厦门大学计算机科学系研究生课程

《分布式数据库技术》

专题四 分布式数据库的事务管理和恢复（2012年新版）

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：http://www.cs.xmu.edu.cn/linziyu

专题四 分布式数据库的事务管理和恢复

第5章 分布式事务管理

第6章 分布式并发控制

第7章 分布式恢复

第5章 分布式事务管理

- 5.1 分布式事务概念与模型
- 5.2 分布式事务的原子性
- 5.3 分布式事务可串行化理论

5.1 分布式事务概念与模型

5.1.1 集中式事务概念

1、 事务

事务是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

2、 集中式事务模型

事务由标识符、头部、数据库的操作、结束处理组成。

其中Commit表示提交，Abort（或Rollback）表示事务夭折回滚。

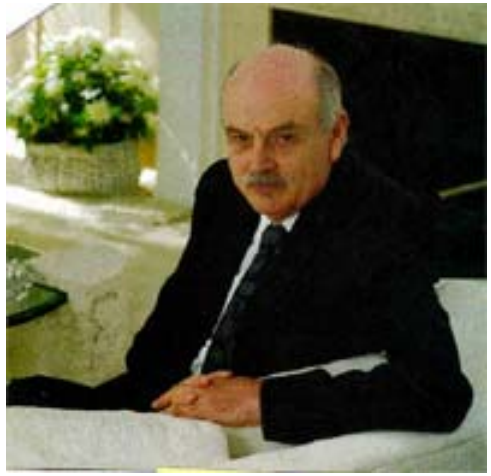
T_i： Begin Transaction

.
.
. } DB operation

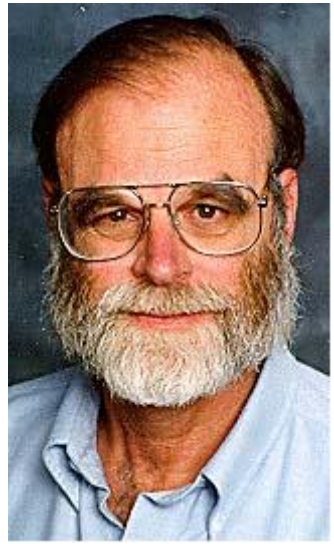
Commit or Abort opertion

（回顾）数据库研究界三大功勋人物

查尔斯·巴赫曼
Charles W.Bachman
网状数据库之父
1973年图灵奖获得者



埃德加·科德
Edgar Frank Codd
关系数据库之父
1981年图灵奖获得者



詹姆斯·格雷
James Gray
1998年图灵奖获得者
数据库事务处理专家

5.1.1 集中式事务概念

3、集中式事务特性（事务四性： ACID或ASID）

- 原子性(Atomicity)
- 可串行性 (Serializability)
  (一致性Consistency)
- 隔离性(Isolation)
- 持久性(Durability)

5.1.2 分布式事务模型

5.1.2.1 分布式事务概念

■在分布式数据库系统中，任何一个应用的请求最终将转成对数据库的存取操作序列

■从外部特征看，分布式事务与前面讲的集中式事务一样，它也是一个或一段应用的操作序列，只是它的执行方式和集中式事务的执行方式不同而已

定义1：分布式事务是一个应用的操作序列，是用户对数据库存取操作序列的执行的最小单元。

■分布式事务也具有事务的头部操作序列和事务的尾部

■从其外表上看，和集中式事务具有相同的事务模型，它也使数据库从一个一致的状态改变到另一个一致的状态

5.1.2.1 分布式事务概念

定义2：分布式事务对分布式数据库的存取，经转换、分解、优化后，产生一个涉及到用通讯原语联系的、针对多个局部数据的存取操作序列，称为分布执行计划（DEP）。这个分布执行计划可以分解为“涉及相应场地上的局部数据库的操作序列”，即各场地上的子DEP组成。通常称分布式事务为全局事务，而涉及各相应场地的子DEP称之为子事务。

定义3：设事务T：｛O_1 ,O_2 ,…,O_m｝，则全局事务GT可表示为：

$$GT: \xrightarrow[\text{优化}]{\text{转化、分解}} DEP\{O_1^1, O_2^2, \cdots\cdots O_m^n\}$$

DEP可以分解为：｛DEP^1（ $O_1^1, O_2^1, \cdots\cdots O_m^1$），……，$DEP^n$（ $O_1^n, O_2^n, \cdots\cdots O_m^n$ ）｝

其中：1，…，n为场地号，m为操作种类，DEP^i（i=1，…，n）为涉及场地i的操作序列组成的子DEP，即全局事务的第i个子事务。

5.1.2.2 分布式事务过程

- **分布式事务过程和集中式事务过程的相似性**

和集中式事务一样，分布式事务具有**ACID**特性，并且更带有分布执行时的特性，即全局事务在并发执行时要求可串行性、原子性、持久性、隔离性。而全局事务的子事务在调度执行时也要求可串行性、可恢复性

- **分布式事务要保证全局和局部数据库一致性**

全局事务的调度保证事务的正确执行，包括涉及各物理场地的局部调度，保证局部数据库的一致，同时也包括通过通讯协议对子事务的执行进行协调，保证全局数据库的一致性

- **分布式事务的协调者进程和代理者进程**

为了完成各场地的子事务的处理功能，全局事务必须为每一子事务在相应的场地上创建一个代理者进程，每一个代理者都是一个局部进程。同时，为了协调各子事务的操作，全局事务还要启动一个协调者进程，来控制和协调各代理者间的操作，进行代理者间的通讯

5.1.2.2 分布式事务过程

从分布式系统的概念来分析，分布式事务的过程如下：

- 分布式事务在执行时将被分解为若干个和各场地上计算机相关的操作序列，即子事务
- 为了保证事务的正确调度执行，分布式事务必须为每个子事务在相应场地上的计算机上创建一个代理进程执行该子事务，称局部进程。只有当各子事务均正常结束后，分布式事务才可以提交
- 在子事务的调度执行中，必须有一个协调者进程去协调各子事务的执行。一般来说，这样的协调进程由分布式事务的始发场地事务，即根事务承担
- 在分布式事务中，由于各子事务分布在系统中的多个场地的计算机上执行，因此子事务之间存在大量的数据及控制信息需要传输。这些传输由系统中提供的通讯原语完成，同时子事务间的协调亦由控制原语完成

综上所述，分布式事务的操作包括：有关数据的操作（即子事务操作）、通讯原语和控制原语。

5.1.2.3 分布式事务进程结构

一般有多种方法来构造协调进程的结构，这里我们先假设如下所述的进程结构：

（1）存在一个协调者进程，这个进程一般由事务的始发场地启动，当用户提出一个用户请求时，就在请求的始发场地启动该进程。

（2）协调者进程负责控制事务的执行，协调代理者之间的操作，发出事务的开始、提交或夭折等原语。

（3）只有协调者才可以请求创建新的代理者。

5.1.2.4 分布式事务实例

例5.1：本例给出了一个应用程序员编写的事务，该事务执行两个帐户间的转帐操作，其操作的对象是全局关系ACCOUNT（Acc No，Amount）。该应用首先读取转帐金额（$Amount）及贷方帐号和借方帐号（$From Ace和$To Ace），然后启动事务，进行事务的处理操作。在事务的运行过程中，若贷方金额小于转帐金额，则中止事务；否则，更新借贷双方的金额，提交事务。

5.1.2.4 分布式事务实例

集中式事务

```
Fund_Transfer:
Read (Temninal, $Amount, $Form_Acc, $To_Acc) ;
begin_Transaction;
Select Amount into $Form_Amount
From ACCOUNT
Where  ACC_No = $From_ACC
If  $From_Amount-$Amount < 0 then Abort
else begin
Update ACCOUNT
Set Amount=Amount－ $Amount
Where ACC_No= $From_ACC
Update ACCOUNT
Set Amount=Amount+$Amount
Where ACC_No=$TO_ACC;
Commit
End
```

全局级资金转移事务

5.1.2.4 分布式事务实例

分布式事务

```
ROOT:
   read (terminal,$Amount, $From_ACC,
         $From_ACC, $TO_ACC) ;
   begin_Transaction;
   Select Amount into $From-Amount
   From ACCOUNT
   Where Acc_NO=$From-Acc;
   if $From_Amount-$Amount<0 then abort
   else begin
      Update ACCOUNT
      Set Amount=Amount-$Amount
      Where Acc_No=$From_Acc;
      Create AGENT1     (创建代理者进程)
        Send to AGENT1 ($Amount, $To_Acc) ;
      Commit
   End
AGENT1:
      Receive From ROOT ($Amount, $To_ACC) ;
      Update ACCOUNT
      Set Amount=Amount+$Amount
      Where Acc_No=$To_Acc;
```

全局事务协调者进程

代理者进程

5.1.3 分布式事务特性

与集中式数据库一样具有ACID性，此外，还有分布式数据库自己的特性：

- **执行特性：**必须创建一个控制进程，称协调进程，以协调各子事务的执行
- **操作特性：**数据的存取操作序列之外，还必须加入大量的通讯原语，即事务组成复杂，执行方式也复杂
- **控制报文：**增加了控制报文，以对各子事务的操作进行协调

5.1.4 分布式事务管理目标

- **宏观目标**：追求高效、可靠、并发地执行事务
  - 这三个目标是相互矛盾的
  - 影响执行效率的因素：
    - CPU和内存的利用率
    - 控制报文的数量及长度
    - 响应时间
    - 可用性
- **具体目标**：维护分布事务的原子性、可串行性及隔离性、持久性，尽量减少CPU及内存的开销，减少控制报文传送的次数，加速事务的响应速度，获得最大的系统可用性

5.1.5 分布式事务管理的实现

- 在分布式数据库系统中，事务由若干个不同场地上的子事务组成。因此，为了维护事务的特性，分布式事务管理不仅要将每个场地上的子事务考虑在内，而且要在全局上对整个分布式事务进行协调和维护。
- 5.1.5.1 LTM与DTM
- 5.1.5.2分布式事务执行的控制模型
- 5.1.5.3分布式事务管理的实现模型

5.1.5.1 LTM与DTM

- 与集中式数据库系统中的事务管理不同，分布式事务管理在功能上分类两个层次：局部事务管理器（LTM）和分布式事务管理器（DTM）。
- LTM类似于集中式数据库系统中的事务管理器，用来管理各个场地的子事务，负责局部场地的故障恢复和并发控制；而对于整个分布式事务，由驻留在各个场地上的DTM协同进行管理。
- 由于各场地上的DTM之间可以互相通信且目标一致，因此，在逻辑上可以将这些DTM看作是一个整体。

5.1.5.1 LTM与DTM

LTM和DTM之间的功能和特点的比较：

- LTM将各个场地上的子事务作为操作对象；而DTM的操作对象是分布式事务。
- LTM的操作范围局限在某个场地内，而DTM的操作范围是该事务所涉及的所有场地。
- LTM要实现的目标是保证局部事务的特性；而DTM的目的是保证全局事务的特性，特别是分布式事务的原子性。DTM要保证每一场地的子事务要么都成功提交，要么都不执行。也就是说，所有LTM都必须采用相同的事务执行策略（提交或终止），使得各个子事务遵循一致的决定。
- LTM负责接收DTM发来的命令，记入日志后执行命令，并向DTM发送应答；DTM负责向LTM发送命令，通过接收应答对这些场地进行监控，以实现合理地调度和管理各个子事务。

5.1.5.1 LTM与DTM

表 局部事务管理器和分布式事务管理器的特点比较

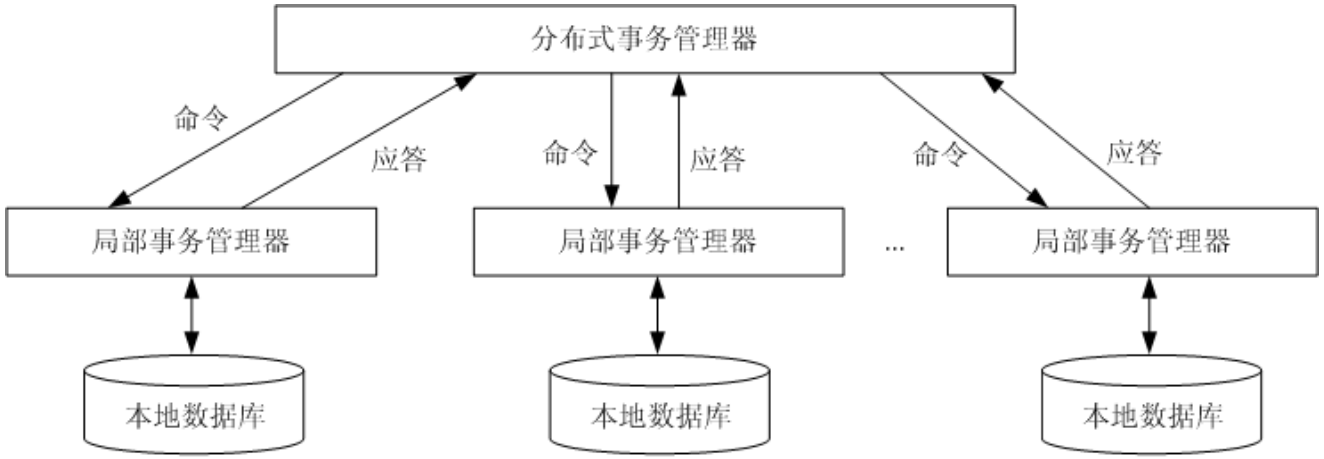
比较项	局部事务管理器	分布式事务管理器
操作对象	子事务	分布式事务
操作范围	局限在某个场地内	事务所涉及的所有场地
实现目标	保证本地事务的特性	保证全局事务的特性
执行方式	接收命令，发送应答	发送命令，接收应答

5.1.5.2分布式事务执行的控制模型

- DTM和LTM是分布式事务管理的两个重要组成部分，如何使得二者能够有效地进行协同工作，既保证本地事务的特性，同时也保证全局事务的特性呢？
- 为此，需要针对分布式事务的执行过程建立控制模型，以实现DTM与各个LTM之间的协作有条不紊地进行。
- 在分布式数据库系统中，分布式事务执行的控制模型主要包括三种：主从控制模型、三角控制模型和层次控制模型。

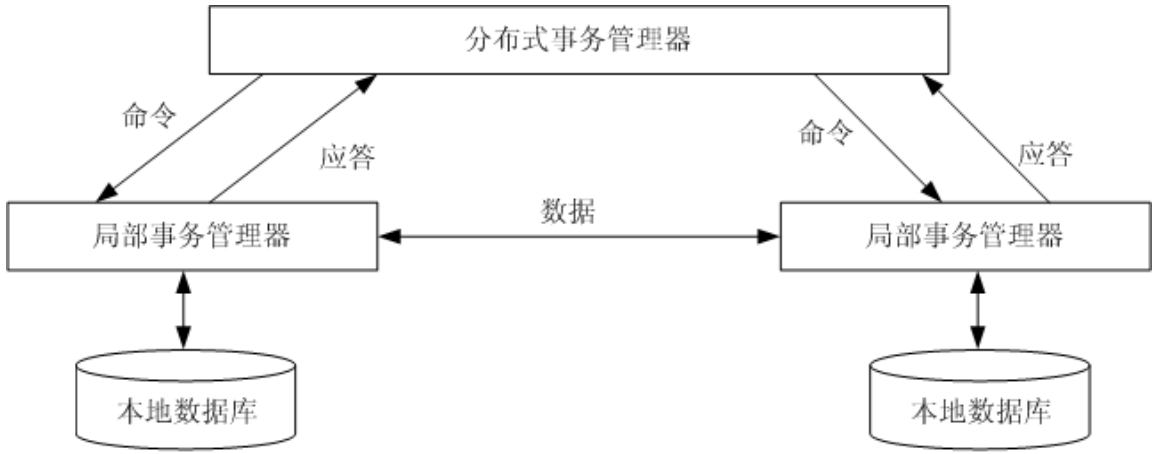
5.1.5.2.1 主从控制模型

- 如图所示，在主从控制模型中，DTM作为主控制器，而LTM则作为从控制器。
- DTM通过向LTM发送命令并收集应答来对这些场地进行状态监控，并产生统一的命令供各个LTM执行。
- LTM根据DTM的命令来执行本地的子事务，并将最终结果返回给DTM。
- 也就是说，DTM与各个LTM采用这种一问一答的方式来控制需要同步的操作。
- 需要强调的是，在分布式事务执行的主从控制模型中，事务之间的通信，只发生在DTM和LTM之间，而LTM之间没有通信。若LTM之间需要传递参数等信息，则必须经过DTM转发来实现。



