# 8.分布式容错管理

## 1.容错性概述

### 1.1 基本概念

**可依赖系统（Dependable, Trustworthy）：**

- 可用性（availability）：在定时刻能正确操作的概率，可用性高表示任何给定时刻都能正常运行
- 可靠性（Reliablity）：在给定期间能正确操作的概率，可靠性高的系统可以连续运行很长时间
- 安全性（Safty）：临时失效不会造成灾难
- 可维护性（Maintainability）：故障系统被恢复的难易程度

**失效（fail, failure）、失灵**：§ 系统不能兑现它的承诺（提供服务）

**差错（error）**：导致系统失效的系统状态， 例如， 一个图片没能打开

**故障（fault）**：导致差错发生的原因， 例如， 图片服务器挂掉

建立一个可靠系统与故障控制紧密相关

- 预防（preventing）
- 去除（removing）
- **容错(tolerating)**
- 预告(forecasting)

**容错（fault tolerance）**：即使发生故障， 系统仍能提供服务

故障类型

- 短暂型(transient): 出现一次， 再也不出现
- 间歇型(intermittent): 消失后， 再重复出现
- 永久型(permanent): 一直存在

## 1.2 失效（失败）类型

| 失效类型 | 描述 |
| --- | --- |
| 崩溃性失效 | 服务器停止，但在停止前一直正确工作 |
| 遗漏性失效<br>接收遗漏<br>发送遗漏 | 服务器不能响应到来的请求<br>服务器不能接收到来的消息<br>服务器不能发送 |
| 定时性失效 | 服务器的响应超出规定的时间间隔 |
| 响应性失效<br>值失效<br>状态变迁失效 | 服务器的响应不正确<br>响应的值是错误的<br>服务器偏离正确的控制流 |
| 任意性失效<br>（拜占庭型故障） | 服务器在任意的时刻产生任意的响应 |

## 1.3 失效（失败）模型（halting failure）

- 失败即停 (fail-stop)。 良性、 易检测。
- 失败缄默(fail-silent)。 不通知失效。
- 失败安全(fail-safe)。 不产生恶果。

## 1.4 基于冗余的失效屏蔽技术

### （1）冗余类型

信息冗余： 如， 海明码。

时间冗余： 如， 重发， 重做

物理冗余：

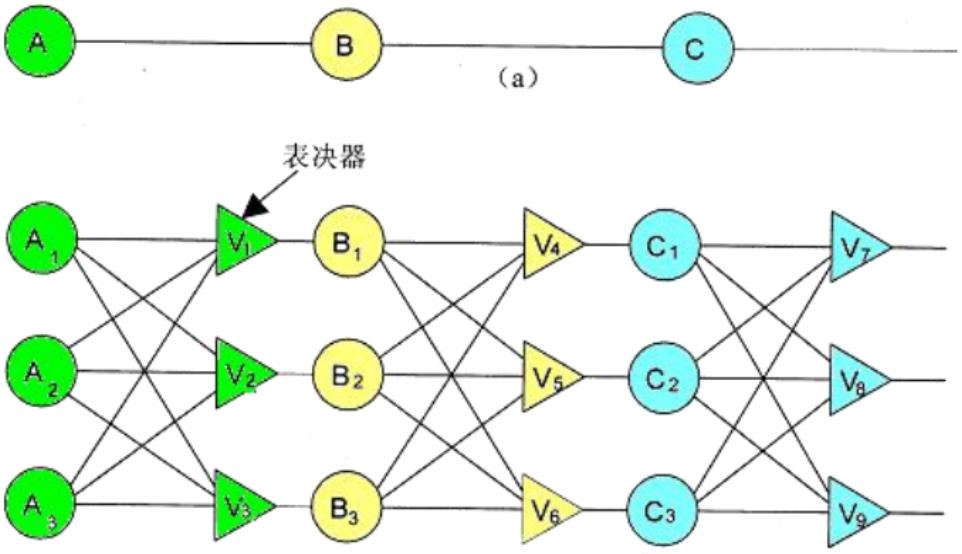
- 软件:如复制进程
- 硬件： 如复制电路

信息冗余和物理冗余都属于空间冗余

### （2）三模冗余方法（TMR，Triple Modular Redundancy）

三路表决器(voter)： 三路输入， 一路输出

可屏蔽一路错误（任意性失效）




# 2.进程的可靠性

## 2.1 概念

**进程容错**

- 进程组： 具有相同功能的进程集合；组中其他进程可以接管失败进程
- 组成员籍
  - 加入： 具有成员籍
  - 脱离： 注销成员籍
  - 多成员籍： 同时属于不同的组

