数据流系统中卸载技术研究综述

韩东红　王国仁
(东北大学信息科学与工程学院　沈阳 110004)

摘　要　随着数据流应用系统的快速流行,流数据管理对数据库技术提出了巨大挑战。由于数据流经常是爆发性的且数据特征可能随时变化,因此要求数据流管理系统具有很好的自适应性。当输入速率超过系统处理能力时,系统会产生过载且性能下降。为了解决这一问题,卸载技术是有效的途径之一。卸载时间、卸载地点和卸载数量是与卸载技术密切相关的三个主要问题,本文主要从这三个方面来综述和分析目前各个数据流系统所采用的卸载技术。

关键词　数据流,卸载,自适应性

A Survey of Load Shedding Techniques in Data Stream Systems

HAN Dong-Hong　WANG Guo-Ren
(College of Information Science and Engineering, Northeastern University, Shenyang 110004)

Abstract　With the rapid popular of data stream application systems, management of data streams is a huge challenge on database technology. Since data streams are often bursty and data characteristics may vary over time, it is required that data stream systems must have a good adaptability. When input rates exceed the system processing capacity, the system will become overloaded and the performance will deteriorate. To solve this problem, load shedding is one of promising approaches. When, where in the query network and how much load to shed are three main problems closely related to load shedding techniques. This paper surveys and analyzes from these three points load shedding techniques adopted in various existing data stream systems.

Keywords　Data streams, Load shedding, Adaptability

1　引言

近年来,处理大量输入数据流的新应用正变得越来越普遍,这些应用包括处理来自小的嵌入式传感器的数据、电信数据管理、金融分析、网络入侵检测、管理来自大量地理定位设备的输入数据等等。数据流应用的出现对数据库管理技术提出了巨大的挑战。首先,数据流中的数据以在线的方式到达,数据量极大,并且连续到达的数据的速度和特征是不可预见的。系统必须在资源有限的情况下,实时处理数据流中的数据。其次,传统数据库系统中的查询为作用在有限数据集上的一次查询,而作用在连续、无界的数据流上的查询为连续查询,即当数据流数据流经查询网络时,查询可能被连续执行运行很长一段时间。第三,当前数据库管理系统的体系结构采用基于拉(pull-based)的数据访问模型,而在面向流的应用中,同时可以采用基于推(push-based)的模型。为解决上述问题,出现了一类不同于传统数据库管理系统(DBMS)的数据流管理系统(DSMS),其中较为著名的系统有STREAM[1]、Aurora[2]、TelegraphCQ[3]、NiagaraCQ[4]、StatStream[15]、OpenCQ[18]等。

在数据流管理中遇到的主要挑战之一,是我们希望DSMS具有自适应的能力,以便在有限的系统资源(如CPU、内存、带宽等)环境中,在数据的速度及特征不可预知的情况下支持实时处理。数据流源具有相当高的突发性[5~7],这就意味着在长时间运行的连续查询当中,数据的速度和系统的负载可能具有高度的可变性。当数据的速度低于系统的处理能力时,系统被称之为稳定的;当数据的速度超出系统的处理能力时,系统被称之为饱和的或不稳定的。当出现后一种情况时,系统会出现过载并且性能下降,此时,可以通过增加更多的源或将计算分布到多个节点上,但这种方法在经济上可能是不可行的。另外一种行之有效的方法是卸载,即丢掉部分尚未处理的数据,以便使系统减轻负载,恢复正常工作。

在网络[13]和多媒体研究领域[14],早就开始了对卸载问题的研究,并且与数据流中的卸载有许多相似之处,但也存在一些本质上的区别:第一,在数据流的卸载当中,查询网络是集中控制的,不需要卸载的分布算法;第二,在后者的研究中,需要考虑共享处理的应用;第三,查询节点不是简单地将数据发送到其它节点,而是在其上执行操作等等。

目前,在数据流管理中的卸载研究方面已经开展了一些工作。例如,Aurora系统[8]用到了卸载技术,它依赖的是详细说明处理每一个元组的服务质量函数。Aurora方法适用于对每一个单独元组的处理值是独立于其它元组的情况,不适于聚合查询,而文[10,22]则侧重于数据流上聚合查询的卸载处理。文[9]涉及到流上连接查询的卸载问题,文[16,17]侧重于数据流之间基于窗口连接的卸载处理。

