并行与分布式计算 - June 26’th, 2018

Coordination and Agreement Concepts

`https://github.com/rh01`

Reliable Communication ⇒ 可靠通信

- 可靠通信用通信的有效性和完整性定义：
  - 有效性 (validity, liveness)：在外发消息缓冲区的任何消息最终能被传递到接收消息缓冲区
  - 完整性 (integrity, safety)：接收到的消息与发送的消息一致，没有消息被传递两次
  - 对完整性的威胁
    - 通信协议允许重发消息，或不拒绝到达两次的消息
    - 可能有恶意用户插入伪造消息、重放旧的消息或窜改消息

可靠组播

- 组播是不可靠的
  - 组播会遭遇遗漏故障。
    1. 任何一个接收者都可能因为它的缓冲区满了而丢弃消息。
    2. 从一个组播路由器发送到另一个路由器的数据报也可能丢失
  - 组播路由器会出现故障
- 组播应用需要可靠组播
  - 举例：复制服务，要求或者所有的服务器或者没有服务器接收到一个操作请求
- 封闭组/开放组：组播发生在组的成员内/组外的进程也可以组播到它
- 重叠组：一个进程属于两个组
- 静态组/动态组：组成员不变/可变

可靠组播的性质

- 完整性 (Integrity, safety)：一个正确的进程 p 传递一个消息 m 至多一次。其中，p group(m)，m 由 sender(m) 提供
- 有效性 (Validity, liveness)：如果一个正确的进程组播消息 m，那么它终将传递 m
- 协定 (Agreement, liveness)：如果一个正确的进程传递消息 m，那么在 group(m) 中的其它所有正确的进程终将传递 m。
  - 有效性和协定一起得到一个全面的活性要求
  - 协定条件与原子性相关

System model

- Processes communicate reliably over one-to-one channels
- Processes may fail only by crashing

- The members of a group are known without any change of members
- Processes are allowed to belong to several closed groups
- multicast(g,m) sends the message m to all members of the group g of processes.
- deliver(m) delivers the message m sent by multicast to the calling process
- Every message m carries the unique identifier of the process sender(m) that sent it , and the unique destination group identifier group(m)

Implement reliable multicast R-multicast over B-multicast

Algorithm 1 Implement reliable multicast R-multicast over B-multicast

```
On initialization:
    Received:={}
For process p to R-multicast message m to group g:
    B-multicast(g,m);
    //p ∈ g, p is included as a destination
On B-deliver(m) at process q with g=group(m):
    If (m ∉ Received)
    then
        Received := Received ∪ m
        If(q ≠ p) then B-multicast(g,m); endif
        R-deliver m;
    endif
```

关于协定的证明

- 协定的定义：如果一个正确的进程传递消息 m，那么在 group(m) 中的其它所有正确的进程终将传递 m。
- 反证：设在 group(m) 中的一个正确的进程 A 没有传递 m (即：R-deliver m)，那么，这只能是因为它从没 B-deliver 此消息。而且，也没有其它正确的进程 B-deliver 此消息 (如果有，那么会发 B-multicast 给 A，A 就会 B-deliver 该消息)。因此，没有进程会 R-deliver 消息 m。这个结果与前提“一个正确的进程传递消息 m”矛盾。所以，得证。

基于 IP 组播实现的可靠组播的性质

- 完整性成立，基于检测副本和 IP 组播性质 (使用校验和来除去损坏的消息)
- 有效性成立，因为 IP 组播具有该性质
- 协定成立，在下列假设下：
  - 每个进程都无限组播消息，这时，该算法能保证检测到丢失的消息
  - 进程无限地保留它们已传递消息的副本

组成员管理

- 提供组成员改变的接口
- 提供失效检测
- 组成员改变时通知成员：传递组视图

- 执行组地址扩展
- 对组视图传递的要求：
  - 顺序 (order)：如果一个进程 p 传递了视图 v(g)，然后传视图 v’(g)，那么不存在这样的进程 q p，它在 v (g) 之前传递 v’(g)。
  - 完整性 (integrity)：如果进程 p 传递了视图 v(g)，那么 p   v(g)。
  - 非平凡性 (Non-triviality)：如果进程 q 加入到一个组中，并且对于 q 来说变为从进程 p q 无限期地可达的话，那么最终 q 总是在 p 传递的视图中

视图同步的组通信：性质

- 协定：正确的进程传递相同序列的视图 (从加入组的视图开始)，并且在任何给定的视图中传递同样的消息集合。换句话说，如果一个正确的进程在视图 v(g) 中传递了消息 m，那么所有其他传递 m 的正确的进程都在视图 v(g) 中传递 m。
- 完整性：如果一个正确的进程 p 传递消息 m，那么它不会再传递 m，而且，p ∈ group(m)，并且发送 m 的进程处于 p 传递 m 的视图中。
- 有效性 (封闭组)：正确进程总是传递它们发送的消息。如果系统在向进程 q 传递消息时发生了故障，那么它将传递一个删除了 q 的视图给余下的进程。也就是说，设 p 是一个正确的进程，它在视图 v(g) 中传递消息 m。如果某个进程 q ∈ v(g) 没有在视图 v(g) 中传递 m，那么在 p 传递的下一个视图 v’(g) 中将有 q ∉ v’(g)。

