《Architecture of a Database System》

（中文版）

Joseph M. Hellerstein, Michael Stonebraker and James Hamilton

翻译：林子雨

厦门大学数据库实验室

http://dblab.xmu.edu.cn

中文版网址：**http://dblab.xmu.edu.cn/node/459**

厦门大学计算机科学系教师 林子雨 翻译作品

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本文翻译自经典英文论文《Architecture of a Database System》，原文作者是 Joseph M. Hellerstein, Michael Stonebraker 和 James Hamilton。该论文可以作为中国各大高校数据库实验室研究生的入门读物，帮助学生快速了解数据库的内部运行机制。

本文一共包括 6 章，分别是：第 1 章概述，第 2 章进程模型，第 3 章并行体系结构：进程和内存协调，第 4 章关系查询处理器，第 5 章存储管理，第 6 章事务：并发控制和恢复，第 7 章共享组件，第 8 章结束语。

本文翻译由厦门大学数据库实验室林子雨老师团队合力完成，其中，林子雨老师负责统稿校对，刘颖杰同学负责翻译第 1 章、第 2 章和第 6 章，罗道文同学负责翻译第 3 章和第 4 章，谢荣东同学负责翻译第 5 章，蔡珉星同学负责翻译第 7 章和第 8 章。

如果对本文翻译内容有任何疑问，欢迎联系林子雨老师。

林子雨的E-mail是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

厦门大学数据库实验室网站是：http://dblab.xmu.edu.cn。

本文中文版的网址是：http://dblab.xmu.edu.cn/node/459。

林子雨于厦门大学海韵园

2013 年 9 月

摘 要

数据库管理系统（DBMS）广泛存在于现代计算机系统中，并且是其重要的组成部分。它是学术界以及工业界数十年研究和发展的成果。在计算机发展史上，数据库属于最早开发的多用户服务系统之一，因此，它的研究也催生了许多为保证系统可拓展性以及稳定性的系统开发技术，这些技术如今被应用于许多其他的领域。虽然许多数据库的相关算法和概念广泛见于教科书中，但关于如何让一个数据库工作的系统设计问题却鲜有资料介绍。本文从体系架构角度探讨数据库设计的一些准则，包括处理模型、并行架构、存储系统设计、事务处理系统、查询处理及优化结构以及具有代表性的共享组件和应用。当业界有多种设计方式可供选择时，我们以当前成功的商业开源软件作为参考标准。

第 5 章 存储管理

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

本文中文版网址：http://dblab.xmu.edu.cn/node/459

2013 年 9 月

第 5 章 存储管理

在今天的商业应用中，主要有两种基本类型的 DBMS（数据库管理系统）存储管理器：（1）DBMS 直接与底层的面向磁盘的块模式设备驱动程序进行交互（通常称为原始模式访问）;（2）DBMS 使用标准的 OS 文件系统设施。这个决定会在空间和时间上同时影响 DBMS 控制存储的能力。下面我们先分别考虑这两个维度（空间和时间），然后继续深入地讨论存储层次的使用。

5.1 空间控制

从磁盘中读取和写入数据时，顺序读写带宽要比随机读写带宽快 10 到 100 倍，并且这个差距还在增加。磁盘密度每 18 个月翻一番,带宽增长速度与磁盘密度呈平方根的关系(与旋转速度呈线性关系)。但是，磁盘机械臂移动速度的增长率则较低，约为每年 7% [67]。因此，对于 DBMS 存储管理器来说，如何把数据块放置在磁盘上就显得尤其重要，从而使得需要访问大量数据的查询可以顺序地访问磁盘。因为 DBMS 能够比底层操作系统更理解它的工作负载访问模式，所以，完全由 DBMS 设计师来确定如何把数据库块放置到磁盘上，是有意义的。

对于 DBMS 而言，控制数据空间局部性的最好的方式，就是将数据直接存储到“原始”磁盘设备中，完全绕过文件系统。这种做法是可行的，因为，原始设备地址通常对应于存储位置的物理临近性。为了获得峰值性能，大部分商业数据库系统能够很好地提供这项功能。这种技术虽然有效，但还是有一些缺点。第一，它需要数据库管理员将整个磁盘分区都分配给数据库管理系统，这就使得这些磁盘空间无法提供给那些需要使用文件系统接口的工具。第二，“原始磁盘”的访问接口往往是与特定操作系统相关的，这使得 DBMS 的可移植性变得更差。这是一个难点，不过近来大多数商业 DBMS 厂商都已经解决这一问题。最后，随着存储行业的发展，RAID、存储区域网络和逻辑卷（volume）管理器，已经非常普及。现在我们所处的情况是，在许多应用场景中，“虚拟”磁盘设备已经成为规范——“原始”设备接口实际上已经被很多硬件和软件拦截掉，它们会在一个或多个物理磁盘之间重新定位数