卸载主要可以通过两种方法来实现,即随机丢弃(random drop)[9]和语义丢弃(semantic drop)[8]。卸载处理由以下三个关键问题组成。

1. 卸载时间。必须对查询网络的负载处理情况进行连续的评估,应能及时检测到系统存在的超载情况,并且实施卸载。

2. 卸载地点。在处理网络的任意点均可以丢弃元组。显而易见,尽量早地丢弃元组可以避免多余的工作。但同时

应注意，如果存在一个流扇出为多个流的情况，被过早丢掉的元组会对多个应用产生不利的影响。

3. 卸载数量。一旦确定了插入丢弃(drop)操作符的位置，接下来应该决定卸载的数量。在随机丢弃的情况下，应考虑丢弃元组的百分比。在语义丢弃的情况下，我们应该决定选用的谓词形式。

本文首先介绍三个应用卸载技术的数据流系统，然后基于这三个流系统从卸载时间、卸载地点、卸载数量、实现方法等几个方面进行总体的综述，最后给出结论和展望。

2 系统简介

STREAM[1]是由 Stanford University 研制的一个通用的基于关系的 DSMS 原型，用于支持作用在众多大容量和爆发式数据流上的大量复杂的连续查询，侧重于内存的管理和近似的查询处理。

NiagaraCQ[4]是 University of Wisconsin-Madison 和 Oregon Graduate Institute 联合开发的网络数据管理 Niagara 项目当中的可扩充的连续查询子系统，允许在动态的 Web 环境中建立连续的 XML-QL 查询。由于网络的可伸缩性，一个连续查询系统需要支持大量的查询，但目前不存在达到这种可扩充性的系统。为解决这个问题，NiagaraCQ 将连续查询分组，分组的原则是将许多共享相似结构的 Web 查询分为一组。分为一组的查询能够共享相同的计算，有效地减少 I/O 代价。

The Aurora Data Stream Management System[2,19,20]是由 Brandeis University、Brown University 和 M. I. T. 联合研制的用来处理数据流的系统，如图 1 所示。它是一个面向工作流的系统，允许用户通过安排 boxes(操作符)和 arrows(数据的流向)来建立查询计划。数据流是一个潜在的无穷元组队列，这些元组由一个数据源产生。可能的数据源如硬件传感器，它连续地将数据推向 Aurora。Aurora 可以处理大量查询，这些查询由 7 个简单的操作符组成的重复集合构成。一个查询可以处理任意数量的数据流，并经常以一个输出流结束。一个操作符可以被连接到多个下游的操作符。所有这样的分裂点能够向其下游的多个操作符传送同样的元组，并且能够在不同的查询之间共享计算。因为一个操作符能够接受多流输入，所以多流也能被合并。一个 Aurora 查询网络就是这样的查询构成的集合。实施卸载是由在查询网络中插入卸载操作符来实现的。

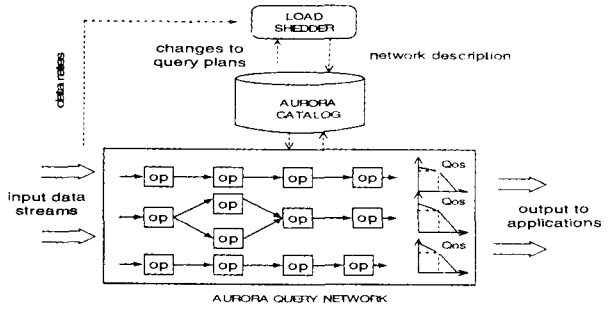

图 1 Load Shedder in Aurora

3 卸载时间

一般说来，当数据流中的数据到达速度超过系统的处理能力时，需要进行卸载操作。为此，应对当前的查询网络的负载情况进行连续的评估，以便及时检测到出现的超载情况，然后实施卸载处理。

