Fundamentos de los Algoritmos Distribuidos

Pablo Galdámez Saiz

Universidad Politécnica de Valencia

DSIC

Objetivos

1. Conocer las características que diferencian a los algoritmos distribuidos, respecto a otros tipos de algoritmos.
2. Entender el modelo de computación distribuida.
3. Conocer la importancia de los formalismos y modelos para diseñar algoritmos distribuidos
4. Familiarizarse con un formalismo que permita diseñar algoritmos y razonar sobre su corrección

Contenidos

I. Sistemas Distribuidos y Algoritmos Distribuidos: Conceptos Fundamentales

II. Formalismos para modelar algoritmos distribuidos: Autómatas de Entrada/Salida

III. Ejemplos de Algoritmos Distribuidos

Bibliografía

1. **Distributed Algorithms. Nancy Lynch.** The Morgan Kaufmann Series in Data Management Systems) 1st Edition
2. Apuntes de la asignatura

Evaluación

1. Trabajo de la asignatura: Diseño de un algoritmo distribuido, prueba de corrección. (80%)

2. Examen online de tipo test V/F (20%)

Ambos al final del curso. A completar antes del 20 de febrero.

Fundamentos de los Algoritmos Distribuidos

Pablo Galdámez Saiz

Universidad Politécnica de Valencia

DSIC

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.1 Concepto de Sistema Distribuido

- Conjunto interconectado de entidades autónomas
  - Computadoras
  - Procesadores
  - Procesos
    - *Nodos* del sistema distribuido

Definición (cont)

- Autónomo
  - Control privado
  - Decisiones independientes del resto de componentes
- Interconectado
  - Debe ser posible que los componentes intercambien información
  - *Debe intercambiarse información* en algún momento para hablar de sistema distribuido

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.2 Motivación

- Intercambio de información
- Compartición de recursos
- Fiabilidad
- Rendimiento
- Modularización y especialización

I.2.1 Intercambio de Información

- Facilitar el intercambio de información
  - Entre computadores diferentes
    - Inicialmente *Mainframes* en WANs
    - *Workstations* en redes corporativas
    - En un cluster
    - En sistemas en la nube, en grid, en p2p….
    - Sistemas móviles, sensores...
  - — Generalizar la comunicación entre componentes
    - Algoritmos distribuidos y protocolos son necesarios para implementar los conectores entre componentes

I.2.2 Compartición de Recursos

- Los recursos son caros
- Un nodo o varios disponen de recursos. Los demás nodos quieren acceder a los recursos de forma remota.
- Varios nodos acceden simultáneamente: implicaciones de rendimiento, concurrencia y seguridad.

I.2.3. Fiabilidad

- La posibilidad de tener computadoras con funcionalidad redundante permite a priori soportar fallos parciales del sistema.
  - Posibilidad real de conseguir una mayor disponibilidad de las funciones implantadas.
  - El fallo de algún componente puede ser manejado transparentemente por el software del sistema.

I.2.3. Fiabilidad (cont)

- Necesidad de mecanismos de manejo de fallos en sistemas distribuidos con interdependencias entre componentes.
  - La probabilidad de fallos aumenta
  - La robustez de las aplicaciones se ve comprometida
    - Detección de fallos
    - Compromiso distribuido
    - Recuperación de los fallos

I.2.4. Mayor rendimiento

- Más computadores implica mayor poder de cálculo.
- Potencialmente, algunas computaciones pueden realizarse en menor tiempo.
  - Sistemas multicomputadores.
  - Redes de estaciones de trabajo (NOWs)
  - Clusters de supercomputación
    - Software de simulación de multicomputador (PVM, …)
  - Computación en la nube, grid.

I.2.5. Modularización

- Especialización de la función de cada ordenador
  - Mayor simplificación conceptual del diseño.
- Nodos especializados:
  - Servidor de nombres
  - Servidor de correo, etc.
- *Network appliances*:
  - Discos de red.
  - Cortafuegos.

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.3.- Problemas algorítmicos. Algunos ejemplos.

- Problemas en redes de comunicación
- Problemas en multicomputadores
- Problemas en sistemas de procesos cooperantes
- Problemas comunes
- … problemas en cloud, grid, p2p, sensores, etc.