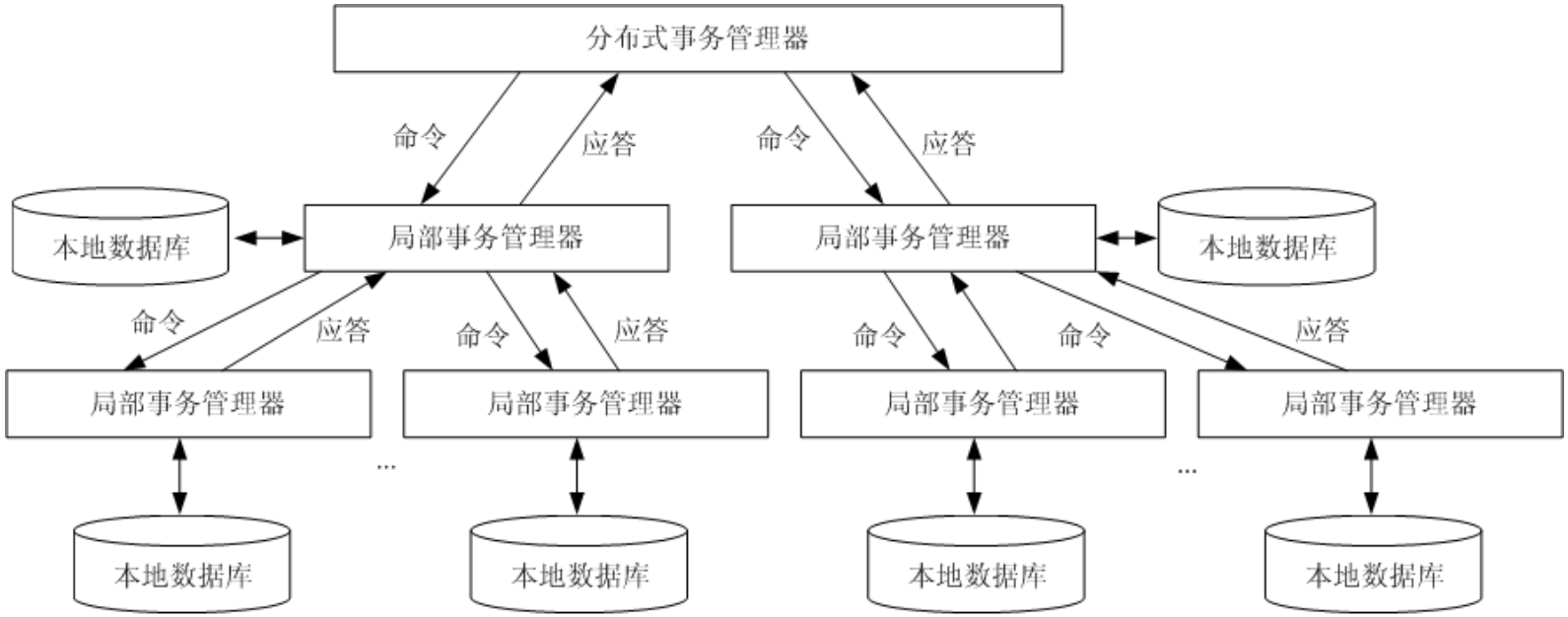
5.1.5.2.2 三角控制模型

- 如图所示，三角控制模型与主从控制模型类似，在DTM与LTM之间可以传递命令和应答。
- 二者的差别是：三角控制模型中，LTM之间可以直接发送和接收数据，而不需要通过DTM作为中介。
- 因此，与主从控制模型相比，三角控制模型在一定程度上减少了不必要的通信代价，但同时也使得DTM与LTM之间的控制变得更加复杂。



5.1.5.2.3 层次控制模型

- 如图所示，在层次控制模型中，每个LTM本身可以具有双重角色，除了用来管理其本地事务以外，LTM也可以兼职作为一个新的DTM。
- LTM可以将其负责的子事务进一步分解，同时衍生出一系列下一层的局部事务管理器，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DTM，由其来控制下一层各个LTM的执行，向它们发送命令并接收应答。
- 分布式事务执行的层次控制模型，允许进行扩展设计，其层数可以随着任务量的增大而增加，因此，实现复杂性要比主从和三角控制模型高。

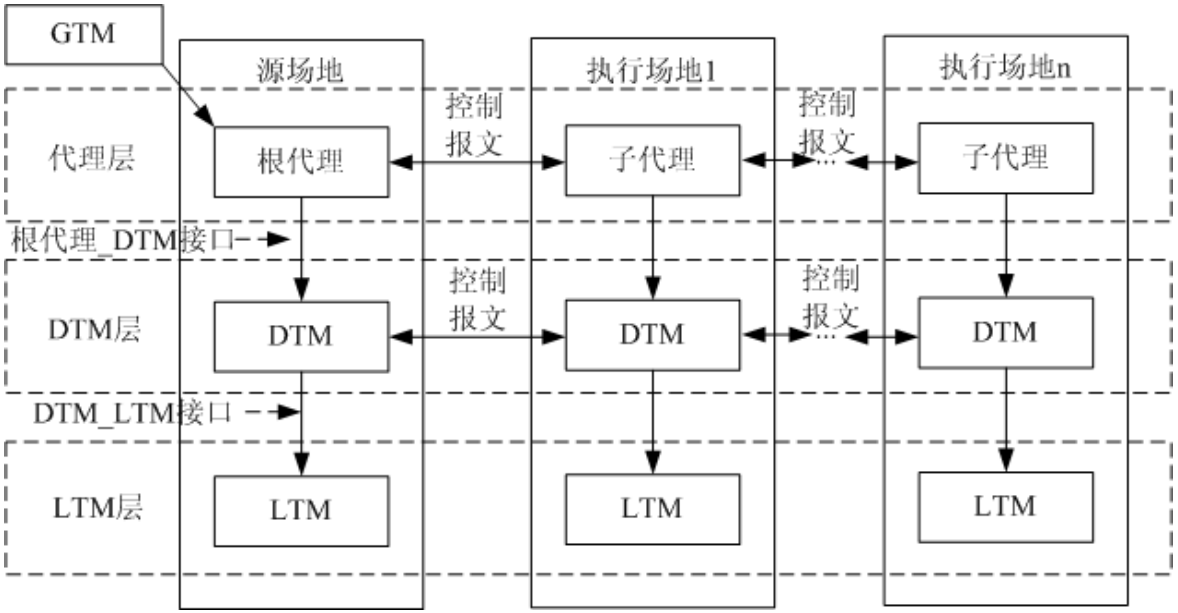


5.1.5.3分布式事务管理的实现模型

分布式事务管理的实现模型，自顶向下包含3个层次：

- 代理层：由1个根代理和若干个子代理组成，它们之间通过发送控制报文来协同工作；
- DTM层：由驻留在各个场地上的DTM组成，负责管理分布式事务，调度子事务的执行，保证分布式事务的原子性；
- 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，保证子事务的原子性。

除此以外，在分布式事务管理的实现过程中，还需要一个全局事务管理器(GTM)，负责处理全局事务调度及将全局事务划分成多个子事务。



5.1.5.3分布式事务管理的实现模型

- 在分布式事务管理的实现模型中，层次间的通信涉及两种接口类型：根代理_DTM接口和DTM_LTM接口。
- 其中，代理层与DTM层之间的通信是通过根代理_DTM接口来实现的，根代理可以向DTM发送begin_transaction、commit、abort或create原语。
- 需要注意的是，一般规定只有根代理才能与DTM层具有接口关系，而不允许子代理与DTM层直接进行通信。
- DTM层与LTM层之间的通信是通过DTM_LTM接口实现的，由DTM向LTM发送local_begin、local_commit、local_abort或local_create原语。

根代理_DTM接口

begin_transaction	全局事务开始命令
commit	全局事务提交命令
abort	全局事务终止命令
create	全局创建子代理

DTM_LTM接口

Local_begin	局部事务开始命令
Local_commit	局部事务提交命令
Local_abort	局部事务终止命令
Local_create	局部创建子代理

5.1.5.3分布式事务管理的实现模型

例5.1中银行账户转账事务的实现过程如图所示，执行过程如下：

（1）根代理通过“根代理_DTM接口”向本地（场地i）的DTM发送begin_transaction命令，全局事务开始执行；

（2）DTM通过“DTM_LTM接口”向本地的LTM发送local_begin命令，启动局部事务执行，在局部事务执行之前，LTM要将当前局部事务信息写入日志；

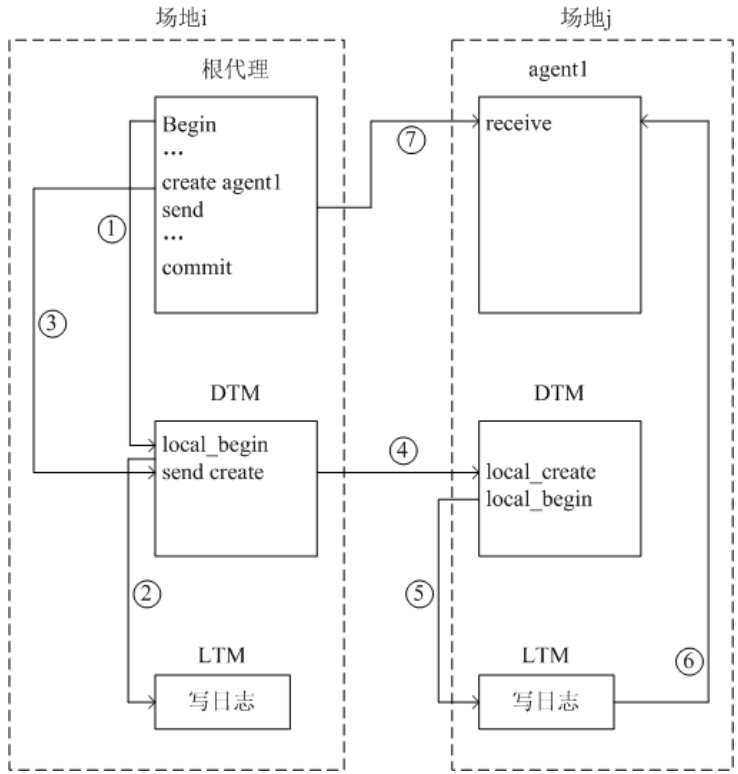
（3）根代理通过create原语创建子代理agent1；

（4）根代理的DTM通知子代理agent1所在的场地（场地j）的DTM，子代理agent1所在场地的DTM向本地LTM发送local_begin命令，建立并启动局部事务进程；

（5）在局部事务执行之前，子代理agent1所在场地的LTM会将当前局部事务信息写入日志；

（6）子代理agent1所在场地的LTM通知并激活子代理agent1来执行子事务；

（7）子代理agent1接收根代理发来的参数信息。



5.2 分布式事务的原子性

5.2.1 关于原子性

■原子性：一个事务的操作要么全部执行，要么全部不执行。事务的原子性保证数据库的状态总是从一个一致的状态变化到另一个一致的状态，而不会出现不一致的中间状态

■为了维护分布式事务的原子性，要求分布式事务管理程序具有实现全局原语Begin_Transaction、Commit / Abort的能力

■全局原语的执行依赖于在各场地上执行的一系列的相应操作。只有当各局部场地上的操作正确执行后，全局事务才可以提交，当发生故障要夭折全局事务时，则所有的局部场地上的操作都应失效

■为了完成全局事务的处理，局部场地应有实现局部事务的局部事务管理程序（LTM），因此可以利用局部场地提供的事务管理机制实现分布式事务的原子性

5.2.2 局部场地事务保证原子性的管理机制

(1) 局部场地事务管理机制

局部场地提供的事务管理相当于集中事务管理，而集中式事务管理保证事务原子性的方法是事务恢复机制。

(2) 集中式事务恢复机制

事务恢复机制的目的在于，系统发生故障时，保证数据库处于一种正确的状态。

5.2.3 分布式事务保证原子性的管理机制

对分布式事务来说，其原子性的维护也是由恢复机制保证的

（1）分布式事务恢复机制的基本原理

（2）保证分布式事务的原子性的措施

5.2.3 分布式事务保证原子性的管理机制

(1) 分布式事务恢复机制的基本原理

① 每一个场地有局部事务管理程序（LTM），并具有进行局部恢复的机制

② 全局事务管理程序(GTM)具有Begin-Transaction、Commit和Abort原语，可以向LTM发送Commit和Abort命令

③ GTM要求LTM具有以下功能：

- ✓ 确保子事务的原子性
- ✓ 代替GTM在日志文件中写入一些记录

④ 全局事务的原子性由全体子事务共同保证：

- ✓ 每个场地的子事务的原子性
- ✓ 全部场地的子事务同时提交或中止

⑤ 全局事务的管理除了调用LTM外，还增加了子事务间的协调工作，以决定全体子事务是提交还是夭折，最终决定全局事务是提交或是夭折

（2）保证分布式事务原子性的措施

为了实现分布式事务的提交，最基本的方法是**两段提交协议（2PC）协议**。另外还有：

- ✓ 非阻塞提交协议（3PC协议）
- ✓ 多副本恢复
- ✓ 网络分割

两段提交协议（2PC）

■**第一阶段：预提交阶段**

（1）由协调者进程向各场地发出子事务；

（2）当各参与场地子事务完成后，向协调进程发出预提交命令，同时在日志文件中记入该子事务提交所需的全部信息及子事务预提交的事实；

（3）协调进程从各子事务收到预提交命令，如果所有的子事务均回答了预提交，则全局事务可以提交；如果至少有一个子事务回答了Abort，或超时未回答，则全局事务做出夭折的决定。

■**第二阶段：执行阶段**

（1）协调者在日志中记入其决定后，由协调者向各子事务发出其决定（提交或夭折）；

（2）所有的参与者根据协调者的命令，先记入日志文件，然后执行提交或中止的命令，从此时起，恢复机制可以保证子事务的实施，最后向协调者发出最终回答；

（3）协调者根据回答写入“完成”记录到日志文件中。这样，全局事务即是可恢复的，从而保证了分布式事务的原子性。

5.3 分布式事务可串行化理论

5.3.1 基本概念

5.3.1.1 集中式事务的可串行性概念

5.3.1.2 串行调度

5.3.1.3 可串行化调度

5.3.2 可串行化原理

5.3.3 可串行化的判定

5.3.1.1 集中式事务的可串行性概念

可串行性是并发执行的几个事务，其操作的结果应与以某种顺序串行执行这几个事务所得出的结果相同，因此称为可串行性。

这种可串行化的并行调度是由数据库系统的并发控制机制来完成，以保证并发事务执行时的数据库状态的一致。所以这种性质也称为事务的一致性。

5.3.1.2 串行调度

定义：如有一组事务{T_1，T_2，…，T_i，…，T_n}，假设事务T_i的操作先于T_j，记为$T_i<T_j$。如果一个调度S，其每一个事务的执行，均有 $T_i<T_j$（i、j的序号无关），记为：S={…$T_i<T_j$<…}，则称S是一个**串行调度**。

S的执行可能出现以下三种情况：

① S正确执行完成，则S中每一事务均提交；

② S在执行T_k时发生故障，T_k以前执行的事务均提交，则夭折T_k，使数据库的状态恢复到T_k以前的状态；

③ S在执行T_k时发生故障，T_k以前执行的事务有的被夭折，则夭折T_k，恢复数据库状态时，重做T_k以前提交的事务，撤消T_k以前夭折的事务。

结论：一个串行调度总可以正确执行，使数据库的状态保持一致。

5.3.1.3 可串行化调度

可串行化调度：执行结果和某种串行调度执行结果等价的并发调度。

- **事务的并发调度要解决的最基本问题**是并行执行事务对数据库的冲突操作（如读一写、写一写等），让冲突操作串行执行，非冲突操作并行执行
- 分布式事务之间的冲突操作最终转化为同一场地上子事务间的冲突操作，所以其冲突的几率比集中式事务要小
- 分布事务的可串行性调度可以转化为子事务的可串行化调度。涉及多副本选择时，分布式事务调度要多做一个副本选择操作，以避免冲突操作，各子事务的调度执行则是由LTM去实现

5.3.2 可串行化原理

定义5.1 一个事务是一个偏序集，$T_i = (\sum_i, \angle_i)$，其中，

（1）$\sum_i$ 是操作符集，包含 $\{r_i[x], w_i[x] \mid x\text{是数据项}\} \cup \{a_i \mid c_i\}$

（2）$\sum_i$ 中，a_i和c_i两者只出现一个

（3）a_i或c_i 是 $\sum_i$ 中的最后一个操作符

（4）如果 $r_i[x], w_i[x] \in \sum_i$，则它们必满足$r_i[x]<w_i[x]$或$w_i[x]<r_i[x]$

这里只给出了事务几种简单的定义，事实上，事务可能不止以上几种操作符，比如，以锁方式实现并发控制时，事务包含加锁、解锁操作，分布式事务还包含通讯原语等。

定义5.2 事务的一个操作是指，组成事务T_i操作符序列中的任一个$r_i[x]$,$w_i[x]$,c_i或a_i或公式。

5.3.2 可串行化原理

定义5.3 如果两个操作p和q对同一数据项x进行操作，其中一个是写操作w[x]，则p和q称为冲突操作。

定义5.4 对于一组并发事务$T_1,T_2,\cdots,T_n$的一个调度S，S是一个偏序集 $S(\sum,\angle)$ 其中，

（1）$\sum=\bigcup_{i=1}^{n}\sum_i$

（2）$\angle\supseteq\bigcup_{i=1}^{n}\angle_i$

（3）对于任意两种冲突操作p和q$\in$ S,则有p<q或q<p

定义5.5 对于调度S中的任意两个事务T_i和T_j，若 $\bigcup_{i=1}^{n}\sum_i<\bigcup_{j=1}^{m}\sum_j$或$\bigcup_{j=1}^{m}\sum_j<\bigcup_{i=1}^{n}\sum_i$ 则称S是串行调度。

5.3.2 可串行化原理

定义5.6 在调度中，使得数据库从一个一致的状态改变到另一个数据状态的调度，称为一致性调度。

定义5.7 两个调度S_1和S_2是等价的，当且仅当：

（1）调度S_1和S_2在同一事务集上定义，且具有相同的操作集；

（2）对于有冲突操作的事务，有

① $a_i, a_j \notin S_1$，且$a_i, a_j \notin S_2$；

② 若$p_i <_{S_1} q_j$，则$p_i <_{S_2} q_j$。

定义5.8 与串行调度等价的并发调度，称为可串行化调度。

5.3.2 可串行化原理

例**5.2**： 让我们对以下的两个事务进行调度，共产生五种不同的调度：**S1,S2, S3, S4,S5**。

T1： Read A; A=A+10; Write A;
Read B; B=B-15; Write B;

T2： Read A; A=A-20; Write A;
Read B; B=B*2; Write B;

5.3.2 可串行化原理

T_1	T_2
Read A	
A:=A+10	
Write A	
Read B	
B:=B-15	
Write B	
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B

(a)S_1

T_1	T_2
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B
Read A	
A:=A+10	
WriteA	
Read B	
B:=B-15	
Write B	

(d)S_4

S1、S4是串行调度，是一致性调度

5.3.2 可串行化原理

T_1	T_2
Read A	
A:=A+10	
Write A	
Read B	
B:=B-15	
Write B	
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B

(a)S_1

T_1	T_2
Read A	
A:=A+10	
Write A	
	Read A
	A:=A-20
	Write A
Read B	
B:=B-15	
Write B	
	Read B
	B:=B*2
	Write B

(b)S_2

S2和**S1**的冲突操作具有相同的序，因此是二者等价的，且**S2**是一致性调度， **S2**还是一个可串行化调度

5.3.2 可串行化原理

T_1	T_2
Read A	
A:=A+10	
Write A	
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B
Read B	
B:=B-15	
Write B	

(c)S_3

T_1	T_2
Read A	
A:=A+10	
Write A	
Read B	
B:=B-15	
Write B	
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B

(a)S_1

T_1	T_2
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B
Read A	
A:=A+10	
WriteA	
Read B	
B:=B-15	
Write B	

(d)S_4

S3虽是一个一致性调度，但根据定义**5.7**，它既不与**S1**等价，又不与**S4**等价，所以**S3**不是可串行化调度。

5.3.2 可串行化原理

T_1	T_2
	Read A
	A:=A-20
	Write A
Read A	
A:=A+10	
WriteA	
	Read B
	B:=B*2
	Write B
Read B	
B:=B-15	
Write B	

(e)S_5

T_1	T_2
	Read A
	A:=A-20
	Write A
	Read B
	B:=B*2
	Write B
Read A	
A:=A+10	
WriteA	
Read B	
B:=B-15	
Write B	

(d)S_4

S5和S4等价，S5是一致性调度，也是可串行化调度

5.3.2 可串行化原理

结论

- 一个可串行化调度，必定与某一串行调度等价，且是一致性调度
- 一致性调度却不一定是可串行化调度
- 同一事务组的几个可串行化调度，其结果未必相同（比如：S2和S5都是同一事务组上的可串行调度，但结果不同）

T_1	T_2
Read A	
A:=A+10	
Write A	
	Read A
	A:=A-20
	Write A
Read B	
B:=B-15	
Write B	
	Read B
	B:=B*2
	Write B

(b)S_2

T_1	T_2
	Read A
	A:=A-20
	Write A
Read A	
A:=A+10	
WriteA	
	Read B
	B:=B*2
	Write B
Read B	
B:=B-15	
Write B	

(e)S_5

5.3.3 可串行化的判定

对并发调度的可串行化判定，若用定义去判断比较复杂，一般利用有向图进行判断

定义5.9 对于一个事务的集合$S=\{T_1,T_2,\ldots,T_n\}$，S转化为有向图SG。图中的结点是每个事务T_i，图中的边$T_i \rightarrow T_j$，表示两个事务T_i与T_j具有冲突操作，$p \in T_i$，$q \in T_j$，且p先于q执行。

如果能够找到一个串行调度和有向图SG的所有边一致，那么S与该串行调度等价，则S是可串行化调度。

定理5.1 调度S是可串行化的，当且仅当S的有向图是无回路的。（证明：略）

5.3.3 可串行化的判定

■可串行化理论是事务并发的基础。判断一个调度是否为一致性调度，只需判断其是否可串行化就行了。任何可串行化的调度都是一致性的。

■在分布式数据库中，事务是分解为子事务在相应的场地上执行的。对一组分布事务$T_1,T_2,\ldots,T_n$，在m个场地上的一次执行，是由每个场地上的调度序列$S_1,\ldots,S_m$来安排的。而每个场地的调度序列由LTM调度，其可串行性可由前面的可串行化理论进行判定，然而，仅仅对局部进行可串行化调度是不够的，还需要考虑全局的串行执行，因为当全局不串行时，冲突操作可能互相锁住对方的资源，从而造成死锁。

5.3.3 可串行化的判定

■全局正确调度要满足的条件是：

①每个局部调度的可串行性；

②对于两个已执行某操作的事务所在的场地，它上面的事务存在一个总的次序。若在此次序中存在$T_i<T_j$，则对于一串行调度S也应满足$T_i<T_j$。

定理5.2 令$T_1,T_2,\ldots,T_n$是一组事务，E是这组事务的一个调度，$S_1,S_2,\ldots,S_m$表示对应场地上的调度序列，则E是一致性调度的条件为：如果这些事务存在一个总次序，在此总次序中T_i先于T_j执行，那么事务T_i和T_j的每对冲突操作O_i和O_j，在相应的局部调度中，O_i也总是先于O_j执行。

定理5.2可用于判定一个分布式事务的正确性。

第6章 分布式并发控制

6.1 分布式并发控制概念

■分布式事务包括全局事务和子事务，分布式事务之间的冲突操作可以被转化为同一场地上的子事务的冲突操作

■多个分布式事务 T_i，T_j 及 T_k ……中，几个读写操作的执行序列为：

S1：$R_i(x)$ $R_j(x)$ $W_i(y)$ $R_k(y)$ $W_j(x)$……

其中，任一事务T_i的读操作记为R_i（x），写操作记为W_i（x），x 是一数据项，其粒度在此不考虑；T_i 读出的数据项的集合称之为读出集合，T_i 写入的数据项称之为写入集合，事务的读出和写入集合可以是相交的

■事务中操作执行的序列称之为调度；事务 T_i 与 T_j 可以串行执行，也可并行执行。保证事务的并发操作正确执行的机制称为并发控制，即必须是可串行化调度

6.2 分布式锁技术

6.2.1 集中式并发控制中的锁技术

所谓“封锁”就是事务T在对某个数据操作之前，先向系统发出请求，对其加锁。加锁后事务T就对该数据有了一定的控制，在事务T释放它的锁之前，其他事务不能更新此数据对象。

6.2.2 锁模型概念

■原则

事务对任何数据的操作必须先申请该数据项的锁，只有申请到锁以后，即加锁成功之后，才可以对数据项进行操作。操作完之后，要释放已经申请的锁。通过锁的共享及排斥的特性，实现事务的可串行化调度

■锁类型

- ✓读锁：对数据项进行读操作时加的锁，共享锁
- ✓写锁：对数据项进行写入操作时加的锁，排他锁

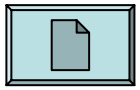
■原理

- ✓可共享时，事务并行执行，排他时，事务串行执行

■两段锁协议

- ✓任何事务对数据项的操作先加锁，加锁的方法是事务中的全部加锁操作在第一个解锁操作的前面完成

6.2.3 分布式两段锁协议



6.2.3.1 2PL协议

- **在集中式数据库系统中**
  - 2PL协议是并发控制算法中的重要算法之一
  - 并发执行的多个事务中，事务对数据进行操作以前要进行加锁，且每个事务中的所有加锁操作，在第一个解锁操作以前执行
  - 每个事务中的加锁操作和解锁操作分布在两个部分中，所以称此协议为2PL协议
- **在分布式数据库系统中**
  - 如果全部的分布式事务均以2PL协议加锁，则系统各场地上的局部调度是可串行化的
  - 换言之，如果分布事务采用2PL协议加锁，那么它在不同场地（站点）的全部子事务也是2PL协议加锁的。对局部调度而言，2PL协议是正确的并发控制方法，所以每个局部场地的子事务是可串行化调度

6.2.3.2 2PL协议的正确性

2PL协议是一种正确的分布式并发控制方法

假设对一个给定的调度E，不存在定理5.2所要求的总的调度次序。这种情况下肯定存在几对冲突操作，因此在E的某些调度表中存在：

$O_1(x)<O_2(x)$

$O_2(y)<O_3(y)$

……

$O_{n-1}(v)<O_n(v)$

$O_n(z)<O_1(z)$

否则它一定有满足定理5.2的总调度次序。

如果事务$T_1,T_2,...T_n$是2PL协议加锁的，上述情况不可能发生

6.2.3.2 2PL协议的正确性

命题：如果事务T_1，T_2，…,T_n，是2PL协议加锁的，上述情况不可能发生。

证明：考虑上述情况：事务T_1锁住了x, T_2锁住了y，……,T_{n-1}锁住了v，T_n锁住了z。现在每一个事务要去锁住一个已被另一事务锁住的数据项，例如，T_1想锁住z，T_n想锁住v，……, T_2想锁住x。所以，每个事务必须等待其它的事务先释放这些锁；但是，如果这些事务释放锁的话，则违反了2PL协议；如果这些事务不释放锁，这几个事务将进入死锁状态，须由死锁解除算法来中止事务。因此，在任何情况下执行调度E都将不可能发生。所以对全局事务而言，除要求局部调度满足可串行化外，全局调度也应满足某一总的次序。

6.2.3.3 2PL协议的性质

2PL协议保证了全局事务执行的可串行性，但它并不允许产生全部的可串行的执行。换言之，2PL协议的要求比可串行性的要求还严格，某些事务在采用2PL协议时可能被迫等待比可串行性条件所要求的更长的时间

例6.1 考虑以下两个事务T1，T2，它们的操作序列如下（设从一个场地读出x，减少它，在另一场地读出y，增加它，然后回写）：

T1：R1（x）W1（x）R1（y）W1（y）

T2：R2（x）W2（x）R2（y）W2（y）

设两个事务几乎被同时激活，2PL协议保证它们的可串行性，所以有一个总的次序。假设T1＜T2，把T1，T2分解执行得到以下两个局部场地的调度：

S1：R1（x）W1（x）R2（x）W2（x）

S2：R1（y）W1（y）R2（y）W2（y）

6.2.3.3 2PL协议的性质

■若允许两个场地上操作的最大并发程度，则**S1**中的**R2**（**x**）**W2**（**x**）和**S2**中的 **R1**（**y**）**W1**（**y**）可以并发执行，这是可串行性所允许的并发执行。然而这样的并发执行在**2PL**协议中是不允许，因为**T1**直到获取对**y**的锁之前是不会释放对**x**的锁的，而且如考虑两段提交协议（**2PC**），事务只有在提交时才释放锁，则事务**T2**被迫等待的时间将大于纯可串行性条件所要求的时间

■**2PL**协议虽然比可串行性条件更严格，降低了并发执行的程度，然而这种要求对保证分布式事务的正确调度是必不可少的。因为如果简单地取消**2PL**，不仅上例中的可串行化的调度是可执行的，而且是一些不可串行化的调度也是可以执行的，从而无法保证正确的并发调度

例如：例**6.I**中增加一个事务**T3**，假设**T1**＜**T3**，从一个场地读出**y**减少它，回写，然后在另一场地读出**x**增加它，回写，其操作序列为：

T3：R3（y）W3（y）R3（x）W3（x）

则下面的非可串行化的操作也是可执行的：

S1：R1（x）W1（x）R3（x）W3（x）

S2：R3（y）W3（y）R1（y）W1（y）

6.2.3.3 2PL协议的性质

- 分布式2PL协议是一种正确的分布式并发控制方法
- 分布式2PL协议简单归纳如下

对于几个分布式事务T_1，T_2，…，T_n，在m个场地上的执行，其全局调度E可以由一组局部调度序列S_1，S_2，…，S_m来描述，每一个局部调度S_k（k=1，…，m）均遵守2PL协议，且对于有冲突操作事务T_i，T_j的冲突操作对 O_i，O_j，如果有 $T_i < T_j$时，也满足 $O_i < O_j$，则全局调度 E满足分布式2PL协议，即是可串行化的

