数据压缩和数据库性能

Goetz Graefe
科罗拉多大学
博尔德，CO 80309-0430

Leonard D. Shapiro
波特兰州立大学波特兰，
OR 97207-0751

以前在 Proc. ACM/IEEE-CS Symp 中出现，性能结果较少。应用计算，

密苏里州堪萨斯城，1991 年 4 月。

摘要

数据压缩被广泛应用于数据管理中，以节省存储空间和网络带宽。在本报告中，我们概述了在查询处理中利用数据压缩可以实现的性能改进。新颖的想法是尽可能长时间地让数据处于压缩状态，并且只在绝对必要时才解压缩数据。我们将展示许多查询处理算法可以像处理解压缩数据一样处理压缩数据，并且处理压缩数据可以使查询处理速度提高一个比压缩因子大得多的因子。

1.介绍

数据压缩是节省存储空间和网络带宽的有效手段。基于字符编码或重复字符串检测已经设计了大量的压缩方案[2,18]，许多压缩方案对英语文本[2]实现了每个字符 2.3– 2.5 比特的数据压缩率，即压缩因子约为 $3^{1}/_{4}$。由于压缩方案对网络带宽是如此成功，数据压缩对数据库系统中 I/O 性能的有利影响是相当明显的，即它对磁盘空间、带宽和吞吐量的影响。然而，我们相信，在数据库系统中压缩的好处可以被观察到，并在 I/O 性能之外加以利用。

数据库性能很大程度上依赖于可用内存的数量，无论是作为 I/O 缓冲区还是查询处理算法的工作空间。因此，尝试尽可能有效地使用所有可用内存似乎是合乎逻辑的——换句话说，以压缩的形式在内存中保存和操作数据。当然，这要求查询处理算法能够对压缩的数据进行操作。在本报告中，我们介绍了允许这样做的技术，并演示了它们对数据库性能的影响。

在下一节中，我们将简要说明相关工作。在第 3 节中，我们确定了允许数据库处理算法操作压缩数据的技术。第 4 节包含了关系数据库系统中常用的简单连接查询的初步性能分析。我们在第 5 节提供了我们的结论。

2.相关工作

许多研究人员已经考虑了基于字母频率的文本压缩方案，由 Huffman 首创[10,11,16,17]。其他最近的研究考虑了基于字符串匹配的方案[22,30,32,33]，对压缩方法和方案的全面调查在[2,18,27]中给出。其他人则专注于算法的快速实现、并行算法和 VLSI 实现[12,29]。

很少有研究人员调查了压缩对数据库系统及其性能的影响[20,23,25]。Alsberg 考虑通过使用定长最小位压缩[1]来节省磁盘时间和空间。他还指出，如果记录更短，则可以容纳更多的记录到一个页面中，因此可以更有效地将一起使用的记录聚类。Toyama 和 Ura 通过关联压缩值和未压缩值[28]的顺序，改进了 Alsberg 的压缩方案。Eggers 和 Shoshani 探索了高效地搜索和检索压缩数据[8]。Goyal 考虑了允许在压缩数据[13]上搜索(选择)的文本压缩方法。Cormack 讨论了 IMS 数据库管理系统[6]中整个记录(不是字段)的压缩。

其他研究人员研究了具有非常多常数(例如，零)的科学数据库的压缩和访问方案[9,19,21]，并考虑了对压缩数据的非常特殊的操作(矩阵转置)[31]。

对于索引，许多数据库系统使用前缀和后缀截断来节省空间并增加节点的扇出，例如 Starburst[15]。然而，在大多数数据库管理系统中，通常根本不使用实际的压缩方法。特别是，数据压缩对更常用的数据库操作(如关系连接、重复消除和集合交集)的性能影响，到目前为止还没有被考虑。