不同的系统采用的检测方法也有所不同。在 Aurora 系统[8]中，当前网络的负载情况用简单的计算来评估，其中涉及到的参数包括当前的输入速度、操作符执行代价以及选择度。网络的每一个输入有一个相关的负载系数(load coefficient)，它代表了将一个输入元组通过网络推向输出所需要的处理机周期数。所有这些负载系数均可以静态地计算出来，然后，利用这些提前计算出来的系数值，在已知每个输入流的输入速度和输入流个数的情况下，我们可以计算出实际运行时的负载。运行时的负载表示单位时间内所有通过网络的元组所需要的处理机周期数，而系统的处理能力可以表示为单位时间内处理机周期数。如果这时的负载值超出系统的处理能力，则进行卸载处理。

检测负载是否超过系统处理能力的方法还有很多。例如，在文[9]的处理代价模型中，已知一个选择或投影操作符处理一个元组的执行代价，其中包括了读、检查条件和写出结果的时间，同时已知元组到达的速度。对选择或投影操作符而言，能够正确处理一个正在到来的元组的前提，是必须在下一个元组到来之前处理完当前元组。因此，在单位时间内，如果不能处理完到达的多个元组，表示相应的查询计划是不可行的。如果所有的查询计划均不可行，则需要进行卸载操作。

在文[10]提供的处理代价模型中，作用在若干个数据流 $S_1,\cdots,S_m$ 上的查询集合与查询操作符的集合组成了一个有向非循环数据流图(类似于文[2])。在 STREAM[1]中，模型中用到的一些参数值，如操作符的选择度、执行代价、流的速度等，由一个统计管理模块对这些参数进行评估。在系统中，元组被处理的速度至少和新的元组到达的速度一样，否则应执行卸载，以提高系统的吞吐能力。将这个要求表示为一个方程，即负载方程。基于负载方程可以决定是否进行卸载操作。在假设卸载的花费可以忽略的前提下，方程的左半部分给出了处理在一个时间单位内到达的元组所需要的全部时间。显而易见，这个处理时间至多为一个时间单位，否则，系统将不能及时处理不断到来的数据。

可以看出，文[9,10]在判断是否卸载的方法有相似之处，均是将处理单位时间内到达的所有元组所需要的时间与单位时间相比较，如果前者大于后者，则表示系统超载。文[10]利用单位时间内处理机周期数表示系统的处理能力，判断系统是否超载，利用处理单位时间内到达的全部元组所需要的处理机周期数与系统处理能力进行比较。

4 卸载地点

如果系统超载，接下来需要决定的是在查询计划的什么位置插入卸载操作符。元组可以在处理网络的任意点被丢弃。如何决定卸载的位置，有许多不同方法。

在 Aurora[8]系统中，做卸载处理时依赖的是服务质量(QoS)信息。假设为每个应用分别指定 QoS，它描述了一个答案的多个特征和利用率(utility)之间的关系。我们将 QoS 看作是输出的一个参数和它的利用率的函数集。例如，一个答案的利用率可以是输出值的函数，用来表达每一个值的重要程度。

首先，考虑与其它查询无共享部分的查询计划，见图 2 下图。在这种情况下，一个卸载操作符可以插在查询计划的任意点，同时不会影响其它的查询。但一般情况下，最好是将卸

载操作符越早插入越好，这样可以减少“下游”操作符的数据输入量，最大限度地减少多余的工作。所以，对没有共享的查询计划而言，最好的插入卸载操作符的位置总是在它的输入位置之后，第一个操作符之前。

然后，考虑带共享的查询网络。当一个操作符的输出扇出到多于一个的“下游”操作符，这些操作符导致不同输出的时候，共享就产生了。如图 2 上图所示。任何从操作符 1 出来的元组均被发送到输出 O_1 和 O_2。这样，卸载操作符插在 A 点将影响到两个输出，而插在 B 或 D 点只会影响单个的输出。如果 A 点之前还有其它操作符，卸载操作符应该推向“上游”的上一个分裂点处。所以，带有分裂点(B,D)的操作符输出处和 Aurora 的输入处(A,F)是插入卸载操作符的位置。对每一个候选位置，计算损失/收获率(Loss/Gain ratio)并按升序排序，Loss 是指从损失容差图得到的当每百分之一的元组被丢弃掉时失去的利用率，Gain 表示当每百分之一的元组被丢弃时获得的处理机周期数，最佳的插入卸载操作符的位置拥有最小损失/收获率。

