计算机科学1994 Vol. 21№. 6

并行数据库系统：目标、体系结构及研究方向

廖朝晖 张 鹏 王 珊
（中国人民大学数据与知识工程研究所 北京100872）

本文主要参考《Distributed and Parallel Database System》杂志1993年第4期上刊登的论文"Parallel DBs：Open Problems and New Issues"并结合自己的研究工作综合编写而成。全文综述了并行数据库系统的目标、体系结构及尚未解决的问题。

一、引言

计算机系统性能价格比的不断提高迫切要求硬件和软件结构的改进。硬件方面，传统上靠提高微处理器速度和缩小体积来提高性能价格比的方法正趋于物理的极限；磁盘技术的发展滞后于微处理器的发展速度，使得磁盘"I/O"瓶颈问题日益突出。软件方面，数据库服务器对大型数据库各种复杂查询和联机事务处理的支持使得对响应时间和吞吐量的要求顾此失彼。同时，应用的发展超过了主机处理能力的增长速度。数据库应用（DSS、OLTP 等）的发展对数据库的性能和可用性提出了更高的要求，能否为越来越多的用户维持高事务吞吐量和低响应时间已成为衡量 DBMS 性能的重要指标。

计算机多处理器结构以及并行数据库服务器的实现为解决以上问题提供了极大可能。将数据库管理与并行技术相结合，可以开发多处理器结构的优势，从而提供比相当大型机系统要高得多的性能价格比和可用性。通过将数据库在多个磁盘上分布存储，可以利用多个处理器对磁盘数据进行并行处理，从而解决了磁盘"I/O"瓶颈问题。同样，潜在的主存访问瓶颈也可通过并行性而获得解决。并行数据库系统通过开发查询间并行性（不同查询并行执行）、查询内并行性（同一查询内的操作并行执行）以及操作内并行性（子操作并行执行）可以大大提高查询效率。

为了充分开发多处理器硬件，并行数据库的设计者必须努力开发面向软件的解决方案。为了保持应用的可移植性，这一领域的多数工作都围绕着支持 SQL 查询语言进行。目前已经有一些关系数据库产品如 Teradata 的 DBC 和 Tandem 的 Non StopSQL 不同程度地实现了并行性。INGRES、ORACLE 等商用 RDBMS 产品也都在并行计算机上得以实现。

但是上述系统一般都依赖于简单的并行技术（如"完全划分"技术），并且对各结点的负荷量给予了过多的假设（如 TPC－B 基准程序）。一系列尚未解决的问题仍然阻碍着对并行系统能力的充分开发。同时，并行系统在 CAD/CAM、CASE、OIS 等复杂领域中的应用也带来一系列的新问题，需要我们去探索和解决。

可以相信，多处理器系统是现在以及未来开放系统数据库服务器平台的优选结构，并行数据库系统必将在九十年代的商用信息系统中占据主导地位。

二、目标及功能结构

一个并行数据库系统应该实现如下目标，以提供比同等大型机要高得多的性能价格比。

（1）高性能　并行数据库系统可通过面向数据库的操作系统支持、并行性、优化以及负载均衡等相互补充的方案来获得高性能。让操作系统面向数据库操作可以简化底层数据库功能的实现，从而降低成本；并行性能够增加吞吐量（通过查询间并行性）和减少事务响应时间（通过查询内和操作内的并行性）；为了使并行性所带来的通信开销降至最低，对查询进行优化和并行化是至关重要的；负载均衡是指系统将一定量的工作负荷在多个处理器之间均匀分配，这可通过适当设计物理数据库的静态方式或在运行时调整负荷的动态方式来实现。

（2）高可用性　并行数据库系统可通过数据复制来增强数据库的可用性。这样，当某一磁盘损坏时，该盘上数据在其它磁盘上的副本仍可供使用，且无需额外开销（与基于日志的恢复不同）。当然，对于数据复制的支持需要有控制协议来保证系统数据的一致性，例如最常用的 ROWA（Read One Write All）协议。此外，如果磁盘失败频繁发生，则系统负载会严重失衡，因此数据复制应与数据划分技术相结合以保证当磁盘失败时系统仍能并行访问数据。