3.使用压缩数据进行数据库处理

在本节中，我们将概述如何在数据库系统中利用压缩，以及如何适应标准查询处理算法来处理压缩数据。我们将讨论分为三个部分，分别是 I/O 性能和缓冲、事务处理和

查询处理。

当然，可以为任何数据集实现的压缩率取决于属性类型和值分布。例如，压缩二进制浮点数是很困难的，但是压缩英文文本的倍数是 2 或 3[2]则相对容易。下面，我们不要求所有数据都是英文文本;我们只要求可以实现一些压缩。由于文本属性往往是数据库文件中最大的字段，我们怀疑，对于许多甚至大多数数据库文件来说，期望整体压缩因子为 2 是现实的。只有通过明智地决定压缩哪些属性和采用哪种压缩方法才能获得最佳性能。

最明显的是，压缩可以减少给定数据集所需的磁盘空间量。这对 I/O 性能有很多影响。首先，减少的数据空间适合更小的物理磁盘区域;因此，寻道距离和寻道时间减少。其次，更多的数据适合于每个磁盘页面、轨迹和圆柱体，允许更智能

将相关物体聚类到物理上接近的位置。第三，未使用的磁盘空间可用于磁盘阴影，以提高可靠性、可用性和 I/O 性能[3]。第四，压缩后的数据可以更快地存取磁盘。换句话说，数据压缩是增加磁盘带宽(不是通过增加物理传输率，而是通过增加传输数据的信息密度)和缓解许多高性能数据库管理系统[4]中存在的 I/O 瓶颈的有效手段。第五，在分布式数据库系统和客户机-服务器情况下，压缩数据可以比未压缩的数据更快地通过网络传输。未压缩的数据要么需要更多的网络时间，要么需要单独的压缩步骤。最后，在 I/O 缓冲区中保留压缩形式的数据，可以让更多的记录保留在缓冲区中，从而提高缓冲区命中率，减少 I/O 数量。

最后三点其实比较一般。它们适用于磁带、磁盘、控制器缓存、本地和远程主存以及 CPU 缓存的整个存储层次结构。如果所有层次上的存储空间都被更有效地利用，那么在层次结构中向上或向下移动数据时，横向移动数据时(例如，在内存或缓存之间)，以及通过在每个层次上实现更高的命中率，都可以节省带宽。在共享内存系统中减少总线流量也可能允许更高的并行度，而不需要总线饱和[14]。

对于事务处理来说，压缩可能会有两个主要的影响。首先，缓冲区的命中率应该会增加，因为更多的记录可以放入缓冲区空间。第二，对日志设备的 I/O 应该减少，因为日志记录可以变得更短。大多数系统已经采用了“日志压缩”，当将日志设备归档到廉价的存储介质(磁带)上时，只保存已提交事务的日志记录。除了这些方法之外，我们还提出了一种廉价的技术来减少进程早期的日志流量，即在日志记录首次写入磁盘之前。

如果数据库包含压缩值，那么在日志中包含压缩值是微不足道的。换句话说，压缩数据库值也减少了日志记录的大小，从而减少了到日志设备的 I/O 流量。因此，压缩数据库值可以提高主数据库和日志上的 I/O 性能。可以想象，仅日志中的节省就证明了压缩数据库值输入数据库的开销是合理的。

对于查询处理，我们要求对每个属性使用固定的压缩方案。每个属性的方案都应该是固定的，以允许数据库值与(压缩的)谓词常量进行比较。实际上，对于每个域，而不是每个属性，它应该是固定的，因为这将允许比较来自不同来源的压缩值。这并不排除动态压缩方案，即使乍一看是这样的。而不是频繁地调整编码，例如，在传输流的动态压缩和解压中可以做的每个字符之后，压缩编码只能在数据库重组期间进行调整。在卸载数据库时可以收集合适的统计数据，并且可以在重新加载数据库时开始使用新的编码。事实上，在重新加载期间卸载数据和压缩数据期间分离动态压缩方案的统计信息收集，消除了动态压缩期间的启动和调整期