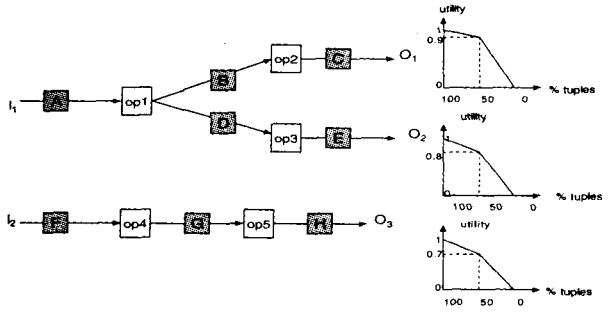

图 2　Example query network with candidate drop locations

在文[9]中，如果对一个只有若干个过滤器的连续查询计划，或者对一个连接 n 个流的连续查询计划，采用的卸载策略，亦是从数据流源流入处，即未被任何过滤器处理之前丢弃元组。

在文[10]中，系统可以预先计算出在每个查询路径中插入若干卸载操作符后对应的总的抽样比，即有效抽样比 P_i。如果查询中无共享的操作符，最佳方法是在每个查询 q_i 的查询路径中第一个操作符之前插入卸载操作符，且卸载操作符的抽样比等于此查询的有效抽样比。

当查询计划中有共享部分时，确定引入卸载操作符的位置相对比较复杂。在查询路径的共享部分加入卸载操作符，将对两个查询构成影响。在这种情况下，遵循的结论有三条。第一，在共享片段的起始处卸载，因为尽早卸载可以使整个的执行时间最小。第二，假设 q_{max} 在所有查询中拥有最高有效抽样比，这些查询有共享的部分(父片段)，B 为分支点。那么，B 的对应查询 q_{max} 的子片段中不包含卸载操作符，而所有其它的 B 的子片段包含卸载操作符，它们的抽样比为 P_{child}/P_{max}，这里的 q_{child} 为每一个子片段中拥有最高有效抽样比的查询。第三，假设 q_{max} 在所有共享最初片段 S 的查询中有最高的有效抽样比，则 S 将包含卸载操作符，其抽样比为 P_{max}。运用以上三个结论，就可以决定卸载位置和数量。这个策略可以通过简单的自顶向下的算法来实现，而且经证明，通过此算法得到的解决方案有最少的处理时间。

可以看出，当查询网络中无共享部分时，文[8～10]采用的方法均是在数据源流入处，即在第一个操作符之前插入卸载操作符，这样可以有效地减少多余工作，使系统具有最少的处理时间。文[8,10]同时考虑了查询网络中有共享部分的情况。文[8]采用的策略是源的流入处和带有分裂点的操作符输出处都可以插入卸载操作符。根据已知参数，计算出每个候选位置的损失/收获率，将卸载操作符插在具有最小损失/收获率的位置上，即在此处卸载后，将对服务质量的影响最小。文[10]采用的方法，是首先在共享片段的起始处卸载，其抽样比等于所有查询中最大的有效抽样比，然后再根据每个查询的有效抽样比来决定插入卸载操作符的位置和数量。

5　卸载数量

在确定了何时、何处卸载之后，接下来需要确定卸载多少元组。

文[9]中提出的目标，是保证卸载后输出速度最大。因为可以根据每个流的输入速度、操作符的选择度、卸载操作符的抽样比等参数计算出卸载后输出速度，决定卸载的数量实际上是在确保系统不超载的情况下，选择适当的卸载抽样比，使输出速度最大。

因为卸载是将尚未处理的元组丢弃，所以一定会对查询答案的正确性产生不利的影响，即产生近似答案。如何在复杂的系统中利用卸载技术，同时考虑减少对正确性的不利影响，在早期的工作中有所涉及[2,8,12,21]。文[10]中提出的目标，是将所有查询的最大相对误差最小化，同时证明了在最佳解决方案中，所有查询的相对误差是相等的。通过文中设计的自顶向下的算法和负载方程，可得到相对误差的值，继而计算出每个查询的有效抽样比 P_i 后，即可确定卸载的位置和数量。