（3）可扩充性　并行数据库系统环境可以更容易

地满足增大数据库规模或提高系统性能的需要。这里，数据库系统的可扩充性指系统通过增加处理和存储能力而平滑地扩展性能的能力。理想情况下，并行数据库系统应具有两个方面的可扩充性优势：线性伸缩和线性加速。其中，线性伸缩指当数据库规模与处理和存储能力都呈线性增加时，系统性能将保持不变；线性加速指当数据库规模不变而处理和存储能力呈线性增加时，系统性能也线性增长。

一个并行数据库系统可以作为服务器面向多个客户机进行服务。典型的情况是，客户机嵌入特定应用软件，如图形界面、DBMS 前端工具4GL 以及客户/服务器接口软件等。因此，并行数据库系统应该支持数据库功能、客户/服务器接口功能以及某些通用功能（如运行 C 语言程序等）。此外，如果系统中有多个服务器，那么每个服务器还应包含额外的软件层来提供分布透明性。

对于客户/服务器体系结构的并行数据库系统，它所支持的功能一般可分为三个子系统：

·会话管理子系统：提供对客户与服务器之间交互能力的支持。

·请求管理子系统：负责接收有关查询编译和执行的客户请求，触发相应操作并监督事务的执行与提交。

·数据管理子系统：提供并行执行编译后查询所需的所有底层功能，例如并行事务支持、高速缓冲区管理等。

上述功能构成类似于一个典型的 RDBMS，不同的是并行数据库系统必须具有处理并行性、数据划分、数据复制以及分布事务等的能力。依赖于不同的并行系统体系结构，一个处理器可以支持上述全部功能或其子集。

三、体系结构

并行数据库系统的实现方案多种多样。譬如有这样两种选择：一种是直接移植已经存在的 RDBMS，这样仅需修改与操作系统的接口程序；另一种是重新设计一个新的包含并行处理和数据库系统功能的硬/软件体系结构。这种系统效率很高，但难以移植。因此设计一个并行数据库系统时，需要在效率和可移植性之间权衡折衷。下面分别介绍三种基本的并行系统体系结构，并从性能、可用性和可扩充性等三个方面来比较这些方案。

3.1 Shared-memory（共享内存）结构

如图1所示的 Shared-memory 方案中，任意处理器可通过快速互连（高速总线或纵横开关）访问任意内存模块或磁盘单元，即所有内存与磁盘为所有处理器共享。IBM3090、Bull 的 DPS8 等大型机以及 Sequent、Encore 等对称多处理器都采用了这一设计方案。

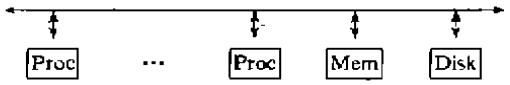

图1 Shared-memory 结构

并行数据库系统中，XPRS、DBS3以及 Volcano 都在 Shared-memory 体系结构上获得实现。但是迄今为止，所有的共享内存商用产品都只开发了查询间并行性，而尚未实现查询内并行性。

Shared-memory 方案的优势在于实现简单和负载均衡。由于元信息（目录表）和控制信息（锁表等）可被所有处理器共享，因此数据库软件的编制与单处理机情形区别不大。特别是，查询间并行性的实现不需额外开销，查询内并行性的实现也不太困难。这种系统可以基于实际负荷来给各处理器动态地分配任务，因而可以很好地实现负荷均衡。

但是这种结构的系统由于硬件成员之间的互连很复杂（每个处理器要与每个内存模块和每个磁盘相连），因而成本高昂。并且，为了避免由于内存访问冲突增多而导致系统性能下降，结点数目必须限制在几十个以内。此外，内存的任何错误都将影响到多个处理器，因此系统可用性不是很好。

3.2 Shared-disk（共享磁盘）结构

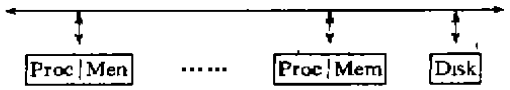

图2 Shared-disk 结构

如图2所示的 Shared-disk 方案中，各处理器拥有各自私用的内存，但共享共同的磁盘。每一处理器可以访问共享磁盘上的数据库页，并将之拷贝到各自的高速缓冲区中，为避免对同一磁盘页的访问冲突，应通过全局锁和协议来保持高速缓冲区的数据一致性。

