《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 章和第 2 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、第 6 章、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 2 章 进程模型

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

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

2013 年 9 月

第 2 章 进程模型

在设计多用户服务器时，必须要尽早决定并发用户请求的执行以及它们是如何被映射到操作系统的进程和线程的。这个决定将对之后的系统软件架构以及运行效率、稳定性、轻便性产生深远影响。在这一章，我们测试一些 DBMS 的运行数据，它们也可以作为其他高并发系统的参考模版。我们考虑一个简化的框架，假设系统支持线程并且仅考虑单片机的情况。然后我们进一步阐述现代 DBMS 在简化的框架下是如何实现它们的进程模型的。在第三章，我们将介绍机群技术以及多进程系统和多核系统。

我们接下来的讨论基于以下定义：

- 一个**操作系统进程**包含了一个正在执行的程序的活动单元以及专属的地址空间。为一个进程维护的状态，包括系统资源句柄和安全性上下文环境。程序执行的单元被操作系统内核管理，每一个进程有其自己的地址空间。
- 一个**操作系统线程**是操作系统程序执行单元，它没有私有的地址空间和上下文。在多线程系统中，每个线程都可以共享地访问所属进程的地址空间。线程的执行是由系统内核来管理的，通常被称为内核线程或者 k-线程。
- **轻量级线程包**是一个应用层次上的结构，它支持单系统进程中的多线程。不同于由操作系统内核调度的线程，轻量级线程由应用级线程调度程序来负责调度。它们之间的区别是，轻量级线程仅在用户空间内调度而没有内核调度程序的参与。轻量级线程仅运行在一个进程中，从操作系统调度程序的角度来看只有一个线程在运行。轻量级线程相对于系统线程而言，可以更快地切换，因为轻量级线程不必通过系统内核来切换调度。轻量级线程也有它的缺点，任何线程中断操作，比如 I/O 中断，都会打断进程中的其他线程。这使得一个线程中断等待系统资源时，其它线程就不能运行。轻量级线程包通过以下两点来避免该情况的发生：（1）只接受无中断的异步 I/O 操作；（2）不调用可以导致中断的系统操作。总的来说，轻量级线程相对于系统进程和系统线程，提供了一个更困难的编程模型。一些 DBMS 系统实现了它们自己的轻量级线程包，这些是轻量级线程包的特殊实现。当需要区别于其它线程时，

我们将这些线程称为 DBMS 线程和简单线程。

- **DMBS 客户端**是应用程序实现与数据库系统通信 API 的软件组件，JDBC、ODBC 和 OLE/DB 是我们熟知的例子。除此之外，还有许多其他种类的 API 集合。有一些程序使用了嵌入式 SQL，这是一种混合了数据库访问表的编程方法。这种方法最初存在于 IBM COBOL 和 PL/I 中，后来 SQL/J 也为 Java 实现了嵌入式 SQL。嵌入式 SQL 会被预处理器处理，从而把嵌入式 SQL 转换为对数据访问 API 的直接调用。不管客户端程序使用什么语法，最终的结果就是一系列针对数据访问 API 的直接调用。这些调用由 DBMS 客户端组件整理然后通过通信协议与 DBMS 交互。这些协议通常拥有专利但是没有正式规范文档。在过去，开源组织 DRDA 曾试图将客户端-数据库通信协议规范化，但并没有被广泛使用。
- **DBMS 工作者**是指 DBMS 中为客户端工作的线程。它与客户端是一一对应的，即一个 DBMS 工作者处理来自一个 DBMS 客户端的所有 SQL 请求。DBMS 客户端发送 SQL 请求给 DBMS 服务器。DBMS 工作者处理每一条请求并将结果返回客户端。在下文中，我们将研究商业数据库把 DBMS 工作者映射到系统线程或者进程的不同方法。当需要区分进程和线程的区别时，我们将分别称之为工作者线程或者工作者进程，如果不需要，那么我们直接用 DBMS 工作者来表示。

2.1 单处理机和轻量级线程

在这一部分，我们介绍一个简单的 DBMS 进程模型的分类方法。尽管先进的 DBMS 很少完全按照我们将介绍的内容来构建，但是在这个基础上，我们可以更详细地讨论新一代产品。现今的每一款数据库系统的核心都是下列所述模型的扩展或者增强。