I.3.1. Problemas algorítmicos en redes de comunicación

- Red de comunicaciones
  - Conjunto de computadoras conectadas
  - Mecanismos de interconexión
- Envío y recepción de mensajes
- División típica entre redes LAN y WAN, con algunas diferencias típicas
- Problemas algorítmicos motivadores

I.3.1... WANs: Algunos problemas algorítmicos

- Encaminamiento
- Fiabilidad
- Control de congestión
- Prevención de interbloqueo
- Seguridad

I.3.1... WAN: Encaminamiento

- Store-and-forward: Topología conexa no total
  - Nodos *a* y *b* están directamente conectados
  - Nodo *a* precisa pasar por *b* para comunicar con *c*
- Necesario un algoritmo de encaminamiento
  - Encuentra el “mejor” camino entre dos nodos a y b
    - Adaptable (i.e., soportando desconexiones)
    - Minimizando algún parámetro
    - Posible uso de nominación para encaminar
      - Problema genérico de la localización de recursos

I.3.1... Fiabilidad

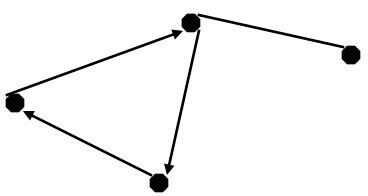
- Fiabilidad del paso de mensajes.
  - Una línea de comunicación puede
    - Corromper la información
    - Desconectarse
    - Perder información
  - Generalmente resultado de las características físicas del medio
  - También relevante en canales virtuales

I.3.1... Control de congestión

- El *Throughput* de una red depende de la inyección de mensajes en la misma.
  - A partir de un cierto número de mensajes, un aumento del mismo produce una disminución del throughput.
    - Congestión
  - Necesario un algoritmo que controle la inyección de mensajes en la red.
    - Hay que cuidar los efectos en la latencia

I.3.1... Prevención de interbloqueo

- Mecanismo de store-and-forward
  - Provoca intercambios de mensajes entre nodos vecinos.
  - Los nodos de reenvío tienen recursos finitos
  - Puede formarse un bucle de interbloqueo



I.3.1... Seguridad

- En una WAN, propietarios diferentes
- Los mensajes que pasan por un nodo para llegar a su destino pueden ser manipulados:
  - Inspeccionados (privacidad)
  - Modificados y suplantados
- Introducción de Viruses+Troyanos+Gusanos
- Necesidad de mecanismos de autenticación
- Políticas de acceso a recursos

I.3.1... LANs

- Generalmente concebidas como buses
- Gran ancho de banda y baja latencia
  - Ethernet (varios formatos)
  - Token Ring (anillo)
  - Myrinet (estrella, con conmutadores)
  - SCI (anillo o estrella, con conmutadores), memoria compartida

I.3.1... LANs: algunos problemas algorítmicos

- Algunos de los problemas anteriores se repiten
  - Encaminamiento: se simplifica
  - Fiabilidad: mucho mayor: se simplifica también
  - Congestión: simplificación (bus)
  - Interbloqueos: no existen al no utilizarse store-and-forward
  - Seguridad: desaparece (?) si todos son confiables

I.3.1... Redes de Comunicación: problemas algorítmicos generales

- Uso para la ejecución de programas
  - Difusión y Sincronización
  - Elección
  - Detección de finalización
  - Reserva y localización de recursos
  - Exclusión Mutua
  - Detección de interbloqueos y su resolución
  - Información compartida
  - Consistencia de datos

I.3.1... Difusión y sincronización

- Los nodos que participan en un algoritmo deben hacer llegar la información a todos los demás
- Todos deben saber cuando los demás han recibido cierta información
- Todos los nodos deben esperar a que alguna condición global sea satisfecha.

I.3.1.... Elección

- Algunos problemas se resuelven fácilmente cuando uno de los nodos realiza un trabajo específico:
  - Coordinar el trabajo de los demás
  - Realizar un cálculo concreto
- Es necesario ejecutar un algoritmo que elija el nodo responsable
  - Las características el sistema determinarán la dificultad de esta tarea.

I.3.1... Detección de finalización

- Caso especial de sincronización
- En muchos casos es necesario que los procesos determinan cuando una computación ha finalizado.
- El estado global “computación finalizada” no es observable directamente
  - Necesario un algoritmo para verificar el estado global de terminación.

I.3.1... Reserva de recursos

- Un nodo precisa recursos del sistema
- Los recursos se sitúan de un modo no preconocido para una aplicación
  - Movilidad de los recursos
  - Replicación de los recursos
- Necesario descubrir dónde están.
  - Sistemas de directorios
  - Servidores de nombres

I.3.1... Exclusión mutua

- Cuando varios procesos necesitan acceder a un recurso común.
  - Necesidad de dar paso tan sólo a uno
  - Necesidad de hacerlo de forma distribuida
    - Sin control centralizado a priori
- Condicionantes típicos del problema clásico
- Importantes los parámetros de rendimiento

I.3.1... Detección y resolución de interbloqueos

- Conjunto de recursos a ser accedidos por un conjunto de nodos
  - Los recursos son accedidos en orden arbitrario
- Se deben satisfacer condicionantes de exclusión mutua en el acceso a cada recurso
  - Se pueden formar cadenas de espera recíproca, dando lugar a interbloqueos.
- En general no se pueden evitar en S.D.
  - Hay que detectar y resolver

I.3.1... Gestión de información compartida (archivos)

- Nodos acceden a los ficheros con operaciones de lectura y escritura
- Debe mantenerse una visión consistente de los cambios en la información compartida
  - Memoria compartida distribuida
    - Modelos de consistencia de datos
      - Secuencial, causal, PRAM, cache, etc…
    - Caso particular: ficheros compartidos
      - Complicaciones con la mezcla de otros canales de comunicación

I.3.1... Mantenimiento de la consistencia de datos: Transacciones

- Contención de daños frente a fallos
- Una transacción es una unidad atómica de trabajo
- Si algo ocurre en medio de la ejecución de una transacción, el sistema alcanza un estado "consistente"
  - Imprescindible su uso para un manejo adecuado de la resolución de interbloqueos.

I.3.2. Problemas algorítmos en Multicomputadoras

- Múltiples computadoras compartiendo, por lo menos, una infraestructura de computación.
- Procesadores homogéneos.
- Escala geográfica pequeña.
- Incremento de velocidad (computadora paralela)
- Incremento de fiabilidad (sistema replicado)

I.3.2. Multicomputadoras (cont)

- Varios tipos
  - SIMD (Single Instruction Multiple Data) No SD
  - MIMD (Multiple Instruction Multiple Data)
    - Memoria compartida simétrica
      - Igualdad de tiempo de acceso a los datos desde cualquier nodo
    - Memoria compartida no uniforme
      - Cada nodo tiene una sección favorecida, aunque puede acceder a toda la memoria
    - Memoria distribuida independiente
      - Cada procesador accede tan sólo a su memoria

I.3.2. Multicomputadoras (cont)

- Evolución
  - Arquitecturas específicas
    - Hardware dedicado para la comunicación
      - Buses
      - Redes de topologías específicas
        - Transputer, CM-5, etc...
  - Redes de ordenadores (NOW)
    - Red local de altas prestaciones
    - Concepto de cluster
      - De computación paralela
      - De propósito general

I.3.2... Problemas algorítmicos

- Implementación del sistema de paso de mensajes
- Sistema de memoria virtual compartida
- Reparto de carga
- Robustez frente a fallos

I.3.2... Implementación de paso de mensajes

- Como en redes
  - Soluciones a menudo más simples
    - Asistencia por el hardware específico
    - Topología de comunicación conocida de antemano.
      - Mallas, Hipercubos
  - Encaminamiento en general simplificado.

I.3.2... Memoria Virtual Compartida

- Límite de escalabilidad limitado para la compartición física de memoria
- Mecanismos de simulación de compartición de memoria
  - Primeras aproximaciones: memorias caches
- Consistencia en el acceso a la memoria compartida
  - Desarrollo de modelos y algoritmos específicos a las características del medio de comunicación

I.3.2... Reparto de carga

- Una mayor velocidad por repartir carga
- Necesario un medio de medir la carga en recursos de cada nodo
  - Si un nodo degrada su rendimiento, ralentiza al resto
- En algunos casos, es posible realizar un reparto estático
  - Algoritmos inflexibles

I.3.2... Robustez

- Mecanismo para recuperación frente a fallos es necesario en un sistema replicado.
  - Se asume que los fallos no son detectables.

- Diferentes tipos de fallos
  - En particular, un nodo puede dar respuestas erróneas mientras sigue funcionando
    - Necesario establecer mecanismos de votos
      - Filtran resultados de procesadores fallidos

I.3.3. Problemas algorítmos en Procesos cooperantes

- Conjunto de procesos autónomos que cooperan para resolver un problema concreto
  - Distribución física forzosa.
    - El problema sólo tiene sentido dentro de un contexto físicamente distribuido
  - Distribución lógica
    - Conveniencia de expresar la solución como módulos autónomos
    - Puede dar lugar, con los mecanismos de comunicación adecuados, a una distribución física.

I.3.3... Problemas algorítmicos

- Mecanismos de comunicación entre los procesos
- En un procesador hay memoria compartida
  - Sincronización entre procesos
  - Atomicidad de acceso a memoria
  - El problema del productor-consumidor
  - Recolección de residuos

I.3.3... Mecanismos de comunicación entre procesos

- Memoria compartida
  - Amplia flexibilidad
  - Falta de estructura
  - No aporta sincronización
- Tubos
  - Modelo de “stream”: un extremo produce y el otro consume
    - Ampliamente utilizado en UNIX

I.3.3... Mecanismos de comunicación (cont)

- Paso de mensajes
  - Suelen servir para sincronizar al emisor y receptor
    - Eventos de envío y recepción quedan relacionados temporalmente
    - Variaciones en cuanto a la sincronización proveída
  - Mecanismo natural para sistemas físicamente distribuidos
  - Elaboraciones posteriores
    - RPC, Invocaciones a objetos, etc...

I.3.3... Sincronización entre procesos

- Semáforos
  - Operaciones P, V
  - Variaciones (binarios, contadores, etc…)
  - Apropiados para exclusión mutua
- Monitores
  - Mecanismo más sofisticado
  - Encapsula también datos y código, junto a las necesidades de sincronización

I.3.3... Atomicidad de acceso a memoria

- Se supone que la lectura/escritura de una variable es atómica
- Para estructuras mayores es necesario utilizar exclusión mutua
  - Complicado con memoria compartida
  - Exclusión mutua implica que un proceso debe esperar a que otro finalice el acceso, si el que espera es más prioritario se obtiene un mal resultado
    - Implementaciones de acceso atómico libres de espera

I.3.3... Productor consumidor

- Utilización de buffers para comunicar dos procesos: uno escribe, el otro lee
  - El tamaño del buffer se escoge de modo que se desacoplen los procesos
  - Los dos procesos deben coordinarse
    - El escritor no debe sobreescribir los datos
    - El lector debe esperar cuando el buffer está vacío

I.3.3... Recolección de residuos

- Es frecuente el uso de datos dinámicos que son finalmente desechados.
  - Los datos desechados quedan inaccesibles: *residuos*
  - En un principio, la ausencia de memoria provocaba la ejecución de un *recolector de residuos*
  - Es posible ejecutar la recolección al vuelo mediante un proceso recolector
    - Es necesario coordinar cada proceso con el recolector
- Problema relevante en sistemas Físicamente D.

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.4. Clasificación de los Sistemas Distribuidos

Ejes relevantes de clasificación:

- Topología
- Propiedades de los enlaces
- Conocimiento de los procesos
- Nivel de sincronía

Se deben hacer suposiciones aceptables

I.4.1. Topología

- Un SD es un grafo (dirigido, en general)
  - Anillos
  - Árboles
  - Estrellas
  - Hipercubos
  - …
- Estática o dinámica
  - En canales y nodos

I.4.2. Propiedades de los enlaces: Fiabilidad

- Relacionado con la topología
  - Si un enlace puede caer, la topología varía con el tiempo
- Pérdida de los mensajes
- Corrupción de los mensajes
  - Con checksums, equiparable a la pérdida
- Duplicación de mensajes
  - Como resultado de protocolos de recuperación

I.4.2. Propiedades de los canales: Ordenación

- Reparto ordenado local
  - En cada canal FIFO
- Reparto ordenado global
  - La intersección del conjunto de mensajes recibidos por cualquier par de nodos es recibida en el mismo orden en los dos nodos
- Reparto causal global
  - El orden de recepción de mensajes es causalmente consistente

I.4.2. Propiedades de los canales: Capacidad

- Un canal puede tener una capacidad máxima
  - Número máximo de mensajes en tránsito
- Si un canal está lleno dos cosas pueden pasar
  - La acción de envío no queda habilitada
    - Los procesos conocen el estado del canal
  - Se produce un envío con pérdida automática
    - Necesario si el estado del canal es desconocido por los procesos