采用这一方案的数据库系统有 IBM 的 IMS/VS Data Sharing 和 Dec 的 VAX DBMS 和 Rdb 产品。在 Dec 的 VAX 群集机和 NCUBE 机上实现的 ORACLE 系统也采用这一方案。

Shared-disk 方案具有成本低、可扩充性好、负载均衡、可用性强以及容易从单处理机系统迁移等优点。首先，具有这种结构的系统可以使用标准总线互连，因而成本较之 Shared-memory 要低；其次，当每个处理器有足够大的高速缓存空间，对共享磁盘的访问冲突可降至最低，因此可扩性就很好，结点数目可达数百个；单个处理器结点的失败不致影响其它处理器

的工作，因而系统可用性高；由于磁盘数据不需重新组织，所以从一个单处理机的集中系统向一个Shared-disk系统迁移简便而直接；至于负荷均衡的理由则与Shared-memory结构相同。

该方案的不足在于实现起来更加复杂以及存在潜在的性能问题。这是由于它需要诸如分布锁和两段提交等分布式数据库系统协议，并且维护高速缓冲区中的数据一致性会带来额外的通讯开销，此外对共享磁盘的访问是潜在的"瓶颈"。

3.3 Shared-nothing(分布内存)结构

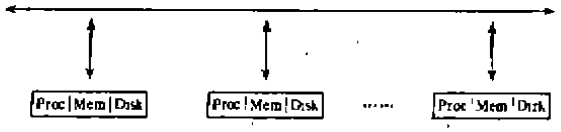

图3 Shared-nothing结构

如图3所示的Shared-nothing方案中，每一处理器拥有各自的内存和磁盘。由于每一结点可视为分布式数据库系统中的局部场地(拥有自己的数据库软件)，因此分布式数据库设计中的多数设计思路，如数据库分片、分布事务管理和分布查询处理等，都可以在本方案中利用。

采用Shared-nothing方案的有Teradata的DBC和Tandem的Non StopSQL产品以及Bubba、Eds、Gamma、Grace、Prisma和Arbre等原型系统。所有这些系统都开发了查询间和查询内的并行性。

与Shared-disk结构一样，Shared-nothing结构成本较低。而且，Shared-nothing系统可扩充性更强，结点数可达数千个，例如Teradata的DBC可拥有1024个处理器。此外，具有该结构的系统可以通过在多个结点上复制数据从而实现高可用性。

该方案的不足在于实现复杂以及结点负荷难以均衡。Shared-nothing结构不同于前面两种方案，它只能基于数据位置而不能按系统的实际负荷来均匀分配工作量。并且，系统中新结点的加入将导致重新组织数据库以均衡负载。

3.4 方案比较

对并行数据库体系结构的比较应从性能、可用性和可扩充性这三个方面进行。简而言之，对于20个处理器以下的小型配置，Shared-memory结构由于易于实现，负载均衡，可以提供最佳的性能优势，而Shared-disk和Shared-nothing在可用性和可扩充性方面要胜于Shared-memory方案。看来，Shared-nothing方案是高端系统的唯一选择。对于中小型系统，Shared-memory和Shared-disk这两种方案则较为适合。

四、尚未解决的问题

尽管并行数据库系统从研究到产品都颇为引人注目，但仍有一些未解决的问题阻碍了对多处理器系统能力的完全开发。这些问题涉及到并行体系结构选择，操作系统支持，基准测试程序，数据分布，并行数据库语言和并行查询处理等多个方面。

4.1 体系结构选择

虽然我们已经了解了三种基本体系结构的特征和优缺点，但还有一些体系结构却并不属于上述任何一种基本模型，例如仅有部分内存模块或磁盘被处理器共享的情形，我们将三种基本体系结构以外的情形笼统称为混合型体系结构。