文[8]保证在插入卸载操作符，丢弃掉一部分元组之后，系统的获得应大于其损失，即单位时间内获得的周期数应大于卸载操作符本身的代价。

可见，卸载的数量与系统提出的卸载目标关系密切，卸载目标包括卸载后输出速度最大、对答案精确度影响最小等。

6　实现方法

卸载时需要选择一些元组丢弃，可以通过两种常用的方法实现。最简单的是随机丢弃，即在系统超载时，在查询计划中插入随机卸载操作符，以便随机丢弃一部分元组。它的参数为 p，表示被丢弃元组的百分比。卸载后得到的查询结果为近似值，随机丢弃没有考虑丢掉的元组对结果的影响。为降低对系统性能的影响，卸载时应能识别和丢弃相对次要的数据。语义丢弃就是考虑了被丢弃元组内容的重要性，语义丢弃操作符本质上是过滤器操作符，其谓词的选择度为 $1-p$，丢弃的是内容最不重要的元组，如文[2]即采用了语义丢弃。

如果选用语义丢弃，在确定了语义丢弃操作符谓词的选择度之后，需要考虑选用何种谓词[8]。基于值的 QoS 图反映了元组某一属性值在不同区间的利用率情况，如果丢弃一些元组，应当从利用率最低的区间开始。输出数据的矩形图表示了相应 QoS 图中元组某一属性值在不同区间[0,50]出现的相对频率。损失容差图(loss-tolerance)由上述两个图得到，能够反映出当丢弃不同比例的元组时，利用率的变化情况。在已知卸载数量(在 5 中得出的丢弃比)和损失容差图的情况下，我们可以确定适当的谓词。例如，假设已经计算得到应丢弃 20%的元组，元组的某属性值区间[0,50]出现的相对频率为 40%，而且是利用率最低的区间。我们应该从此区间

丢弃20%的元组，即这个区间的一半数据。被丢弃的区间为[0,25)，插入的语义丢弃操作符谓词应为值≥25。如果需要丢弃70%的元组，则区间[0,50]将不能满足要求。此时，需要丢弃区间[0,50]的全部元组以及区间[51,100]元组的一半。插入的语义丢弃操作符谓词应为值75。

结论和展望 对计算机系统来说，能够适应操作环境的变化是非常重要的。特别是对监测连续的数据流，这一点显得尤为重要，因为数据流中的数据的速度和特征是不可预测地发生着变化。当数据的到达速度超出系统的处理能力时，需要卸载来保证系统正常运行。常用的卸载方法是在查询计划当中插入随机卸载操作符或语义卸载操作符。卸载时需要解决的关键问题是何时、何处卸载以及丢掉多少负载。卸载的目标可以是使系统的输出速度最大，或者是对答案的精确度影响最小等。研究的卸载问题涉及到聚合查询、连接查询等。

对数据流中卸载技术的深入研究工作包括：(1)对低负载系统的卸载处理。当数据速度太低时，元组也应该丢弃掉(超时)。下一步的工作应该包括研究当发生高负载与低负载时，进行卸载有哪些相同和不同；(2)如何将卸载算法扩展到其它的资源管理；(3)语义卸载的处理还应进一步完善。