这些计划并不是很有效。在重新加载期间使用的压缩方案的参数是元数据或目录的一部分。

已经使用的一种压缩形式是对大值的小集进行编码。基本上这个想法是在[1]中从压缩(而不是数据库设计和规范化)的角度引入的。例如，在一个大的“parts”表中，不是在每条记录中存储一个长字符串值的“color”属性，而是习惯使用一个小整数来编码颜色，将编码保存在一个单独的关系中，并将大表与相对较小的编码表连接起来，以用于需要字符串值输出 color 属性的查询。由于这样的编码表通常很小，例如在几 kb 的顺序，因此可以使用高效的基于哈希的算法来进行连接，其性能优于嵌套循环连接或基于排序的合并-连接[5,7,26]这两种朴素方法。从长远来看，我们希望这样的编码可以像今天的数据库管理系统中的索引一样被自动管理，并且可以由自动化的物理数据库设计和调优软件在适当的地方推荐它们。

在查询处理中，压缩的应用可以远远超出 I/O 性能的提升。首先，可以在压缩数据上执行精确匹配比较。考虑一个检索社会安全号码为“123-45-6789”的记录的查询。如果选择谓词中的常量以与数据文件中的社会保险号码相同的方式压缩，则可以在不解压缩数据文件中的属性值的情况下执行比较。注意，如果使用一致的压缩方案，则可以对压缩后的数据进行精确匹配索引查找。

其次，投影和重复消除可以在不解压缩数据的情况下完成。由于我们假设属性和整个记录的编码是固定的，相等的未压缩记录将具有相等的压缩图像。虽然用于重复消除和用于聚合和分组的算法基本上是相同的，但用于聚合的情况稍微复杂一些 [1]。虽然分组属性可以保持压缩，但执行算术的属性通常必须进行解压缩。

第三，对于大多数连接来说，连接属性和其他属性都不需要解压缩。几乎所有的连接都是键和外键上的相等连接，也就是说，它们把规范化和分解的对象重新放在一起。由于键和外键来自同一个域，并且由于我们要求每个域的压缩方案都是固定的，因此在压缩键值上的连接将与在正常的、解压缩的键值上的连接得到相同的结果。按压缩值的顺序执行合并连接似乎不太寻常，但这仍然是可能的，并且会产生正确的结果。与 join 相同的论点适用于半连接、外部连接、并集、交集和差异。

[1]聚合的标准例子是“按部门计算的工资总和”。部门在这里称为分组属性。

到目前为止，我们已经证明了从磁盘读取和写入压缩数据的速度更快，并且对于最频繁的数据库操作，没有必要对数据进行解压缩。让我们考虑一下处理压缩数据的一些额外优点，并将重点放在连接上。

首先，物化连接输出记录更快，因为记录更短，需要的复制更少。在许多数据库系统中，复制是 CPU 和总线活动的重要组成部分，在共享内存单总线系统中减少复制尤为重要。

其次，对于比内存大的连接输入，更多的记录可以放入内存。例如，在混合哈希连接中，可以保留在哈希表中，从而在没有任何 I/O 的情况下进行连接的文件的比例更大。在合并连接的排序过程中，内存中的记录数量以及每次运行的记录数量会更大，从而导致更少的运行和可能更少的合并级别。

第三，也是非常有趣的一点，倾斜不太可能成为问题。压缩的目标是用尽可能少的比特来表示信息。因此，一个好的压缩方案的输出中的每一位都有接近最大的信息内容，在整个文件上看到的位列不太可能发生倾斜。此外，位列也不会相互关联。因此，压缩的键值可以用来创建一个散列值分布，几乎可以保证是均匀的，即最适合在内存中散列和分区到溢出文件以及并行连接算法中的多个处理器。

