行列混合存储数据库系统的研究*

孙林超，陈　群，肖玉泽，白　松
（西北工业大学 计算机学院，西安 710072）

摘　要：通过研究列存储技术的特点，提出了一种行列混合存储数据库系统的设计方案。该方案在存储层设立独立的行存储引擎和列存储引擎，采用早物化技术在数据读出之后将列表转换成行表，然后以行的形式完成后续处理。因此，该方法既获得了列存储的读优势又复用了行数据库系统的成熟部件，降低了开发的风险和复杂度。基于 PostgreSQL 的原型开发与测试证明了该方案的可行性和有效性。

关键词：数据库管理系统；行列混合存储；存储引擎
中图分类号：TP311.13　　**文献标志码**：A　　**文章编号**：1001-3695(2013)02-0480-03
doi:10.3969/j.issn.1001-3695.2013.02.044

Research of row-column mixed storage DBMS

SUN Lin-chao, CHEN Qun, XIAO Yu-ze, BAI Song
(*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710072, *China*)

Abstract: By analyzing the technical features of column store, this paper proposed a design of mixed storage DBMS. On the storage level, both row engine and column engine were built. After the column table was read, it was transformed to row table, so that subsequent processing could be done. Therefore, the design gets the advantages of column store as well as reuses the mature parts of row DBMS, reducing the risk and complexity of the development. PostgreSQL-based prototype and testing prove the feasibility and effectiveness of the proposed design.

Key words: DBMS; row-column mixed storage; storage engine

随着社会信息化程度的不断加深，不断累积的海量数据和不断增长的数据膨胀速度对数据库系统提出了新的需求。目前，基于数据库的应用主要分为 OLTP 和 OLAP 两类，前者需要处理涉及频繁写操作的事物型查询，后者侧重于处理涉及大量读操作的分析型查询。最新的研究表明，列存储在读操作上有着较大的优势，非常适用于 OLAP 查询[1]，然而其对写操作的支持并不理想，因此并不适合 OLTP 查询；另一方面，多年的实践表明，传统的行式数据库对 OLTP 查询支持得非常好。由于上述原因，当前面向 OLTP 数据库和面向 OLAP 的数据库往往彼此独立。然而很多企业和组织（银行、电子商务企业等）同时拥有这两种业务，不得不部署两套系统。这种做法不仅增加了企业成本，而且在实际使用中要在不同数据库之间进行数据迁移，不利于数据的管理和使用。因此，构建一种既支持列存储，具有良好的读性能，又支持行存储，具有良好写性能，从而能同时支持 OLTP 和 OLAP 应用的混合存储数据库有着巨大的现实需求。

针对上述问题，笔者在深入研究列存储技术和数据库系统体系结构的基础上，提出一种基于现有关系型行数据库构建行列混合存储数据库的方案。首先，在传统行式关系数据库系统的存储层（以下简称存储引擎）加入一个列存储模块，构成混合存储引擎；然后，在访问数据时采用早物化技术将列表的目标属性列转换成元组的形式，利用行数据库的查询引擎完成后续处理。同时，利用选择下压技术将选择操作下压到物化操作之前进行，从而在物化之前过滤数据，减少从磁盘读取的数据量、提高查询效率。从外部看，该数据库既支持按行存储，也支持按列存储；用户可以在创建表时指定存储方式，以获得相应的读或写性能，进而满足其业务需求。在此基础上笔者基于开源数据库 PostgreSQL 进行了原型开发和测试，证明了该方案的可行性和有效性。

1　相关工作

在关系数据库系统中，数据是在逻辑上组织成表的形式，存取表的性能是影响数据库系统性能的一个重要因素。表的存储模式分为行存储和列存储两种。行存储模式下数据按元组存储，每个元组的所有属性都存储在一起，如果要查询一个元组的某个属性值，则需要先读取整个元组的数据。当前大多数的关系数据库系统都采用行存储技术。列存储将表的每一列组织在一起进行存储，不同的列独立存储。行存储与列存储的逻辑结构如图 1 所示。

行存储和列存储技术有各自的优缺点：

a）采用行存储技术时，数据按照元组直接进行存储，写的效率较高；采用列存储技术时，元组必须首先拆分成独立的属性列，再独立存储，写效率较低[2]。

b）查询密集型应用中绝大多数查询一般只涉及少数几个

收稿日期：2012-06-14；**修回日期**：2012-07-23　　**基金项目**：国家自然科学基金重点资助项目（61033007）；华为创新研究计划资助项目（IRP-2011-02-03）

