# 分布式数据库系统-大数据时代新型数据库技术

## 概论

### 分布库系统的基本概念

节点/场地：分布式数据库系统是地理上分散而逻辑上集中的数据库系统。管理分布式数据库的软件称分布式数据库管理系统。分布式系统通常是由计算机网络将各地理上分散的逻辑单位连接起来。被连接的逻辑单位称为节点（node）或场地(site)。节点/场地可以是单独一台计算机，也可以是由局域网组成。

集中数据库系统：所有数据驻留在同一场地上。如图1.4：我们称传统的数据库系统为集中式数据库系统（DB）。

分布式数据库（DDB）：是分布在一个计算机网络上的多个逻辑相关的数据库的集合。也就是说，分布式数据库是一组结构化的数据集合，逻辑上属于同一系统，物理上分布在计算机网络的各个不同节点上。

分布式数据库管理系统（DDBMS）：分布式数据库系统由分布式数据库（DDB）和分布式数据库管理系统（DDBMS）组成。分布式数据库管理系统是分布式数据库系统的一组软件。负责分布式数据库的数据管理和操作。由于分布式数据库管理系统基于分布式环境实现，必须保证数据的一致性、完整性等，因此，分布式数据库管理系统的实现复杂度远高于集中数据库管理系统。

分布式数据库的两大特性：数据透明性、场地自治性。

数据透明性：基于分布库的划分层次：全局外层（用户层）、全局概念层、局部概念层和局部内层。应用程序与系统实际数据组织相分离，即数据具有独立性或透明性。具体体现为分布透明性、复制透明性和分片透明性。

分布透明性：全局用户看到的是全局数据模型的描述，用户像使用集中数据库一样，不需考虑数据的存储场地和操作的执行场地。

复制透明性：分布式数据库支持有控制的数据冗余，即数据可重复存储在不同的场地上，提高系统的可用性和可靠性，加强系统处理的并行性。

分片透明性：分片是将一个关系分成几个子关系之中的一个子关系。根据实际需求，一个分片可能存储在不同的场地上（实际存储在场地上的分片称分段）

逻辑层：用户语义层，物理层是具体实现细节。逻辑层的语义与物理层的实现分离开来，对高层系统和用户隐蔽了实现细节。即应用程序无须考虑数据分片细节，对分片的调用（分片到分段的映射）由系统自动完成。

场地自治性：在分布式数据库系统中，多个场地或结点的局部数据库在逻辑上集成为一个整体，并为分布式数据库系统的所有用户使用，这种应用称全局应用，其用户称全局用户。分布式数据库系统也允许用户只使用本地的局部数据库，这种应用称为局部应用，其用户为局部用户。这种局部用户独立于全局用户的特性称局部数据库的自治性，也称场地自治性。具体体现为：设计自治性、通信自治性、执行自治性。

设计自治性：局部数据库管理系统（DBMS）能独立决定它自己局部库的设计。

通信自治性：局部数据库管理系统（DBMS）能独立决定是否和如何与其他场地的DBMS通信。

执行自治性：局部数据库管理系统（DBMS）能独立决定以何种方式执行局部操作。

分布式的多处理器系统：集中式管理系统存在于一个场地上；分布式管理系统分布在网络上的多个场地上。并行系统根据三要素P（处理机）、M（内存）、D（磁盘）组成方式不同，有三种体系结构。

无共享系统（Shared-NothingArchitecture）：最基本不同是操作模式。多处理器系统是对称的，由一定数量的处理器、内存组件组成，由一个或多个同样的操作系统严格控制每一个处理器的任务。而分布计算系统是异构的操作系统以及硬件环境。

分布式数据库系统既不是多处理机系统也不是基于服务器的系统。

### 分布库系统的作用和特点

分布式数据库系统的作用：分布式数据库系统是地理上（或物理上）分散而逻辑上集中的数据库系统。适合于分散型组织结构的任何信息系统。如：航空公司订票系统、陆、海、空军事指挥系统、CIMS制造业、银行通存通兑系统和连锁配送系统等等。以上每一系统都涉及分散在不同地理位置上数据的一致性、完整性及有效性，是集中式数据库无法满足的。在此推动下，研制出分布式数据库系统，有效地适应了地理上分散的、网络环境下的、逻辑上统一的分布式系统的需求。

分布式数据库系统的特点：分布式数据库系统是在集中式数据库和计算机网络技术基础上发展起来的，同时提出了许多新观点、新方法和实现的新技术，有效地提高了分布式系统的性能。因此，分布式数据库系统具有许多集中库系统所不具备的优点。但由于分布库系统复杂，有些技术的实现还不完善，如：恢复开销庞大，导致系统效率严重下降；难于达到完全透明等。

分布式数据库系统的优点：提高系统性能，如：吞吐率和响应速度提高。可利用现有设备和系统，降低投资。系统允许存储副本，提高可靠性、可用性和并行执行度。根据实际需要，可增加或减少某一场地。系统具有可扩展性。系统资源和数据分布在物理上不同的场地上，为系统所有用户共享。

分布式数据库系统的缺点：系统不很完善。系统复杂。维护开销大。需分布控制。安全问题。系统难以改变。

分布库系统复杂因素：数据复制、场地故障、分布式事务处理。

### 关键技术

分布式数据库设计：分布式数据库设计中需要考虑下面几个问题：如何合理分布数据于各个场地上；如何设定复制型和非复制型数据；如何实现目录管理。

查询处理：事务到数据操作命令的转换问题；以最小代价（访问时间最少）执行查询的优化问题。

并发控制：对多个用户并发访问的同步问题；事务的一致性和隔离性问题；解决和预防死锁问题。

可靠性：系统故障恢复问题；事务的原子性和持久性实现问题。

安全性与完整性：用户授权和认证问题；访问权限控制问题；数据完整性约束问题。

VDN系统：德国柏林大学1977-1979开发的关系数据模型，同构型的DDBS。支持关系的水平分片和副本，采用封锁和预防死锁。

POREL系统：德国斯图加特大学1975-1981开发的关系数据模型，同构型的DDBS。支持关系的水平分片和副本，采用封锁和预防死锁，共用系统的并发控制、恢复及存取机制。

大数据管理新模式：支持海量数据管理的系统应具有如下特性

* 高可扩展性（满足数据量增长的需要）
* 高性能（满足数据读写的实时性和查询处理的高性能）
* 容错性（保证分布系统的可用性）
* 可伸缩性（按需分配资源）
* 经济性（尽量低的运营成本等）

分布式大数据库系统及关键技术：面向大数据管理的分布式数据库管理系统，称为分布式大数据库管理系统，简称为大数据库管理系统。

* 大数据库系统结构
* 大数据存储与管理
* 大数据查询处理
* 大数据事务管理
* 负载均衡策略
* 大数据副本管理
* 大数据安全管理
* 大数据管理支持理论

## 分布式数据库系统体系结构

### 系统结构

系统的体系结构（Architecture）：指按组件、组件功能以及它们的交互作用定义系统的结构。

参考体系结构的目的:讨论的框架、标准。

DBMS标准化：基于功能、基于组件、基于数据、最早的是基于数据组织ANSI/SPARCArchitecture（externalview,conceptualview,internalview）的三级模式结构。

### 不同系统的系统结构图

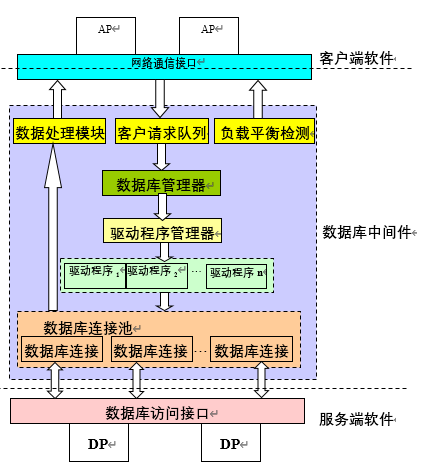
负载平衡监测:负责监控数据库服务器的状态及性能，调度依据

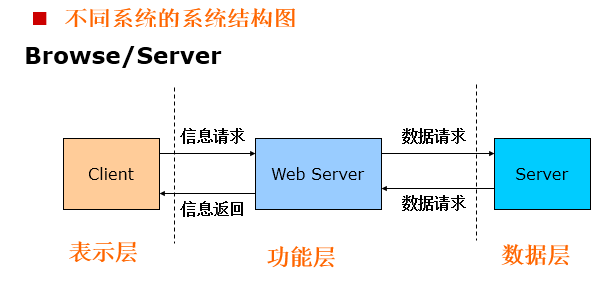
数据处理:负责处理从数据库返回的数据，按一定的方式将数据传送给AP。

数据库管理器:负责接收客户请求队列中的客户请求，调用相应的驱动程序管理器，完成相应的数据库查询任务。

驱动程序管理器:负责调度相应的数据库驱动程序，实现与相应的数据库的连接。

基于“中间件”的客户/服务器的体系结构





### 组件结构

#### 2.3.1应用处理器（AP）功能

用户接口：检查用户身份，接受用户命令，如：SQL命令。

语义数据控制器：视图管理、安全控制、语义完整性控制（全局概念模式）。这些约束定义在字典中。

全局查询处理器:将用户命令翻译成数据库命令；生成全局查询的分布执行计划；收集局部执行结果并返回给用户。

全局执行监控器（全局事务管理器):调度协调和监视AP和DP之间的分布执行；保证复制数据的一致性；保证全局事务的原子性。

#### 2.3.2数据处理器DP功能

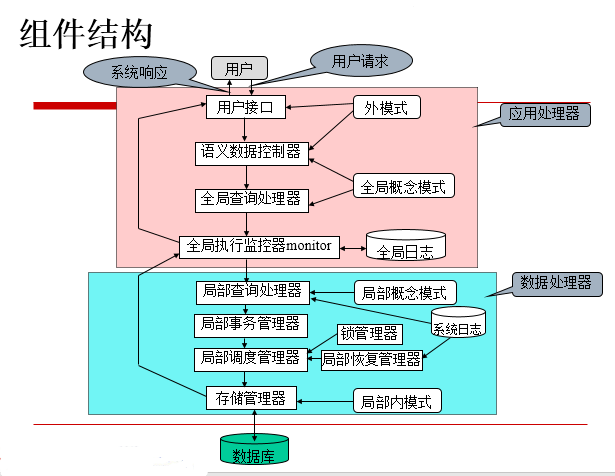
局部查询处理：实现全局命令到局部命令的转换；访问路径选择器，选择最好的路径执行。