为了总结这一节，在数据库管理系统中有许多迄今为止被忽略的数据压缩用途。压缩不仅可以提高存储层次结构中每个“缓冲”级别的 I/O 性能和命中率，还可以在数据库查询处理中加以利用，而无需对已实现的算法进行太多更改。我们已经概述了数据压缩对性能的一些影响;在下一节中，我们将在一个示例查询中量化其中的一些影响。

4.性能

为了进行性能比较，请考虑混合哈希连接[7,26]使用两个关系 R 和 S 的 400 页内存(R 需要未压缩 1000 页，S 需要未压缩 5000 页)的 I/O 成本。为了简单起见，我们忽略了碎片化，并假设哈希值分布是一致的。

首先，考虑使用未压缩数据的成本。读取存储数据的成本是 6000 个 I/ o。在 R 上构建哈希表时，398 页可以留在内存中，602 页必须写入两个构建溢出文件。在使用输入 S 进行探测时，相当于 1990 页的记录可以立即与常驻哈希表合并，必须将 3010 页写入两个溢出文件。接下来，这两对溢出文件必须被连接起来，需要 602+ 3010 个 I/ o 来读取它们。整个成本是永久文件 6000 个 I/ o，临时文件 7224 个 I/ o，总共 13224 个 I/ o。

现在考虑加入压缩的输入关系。读取初始压缩数据需要 500+ 2500 个 I/ o。在构建阶段，399 页保留在内存中，101 页被写入溢出文件。在探查阶段，相当于 1995 页的记录立即被连接，

505 页被写入磁盘。连接两个溢出文件需要 101+505 个 I/ o。整个成本为永久文件 3000 个 I/ o，临时文件 1212 个 I/ o，总共 4212 个 I/ o。

总的 I/O 成本相差超过 3 倍。永久文件的 I/O 成本相差 2 倍，正如预期的那样，压缩比为 50%，而临时文件的 I/O 成本相差近 6 倍。2 的倍数很容易被预料到;然而，内存利用率的提高(在构建阶段有更多的记录保留在哈希表中)显著减少了必须写入溢出文件的记录数量。因此，压缩减少了写入临时文件的记录的数量和大小，从而使临时文件上的 I/O 成本降低了六倍。

如果压缩方案再有效一点，即将 2 / 2 的倍数从 2^1 降低到 40%，或者将 50%降低到 40%，那么对于压缩的数据，就可以完全避免溢出文件，只留下永久数据上的 I/O。总 I/O 成本将相差 $5^1/_2$ 倍，2,400 到 13,224 个 I/O。图 4.1 显示了压缩因子对关系 R 和 s 的混合哈希连接性能的影响，曲线上方的数字表示了底部轴上标记的压缩因子的确切 I/O 成本。

图中可以分为两个区域。对于压缩因子低于 $2^1/_2$，构建输入大于内存，发生哈希表溢出，通过压缩减少的 I/O 大于压缩因子，类似于上面的例子。对于 $2^1/_2$ 以上的压缩因子，不会发生溢出，压缩只会减少永久文件上的 I/O。在这张图中观察到非常令人鼓舞的是，已经非常温和的压缩因子(例如 $1^1/_2$)显著降低了总 I/O 成本。即使为了解压输出数据而产生了一些额外的成本，通过压缩磁盘和内存上的永久和临时数据所获得的性能收益也远远超过了解压的成本。