我们首先做两个简单的假设（在后文的讨论中我们将放宽条件）：

- **系统线程支持**：我们假设操作系统内核有效地支持线程，并且一个进程可以拥有很多线程。我们假设线程存储空间很小，所以，它们在切换的时候并不需要太多的代价。在现代操作系统中这个假设基本是成立的，但是，在 DBMS 设计之初却一定不是这样的。因为，在很多平台上，系统线程或者不支持，或者效率落后，所以，很多 DBMS 系统在开发时没有用到系统线程支持。
- **单处理机**：我们假设我们的设计针对仅拥有一个 CPU 的一台机器。因为多核系统已经广泛存在，所以即便是在低端机型方面，这个假设也是不现实的。但是，该假设

可以简化我们最初的讨论。

通过上下文我们了解到，DBMS 拥有三个天然的进程模型性质。从最简单到最复杂依次为：（1）每个 DBMS 工作者拥有一个进程；（2）每个 DBMS 工作者拥有一个线程；（3）进程池。尽管该模型被简化了，但是，今天的商业 DBMS 系统都遵循这三条性质。

2.1.1 每个 DBMS 工作者拥有一个进程

每个 DBMS 工作者拥有一个进程的模型（如图 2-1 所示），早期被用于 DBMS 开发，并且今天仍被一些商业系统所采用。这个模型由于 DBMS 工作者被直接映射到系统进程，因而相对容易实现。系统调度管理 DBMS 工作者的运行时间，而 DBMS 编程人员也可以利用系统的一些保护措施来排除标准错误，如内存溢出。而且，不同的编程工具，比如调试器和存储检测器，都与这种进程模型相适应。使这个模型变复杂的是 DBMS 连接所需要的内存中的数据结构，包括锁机制和缓冲池（在第 6.3 节和第 5.3 节将详细讨论）。这些共享的数据结构必须分配在所有 DBMS 进程都可以访问的共享内存中。这需要系统支持（一般系统均支持）和 DBMS 相关的代码。实际中，由于这种模型需要广泛使用共享内存，使得地址空间分离的优势被削弱了。