局部恢复管理器：维护本地数据库一致性的故障恢复。

存储管理器：按调度命令访问数据库；控制数据库缓存管理器；返回局部执行结果；保证子事务执行的正确性。

局部事务管理器：以局部子事务为单位调度执行，保证子事务执行的正确性。

局部调度管理器：负责局部场地上的并发控制，按可串行化调度和执行数据操作。



### 模式结构

全局模式或外模式（ES）:全局外模式即全局用户视图，是分布式数据库的全局用户对分布式数据库的最高层抽象。全局用户使用视图时，不必关心数据的分片和具体的物理分配细节。

全局概念模式（GCS）:全局概念模式即全局概念视图，是分布式数据库的整体抽象，包含了全部数据特性和逻辑结构。像集中式数据库中的概念模式一样，是对数据库全体的描述。全局概念模式再经过分片模式和分配模式映射到局部模式。

分片模式是描述全局数据的逻辑划分视图。即全局数据逻辑结构根据某种条件的划分为局部数据逻辑结构。每一个逻辑划分形成一个分片。在关系数据库中，一个关系中的一个子关系称该关系的一个片段。

分配模式是描述局部数据逻辑的局部物理结构，即划分后的分片的物理分配视图。

局部概念视图（LCS）：局部概念模式为局部概念视图，是全局概念模式的子集，用于描述局部场地上的局部数据特征和逻辑结构。当全局数据模型与局部数据模型不同时，还涉及数据模型转换等内容。

局部内模式定义局部物理视图，是对物理数据库的描述，类似集中数据库的内层。

分布式数据库的四层结构及模式定义描述了分布式数据库是一组用网络联结的局部数据库的逻辑集合。它将数据库分为全局数据库和局部数据库。全局数据库到局部数据库由1:N的映射模式描述。全局数据库是虚拟的，由全局概念层描述。局部数据库是全局数据库的内层，由局部概念层和局部内层描述。全局用户只关心全局外层定义的数据库用户视图，其内部数据模型的转换、场地分配等由系统自动实现。

### 分布式数据库系统分类

分布式数据库系统的分类方法很多。为全面、系统地对分布式数据库系统进行分类，采用分布式数据库的三个特性（分布性、异构性、自治性）组成的三维空间图来描述分布式数据库的类型。三个基本特性。

分布性是指系统的各组成单元是否位于同一场地上。分布式数据库系统是物理上分散、逻辑上统一的系统，即具有分布性。而集中式数据库系统集中在一个场地上，所以不具有分布性。

异构性是指系统的各组成单元是否相同，不同为异构，相同为同构。异构主要:数据异构性：指数据在格式上、语法和语义上存在不同。数据系统异构性：指各个场地上的局部数据库系统是否相同。如：均采用ORACLE数据库系统的同构数据系统，或某些场地采用SYBASE数据库系统，某些场地采用INFORMIX数据库系统的异构系统。平台异构性：指计算机系统是否相同。如：均为微机系统组成的平台同构系统或由VAX或ALPHA系统等异构平台组成的系统。

自治性是指每个场地的独立自主能力。自治性通常由设计自治性、通讯自治性和执行自治性三方面来描述。根据系统的自治性，可分为集中式系统、联邦式系统（FederatedDB）和多数据库系统（MultiDB）。集中式系统即为传统的数据库系统。联邦式系统：实现需要交互的所有数据库对之间的一对一连接。多数据库系统：是指若干相关数据库的集合。各个数据库可以存在于同一场地，也可分布在多个场地。对多数据库系统进行管理的软件称多数据库管理系统，多数据库管理系统是对一组自治的数据库进行管理，并提供透明访问。

## 分布式数据库的设计

### 两种设计策略构

Top\_Down和Bottom-UP应用场合:Top\_Down:适于新设计一个数据库系统。（designedfromscratch）Bottom-UP:已存在许多数据库系统，并将它们集成为一个数据库。典型的数据集成研究。

### 分片的定义

片段（Fragment）：是分布式数据库中数据的存储单位。

分片（Fragmentation)：对全局数据的划分。

片段:划分的结果称为片段。

分配（Allocation）：对片段存储场地的指定，称为分配。当片段存储在一个以上场地时，称为数据复制（Replication）存储。如果每个片段只存储在一个场地，称为数据分割（Partition）存储。

分片的作用：减少网络传输量。如：采用数据复制，可就近访问所需信息。需频繁访问的数据分片存储在本地场地上。增大事务处理的局部性。局部场地所需的数据分片分配在各自的场地上，减少数据访问的时间，增强局部事务效率。提高数据的可用性和查询效率。就近访问数据分片或副本，可提高数据的访问效率。同时当某一场地发生故障时，若存在副本，非故障场地上的数据副本均是可用的。保证了数据的可用性、数据的完整性和系统的可靠性。使负载均衡。减少数据访问瓶颈，提高整个系统效率。

分片设计过程:分片过程是将全局数据进行逻辑划分和实际物理分配的过程。全局数据由分片模式定义分成各个片段数据，各个片段数据由分配模式定义存储在各场地上。

分片的原则:在构成分布式数据库系统时，设计者必须考虑数据如何分布在各个场地上，即全局数据如何进行逻辑划分和物理分配问题。哪些数据须分布存放、哪些数据不需要分布存放、哪些数据需要复制等，对系统进行全盘考虑，使系统性能最优。但无论如何进行分片，必须遵循下面原则：完备性：所有全局数据必须映射到某个片段上。可重构性：所有片段必须可以重新构成全局数据。不相交性：划分的各片段所包含的数据不允许相同。

分片的种类:分布式系统按系统实际需求对全局数据进行分片和物理分配。分片的种类有三种。水平分片：按元组进行划分，由分片条件决定。垂直分片：按关系属性划分，除关键字外，其他属性不允许出现在多个分片中。混合分片：即包括水平分片也包括垂直分片。

分布透明性:透明性：指对用户和高层系统隐蔽具体实现细节。包括：分片透明性、分配透明性和局部映射透明性。分片透明性：用户不必考虑数据属于哪个片段。分配透明性：用户不必考虑各个片段的存储场地。局部映射透明性：用户不必考虑数据的局部存储形式。

水平分片的定义:水平分片是将关系的元组集划分成若干不相交的子集。每个水平片段由关系中的某个属性上的条件来定义，该属性称为分片属性，该条件称为分片条件。

基本水平分片(Primaryhorizontalfragmentation):定义：基本水平分片针对一个关系，基于选择操作定义的。基本水平分片步骤：

* 第一步：找到具有完备性和最小性的简单谓词集合
* 第二步：派生小项谓词集合
* 第三步：确定有意义的小项谓词集，去除无意义的小项谓词。

导出水平分片定义:一个关系的分片不是基于关系本身的属性，而是根据另一个与其有关联的关系的属性来划分，这种划分为导出水平划分。定义：如果一个关系的水平分片的分片属性属于另一个关系，则该分片称为另一个关系的导出分片。

垂直分片是将一个关系按属性集合分成不相交的子集（主关键字除外），属性集合称为分片属性。即垂直分片是将关系按列纵向以属性组划分成若干片段。

### 分片的表示方法

采用直观的图形表示法外，还提供了分片树表示法。

图形表示法:用图形直观描述。说明：

* 整体全部表示为全局关系；
* 部分表示分段；
* 按水平划分表示水平分段；
* 按垂直划分表示垂直分段；
* 混合划分既有水平划分，又有垂直划分。

分片树表示方法:一个分片映射可用分片树表示。先了解分片树的构成。定义：一个分片树由以下几部分组成：

* 根节点表示全局关系；
* 叶子节点表示最后得到的片段；
* 中间节点表示分片过程的中间结果；
* 边表示分片，并用h（水平）和v（垂直）表示分片类型
* 节点名表示全局关系名和片段名

### 分配设计

全局数据经过分片设计，得到各个划分的片段，片段到物理场地的存储映射过程称为分配设计。

分配类型:

非复制分配:如果每个片段只存储在一个场地上，称为分割式分布，对应的分布库，称为全分割式分布库。

复制分配:如果每个片段在每个场地上存有副本，称为全复制分配，对应的分布库称为全复制分布库。

如果每个片段只在部分场地上存有副本，称为部分复制分配，对应的分布库称为部分复制分布库。

分配类型:系统是采用全分割数据库或全复制数据库或部分复制数据库，需根据应用需求及系统运行效率等因素来综合考虑。一般，从应用角度出发，须考虑：

* 增加事务处理的局部性；
* 提高系统的可靠性和可用性；
* 增加系统的并行性。

从系统角度出发，须考虑：

* 降低系统的运行和维护开销；
* 使系统负载均衡；
* 方便一致性维护。

从上面几点考虑因素，可知：采用数据复制式分配可增加事务处理的局部性、提高系统的可靠性和可用性，但增加了系统运行的维护开销和系统一致性维护的复杂性；采用全数据分割式分配可增加系统的并行性并可使系统负载均衡，但降低了事务处理的局部性和系统的可靠性和可用性。

如何进行片段的分配需综合考虑应用和系统两方面的需求，以求得到一个最佳的分布式数据库系统。

一般，如果存在只读查询/更新查询>>1，则定义为复制式分配好些。

### 分配设计原则

分配设计时应考虑的具体因素。

数据库因素

* 片段的大小；
* 查询对片段的选择度（查询结果的大小）。

应用因素

* 查询对片段的读频度；
* 查询对片段的更新频度；
* 更新查询需访问的片段；
* 每个查询的启动场地。

场地因素

* 场地上存储数据的单位代价；
* 场地上处理数据的单位代价。

网络通讯因素

* 网络带宽及网络延迟；
* 场地间的通信代价；
* 传输的数据帧大小。

### 数据复制技术

#### 数据复制的优势

* 减少网络负载。就近访问，减少网络传输代价。
* 提高系统性能。利用本地资源，并行处理。
* 更好地均衡负载。负载分布到多个节点。

采用复制式存储技术也增加了数据一致性的维护代价。如同步数据时，如何有效地解决冲突等。

#### 数据复制的分类

根据更新传播方式不同，分为同步复制和异步复制:

同步复制方法是指所有场地上的副本总是具有一致性。

* 优点：同步复制方法的优势是实时保证了副本数据的一致性。
* 不足：1）需要场地间频繁通信并及时完成事务操作；2）导致冲突增加，增加了事务响应时间。
* 适用场合：对于一致性要求较高的商业应用中。
* 异步复制方法是指各场地上的副本不要求实时一致性，允许在一定时间内是不一致的。
* 优点：降低了通信量和冲突概率，缩短了事务响应时间，提高了系统效率。
* 不足：系统不能显示实时的结果，同时也存在潜在的数据冲突，增加了事务回滚的代价。
* 适用场合：侧重提高系统效率的应用，如大用户群实时访问大数据量的查询应用。

根据参与复制的节点间的关系不同，分为主从复制和对等复制。

主从复制也称单向复制。主场地上的副本称为主副本，从场地上的副本称为从副本。更新操作只能在主场地上进行，并同步到从场地的从副本上。

分析：主从复制实现简单，易于维护数据一致性。但由于数据只能在主场地上更新，降低了系统的自治性。

对等式复制也称双向复制。各个场地的地位是平等的，可修改任何副本。被修改的副本临时转换为主副本，其它为从副本。主副本所在场地为协调场地，协调同步所有从场地上的从副本。

分析：对等式复制中，各场地具有高度自治性，系统可用性好。但由于允许更新任何副本，会引起事务冲突，需要引入有效的冲突解决机制，处理复杂，系统开销大。

#### 复制的常用方法

* 基于触发器法：在主场地的主数据表（主副本）中创建相应的触发器，当主表数据进行更新、插入和删除操作并成功提交时，触发器就会被触发，将当前副本的变化反映到从副本中。可用于同步复制，但对于对等式复制和异构复制较难实现。占用的系统资源较多，影响系统运行效率，较适宜于小型数据库应用。
* 基于日志法：通过分析数据库的操作日志信息来捕获复制对象的变化。当主副本被更改时，复制代理只需将修改日志信息发送到从场地，由从场地代理实现本地数据的同步。实现方便，不占用太多额外的系统资源，对任何类型的复制都适用。
* 基于时间戳法：主要根据数据的更新时间来判断是否是最新数据。需要为每一个副本数据表定义一个时间戳字段，用于记录每个表的修改时间，并需要监控程序监控时间戳字段的时间。该方法适合于对数据更改较少的系统。
* 基于API 法：在应用程序和数据库之间引入第三方的程序（如中间件），通过API完成。在应用程序对数据库修改的同时，记录复制对象的变化。该种方法可减轻DBA的负担，但无法捕获没有经过API的数据的更改操作，具有一定局限性。

### 分布式文件系统-HDFS

HDFS：分布式文件系统，适合存储超大文件（几百MB、GB甚至TB级别的文件）。将超大文件分割成多个块（block），block大小默认为64MB，每一个block会在多个数据节点（DataNode）上存储多份副本，默认是3份。HDFS采用一次写入、多次读取的流式数据访问模式。

数据在HDFS文件块中的存储模式:

NSM(N-ary Storage Model)存储模式：主要由页头（Page header）、数据 (Body)、以及每行数据（或称每条记录）在当前page中位置的偏移量 (Trailer)组成。记录数据从每一磁盘页的开始连续存放，在页的尾端存放记录的偏移地址（offset），用于定位每一记录的开始。

DMS (Decomposition Storage Model)存储模式：是将关系垂直划分为多个子关系，可细分为全DSM（full DSM）模式和部分DSM（partial DSM）模式

PAX (Partition Attributes Across)存储模式：针对每一页内的记录的进一步垂直分区的模式，即按同一属性值再分组为迷你页（minipage）

## 分布查询处理和优化

### 优化目标

优化就是寻找执行代价（费用和时间）最小的查询执行策略，使系统执行效率降到最低。优化的目标就是指局部执行代价和网络传输代价的和最小。

* 局部执行代价：主要指输入/输出次数（I/O代价）及CPU处理代价。
* 网络传输代价：主要指传输启动代价和数据传输代价。

### 优化内容

优化内容体现如下几点：

* 执行运算的次序。
* 执行每种运算的方法。如上例，不同方法代价不同。
* 所访问的副本场地。如：选择就近的场地，节约传输代价。
* 执行运算的场地的选择，使总的传输代价或总代价最低。
* 影响查询处理效率的因素有：网络传输代价（数据量和延迟等）、局部I/O代价及CPU处理代价等，但主要由网络通信代价和局部I/O代价来衡量。不同的分布式数据库系统可能对评估查询处理的传输代价和I/O代价的侧重不同。
* 为提高查询的效率，在查询处理过程中还要进行优化处理。
* 查询优化就是确定出一种执行代价最小的查询执行策略或寻找相对较优的操作执行步骤。一般采用多级优化的策略。本章介绍全局查询的处理与优化。

### 查询处理概述

优化类型：穷举法（Exhaustive search）– workable for small solution space.启发式（Heuristics） – σ，π perform first, semijoin, etc. for large solution space.

优化时机：静态的(Static) – 在编译时进行优化，基于统计信息进行穷举优化，优化一次，执行多次

动态的(Dynamic) – 在执行中优化，准确性好，但每次执行都进行优化，代价高

统计数据（Statistics）：元组基数, 属性值分布，关系大小等。信息提供给查询优化器并定期修改。

决策场地（Decision Site）查询优化场地可以：

* 单场地：集中的方法，简单，需要整个分布库的知识。
* 分布式：涉及所有的场地协调确定调度，只需本地知识，需要协调代价。
* 混合型： 一个场地确定全局调度，其他场地优化局部子查询。

复制的片段：最小化通信代价。使用半连接。

### 查询处理层次

查询分解（Query Decomposition） 基于全局概念模式将演算查询分解为代数查询。

* Step1 – 演算规范化
* Step2 – 语义分析，去掉不正确的查询。
* Step3 – 简化，去除冗余的部分
* Step4 –将演算查询转化为优化的代数查询。

数据本地化（Data Localization）：分布查询映射为片段查询，简化、重组为优化的查询。

全局查询优化（Global Query Optimization）：找接近于最优的执行策略；找片段查询中最佳的操作顺序，包括通信操作；需要实时定义代价函数。

局部查询优化（Local Query Optimization）：集中的系统算法、INGRES –动态优化（ dynamic optimization）、System R – 基于穷举法的静态优化（static optimization based on exhaustive search）

### 查询分解（QUERY DECOMPOSITION）

* 查询分解（QUERY DECOMPOSITION） 输入 : 基于全局关系的演算查询
* 规范化（Normalization）将查询转换为规范化形式（AND形式或OR形式 ）
* 分析（ Analysis）检查不正确的查询得到关系演算子集
* 约简（Simplification）删除冗余谓词
* 查询重写（Restructuring）

将演算查询转换为关系代数查询

可能得到多种转换结果

使用转换规则

* 分析

目的: 拒绝类型不正确或语义不正确的查询

查询重写：直接将关系演算转换为关系代数；重写关系代数查询以提高性能。通常使用操作树来表示关系代数查询。关系代数树定义为：

* 根节点表示查询结果
* 叶子节点表示关系；
* 非叶子节点表示中间结果关系
* 叶子到根的边表示序列操作

### 数据本地化

查询的处理过程是从全局关系到片段关系，最后再到实际操作的副本关系。

数据本地化将介绍全局查询到片段查询的变换。即利用全局关系与其片段关系的等价变换，将分布查询中的全局关系替换为对片段关系的查询，变换后的查询称为片段查询。对应于片段查询的查询树，称为片段查询树。

片段查询优化规则

准则1：对于一元运算，根据一元运算的重复律，将叶子节点之前的选择运算作用于片段，如果不满足片段的限定条件，则置为空关系。

准则2：对于联接运算的树，若联接条件不满足，则将其置为空关系。

准则3：在查询树中，将联接运算（∞）下移到并运算（∪）之前执行。

准则4：消去不影响查询运算的垂直片段。

## 分布查询的存取优化

### 基本概念

分布执行过程实际上就是从查询场地发出查询命令、从数据源获取数据、确定最佳的执行场地和返回执行结果的过程。

* 查询场地：指发出查询命令和存储最终查询结果的场地。查询场地也称最终结果文件。
* 源数据场地：指查询命令需要访问的数据副本所在的场地，可能涉及到一个或一个以上的场地。源数据场地也称源数据文件。
* 执行场地：指查询操作执行所在的场地。执行场地可以和查询场地或源数据场地处于同一场地，也可不处于同一场地。执行场地也称中间结果文件。

分布查询的存取优化的目标

* 远程网：主要考虑通信开销，使通信代价最小。
* 局域网：需同时考虑通信代价和本地处理代价， 使综合代价最小。

优化的内容：优化是在片段查询的基础上进行的实际物理副本查询操作的优化。具体如下：

输入：片段查询表达式。输出：分布执行计划

查询存取优化内容：

* 确定片段查询需访问的物理副本。通常：①本场地上的物理副本优先；②若二元运算存在尽量选择本场地上的二元运算；③数据最小的物理关系应被优先选中；④网络通信代价小的应优先选中。
* 确定片段查询表达式操作执行的最优顺序。包括从叶到根的执行和同一层叶子上表达式执行的先后，特别是对查询树上的并操作和联接操作的执行次序的确定，其代价差别很大。
* 选择执行每个操作的方法。如：尽量将同一场地上的、同一物理副本的全部操作组合在一起统一考虑完成。

### 半联接优化方法

对联接操作的优化有两种趋势，一种为采用半联接技术，减少联接操作的操作数，以降低传输费用；另一种为采用全联接技术，主要考虑局部代价。一个系统需根据其目标综合确定其优化算法。

半联接的作用：采用半联接技术的优化目标是减少联接操作的操作数，以降低传输费用。

### SDD-1系统优化技术

优化模型：在SDD-1算法中，分别使用连接图（join graph）和概要图来描述查询中的条件限制和在关系上的特征参数。

关系的概要图:主要用于表示一个关系上的特征参数，其中数据包括关系中元组数量Card(R)、每个关系属性的长度Length(A)和属性不同值的数量Val(R,A)。

SDD-1算法的基本优化思想：

* 联接图中的每个关系都用一元运算进行局部缩减；
* 对受益半联接集合中的所有半联接进行操作，逐个找出最优的操作；
* 选择一个要求最少传输代价的场地，执行操作。

受益半连接集：对于一个给定的半连接集合，所有利益超过代价的半联接操作的集合，称为受益半联接集，记为P。

半联接技术的不足:半联接技术是通过局部缩减操作缩减关系的数据量，发送缩减的关系到执行场地，在执行场地对缩减后的关系进行查询处理。

优点是大大降低了场地间传递的信息量，减少了整个系统的传输代价。不足是增加了系统的局部处理代价：

（1） 没有考虑局部代价；如 ：R∞S= (R∝S)∞S= R∞∏B(S)

① ∏B(S)的代价 ② R∞∏B(S)的代价

（2） 当选择度较低时，半联接技术才可行。 因此，在应用半联接技术时，要考虑其适应的环境。

### 枚举法优化技术

查询操作的代价评估：综合考虑局部代价和传输代价。

* 若侧重传输代价，局部代价可以忽略不计时，采用半联接技术较好；
* 侧重局部代价时，采用直接联接比采用半联接优越。

枚举法是基于直接联接的实现方法。

常见的直接连接算法主要有：

* 嵌套循环连接算法(nest-loop)
* 归并排序连接算法（merge-scan）
* 哈希连接算法（Hash）
* 基于索引的连接算法

对每种直接连接算法进行代价估计：考虑执行连接操作所需的磁盘I/O代价。

对磁盘I/O代价的估计主要依赖于数据库的特征参数。

嵌套循环连接算法

* 是一种最简单的连接算法，原理是对连接操作的两个关系中的一个仅读取其元组一次，而对另一个关系中的元组将多次读取。
* 特点是可以用于任何大小的关系间的连接操作，不必受连接操作所分配的内存空间大小的限制。
* 基于元组的嵌套循环连接
* 在执行嵌套循环连接时，仅对外关系进行1次读取操作，而对内关系则需要进行多次读取。
* 如果不进行优化的话，执行代价很大，以磁盘IO代价计算最多可能多达Card(S)\*Card(R) =mn次。
* 通常对这种算法进行修改，一种方法是通过尽可能多地使用内存以减少磁盘IO次数；另一种方法是使用连接属性上的索引，以减小参与连接元组的数量。
* 基于块的嵌套循环连接
* 通过尽可能多的使用内存，减少读取元组所需的I/O次数。
* 对连接操作的两个关系的访问均按块（也称为页面）进行，同时使用尽可能多的内存来存储嵌套循环中外关系的块。

### 分布式查询优化问题

代价模型

* 多查询优化
* 启发式规则裁剪

大的查询集合

* 优化 select-project-join查询
* 也需要处理复杂查询(unions、disjunctions、aggregations、sorting)平衡优化代价与执行代价
* 启发式规则裁剪heuristics to cut down on alternatives
* 控制搜索策略

优化与重优化间隔

* 数据库概要信息变化时需要重优化

分布式系统中的查询优化算法

* Distributed INGRES方法
* System R\*方法

Distributed INGRES方法：当一个查询Q被分解为一组不可规约的子查询q1,q2,…,qn后，

* 集中式环境中可以简单地顺序执行每个子查询。
* 在分布式环境中，由于关系以分片的形式存储在多个站点上，因此子查询可能还需要继续细分以便能够在多个场地上执行，而这将产生在站点之间复制关系所需的通信代价。
* 一种简单的优化方式是优先处理无前继查询且所涉及的关系较小的子查询。一方面可以获得较小的中间结果，另一方面可以减少复制关系所需的传输代价。

并行查询处理与分布式查询处理

* 分布式数据库而言，通常数据在物理上是异地分布，在查询优化算法中更加注重通信代价的优化，尽量将传输的数据量降到最低，如采用半连接的优化算法
* 并行查询处理，通常处理节点采用高速网络连接，同时考虑通信代价和本地处理代价，通常以分布式查询算法为基础，但更注重操作符内并行性和操作符间的并行性，通过使用多处理节点提高查询处理的效率，降低响应时间
* 大数据库系统通常同时采用了并行数据库和分布式数据库的技术，在数据存储和事务管理方面主要采用分布式数据库技术，在海量数据分析所需的查询处理上则更多的使用并行数据库的查询处理技术。

基于分析引擎的大数据库查询优化

* 面向事务型应用的查询处理中，通常是针对某一指定键值（Key）的数据进行简单查询操作；
* 面向分析型应用的查询处理中，查询具有更高的复杂性，查询中不但包含各类聚合函数，还可能包含复杂连接操作；

借助分布式或并行的分析引擎来执行分析型复杂操作。

## 分布式事务管理

### 事务的基本概念

#### 事务的定义

* 任何数据库应用最终都将转换为一系列对数据库进行存取的操作系列；
* 为了保证数据库的正确性及操作的有效性，将数据库应用中全部或部分操作序列的执行定义为事务；
* 一个事务所包含的所有操作，要么全做，要么全不做，是一个不可分割的整体；

事务是由若干个为完成某一任务而逻辑相关的操作组成的操作序列，是保证数据库正确性的基本逻辑单元。

* 显式声明：应用程序通过事务命令显式地划分事务；
* 隐式声明：系统按照默认规定自动地划分事务；

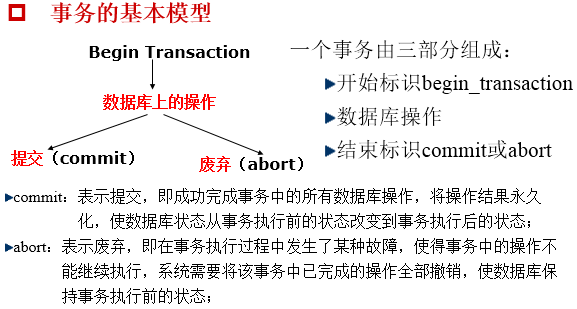
#### 事务的基本性质

原子性(atomicity):事务所包含的操作要么全做，要么全不做。由于输入错误、系统过载、死锁等导致的事务废弃而需要进行的事务的原子性维护，称为事务恢复。由于系统崩溃（死机、掉电）而导致的事务废弃或者提交结果的丢失而需要进行的原子性维护处理，称为故障恢复。对于废弃的事务，需要撤销事务已经完成的部分操作，使得数据库恢复到事务执行前的状态，称为反做(undo)；对于已经提交的事务，由于不能保证事务所作的修改已经被更新到外存数据库，因此需要重新执行事务的全部操作，称为重做(redo);

一致性(consistency):假如数据库的状态满足所有的完整性约束，则称该数据库是一致的。事务的一致性，是事务执行的结果必须是使数据库从一个一致性状态变化到另一个一致性状态，而不会停留在某种不一致的中间状态上。

隔离性(isolation):当多个事务的操作交叉执行时，若不加控制，一个事务的操作及所使用的数据可能会对其它事务造成影响。事务的隔离性是指：一个事务的执行既不能被其它事务所干扰，同时也不能干扰其它事务。具体来讲，一个没有结束的事务在提交之前不允许将其结果暴露给其它事务。因为未提交的事务有可能在以后的执行中废弃。若存在其它事务使用了这种无效的数据，则这些事务同样也要进行废弃。这种因一个事务的废弃而导致其它事务被牵连地进行废弃的情况称为“级联废弃”。

耐久性(durability):事务的耐久性体现在：当一个事务提交后，系统保证该事务的结果不会因以后的故障而丢失。也就是说，事务一旦被提交，它对数据库的更改将是永久性的。即使发生了故障，系统应具备有效的恢复能力，将已提交事务的操作结果恢复过来，即重做（REDO）处理，使这些事务的执行结果不受任何影响。



#### 事务的种类

按照组成结构的不同，可以将事务划分为两类：平面事务（flat）和嵌套事务(nest)。

平面事务是指每个事务都与系 统中其它事务相分离，并独立于其它事务。平面事务是用begin和end括起来的自治执行方式。

嵌套事务是指一个事务的执行包括另一个事务。其中，内部事务称为外部事务的子事务，外部事务称为子事务的父事务。

嵌套事务具有：提交依赖性（commit\_dependency）:子事务提交，必须等待父事务提交。

废弃依赖性(abort\_dependency) :父事务废弃，则子事务必须废弃。

#### 分布式事务的定义

从宏观上来看，分布式事务是由一系列分布在多个场地上执行的数据库操作所组成的。

分布式事务：是指分布式数据库应用中的事务，也称为全局事务。

子事务：一个分布式事务在执行时将被分解为若干个场地上独立执行的操作序列，即一个分布式事务在某个场地上操作的集合。显然，分布式事务是典型的嵌套事务。分布式事务同样具有ACID四个特性。由于子事务的分布式执行，因此保证分布式数据库系统的ACID特性更加复杂。

例如，为了保证全局事务的原子性，必须保证组成该事务的所有子事务要么全部提交，要么全部撤销，不允许出现部分场地上的子事务提交，部分场地上的子事务撤销的情况发生。因此，在分布式事务执行过程中，需要对子事务进行协调，从而决定事务的提交与撤销。

另外，在分布式事务中，除了要考虑对数据的存取操作序列之外，还需要涉及大量的通信原语和控制报文。通信原语负责在进程间进行数据传送，控制报文负责协调各子事务的操作。

### 分布式事务的实现模型

一个分布式事务所要完成的任务是由分布于各个场地上的子事务相互协调合作完成的。有两种实现模型：进程模型、服务器模型。

* 进程模型：全局事务为每一个子事务在相应的场地上创建一个代理者进程（也称局部进程或子进程），由代理者进程执行该场地上的有关操作。同时，为协调各子事务的操作，全局事务还要启动一个协调者进程，来进行代理者进程间的通讯，控制和协调各代理者进程的操作。发出分布式事务的场地（根代理所在的场地）称为该事务的源场地。
* 服务器模型：服务器模型要求在事务的每个执行场地上创建一个服务器进程，用于执行发生在该场地上的所有子事务。每个服务器进程可以交替地为多个事务的子事务服务。也就是说，不同全局事务中的子事务在同一个场地上共用一个服务器进程。

#### 分布式事务管理的目标

分布式事务管理的目标是使事务的执行具有较高的执行效率，具有较高的可靠性和并发性。与事务执行效率有关的因素还包括： CPU和内存利用率、控制报文开销、分布式执行计划、系统可用性。为此，提出分布式事务管理的目标是：

* 维护分布式事务的ACID性质；
* 提高系统的性能，包括CPU、内存等资源的使用效率和数据资源的使用效率，尽量减少控制报文的长度及传送次数，加速事务的响应速度；
* 提高系统可靠性和可用性，当系统的一部分或者局部发生故障时，系统仍能正常运转，而不是整个系统瘫痪。

#### 分布式事务的提交协议

分布式数据库中的全局事务由被分解为在各个场地上执行的子事务所组成。只有当各个场地上的子事务都正确执行后，全局事务才可以提交。只要有一个子事务不能提交，则全局事务应该废弃，接下来所有的子事务也应全部废弃。因此，所有子事务均可以正确提交是分布式事务提交的前提。上述过程，通过两阶段提交协议来实现，简称2PC（2-Phase-Commit）协议。

协调者和参与者

* 协调者：是协调者进程的执行方，即根代理，负责决定所有子事务的提交或废弃；
* 参与者：是代理者进程的执行方，即子代理，负责执行本地数据库操作，负责本地子事务的提交或废弃，并向协调者报告本地子事务的执行状态；

协调者和参与者都有本地日志，他们在进行操作前都必须将该操作记录到相应的日志文件中，以进行事务故障恢复和系统故障恢复。

两阶段提交协议的基本思想

* 决定阶段：有协调者发送预提交(prepare)命令，然后等待参与者的应答。如果所有的参与者都返回“准备提交(ready)”。那么协调者做出提交决定；如果至少有一个参与者返回“准备废弃”，那么协调者做出废弃决定。
* 执行阶段：协调者把在决定阶段做出的决定发送给参与者。如果协调者向各个参与者发“提交”（Commit）命令，各个参与者执行提交；如果协调者向各个参与者发出“废弃”（Abort）命令，各个参与者执行废弃，取消对数据库的修改。无论是“提交”还是“废弃”，各参与者执行完毕后都要向协调者返回“确认”（Ack）应答，通知协调者执行结束。

分布式事务管理在功能上分为两个层次：

局部事务管理器（LTM）:负责本场地的子事务执行、故障恢复和并发控制；

分布式事务管理器（DTM）:协同全局事务的执行、物理上多个、逻辑上是一个整体；

### 分布式事务执行的控制模型

LTM和DTM的协同工作。三种控制模型：主从控制模型、三角控制模型、层次控制模型

主从控制模型：DTM作为主控制器，LTM作为从属控制器。一问一答方式： DTM向LTM发送命令并收集应答来实施状态监控。 LTM根据DTM的命令来执行本场地的子事务，并向将DTM返回最终结果。LTM之间无直接通信（由DTM转发 ）。

三角控制模型：三角控制模型与主从控制模型相类似，差别是：LTM之间可以直接通信，不需要DTM作为中介转发。因此，同主从控制模型相比，三角控制模型在一定程度上减少了通信代价，但使得DTM和LTM之间的协作控制变得更加复杂。

层次控制模型：每个LTM可以具有双重角色，除了用来管理其本地的子事务外，还可以将其负责的子事务进一步分解，衍生出一系列更下层的LTM ，每个分解部分由下一层的LTM负责执行。这时， LTM自身将成为一个新的DTM，并控制更下层的LTM的执行。层次控制模型扩展性好，其层数可以随着任务量的增大而增加，但实现上更加复杂。

实现模型的三层：

代理层：由1个根代理和若干个子代理组成，通过控制报文来协同工作；根代理包含一个GTM（全局事务管理器），负责全局事务的划分和调度。

DTM层：由各场地上的DTM组成，负责管理分布式事务，调度子事务的执行；

LTM层：由所有场地上的LTM组成，负责管理DTM交付的子事务的执行。

### 两段提交协议(2PC)的实现方法

集中式2PC：首先确定一个协调者场地，通常由事务的发起者场地（源场地）充当，完成事务提交的初始化。“集中式”主要体现在协调者在整个分布式事务提交过程中始终作为唯一的总控节点（发布命令、搜集状态信息、做出决定）。通信只发生在协调者与参与者之间，参与者与参与者之间没有直接通信。

分布式2PC: “分布式”是指事务的所有参与者同时也都是协调者，都可以决定事务的提交和废弃，提交过程是完全分布地完成。由事务的始发场地完成提交初始化工作。分布式2PC最大的特点是事务的提交过程只需要一个阶段，即决定阶段。所有场地都可以互相通信，使得各个场地均可以获悉其它场地的当前状态（“准备提交”或“准备废弃”），独立地做出事务是否提交的决定。

分层式2PC: 分层式2PC也称树状实现方法，协调者与参与者间的通讯结构如同一棵树。其中，协调者为根节点，参与者构成树的中间节点或叶子节点。 分层式2PC的特点是协调者与参与者间的通信没有采用广播方式，而是借助于树型结构将报文逐层传递。集中式2PC可以看作是分层式2PC的特例（只有两层）。

线性2PC: 在线性2PC中，允许参与者之间相互通信，由事务的始发场地构造一个线性有序的场地链表。表中第一个场地（源场地）和最后一个场地共同担任协调者，在第一个场地后，依次为第一个参与者场地，第二个参与者场地，直到第n个参与者场地。 事务的源场地进入“预提交”状态，向第一个参与者场地发送“预提交”命令；若第一个参与者场地准备好提交，则向上一个场地发“准备提交”应答，并向下一场地发“预提交”命令；以此类推，直到最后一个场地。当最后一个参与者场地收到“预提交”命令，且已准备好提交时，最后一个参与者场地充当协调者，自己首先提交，并向前一场地发提交命令，依此类推直到始发场地，事务提交完成。若任何场地收到“预提交”命令时，没准备好提交，则向前一场地发“废弃”应答，并废弃事务，依次类推直到事务始发场地，事务废弃完成。 线性2PC的特点是决定阶段和执行阶段都是采用串行方式完成的。

两阶段提交协议存在的问题：如果在两段提交协议执行的过程中出现协调者故障或网络故障，那么参与者就不能及时收到协调者发出的“提交”命令，参与者处的子事务将处于等待状态。在故障恢复前，参与者子事务所占有的系统资源也不能被释放，参与者的子事务进入了阻塞状态。若参与者一直收不到协调者的命令，则子事务将始终处于阻塞状态而挂在相应的执行场地上，所占用的系统资源也不能被其它事务利用。

### 非阻塞分布式事务提交协议

三阶段提交协议的基本思想：全局事务的提交分为三个阶段：一：投票表决阶段、二：准备提交阶段、三：执行阶段。

阶段一：投票表决阶段

由协调者向各个参与者发“预提交”（Prepare）命令，然后等待回答。每个参与者根据自己的情况进行投票，若参与者可以提交，则向协调者返回“赞成提交”（Ready）应答，否则向协调者发送“准备废弃”（Abort）应答。

阶段二：准备提交阶段

若协调者收到的应答中存在“准备废弃”（Abort）应答，则向各个参与者发“全局废弃”（Abort）命令，各个参与者执行废弃，执行完毕后向协调者发送“废弃确认”（Ack）应答；若协调者收到的应答均为“赞成提交”（Ready）应答，则向各个参与者发“准备提交”（Prepare-to-Commit）命令，然后等待回答，若参与者已准备就绪，则向协调者返回“准备就绪”（Ready-to-Commit）应答。

阶段三：执行阶段

当协调者收到所有参与者的“准备就绪”（Ready-to-Commit）应答后，向所有参与者发送“提交”（Commit）命令，此时各个参与者已知道其它参与者均赞成提交，因此可以执行提交，提交后向协调者发送“提交确认”（Ack）应答。

### 大数据库的事务管理

大数据库系统是对关系型SQL数据系统的补充，是一种分布式、不保证遵循ACID特性的数据库设计模式。大数据库系统中的事务管理需要解决如下问题：高并发读写的需求、高可用性和高可扩展性的需求、高效率读写的需求。设计者尝试通过牺牲ACID和SQL等特性来提升对海量数据的存储管理能力。



## 分布式恢复管理

### 分布式恢复概述

数据库系统的可恢复性和高可靠性是保证各种应用正确而可靠地运行所不可缺少的重要组成部分。

可恢复性，提供恢复措施，把数据库系统从故障状态恢复到一个已知的正确状态；

高可靠性，尽量将崩溃后数据库的不可用的时间减少到最低，并保证事务的原子性和耐久性。

故障模型: 故障（Fault）：系统单元内部发生的错误或系统内部设计错误；错误（Error）：系统单元内出现了不正确的状态，是故障的内在表现形式；失效（Failure）：指系统的外部状态中所表现出来的错误；当系统单元组建不合理或系统内部设计存在不足时，将会引发系统故障，此时系统的内部状态处于错误的状态，进而使系统的外部环境受到影响，最终产生失效。

数据库系统中的故障通常分为4类：

（1）事务内部的故障

事务内部的故障可细分为可预期的和不可预期的。

可预期的事务故障是指故障的发生可以通过事务程序本身来检测。例如：在转帐事务中，事务主动检查贷方余额是否足够支持转帐，若发现不足，则废弃该子事务，数据库状态没有因为此事务发生变化；

事务内部的故障大多数都是不可预期的，发生这类故障的事务将不能正常运行到其终点位置（Commit或Abort）。因此，针对这类故障，数据库恢复机制要强行废弃该事务，使数据库回滚到事务执行前的状态。

（2）系统故障

表现：使系统停止运转，必须经过重启后系统才能恢复正常。例如，CPU故障、系统死循环、缓冲区溢出、系统断电等。

特点：仅使正在运行的事务受到影响，但数据库本身没有被破坏；内存中的数据全部丢失。一方面，一些尚未完成的事务的结果可能已被写入数据库中；另一方面，一些已提交的事务的结果可能还未更新到磁盘上。

结果：系统故障发生后数据库可能处于不一致的状态。

对于系统故障，数据库恢复机制要在系统重启后，将所有非正常终止的事务强行废弃，同时将已提交的事务的结果重新更新到数据库中，以保证数据库的正确性。

（3）存储介质故障

存储介质故障是指存储数据的磁盘等硬件设备发生的故障。例如，磁盘坏损、磁头碰撞、瞬时强磁场干扰等均为存储介质故障。

特点：不仅使正在运行的所有事务受到影响，而且数据库本身也被破坏。

结果：同前两种故障相比，存储介质故障是一种较严重的故障类型。

对于存储介质故障，数据库恢复机制要定期地对数据库进行转储，借助于备份数据库和日志文件来进行故障恢复。

（4）通信故障

前三个故障都是单场地故障、也是集中式数据库中的故障。对于分布式数据库而言，还要进行各个场地的通信，因此还会产生通信故障。

通信故障可细分为网络分割故障和报文丢失故障。

* 网络分割是指通讯网络中一部分场地和另一部分场地之间完全失去联系。
* 报文丢失是指报文本身错误或在传送过程中丢失而导致数据不正确。

对于通信故障，数据库恢复机制要针对故障的表现形式进行判断，分析故障产生原因并进行相应的处理，如重发报文等。

**恢复模型**:在故障恢复过程中，数据库恢复管理器依据数据库日志文件（log）对数据库事务进行恢复操作。

日志文件的格式和内容、反做(undo)和重做(redo)恢复策略、故障恢复模型 。

数据日志记录包括：

* 事务标识符（标明是哪个事务）
* 操作类型（插入、删除或修改）
* 操作的数据项
* 数据项的旧值（前像，BeforeImage）
* 数据项的新值（后像，AfterImage）

检查点记录：

* 检查点是在日志中周期设定的操作标志，目的是减少系统故障后的恢复的工作量。在检查点上，需要完成的操作包括：
* 首先，将日志缓冲区中的内容写入外存中的日志；
* 然后，在外存日志中登记一个检查点记录；
* 接下来，将数据库缓冲区的内容写入外存数据库；
* 最后，把外存日志中检查点的地址写入重启动文件。
* 上述工作的目的就是把检查点之前的工作持久化。

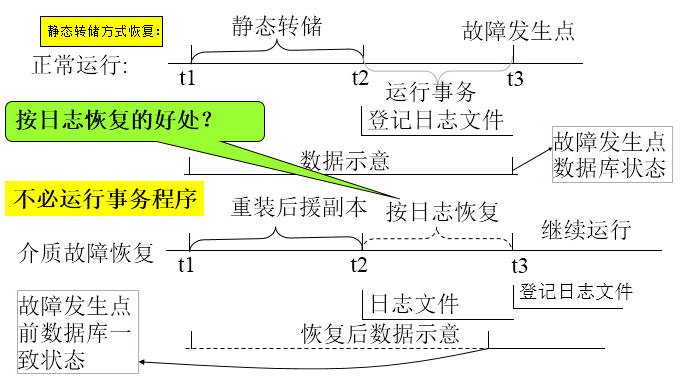
反做(undo)和重做(redo)恢复策略

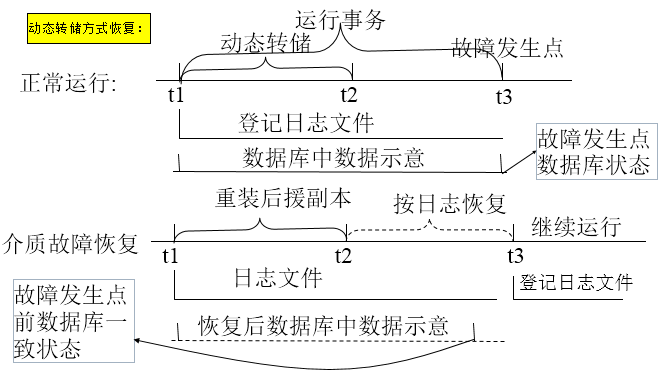
反做（undo）也称撤消、回滚（rollback） ，是将一个数据项的值恢复到其修改之前的值，即取消一个事务所完成的操作结果。当一个事务尚没提交时，如果缓冲区管理器允许该事务修改过的数据写到外存数据库，一旦此事务出现故障需废弃时，就需对被这个事务修改过的数据项进行反做，即根据日志文件将其恢复到前像。反做的目的是保持数据库的原子性。

重做（redo）也称为前滚操作（rollforward），是将一个数据项的值恢复到其修改后的值，即恢复一个事务的操作结果。当一个事务提交时，如果缓冲区管理器允许该事务修改过的数据不立刻写到外存数据库，一旦此事务出现故障，需对被这个事务修改过的数据项进行重做（redo），即根据日志文件将其恢复到后像。重做（redo）的目的是保持数据库的持久性。

软故障的恢复模型：当软故障发生时，造成数据库不一致状态的原因包括：一、一些未完成事务对数据库的更新已写入外存数据库；二、一些已提交事务对数据库的更新还没来得及写入外存数据库。因此，需要的恢复操作有undo和redo。