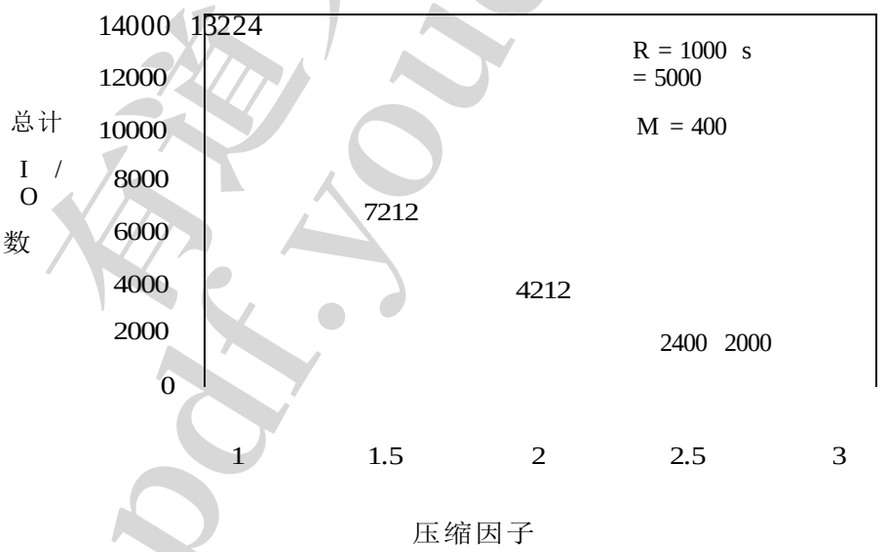

图 1所示。压缩对混合哈希连接性能的影响。

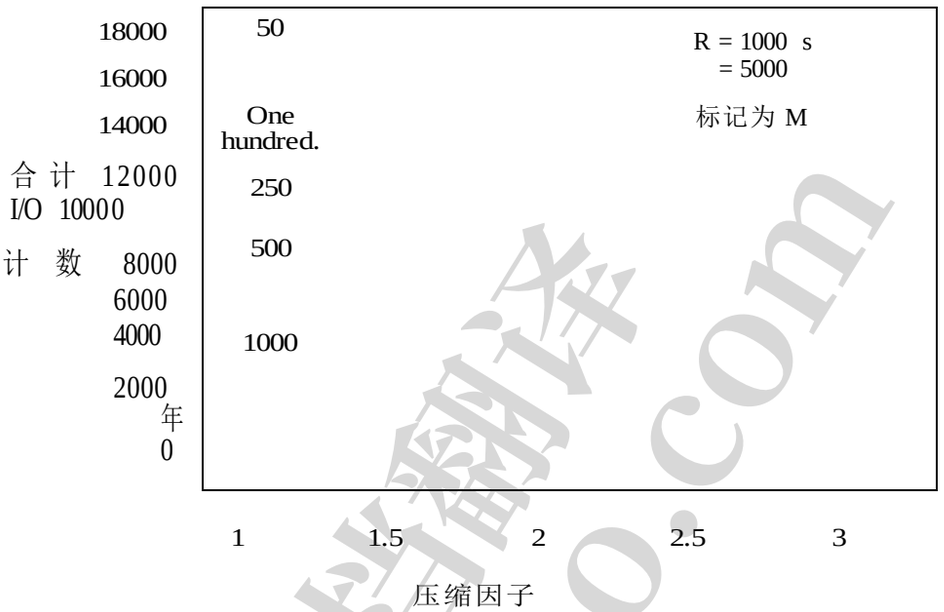

图 2。压缩和内存大小对混合哈希连接性能的影响。

图 4.2 显示了不同内存大小下压缩对 R 和 S 混合连接性能的影响。对于 1000 页的内存大小，最底部的曲线反映了没有溢出的情况。500 页内存的曲线有一个陡峭的梯度，直到压缩因子 2。超过这一点，哈希表就适合内存，500 页和 1000 页的曲线重合。对于 250 页内存，即 $^{1}/_{4}$ of R，曲线在压缩因子为 4 时连接其他曲线而不溢出。对于所有较小的内存大小，在考虑压缩因子的频谱时，哈希表不适合内存。然而，对于所有内存大小，性能增益都大于压缩因子。对于 50 页或 100 页的内存，曲线彼此非常接近，因为几乎所有的 R 和 S 都必须写入溢出文件。如果一个系统的内存非常有限，那么把它分配给那些可能能够在没有溢出的情况下运行的操作符，并以最大的效率使用那里的内存，也就是尽可能使用最好的压缩方案，可能会更重要。