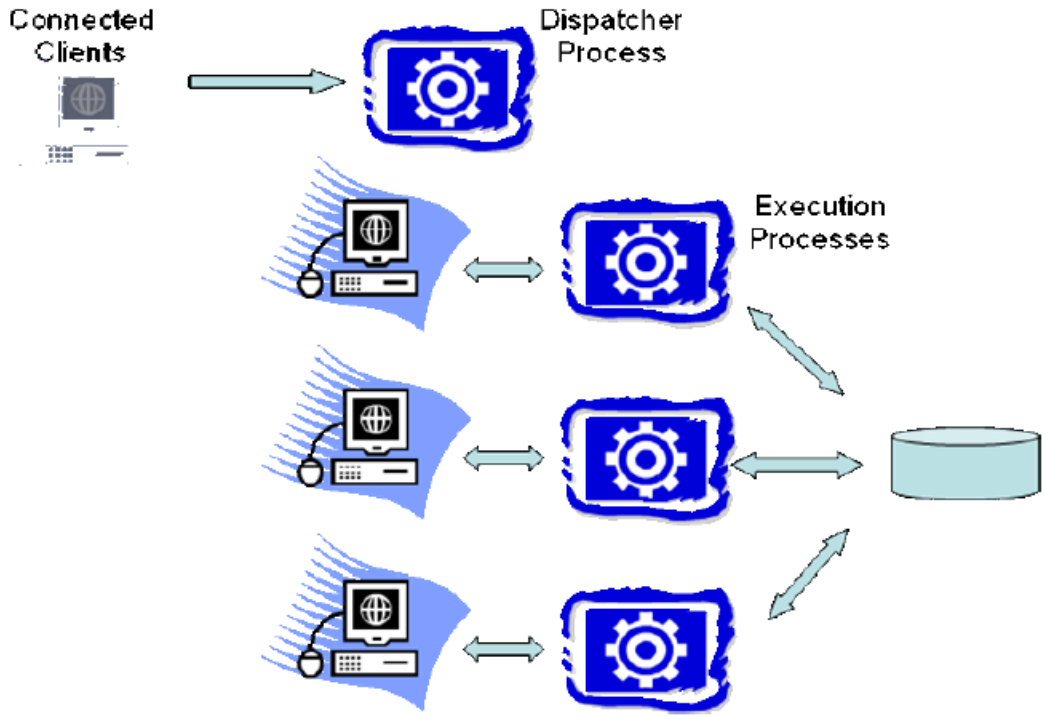

图 2-1 每个 DBMS 工作者拥有一个进程的模型

如果把“扩展到大量并发连接”作为衡量标准，那么，每个 DBMS 工作者拥有一个进程

的模型并不十分有效。这是因为，进程相对于线程而言，拥有更多的环境变量并且消耗更多的存储空间。进程切换需要切换安全的上下文环境、存储空间变量、文件和网络句柄列表以及其他一些进程上下文。这在线程切换时是不需要的。尽管如此，每个 DBMS 工作者拥有一个进程的模型，还是比较受欢迎的，并得到 IBM DB2、PostgreSQL 和 Oracle 的支持。

2.1.2 每个 DBMS 工作者拥有一个线程

在每个 DBMS 工作者拥有一个线程的模型（图 2.2）中，一个多线程进程负责所有的 DBMS 工作者的工作。一个调度线程监听新的 DBMS 客户端连接。每个连接都被分配一个新的线程。当每个客户端提交 SQL 请求时，该请求都由对应的 DBMS 工作者线程来执行。这个线程在 DBMS 进程中运行，一旦运行结束，结果将返回给客户端，然后该线程等待来着这个客户端的下一个请求。

多线程编程在这种架构下有以下困难：操作系统对线程不提供溢出和指针的保护；调试困难，尤其是在运行情况下更是如此；由于不同操作系统在线程接口和多线程扩展性方面的不同，使得软件可移植性较差。“每个 DBMS 工作者拥有一个线程模型”中的很多编程问题，同样存在于每个 DBMS 工作者拥有一个进程模型中，因为它们都需要共享内存的使用。

尽管不同操作系统在线程 API 的多样性方面，近年来不断缩小，但不同平台之间微小的差别还是不断带来调试上的麻烦。如果不考虑实现上的困难，那么，每个 DBMS 工作者拥有一个线程的模型，还是可以很好地扩展到高并发系统中的，并且这种模型在一些现代 DBMS 产品中也被使用了，如 IBM DB2、微软 SQL Server、MySql、Informix 和 Sybase。