并行数据库方面的多数研究课题在假定一种体系结构设计(通常假定为Shared-memory或Shared-nothing)的前提下都已得到验证。这种假定既取决于设计者的意愿和偏好，也出于实现简便方面的考虑。因为使用运行时并行化来简单地扩充一个单处理器DBMS设计即可得到Shared-memory设计。而Shared-nothing设计能够充分利用和扩充分布式数据库中已经实现的技术。然而，由于这两种设计在可扩充性和负载均衡方面分别存在困难。我们考虑一种或许更有效的混合型体系结构，即系统从整体上看是一种Shared-nothing设计，各结点彼此不共享内存和磁盘，但结点自身却可设计为Shared-memory的多处理器结构。在这种结构中，某几个结点的功能可能很强，这使得数据分布的解决方案更为明确。不过，这种结构也带来了新的问题：出于可扩充性和可伸缩性的需要，如果选择功能强大的Shared-memory结点，结点数目会受到限制，而选择功能稍弱的Shared-memory结点，则允许有更多的结点数目，这两种选择孰优孰劣呢？在实际的系统中，Encore93遵循了第一种选择，它仅允许有数个Shared-memory结点，但每个结点包含有32个处理器；Teradata的P90则遵循了后一种选择，该系统可拥有512个结点，每一结点包含4个处理器并通过快速树型总线(Bynet)相连。

总之，如何根据负载类型、应用复杂性以及数据库规模等因素来选择一种最佳的并行数据库体系结构，这一问题仍需进一步研究。

4.2 操作系统支持

对于一个专用系统，由操作系统对并行数据管理提供专门支持所需要的代价高昂，但获得的效益也很大。一般而言，操作系统可以通过两种方式来支持并行数据库系统。一种方案是重新建立一个大型的专用操作系统以支持并行数据管理，例如Bubba操作系统以统一的虚地址来规范地表示所有数据，尽管这种方

法可带来最佳的系统性能，但限制了并行系统只能执行数据库操作而不能执行 C 或 COBOL 程序。另一种方案则试图利用新一代操作系统微核，如 Chorus 或 Mach 等，并且扩充它们，以提供面向数据库功能的高效支持。在这种方案下，面向数据库的操作系统（如 UNIX 操作系统）仅仅只能作为整个并行系统的一个子系统，因而也不能很好地支持非数据库的应用。

4.3 基准测试程序

评价一个并行数据库系统（体系结构）需要特定的基准程序。对于给定的工作负荷，基准测试程序是评估系统性能价格比的唯一公正的方法。现在已经有了针对 DBMS 和事务处理系统的标准的基准测试程序，它们主要来自 TPC、Wisconcin 以及工程数据库基准程序等研究成果。其中 TPC 用于量度一个简单（信贷）事务在给定负荷下的吞吐量。Wisconcin 用于量度复杂的（决策支持）查询的响应时间，AS³AP 则可对包括简单和复杂事务的混合负荷情形进行量度。对于并行数据库系统，在基准程序方面有很多工作要做，例如对线性加速比和线性伸缩比的量度。

4.4 数据分布

并行数据库系统必须适当分布数据以利于负载均衡。我们可以通过应用数据的某些属性并基于一个函数（hash 函数或范围索引）将关系进行水平划分，然后将得到的各个独立的数据集分配到不同的磁盘上，显然这类似于分布式数据库中的水平分片技术。这种数据分布技术不仅有利于均衡负荷，也有益于获得查询间/内并行性，因为不相关的查询或者同一查询内的独立操作可以同时处理不同数据集中的数据。对关系数据的划分可以依据单属性或者多属性，对后一种情形，一个精确的多属性查询的匹配处理可以在单结点上处理，而无需额外的通信开销。对数据的划分可以基于 hash 函数或者范围索引，前者比较节省存储，但只能提供对精确匹配查询的直接支持，而后者虽需要一定空间开销，但可支持范围（区间）查询。

按照存放关系的结点数目的不同，数据划分技术可分为完全划分和变量划分两种类型。前者将每一关系划分存储到所有结点上，这种方法不适合于小关系以及结点数目大的系统；后者将每一关系划分存储到特定数目的结点上，其中结点数目是关系大小和访问频率的一个函数。