图 4.3 显示了前一个图的加速。最底部的曲线，对于 1000 页内存，表示线性加速。所有其他曲线表示超线性加速。500 页的曲线在压缩因子 2 处有一个明显的“膝盖”，这在前面的图中已经可以看到。对于 250 页的内存，膝盖将位于压缩因子 4，在图的边缘，这里的曲线表明加速因子略大于 10。考虑到在压缩因子仅为 4 的情况下就可以实现 10 的加速，因此进一步研究压缩对数据库的影响势在必行

查询处理算法及其性能。

图 4.4 显示了更大输入关系的性能。和前面的图一样，即使是少量的压缩也能显著提高性能。然而，与构建输入相比，对于较小的内存大小，在内存中拟合更多压缩记录比未压缩记录的效果不如中等大小的关系那么强。尽管如此，在 I/O 性能上的增益至少与压缩因子相等，并且有最急剧的下降

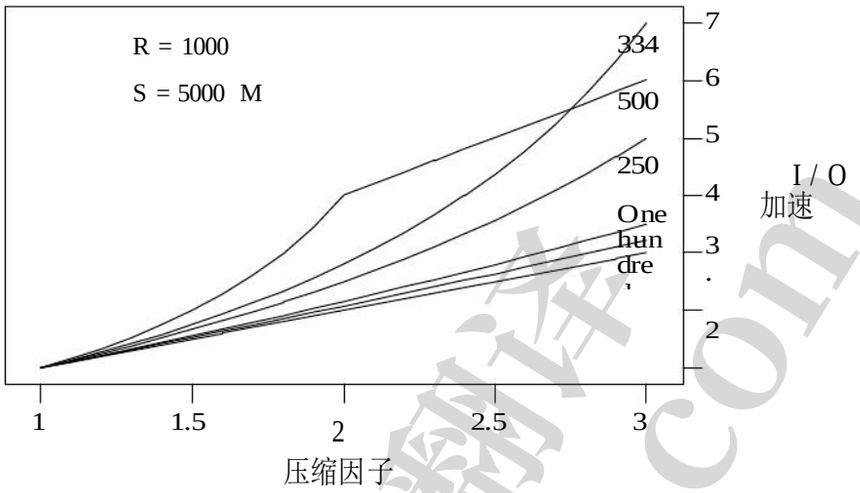

图 3。通过压缩的 Hybrid Hash Join 的加速。

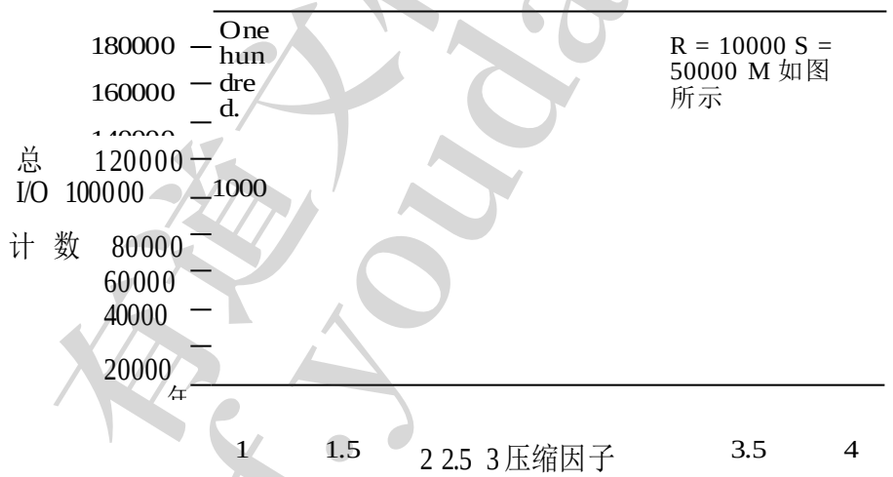

图 4。压缩对大型混合哈希连接性能的影响。

对于小压缩因子。对于较大的内存大小，图 4.4 中的 5000 页，其中构建输入大小是内存的一个小倍数，我们看到了与之前相同的效果——对于小的压缩因子有一个陡峭的梯度，其中压缩因子等于构建输入大小和内存大小的商。