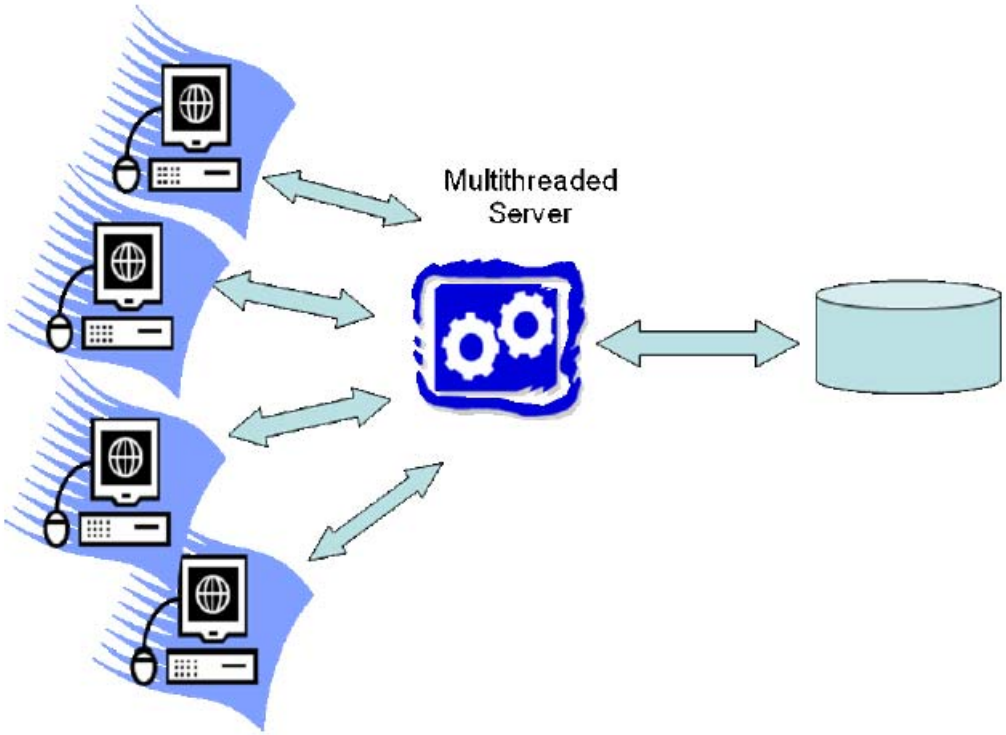

图 2-2 每个 DBMS 工作者拥有一个线程的模型

2.1.3 进程池

这个模型是“每个 DBMS 工作者一个进程”这种模型的变体。我们知道，每个 DBMS 工作者拥有一个进程的模型的优点是编程实现较为简单，但是，每一个连接都需要一个进程却是一个缺点。使用进程池（如图 2-3 所示）后，不必每个 DBMS 工作者都分配一个进程，而是由进程池来管理所有 DBMS 工作者。一个中央进程控制所有的 DBMS 客户端连接，每个从客户端到来的 SQL 请求将被分配一个进程池中的进程。SQL 请求处理完之后，结果返回客户端，进程回到缓冲池中准备分配给下一个请求。缓冲池的大小是一定的，且通常不可变。如果一个请求到来而没有进程空闲，那么新的请求必须等待进程。

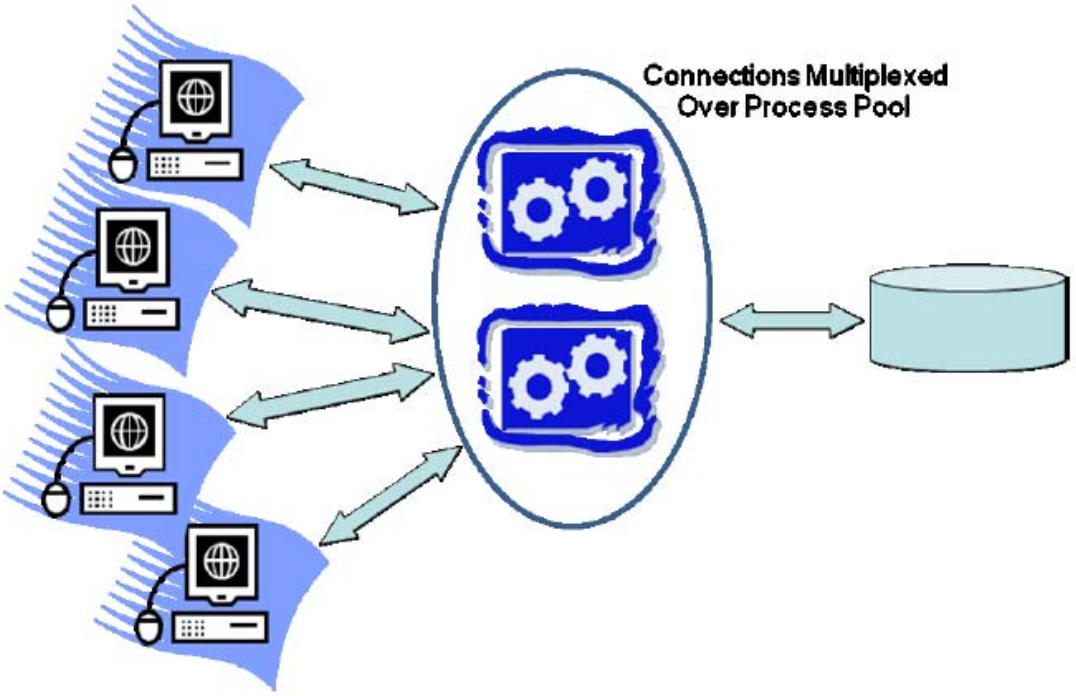

图 2-3 进程池

进程池拥有每个 DBMS 工作者一个进程模型的所有优点，而且，由于只需要少量的进程，所以，其内存使用效率也很高。进程池一般被设计成大小可动态变化的，以此应对大量的高并发请求。当请求负载较低时，进程池可以缩减为较少的等待进程。跟每个 DBMS 工作者一个进程一样，很多现代 DBMS 产品也支持进程池。