据。因此，随着时代的发展，由 DBMS 显示地控制磁盘所带来的收益已经越来越少。我们将在第 7.3 节进一步讨论这个问题。

原始磁盘访问的一种替代方式是，由 DBMS 在操作系统的文件管理系统中创建一个非常大的文件，然后采用数据在文件中的地址偏移量来定位数据。这个文件在本质上可以视为磁盘页面的一个线性阵列。这可以避免原始设备访问的一些缺点，并且仍然能够提供相当好的性能。在大多数流行的文件系统中，如果你分配一个非常大的文件到一个空磁盘上，文件中的地址将会和存储位置的物理临近性非常吻合。因此，这是一个原始磁盘访问的很好的近似方法，而不需要直接访问原始设备接口。大多数虚拟化存储系统在设计时，都会把文件中临近的地址放置在临近的物理位置中。因此，随着时间的推移，使用大文件而不是原始磁盘的相对控制代价，已经变得越来越不明显了。使用文件系统接口对时间控制有其他后果，我们将在接下来的内容中讨论。

我们最近在一个中等规模的系统中，使用一个主流的商业 DBMS，对直接原始磁盘访问和大型文件访问这两者进行了比较，我们发现，当运行 TPC-C 测试基准[91]时，只有 6%的性能降低，而对于较少包含密集 I /O 的工作负载而言，几乎没有负面影响。DB2 的报告显示，当使用直接 I / O（DIO）和它的变体比如并发 I / O(CIO)时，文件系统开销可以低至 1%。因此，数据库管理系统厂商通常不再推荐原始数据存储，而且很少用户会使用这种配置。一些主流商业系统还在支持这种特性，主要是用于测试基准（benchmark）。

一些商业 DBMS 还允许自定义数据库页面大小，使它能够适合预期的工作负载。IBM DB2 和 Oracle 等都支持这个功能。其他的商业系统，比如 Microsoft SQL Server，不支持多种页面大小，因为这会增加管理的复杂性。如果支持页面大小可调，那么可供选择的页面尺寸应该是一个文件系统（如果使用原始 I / O，这里就是原始设备）所使用的页面尺寸的倍数。在“5 分钟法则”的论文中，讨论了如何选择合适的页面大小，这个法则后来又被更新为“30 分钟法则”[27]。如果使用文件系统而不是原始设备访问，就需要特定的接口来写入那些与文件系统页面大小不同的页面，例如 POSIX 的 mmap / msync 调用，就提供了这种支持。

5.2 时间控制:缓冲

除了控制数据在磁盘上的放置位置，一个 DBMS 还必须控制数据什么时候被物理地写入到磁盘中。正如我们将在后面内容讨论的那样，DBMS 包含了关键的逻辑程序，它可以

判断什么时候把数据写入磁盘。大多数操作系统的文件系统还提供内置的 I / O 缓冲机制，来决定何时读取和何时写入文件块。如果 DBMS 在执行写操作时使用标准的文件系统接口，操作系统缓冲机制将会打乱 DBMS 逻辑程序的意图，因为，操作系统缓冲机制会悄悄地推迟 DBMS 写操作或者重新排序写操作。这可能会给 DBMS 带来大问题。

第一类大问题是，数据库的 ACID 事务承诺的正确性：如果不能对磁盘写操作的时间和顺序进行显式的控制，那么，在发生软件或硬件失败后，DBMS 不能保证原子恢复。正如我们将在第 5.3 节讨论的那样，写前日志（write ahead logging）要求一个写操作在写入到数据库相应的设备之前，必须先写入日志设备，并且在提交日志记录没有写入日志设备之前，不能给用户返回提交请求。

OS 缓冲问题的第二个方面是性能问题，但是没有对正确性提出要求。现代操作系统的文件系统，通常提供了内置功能支持“预读取（read-ahead）”（随机的读取）和“后写入（write-behind）”(延迟，批处理)。这些对于 DBMS 访问模式而言，通常都不适合。文件系统的逻辑程序，会根据文件中的物理字节偏移量连续性来做出提前读取的决定。DBMS 级别的 I/O 设施，可以根据未来的读请求来做出逻辑预测 I/O 决定，这些未来的读请求在 SQL 查询处理层面是可以知道的，但是在文件系统层面上则很难做到。例如，当扫描不一定被连续存储的 B+-树叶子节点时（行被存储在一个 B +-树的叶子节点中），就可能会发生逻辑的、DBMS 层面的预读取请求。逻辑预读取是很容易在 DBMS 逻辑中实现的，只要让 DBMS 在它产生实际需求之前就发出 I / O 请求。查询执行计划包含了关于数据访问算法的相关信息，并且拥有关于这个查询的未来访问模式的完整信息。类似地，DBMS 可能想对何时刷新日志尾部做出自己的决定，它会综合考虑锁冲突和 I / O 吞吐量。DBMS 可以获得这些详细的未来访问模式信息，而操作系统的文件系统则无法获得这些信息。