作者简介：孙林超（1987-），男，陕西宝鸡人，硕士研究生，主要研究方向为数据库、并行计算（wrong1121@163.com）；陈群（1976-），男，福建莆田人，教授，博导，主要研究方向为 RFID 数据管理技术、XML 数据管理技术和 Web 服务技术；肖玉泽（1989-），男，四川达州人，硕士研究生，主要研究方向为数据库、软件工程；白松（1989-），男，陕西蓝田人，硕士研究生，主要研究方向为数据库、海量存储.

属性。按行存储时,必须读取整条记录,当属性较多时读取数据代价较大;按列存储时,只需读取所需要的属性列,可大大减少读取数据的代价[3]。

c)采用列存储时,可以获得较高的压缩率[1];行存储不利于压缩。数据压缩可以减小存储空间,减少读操作时的 I/O 量,但写数据时需要进行解压和再压缩,代价较大[3]。

d)关系型数据库系统最终以表的形式向用户返回查询结果,因此采用列存储时必须将分开存储的属性列重新组织成元组,即进行物化操作。根据元组物化操作的执行时机可分为早物化和延迟物化两类。早物化是在元组从磁盘读出之后立即执行,查询过程依照行数据库的模式进行;延迟物化则将物化过程当做一个标准操作,由优化器决定物化操作的时机,可以获得较好的查询效率[4]。采用行存储则不需要进行物化操作,而是进行投影操作,即从整个元组中分离出目标属性列。

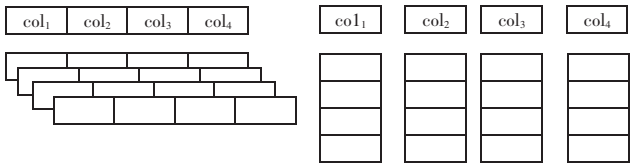

图 1　行/列存储逻辑结构对比

单独基于行存储技术和列存储技术的数据库系统只能满足读优化和写优化之一,不能很好地解决读写同时优化的问题,因此基于行列混合存储的数据库系统是必要的。目前国内外行列混合存储数据库的研究还比较少。在传统行式数据库上实现行列混合存储的主要难点在于两种存储技术对文件的组织方法截然不同,因此它们的查询执行方法也有很大的差别,难以统一。针对该问题,本文提出了一种基于早物化技术的方案。在数据库系统的存储层分别设立一个行存储引擎和一个列存储引擎,然后由一个访问接口层将两者封装起来,对列表进行元组物化,对行表进行投影,向查询引擎提供统一的数据访问接口,从而隐藏存储差别,实现查询处理的统一化。

2　行列混合的存储引擎

定义 1　存储引擎。数据库系统中负责与文件系统交互,完成数据读写的部件。

定义 2　查询引擎。数据库系统中负责执行查询计划、完成查询处理流程的部件。在执行查询计划的过程中,查询引擎访问存储引擎以获取数据。

2.1　行列混合的存储模型

本文提出的行列混合数据库系统是指数据库系统中一个表可以选择按行或者按列中一种方法存储,每个表在数据库中仅存储一份,用户在创建表时根据需求指定表的存储方式以获得相应的读写性能。从体系结构上数据库系统可分为查询分析和查询执行两层,查询分析层负责查询计划的制定与优化,执行层根据查询计划访问数据完成查询。执行层包括存储引擎和查询引擎,存储引擎负责直接访盘实现数据读写;查询引擎负责整体流程控制。在行列混合存储数据库系统中,按表存储类型可将查询分为行表查询、列表查询和行列混合查询三类。由于数据存储模式的不同,它们的查询执行方法也有很大差异。设置三个不同的查询引擎虽然可以实现功能,但是开发的复杂度和风险都非常大,因而可行性较小。