并行数据库系统经过一段时间的运行以后，如果负载均衡程度明显降低，则有必要进行高效的（通过并行性）、联机的（不中断其它事务的执行）动态数据重组。然而现有的数据库产品还只能进行静态的数据重组。另外，数据重组应对并行系统中经过编译的程序透明，即程序不会因为数据重组而必须重新编译。因此经过编译的程序应独立于数据的位置。这无疑给优化器的设计增加了难度。

数据分布方面的一个难题是关于错开数据（Skewed data）的处理。错开数据会导致非规范的数据集，并影响到负荷的均衡性。由于混合型体系结构中各结点的存储和处理能力有差异，我们可以利用这种"差异"来解决错开数据问题。另外一种解决方案是适当处理非规范的数据集，对大的数据集可以进一步划分，此外，逻辑结点和物理结点在概念上分离也有利于解决错开数据问题，因为一个逻辑结点可以对应多个物理结点。

与数据分布相关联的一个复杂问题是数据复制。对于数据复制，一种自然的考虑是为同一份数据保持两个副本并存储于不同的结点上（一个为基数据，一个备份数据）。但当某一结点失败时，拥有该结点数据备份的结点上的工作量将成倍增加，这就破坏了负载均衡。为避免这种情况，可采用其它的数据复制策略，例如，Teradata 的交错存储策略将基数据的备份划分存储到多个结点上，这样当基数据所在的结点失败时，其工作负荷被多个拥有（子）备份的结点均匀负担。这种方法尽管保持了负荷均衡，但重建数据的代价较高，一种更好的解决方案是 Gamma 的链式分布存储策略。总之，在设计数据分布时必须考虑数据复制，而且应把它作为一个优化问题加以研究。

4.5 并行数据库语言

并行的数据处理通常有如图4所示几种形式：

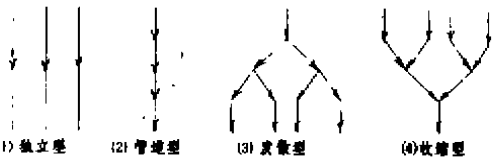

图4 并行处理的形式

目前，并行数据处理系统主要采用"分而治之"的策略，这一技术包括问题的分解及问题的解决两个步骤。由关系代数描述的所有操作都可以很自然地描述为"分而治之"的计算模型，此模型包含了上述所有类型的结点。例如，对于问题的分解，我们可以采用发散型结点；对于集合操作可以用独立型结点表示；对于流操作则可以采用管道型结点描述；对于结果的聚集，我们可以利用收缩型结点表述。

然而，基于"分而治之"策略的并行数据库系统目前面临下面一些问题：

1. 对流处理操作的支持不足 在关系模型中，元组之间是无序的，关系代数并不支持流操作，换句话

说，如果采用传统的关系数据语言，我们将无法利用它实现管道型的并行化。然而，在那些用于实现关系代数语言的低级过程化的语言中，流处理却被广泛运用。这说明，在关系代数语言中实现流处理是完全可能的。

2. 对数据划分的支持不足　并行数据处理，特别是在 Shared-nothing 体系结构中，需要尽可能地利用数据划分来提高能力。数据的分布存放可以将数据的全局操作分散到多个处理器上并行执行。在许多情形下，为了得到最终结果，各处理器之间还需要互相交换中间结果。由这样的要求所推动，并行数据语言应该具有定义数据划分的能力——即定义数据如何分布存储及分布的结果如何集中为最终的结果。这种能力对于并行数据库系统是否能自动地实现有效的并行化处理是极其重要的。

SQL 语言以其高度非过程化成为关系数据库系统的标准语言，但是，它对于并行数据库系统的支持尚显不足。为了能使 SQL 语言更好地支持并行数据库系统，需要对其作某些扩充，以适应上述四种形式的并行处理。目前，Bubba 系统中的 FAD 语言提供了可以描述发散型和收缩型并行操作的操作符；而 SVP 模型则更为全面，提供了描述所有上述四种并行处理的操作。SVP 模型很可能会成为并行数据语言的形式化基础。