I.4.3. Conocimiento de los procesos

- Identidad de los procesos
  - Cada proceso puede o no poseer una identidad única
    - Muchos algoritmos lo requieren
    - Cada proceso debe conocer dicha identidad
  - Adicionalmente se pueden relacionar suposiciones acerca del dominio de las identidades
  - Un sistema es *anónimo* si los procesos no tienen identidad o ésta no es única

I.4.3. Conocimiento de los procesos (cont)

- Conocimiento topológico
  - Topología global
    - Propiedades del grafo (i.e número de procesos), topología del grafo
  - Topología local: vecinos
    - Conocimiento de los nombres de los vecinos (conocimiento vecinal)
      - Direccionamiento directo de mensajes
    - Desconocimiento de la identidad de los vecinos
      - Direccionamiento indirecto: cada proceso usa nombres diferentes para enviar un mensaje al mismo proceso vecino
    - Desconocimiento del número de vecinos
      - Difusión

I.4.4. Nivel de sincronía

- Sistemas asíncronos
  - No hay garantías en el tiempo que un canal tarda en entregar un mensaje
  - No hay garantías en las velocidades relativas de ejecución de nodos diferentes
- Sistemas síncronos
  - Todo mensaje que se envía, se entrega en un tiempo máximo
  - La velocidad de proceso de los mensajes de todos los procesadores está en unos límites conocidos

I.4.5. Suposiciones aceptables

- Los ejemplos de sistemas distribuidos caen en varias de las clasificaciones anteriores.
- En la mayoría es difícil saber la velocidad de cómputo de cada nodo, así como las garantías de tiempo de entrega de la información
  - La entrega debe incluir la recogida de la misma
  - Los sistemas no anónimos asíncronos y fiables forman un modelo bastante realista
    - Soluciones para ellos demuestran la existencia de soluciones en sistemas menos restrictivos.

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.5 Algoritmos distribuidos

- Destinados a cubrir funciones en sistemas distribuidos
- El desarrollo de algoritmos distribuidos es esencialmente diferente de algoritmos centralizados
  - Diferencia derivada de la diferencia entre sistemas centralizados y distribuidos

I.5. Diferencias centralizados/distribuidos

- Desconocimiento del estado global
- Ausencia de un marco de referencia temporal global
- No determinismo

Consideraciones generales

I.5.1. Desconocimiento del estado global

- Estado de un sistema distribuido = conjunción de:
  - Estados de cada uno de los nodos
  - Estado del sistema de comunicaciones
- Cada nodo conoce su propio estado en cada instante
- El estado del sistema de comunicaciones no es conocido directamente por ningún nodo

I.5.1. Estado global (cont)

- *Conocer el estado* = ser capaz de tomar decisiones basadas en dicho estado
  - Evaluación de proposiciones basadas en dicho estado
  - Basar algoritmos en el valor instantáneo del estado

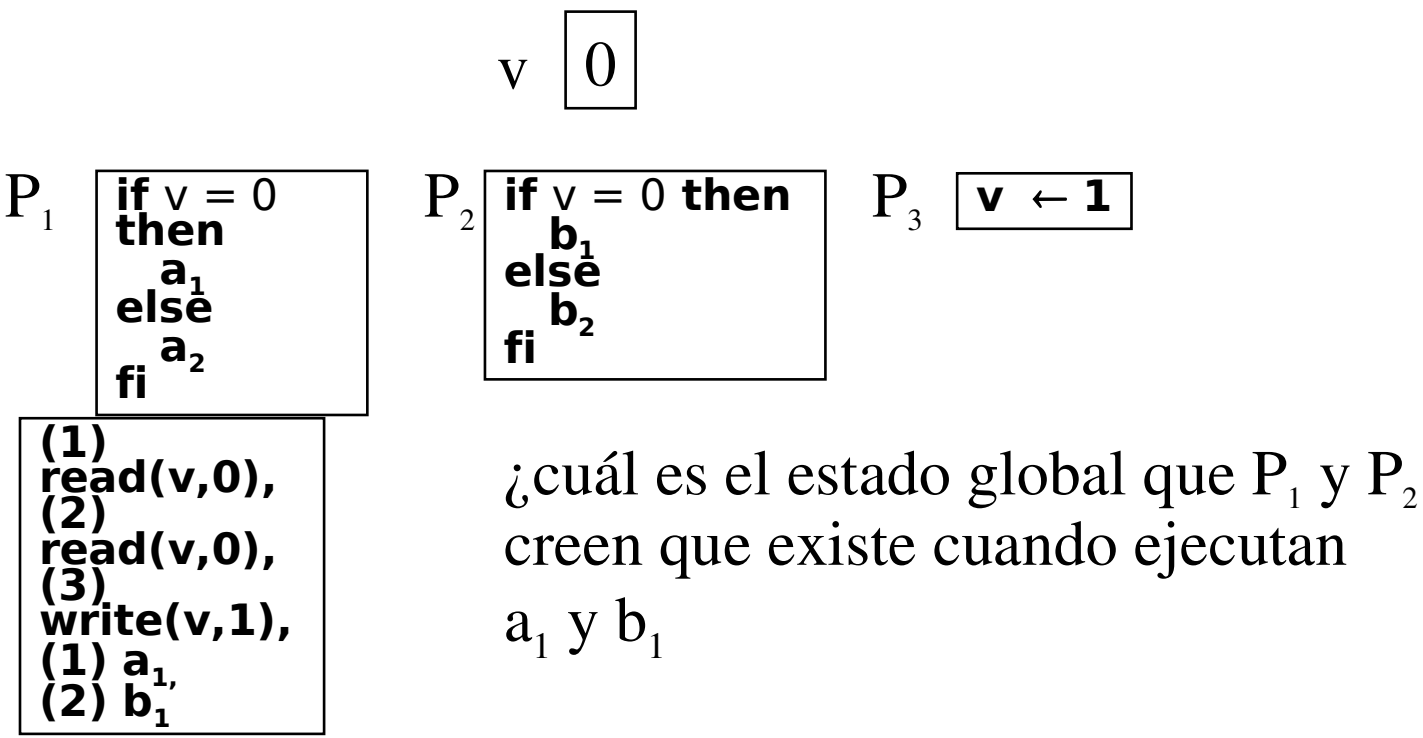
If *estado* == *S* **then** *acción_1* **else** *acción_2* **fi**

I.5.1. Estado global (cont)

- Cada nodo puede estimar
  - El estado de los demás nodos
  - El estado del sistema de comunicación
    - Canales de mensajes
      - A través de información transmitida y recibida de otros nodos, donde se da cuenta de los mensajes enviados por cada cual
    - Memoria compartida
      - A través de la lectura de la misma, y su depósito en memoria local al nodo

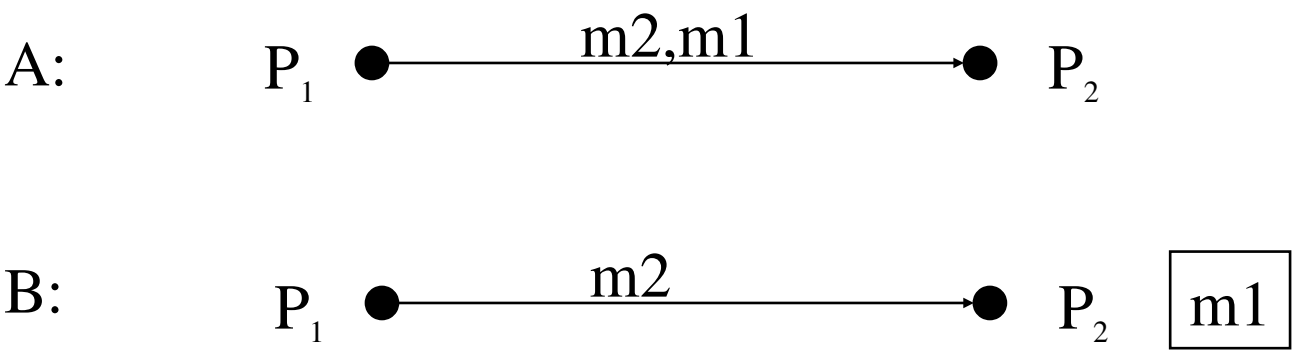
I.5.1 Estado global (memoria compartida)

- En memoria compartida



I.5.1. Estado global: mensajes

- Quien envía un mensaje no sabe si el receptor ha recibido y procesado el mensaje



P_1 no tiene modo de distinguir A de B

I.5.1. Estado global (centralizado)