本文提出了一种基于早物化技术的解决方案,整个系统框架如图 2 所示。存储引擎由行列两个独立的子存储引擎构成,它们分别负责行式文件和列式文件的访问;然后接入存储引擎分装层,从而向上隐藏存储细节,提供一致的数据访问接口。查询引擎访问存储引擎时提供四个访问参数〈dataFileID,storeType, attList,conditionList〉(表 1 给出了各参数含义),存储引擎封装层根据参数 storeType 选择行存储子引擎或列存储子引擎完成数据访问。行存储引擎根据参数 attList 和 conditionList 对数据进行选择和投影,列存储引擎则进行选择和元组物化。存储引擎与查询引擎间的数据交互以元组(仅包含目标属性)的形式进行,从存储引擎以上的部分来看,表都是以行的方式存储的,因此查询分析和执行过程可以复用行式数据库的成熟逻辑。对于数据插入,查询引擎将数据元组连同访问参数传给存储引擎,如果 storeType 值为 ROW,则由行存储子引擎将数据元组作为整体写入表文件;如果 storeType 值为 COLUMN,则在封装层拆分,然后由列存储子引擎将数据写入各自的文件。

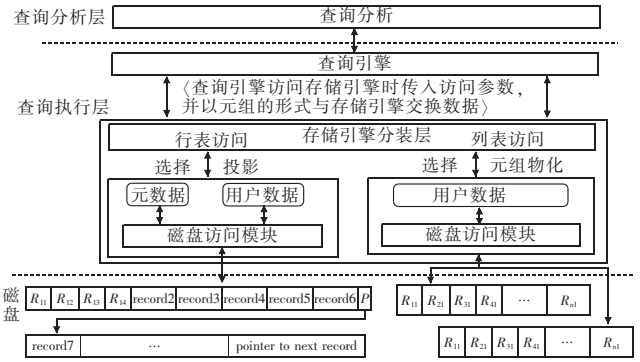

图 2　行列混合存储框架

表 1　访问参数说明

参数名	含义
dataFileID	文件 ID,用以指定目标文件,由查询语句中的表名称解析而来
storeType	存储模式,用以指定数据访问方式
attList	属性列表,用以指定目标属性列
conditionList	选择条件列表,用以对数据进行选择,由查询语句的选择条件解析而来

2.2　文件组织方法

数据库中的数据在逻辑上表现为表,物理上以文件的形式存储于磁盘。本文提出的方案通过以下方法组织文件:

a)对于行表,同一表的数据存储在一个文件中,文件名为表名通过映射函数生成的 dataFileID。文件的内部结构可以完全继承相关行数据库的方法,这里不再赘述(笔者在开发原型时对行表直接采用 PostgreSQL 的既有方法进行存储)。

b)对于列表,表的每个属性列独立地组织成一个文件,每个表对应一组文件。文件名的格式为〈dataFileID-attID〉,其中 dataFileID 由表名经过映射得到,attID 由属性名映射得到。此外还有一个以 dataFileID 命名的文件,用以存储元组的公共描述信息(时间戳、版本号等)。在列表文件内部,首先在文件头部存储数据类型、数据长度等描述信息,之后连续地存储有效数据。列表在数据读出后需要进行元组物化操作,因此需要存储各属性列之间的对应信息。一种方法是在存储数据时加入定位列(int 型),然后通过在定位列上做连接操作来实现元组物化;另一种方法并不显式地存储定位信息,而是通过数据在文件中的相对位置定位,这种方法节省存储空间且有利于压缩,但是数据的插入、删除、更新操作过程比较复杂。笔者在进

行原型开发时采用第二种方法。

2.3 行列混合存储数据库的查询执行流程

前面论述了行列混合存储引擎的架构和文件组织方法,本节将描述基于该引擎的查询执行流程。在创建表时将表的存储模式作为元数据写入数据字典(数据字典是一组存储元数据的表,存储在行引擎中)。在查询语句的解析阶段查找数据字典,获取表的存储模式;然后结合查询分析的相关信息生成〈dataFileID,storeType,attList,conditionList〉四个访问参数。执行引擎在访问数据时将访问参数传给存储引擎,存储引擎根据参数选择适当的方法读取数据并进行选择、投影(或者物化)处理后返回数据。数据库系统对数据操作分为读和写两类,其宏观访问过程一致,只是在对数据的具体操作不同。因此笔者以读操作为例说明本文提出的行列混合数据库的查询处理过程(写操作与之类似)。

算法1 SELECT语句的查询分析算法

输入:查询语句 sql〈SELECT table. col1, table. col2 FROM table WHERE table. col1 = "mark;〉。