2.1.4 共享数据和进程空间

上述的每一个模型都试图尽可能地独立处理并发客户端请求。但是，DBMS 工作者完全地独立工作是不可能的，因为它们处理的是共享的数据库。对于每个 DBMS 工作者拥有一个线程的模型，数据共享较为简单，因为，所有线程共享内存空间。在其它模型中，共享内存被用来存放共享数据结构。在所有三个模型中，数据必须从 DBMS 传送给客户端。这使得所有的 SQL 请求都必须传给服务器来处理，然后得到的结果需要返回给客户端。要实现这个功能，简单地说，就是使用不同的缓冲区。两种主要的缓冲区是磁盘 I/O 缓冲区和客户端通信缓冲区。我们简要地介绍一下它们以及它们的管理方式。

磁盘 I/O 缓冲区：最常见的工作者之间的数据依赖是读写共享的数据。所以，工作者之间的 I/O 中断是必要的。有两种不同的 I/O 中断需要分别考虑：(1)数据库请求；（2）日志

请求。

- **数据库 I/O 中断请求：缓冲池**。数据库提交的数据存放在数据库缓冲池中。对于每个 DBMS 工作者拥有一个线程的模型，缓冲池中的数据以堆结构的形式存放，允许共享该存储空间的所有线程来访问。在另外两种模型中，缓冲池被作为所有进程共享的空间。总的来说，三种模型都是将缓冲池中的数据共享给所有的线程或进程。当一个线程需要从数据库中读取一个数据页时，它产生一个 I/O 请求并且获得空闲的内存空间来存放得到的数据。当需要将缓冲区中的页存入磁盘时，线程产生一个 I/O 请求将缓冲池中的页存入磁盘中的目标地址处。在第 4.3 节中我们将详细讨论缓冲池。
- **日志 I/O 请求：日志尾部**。数据库日志是存储在硬盘上的一组条目（entry）。一个日志条目是在事务处理过程中产生的，它们会被暂时存储在内存队列中，然后，周期性地按 FIFO 顺序刷新至日志磁盘中。这个队列常被称为日志尾部。在许多系统中，有一个独立的进程或者线程负责将它刷新至磁盘。

对于每个 DBMS 工作者拥有一个线程的模型，日志尾部只是一个堆数据结构。在另外两种模型中，它们的设计方式比较类似。一种方法是一个独立的进程负责管理日志。日志记录通过共享内存或者其他有效的通信协议与日志管理器进行交互。另一种方式是，日志尾部像上文中提到的缓冲池那样分配给共享内存。关键的一点是，所有的线程或者进程在处理客户端请求时，需要写日志记录并将日志尾部刷新至磁盘。

一种重要的日志刷新方法是提交事务刷新。一个事务在日志记录被刷新到日志存储器之前不能被成功提交。这意味着，客户端代码必须等待日志刷新，DBMS 服务端在刷新之前必须保存所有资源。日志刷新请求可能会被推迟一段时间，这样可以实现在一个 I/O 请求中批量提交记录。

客户端通信缓冲区：SQL 通常被用于“拉”（pull）模型，也就是说，客户端通过重复发送 SQL FETCH 请求，不断地获取结果元组，SQL FETCH 每个请求会获取一个或多个元组。大多数 DBMS 尽可能地在 FETCH 流到来之前做一些数据预提取工作，以此确保在客户端请求之前将结果存入队列。

为支持预提取功能，DBMS 工作者使用客户端通信套接字作为元组队列。将来更复杂的方法是实现客户端游标缓存功能，并使用 DBMS 客户端存储近期即将被访问的结果，而不是将不依赖于操作系统通信缓冲区。

锁表：锁表由所有的 DBMS 工作者共享，由锁管理器实现数据库锁机制。锁共享技术类

似于缓冲池共享，并可以用类似的方法实现 DBMS 开发需要的其他数据结构共享。

2.2 DBMS 线程

前文中我们简单描述了 DBMS 进程模型。我们假设系统线程有良好的的性能，DBMS 着眼于单处理机系统。在本节剩余的篇幅中，我们取消第一个假设，然后来看看它对 DBMS 实现的影响。接下来讨论多进程和并行化操作。

2.2.1 DBMS 线程

当前的大多数 DBMS 技术，都离不开 19 世纪 70 年代开始的系统研究和 19 世纪 80 年代开始的商业化发展。在数据库开发的早期阶段，标准操作系统的技术无法用到数据库开发中。高效的操作系统线程支持就是其中一种技术。直到 19 世纪 90 年代，操作系统线程的出现和广泛应用，才使得数据库开发发生了很大的变化。即便是今天，操作系统线程开发都没有很好地支持 DBMS 的工作负载[31,48,93,94]。