硬故障的恢复模型：硬故障的主要恢复措施是进行数据转储和建立日志文件。首先，DBA要定期地将数据库转储到其它磁盘上，形成一系列备份数据库（也称“后备副本”）。接下来，利用日志文件重新运行转储以后的所有更新事务，使数据库再进一步地恢复到故障发生时的状态。





### 集中式数据库的故障恢复

局部恢复系统的体系结构：数据库存储在永久性的外存设备上。数据库缓冲区用来存放最近执行的事务所使用的数据。数据库缓冲区被放置在具有挥发性的内存中，以页为单位来缓存数据。数据库缓冲区管理器负责读写数据库及缓冲区中的数据。局部恢复管理器与缓冲区管理器之间存在两个交互接口：读取数据页（fetch）和刷新数据页（flush）。

局部恢复管理器刷新数据库缓冲区数据页的过程：首先，由局部恢复管理器发出“刷新”（flush）命令并指明它想要刷新的数据页；然后，缓冲区管理器在当前缓冲区中查找该数据页，若存在，则将该页数据写回到数据库中；若不存在，还要将该页数据从数据库读取到缓冲区中，将其更改后写回到数据库。

(1) 数据的更新策略：数据库数据的更新通常采用两种更新方法，即原地更新和异地更新。原地更新，更新操作直接修改数据库缓冲区中的旧值。异地更新，更新操作将数据项新值存放在与旧值不同的位置上。异地更新的两种方法：影子页面和差分文件。影子页面：更新数据时，不改变旧存储页面，而是建立影子页面，将新值存于新建的影子页面上。旧页面用于故障恢复。差分文件：由于更新操作等价于先删除旧值，再插入新值。因此，差分文件(F)由只读部分(FR)加上插入部分(DF+)和删除部分(DF-)组成，即F =(FR∪DF+ ) -(DF-)。

(2) 缓冲区的更新策略：缓冲区更新策略有固定/非固定(fix/non\_fix)和刷新/非刷新(flush/no\_flush) 。固定/非固定(fix/non\_fix)：事务在执行过程中，缓冲区管理器是否要 等LRM下发命令，才将缓冲区中修改的内容写到外存数据库；刷新/非刷新(flush/no\_flush)：在事务执行结束后，LRM是否强制缓 冲区管理器将当前缓冲区中已修改数据写回外存数据库； 四种缓冲区更新策略：fix/flush、fix/no\_flush、non\_fix/flush、non\_fix/no\_flush

### 分布式事务的故障恢复

在分布式数据库系统中，除了可能出现集中式数据库系统中的故障外，还可能出现分布式数据库系统特有的故障。发生在各场地内部的故障称为场地故障。发生在场地间通信过程中的故障称为通信故障。

两阶段提交协议的终结协议的核心思想是：参与者通过访问其它参与者的当前状态来推断协调者的上一个决定，从而确定终结类型（全局提交还是全局废弃）。

两阶段提交协议的终结协议的内容：

步骤1：选择参与者PT作为新的协调者。

步骤2： PT向所有参与者发送“访问状态”命令，各参与者返回自身的状态。

步骤3： PT 根据各参与者当前的状态做出决定：

（1）若部分参与者处于“初始”状态，部分参与者处于“准备就绪” 状态，则PT 发送abort命令；

（2）若所有参与者均处于“准备就绪”状态，则PT 发送Commit命令；

（3）若至少一个参与者处于 “提交”状态，则PT发送Commit命令；

（4）若至少一个参与者处于“废弃”状态，则PT发送abort命令；

### 分布式可靠性协议

可靠性（Reliability）：在给定环境条件下和规定的时间内，数据库系统符合其行为规范的概率。强调正确性。

可用性（Availability）：在给定时刻t上，数据库系统正常运行的概率。强调可运行的能力。

可靠性和可用性往往相互矛盾。例如，数据a的副本分别存放在n个场地上，某事务采用两段提交协议来更新a。当协调者已决定提交，但在给各个参与者发送提交命令时发生通信故障。此时：

* 若要保证系统的可靠性，则要求各个参与者保持等待而不允 许其它事务使用a及其副本，以免造成数据的不一致，这就 降低了系统的可用性；
* 若要保证系统的可用性，则允许其它事务使用a及其副本， 但降低了系统的可靠性；

### 大数据库的恢复管理

大数据库系统每天需要操作的数据量越来越大，集群规模也在不断扩大，难免会发生一些故障。容错就是当由于种种原因在系统中出现了数据损坏或丢失时，系统能够自动将这些损坏或丢失的数据恢复到发生事故以前的状态，使系统能够连续正常运行的一种技术。

大数据库的恢复管理问题

* 支持自适应的故障检测：支持自适应的故障检测是数据库系统具有良好容错性能的前提。
* 支持与传统恢复管理策略的兼容性：将传统分布式数据库中基于事务的故障恢复策略与多副本管理策略相结合。
* 支持系统的高可用性：为了保证系统的高可用性，大数据库系统中的数据一般被冗余存储。

利用冗余资源来替代故障节点继续提供服务

保证故障发生后这些冗余资源之间的一致性

大数据库系统中的故障类型：事务内部的故障，系统故障，存储介质故障，通信故障。

大数据库系统的故障检测技术

* 基于心跳机制的故障检测
* 基于租约机制的故障检测
* 基于Gossip协议的故障检测

大数据库系统的容错技术

* 基于事务的大数据库容错技术
* 基于冗余的大数据库容错技术

以强一致性为目标的容错处理方法

以最终一致性为目标的容错处理方法

基于**心跳机制**的故障检测

* 基本思想：被监控的进程（或节点）定期发送心跳信息给监控进程（或节点），若在给定时间内监控进程（或节点）没有收到心跳信息，则认为被监控进程（或节点）失效。
* 例如，假设A和B分别为监控进程和被监控进程，A需要确认B是否发生故障，那么A每隔一段时间向B发送一个心跳包。如果一切正常，B将响应A的心跳包，并向A做出回应；若A重试一定次数后仍未收到B的响应，则认为机器B发生了故障。
* 存在的问题：会引发“双主”问题。如果B过于繁忙，都会导致B无法立即响应来自A的心跳包。若此时A认为B发生了故障，则A需要将B上面的服务迁移到集群中的其他服务器。而当网络问题被解决或者B闲下来后，B可能会继续提供服务，这样将出现多台服务器同时服务同一份数据而导致数据不一致的情况。

基于**租约机制**的故障检测的基本思想：监控进程定期向被监控进程**发放租约**，被监控进程持有的租约在有效期内才允许提供服务，否则主动停止服务。

### 基于冗余的大数据库容错技术

**以最终一致性为目标的容错处理方法**

* 最终一致性允许某时间片段内副本间数据的不一致，在该时间片段（不一致窗口）之后可以保证数据的一致性。
* 以最终一致性为目标的容错处理方法通过牺牲部分一致性，采用多副本冗余的方式来保证系统的高可用性。
* 在大数据库系统中，如Dynamo、Cassandra以及HBase等，均采用了以最终一致性为目标的容错处理方法。
* 在Dynamo中，采用多副本冗余的方式将一份数据分别写到编号为K、K+1、…、K+N-1的N台机器上。Dynamo的容错机制主要包括数据回传和数据同步两个阶段：
* 数据回传：若编号为K+i的机器(0≤i≤N-1)宕机或无法连接，原本写入该机器的数据将转移到编号为K+N的机器上。若在给定的时间T内机器K+i恢复，并且机器K+N通过Gossip机制感知到机器K+i的恢复，则机器K+N将数据回传到机器K+i上。
* 数据同步：若超过时间T，机器K+N仍未感知到机器K+i恢复，则机器K+N认为机器K+i发生了永久性异常，机器K+N将通过Merkle Tree机制从其他副本（K、…、K+i-1、K+i+1、…、K+N-1）进行数据同步。同一数据的不同副本在进行数据同步时分别对数据集生成一个Merkle Tree。
* Merkle Tree又称哈希树，可以是二叉树，也可以是多叉树。Dynamo通过比较不同副本的Merkle Tree来确定副本之间是否一致，并可以在log(N)时间内快速的定位是哪部分发生了变化。
* 叶子节点保存的是数据集合的单元数据或者单元数据哈希值（如node7的值为f1的哈希值）。非叶子节点保存的是子节点值的哈希值（如node3的值是根据node7和node8的值进行计算而得到的一个哈希值）。

## 分布式并发控制

### 基本概念

简单地讲，并发就是多个事务的同时执行，并发能够提高系统的效率，但也可能会带来三种错误。并发控制的主要目的是保证事务的一致性和隔离性，最终保证数据的一致性。当分布事务并发执行时，并发控制既要实现分布事务的可串行性，又要保持事务具有良好的并发度，以保证系统具有良好的性能。

并发控制就是利用正确的方式调度事务中所涉及的并发操作序列，避免造成数据的不一致性；防止一个事务的执行受到其它事务的干扰，保证事务并发执行的可串行性。

集中式数据库的可串行化问题：无论在集中式数据库系统中，还是在分布式数据库系统中，并发调度都要解决并发事务对数据库的冲突操作问题，使冲突操作串行执行，非冲突操作并发执行。在分布式数据库系统中，事务是由分解为各个场地上的子事务的执行实现的。因此，分布式事务之间的冲突操作，就转化为了同一场地上的子事务之间的冲突操作，分布式事务的可串行性调度也转化为了子事务的可串行性调度问题