输出:查询计划 queryPlan。

```
tableName = analyse(sql). getTableName();
dataFileID = getFileID(tableName);/*获取 dataFileID*/
/*以表名 tableName 查数据字典,获取存储信息,包括 storeType、索引信息。数据字典以行表形式存储*/
tableInfo = accessDic(data-dictionary,ROW,all,tableName);
storeType = tableInfo. getStoreType()/*获取存储类型*/
attList = analyse(sql). getAttlist();/*获取目标属性列表*/
conditionList = analyse(sql). getConList();
/*获取选择条件列表*/
queryPlan = buildPlan();
/*按照行数据库的策略执行查询计划*/
queryPlan. add(dataFileID,storeType,attList,conditionList);
/*向查询计划中加入基本参数*/
```

算法2 存储引擎读操作的处理算法

输入:访问参数〈dataFileID, storeType, attList, conditionList〉。

输出:数据列表 dataList。

```
if(storeType == ROW)
/*读取行表时以元组为单位,读出后进行选择和投影,满足条件的插入 dataList*/
    while((Data = readRow(dataFileID,conditionList)) ! = EOF)
      data = project(data,attList);
      dataList. add(data);
    end while
endif
else
/*读列表时以属性列为单位,各列按其约束条件选择;目标列都读出后执行物化操作,然后插入 dataList*/
    for(i = 0;i < attList. length;i ++)
      col[i] = readCol(dataFileID,attList. get(i),conditionList. get(i))
    end for
    dataList = material(col,attList. length)
end else
```

3 测试分析

3.1 基于 PostgreSQL 的实现

PostgreSQL是一款开源的行式关系数据库,广泛地应用于Linux系统中。笔者基于PostgreSQL进行了原型开发以验证本文提出方案的有效性和可行性,主要从以下三个方面展开:

a)SQL接口层实现。在CREATED TABLE语句的末尾引入storeType row/column关键字,以指定表的存储类型。首先修改scan.l和gram.y文件中的语法识别规则,重新生成scan.c和gram.c从而识别storeType关键字,并提取存储类型信息。

b)查询处理层实现。在数据字典中引入pg_storetype表,以存储该数据表的存储类型信息。pg_storetype表包含两个属性列〈tableID,storeType〉,它们的数据类型分别为oid(PostgreSQL定义的表ID数据类型)和bool。storeType为true表示按行存储,为false表示按列存储。定义两个访问pg_storetype表的函数:

```
void setStoreType(oid relationID, bool storeType);/*用于向 pg_storetype 表写入新建表的存储类型信息*/
bool getStoreType(oid relationID);/*用于从 pg_storetype 表读取的存储类型信息*/
```

定义数据结构columnFileInfo以向文件访问层传递列表的访问信息:

```
struct{
  oid dataFileID;/*由表名映射而来的文件序列号*/
  int attLength;/*用以记录该表的投影列数量*/
  LIST * att_list;/*存储各属性列的映射值*/
}
```

在relation结构体中定义一个columnFileInfo类型的变量columnInfo以及一个列存储标志isColumn。语义分析完成后用从数据字典读取存储类型信息和查询指定的投影属性列对其进行赋值。

c)文件访问层实现。文件访问层主要用以实现列表文件的创建和读写。PostgreSQL中新文件的创建由relationSetNewRelfilenode函数完成,对文件的访问由heap_begincan_internal函数完成,这两个函数依靠relation类型的参数定位文件。经过对relation结构的修改使得其已经包含列存储信息,然后在上述两个函数中加入基于relation.isColumn的判断模块。如果是行表,则按照原来方式进行;如果是列表,则调用新定义的columnBuild和columnScan函数来完成列表文件的创建和读写。

3.2 实验与分析

笔者基于上述原型作了相关测试,以验证该系统是否同时具备读和写的优势。实验涉及两个数据表student_col和student_row,前者为列表,后者为行表;它们的schema相同,包括八个属性列〈id,name,age,gender,region,native, phonenumber,IDcard〉。其中id,age为int类型,gender为smallint类型,其余为varchar类型。在两个表上进行INSERT和SELECT操作以比较不同存储模式下系统的读写性能。