对于单个 hybrid-hash 连接观察到的相同效果，也可以在复杂的查询和排序-合并连接中观察到。我们省略了这个分析，而是参考了[24,26]中排序-合并连接和混合-哈希连接的比较。

5.总结与结论

在本文中，我们概述了一组相当简单的技术，通过数据压缩来实现数据库性能的提升。关键思想是将属性单独压缩，对一个域的所有属性采用相同的压缩方案，并在解压缩数据之前执行数据操作。这些技术不仅减少了对磁盘和 I/O 性能的空间要求，当对永久数据和临时数据进行每次记录的测量时，它们还减少了对内存的要求，从而减少了导致 I/O 的缓冲区故障的数量。当数据压缩与使用大工作空间的算法一起使用时，即使是适度的压缩也能带来显著的性能提升。此外，我们处理压缩数据的技术非常容易实现。

在一个简单的性能比较中，我们已经看到，对于大于内存的数据集，可以获得比压缩因子更大的性能增益，因为分配给查询处理操作符的工作空间中可以保留更大一部分数据。对于压缩因子为 2 的数据，我们观察到因子为 3 及以上的性能增益。

确认

本研究得到了俄勒冈高级计算研究所(OACIS)和美国国家科学基金会 IRI-8805200、IRI-8912618 和 IRI-9006348 奖项的部分支持。

参考文献

[1] P. A. Alsberg，通过大数据库压缩和动态重构节省空间和时间，*Proc. IEEE*63,8(1975 年 8 月)，1114。

[2] T. Bell, i.h. Witten 和 J. G. Cleary，文本压缩建模，*ACM 计算调查* 21,4(1989 年 12 月)，557。

[3] D. Bitton 和 J. Gray，磁盘阴影，*Proc. Int 'l。Conf. on Very Large Data Bases*, Los Angeles, CA, 1988 年 8 月，331。

[4] H. Boral 和 D. J. DeWitt，《数据库机器:一个时代已经过去的想法?》A Critique of the Future of Database Machines, *Proc. Int 'l。数据库机研讨会*，慕尼黑，1983 年，166。转载于 A. R. Hurson、L. L. Miller 和 S. H. Pakzad，《数据库系统的并行架构》，IEEE 计算机协会出版社，华盛顿特区，1989 年。

[5] K. Bratbergsengen，哈希方法和关系代数操作，*Proc. Int 'l。Conf. on Very Large Data Bases*，新加坡，1984 年 8 月，323。

[6] G. V. Cormack，《数据库系统中的数据压缩》，*ACM 通讯 28,12*(1985 年 12 月)，1336。

[7] D. J. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker 和 D. Wood，主存数据库系统的实现技术，*Proc. ACM SIGMOD Conf.*，波士顿，马萨诸塞州，1984 年 6 月，1.。

[8] S. J. Eggers 和 A. Shoshani，压缩数据的高效访问，*Proc. Int 'l。Conf. on Very Large Data Bases*，加拿大蒙特利尔，1980 年 10 月，205。

[9] S. J. Eggers, F. Olken 和 A. Shoshani，大型统计数据库的压缩技术，*Proc. Int 'l。Conf. on Very Large Data Bases*，法国戛纳，1981 年 9 月，第 424 页。

[10] L. Felician 和 A. Gentili，微机环境中的一种近乎最优的霍夫曼技术，Inf. *Sys. 12,4*(1987)， 371。

[11] R. G. Gallager，《主题变奏曲》(Variations on a Theme)， Huffman 著，*IEEE Trans。论 Inf. Theory it-24,6*(1978)， 668。