4.6　并行查询处理

并行查询处理指的是将概念上集中描述的查询转换为一个正确高效的执行计划，并且利用并行体系结构并行地执行之。并行查询处理包括优化、并行化和执行三个步骤。每一步骤目前都面临着一些尚未解决的问题。由于数据分布技术是并行查询处理的基础，我们将首先讨论数据分布方面的一些问题。

1. 数据分布　关于如何有效地利用数据的分布存储并行处理数据库查询，目前已有了很多算法。然而，许多算法还不能很好地处理错开数据。

将一个面向集合的操作分解为单指令多数据流的形式，可以很容易得到操作间的并行化。在分解过程中，最简单的一个原则是"哪儿有数据就在哪儿执行"，即将操作送至所有操作所涉及的数据所在的结点上，从而得到操作的并行处理。例如，对于 Select(R)，如果 R 分布于 n 个结点，可以将 Select 操作发送到这 n 个结点上，即 $\mathrm{Select}(R)=\bigcup_{i=1}^{n}\mathrm{Select}(R_i)$。当然，如果 Select(R) 中的谓词包括涉及 R 分布的谓词时，也许只需要把 Select 操作送至少一些的结点。对于二元连接操作来说，就比较麻烦了。只有当连接表在 n 个结点上的分布都是一致的时候，才可以将连接操作传送至每一个结点。例如，$\mathrm{Join}(R,S)=\sum_{i=1}^{n}\mathrm{Join}(R_i,S_i)$，当且仅当 R 和 S 都是在连接属性上根据相同的分布函数分布于 n 个结点上。如果这一条件不能被满足，情况就比较复杂了。目前，大部分连接算法在上述条件不满足时，都是重新组织这两个表，以使它们满足此条件。

对于二元操作，如 Join、Union、Difference 的并行化，hashing 技术是最主要的技术。对于 Sort 操作，可以采用保序的 hashing 技术加以实现，其中需要注意的一个问题是，为了确保负载均衡，必须使各处理器所处理的数据量基本一致。

2. 查询优化　在并行数据库系统中，查询优化也面临着许多新问题，最主要的是代价函数的构造。

由于策略搜索空间庞大，代价模型的构造异常复杂，从而导致了极高的优化代价。现有的一些系统，往往都对代价函数的构造人为地加了许多假设性限制，在这些限制下，尽管可以构造出一些代价模型，但是优化后执行计划的效率不太高。除了策略空间庞大外，代价函数需要考虑的因素也很多，而其中某些因素往往又是互相矛盾的，例如响应时间(RT)与总执行时间(TT)，对于 TT 的优化可以提高系统的吞吐量；而对于 RT 的优化在获得较好的单个事务的响应时间的同时却会损害整个系统的吞吐量。在混合的工作平台上，查询种类很多，各查询优化的目标却不尽相同，甚至可能相互矛盾。如何实现这个问题实际上属于数学上多目标规划的范畴。

构造代价函数时还应考虑到在策略空间的搜索上尽可能利用启发式方法将许多不必要的执行计划分支裁剪掉，从而可以尽量缩小搜索空间的代价，进而减少优化的代价。

另一个问题是，查询优化的代价与生成的执行计划的质量之间还有一个权衡。查询优化的代价比较高，一般来说生成的执行计划质量就比较高；反之，执行计划的质量就比较低。对于那些只执行一次的查询来说，使用代价很高的优化策略就显得不太合适。这就要求，在混合的工作平台上，查询优化应该能够根据不同的查询类型选择不同的执行计划空间的搜索策略。

此外，在并行体系结构下，如何并行地执行优化本身也是一个很有趣的问题。

3. 并行化　对于执行计划的并行化，目前有静态和动态两种方法。DBS3 系统采用前者，而 XPRS 则采用后者。

采用静态方法，允许对并行程序进行分散控制，从而使得优化的效率比较高；但是，为了保证负载均衡，有些因素，如物理处理器的分配，只能在执行时才

能作出决定，这对于静态并行化确实是一个比较棘手的问题。

采用动态方法，需要先进行集中式优化，即优化时并不考虑并行时能利用的处理器等因素，得到一个执行计划，到运行时才将此执行计划并行化处理。这一方法实现简单，而且易于实现负载均衡。然而，它的问题是，集中式优化过程不一定能得到一个较优的执行计划。