实验在Ubuntu 10.10系统上进行,系统处理器为AMD Athlon 5000B double 2.6 GHz,2 GB内存;实验数据为50万条仿真记录。其中INSERT语句写入所有的八个属性列,SELECT语句投影在〈id,name,phonenumber,idcard〉四个属性列上。实验结果如表2所示,列表上INSERT操作比行表多耗时114.69%,但其SELECT操作比行表节约28.49%的执行时间。对于一次数据导入后只做读操作的OLAP应用而言,采用列表牺牲一定的写性能换取读性能是值得的;而且在列表上采用轻量级压缩技术可以进一步提高读性能。对于OLTP应用,采用原有的行存储更为有效。实验表明该原型系统基本实现了设计目标,证明了本文所提出的方案是可行、有效的。

表2 查询运行时间表

操作	目标表	耗时/s
INSERT	student_row	19.26
INSERT	student_col	41.33
SELECT	student_row	15.27
SELECT	student_col	10.92

4 结束语

本文提出的设计方案实现了行存储和列 (下转第486页)

此外,AO方案的设计还可以避免因开发人员的疏忽而遗漏了对某个保单或某个属性的跟踪。由上述分析不难得出,AO方案的体系结构设计比non-AO更为合理。

4 结束语

软件体系结构的评估对系统的质量起着至关重要的作用。本文在现有的体系结构度量研究成果的基础上,利用场景技术提出一种支持面向方面体系结构的度量方法。运用面向方面体系结构描述语言AC2-ADL对某保险案例的体系结构进行描述,并提出基于AC2-ADL概念的度量指标,通过这些指标从组件和场景两个角度对该案例进行度量。案例研究进一步证明了本方法的可行性,为该方法的实际应用提供了一定的参考。

在未来的工作中,本文将进一步完善度量指标,探索该方法在其他质量属性中的应用,如可适应性、可扩展性等,从而使本方法更加合理、更加科学。

参考文献:

[1] 梅宏,申峻嵘.软件体系结构研究进展[J].软件学报,2006,17(6):1257-1275.

[2] BANANI R,GRAHAM N. Methods for evaluating software architecture:a survey,Technical Report 2008-545[R]. Kingston:Queen's University,2008:1-82.

[3] 张莉,高晖,王守信.软件体系结构评估技术[J].软件学报,2008,19(6):1328-1339.

[4] 宋光宇.软件体系结构度量综述[J].计算机与数字工程,2008,36(1):41-44,69.

[5] 文静,应时,张琳琳,等.面向方面的体系结构描述语言AC2-ADL[J].计算机科学,2009,36(8):126-132.

[6] BREIVOLD H P,CRNKOVIC I. A systematic review of software architecture evolution research[J]. Information and Software Technology,2012,54(1):16-40.

[7] KAZMAN R,ABOWD G,BASS L,*et al*. Scenario-based analysis of software architecture[J]. IEEE Software,1996,13(6):47-55.

[8] HEIKO K. Sustainability evaluation of software architectures:a systematic review[C]//Proc of the 7th International ACM SIGSOFT Conference on the Quality of Software Architecture. New York:ACM Press,2011:3-12.

[9] DOBRICA L,NIEMELA E. A survey on software architecture analysis methods[J]. IEEE Trans on Software Engineering,2002,28(7):638-653.

[10] RICK K,MARK K,MARIO B,*et al*. The architecture tradeoff analysis method[C]//Proc of the 4th International Conference on Engineering of Complex Computer System. Washington DC:IEEE Computer Society,1998:68-78.

[11] TEKINERDOGAN B. ASAAM:aspectual software architecture analysis method[C]//Proc of the 4th Working IEEE/IFIP Conference on Software Architecture. Washington DC:IEEE Computer Society,2004:5-14.

[12] BENGTSSON P,LASSING N,BOSCH J,*et al*. Architecture-level modifiability analysis[J]. Journal of System and Software,2004,69(1-2):129-147.

[13] 刘霞,李明树,王青,等.软件体系结构分析与评价方法评述[J].计算机研究与发展,2005,42(7):1247-1254.

[14] 孙昌爱,刘超,金茂忠.基于场景的软件体系结构分析[J].计算机工程与应用,2000,36(9):75-79.

[15] 沈群力,刘杰.基于场景的两种软件体系结构评估方法[J].计算机应用研究,2008,25(10):3015-3017.