参 考 文 献

1 The Stanford Stream Data Manager. http://www-db.stanford.edu/stream
2 Carney D, Cetintemel U, Cherniack M, et al. Monitoring streams—a new class of data management applications. In: Proc. Int. Conf. on Very Large Databases (VLDB), 2002
3 Chandrasekaran S, Deshpande A, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR, Jan. 2003
4 Chen J, DeWitt D J, Tian F, et al. NiagaraCQ: A scalable continuous query system for internet databases. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000. 379～390
5 Floyd S, Paxson V. Wide-area traffic: The failure of poisson modeling. IEEE/ACM Transactions on Networking, 1995, 3(3): 226～244
6 Kleinberg J. Bursty and hierarchical structure in streams. In: Proc. 2002 ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, Aug. 2002
7 Leland W, Taqqu M, Willinger W, Wilson D. On the self-similar nature of ethernet traffic. IEEE/ACM Transactions on Networking, 1994, 2(1): 1～15
8 Tatbul N, Cetintemel U, et al. Load Shedding in a DataStream Manager. VLDB 2003
9 Ayad A M, Naughton J F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. SIGMOD 2004
10 Babcock B, Datar M, Motwani R. Load Shedding for Aggregation Queries over Data Streams. ICDE 2004
11 Hoeffding W. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 1963, 58: 13～30
12 Motwani R, Widom J, Arasu A, et al. Query processing, approximation, and resource management in a data stream management system. In: Proc. First Biennial Conf. on Innovative Data Systems Research (CIDR), Jan. 2003
13 Yang C, Reddy A V S. A Taxonomy for Congestion Control Algorithms in Packet Switching Networks. IEEE network, 1995, 9(5): 34～44
14 Compton C L, Tennenhouse D L. Collaborative Load Shedding for Media-Based Applications. In: Intl. Conf. on Multimedia Computing and Systems, Boston, MA, May 1994. 496～501
15 Zhu Y, Shasha D. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In: Proc. Int. Conf. on Very Large Data Bases, 2002. 358～369
16 Das A, Gehrke J, Riedwald M. Approximate join processing over data streams. In: Proc. 2003 ACM SIGMOD Conf., 2003. 40～51
17 Kang J, Naughton J F, Viglas S. Evaluating window joins over unbounded streams. In: Proc. 2003 Intl. Conf. on Data Engineering, Mar. 2003
18 Liu L, Pu C, Tang W. Continual Queries for Internet-Scale Event-Driven Information Delivery. IEEE Trans. Knowledge and Data Eng., 1999, 11(4): 610～628
19 Abadi D, Carney D, Cetintemel U, et al. Aurora: A Data stream Management System. In: ACM SIGMOD conference, San Diego, CA, June 2003. 666
20 Abadi D, Carney D, Cetintemel U, et al. Aurora: A New Model and Architecture for Data Stream Management. To be appeared in VLDB Journal
21 Tatbul N, Cetintemel U, Zdonik S, et al. Load Shedding in a Data Stream Manager: [Technical Report CS-03-03]. Brown University, Computer Science, Feb. 2003
22 Babcock B, Datar M, Motwani R. Load shedding techniques for data stream systems (full version). Draft in preparation, 2003

(上接第79页)

了新的要求。本文结合一个具体的嵌入式实时内存数据库系统，提出了一个适合于嵌入式实时内存数据库的基于日志的故障恢复模式并给出了相应数据结构和实现算法。不同于传统的顺序日志模式，在我们的恢复模式中，为并发的每一事务分配独立的活动日志区，不同事务的日志记录被临时记录在各自的不同活动日志区，从而解决了顺序日志模式中当前日志缓冲页因为严重的访问竞争而影响系统性能的问题；在检验点模式上，我们采用了执行频率动态可调的动态检验点模式，提高了并发度，很好地解决了如何确定检验点触发时机和执行频率的问题。系统故障发生后，在恢复处理过程中，就Redo起始点的选择做了仔细分析，确保提交事务持久性的满足。在恢复SDB的同时，重建了内存数据库(MMDB)，从而降低了故障恢复时间。在恢复的具体实现算法中，充分考虑了数据和事务的时限要求，尽可能地减少内外存I/O操作的次数，从而有利于实时事务截止期的满足。

本文提出的故障恢复模式也存在一定局限性，如基于活动日志区的日志执行模型的正确性是以严格两段锁协议(并发控制协议)作为前提。进一步的改进在于：采用更具一般性的分区日志技术(放宽对并发控制协议的要求)；采用非易失高速存储设备作为日志存储区；采用多日志文件代替单一日志文件等。这些技术的使用可进一步改进恢复的效率，但同时也可能增加实现的复杂度。在后续的工作中，我们将在恢复效率和实现复杂度方面进行权衡，进一步改进我们的恢复策略。

参 考 文 献

1 胡国玲，刘云生，彭嘉雄. 内存数据库系统的恢复技术. 华中理工大学学报，1996，24(3)：35～38
2 Eliezer L, Avi S. Incremental Recovery in Main Memory Database Systems. IEEE Transactions on Knowledge and data Engineering, 1992, 4(6): 529～540
3 Lam K Y, Kuo T W. Real-Time Database Architecture and Techniques. Boston: Kluwer academic publishers, 2001
4 Huang J. Recovery Techniques in Real-Time Main Memory Databases: [Ph. D. Dissertation]. School of Computer Science, The University of Oklahoma
5 刘云生. 现代数据库技术. 北京：国防工业出版社，2001