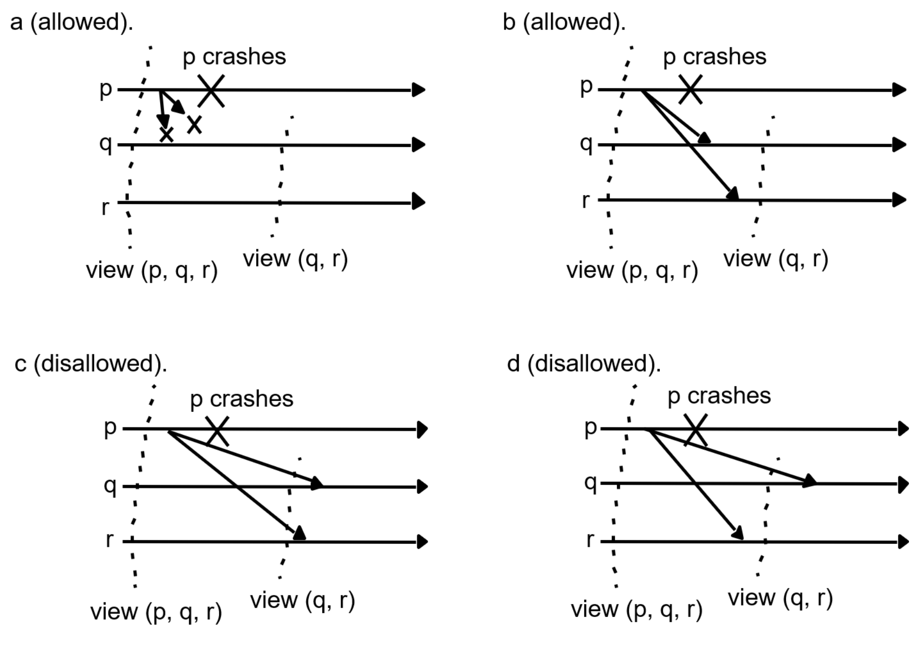


图 1：Example

Agreement in Fault Systems

- Agreement: Agree on a value after one or more of the processes have proposed what that values should be under the circumstance that processes may fail.
- Consensus, Byzantine generals and interactive consistency are referred to problems of agreement.
- Agreement is needed in many cases.

- Have all the non-faulty processes reach consensus with in finite number of steps.
- Problems
  - errors in process: crash failure or Byzatine (arbitrary) process failure

共识问题

- 定义：每个进程 pi 开始于一个未决状态，并且提议集合 D 中的一个值 vi (i = 1, 2, ..., N)。进程之间互相通信，交换值，做决定，然后，每个进程设置一个决定变量 di (i = 1, 2, ..., N) 的值。di 可以是用大多数进程提议的值，也可以用进程提议值中的最小值或最大值
- 对共识算法的要求：
  - 终止性：每个正确进程最终设置它的决定变量
  - 协定性：所有正确进程的决定值都相同：如果 pi 和 pj 是正确的并且已进入决定状态，那么 di = dj (i, j = 1,2, ..., N)
  - 完整性：如果正确的进程都提议相同值，那么处于决定状态的任何正确进程已选择了该值
- 解决共识问题等同于解决可靠全排序组播

Consensus in a synchronous system

- Assuming that up to f of the N processes exhibit crash failures
- Algorithm for process pi ∈ g, which need to proceed in f+1 rounds

Algorithm 2 Consensus in a synchronous system

```
On initialization:
    $Values_i^1$:={$V_i$}; $Values_i^0$={};
In round r (1 ≤ r ≤ f+1):
    B-multicast(g, $Values_i^r$ -$Values_i^{r-1}$);
    //Send only values that have not been sent
    $Values_i^{r+1}$ : = $Values_i^r$
    While(in round r)
    {
        on B-deliver($V_j$) from some $p_j$:
        $Values_i^{r+1} := Values_i^{r+1} \cup V_j$ ;
    }
After (f+1) round:
    Assign $d_i$ = minimum($Values_i^{f+1}$);
```