6.2.4 分布式死锁及其处理

6.2.4.1 超时法解决死锁

- **超时法是解决死锁问题的最简单的方法**

原则

当事务申请对某数据项加锁时，或在一个一定长的时间内未申请到锁，则认定系统处于死锁状态，进入死锁处理过程，夭折该事务，释放由其占用的资源

优点

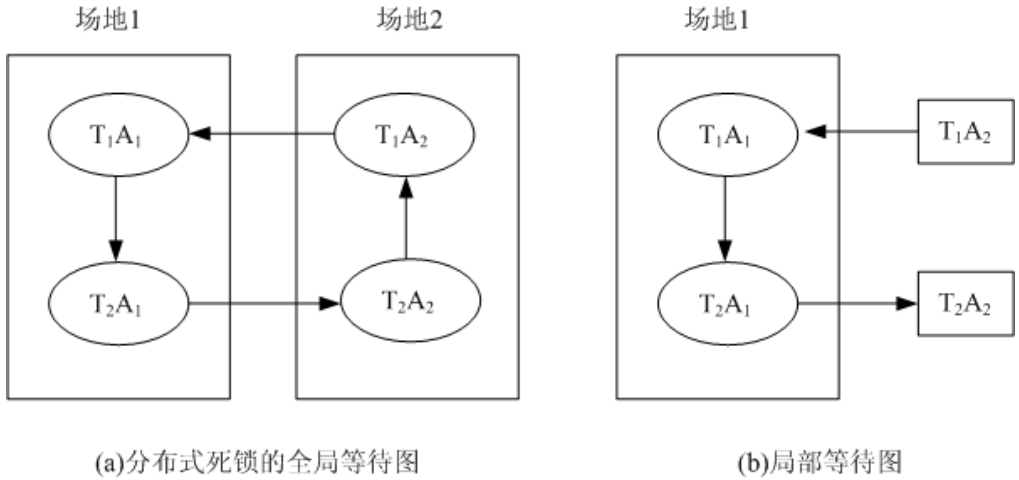
没有额外的控制报文的传送，也不像基于时间印的死锁检测算法要经常夭折事务

关键

在于如何确定超时应有多长的等待时间

6.2.4.2 死锁等待图

等待图：节点表示每个事务，当事务T_1对数据项X要获得锁时，X已被另一事务T_2锁住，这时T_1必须等待T_2释放对X的锁，在图上即有从T_1指向T_2的边。出现死锁，当且仅当图中的边有回路。



(a)分布式死锁的全局等待图　　(b)局部等待图

图6.1 全局和局部等待图

•当系统中出现死锁后，要解除死锁，就要消去等待图中的回路，一般的方法是：中止或重新启动一个或几个事务，从而消除等待回路，让其他事务可以继续执行。选择中止事务的原则是：
（1）中止最年轻的事务；（2）中止占用资源最少、代价最小的事务；（3）中止预期完成时间最长的事务；

6.2.4.3 集中式或分层控制检测死锁

（1）集中式死锁检测

- 选择一个场地来运行集中式的死锁检测程序
- 从系统中的其他场地接收**LWFG（Local Wait-for-Graph）**，产生全局等待图，检测回路，如有回路，发现死锁并进行死锁解除处理；否则，没有死锁

6.2.4.3 集中式或分层控制检测死锁

(1) 集中式死锁检测

- 从LWFG导出的潜在的死锁回路

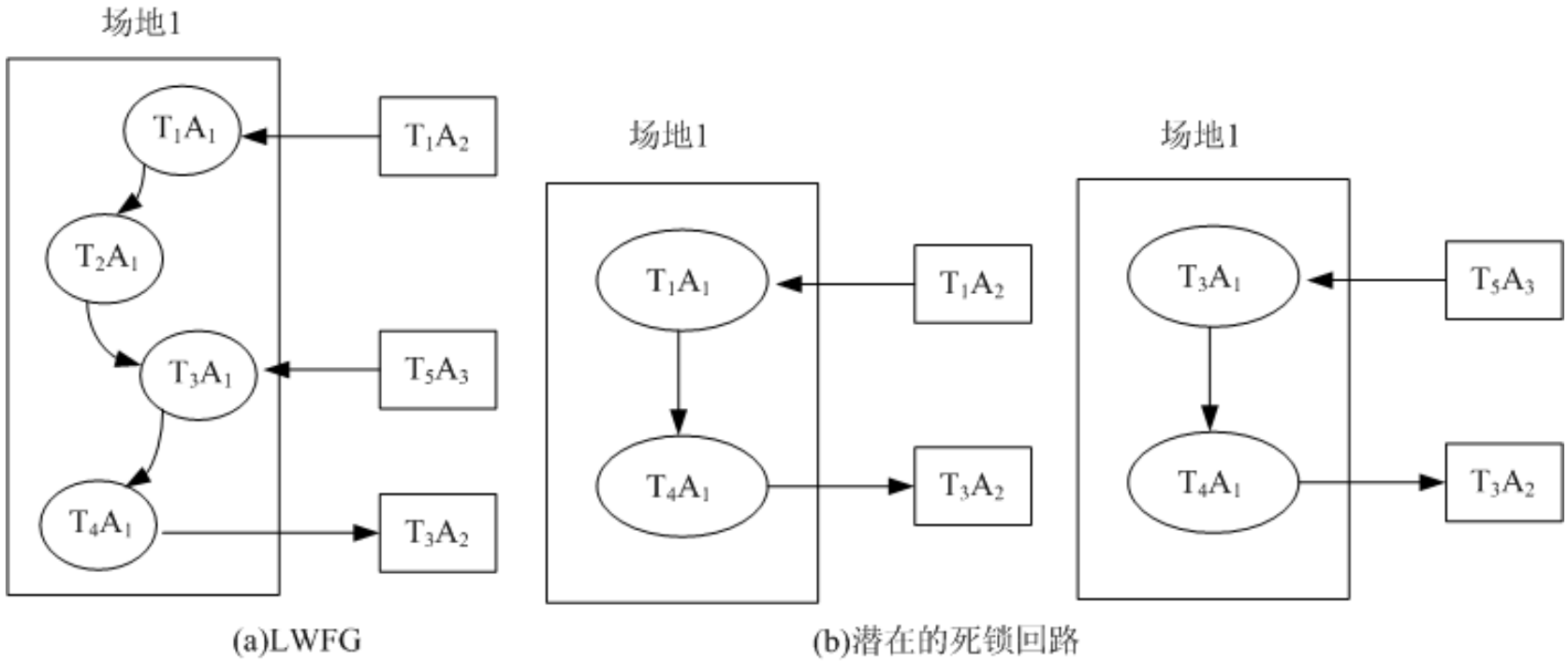


图6.2 从LWFG导出的潜在的死锁回路

缺点：容易受运行集中式死锁检测程序的场地故障的影响；需要大量的通讯费用

6.2.4.3 集中式或分层控制检测死锁

- （2）分层控制方法

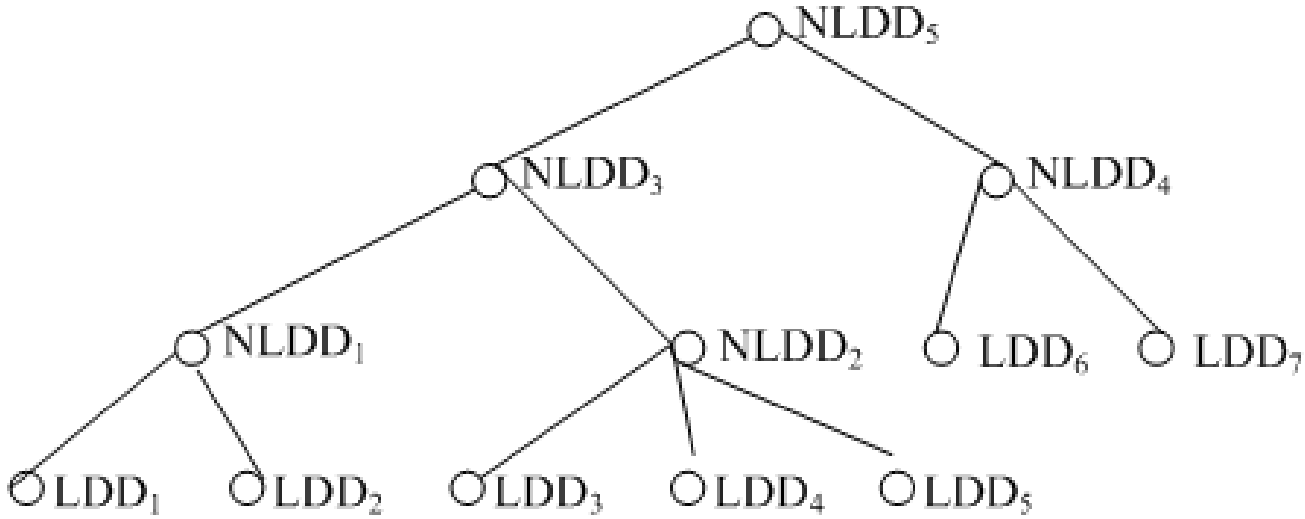


图6.3 分层控制死锁检测树

分层控制方法的运行性能与层次的选择有关，选择时应反映网络的拓扑及数据库应用的要求，即网上地理位置靠近及访问请求多数只集中在几个场地上时，应考虑把这几个场地分在一组，由一个NLDD(集中式全局死锁检测程序)实现该组内的死锁检测。NLDD的个数应当适中，太多会增加运行费用，太少又会退化成和集中式死锁检测方法一样。

6.2.4.4 分布式死锁检测

- 在分布式死锁检测机制中，没有局部和全局死锁检测程序的区别，每个场地都具有检测死锁的责任
- 算法的基础是寻找有无死锁回路，因此需要在各场地间传送潜在的死锁回路的信息，由各场地在自己的LWFG上寻找死锁回路

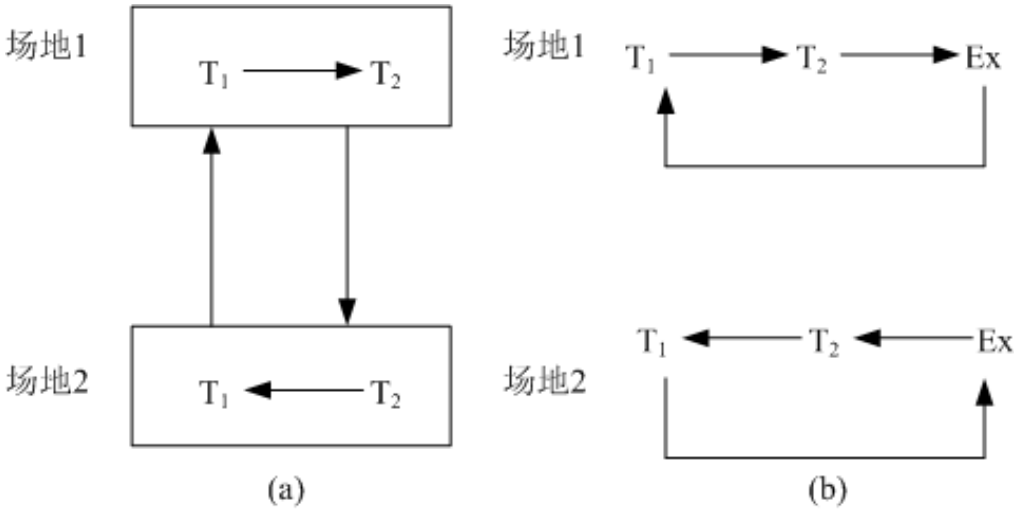


图6.4 分布式死锁检测算法

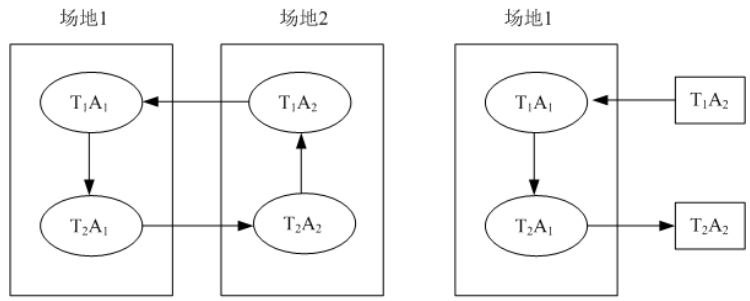


图6.1 全局和局部等待图

6.2.4.5 分布式死锁的预防

■基本思想

在可能出现死锁的情况下，先中止或重新启动若干事务，从而避免死锁的发生

■预防过程

如事务T_1申请一资源，而它被另一事务T_2占有，这时进行一次预防性测试，若测试表明可能出现死锁的危险，则不让T_1进入等待状态

■具体方法

- ✓ 采用时间印的非强制性死锁预防算法
- ✓ 采用时间印的强制性死锁预防算法

6.2.4.5 分布式死锁的预防

- **采用时间印的非强制性死锁预防算法**
  - ✓ 如果T_i申请对T_j已加锁的数据加锁，则只有当T_i比T_j老时才允许T_i等待。如果T_i比T_j年轻，则T_i被终止且以同一时间印重新启动
- **采用时间印的强制性死锁预防算法**
  - ✓ 如果T_i申请对T_j已加锁的数据加锁，只有当T_i比T_j年轻时才允许T_i等待，否则，中止T_j且T_i得到申请的锁

6.3 多副本并发控制

6.3.1 概述

在分布式数据库中,为了提高系统可用性,可靠性和存取效率,经常存放多个数据项的副本.当某一或者几个场地发生故障时可以通过读别的场地上的副本数据来保证正常数据的处理,同时副本可以减少通讯的花销和提高系统效率.

基于锁机制的多副本并发控制算法中,锁机制要考虑:

- 对各场地数据项加锁,而不是对全局数据项
- 对各副本的加锁方法

6.3.2 读一写全法

读一写全法:适合查询操作多的系统

■特点

当事务对某一数据项加锁时,若加读锁,则只对其多副本中任何一个副本加锁.若加写锁,则要对该数据所有副本加锁

■控制报文

对于读锁来说,只要向选中的副本所在场地发送控制报文申请锁,然后等待回答.而写锁则要向**DDB**中所有拥有被锁对象的副本场地发送报文申请锁.这样,写锁通讯费用较大

6.3.3 多数法

多数法:适合更新操作多的系统

■**特点**

只有在获得多于副本数一半以上的锁(读,写),才可以获得对数据项的加锁

■**报文传送**

对于写锁,向至少(n+1)/2个场地副本发加锁请求.如果收到至少(n+1)/2个场地的批准,向n个副本发送新值

读锁类似

6.3.4 主副本法

主副本法

■特点

对某一数据项加锁时,只要对一个主副本加锁,就可以得到该数据项的锁。一般,主副本选择在用户提出锁某项数据最多的场地

■优点

报文传送次数很少

■缺点

并发度低

6.3.5 中心场地法

中心场地法

■特点

在DDB中由一个专门的场地来管理加锁请求.对于读锁,由中心场地向一副本发报文,将数据传给锁申请地.对于写锁,锁申请地把数据发到中心场地,再由中心场地向所有数据副本发送数据项新值

■缺点

很容易形成瓶颈.且中心场地瘫痪时系统将瘫痪

6.3.6 主副本令牌法

主副本令牌法

特点

读/写令牌在网络中不断传递.一场地只有拥有令牌,才能得到相应的锁

过程

1)始发场地向所有场地发送报文请求与令牌

2) 其他场地接收到报文后

- ✓a 没有该数据项写令牌, 或者有,但准备释放
- ✓b正在使用令牌

3)当所有场地回答a时,始发场地得到写令牌并命令其他场地释放所有该数据项令牌.否则,始发场地得不到令牌,处于等待

6.4 时间印方法

6.4.1 时间印模型

对每个事务赋予一个唯一的时间印,事务的执行等价于按时间印的先后次序串行执行

■时间印

事务在某一站点激活时，有系统赋予的全系统唯一的能够识别事务激活的先后次序的一个标识

■并发机制

只有当数据项x上由一年长的事务写入后，才允许另一年轻的事务对x进行读写.否则拒绝操作，并重新启动该事务.即一个事务只能读写在它之前的事务所写入的数据，而不能读写在它之后的事务所写入的数据.对年长事务读写年轻事务写入的数据,则要重新启动年长事务,赋予一个新的时间印,直至更年轻

■方法

基本时间印方法,保守时间印方法

6.4.2 基本时间印方法

规则：

1) 每个事务在激活时得到系统赋予的时间印

2) 事务执行的每个读写操作都具有该事务的时间印

3) 对每个数据项**x**,记录了最大时间印的读操作和写操作,称为**x**的读时间印**RTM(x)**和写时间印**WTM(x)**

4) 令**TS**为对**x**的读操作时间印,如果**TS<WTM(x)**,则拒绝该操作并重启事务.否则执行读操作把**x**的读时间印改为**max(RTM(x),TS)**

5) 令**TS**为对**x**的写操作时间印,如果**TS<RTM(x)**或**TS<WTM(x)**,则拒绝该操作并重启事务.否则执行写操作把**WTM(x)**改为**TS**

缺点: 事务重新启动次数较多

优点: 无死锁问题

6.4.3 保守时间印方法

特点

不会拒绝任何操作,因此不会重启事务

处理方法

出现冲突时,把年轻的操作缓冲起来,等待年老的操作执行完后再执行

规则

1)每个事务只在一个场地上执行,只能向远程进程发读写请求.

2)一个场地按时间印次序接受另一场地读写请求.(串行)

3)若一场地i从网络每个其他场地接受至少一个缓冲读/写操作,用以下方法处理:

- 对于到达本地的读操作R,如果有个写操作W被缓冲，且TS(R)>TS(W),则R被送入等待队列，直到写操作全部执行才执行R
- 对于写操作W,如果本地有某个读操作被缓冲，且TS(W)>TS(R)，或者一个写W'被缓冲，且TS(W)>TS(W'),则W进入等待队列，直到缓冲的操作执行完毕才执行W

6.5 “乐观”并发控制方法

乐观方法总是让一事务执行完.其写操作的结果暂时保存,事务结束后,由一个专门检测确认过程,检验事务的执行是否可串行.通过检测的话,就把写操作结果永久化.否则重启事务

■ **事务的执行过程分三个阶段:**

1) **读出阶段**:读出所需的数据项,进行数据处理.决定回写结果的值,把值保留在暂存空间

2) **检测阶段**:检测更新操作是否引起数据库的不一致

3) **写入阶段**:把结果写回数据库

■ **基于数据项的事务时间印的检测算法**

在执行期间把事务的更新全部记入一更新表,检测阶段检测对其他场地的所有更新操作是否可行

结论：一般来说，乐观方法对冲突较少的系统比较方便；对于冲突较多的系统，使用乐观方法是不明智的

第7章 分布式事务恢复

7.1 故障模型

7.1.1 局部场地数据库的故障模型

■不丢失信息的故障

这一类故障主要由于命令无法执行引起事务中止，而不会对存储介质上的数据产生不正确的操作，存储介质中的数据全部是正确的(如进入死循环引起超、除操作溢出等)。这类故障很易恢复，重新启动事务即可

■丢失主存信息的故障

由于这类故障的出现，主存中的数据库处于一种不正确状态，即部分或全部数据丢失或出错。但在辅存上的数据库仍处于一正确的状态，如事务未正确提交或中止，破坏了事务的原子性。这类故障可以利用集中式恢复机制对其进行恢复

■丢失辅存中信息的故障

这种故障表现在辅存或永久存储的数据库信息的丢失，如磁盘或磁带中存储的对于数据库系统信息的丢失。这类故障对于数据库系统是致命的故障。当发生这种故障时一般是不可恢复的，只能重建

7.1.2 分布式数据库系统的通讯故障模型

为了对通讯故障进行分类，需要做以下假设，即当把一报文从场地x发向场地y时，要求通讯网络具有以下特性：

（1）在一段时间延迟后，x收到一个肯定的应答信息

（2）对于x和y间的多个报文来说，报文以正确的次序发送到场地y

（3）报文的内容是正确的

有了以上要求后，通讯故障可能是以下多种情况：

- ✓x收不到回答
- ✓x发送报文的次序不对
- ✓报文内容不对
- ✓x发不出通讯请求
- ✓未发请求收到应答

多数网络是可以自行解决上述的故障而不必由数据库系统来解决，所以可以做如下假设：

（1）如报文已从x场地发送到了y场地，则报文是正确的，其发送次序也是正确的

（2）如x场地收到一肯定的回答，则报文已经正确地送到y场地

7.1.2 分布式数据库系统的通讯故障模型

■通讯故障可以分为两种

- ✓报文丢失：是指传送过程中报文的丢失导致数据不正确
- ✓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

■故障的恢复处理难度分析

按照恢复处理难度从小到大排列为

- ✓ 1：场地故障
- ✓ 2：场地故障和丢失报文，但无网络分割
- ✓ 3：场地故障，丢失报文及网络分割

■故障处理方法分类

✓第一类恢复处理只处理局部场地的故障，这主要基于集中式数据库系统的恢复策略

✓第二类处理报文丢失和场地故障

✓第三类可处理除第二类故障外的网络分割故障。当恢复算法可以处理同时发生的多个故障时，我们称之为N度恢复算法。N越大，恢复的算法的能力越强

✓最糟糕的是灾难性故障，超出了系统恢复能力

7.2 分布式事务的两段提交协议

2PC协议是一种故障恢复方法，在系统运行日志没有丢失的情况下，2PC协议对任何故障均有一定的恢复能力。在2PC算法中，分布式事务要指定一个**协调者**的进程，由该协调者负责分布事务的提交或中止。一般该进程由分布式事务的始发场地上的**代理者**进程担任，其它场地上的代理者进程称之为分布事务的**参与者**，每个参与者负责其局部场地上的写操作，并向协调者进程提出提交或中止事务的建议

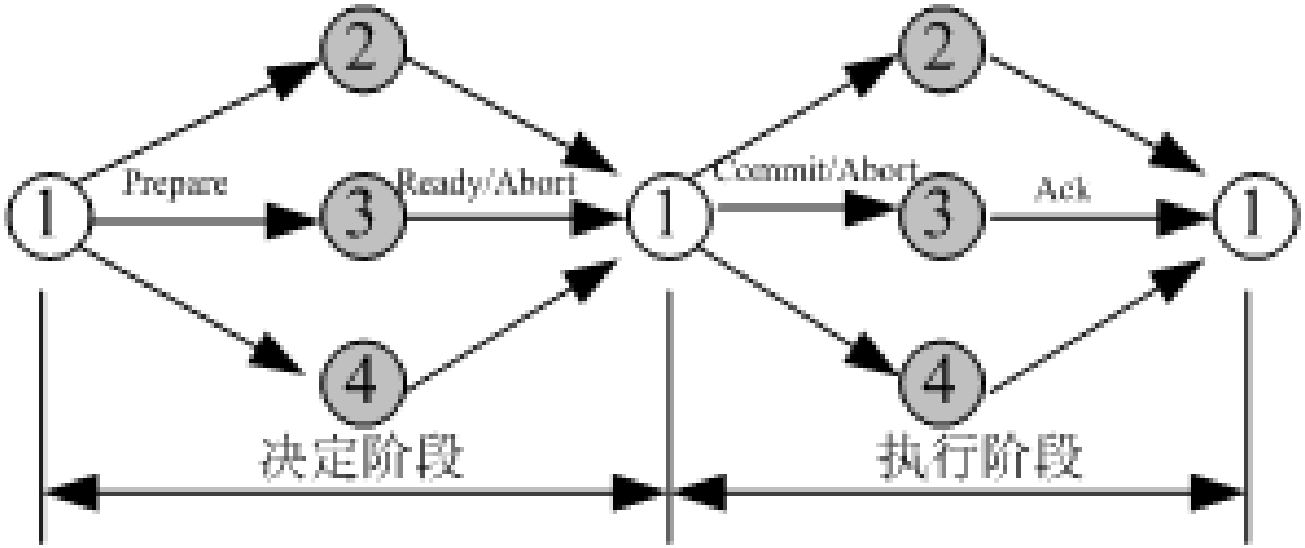
7.2.1 2PC协议

■基本思想

为全部参与者做出提交或中止全部子事务的唯一决定。如果有一个参与者不能提交其子事务，全部局部子事务全部中止。分布式事务提交，当且仅当全部子事务均提交

■2PC协议由两个阶段组成

- ✓**决定阶段**：做出提交或中止全部子事务的决定
- ✓**执行阶段**：实现第一阶段的决定



7.2.1 2PC协议

第一阶段

- 协调者要求所有的参与者均进入准备提交阶段，发送“准备提交”命令，若参与者准备好了并愿意提交，就回答“准备提交”，否则回答“夭折”

- 在发出准备提交命令前，协调者要在日志中记入“准备提交”记录及所有参与者的子事务标识符，并进入等待回答状态且开始计时

- 当一个参与者回答“准备提交”应答时，它保证甚至在其它场地故障时也能提交子事务。因此，在子事务所在场地的日志中要记录以下内容：

  - ✓ (1)本场地提交子事务所需的全部信息，即该子事务的全部运行记录记入日志中
  - ✓ (2)该子事务准备就绪的事实

以上两点保证该子事务的状态已不受其它场地故障的影响

7.2.1 2PC协议

第二阶段

- 协调者从各参与者接收消息，若超时或收到一个“夭折”回答，则决定“夭折”事务，否则收到所有的“准备提交”回答，决定提交事务

- 当协调者做出决定后，在日志中记入相应的记录，将“全部提交”或“全部夭折”写入运行记录。这表明，不论发生什么情况，分布式事务最终将“提交”或“夭折”，然后协调者向所有的参与者发送“提交”或“夭折”命令

- 所有参与者根据协调者的命令在本场地日志中写入“提交”或“中止”记录。从此时起，局部恢复程序保证不丢失该子事务的实施，执行“提交”或“中止”命令，并向协调者发“执行”信息

- 协调者从所有参与者收到执行报文，在日志中记入“完成”记录，此时分布事务的2PC完成。在恢复算法的下一个检查点之后，日志中关于此事务的全部记录可以离线

7.2.1 2PC协议

```
协调者   在日志中写入“预提交”命令
         发送“预提交”命令给所有参与者且开始计时
参与者   等待“预提交”命令
         IF   参与者可以提交   THEN
              BEGIN
              在日志中写入子事务的记录
              在日志中写入“准备提交”
              向协调者发“准备提交”信息
              END
         ELSE   BEGIN
                在日志中写入“夭折”
                向协调者发“夭折”信息
         END
```

（待续）

图7.1 2PC协议（a）

7.2.1 2PC协议

（续）

协调者　等待接收所有参与者的应答信息并检查限时

IF　超时或收到至少一个“夭折”　THEN

BEGIN

在日志中记入“全部夭折”

向所有的子事务发“夭折”命令

END

ELSE BEGIN〈收到全部“准备提交”〉

在日志中记入“全局提交”

向所有的参与者发“提交”命令

参与者　等待协调者的命令：

根据命令在日志中记入“夭折”或“提交”

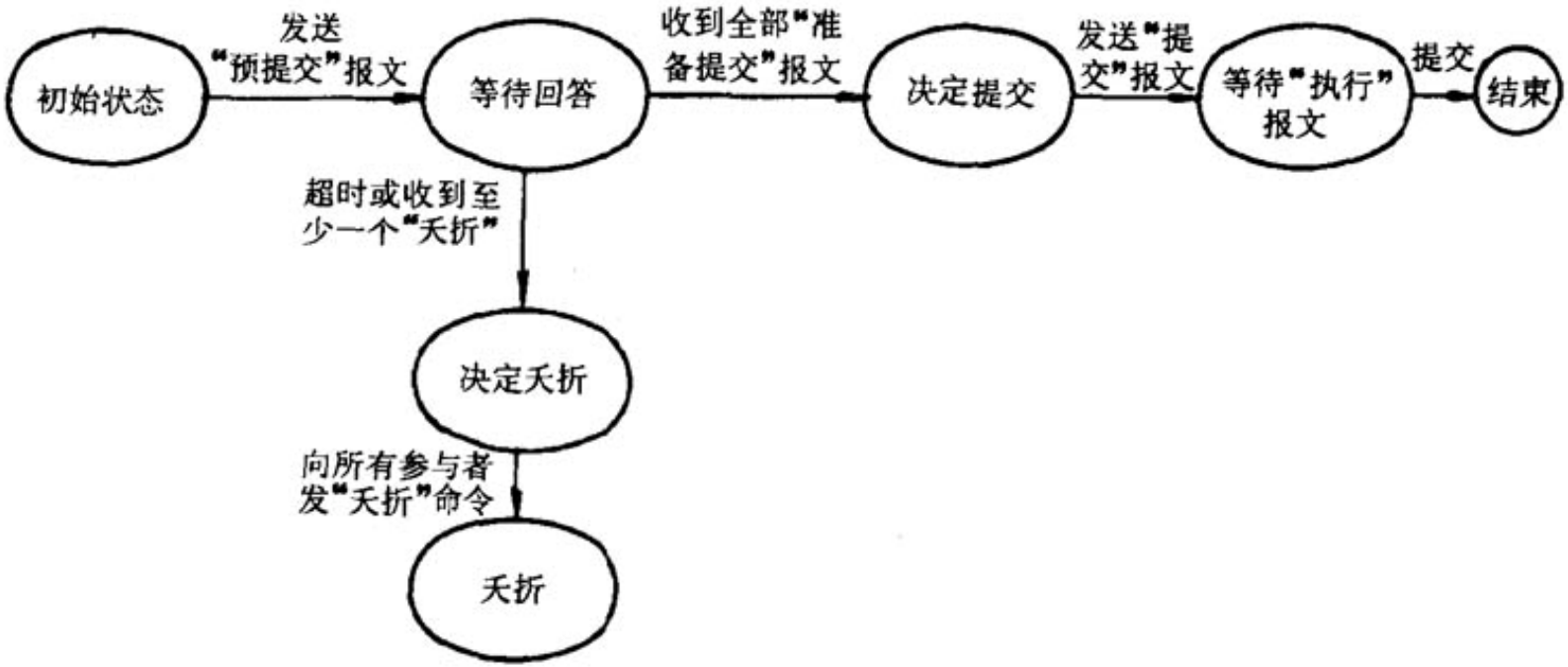
向协调者发送“执行”信息，然后执行“提交”或“夭折”命令

协调者　等待接收所有场地的“执行”信息

在日志中记入“完成”