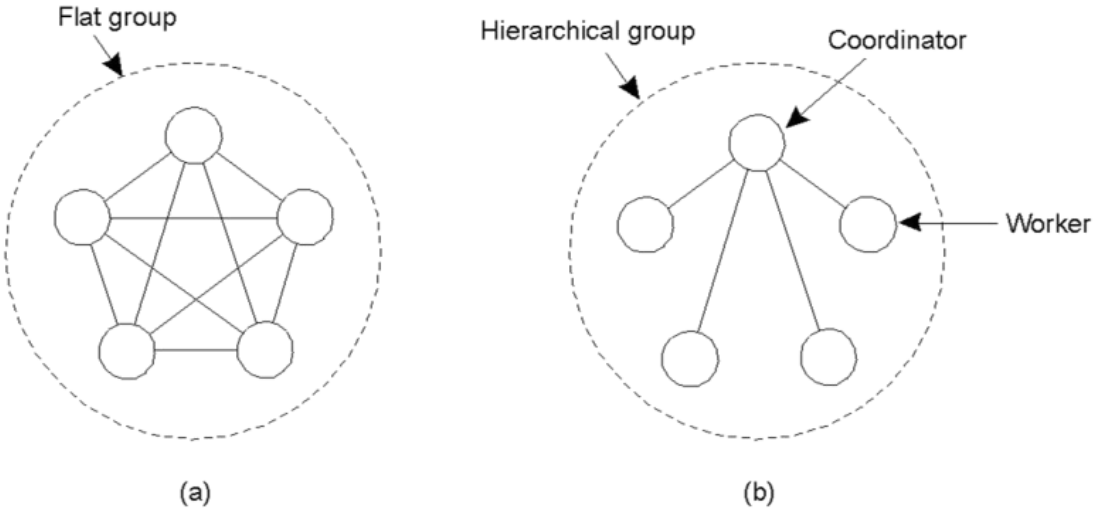
**设计问题**

- 需要复制的程度
- 无故障时， 平均情况和最坏情况下的系统性能
- 有故障时， 平均情况和最坏情况下的系统性能

## 2.2 组的管理

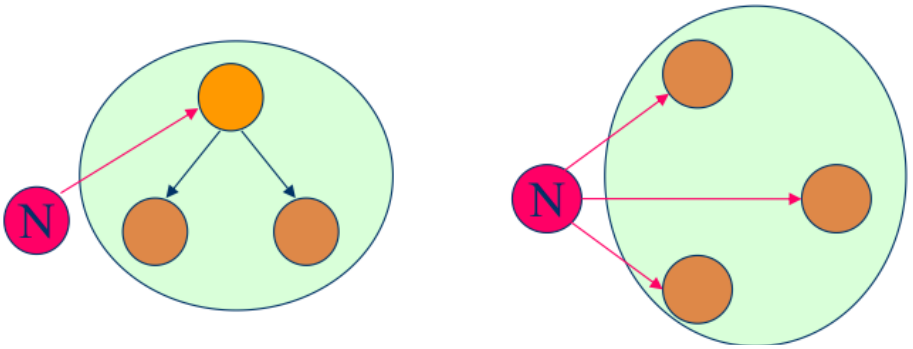
平等组：所有成员是等同的

层次组：由协调者和工作者组成




组成员籍管理

- 组服务器：集中式管理
- 多播通信：分布式管理




故障后，组的退出

- fail–stop 类型：发送 Goodbye 信息

- fail-silent 类型：需其他人员发现

消息同步：

- 加入组时：立刻收到改组的所有消息
- 退出组时：不再收到改组的任何消息

组的重建

- 当组崩溃后，根据协议重新建立组

## 2.3 复制容错技术

**复制容错**

- 用多个相同的进程， 屏蔽个别故障进程的故障
- 冗余度： 相同进程的个数

**基于主进程协议(primary-based)**

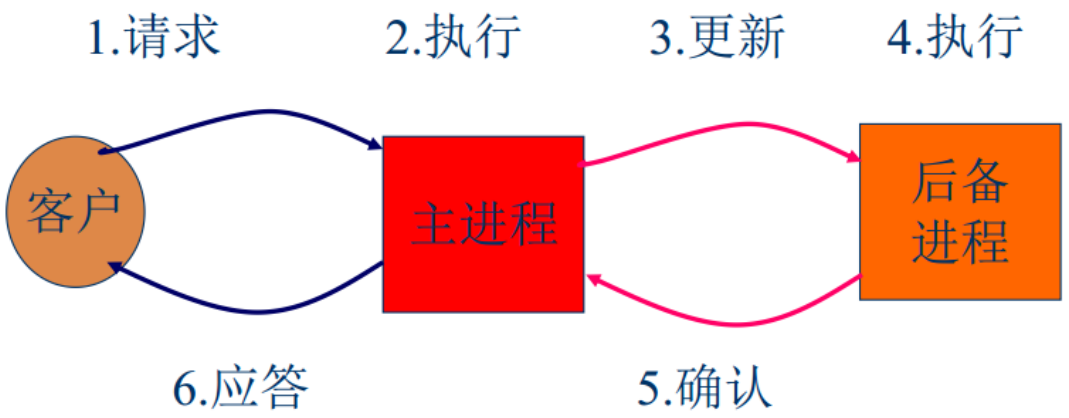
- 结构： 层次组结构
- 协议： primary-backup 协议
- 接管： 主进程崩溃， 后备进程选举新的主进程

**复制写协议(replicated-write)**

- 结构： 平等组结构
- 协议： 基于表决数协议

**主备份方法**

- 主进程失效，则后备进程接替其任务
- 接管模型




**k-容错度：**

- 在有 k 个进程发生故障时， 系统仍能正确运行
- 在复制写协议情况下：
  - Fail-stop 型故障： 对 k-容错度， 需 k+1 冗余度
  - 拜占庭型故障： 对 k-容错度， 需 2k+1 冗余度
- 容错的前提条件：
  - 所有的请求到达所有服务器的顺序应相同 ， 要不全收到， 要不全收不到
  - 原子广播问题（atomic broadcast problem）

## 2.4 故障系统的协定问题

### （1）基本概念

**协定(agreemnt)** ：对某些问题的一致意见。 如， 是否提交事务， 负载划分， 同步等

**分布式协定算法**：在有限的步骤内， 所有非故障进程达成协定

底层系统中可能的情况：

1. 同步 vs 异步系统
2. 限定的或无限定的通信延迟
3. 有序的或无序的消息递交
4. 单播(unicast)或多播(multicast)式消息传送

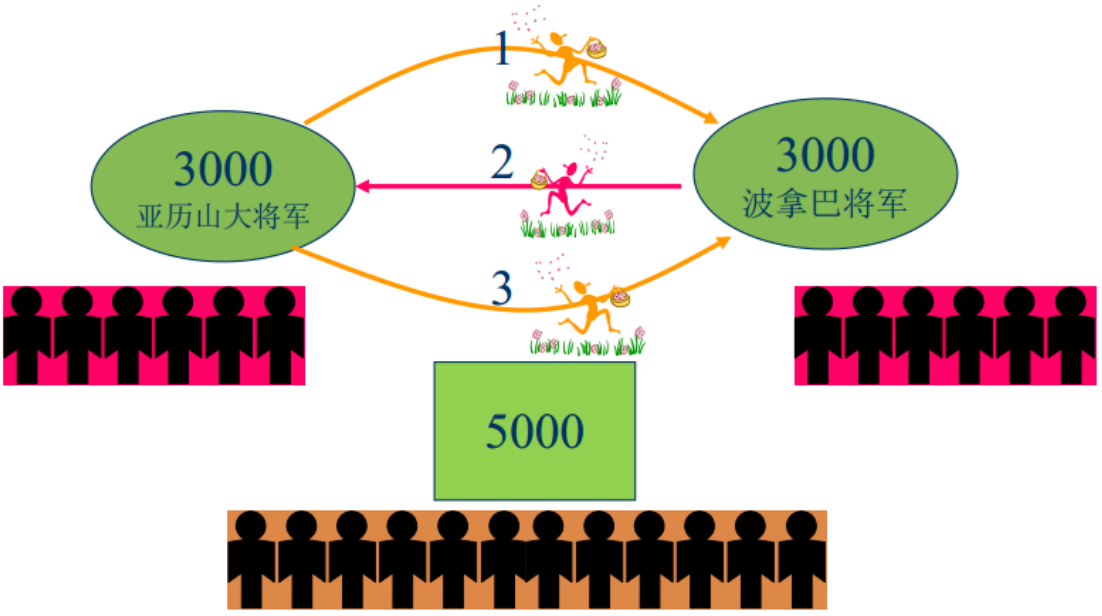
## （2）三种情况

可以取得一致的三种情况

1. **处理机同步方式、 通信延时有限** ： 处理机可用超时检测机制， 确定其他失败进程
2. **消息有序， 广播式传输**：每个处理机原子式广播一个初始值， 其他处理器按照次序接收， 能够同意谁是第一个发送的
3. **处理机同步， 消息有序**

## （3）两军问题

两个将军 A1 和 A2 决定攻打同一个敌人 B ，任一方都没有敌人强大，所以只有共同进攻才有胜算。敌军正好位于两个将军之间，意味着信使可能会被敌军抓住。两军问题指的就是在这样的条件下两个将军如何就是否进攻达成共识。



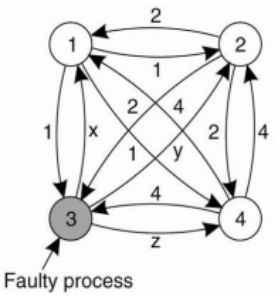

对于不可靠通信， 即使进程是可靠的， 也不可能达成协定

**拜占庭将军协定问题**

- 假设通信是可靠的， 但进程可能是不可靠的
- 例： 3 个忠诚将军， 1 个叛变将军

**Lamport 递归算法**

共 4 步： (a)对外报告 (b)收集向量(c)报告向量(d)生成结果向量： (1， 2， 未知，4)

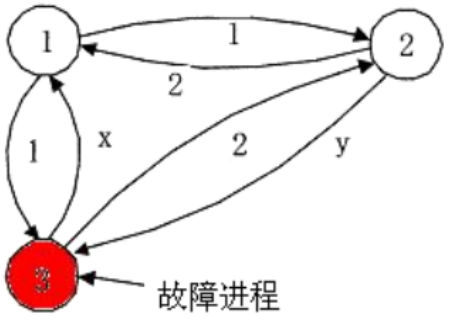



1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

| 1 Got | 2 Got | 4 Got |
|---|---|---|
| (1, 2, y, 4) | (1, 2, x, 4) | (1, 2, x, 4) |
| (a, b, c, d) | (e, f, g, h) | (1, 2, y, 4) |
| (1, 2, z, 4) | (1, 2, z, 4) | (i, j, k, l) |