### 基于锁的并发控制方法

基本思想：事务在对某一数据项操作之前，必须先申请对该数据项加锁，申请成功后才可以对该数据项进行操作。如果该数据项已被其它事务加了不相容的锁，那么后申请使用数据的事务必须等待，直到该数据项被解锁为止。锁分两种类型：排它锁(exclusive)和共享锁(shared)

封锁规则：事务在执行过程中需对其访问的数据项进行加锁，访问结束要及时释放其对数据项加的锁，以便供其它事务访问，保证多个事务正确地并发执行。具体封锁规则为：事务T在对数据项A进行读/写操作之前，必须对数据项A施加读/写锁，访问后立即释放已申请的锁；如果事务T申请不到希望的锁，事务T需等待，直到申请到所需要的锁之后，方可继续执行。

锁的粒度：封锁数据对象的单位称锁的粒度，指被封锁的数据对象的大小。锁的粒度也称锁的大小。系统根据自己的实际情况确定锁的粒度，锁的粒度可以是关系的属性（或字段）、关系的元组（或记录）、关系（也称文件）或整个数据库等。系统的并发度与锁粒度成反比。锁粒度越大，系统的开销越小，但降低了并发度。

两段封锁协议(2PL)是数据库系统中解决并发控制的重要方法之一，保证事务的可串行性调度。

2PL的实现思想是将事务中的加锁操作和解锁操作分两阶段完成，要求并发执行的多个事务要在对数据操作之前进行加锁，且每个事务中的所有加锁操作要在解锁操作以前完成。

两段封锁协议分为：基本的两段封锁协议、严格的两段封锁协议。

基本的两段封锁协议。基本的两段封锁协议分加锁阶段和解锁阶段。

阶段1：加锁阶段

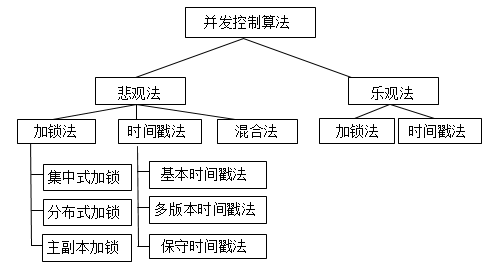
* 事务在读写一个数据项之前，必须对其加锁；
* 如果该数据项被其它使用者已加上不相容的锁，则必须等待。

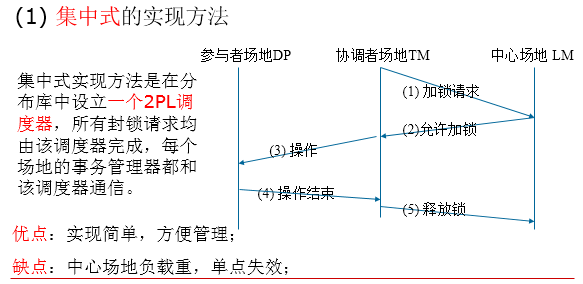
阶段2：解锁阶段

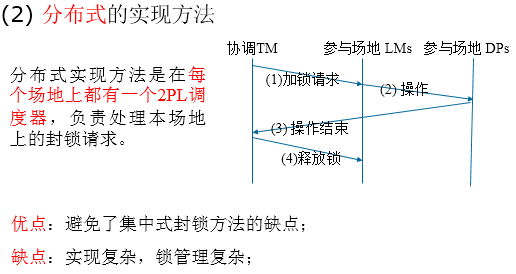
事务在释放锁之后，不允许再申请其它锁；

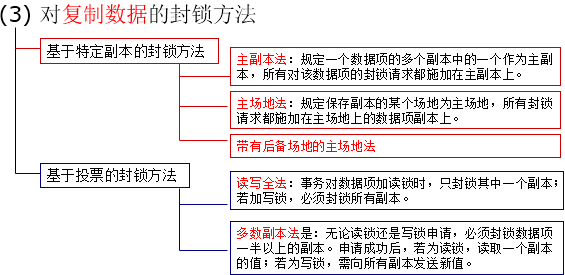
严格的两段封锁协议与基本的两段封锁协议内容基本上是一致的，只是解锁时刻不同，是在事务结束时才启动解锁，保证了事务所更新数据的永久性。采用两段封锁协议（2PL）的事务执行过程为：begin\_transaction→加锁→操作→ commit/abort→解锁。

### 分布式数据库的并发控制









基于时间戳的并发控制方法的实现：基于锁的并发控制是通过事务的互斥来维护并发事务的可串行化的；基于时间戳的方法是通过事务的优先级来实现并发事务的可串行化。事务的优先级通过时间戳来实现，事务管理器为每个事务Ti在其产生时都设置时间戳，TS(Ti)。

时间戳**(TimeStamp)**是基于事务启动时间点，由系统赋予该事务的全局唯一标识。同一事务管理器产生的时间戳是单调增加的，可以通过时间戳来区分事务。

设置时间戳的方法主要有以下几种：一种是使用全局单调递增计数器，但是全局的计数器比较难维护。

一般每个场地都有自己的单调递增的计数器,为了保证唯一性，时间戳是一个二元组<本地计数器值，场地id>，要保持各局部时钟的同步，则才能实现全局有序。

基于时间戳的并发控制方法的思想：给每个事务赋予一个唯一时间戳，根据时间戳对事务的操作执行顺序进行排序，则事务按时间戳顺序串行执行。如果发生冲突，撤销一个事务并重新启动，同时为重新启动事务赋予新的时间戳。

时间戳排序(TimeStamp Ordering)规则：分别属于事务Ti和事务Tk的两个冲突操作Oij和Okl， Oij在Okl之前执行当且仅当ts(Ti)<ts(Tk)，称Ti是较老的事务，Tk是较新的事务。

基于时间戳的事务并发控制的调度方法

①设ts是对数据X进行读操作的时间戳，若ts<rts(X)，则拒绝该操作，重新启动该事务并赋予新的时间戳；否则执行读操作，rts(X)=ts。

②设ts是对数据X进行写操作的时间戳，若ts<rts(X)或ts<wts(X)，则拒绝该操作，重新启动该事务并赋予新的时间戳；否则，执行写操作，wts(X)=ts

基于时间戳的并发控制方法不会出现死锁，任何事务不会被阻塞。若某个事务的某个操作不能执行，则事务重新启动。但是，基于时间戳的并发控制方法避免死锁是以重启动为代价的。

乐观的并发控制算法

* 悲观算法假设冲突经常发生，而乐观算法等到写阶段开始时，才进行冲突验证，也就是说延迟验证阶段直到执行写操作之前。
* 提交给乐观调度器的操作不会被延迟，每个事务的读、计算和写操作可以自由处理，不需要更新实际的数据库。
* 每个事务在初始时都在本地更新副本，验证阶段检查这些更新是否能够维护数据库的一致性，如果结果是肯定的，就把这些更新写到实际的数据库中。否则，该事务就废弃并被重启。

### 分布式死锁管理

在使用锁的并发控制中，可能导致死锁的产生。产生死锁原因：

* 因为在多个事务并发执行的情况下，每个正在执行的事务已对其拥有的资源进行加锁。
* 已经占有了部分资源的事务可能会因为其他事务对其需要的资源已经加了不相容的锁而处于等待状态。
* 多个彼此相互等待对方释放自己需要的资源的事务就会产生死锁。

因此，在分布式数据库管理系统中必须设计合适的处理死锁的算法。

死锁的检测：

(1) **集中式死锁检测**

* 选择一个场地执行整个系统的死锁检测程序(也称为集中死锁检测器)。
* 每个场地的锁管理器都要向该死锁检测器传输其LWFG，从而逐渐构成GWFG。如果在GWFG中发现回路，则认为存在死锁并采取相应的处理措施。
* 不足：频繁的LWFG传输会带来大量的通信代价，因此，各个场地的锁管理器只需传输其等待图中变化部分的信息，以减少通信量。

(2) **层次死锁检测**

* 是对集中式死锁检测方法的改进，目的是减少通信量。
* 层次死锁的检测方法是建立一个死锁检测层次树，每个站点的局部死锁检测程序(LDD)作为一个叶子节点，中间层节点是部分全局死锁检测程序(PGDD)，根节点是全局死锁检测程序(GDD)。
* 每个场地首先在本地用等待图检测死锁并做出相应的处理，然后将局部等待图发送给上层节点。每个非叶子都至少包括两个下层节点，从底层至上层，部分全局死锁检测程序可以检测其所包括的下层节点的事务是否产生死锁，直到最顶层的根节点，由根节点死锁检测程序判断全局事务是否产生死锁。

(3) 分布式死锁检测

* 检测由各个场地共同完成，每个场地都是对等的，不区分全局死锁和局部死锁的检测程序，都承担检测全局死锁的任务。
* 在检测过程中，局部WFG与其它场地的WFG相互通信，测试是否存在回路。

死锁的预防与避免:(1)死锁的预防,顺序封锁法。预先对数据对象规定一个封锁顺序，所有事务都按照这个顺序实行封锁，避免在等待图中出现回路。(2)死锁的解决:使用事务的时间戳来解决死锁，基本思路是：杀死死锁等待图中的某个事务，破坏环路。