受历史发展的影响以及其他一些原因，很多广泛应用的 DBMS 在开发上并不依赖于操作系统线程。一些完全不涉及线程，而是使用每个 DBMS 工作者拥有一个进程或者进程池模型。其他的 DBMS，比如一个 DBMS 工作者拥有一个线程的模型，需要一个方案来应对那些没有好的内核线程的系统。一些技术领先的 DBMS 解决该问题的方式是，开发了自己的高效轻量级的线程包。这些轻量级线程或者说 DBMS 线程，取代了上文提到的系统线程。每个 DBMS 线程都管理自己的变量，通过非中断的异步接口来编写中断操作，并通过调度器来调度任务。

轻量级线程并不是一个新的概念，在文献[49]中以一种怀旧的方式介绍了它，这个概念如今在针对用户接口的循环事务编程方面被广泛应用。此概念在操作系统相关文献[31,48,93,94]中也被不断研究。这个架构为我们提供了快速的任务切换和易移植性，它的代价是需要在 DBMS 中重复实现许多操作系统逻辑。

2.3 标准练习

在现今主要的 DBMS 中，我们可以看到 2.1 节中介绍的所有三种架构以及它们的一些有趣的变化。IBM DB2 是支持四种进程模型的很有趣的例子。对于线程支持良好的操作系统，DB2 默认的模型是每个 DBMS 工作者拥有一个线程，它也支持 DBMS 工作者多路复用一个线

程池。当运行在没有线程支持的系统上时，DB2 默认的模型是每个 DBMS 工作者拥有一个进程，同时也支持 DBMS 工作者多路复用一个进程池。

我们对 IBM DB2、MySql、Oracle、PostgreSQL 和 Microsoft SQL Server 所支持的进程模型做一个总结：

每个 DBMS 工作者拥有一个进程：

这是最直接的进程模型，同时也得到了广泛的应用。DB2 在不支持线程的系统上，默认使用每个 DBMS 工作者拥有一个进程的模型，在支持线程的系统上，默认使用每个 DBMS 工作者拥有一个线程的模型。这也是 Oracle 进程模型的默认设计。Oracle 也支持上文提到的进程池。PostgreSQL 在所有的操作系统上都只运行每个 DBMS 工作者拥有一个进程模型。

每个 DBMS 工作者拥有一个线程：

该模型有两个种类，在目前来看十分高效：

1. 每个 DBMS 工作者拥有一个系统线程：IBM DB2 运行在有良好系统线程支持的系统上时，默认使用该模型；MySQL 同样使用该模型。
2. 每个 DBMS 工作者拥有一个 DBMS 线程：在这个模型中，DBMS 工作者的调度是通过系统进程或线程去调度轻量级线程来实现的。这个模型规避了系统调度的一些问题，但是，它开发代价较高，没有良好的开发工具，也需要运营商的长期维护。它由两种子模型：

   ①通过系统进程来调度 DBMS 线程：轻量级线程的调度是由一个或多个系统进程来完成的。Sybase 和 Informix 支持该模型。现今很多系统采用这个模型来开发实现 DBMS 线程的调度，以此调度 DBMS 工作者发挥多处理器的作用。然而，并不是所有采用这个模型的系统都实现了线程迁移：将 DBMS 线程再分配给不同的系统进程（如考虑到负载均衡）。

   ②通过系统线程来调度系统进程：微软 SQL Sever 以可选择的方式支持这种模型（默认的模型为 DBMS 工作者多路复用线程池）。SQL Sever 中这个选项叫做 Fibers，被用做应对高频事务处理，但是很少被使用。

进程/线程池：

在这个模型中，DBMS 工作者共用一个进程池。随着系统线程支持性能的不断提高，依赖线程池而不是进程池的新的变种开始出现。在后面这种变种模型中，DBMS 工作者复用系统的一个系统线程池：

1. DBMS 工作者共用进程池：这个模型相对于每个 DBMS 工作者拥有一个进程的模型而

言，有更高的内存使用效率。在系统没有良好的线程支持的情况下，它也易于与系统对接，并且在大量用户的情况下表现稳定。这个模型是 Oracle 数据库的可选项，Oracle 建议在大量用户并发操作的情况下使用。Oracle 默认的模型是每个 DBMS 工作者拥有一个进程。这两个选项对于 Oracle 所适用的大多数操作系统而言都是支持的。