## （4）三个将军

若三个将军中， 有两个忠诚将军， 一个叛变将军， 则不能判断出哪个将军叛变。




(a)

1 Got (1, 2, x)
2 Got (1, 2, y)
3 Got (1, 2, 3)

(b)

1 Got (1, 2, y)
(a, b, c)
2 Got (1, 2, x)
(d, e, f)

(c)

若要有 m 个进程出错的系统实现协同一致， 最少要有 2m＋1 个正常进程。 进程总数为 3m+1。

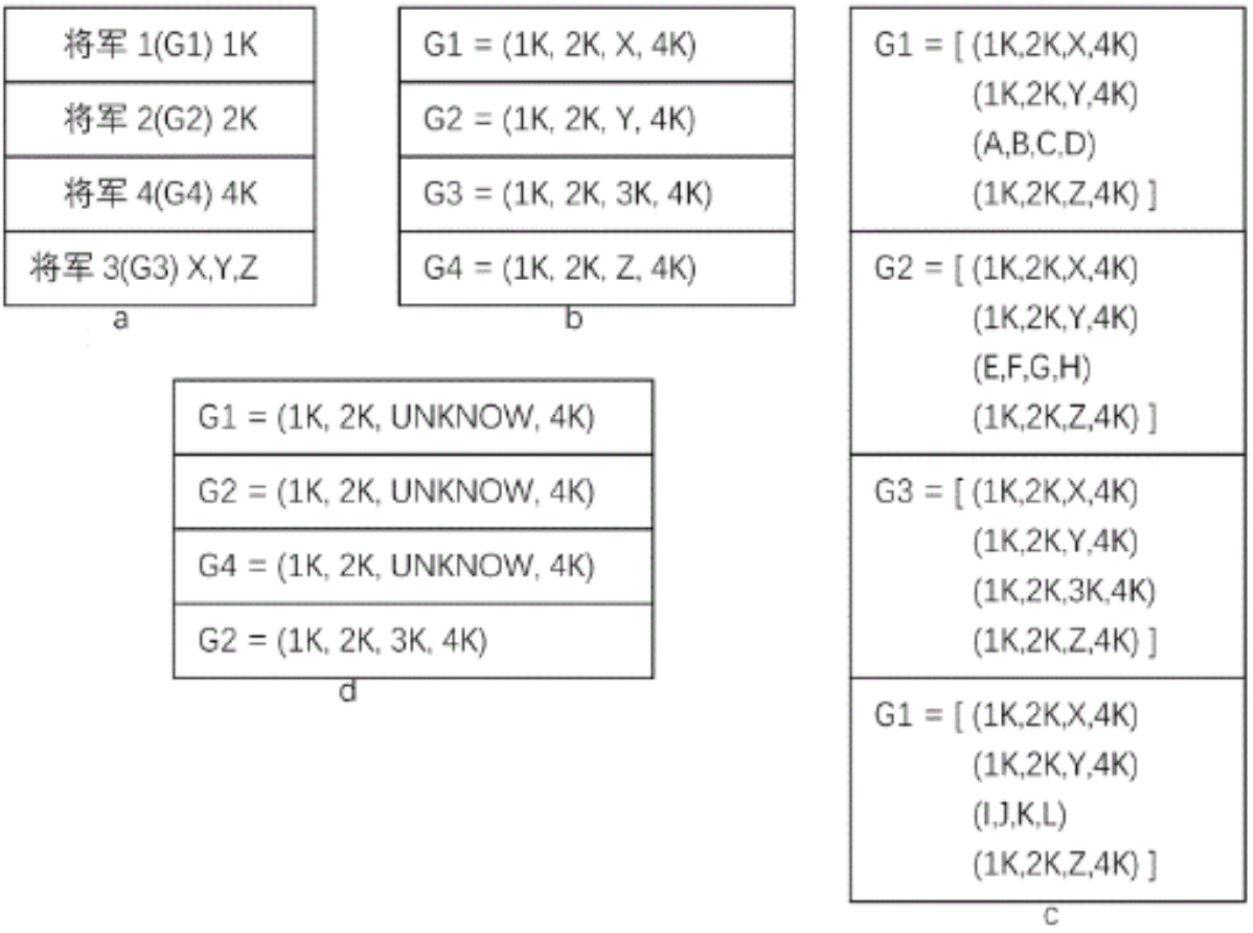
需超过 2/3 多数， 才能达成协定

## （5）Lamport 递归算法示例

问题：N=4（有四个将军）；M=1（有一个叛徒），

Lampot 等人设计了一种递归算法可在特定条件下解决这一问题；共有四步

1. 每个将军发送可靠的消息给其他所有将军，声明自己真实的军队人数，忠诚的将军声明的是真值，叛徒那么可能对其他每个将军都撒一个不同的慌，如图 a
2. 把第一步声明的结果组成向量的形式，如图 b
3. 每个将军把图 b 中各自的向量传递给其他将军，这里叛徒再一次撒谎，使用 12 个新值，A~J，如图 c
4. 每个将军检查所有接收向量的每一个元素，假设某个值占多数，那么把该值放入向量中；




## 2.5 故障检测

无故障进程应能检测出故障进程

故障进程检测方法(超时机制)

- 主动式方法， 发送 “Are you alive?”消息；常用方法， ping 操作
- 被动式方法， 等待其他进程发来的消息；每个结点定期广播状态， 状态较旧的有问题

如何区分网络故障和结点故障：请邻居结点帮忙检查

# 3.客户服务器通信的可靠性

## 3.1 点对点通信

可靠通信：防止通信失效

遗漏型失效：消息丢失

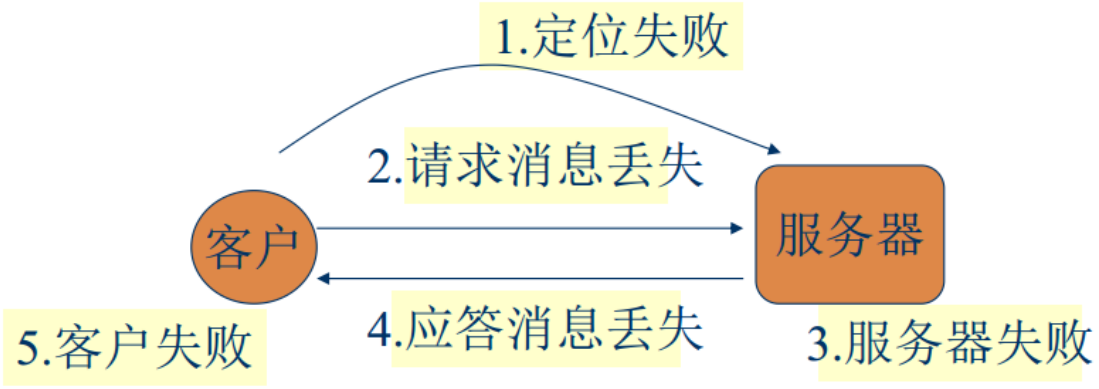
- 解决策略： 利用可靠的传输协议， 如 TCP 协议， 通过消息确认和重新传输来掩盖遗漏性失效。

连接崩溃失效： 连接中断

- 通常不能屏蔽， 需重建连接
- 解决策略： 抛出例外， 通知客户进程

## 3.2 RPC 失效

5 种失效情况：




### （1）客户不能定位服务器

可能服务器被修改， 客户存根（stub） 与新的服务器存根不匹配

解决策略：

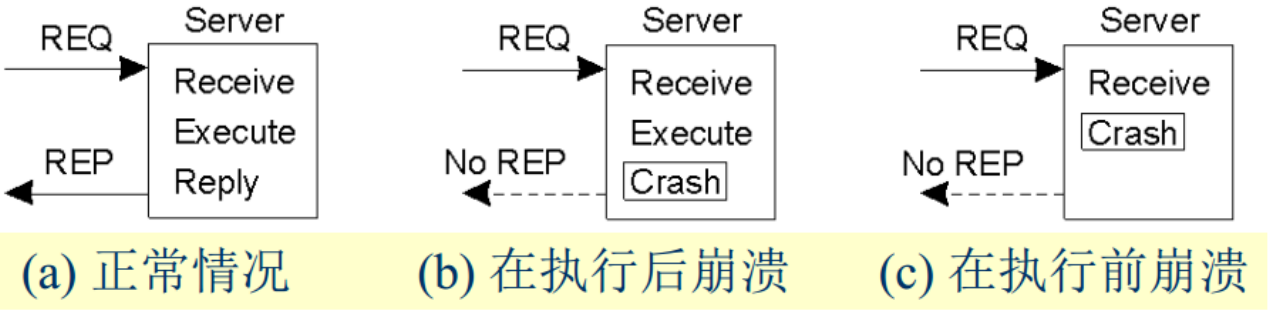
- 抛出一个异常（exception） ， 如 JAVA、 C++
- 使用信号（SIGNOSERVER） ， 如 C 语言， 然后由编写的信号处理程序做相应处理。
- 没有透明性

### （2）丢失请求消息

解决策略：客户发现超时，重发请求

### （3）服务器崩溃

崩溃情况




(a) 正常情况　(b) 在执行后崩溃　(c) 在执行前崩溃

解决策略：

- 至少一次语义：不断尝试，将应答传给客户
- 最多一次语义：立即放弃，并报告失败
- 听之任之：RPC 可能执行任意多次（容易实现）
- 确切一次语义：（目标）

举例： RPC 远程操作为打印文本， 服务器崩溃后恢复， 并向客户通告。

问题： 客户不知其打印请求是否已被执行

服务器端策略

- 在打印之前向客户发送 ACK 消息
- 在打印之后向客户发送 ACK 消息

客户端策略

- 总是重发请求： 文本可能被打印两次
- 绝不重发请求： 可能文本不被打印
- 收到 ACK 时重发
- 没有收到 ACK 时重发

出现服务器失效时， 客户和服务器的策略组合

- 服务器事件： M(发送完成消息)； P(打印)； C(崩溃)

| 客户 | 服务器 | | | | | |
|---|---|---|---|---|---|---|
| | 策略 M -> P | | | 策略 P -> M | | |
| 重发策略 | MPC | MC(P) | C(MP) | PMC | PC(M) | C(PM) |
| 总是重发请求 | DUP | OK | OK | DUP | DUP | OK |
| 总是不重发 | OK | ϕ | ϕ | OK | OK | ϕ |
| 当收到ACK时，重发 | DUP | OK | ϕ | DUP | OK | ϕ |
| 当没收到ACK时，重发 | OK | ϕ | OK | OK | DUP | OK |

### （4）丢失应答消息

解决策略：定时器、超时检测、重发请求

问题：重复操作

解决策略：

- 构造幂等性操作（idempotent）
- 区分原始消息和重发消息：为请求分配顺序号；附加标志位

### （5）客户崩溃

孤儿进程问题

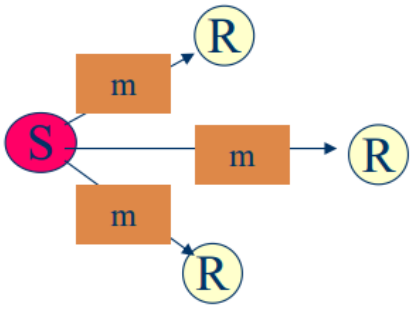
解决策略：

- 根除法：利用日志，撤销孤儿进程
- 再生法：设置按顺序编号的时期（epoch）。客户重启时，广播新时期开始，撤销旧的孤儿进程
- 温和再生法：时期广播到达时，尝试定位远程计算的拥有者，撤销无助的孤儿
- 过期法：设置时间量 T。如果超过 T，则撤销孤儿进程

## 4.分组通信的可靠性

假定：进程操作正确，在通信中不加入或退出组

可靠多播：将一个消息递交给每一个当前组员

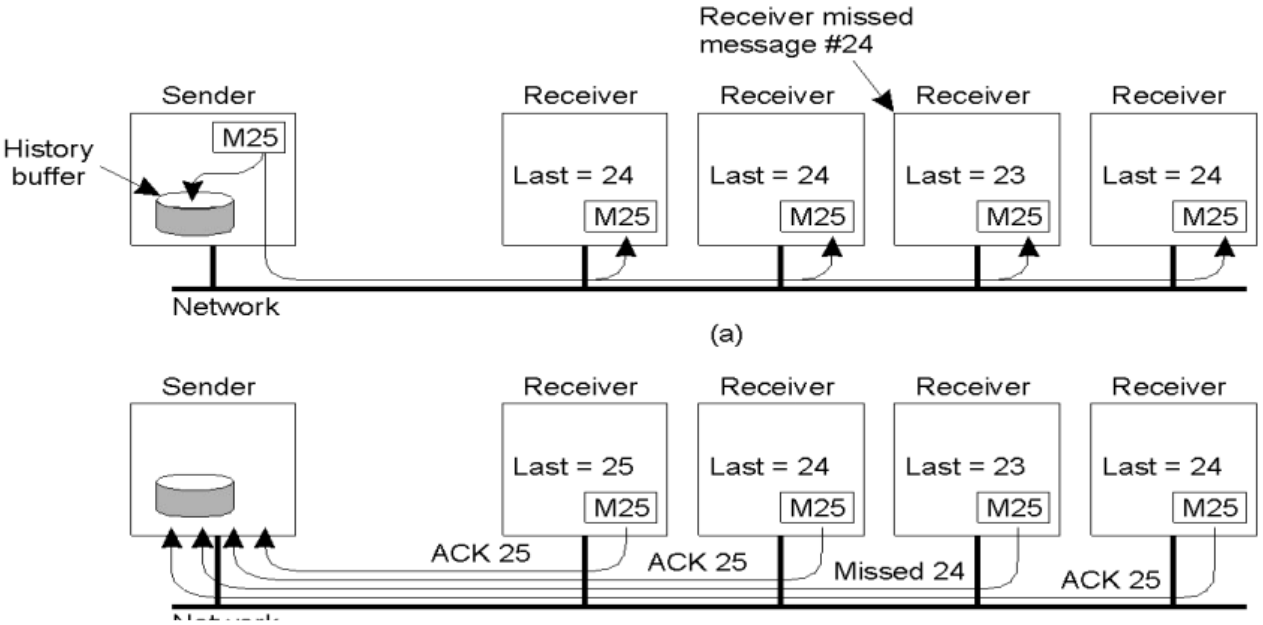



不可靠多播：不能保证将一个多播消息递交给所有组员

## 3.1 基本可靠多播模式

解决策略：

- 消息传播：记录顺序号
- 报告反馈：如果丢失，返回负 ACK，重新发送




## 3.2 可靠多播的可伸缩新

**反馈爆炸问题**：N → 1

**简单解决方案**：

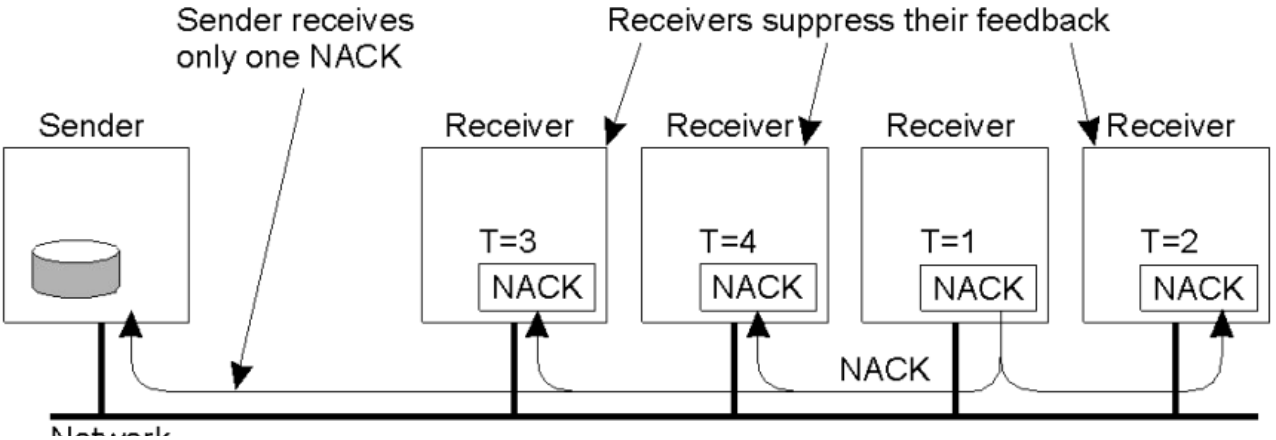
- 接收者只返回 NACK（否定）消息
- 发送者保留消息到历史缓冲区

**仍然存在问题**：

- 历史缓冲区溢出问题
- NACK 反馈爆炸

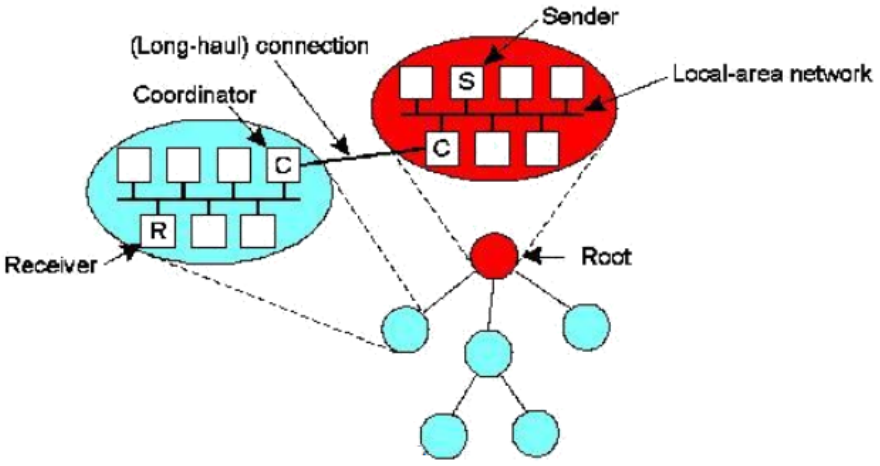
**无层次反馈抑制技术（SRM，可伸缩的可靠多播协议）**

- 不反悔 ACK，只返回 NACK，向发送者和组内广播式发送
- 压缩 NACK。随机延时后，如果接收到其他的重发请求，就抑制自己的 NACK。理想情况下，只有一个重发请求




层次化反馈控制

- 一个大组划分成若干个小组，形成一个树
- 发送者所在的小组为树的根
- 每个局部协调者转发消息给它的孩子
- 局部协调者负责请求重发



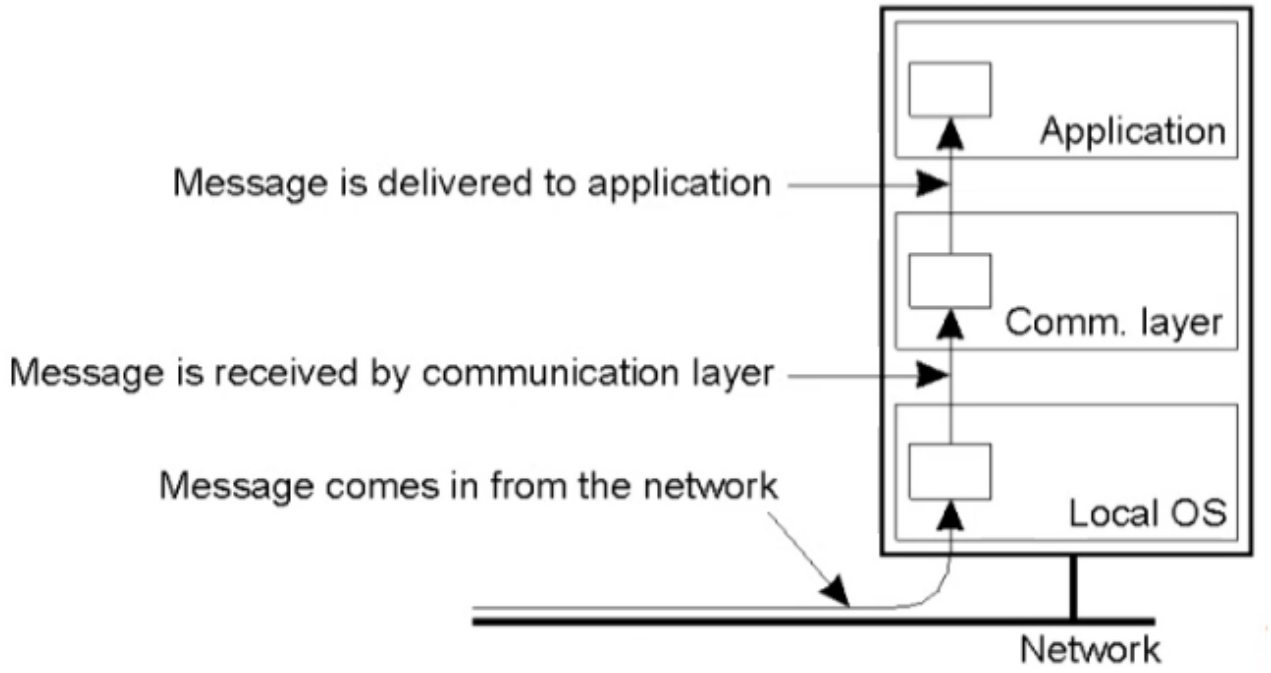
## 3.3 原子性多播

**原子性多播问题**

- 消息要么递交给所有组员，要么一个也不递交
- 对于每个组员，所有消息的递交次序是相同的

**用途举例**：主动式复制协议

**带用通信层的分布式系统结构**：能区分消息接收和消息递交




**故障组员处理**

- 出故障后，自动退出组
- 修复后，重新加入组

**组视图 G（group view）**

- 在发送一个消息时，属于该组的所有进程的名单

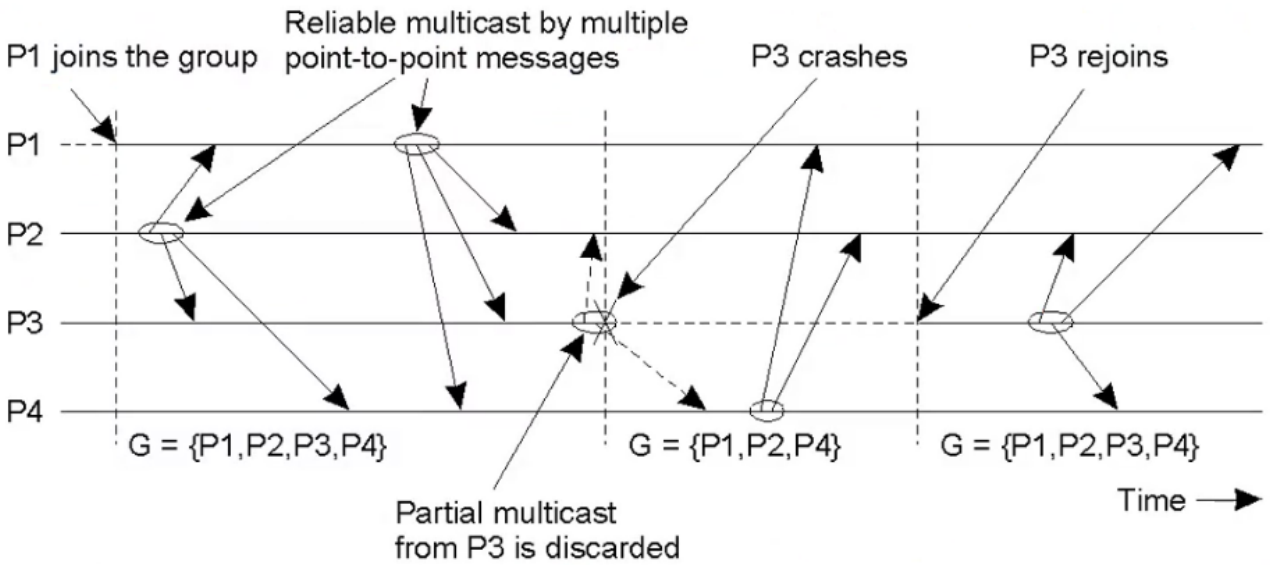
**视图变更：**

- 向所有的组员宣布加入或退出该组

**虚拟同步（virtually synchronous）的可靠多播**

- 如果发送者在多播时崩溃，消息将递交给其他所有组员，或者被它们丢弃

**原理：所有多播在视图变更之间进行**




消息递交次序

1. 可靠的无序多播：不能保证接收到消息的递交次序是相同的

时间

| Process P1 | Process P2 | Process P3 |
|---|---|---|
| sends m1 | receives m1 | receives m2 |
| sends m2 | receives m2 | receives m1 |

2. 可靠的 FIFO 次序多播：从同一发送者接收到的消息的递交次序与发送次序一致

   举例：发送者 P1，P4；接收者 P2，P3

时间

| Process P1 | Process P2 | Process P3 | Process P4 |
|---|---|---|---|
| sends m1 | receives m1 | receives m3 | sends m3 |
| sends m2 | receives m3 | receives m1 | sends m4 |
| | receives m2 | receives m2 | |
| | receives m4 | receives m4 | |

3. 可靠的因果次序多播：具有因果关系的消息的递交次序与发送次序一致，无论消息是否由同一发送者发送；可使用时间戳向量实现
4. 全序多播：对于所有组员的消息递交次序，是相同的

**原子性多播**：提供全序递交的虚拟同步可靠多播

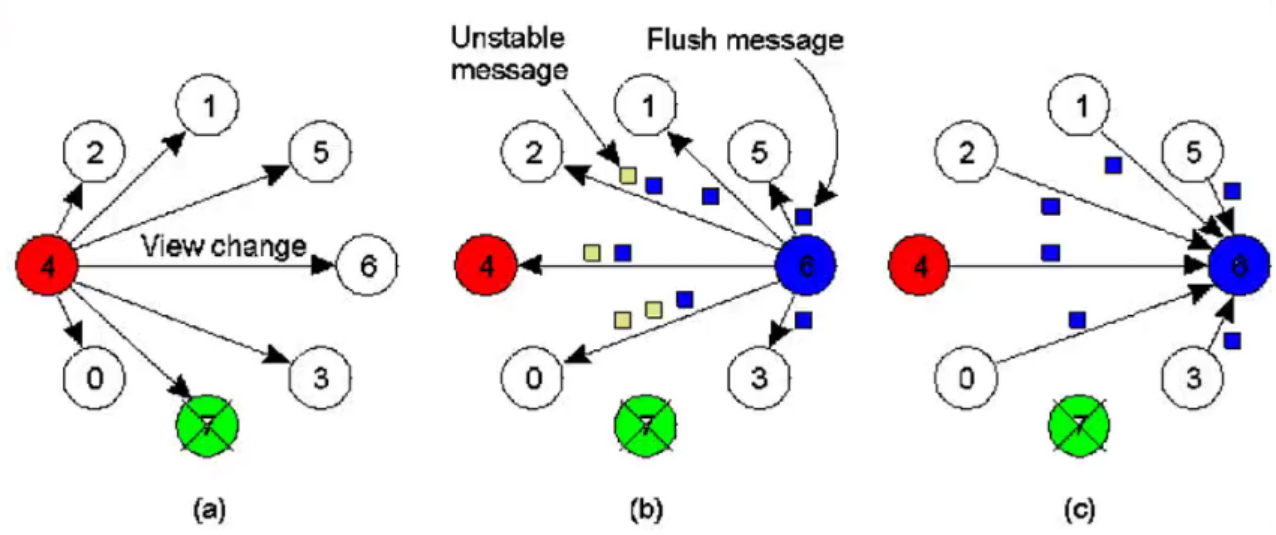
**虚拟同步的可靠多播的 6 个版本**

| 多播 | 基本的消息次序 | 全序递交? |
| --- | --- | --- |
| 可靠多播 | 无 | No |
| FIFO 多播 | FIFO 次序型递交 | No |
| 因果多播 | 因果–次序型递交 | No |
| 原子性多播 | 无 | Yes |
| FIFO 原子性多播 | FIFO–次序型递交 | Yes |
| 因果原子性多播 | 因果–次序型递交 | Yes |

**虚拟同步性的实现：举例：ISIS 系统**

- 底层网络：可靠点对点通信，消息排序
- 稳定消息：G 中所有进程都接收到消息 m

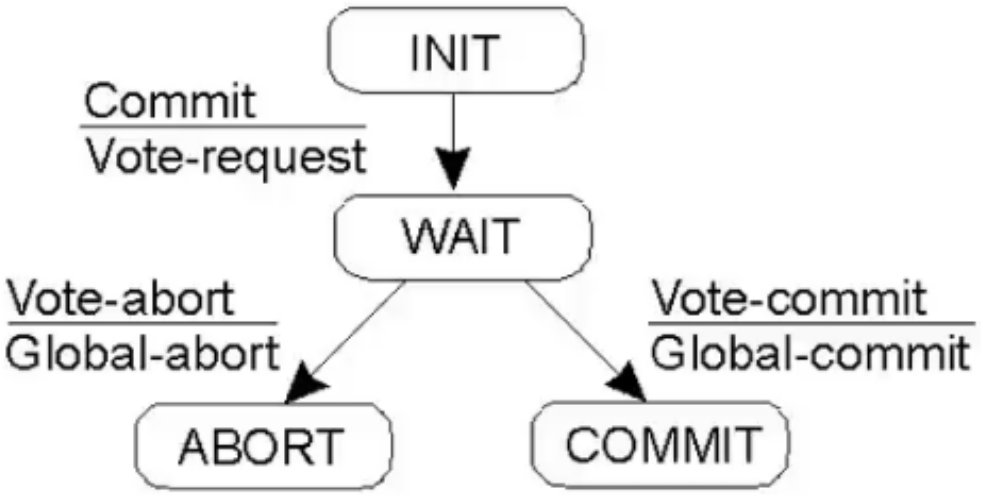
1. P4 发现 P7 崩溃，多播视图变更消息（vc）
2. P6 发送所有不稳定消息（没收到所有进程 ACK 的消息），后续 flush（刷新）消息
3. P6 接收到所有返回的 flush 消息后（证明 P6 没有待发送消息）确定新的视图 $G_{i+1}$



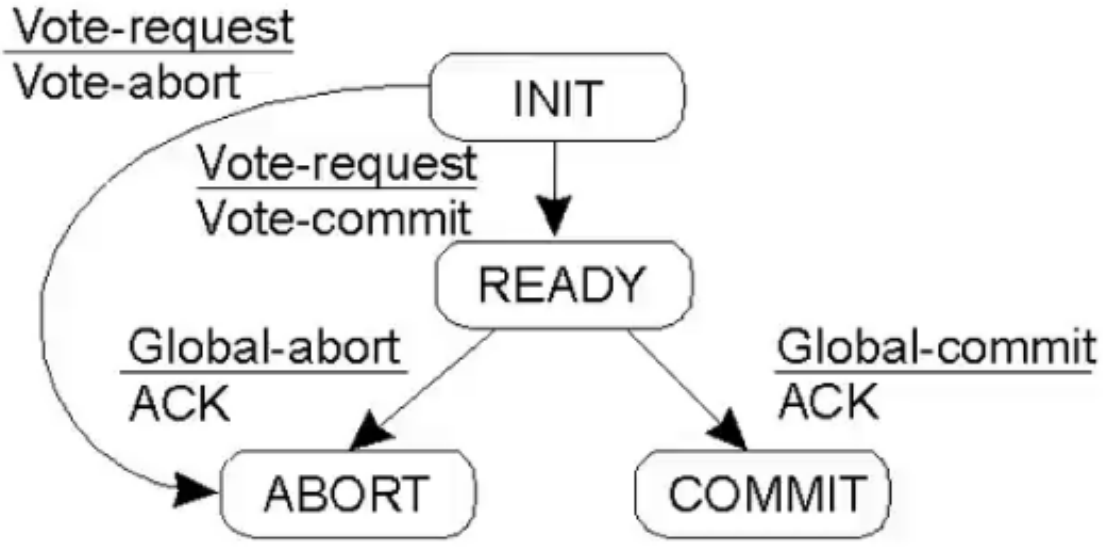

# 5.分布式提交

## 5.1 两阶段提交协议

协调者的有限状态机




参与者的有限状态机

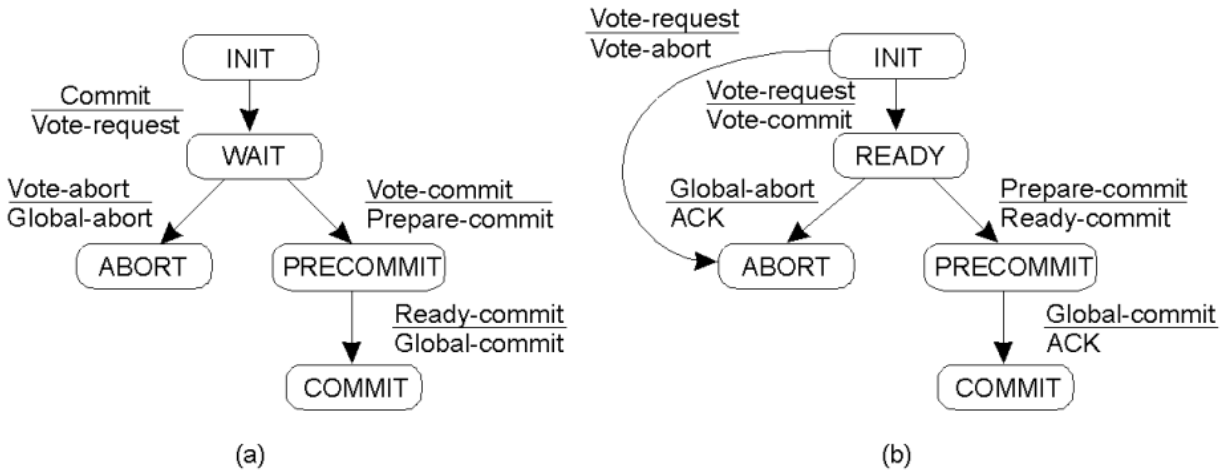



当参与者 P 处于 READY 状态，并已与另一参与者 Q 通信之后，可能采取的动作

| 进程 Q 状态 | 进程 P 动作 |
| --- | --- |
| COMMIT | 变迁到 COMMIT |
| ABORT | 变迁到 ABORT |
| INIT | 变迁到 ABORT |
| READY | 与其他参与者联络 |

## 5.2 三阶段提交协议（不考）

目的：在协调者失败即停故障的情况下，避免进程阻塞




(a) 协调者的有限状态机　　(b) 参与者的有限状态机

协调者（WAIT）： 发现超时,则 abort

协调者(PRECOMMIT)： 发现超时,继续 commit

参与者(INIT)： 发现超时,则 abort

参与者(READY)：

发现超时， 询问其他参与者

- 如果有 COMIT/ABORT,则执行
- 如果大都为 PRECOMMIT,则 commit
- 如果有 INIT， 则 abort
- 如果都为 READY， 则 abort

参与者(PRECOMMIT)并形成多数： 发现超时,继续 commit

# 6.恢复处理

## 6.1 概念

**目的**: 使系统从错误状态到正确状态

**类型**：

- 向后恢复(backward recovery)： 使系统返回到上一个正确状态
- 向前恢复(forward recovery)： 使系统前进到一个正确的新状态

检查点技术(checkpoint)

消息日志技术（logging）

- 基于发送者的写日志
- 基于接受者的写日志

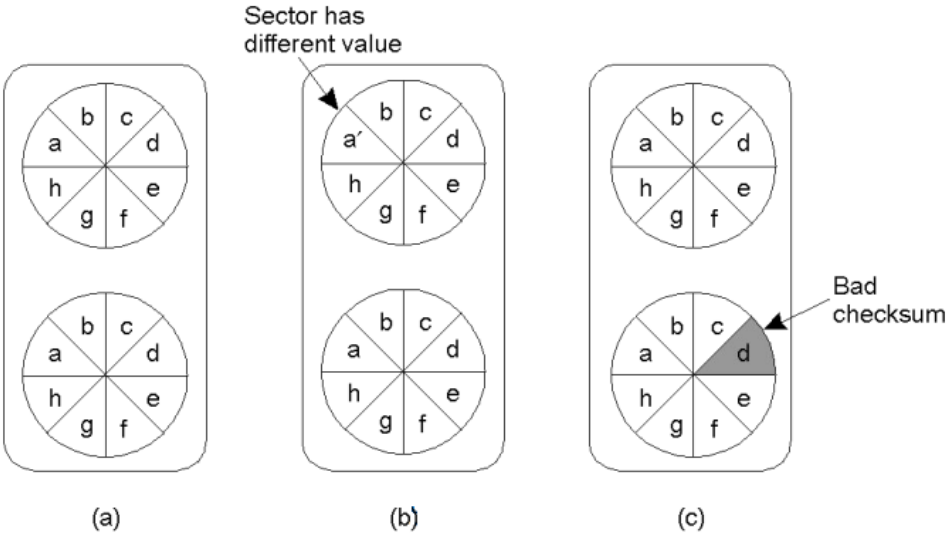
## 6.2 可恢复的稳定存储器

稳定存储器状态

a) 稳定存储状态

b) 崩溃状态： 在更新驱动器 1 后发生

c) 坏点状态： 出现坏扇区

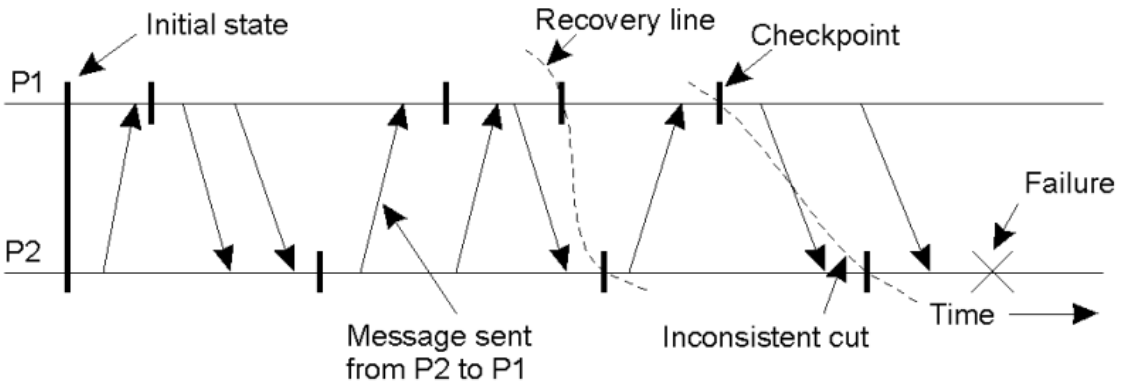



## 6.3 检查点

**分布式快照（snapshot）**：一致的全局状态

**恢复线**：最近的分布式快照， 最近的一致性割集

举例

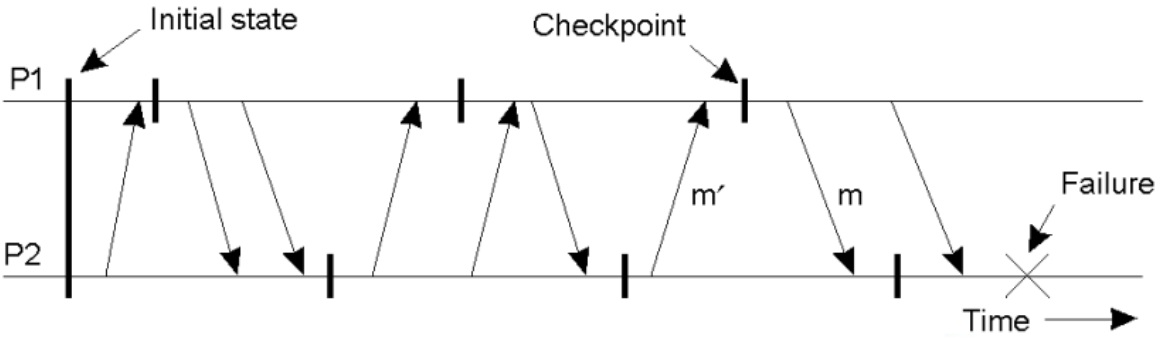



## 6.4 独立检查点

独立检查点：每个进程的检查点是相互独立的

问题： 多米诺效应；局部状态没有形成分布式快照， 导致级联回滚（cascaded rollback） 过程

举例： 只有 m,m'的接受记录， 没有发送记录




解决方法

- 设 CP[i](m)表示 Pi 的第 m 个检查点
- 设 INT[i](m)表示 CP[i](m)和 CP[i](m−1)之间的间隔
- 当 Pi 在 INT[i](m)中发送消息 x 时， 带上(i,m)
- 当 Pj 在 INT[i](n)收到 x 后， 记录依赖关系 ： INT[i](m)→ INT[j](n)
- Pj 在 CP[j](n)中加入该依赖关系
- 当 Pi 需要回滚到 CP[i](m−1)时， 则 Pj 需要回滚到 CP[j](n−1)

## 6.5 协作式检查点

**同步写检查点**：所有进程同步地在本地稳存中写检查点， 使保存的状态自动地保持全局一致。

**非阻塞式算法：**分布式快照算法

**两阶段阻塞式算法**

- CHECKPOINT_REQUEST：协调者发送命令， 所有进程写局部检查点， 将要发送消息插入队列， 向协调者返回 ACK 消息。
- CHECKPOINT_DONE： 当协调者收到所有的 ACK 后， 发送命令， 所有进程继续

**改进算法 -- 增量快照算法**

- 最近发送进程：进程 P 在上一个检查点向其发送过请求的进程。
- 协调者恢复依赖进程： 在上一个检查点， 直接或间接收到协调者消息的进程。因此， 由最近发送进程的闭包集组成。
- 协调者只向其最近发送进程多播命令。
- 当进程 P 收到写检查点请求时， 仅向 P 的最近发送进程， 转发该请求。 每个进程仅转发该请求一次。
- 当所有进程被确认后， 协调者发送第二个多播命令，开始实际写检查点

## 6.6 消息日志

**基本思想**

- 减少检查点的个数
- 如果消息的传送可以重放(replay)， 则可取得全局一致性状态， 而不必从稳存恢复。

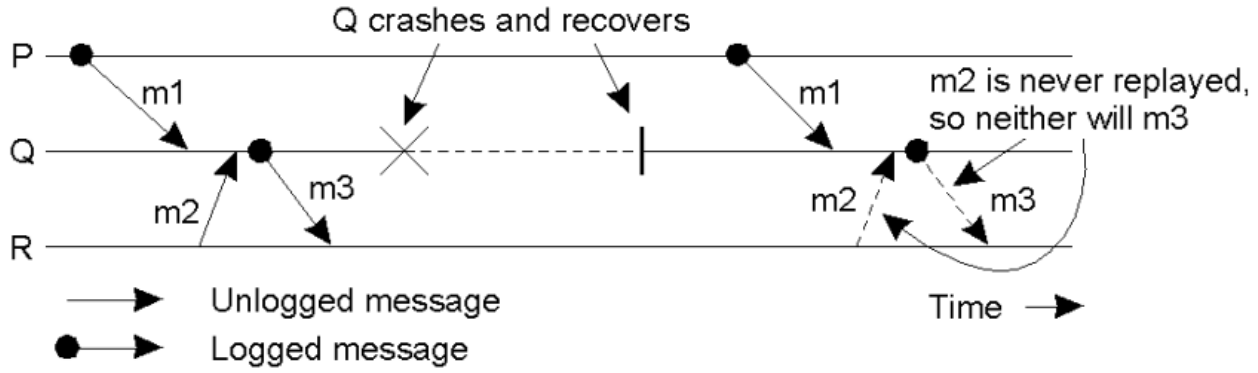
**分段确定性模型(piecewise deterministic model)， 假定**：

- 每个进程在一序列的间隔中执行， 有先后次序， 是确定性的。 每个间隔是可重放的。
- 每个间隔以一个非确定性事件开始， 以下一个非确定形式件发生而结束。
- 如果记录所有非确定性事件， 则全部执行可重放。

**孤儿进程**：在进程 P 崩溃后， 仍然存活的进程。 但在 P 恢复后，其状态与 P 不一致

举例：

- 进程 Q 崩溃， 消息 m2 没有写日志
- Q 恢复后， m2,m3 消息没有重放， 导致 R 为孤儿进程




**消息**

- 消息头： 发送者、 接受者、 顺序号、 递交号
- 稳定消息： 已记入稳存， 不会丢失

**DEP(m)： 依赖于 m 的接受**

- 依赖于消息 m 递交的进程集合。
- 与 m 有因果关系的进程集合

**COPY(m)： 能发 m 的进程**

- 拥有 m 的副本， 但未将 m 写入稳存的进程集合.

**孤儿进程**

- Q 存在于 DEP(m)， 而 COPY(m)中的所有进程崩溃

**避免孤儿进程**

- 如果 $Q \in DEP(m)$ ， 则保证 $Q \in COPY(m)$

**悲观型(pessimistic logging)日志协议**

- 保证每个不稳定的消息 m 最多提交到一个进程
- 不可能出现孤儿进程

**乐观型(optimistic logging)日志协议**

- 在崩溃后处理， 将 DEP(m)中的孤儿进程都回滚到不再属于 DEP(m)的状态