最后的性能问题是“双缓冲”和内存拷贝的昂贵的 CPU 开销。鉴于 DBMS 必须妥善管理自己的缓冲从而保证正确性，因此，任何由操作系统提供的额外缓冲都是多余的。这种冗余会引起两个代价。第一，它浪费了系统内存，因为它显著地减少了系统中可用于有用工作的内存。第二，它浪费了时间和处理资源，因为它会导致额外的复制步骤：执行读取操作时，数据会首先从磁盘复制到操作系统的缓冲区，然后再复制到 DBMS 缓冲池。而写数据时，这两者都刚好相反，即先把数据写入到 DBMS 缓冲池，然后再复制到操作系统的缓冲区，最后写入磁盘。

在内存中复制数据将会是一个严重的性能瓶颈。复制导致延迟，消耗 CPU 周期，并且使 CPU 的数据闪存溢出。这对于一个从没有操作或者实现过数据库系统的人来说，常常是

一个令人惊讶的事，他们往往以为，相对于磁盘 I / O 而言，内存操作是“免费”的。但是在实践中，在一个经过调优后的事务处理 DBMS 中，吞吐量通常不受 I /O 的限制。这在高端 DBMS 配置中，可以通过购买足够的磁盘和内存来实现，从而使得这些重复的页面请求都被缓冲池吸收，磁盘 I/O 会在不同磁盘机械臂之间进行共享，从而使得磁盘 I/O 的速率可以满足数据系统中所有处理器对数据的需求。一旦实现了这种“系统平衡”，I / O 延迟将不再是系统吞吐量的主要瓶颈，剩下的内存瓶颈则会成为系统吞吐量的主要限制因素。内存拷贝正成为计算机体系结构的主要瓶颈：这是由于在性能演变过程中，每秒每美元的原始 CPU 周期（会遵循摩尔定律）和 RAM 的存取速度（明显是在追随着摩尔定律）之间的“鸿沟”越来越大[67]。

操作系统缓冲的问题已经在数据库研究文献中被广泛熟知[86]，被业界所熟知也有一段时间了。大多数现代操作系统现在提供钩子（比如，POSIX mmap 套件调用或者特定于平台的 DIO 和 CIO API 集合），从而使得程序（比如数据库服务器）可以规避对文件的双缓冲。这可以确保当写操作被请求时直接被写入到磁盘，避免了双缓冲，而且 DBMS 可以控制页面置换策略。

5.3缓冲管理

为了提供对数据库页面的有效访问，每个数据库管理系统会在自己的内存空间中实现一个大型共享缓冲池。在早期，缓冲池是被静态设置为管理员设定的值，但是，现在大多数商业 DBMS 会根据系统需要和可用资源来动态调整缓冲池大小。缓冲池会被组织成一个帧数组，其中，每一帧是内存中的一段区域，帧的大小是数据库磁盘块的大小。块从磁盘直接复制到缓冲池中，不会发生格式的变化，在内存中也是以这种原生的格式进行修改操作，然后，写回磁盘。这种不需要转换的方法，避免了在向磁盘写入数据和从磁盘读取数据过程中的 CPU 瓶颈。也许更为重要的是，固定大小的帧，避免了通用技术导致的外部碎片和压缩方面的内存管理复杂性。

和缓冲池中的帧数组相关联的是一个哈希表，它会对以下内容进行哈希映射：（1）把内存中当前的页面编号映射到它们在帧表中的位置；（2）页面在备份磁盘存储中的位置；（3）关于该页面的一些元数据。

元数据包括一个“脏”标记位，用来表示页面在从磁盘中读取出来以后是否已经发生改变；元数据还包括页面替换策略所需要的任何信息，当缓冲区满的时候，页面替换策略会选

择被驱逐出缓冲区的页面。大多数系统还包括一个引脚计数器（pin count），来标记该页面是没有资格参与页面替换算法的。当引脚数是非零时，页面被“钉”（pin）在内存中，不会被强行读取到磁盘或丢失。这允许 DBMS 的工作线程在操作一个页面之前，通过增加引脚数来把页面“钉”在缓冲池中，操作结束后，再减少引脚计数器的值。这样做的目的是，在任何时间点，只让少量的缓冲池空间被“钉”住。有些系统还提供了管理选项，允许把一个表“钉”在内存中。对于较小的、频繁访问的表而言，它可以改进访问时间。但是，被“钉住”的页面，也减少了可以供正常缓冲池活动的页面数量，并且，当被“钉住”的页面百分比增加时，会对性能造成负面影响。