图7.1 2PC协议（b）

7.2.1 2PC协议



(b)协调者

图7.2 2PC协议工作过程示意图

7.2.1 2PC协议

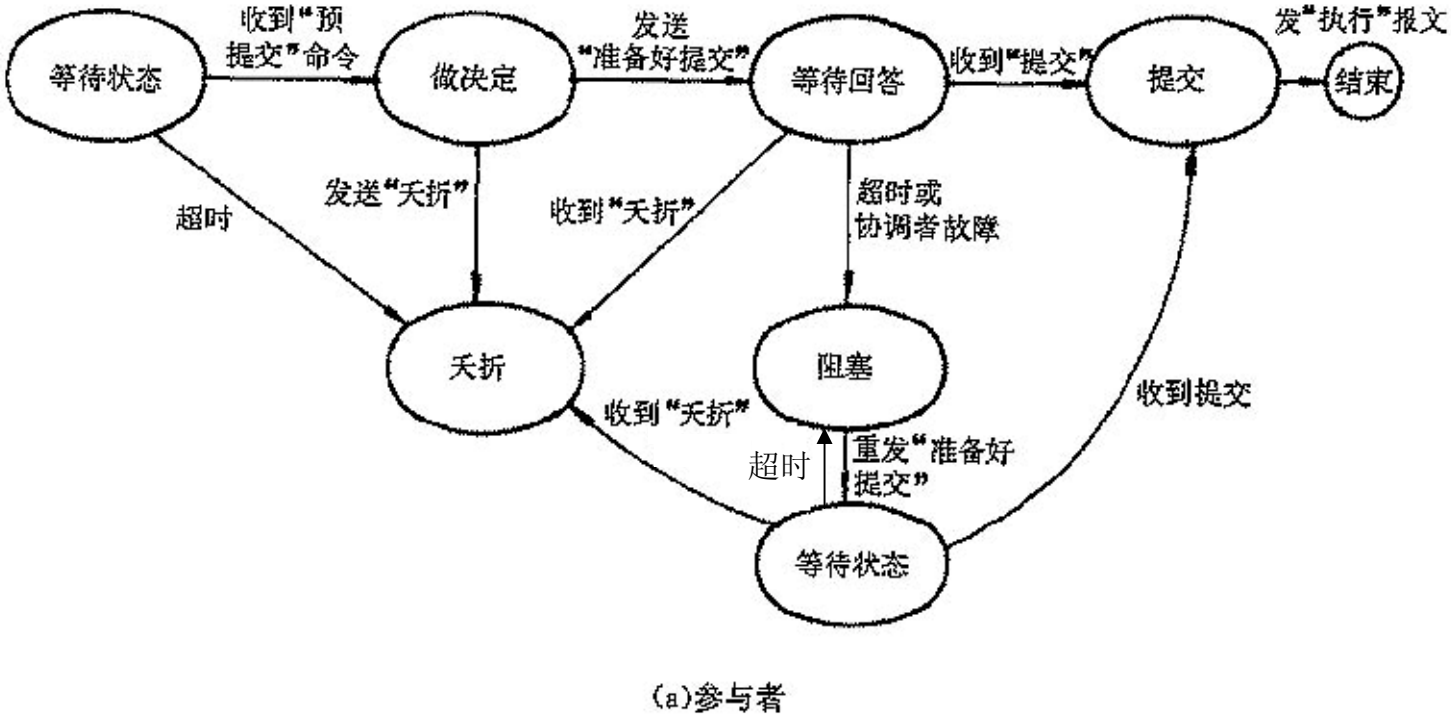


图7.2 2PC协议工作过程示意图

7.2.2 2PC协议对故障的处理

7.2.2.1 场地故障

- **(1)一个参与者在写入“准备提交”前发生故障**

该参与者无法向协调者发回答信息，因此，当协调者等待超时后，将决定中止事务。当参与者故障恢复后，重启动过程无需收集其他场地的信息，即可中止事务

- **(2)参与者进程在写入“准备提交”后发生故障**

其他的参与者可以正常地结束该事务“提交”或“夭折”，因为协调者可以根据收到的应答决定“提交”或“夭折”。因此，故障恢复后，重启动过程要访问协调者或参与者，从而了解事务已经做出的决定，然后执行相应的操作，即“提交”或“中止”。这里需要假设，在日志中写入“准备提交”记录和发送“准备提交”信息给协调者，这两个动作具有原子性，要么都执行，要么都不执行

7.2.2.1 场地故障

- **(3)协调者在日志中写入“预提交”记录后，写入“全部提交”或“全部夭折”前发生故障**

  已发出“准备提交”信息的参与者等待协调者恢复。协调者的重启动过程从头恢复提交协议，从“预提交”记录中读出参与者的标识，重发“预提交”报文给所有参与者，重新执行提交过程

- **(4)协调者在写入“全部提交”或“全部夭折”记录以后，在写入“完成”记录以前发生故障**

  协调者恢复时，必须给所有参与者重发其决定，未收到信息的参与者不得不等待协调者的回复。和上一个故障一样，参与者不应因收到两次一样的命令而受影响

- **(5)协调者在日志中写入“完成”记录后发生故障**

  在这种情况下，事务已经结束，不需要恢复处理

7.2.2.2 报文丢失

■(1)来自参与者的回答报文(“准备提交”或“夭折”)至少丢失了一个

在这种情况下，协调者将等待回答而超时，整个事务被夭折

■(2)丢失“预提交”报文

由于至少一个参与者收不到“预提交”命令，因此参与者处于等待状态，而协调者也等待参与者的回答，所以协调者会因为等待超时而夭折事务

■(3)丢失“提交”或“夭折”报文

这种情况下参与者处于等待协调者命令的状态下，当未收到命令时会因等待而超时，这时向协调者发请求重发该命令的信息

■(4)丢失了“执行”报文

当协调者未收到全部的“执行”报文时，协调者会因等待而超时，这时协调者重发命令报文给参与者，这时参与者必须给予“执行”报文回答，即使此时相应的子事务已不再活动也要重发

7.2.2.3 网络分割

■出现网络分割时，整个网被分为两个组，包含协调者的组称为协调者组，其它的则组成参与者组

■这种情况对于协调者而言，相当于参与者组中的多个参与者同时发生故障，这时协调者可以作出决定，然后把命令发给协调者组中的参与者，因此这些场地上的子事务可以结束。这与场地故障中的(1)与(2)两种情况类似

■对于参与者组而言中的参与者而言，由于无法联系协调者，这时它们认为协调者出现故障。这种情况与场地故障中的(3)与(4)相似

7.2.3 2PC协议的通讯结构

2PC协议按各参与者之间的通讯结构可以分为线性的、集中式的、分层的和全分布式的四种

7.2.3.1 线性的2PC协议

- 线性的2PC协议中，事务的始发场地构造一个线性有序的场地表，该表中第一个元素为协调者场地，接下来依次是第一参与者场地名，第二参与者场地名，…，第n参与者场地名

- 在进行事务的提交过程时，事务的始发场地首先进入“准备提交”状态而后向表中的下一场地参与者发“准备提交”命令

- 若该场地已准备好，并且愿意提交，则由该场地把“准备提交”写入日志，向上一场地发送“准备提交”回答，自己变为当前场地，继续向下一场地发“准备提交”命令；依次类推，直到最后一个场地为止
- 否则，若该场地参与者未准备好，则向其前一场地发送“夭折”回答，当前一场地收到“夭折”回答时，它可以决定中止事务

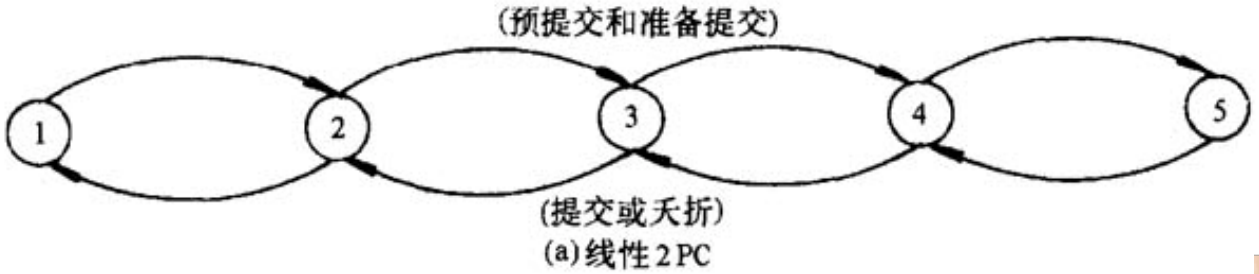


图7.3 2PC通信结构

7.2.3.1 线性的2PC协议

- 当最后一个场地收到“准备提交”命令时，它可以决定前面的场地均进入准备提交状态，这时，它可以根据自己的情况决定事务的提交或终止，此时，最后一个场地的参与者充当了协调者的角色

- 当表尾场地决定提交时，自己先进入提交状态，然后向前一场地发提交命令，前一场地收到提交命令后进入提交状态，完成提交后，向前驱场地发送提交命令，向后续场地发应答报文，发送成功之后，可以结束该子事务，从而进入子事务的完成状态

- 当事务的始发场地收到“提交”命令时，便将“提交”记入日志，发送应答报文给其他后续场地，然后撤销事务进入事务的完成状态

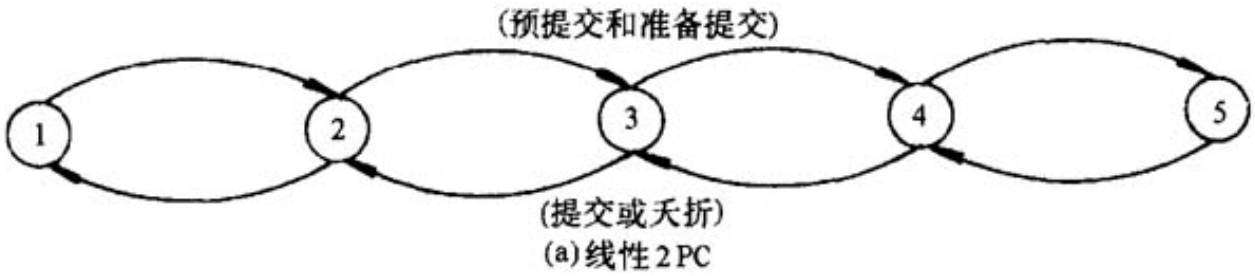
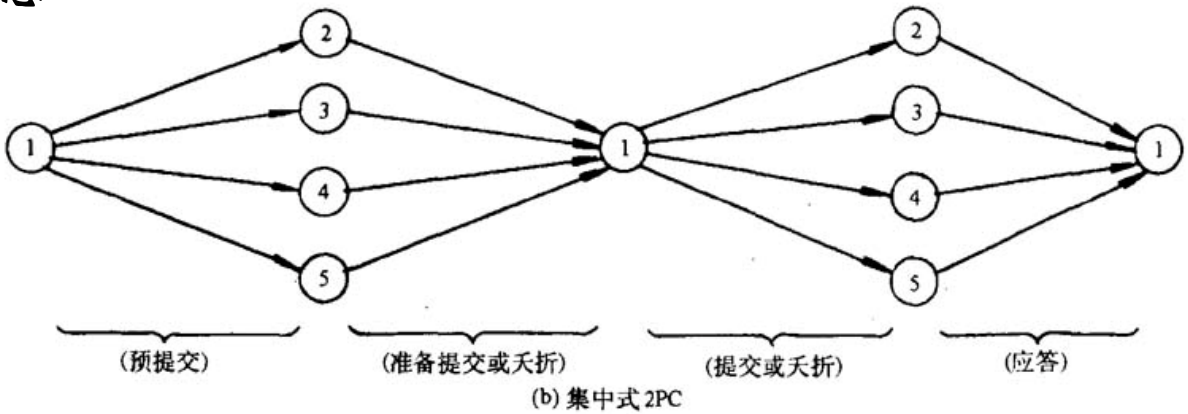


图7.3 2PC通信结构

7.2.3.2 集中式的2PC协议

- 选一个场地作为协调者（一般由事务的始发场地充当），提交的初始化工作由始发场地完成
- 协调者把所有参与者的标识写入日志中，然后，向所有参与者发送“预提交”报文，征求各参与者的意向
- 各参与者收到“预提交”命令后，若准备好提交，则发送赞同应答，并进入就绪状态，否则，发送否决报文
- 协调者只要收到一个拒绝报文，就决定中止，在日志中记入“夭折”记录，并向各参与者发送“夭折”命令，然后，进入“夭折”状态；否则，收到所有赞同应答，决定“提交”，将“提交”记录写入日志，向各参与者发送“提交”命令进入“提交”状态



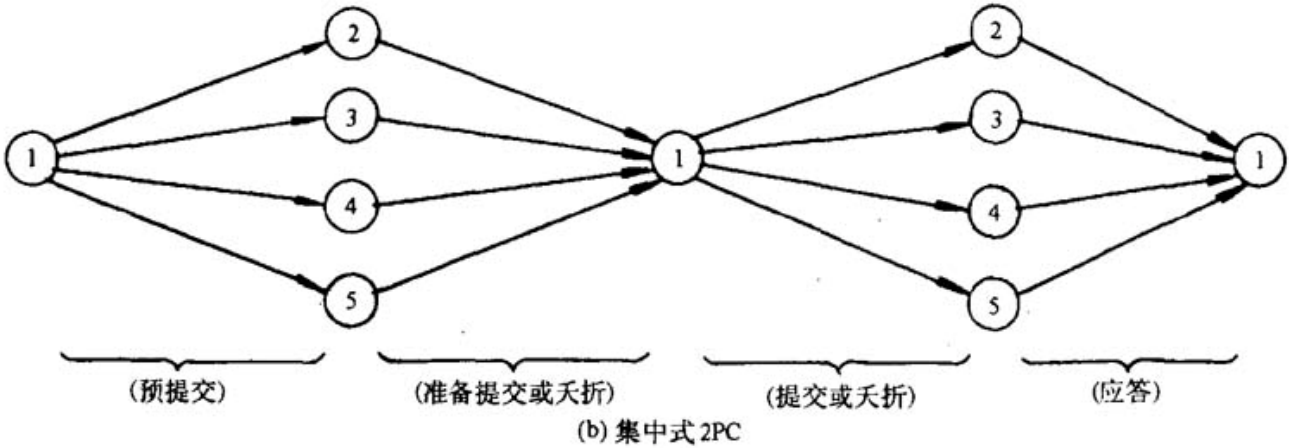
(b) 集中式2PC

图7.3 2PC通信结构

7.2.3.2 集中式的2PC协议

- 处于就绪状态的参与者收到“提交”命令以后，便把“提交”记录写入日志，完成提交后，向协调者发送“提交”应答报文，报文发送成功后，结束子事务，进入子事务完成状态

- 若参与者收到“中止”命令，则处于就绪状态的参与者要恢复子事务，将“中止”记录写入日志，并向协调者发送“中止”应答报文



(b) 集中式2PC

图7.3 2PC通信结构

7.2.3.3 分层的2PC协议

- 分层的2PC协议（又称树状协议）中，协调者所在场地称为树的根
- 参与者场地构成树的中间节点或叶节点
- 在树型结构中，上一层的节点可以向下一层的节点发送信息或收集应答信息
- 中间节点收集了其下一层节点的回答，做出决定，并向上一层节点发送报文，直至根节点
- 根节点收集到应答后，做出决定，再向其下一层节点发送报文，直至事务最后结束

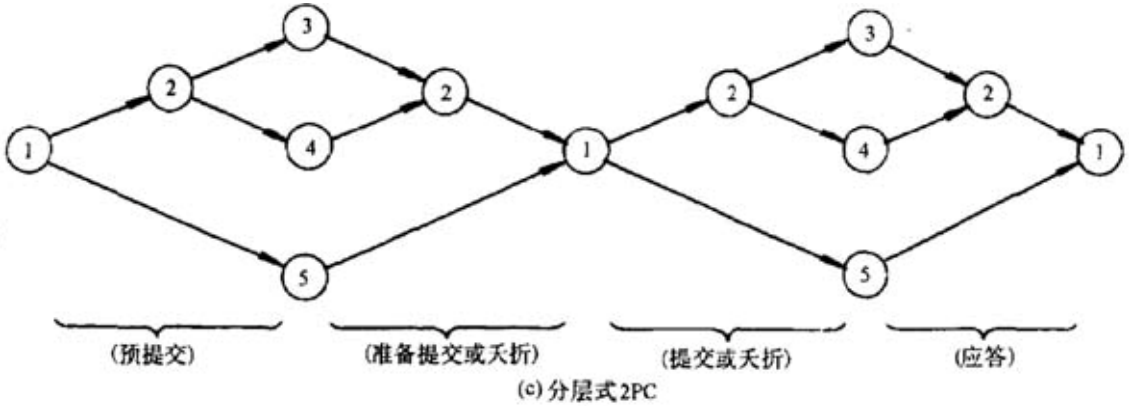


图7.3 2PC通信结构

7.2.3.4 全分布式2PC协议

- 全分布式2PC协议中，事务的所有参与者均可以建立相应的联系，每一场地上的参与者均可以决定事务的提交或中止
- 事务的提交过程如下：
  - ✓事务的始发场地进行提交的初始化工作，然后，向所有参与者广播“预提交”报文，每个参与者收到命令后，做出决定，然后，向所有参与者发送应答报文，每一场地的参与者根据其他参与者的应答做出提交或夭折事务
- 这种通讯协议通讯费用巨大，实际中很少采用