2. DBMS 工作者共用线程池：微软 SQL Server 默认使用该模型，且超过 99%的 SQL Server 产品使用该模型来运行。为有效支持上万的并发连接用户，SQL Server 提供可选支持，允许系统线程调度 DBMS 线程。

我们将在下一章中谈到，现今大多数商业数据库支持内部查询并行化：并行多线程执行一条查询语句或者该语句的一部分。这一节中我们之所以提到这一点，是因为并行化内部查询，使得多个 DBMS 工作者暂时分配给一条 SQL 请求。除了一个客户端连接将拥有多个 DBMS 工作者来处理其请求之外，之前提到的进程模型与该情况并无冲突。

2.4 准入控制

随着多用户系统负载不断升高，吞吐量将达到上限。在这一点上，当系统挂起时应当减少吞吐量。对于操作系统来说，“挂起”经常是由内存问题造成的：DBMS 无法保证在缓冲池中存放数据库页数据的工作空间，导致出现不断的页替换情况。在数据库系统中，一些特殊的操作比如排序和哈希连接往往会造成大量的内存消耗。有时候，系统挂起也会发生在锁竞争上：事务处理发生死锁需要回滚并重启。因此，任何好的多用户系统都有准入控制机制，在系统没有充足资源的情况下，新的任务不被接受。如果拥有一个好的准入控制器，系统将在过载情况下发生比较优雅的性能衰退：事务延迟将随着到达率的增加而适当增加，但吞吐量一直保持在峰值。

DBMS 的准入控制可以在两个层面上来实现：

第一个层面是，一个简单的准入控制可以通过进程来确保客户端连接数处于一个临界值之下。这就避免了网络连接数这类基础资源的过度消耗。在一些 DBMS 中，这种控制并没有被提供，我们可以假设该功能被系统的其他部分实现了，如应用层、事务处理层或者网络服务层。

第二个层面是，直接在 DBMS 内核关系查询处理器上实现。准入控制器在查询语句转换

和优化完成后执行这一步操作，由该操作来决定，是否要推迟执行一个查询，是否要使用更少的资源来执行查询，以及是否需要额外的限制条件来执行查询。准入控制器依靠查询优化器的信息来执行，如查询所需的资源以及系统并发资源等信息。特殊情况下，优化器的查询计划可以：（1）确定查询所需要的磁盘设备以及对磁盘的 I/O 请求数；（2）根据查询中的操作以及要求的元组数目判断 CPU 负载；（3）评估查询数据结构的内存使用情况，包括在连接和其他操作期间的排序和哈希所消耗的内存。正如之前描述的那样，上面的第 3 点对于准入控制而言是最为关键的，因为，内存压力通常是引起“抖动”的主要原因。因此，许多 DBMS 把内存使用情况和活跃的 DBMS 工作者的数量作为主要的准入控制标准。

2.5 讨论及其他资料

进程模型的选择对 DBMS 的表现有很大的影响。因此，商业数据库往往支持多个进程模型。从工程师的角度来看，在所有操作系统以及各种负载级别上使用单进程，很显然会更加简单明了。但是，由于操作系统和应用场合的多样性，三种 DBMS 都选择了支持多种模型。

展望未来，由于硬件瓶颈的变化以及网络的规模和多样性，近年来很多人致力于为服务器系统研究新的进程模型[31,48,93,94]. 在这些设计中，有一种想法是把一个服务器系统分解成一系列独立调度的引擎，并且在这些引擎之间进行异步和批量的消息传输。这有一些类似于上文中提到的进程池模型，即在多个请求之间重用工作者单元。这类最新研究的主要创新是，用一种更加小范围的、面向特定任务的方式来执行一些功能。这使得 DBMS 工作者与请求的关系变为多对多的，即一个查询的工作由多个 DBMS 工作者来完成，一个 DBMS 工作者负责多个查询的同样任务。这种结构带来了更为灵活的调度方式，比如，它允许一个工作者完成许多查询的任务（可能提高系统的整体吞吐量），或者允许一个查询由多个工作者共同完成（可能减少查询操作的延时）。在一些例子中，它在处理器局部性方面显得更有优势，在硬件未命中的情况下，能更好地防止 CPU 空闲。更多关于这一点的研究可以参考 StagedDB 的研究项目[35]，这也是很好的阅读材料。

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