由此可见，如何将静态并行化与动态并行化的优点结合起来，以得到一个更好的并行化方法，尚有许多工作要做。

4、执行　执行部分面临的问题是事务调度、事务的启动与终止。

事务调度的困难在于一个事务可能被多个处理器同时执行，多个事务间有可能协同工作，其间的关系是很复杂的。正如4.5节所述，并行化处理可以有四种形式，事务的调度策略必须能充分考虑到这几种形式。此外，在混合的工作平台上，事务的类型是多样的，可能是联机事务也可能是决策支持事务，可能是长事务也可能是短事务，不同类型的事务所要求的目标也是不同的。这也需要事务的调度能够充分顾及到事务的类型，以满足不同事务的需要。决策支持事务往往会申请许多锁，这将会损害整个系统的吞吐量。如果与此同时还有许多短的联机事务，大量的锁被决策支持事务所占有，显然对联机事务是很不利的。一个实用的解决办法是将整个数据库复制一份，联机事务只访问联机数据库，决策支持事务只访问决策数据库。这种办法的问题在于如何保证两个数据库的一致。有别于这种物理复制整个数据库的办法，另一种办法是支持数据的不同版本，从而利用逻辑方法在同一个数据库上支持两种不同类型的应用，联机事务只访问联机版本，决策支持事务只访问决策版本。这一方法还有待进一步探讨和完善。

事务启动面临的问题在于一个事务可能为多个结点同时执行，执行代码的加载及进程的启动问题都是不容忽视的。事务终止面临的问题与分布式事务管理面临的问题有些类似，需要遵循特定的提交协议。

五、结束语

并行数据库系统利用现代的多处理结构，采用并行技术管理数据，其目标是在性能价格比方面，较相应大型机上的DBMS低得多的前提下，提供一个高性能、高可用性、高扩展性的数据库管理系统。

尽管目前已经有了一些商用的基于SQL的并行数据库系统的产品，但是仍然有许多问题没有得到很好的解决，这些问题阻碍了并行系统充分发挥其能力。

首要的问题是体系结构的问题，即在Shared-memory，Shared-disk，Shared-nothing三种体系结构中，哪种更适合于数据库管理。Shared-memory在负载均衡与易于实现方面优于其它两种结构，而Shared-nothing则在可用性和可扩展性方面优于Shared-disk与Shared-memory。对于小型、中型系统来说，Shared-memory与Shared-disk可能比较有吸引力，而对于一个庞大的系统来说，也许Shared-nothing就较为合适了。另外的一种比较有趣的结构是混合型体系结构。在这种结构中，整个系统是一个Shared-nothing结构，而其中的每一个结点则是Shared-memory结构。这种结构同样也面临着一个选择，到底是在结点数目受限制的条件下，选择功能强大的Shared-memory结点呢，还是选择功能稍弱一些的Shared-memory结点，从而可以拥有更多的结点数目。此外，如果采用磁盘组还会带来更多的问题。

除了体系结构方面的问题外，其它尚需进一步讨论的问题包括：

·操作系统在支持非数据库应用的同时应该能够很好地支持并行数据管理。

·需要有一个侧重于并行数据库系统性能的基准测试，包括混合工作平台上并行系统的线性伸缩比与线性加速比。

·在数据分布方面，为了获得很好的负载均衡，需要有一种分布数据技术能够很好地处理错开数据及数据副本。

·并行数据处理语言应该基于分而治之的技术，并且能够在较高层次上说明各种类型的并行化。

·采用基于代价的优化策略并且自动地进行并行处理的并行查询及优化，并能很好地适应混合的工作平台。

进一步地，如果在并行数据库系统中引入知识和对象的能力，或者说，在知识库系统和面向对象系统中引入并行处理的能力，还会引起一些新的问题，这些问题包括：

·对于并行的知识库系统来说，数据分布技术和并行查询处理技术应该能够支持大规模的知识处理及复杂的演绎查询。

·对于并行的面向对象数据库系统而言，在复杂对象分布存储、事务管理、合适的面向对象的操作系统支持以及对于面向对象查询的并行处理等方面，都会面临许多困难。