[12] M. E. Gonzalez-Smith 和 J. A. Storer，数据压缩的并行算法，*J. of the ACM 32,2*(1985 年 4 月)，344。

[13] P. Goyal，压缩数据库上文本字符串搜索的编码方法，*Inf. Sys. 8*, 3(1983)， 231。

[14] G. Graefe 和 S. S. Thakkar，在共享内存多处理器上调优并行数据库算法，*软件－实践与经验* 22,7(1992 年 7 月)，495。

[15] L. Haas, W. Chang, G. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. J. Carey 和 E. Shekita，《星爆中段:尘埃散去》，*IEEE Trans。on Knowledge and Data Eng. 2,1*(1990 年 3 月)，143。

[16] D. A. Huffman，构造最小冗余码的一种方法，*Proc. IRE 40*(1952)， 1098。

[17] D. E. Knuth，动态霍夫曼编码，*J. of Algorithms 6*(1985)， 163。

[18] D. A. Lelewer 和 D. S. Hirschberg，数据压缩，*ACM 计算调查 19,3*(1987 年 9 月)，261。

[19] J. Li, D. Rotem 和 H. Wong，一种大型数据库上具有快速搜索的新压缩方法，*Proc. Int 'l。Conf. on Very Large Data Bases*, Brighton, England, August 1987,311。

[20] C. a . Lynch 和 E. B. Brownrigg，数据压缩在大型书目数据库中的应用，*Proc. Int 'l。Conf. on Very Large Data Bases*，法国戛纳，1981 年 9 月，435 页。

[21] F. Olken 和 D. Rotem，《重新安排数据以最大化压缩效率》，*J. of 计算机与系统科学 38,2*(1989)，405。

[22] M. Rodeh, V. R. Pratt 和 S. Even，通过字符串匹配进行数据压缩的线性算法，*J. of the ACM 28*, 1(1981 年 1 月)，16。

[23] S. S. Ruth 和 P. J. Keutzer，商业文件的数据压缩，*Datamation 18*(1972 年 9 月)，62。

[24] D. Schneider 和 D. DeWitt，在无共享多处理器环境中四种并行连接算法的性能评估，*Proc. ACM SIGMOD Conf.*，波特兰，OR，1989 年 5 月- 6 月，110。

[25] D. G. Severance，《数据库压缩从业者指南》，*Inf. Sys. 8*, 1(1983)， 51。

[26] L. D. Shapiro, Join Processing in Database Systems with Large Main memory, *ACM Trans。on Database Sys. 11,3*(1986 年 9 月)，239。

[27] J. A.存储器，*数据压缩:方法和理论*，比较科学。出版社，纽约，NY，1988 年。

[28] M. Toyama 和 S. Ura，用于现场级数据文件压缩的定长半序保码，*Proc. IEEE Int 'l。conference .on Data Eng.;*，洛杉矶，加州，1984 年 4 月，244。

[29] t.a. Welch，《一种高性能数据压缩技术》，*IEEE 计算机 17,6*(1984 年 6 月)，8。

[30] R. N. Williams, Dynamic-historic predictive compression, *Inf. Sys. 13,1*(1988)， 141。

[31] H. K. T. Wong 和 J. C. Li，超大压缩数据上的转置算法，*Proc. Int 'l。Conf. on Very Large Data Bases*，京都，日本，1986 年 8 月，304 页。

[32] J. Ziv 和 A. Lempel，顺序数据压缩的通用算法，*IEEE Trans。论 Inf. Theory IT-23*, 3(1977 年 5 月)，337。

[33] J. Ziv 和 A. Lempel，通过可变速率编码压缩单个序列，*IEEE Trans。论 Inf. Theory IT-24*, 5(1978 年 9 月)，530。