三个拜占庭将军

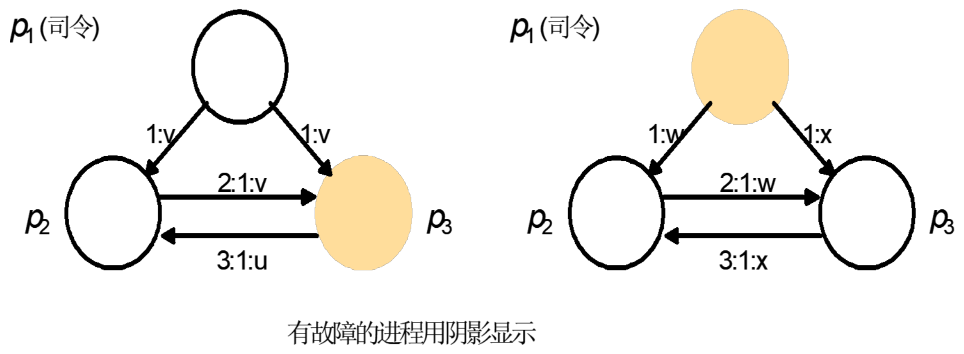


图 2：三个拜占庭将军

- 三个拜占庭将军如果有一个出错，就不能达成协定
- N 个将军，f 个出错，如果 N<3f，就不可能有解决方法

四个拜占庭将军

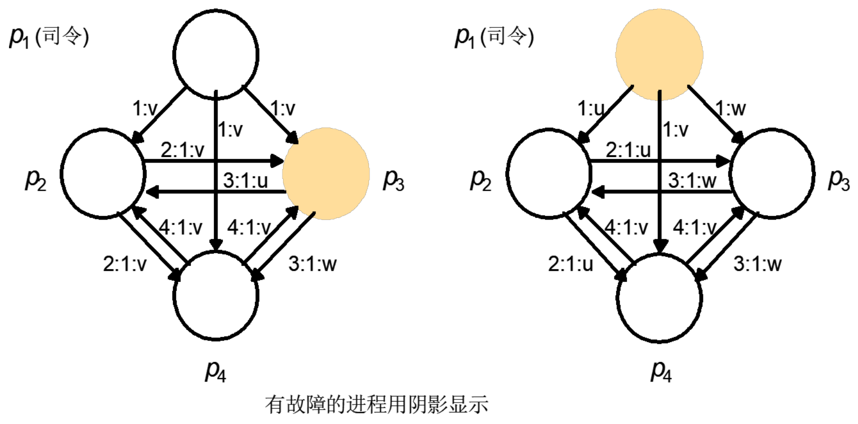


图 3：4 个拜占庭将军

- 四个拜占庭将军如果有一个出错，能达成协定
- 当出现左边情况时，两个正确的中尉进程在决定司令的值时达成一致：
  - p2 决定 majority(v,u,v) = v; p4 决定 majority(v,v,w) = v
- 在右边的情况中，司令是有错的，但是正确的 3 个中尉进程能达成一致：
  - p2 p3 和 p4 决定 majority(v,u,w)= （特殊值　代表没有占多数的值存在）

Byzantine generals problem

- Algorithm OM(0)
  - the commander sends his value to every lieutenant.
  - each lieutenant uses the value he receives from the commander, or uses the value UNKNOWN if he receives no value.
- Algorithm OM(m), m>0
  - the commander sends his value to every lieutenant.
  - For each i, let vi be the value Lieutenant i receives from the commander, or else be UNKNOWN if

he receives no value. Lieutenant i acts as the commander in Algorithm OM(m-1) to send the value vi to each of the n-2 other lieutenants.
  - For each i, and each j $\neq$ i, let vj be the value Lieutenant i received from Lieutenant j in last step (using Algorithm OM(m-1)), or else UNKNOWN if he receives no such value. Lieutenant i uses the value majority(v1, v2,⋯,vn-1).
- Algorithm OM(m) involve sending up to (n-1)(n-2)⋯(m-m-1) messages.

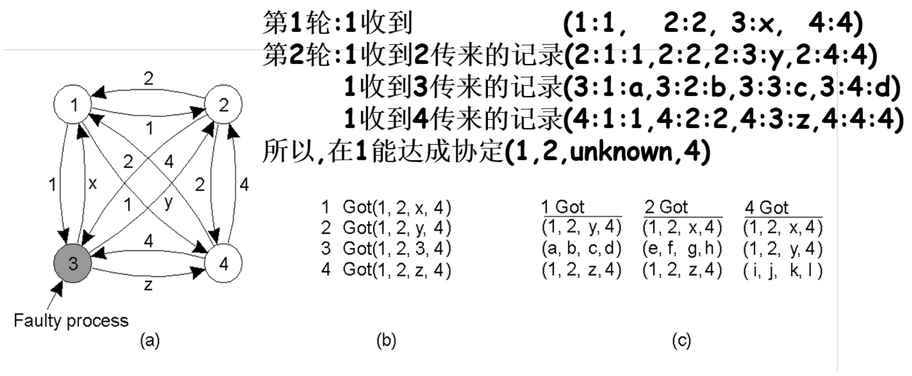
交互一致性问题 (拜占庭将军问题的推广)

- 每个进程都提供一个值。正确的进程最终就一个向量达成一致，向量中的分量与一个进程的值对应。这个向量称为“决定向量”。例如，可以让一组进程中的每一个进程获得相同的关于该组中每一个进程的状态信息
- 算法必须具有的性质：
  - 终止性：每个正确进程最终设置它的决定变量
  - 协定性：所有正确进程的决定向量都相同
  - 完整性：如果进程 pi 是正确的，那么所有正确的进程都把 vi 作为它们决定向量中的第 i 个分量

交互一致性算法

- n 个进程，可能 k 个出错，递归过程 IC(r)
- 初始时，递归次数 r=0，发送者列表 S={}
  1. 发送者将它的值连同发送者列表 S 发送给不在 S 中的进程，共发送 (n-1-r) 次
  2. 设 vi 是进程 Pi 从发送者接收到的值，没有收到值时则使用缺省值
     - 若 r+1<k，则调用 IC(r+1)，将进程 Pi 作为发送者，将 vi 连同发送者列表 S Pi 发送给其它不在 S Pi 中的 (n-2-r) 个进程。
     - 若 r+1=k，则 Pi 作为发送者将 vi 发送给其他 n-k-1 个进程，每个进程使用接收到的值，或者缺省值。
  3. 对每个进程 Pi ，vi 是从发送者接收到的值，vi,j 是发送者通过进程 Pj 转发给 Pi 的值，更新节点使用值

$$v_i = majority(v_i, v_i, j)$$



第**1**轮:**1**收到 **(1:1, 2:2, 3:x, 4:4)**
第**2**轮:**1**收到**2**传来的记录**(2:1:1,2:2,2:3:y,2:4:4)**
1收到**3**传来的记录**(3:1:a,3:2:b,3:3:c,3:4:d)**
1收到**4**传来的记录**(4:1:1,4:2:2,4:3:z,4:4:4)**
所以,在**1**能达成协定**(1,2,unknown,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)

图 4：Generalized Byzantine Generals Problem1

- The Byzantine generals problem for 3 loyal generals and1 traitor.
  - The generals announce their troop strengths (in units of 1 kilo-soldiers).
  - The vectors that each general assembles based on (a)
  - The vectors that each general receives in step 3.
  - Generals 1,2and 4 all come to the agreement on (1,2,UNKNOWN,4)
- The same as in previous slide, except now with 2 loyal generals and one traitor.
- Conclusion:agreement is possible only if more than two-thirds of the process are working properly.

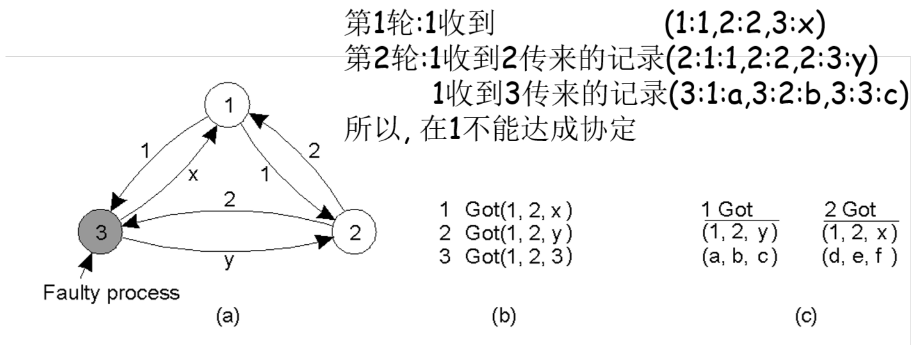


图 5：Generalized Byzantine Generals Problem2

若干结论

- 拜占庭将军、共识问题、交互一致性，从任一个的解决方案中可以构造出另外两个问题的解决方案
- 在存在崩溃故障的系统中，解决共识问题等同于解决可靠且全排序组播，给定其中一个问题解决方案，就可以解决另一个问题
- 同步系统中的共识问题：如果要在至多 f 个进程崩溃/拜占庭故障的情况下仍然能够达到共识，那么，必须要进行 f+1 轮的信息交换
- 同步系统中的拜占庭将军问题：
  - 3 个进程相互之间发送 (未签名) 消息，如果允许一个进程出现故障，那么没有办法能够保证满足拜占庭将军问题的条件。
  - 如果总共有 3f+1 个进程，其中有 f 个进程有错 (随机故障)，2f+1 个进程正确，那么就能达成一致。换句话说，只有多于 2/3 的进程工作正常，才有可能达成协定。否则，没有解决方法

异步系统的不可能性

- 在一个异步系统中，即使是只有一个进程出现崩溃故障，也没有算法能够保证达到共识
- 在异步系统中，没有可以确保的方法来解决拜占庭将军问题、交互一致性问题或者全排序可靠组播问题
- 绕过不可能性结论的三个方法
  - 故障屏蔽
    * 事务系统使用持久储存保存信息
  - 利用故障检测器达到共识
    * 即使是使用不可靠的故障检测器，只要通信是可靠的，崩溃的进程不超过 N / 2，那么异步系统中的共识是可以解决的

    – 随机化进程各方面的行为，使得破坏进程者不能有效地实施他们的阻碍战术

Recovery

Checkpointing

- Depending on the moment and frequency of checkpointing and amount of information saved in a checkpoint, there are different types of checkpoint algorithms.
    – Synchronous checkpointing, i.e., coordinated checkpointing, all processes synchronize to jointly write their states to local stable storage.
    – Asynchronous checkpointing: independent checkpointing at each process.
        * Uncoordinated: completely independent action.

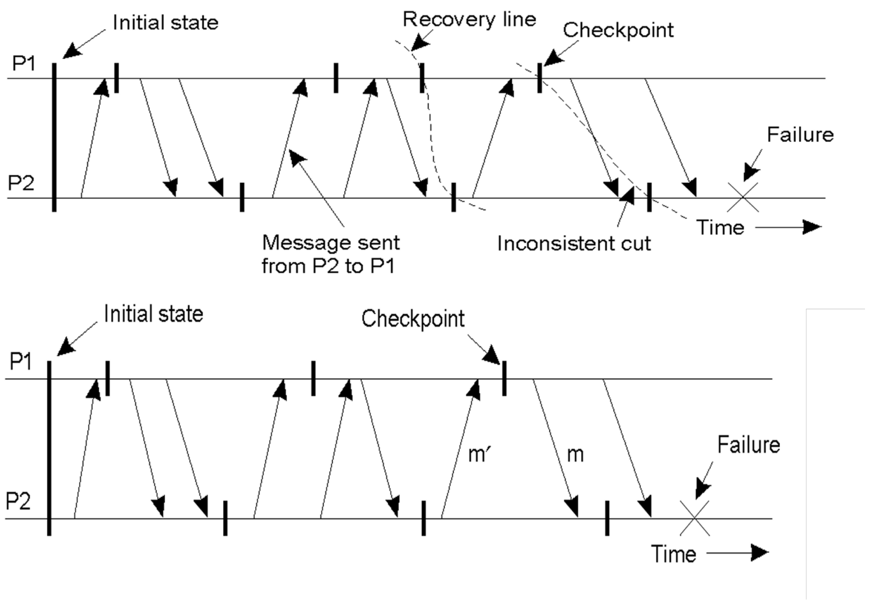


图 6：Checkpointing

Coordinated Checkpointing

- Use a two-phase blocking protocol: a coordinator multicasts checkpoint_request, processes do local checkpoints, queue any subsequent message handed to them by the application they are executing and then send acknowledges to the coordinator, the coordinator multicasts checkpoint_done.
    – Incremental snapshot: the coordinator multicasts checkpoint_request to those processes that depend on the recovery of the coordinator. In other words, the coordinator multicasts a checkpoint request only to those processes it had sent a message to since it last took a checkpoint. When a process P receives such a request, it forwards it to all those processes to which P itself had sent a message since the last checkpoint, and so on.
- Non-blocking method
- distributed snapshot algorithm (such as Chandy-Lamport’s algorithm) can be used to coordinate checkpointing

Independent Checkpointing

- CP[i](m)：进程 Pi 做的第 m 次检查点
- INT[i](m)：表示在 CP[i](m-1) 和 CP[i](m) 之间的时间间隔。
- Pi 要回退到 CP[i](m-1)：要保证在 INT[i](m) 中，从 Pi 接收到
- 消息的进程，要回滚到接收到消息之前的那个检查点状态

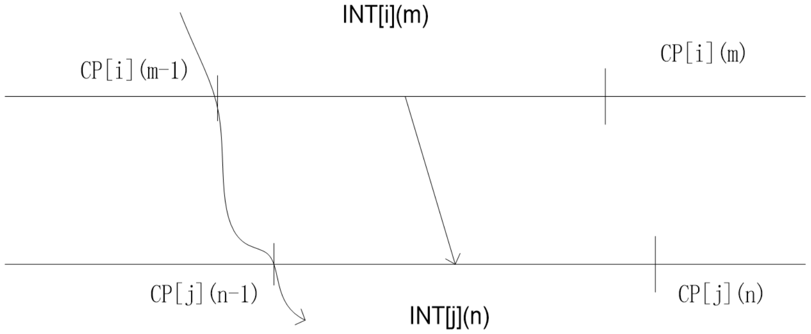


图 7：Independent Checkpointing

- After a failure, the failed process Q recovers by restoring its most recent checkpoint.
  - If Q does not send out any message after its most recent checkpoint, recovery is completed.
  - If Q sends out a message m to a process P, then message m has been received by P but not been sent by Q (after Q restores its checkpoint). This is inconsistent. So Q has to send a message to all processes it has sent a message, and ask them to roll back.
  - When P is requested to roll back, it rolls back by restoring to its most recent checkpoint.
  - If P has sent out messages to other processes, those other affected processes must roll back as well.
- Domino effect is hard to avoid.

Logging Messages for Replaying

- When Q recovers from crash, it restores its most recent checkpoint, and re-executes its program, replaying the log (reading messages received from the log)
  - Under the assumption of a piecewise deterministic model: the execution of the process is deterministic after the receipt of a message (e.g. no clock or random number is used in the program)
- Pessimistic message logging: messages are saved before processing
- Optimistic message logging: independent messages saving / processing

Message Logging: When messages are to be logged

- Orphan process: a process that survives the crash of another process, but whose state is inconsistent with the crashed process after its recovery.
- Process R is an orphan process if there is a message m, such that R is contained in DEP(m), while at the same time every process in COPY(m) has crashed
- DEP(m): processes to which m has been delivered

- COPY(m): process that could hand over a copy of m that can be used to replay the transmission of m.

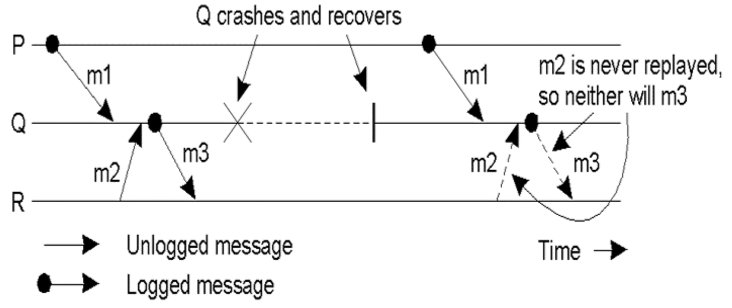


图 8：Message Logging: When messages are to be logged.Incorrect replay of messages after recovery, leading to an orphan process R.

Message Logging: Pessimistic logging protocols

- To avoid orphan process, whenever a process becomes dependent on the delivery of m, it will keep a copy of m.
- Pessimistic logging protocols: Q is not allowed to send any messages after the delivery of m without first having ensured that m has been written to stable storage.
- When Q recovers, it restores its most recent checkpoint, and reexecutes its program, replaying the log.
- No domino effect is possible in pessimistic message logging.
- Optimization
  - If the message m can be deterministically reproduced by Q when it recovers, and P is capable of filtering out the duplicated message m sent by Q after recovering, then P does not need to roll back, since m will be regenerated after Q recovers.

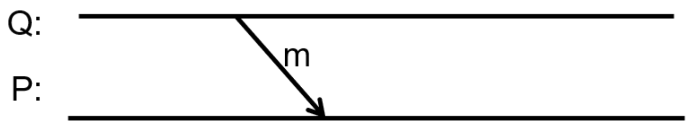


图 9：Message Logging: Pessimistic logging protocols

Message Logging: Optimistic logging protocols

- Message logging can also be asynchronous with respect to processing. This is optimistic message logging. Reason is that logging messages before processing cause a long delay due to disk I/O.
- Frequency of logging messages displays a spectrum:
  - Logging a message whenever it is received will approximate pessimistic message logging; batching messages and logging them just before a checkpoint is like without message logging.
- An unlogged message m received by a failed process P may cause P unable to recover. This makes all messages sent by P after receiving m orphans at other processes. How can P know that it has sent messages after m? P needs to keep track of the processes which depend on m.These processes have to roll back. Any orphan process dependent on m is rolled back to a state in which it no longer dependent on m

- Domino effect is possible, but is restricted.

How to judge a consistent global state (1)

- Recall the definition of a consistent global state: a collection of local states, one from each process, such that no message is received but has not been sent in the global state (no orphan message)
- We may apply the vector clock concept to the recovery model . For a system of n processes, each event is associated with a vector clock of size n.
- Denote S1 by matrix, called the dependency matrix, same for S2.
- Each row of the matrix is formed by vector clock of local state of a process constituting the global state.
- Thus $D(S1) = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 4 \end{bmatrix}$ and $D(S2) = \begin{bmatrix} 2 & 1 & 0 \\ 4 & 3 & 4 \\ 5 & 1 & 6 \end{bmatrix}$
- Rule: state S is consistent if and only if all diagonal elements dominate the columns.

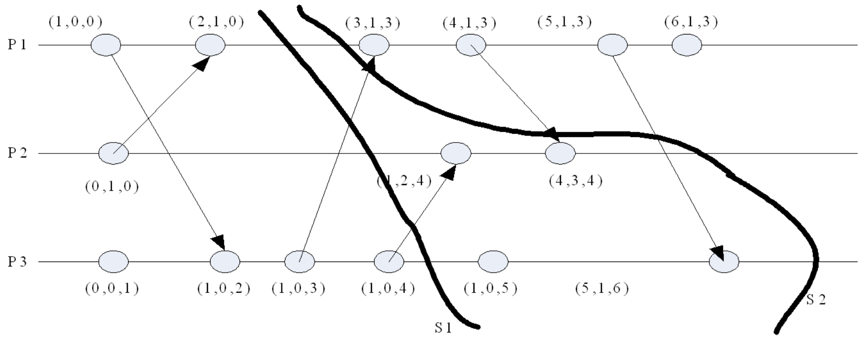


图 10：consistent global state

- Diagonal of the matrix counts the number of events each process has executed, up to global state S.
- A state is consistent if the process knows the best about itself than other processes.
- In the example:
  - You can verify that the diagonal in D(S1) dominates the columns, i.e. diagonal is the largest element along each column.
  - This means that S1 is consistent.
  - For state S2, it can be seen that the diagonal no longer dominates the columns.
  - The largest element of first column is not on the diagonal.
  - Thus S2 is inconsistent. Reason: P3 knows of 5 events from P1, but P1 only executes 2.

State Interval Model for Recovery

- Assuming deterministic process execution and message logging, vector clock can be modified for state intervals instead of for events.
- State interval
  - A state interval is a sequence of events on a process.

  - A state interval is started by receipt of a message.
  - Each state interval has a sequence number, called the state interval index.
  - A state interval ends when a new message is received, including the receiving event of current message but excluding the new message.

- Each message carries the state interval index and each state interval is associated with a dependency vector C, which is like a vector clock for an event.
- When a message from Pj is received at Pi, Ci[j] will be incremented by 1.
- S1 and S2 can be denoted by the dependency matrices.
- Thus D(S1)= $\begin{bmatrix} 1 & 0 & \perp \\ \perp & 0 & \perp \\ 0 & \perp & 1 \end{bmatrix}$ and D(S2)= $\begin{bmatrix} 1 & 0 & \perp \\ 2 & 2 & 1 \\ 2 & \perp & 2 \end{bmatrix}$
- The diagonal of the matrix counts the number of messages each process has received, up to the global state. Since only receiving events are relevant, some global states are considered identical in this model.
- In D(S1), the diagonal dominates the columns, meaning that S1 is consistent, or will be consistent on deterministic recovery.
- In D(S2), the diagonal no longer dominates the columns. Thus S2 is not consistent, and there is an orphan message.
- The diagonal corresponds to a global state.
- A global state in which the diagonal dominates the columns is called a consistent global state.

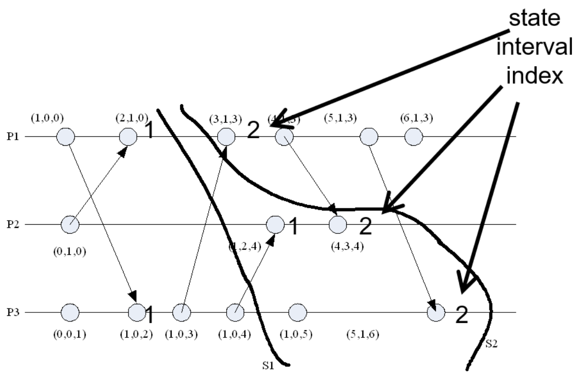


图 11：consistent global state

- In optimistic message logging, messages are logged asynchronously with respect to processing, and checkpoints are taken occasionally.
- A state interval is stable if all messages received between its most recent checkpoint and its starting message are logged.
  - It is easy to see that a stable state interval can be reproduced from the disk when the process recovers after crashing (by restoring the most recent checkpoint and replaying messages from the log).
- A global state is recoverable if all state intervals in the state are stable.
- It is required that upon recovery after a failure, a consistent recoverable global state is restored.
- State interval model optimizes in reducing message overhead from a vector to a single number, and the size of counter.

- It also provides a basis for processes to detect orphan messages and to perform appropriate rollback recovery.

Forward error recovery: Stabilization

- The set of all possible configurations or behaviors of a distributed system can be divided into two classes: legal, illegal.
  - The legal configuration of a nonreactive system is usually represented by an invariant over the global state of the system.
  - In reactive systems, legal configuration is determined not only by a state predicate, but also by behaviors.
- Well-behaved systems are always in a legal configuration, but such a system may switch to an illegal configuration due to the following reasons: transient failures, topology changes, environmental changes.
- A system is called stabilizing when the following two conditions hold:
  - Convergence: regardless of the initial state, and regardless of eligible actions chosen for execution at each step, the system eventually returns to a legal configuration.
  - Closure. Once in a legal configuration, the system continues to be in the legal configuration unless a failure or a perturbation corrupts the data memory

A stabilizing algorithm for constructing a spanning tree

- Assume that failures do not partition the network and now construct from a connected undirected graph G=(V, E) (where |V| = n) a spanning tree rooted at r with each node knowing its level in the tree.
- Each node i other than the root r maintains the two local variables: L(i): level of i; P(i): parent of i. The root node r has L(i) =0 and no parent variable.
- Legal state: if the parent pointers constitute a spanning tree of G rooted at the root, an each node except the root has a level equal to the level of its parent plus one.
- If the following predicate is true, then the system reaches a legal state:

$$GST \equiv (\forall i, p : i \neq r \wedge p = P(i) : L(i) = L(p) + 1)$$

A stabilizing algorithm

- (R0) $L(i) \neq n \wedge L(i) \neq L(p) +1 \neq L(p) \neq n \rightarrow L(i) := L(p) +1$
- (R1) $L(i) \neq n \wedge L(p) = n \rightarrow L(i) := n$
- (R2) let k be some neighbor of i, $L(i) = n \wedge L(k) < n-1 \rightarrow L(i) := L(k)+1;\ P(i) := k$

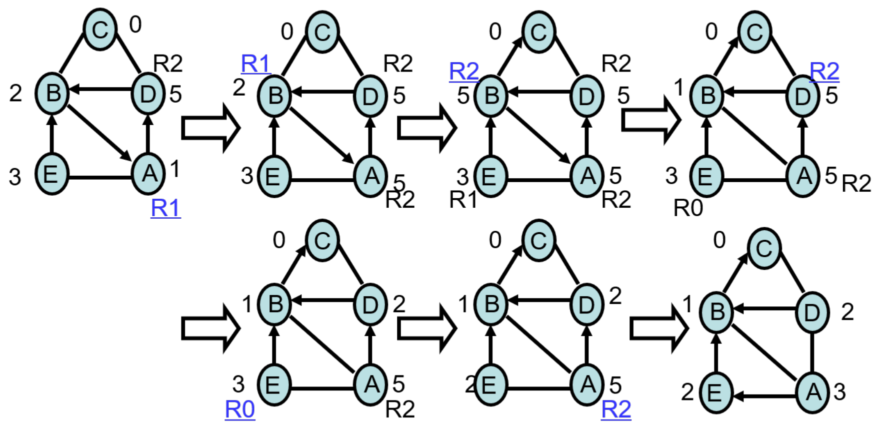
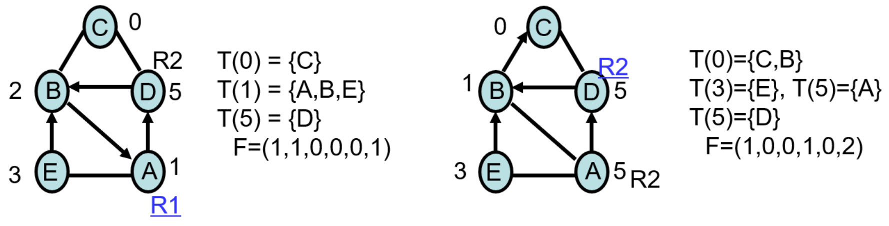


图 12：stabilizing algorithm

- Define an edge from i to P(i) to be well formed, when L(i) $\neq$ n $\wedge$ L(p(i))$\neq$n $\wedge$ L(i) = L(p(i)) +1.
- In any configuration, the nodes and well-formed edges form a spanning forest.
- Delete all edges that are not well formed, and designate each tree in the forest by T(k) where k is the smallest value of a node in the tree.
  - A single node with no well-formed edge incident on it represents a degenerate tree.
- Key is to find a bounded function whose value decreases for each move: Define a tuple F=(F(0),F(1),F(2), ⋯, F(n)) such that F(k) is the count of T(k)’ s in the spanning forest.


T(0) = {C}
T(1) = {A,B,E}
T(5) = {D}
F=(1,1,0,0,0,1)

T(0)={C,B}
T(3)={E}, T(5)={A}
T(5)={D}
F=(1,0,0,1,0,2)

图 13：stabilizing algorithm2

- A lexicographic order(>) on F is defined as follows:

$$F1 > F2 \rightarrow \exists j > 0 : \forall i < j : F1(i) = F2(i) \wedge F1(j) > F2(j).$$

- With n nodes, the maximum value of F is (1, n-1, 0,⋯,0) and the minimum value is (1,0,0,⋯,0) that represents a single spanning tree rooted at 0.
- F monotonically decreases each time when rule (R0),(R1) or (R2) is applied.
  - R0：F(i) 减少，F(j) (j<i) 不减少
  - R1：F(n) 增加，F(i) 减少，F(j) (j<i) 不受影响
  - R2：F(n) 下降，其他 F 不受影响