[16] CHIDAMBER S R,KEMERER C F. A metrics suite for object oriented design[J]. IEEE Trans on Software Engineering,1994,20(6):476-493.

[17] BURROWS R,TAIANI F,GARCIA A,*et al*. Reasoning about faults in aspect-oriented programs:a metrics-based evaluation[C]//Proc of the 19th IEEE International Conference on Program Comprehension. Washington DC:IEEE Computer Society,2011:131-140.

[18] 宋光宇,高晖.软件体系结构度量工具的研究与实现[J].计算机应用研究,2008,25(9):2700-2702.

[19] GARCIA-MAGARINO I,COSSENTINO M,SEIDITA V. A metrics suite for evaluating agent-oriented architectures[C]//Proc of the 25th ACM Symposium on Applied Computing. New York:ACM Press,2010:912-919.

[20] SANT'ANNA C,GAMEZ N,GARCIA A,*et al*. General architecture evaluation process/metrics,AOSD-Europe Deliverable D85[R].[S.l.]:AOSD-Europe,2007.

[21] SANT'ANNA C,LOBATO C,KULESZA U,*et al*. On the quantitative assessment of modular multi-agent system architectures[C]//Proc of International Conference on Multiagent Systems and Software Architecture. London:Springer-Verlag,2006:111-135.

[22] YE Peng,NI You-cong,HU Ming. Research on measurement technique for evaluating adaptability of aspect-oriented software architecture[J]. Advanced Materials Research,2011,268-270:1307-1312.

[23] CLEMENT A,COLYER A,HARLEY G,*et al*. Using eclipse aspectJ:your first steps[EB/OL]. (2005-01-21). http://www.informit.com/articles/article.aspx?p=357692.

(上接第482页)存储的融合,在功能上能够同时满足OLTP和OLAP的技术需求;并且通过原型开发和实验证明了其可行性和有效性,对行列混合存储数据库作了有益的探索。然而早物化技术在一定程度上限制了列存储的读性能,笔者将进一步研究以迟物化方式在现有行式数据库中融入列存储的方法,以进一步提高行列混合数据库系统的性能。

参考文献:

[1] 李超,张明博,邢春晓,等.列存储数据库关键技术综述[J].计算机科学,2010,37(12):1-7.

[2] STONEBRAKER M,ABADI D J,BATKIN A,*et al*. C-store:a column-oriented DBMS[C]//Proc of the 31st International Conference on Very Large Data Base.[S.l.]:VLDB Endowment,2005:553-564.

[3] ABADI D J,MADDEN S R,HACHEM N. Column-stores vs row-stores:how different are they really?[C]//Proc of ACM SIGMOD International Conference on Management of Data. 2008:967-980.

[4] ABADI D J,BONCZ P A,HAIZOPOULOS S. Column-oriented databases systems[C]//Proc of the 35th International Conference on Very Large Data Base.[S.l.]:VLDB Endowment,2009:1664-1665.

[5] MacNICOL R,FRENCH B. Sybase IQ multiplex-designed for analytics[C]//Proc of the 30th International Conference on Very Large Data Base.[S.l.]:VLDB Endowment,2004:1227-1230.

[6] 于胜利,张延松,王珊,等.基于行存储型的模拟列存储策略的研究[J].计算机研究与发展,2010,47(5):878-885.

[7] BONCZ P,ZUKOWSKI M,NES N. MonetDB/X100:hyperpipelining query execution[C]//Proc of the 2nd Biennial Conference on Innovative Data System Research. 2005:225-237.

[8] DAI Xiao-lei,LI Chun-qing. The application of materialization strategies on OLAP in column oriented database systems[C]//Proc of the 3rd International Conference on Communication Software and Networks. 2011:305-308.

[9] IVANOVA M,KERSTEN M L,NES N. Self-organizing strategies for a column store database[C]//Proc of the 11th International Conference on Extending Database Technology. 2008:157-168.

[10] ABADI D J,MYERS D S,De WITT D J,*et al*. Materialization strategies in a column-oriented DBMS[C]//Proc of the 23rd IEEE International Conference on Data Engineering. 2007:466-475.

[11] SILBERSCHATZ A,KORTH H F,SUDARSHAN S. 数据库系统与概念[M].杨冬青,马秀莉,唐世渭,等译.北京:机械工业出版社,2008:307-312.