早期的关系系统的许多研究都是集中在设计页面置换政策，因为，DBMS 中数据访问模式的多样性使得简单的技术变得无效。例如，某些数据库操作往往需要全表扫描，当被扫描的表远远大于缓冲池时，这些操作往往会清除缓冲区中所有常用的数据。对于这样的访问模式而言，如果把近期的引用情况作为判断未来引用情况的依据，那将会是非常糟糕的；因此，操作系统中经常使用的页面替换机制（比如 LRU 和 CLOCK），在许多数据库访问模式下面都表现得非常糟糕[86]，这是众所周知的。研究人员提出了各种替代方案，比如，一些方案试图利用查询执行计划信息来调整替换策略[15]。今天，大多数系统使用简单的增强 LRU 方案来进行全表扫描。一个出现在研究文献中并且已经运用到商业系统中的方案是 LRU-2[64]。商业系统中使用的另一种机制是，根据页面类型来确定替换策略，例如，B+-树的根节点的替换策略，可能和堆文件中的页面替换策略不同。这不禁让人想起了 Reiter’s Domain Separation 方案[15,75]。

最近的硬件发展趋势（包括 64 位寻址和内存价格的下降），使得使用非常大的缓冲池变得经济可行。这开启了利用大内存提高效率的新机遇。但是反过来，一个大的、非常活跃的缓冲池，在重启恢复速度和高效的检查点等方面也带来更多的挑战，当然问题还不仅仅是这些。这些主题将在第六章进一步讨论。

5.4标准实践

在过去的十年里，商业文件系统已经进化到可以很好地支持数据库存储系统。在标准的使用模型中，系统管理员在每个磁盘上创建一个文件系统，或者在 DBMS 的逻辑卷上创建一个文件系统。然后，DBMS 为每一个文件系统分配一个单一的大文件，然后通过低层次的接口（如 mmap 套件）来控制数据的放置。DBMS 基本上把每个磁盘或逻辑卷当作一个(几

乎)连续的数据库页面的线性数组。在这个配置中，现代文件系统为 DBMS 提供了合理的空间和时间控制，并且这种存储模型可在所有数据库系统中实现。在大多数数据库系统中，对原始磁盘的支持仍然是一个常见的高性能选项，然而，它的应用范围迅速变窄，现在一般只用于性能基准测试中。

5.5讨论以及附加材料

数据库存储子系统是一个很成熟的技术，但是，在最近几年，数据库存储方面又出现了许多新的考虑因素，它有可能在许多方面改变数据管理技术。

一个关键的技术变化是闪存的出现，它已经是一种经济可行的、支持随机访问和持久存储的技术[28]。自从数据库系统研究的早期阶段，就一直在讨论，新的存储技术取代磁盘会引起 DBMS 设计的巨大变化。闪存具有技术上和经济上的可行性，并且具有广泛的市场支持，它的性价比介于磁盘和 RAM 之间。在近 30 年里，闪存是第一个成功的、新的持久性存储介质，因此，它的特性将可能显著影响未来 DBMS 的设计。

最近走向前台的另一个传统话题是数据库数据的压缩。早期文献的主题主要集中在磁盘压缩，从而最小化读取时的磁盘延迟，并且最大化数据库缓冲池的容量。由于处理器性能已经改进很多，而内存延迟却没有保持同步改进，这使得数据压缩（即使在计算时）变得越来越重要，从而最大化处理器缓存中的数据延迟。但是，这需要压缩数据表示形式是适合用来进行数据处理的，同时还要能够适用于对压缩数据进行操作的查询处理。另一个关系型数据库压缩的难题是，数据库是可排序的元组集合，而大部分关于压缩的研究工作集中在字节流，而不考虑重新排序。最近关于这一主题的研究表明，在不久的将来可以很好地实现对数据库的压缩[73]。

最后，在传统的关系数据库市场之外，大家对大规模但稀疏的数据的存储技术的兴趣，正在日益增强，这些数据在逻辑上有成千上万个列，但对于某个给定的行，其中的大部分列都是空的。这些场景通常采用某种属性-值对或三元组来表示。实例包括谷歌的 BigTable[9]，Microsoft’s Active Directory and Exchange 产品所使用的标签列，以及为语义网提出的资源描述框架（RDF）。这些方法的共同点是，在使用存储系统方面，都采用数据表列的方式而不是行的方式来组织磁盘存储。在最近的一系列数据库研究成果[36,89,90]中，关于面向列的存储思想又复苏起来，并且有了一些详细的研究。

附录 1:译者介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.
主讲课程：《大数据技术基础》
办公地点：厦门大学海韵园科研 2 号楼
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu