 算法效率高于一般的 huffman 算法。

文献[5]中的构造最优 Delta 传播树的 BinPartition 算法基于一般的 huffman 算法，该算法在比较选择最小权值的结点上所花费的时间较多。本文在 huffman 算法的基础上，加入了快速排序的思想，希望提高在选择结点上的效率。实验用一般的 huffman 算法构造 12 组数据，然后再用改进的加入快速排序的 huffman 算法构造相同的 12 组数据，以查看各组的时间开销。每次实验都重复 10 次，最后取平均值。实验结果如表 1 所列。

表 1　12 组不同数目结点 huffman 树算法性能比较

序号	结点个数	一般 huffman 时间开销(ms)	改进的 huffman 时间开销(ms)
1	200	3.2	2.6
2	500	12.6	3.4
3	1000	44.4	3.8
4	2000	193	6
5	3000	400	8.4
6	4000	694.4	8.6
7	5000	1317.2	11.6
8	6000	1528.8	12.6
9	7000	2059.4	16
10	8000	2678.2	17.4
11	9000	3410	22
12	10000	4225.8	26.6

从表 1 可以看出，加入了快速排序算法的 huffman 算法的效率高于一般的 huffman 算法。特别是当构造的结点数越大时，一般的 huffman 算法构造时间为几千毫秒，而改进的 huffman 算法则保持在几十毫秒以内，其差别是非常明显的。

实验 2　验证改进的基于快速排序和 huffman 树的增量保持更新算法效率高于一般的基于 huffman 树的构造最优 Delta 传播树的 BinPartition 算法。

定义 $V=(TIME \infty TVTYPE \infty FAULTTYPE \infty PRODUCTTYPE \infty TV\_PRODUCT\_APPEARANCE)$。物化视图是由 5 个基本关系组成，其数据来源于“生产线质量控制系统”。

我们将构造最优传播树的 BinPartition 算法与改进的基于快速排序和 huffman 树的物化视图增量保持算法进行性能比较，当基本关系分别产生 2%、6%、10% 的增量时，运用上述不同算法进行视图 V 的更新所需的时间如图 3 所示，时间单位为秒(精确到毫秒)。

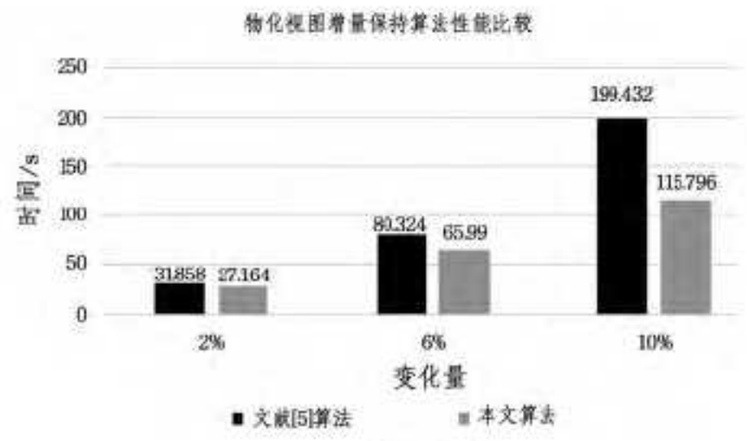

图 3　性能比较

从图 3 中可以看出，改进的基于快速排序和 huffman 树的物化视图增量保持算法在构造最优传播树的 BinPartition 算法的基础上效率有了一定程度的提高，特别是当物化视图的变化量比较大时，改进的算法有比较明显的提高。

结束语　本文通过实验证明了结合快速排序和 huffman 思想的改进的构造最优 Delta 传播树的物化视图增量保持算

法比以一般 huffman 思想为基础的构造最优 Delta 传播树的物化视图增量保持算法在效率上有所提高。本文实验基于生产线质量控制系统，下一步我们将把该算法应用到其他的实验环境中，进一步验证算法有效性的同时对算法进行进一步的优化。

参 考 文 献

[1] Blakeley J A, Larson P, Tompa F W. Efficiently Updating Materialized Views[C]// Proceedings of ACM SIGMOD Conference. 1986:61-71

[2] Gupta A, Mumick I S, Subrahmanian V S. Maintaining views incrementally[C]// Proceedings of ACM SIGMOD Conference. 1993:157-166

[3] Lee K-Y, Son J-H, Kim M-H. Efficient Incremental View Maintenance in Data Warehouses[C]// Proceedings of CIKM01. Atlanta Georgia USA, 2001

[4] 王新军，洪晓光，孙明，等. 物化视图增量保持的改进算法[J]. 计算机工程，2003，29(21)

[5] 王新军，洪晓光，王海洋，等. 数据仓库中多源物化视图的一种有效更新算法[J]. 计算机研究与发展，2004，41(5)

[6] 孙建青，李风云. 多源物化视图更新的一种流水线并行算法[J]. 山东师范大学学报，2004，19(1)

[7] Zhou Jing-ren, Larson P, Elmongui H G. Lazy Maintenance of Materialized Views[C]// VLDB'07. ACM, Vienna, Austria, September 2007

[8] Zhou Li-juan, Shi Qian, Geng Hai-jun. The minimum incremental maintenance of Materialized Views in Data Warehouse[C]// 2nd International Asia Conference on Informatics in Control, Automation and Robotics. 2010

[9] Nica A. Incremental maintenance of Materialized Views with outerjoins[J]. Information Systems, 2012, 37:430-442

[10] 武彤，赵雪，赵洵. 动态更新实物化视图以提高 OLAP 查询效率[J]. 计算机科学，2012，39(6A)

[11] Christopher J V W. Data Structures and C Programs [M]// Reading. Mass.: Addison-Wesley, 1988

[12] [美]Thomas H, Cormen Charles E, Leiserson Ronald L, 等. 算法导论(第三版)[M]. 殷建平，徐云，王刚，等译. 机械工业出版社，2013

(上接第 450 页)

函数调用关系主要填写两张表格：FunctionInfo 表用于存储 export 函数、变量与文件的对应关系；FunctionCall 表用于存储函数之间的调用关系。

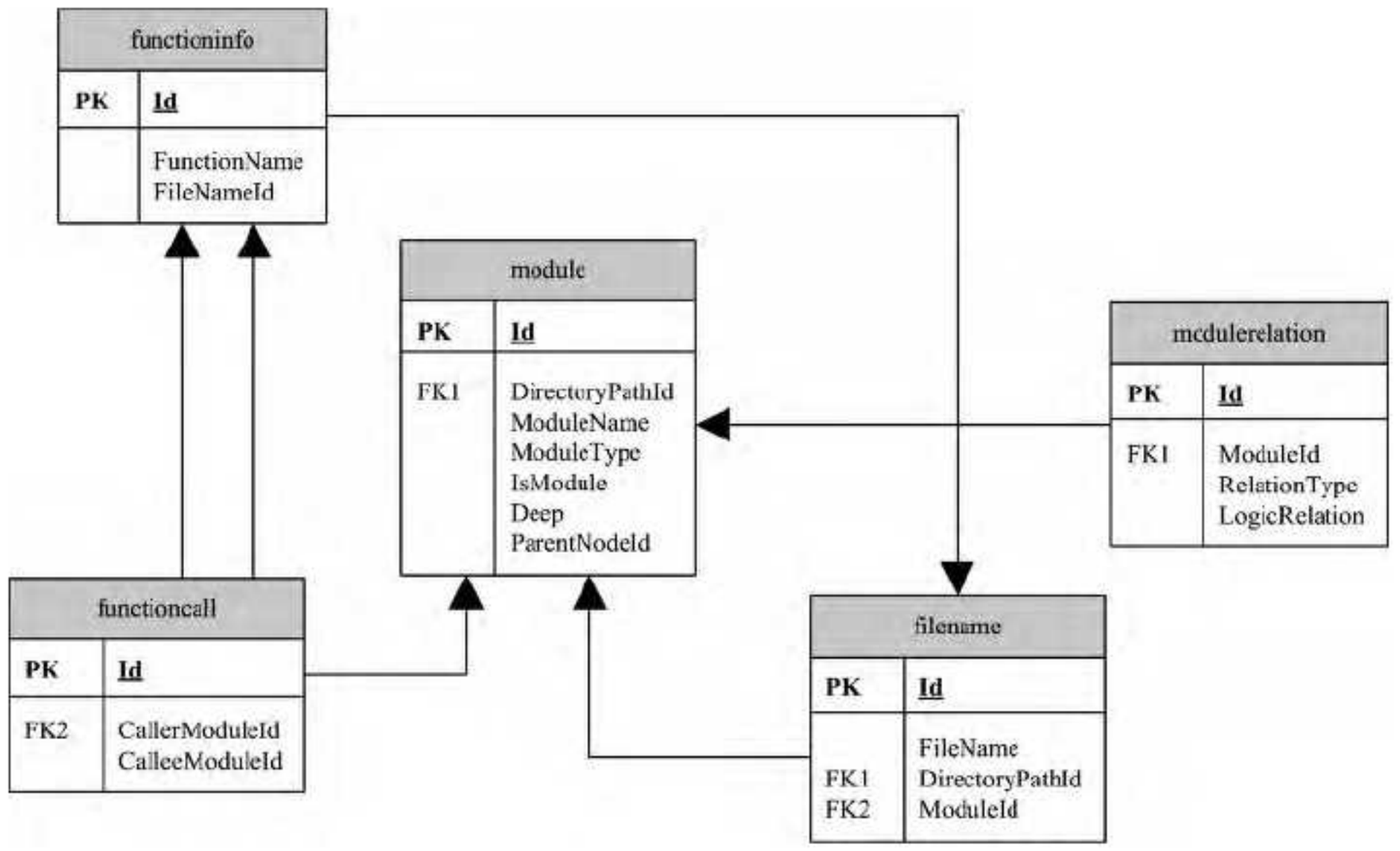

图 19 Linux 内核源码依赖关系 E-R 图表示

以上 ER 图描述可以看作是针对 Linux 内核源码进行了分层模型的建立。从顶层抽象描述了编译选项的 module 实体不仅可以获得本层次之间的关系(moduleRealtion)，也可以获取到其与下层文件以及函数之间的关系。即由以上描述，可以从编译选项的依赖得出文件的依赖，进而追溯到函数调用之间的依赖，顶层抽象的宏观的关系最终落实到了底层的具体的关系之上。

结束语 本文基于 Kconfig 与 Makefile 这两种在工程中大量使用的工具的语法，介绍了一种解析 Linux 内核源码中编译选项、文件以及函数之间的依赖关系的方法，同时为大型软件的源码的静态分析、阅读和学习提供了一种思路。另外本文还提供了编译选项、文件以及函数之间关系展示的简单样例。本文的不足之处在于，所有分析只局限于静态，一些动态运行时才会体现的关系难以通过现有手段进行捕获与存储，希望在以后的研究中能够继续完善。

参 考 文 献

[1] Linux Kernel Organization, Inc. Kconifg Introductionin Linux Kernel Source Code[OL]. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

[2] Linux Kernel Organization, Inc. Makefile Introductionin Linux Kernel Source Code[OL]. https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt

[3] Linux Kernel Organization, Inc. Building External Modules in Linux Kernel[OL]. https://www.kernel.org/doc/Documentation/kbuild/modules.txt