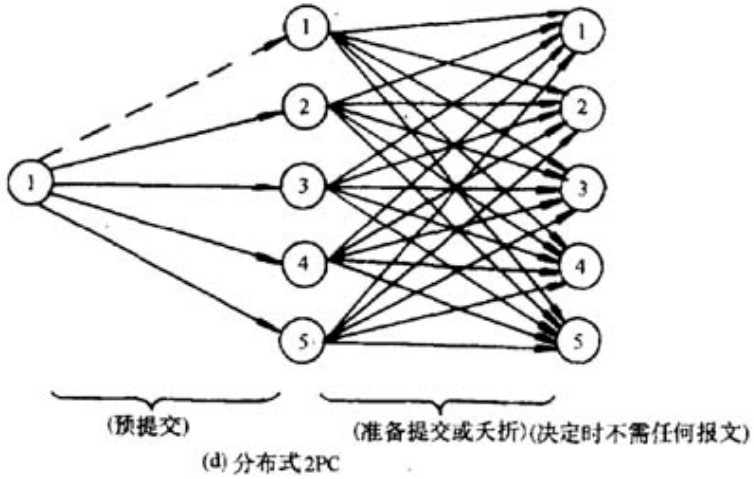


图7.3 2PC通信结构

7.2.3.5 不同通讯结构的比较

四种 2PC 协议中，系统开销评估根据：报文传送的个数、网络延迟

■报文个数

若参与者的个数是N，则报文的个数为：

- ✓线性 2PC 协议为 2N 个
- ✓集中式或分层 2PC 协议为 4N 个
- ✓全分布式的 2PC 协议为：N(N+1) 个

■网络延迟

若报文从一个场地传送到另一个场地的延迟为T，则

- ✓线性 2PC 的延迟为 2NT
- ✓集中式 2PC为 4T
- ✓全分布式 2PC 延迟为 2T
- ✓分层 2PC为4T 到 2NT 之间

结论： 总之，线性2PC的报文传输最少，响应效率最低，对通讯代价较高的系统可以采用。全分布式 2PC报文传输最多，响应效率最高，适合传输代价较小的系统采用。集中式或分层 2PC介于上述两者之间。因此通讯协议的选择要依实际应用环境而定。

7.3 非阻塞提交协议

7.3.1 阻塞

定义：事务的阻塞 所谓事务的阻塞是指一个场地的子事务本来是可以执行结束的(夭折或提交)，然而由于分布式数据库的故障，它必须等待故障恢复以后得到需要的信息后才可以做出决定，而故障情况是不可以预料的，该子事务又占有一些系统资源不能释放，没法继续执行，这时称之为事务进入阻塞状态。

例：图7.2(a)，当参与者等待协调者的回答时，可能因为网络故障或者协调者故障，使之收不到回答信息而出现等待超时，这时事务进入阻塞状态，重发“准备提交”信息要求协调者给予回答，直到网络故障或者协调者恢复给予回答，参与者才做出决定继续执行（或提交或夭折）。若一直收不到回答，则事务一直处于阻塞状态而挂在相应场地上，因此阻塞降低了事务可用性。

7.3.2 提交协议状态图

定义：状态图 用来描述协议执行过程中协调者和参与者状态变化的有向图，图中的节点为协调者和参与者执行过程中的状态，有向边表示状态从一个状态(有向边的起点)变化到另一状态(有向边的终点)。有向边上的描述表示状态变化的条件。

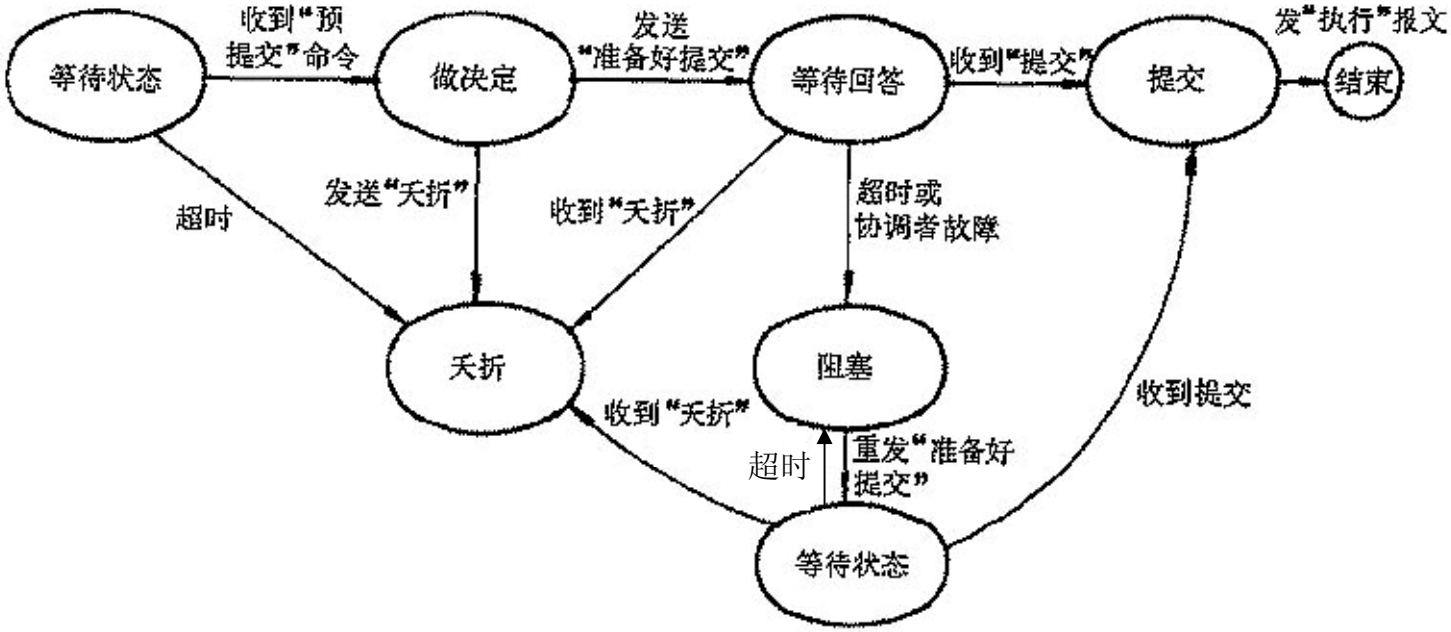


图7.2 2PC协议工作过程示意图

7.3.3 3PC协议

■从2PC到3PC

在2PC协议中，参与者的提交是在它知道了其它所有的参与者均发生了“准备提交”的报文以后进行的。若在2PC中增加一段，使得参与者的提交不仅要等到它知道所有的参与者均发出了“准备提交”的报文，而且还要知道所有参与者的状态（如：它们是处于故障状态，还是已经恢复）以后才执行。这时2PC即变成3PC协议。

7.3.3 3PC协议

■3PC协议的过程

➢ **第一阶段**

协调者向所有的参与者发“开始提交”报文，由每个参与者据自己的情况进行投票，只有收到所有的参与者均赞成提交，才进入第二阶段和第三阶段

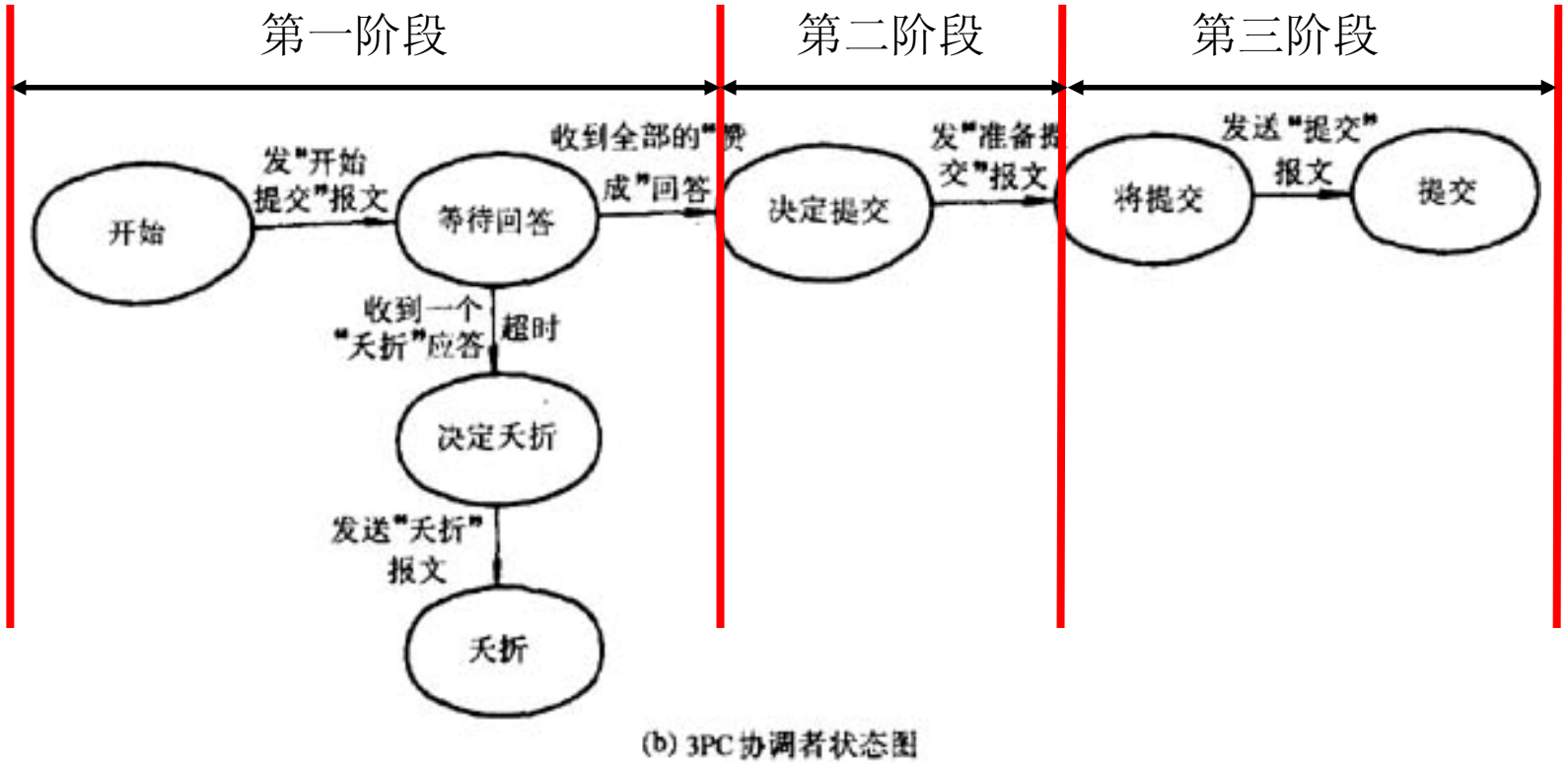
➢ **第二阶段**

协调者向所有的参与者发“准备提交”报文，参与者收到该报文后若已经准备好提交，则回答“准备就绪”报文，否则进行恢复处理

➢ **第三阶段**

当协调者收到所有的参与者“准备就绪”的回答，就向所有的参与者发“提交”报文，此时每个参与者知道其它的参与者将“赞成”提交，并已经进入“准备提交”状态，因此它可以收到“提交”报文后就进行提交

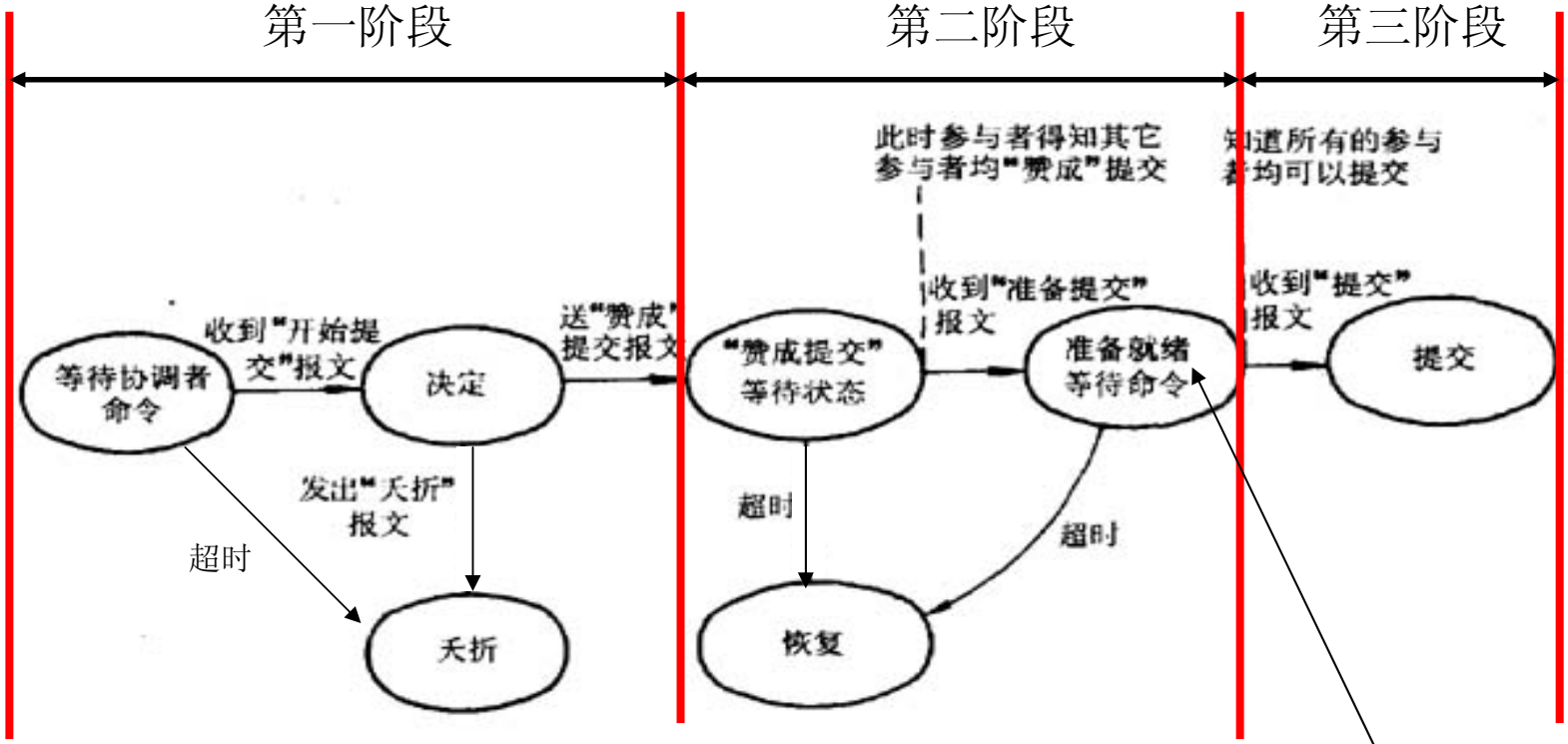
7.3.3 3PC协议



(b) 3PC协调者状态图

图7.4 3PC提交协议状态图

7.3.3 3PC协议



(a) 3PC 参与者状态图

图7.4 3PC提交协议状态图

7.3.3 3PC协议

3PC可以避免阻塞是基于一定的故障模型的。一般来说，在下列故障时，3PC不会进入阻塞状态：

- 只允许出现场地故障而不会出现网络分割的情况
- 场地故障但不影响通讯功能，即它仍能进行正常的收发工作且信息是正确的
- 场地故障使得系统必须进行恢复处理，恢复机制应能知道已发生过故障，且和其它的场地进行通信，了解故障前的情况使参与者恢复到提交状态中去
- 场地无故障时必须在超时以前向曾求助于它的场地发回答报文
- 网络不会丢失报文且场地点传报文的次序和接收报文的次序一样

7.3.4 3PC协议对故障的处理

7.3.4.1 3PC采用超时法处理故障

- 四种情况（参考图）
- **第一种情况**

  协调者没能及时发出“开始提交”报文，导致参与者等待超时，这时参与者决定夭折
- **第二种情况**

  协调者等待参与者投票结果超时，这时协调者决定夭折
- **第三种情况**

  参与者处于“赞成”提交状态，等待“准备提交”命令时出现超时，这时进入恢复处理过程
- **第四种情况**

  参与者处于“准备就绪”状态等待协调者的“提交”命令出现超时，也进入恢复处理

7.3.3 3PC协议

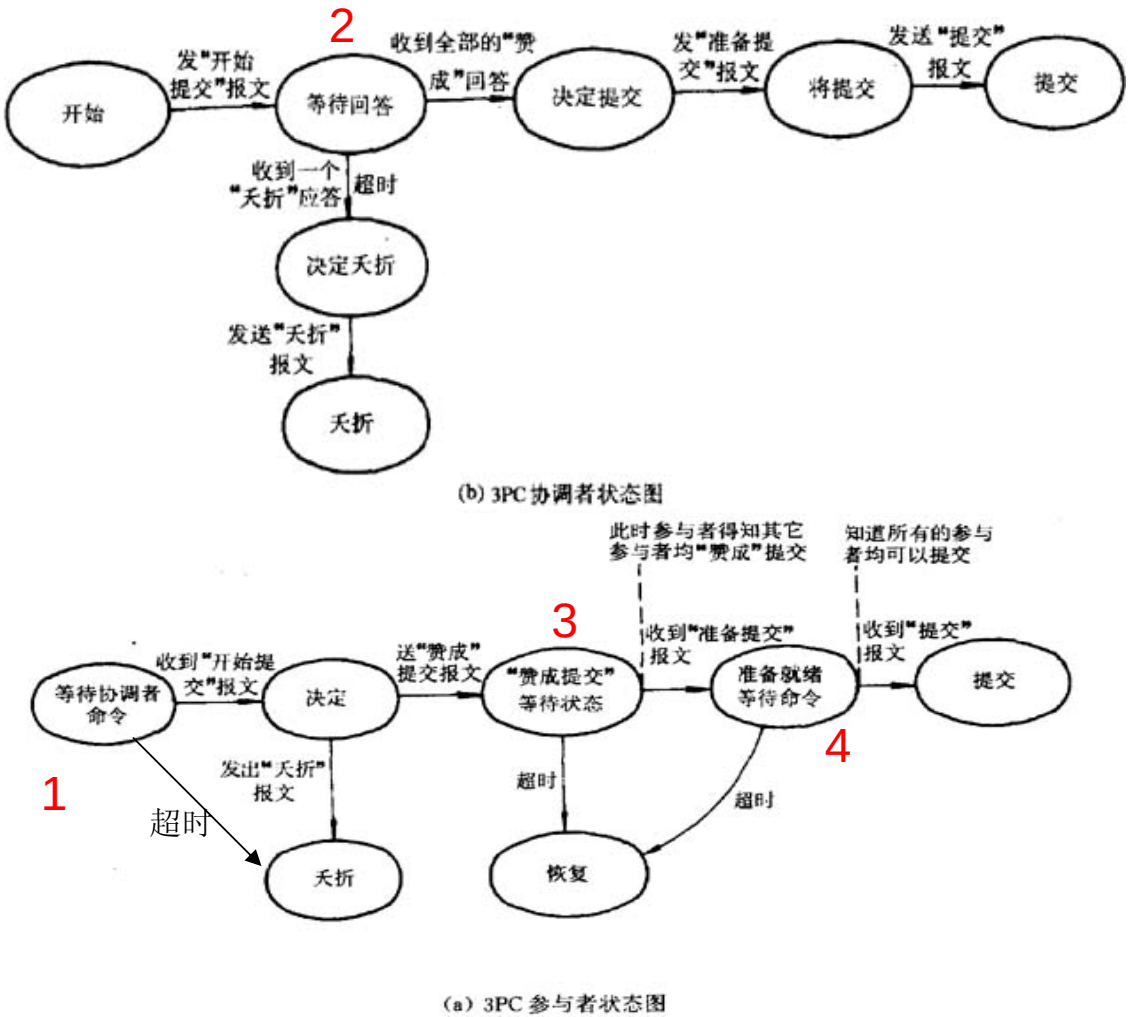


图7.4 3PC提交协议状态图

7.3.4.2 参与者的故障恢复

事务在进入恢复前，有下述两种不可能进入的状态：

- 一个参与者在其它任何一个活动的参与者处于“赞成”提交状态时，不可能进入“提交”状态
  - 因为，当一个参与者处于“赞成”提交状态时，它还没有收到协调者发来的“准备提交”命令，自然，它也无法向协调者发送“准备就绪”报文，协调者也就没有收到所有参与者的“准备就绪”报文，就不会做出“提交”决定，因而也不会向参与者发送“提交”命令，参与者没有收到“提交”命令，是不可能进入“提交”状态的。
- 一个参与者在另一个参与者进入“提交”状态或任何一个参与者都进入了“准备就绪”状态时不能进入“夭折”状态
  - 因为，在上述情况下，肯定是所有参与者都回答了“赞成提交”，因此，所有参与者都会进入到【赞成提交，等待状态】或其以后的状态。这时，根据状态图，是不可能进入夭折状态的。

因此，恢复时，一个参与者可以根据活动事务的状态决定相应的处理

7.3.4.2 参与者的故障恢复

恢复机制就近访问一个活动的参与者

- 如果所有的参与者处于“赞成提交”或“夭折”状态，则肯定没有一个参与者已提交，因此可以通知全部参与者“夭折”

- 如果已经有一个参与者发送了“准备就绪”或收到“提交”，则肯定没有一个参与者被“夭折”，所以可以通知全部参与者提交，在通知“提交”前应先将仍处于“赞成”提交的参与者进入“准备提交”状态，然后再进入“提交”状态。因为，在通知提交前，任何一个参与者均可能再发生故障，所以，应避免其中一个参与者处于“赞成提交”状态而另一个处于“提交”状态

7.3.4.3 协调者的故障恢复

在协调者的恢复方法中，活动的参与者必须设法选一个新的协调者，因为参与者进入恢复处理一般是由等待协调者报文超时引起的。协调者必须是活动的且不能固定，因为固定以后当协调者故障时就无法协调参与者了

- 活动的参与者选择一个新的协调者
- 新的协调者向所有的参与者发报文，使它们进入恢复前的状态；可能是“夭折”、“ 赞成提交”、“准备提交” 或“提交”，并要求参与者根据自己的状态予以回答
- 如果协调者收到任何“ 准备提交” 或“提交”，那么就决定提交；如果收到“夭折” 或“赞成提交”，则夭折事务
- 如果决定夭折，则向每个参与者发夭折命令；如果决定提交，则做以下处理：
  - ✓ A、使处于“赞成提交” 的参与者进入“准备提交”
  - ✓ B、对尚未进入“提交” 的参与者发提交命令

7.4 恢复策略

一、事务的恢复是以运行日志为基础

■运行日志是记录故障前一段时间内系统运行情况的一种文件；在故障发生后，系统的恢复机制根据日志中的内容对系统进行恢复；运行日志记载了下列内容：

- ✓ 事务的标识符
- ✓ 操作对象的标识符
- ✓ 动作的类型
- ✓ 操作对象的新值
- ✓ 操作对象的旧值
- ✓ 事务的执行状态
- ✓ 恢复的辅助信息

■写入日志文件时，采用运行记录和提交写入协议：

- ✓在数据库更新以前，操作对象的旧值已写入日志文件且不丢失
- ✓事务提交以前，有关的运行记录全部写入日志文件且不丢失

二、采用日志对易失存储器的信息进行恢复

■ 恢复过程

当系统出现故障时，恢复机制读取运行日志的内容，根据日志内容进行处理：

- ✓ 1.确定故障前夭折的事务及无法正确执行的事务
- ✓ 2.确定故障前已提交的事务

■ 检查点

- ✓ 目的：为了减少系统故障后恢复的工作量，一般在日志中加入检查点
- ✓ 工作：是一些周期执行的操作，一般几分钟执行一次，其工作是把日志中的全部运行记录及对数据库的更新操作写入到稳定存储器中

三、采用日志对稳定存储器的信息的恢复

- **丢失稳定存储器的信息，但日志文件未丢失**

  把数据库恢复到一个正确状态（某一个时刻转储到别的存储介质上的数据库），然后，根据日志文件的内容，从该时刻起，进行恢复处理，重做所有已提交的事务

- **丢失稳定存储器的信息，且日志文件也受损**

  这种情况下的数据库，不可以恢复到故障以前的状态，因为，日志文件受损，这样的故障是灾难性的故障，出现这种故障时，数据库只能回到一个较早的正确状态（即转存到另外存储介质上的数据库），然后，依据日志中未受损的部分进行恢复

7.5 多副本恢复方法

7.5.1 概述

- **分布式数据库引入冗余的目的**
  - ✓ 1. 提高读操作的局部性
  - ✓ 2. 提高系统的可用性及可靠性
- **引入冗余带来的问题**

  虽然数据的冗余使系统效率提高，可用性及可靠性增强了，但是也带来了保证分布式数据库系统中各副本数据的一致性的问题
- **不同多副本策略的恢复方法**
  - ✓ 读一写全法
  - ✓ 多数法
  - ✓ 主副本法

7.5.2 读一写全法的恢复

读一写全法指写锁全部副本、读锁一个副本，这种方法中读副本不考虑数据的一致性，副本数据有一致性是由写锁全部副本保证的

- 当无故障时，更新肯定对所有的副本更新
- 当有故障而导致有的副本无法加锁时，更新事务肯定不能做更新操作
- 如果系统发生的不是网络分割故障而是场地故障，只读事务的可用性不变，只要能锁定一个副本即可继续执行，对写事务的可用性可用以下的方法使之提高：
  - ✓ 故障时不可访问的数据的更新操作记入一日志文件中
  - ✓ 对可访问的数据继续执行操作，当系统恢复时，重做上述更新操作

7.5.3 多数法的恢复

■在这里考虑一种加权的多数法或法定的人数法，它采用和“基于法定人数的提交协议”相同的规则

■规则包括：每个数据项赋予投票数 V(X) 的总数，并给数据项X的每一个副本赋予一个投票的 $v(x_i)$，使得$v(x_i)$ 之和等于 V(X) ，于是读法定人数 $V_r(X)$ 和写法定人数 $V_w(X)$ 满足：

（1） $V_r(X) + V_w(X) > V(X)$

（2） $V_w(X) > V(X)/2$

第一个条件决定了同一副本读操作和写操作不可同时执行

第二个条件决定了两个事务对同一副本的写操作不能同时执行

7.5.3 多数法的恢复

- **一个事务可以读(写)的条件**

  当它对如此多的副本进行了加锁，以致于这些副本投票数之和大于或等于 $V_r(X)$ ($V_w(X)$)

- **多数法与读一写全法的比较**
  - ✓ 在网络分割情况下系统的可用性，对于写事务而言大于读一写全法，因为，此时不必锁全部副本而只锁部分副本；对于读事务而言，其可用性要小于读一写全法，因为，加锁时至少要锁一个副本，有时可能要锁多个副本
  - ✓ 采用多数法时，若不发生网络分割只发生场地故障，多数法没有读一写全法优越，因为，在场地故障时可能使事务很难达到法定的票数

7.5.4 主副本恢复方法

- 主副本方法假定一数据项x的全部的加锁都是在主副本场地进行的，所有的读写操作均对此副本进行，然后才把它传播给其它的副本
- 系统的可用性是由主副本的状态决定的
- 当允许读不一致数据时，系统的可用性可以提高
- 当主副本故障时，读操作可以借助其它的副本继续执行

7.5.4 主副本恢复方法

主副本法的几个改进的方案

- 允许在不同的副本上进行读出操作，这提高了读操作的局部性，使系统可用性及效率得以加强和提高
- 当主副本因故障无法访问时，允许更换主副本，提高了系统的可用性
- 允许根据主副本的模式更换主副本，即使在主副本无故障时，可根据副本的使用情况把主副本更换，以取得更高的使用效率

7.6 网络分割

- **独立恢复协议**

  有故障的场地在重启动时可以不用访问远程恢复信息而判定事务运行结果

- **一个可从单场地故障恢复的独立恢复协议**

  从基本2PC协议改进而得到，它基于以下假设：

  - ✓ 1.一个场地发现另一场地不工作的方法是，在指定的时间内没收到所要求的报文
  - ✓ 2.只可能因为场地故障丢失报文
  - ✓ 3.每个场地从接收报文、进行处理到发送应答报文，是一原子性的状态改变

  满足上述条件就可以进行单场地独立恢复

- **独立恢复协议只能从单场地故障中恢复，不存在从多场地故障恢复的独立恢复协议**

7.6 网络分割

- 当出现网络分割故障时有以下几个事实存在：

✓ 网络分割时丢失了报文，则不存在非阻塞的协议能从网络分割故障中恢复

✓ 如果所有的不能发送的报文均返回到发送者，那么存在可以从单个网络分割故障中恢复的非阻塞的协议

✓ 不存在可以从多个分割故障中恢复的非阻塞的协议

7.6 网络分割

- 一般不存在网络分割时的非阻塞协议，必须选择另处的策略来避免阻塞
- 比较方便的方法是允许事务至少由一组场地终结。这个组可能是最大组，这样阻塞减少到最少，但一个组无法确定自己是否是最大组，因为它不知道别的组有多大
  - 对于这个问题有两种解决办法：主场地法和多数场地法

7.6 网络分割

一、主场地法

- 主场地法中，只允许包含主场地的组终结事务
- 如果把主场地与2PC协议一起使用，仅当它的所有挂起事务的协调者均在此组中时，才可能使主场地的全部事务终结；可以通过赋予主场地为全部事务的协调者来达到目的；这种方法在多数网络中效率很低且对主场地的故障极敏感
- 可以采用3PC协议，事务由主场地组终结；3PC没办法区别场地故障和网络分割，当分割修复后，非主场地组的场地要像从其它故障中重启动那样工作；在网络分割时它们不释放其所拥有的锁

二、多数法和基于法定人数的协议

- 多数法基本思想是，在该事务中止或提交以前，大多数场地必须同意中止或提交，多数法不能与2PC一起使用
- 基本多数法的一个简单的推广是，给各场地赋以不同的权，使用加权的多数法的协议叫做基于法定人数的协议。赋予各场地的权通常叫做投票数，在对一场地的事务提交或中止“投票”时使用

7.6 网络分割

多数法的基本规则

- ✓每个场地i有许多投票数V_i与之关联，V_i是一正整数
- ✓令V表示此网络中全部站点的投票数之和
- ✓一事务在提交以前必须收集一提交法定人数V_c
- ✓一事务在中止以前必须收集一中止法定人数V_a
- ✓$V_a+V_c>V$（用来保证事务要么提交要么中止）

注意事项

采用这种协议时，重启动的场地必须参与法定人数的形成，否则，有可能永远达不到法定人数，而且，由于协调投票的场地发生故障时，对于是否到达法定人数无法确定，所以，在一个场地参与投票以后，要在日志中记录下该场地的投票方向，即使在故障后重启动也不能改变主意

7.6 网络分割

基于法定人数的3PC的集中式终结协议具有以下结构：

- 选举一个新协调者
- 协调者收集状态信息并按下列规则动作：

(1) 如果至少一个站点提交(中止)，则发一提交(中止)命令给其它场地

(2) 如果已到达 “准备就绪” 状态的场地的投票数V大于或等于V_C,则发送提交命令

(3) 如果处于“准备中止”状态下的场地的投票数达到中止法定人数则发送中止命令

(4) 如果到达“准备就绪”状态的场地的投票数加上不确定场地的投票数大于或等于V_C，则发送 “准备提交”命令给不确定的场地，然后等待上述条件(2)的出现

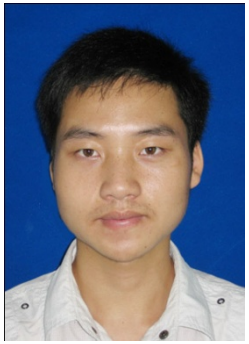
(5) 如果已到达“准备中止”状态的场地的投票数加上不确定的场地的投票数大于或等于V_a，则发送”准备夭折”命令给不确定的场地，然后等待上述条件(3)的出现

(6) 否则等待某个故障的恢复

附件：主讲教师和助教

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn



助教：赖明星

单位：厦门大学计算机科学系数据库实验室2011级硕士研究生
E-mail: mingxinglai@gmail.com

Department of Computer Science, Xiamen University, Sep, 2012