- Un sistema centralizado es como un nodo de un sistema distribuido
  - No existe estado del sistema de comunicaciones
  - Toda la memoria es local
    - El estado global = el estado local
    - Conocimiento perfecto del estado
      - Posible tomar decisiones algorítmicas en base al conocimiento global del sistema

I.5.2. Ausencia de un marco temporal global (reloj universal)

- Dentro de un nodo se ejecuta un algoritmo secuencial
  - Todo evento sucede bien antes, bien después que cualquier otro evento en dicho nodo.
  - En un sistema centralizado, pues, el orden de los eventos es total, y dicho orden puede asimilarse a los pasos de un reloj global
  - I.e. En un sistema centralizado con varios procesos, todos sus eventos están totalmente ordenados

I.5.2. Marco temporal global

- En un sistema distribuido
  - Existen eventos en nodos diferentes no relacionados por un orden temporal
    - No se puede decir cual “sucede antes”
    - Eventos concurrentes
- Los nodos no pueden coordinarse bajo la suposición de existencia de dicho orden global
  - En un SC dos procesos pueden sincronizar accesos en base al instante de petición

I.5.3. No determinismo

- Un programa centralizado puede describir la computación resultante a partir de unos ciertos valores de entrada de un modo preciso.
- Un sistema distribuido es intrínsecamente no determinista, debido a posibles diferencias en velocidad de ejecución entre sus componentes (tanto nodos como enlaces)

I.5.3. No determinismo (cont)

- Incluso en situaciones cliente/servidor
  - Considerando clientes y servidor como sistema
  - El orden de envío de las peticiones puede ser conocido
  - Sin embargo el orden de llegada no es conocido, debido a diferencias en velocidad de transmisión de los mensajes.
    - Si el servidor no sabe cuantas se van a producir, éste no puede esperar a que se produzcan todas para ordenarlas determinísticamente.

I.5.4. Consideraciones generales

- Ausencia de conocimiento del estado global, inexistencia de reloj global y no determinismo se combinan para dificultar el diseño de algoritmos distribuidos
- Tiempo y estado están relacionados.
  - En sistemas centralizados, una ejecución puede verse como la secuencia de estados que el sistema atraviesa
    - El tiempo viene marcado por dicha secuencia de estados
      - La no ambigüedad de la secuencia deja bien definido el tiempo global

I.5.4. Consideraciones generales (cont)

- En un sistema distribuido puede definirse su estado global
- Una ejecución puede verse también como una secuencia de estados globales
  - La secuencia no es única
  - Incluso los estados que aparecen en secuencias alternativas no tienen que ser los mismos
    - La afirmación de que un sistema distribuido ha pasado por tal o cual estado no tiene un significado sólido

Cuestión

- Sean n procesos, $P_1 \ldots P_n$
- Supongamos cada uno tiene una variable v_i
- Supongamos cada uno ejecuta el siguiente código (inicialmente v_i=0)

P_i

```
for j  in [0..50] do done;
 v_i = 1;
```

- ¿Por cuantos estados diferentes puede pasar el sistema?
- ¿Determinar dos secuencias posibles compartiendo tan solo el estado inicial y final?

I. Conceptos Fundamentales

I.1 Concepto de Sistema Distribuido

I.2 Motivación

I.3 Problemas algorítmicos

I.4 Clasificación de Sistemas Distribuidos

I.5 Algoritmos distribuidos

Diferencias con algoritmos centralizados.

I.6 Modelos de Sistemas Distribuidos

I.6 Modelos de Sistemas Distribuidos

I.6.1 Necesidad y motivación
I.6.2 Sistemas Transicionales
I.6.3 Modelo de sistema distribuido asíncrono
I.6.4 Sistemas síncronos
I.6.5 Propiedades
I.6.6 Conmutación de eventos
I.6.7 Computaciones
I.6.8 Relojes lógicos

I.6.1 Necesidad y motivación

- La motivación para producir un algoritmo distribuido es la correcta y eficiente solución de algún problema
  - Para demostrar la corrección
  - Para mostrar la eficiencia
  - Para plantear el problema
  - Para demostrar la inexistencia de soluciones
  - Para demostrar la necesidad de utilizar un mínimo de recursos

I.6.2. Sistemas Transicionales

- Una computación distribuida es vista típicamente como una colección de eventos discretos
- Cada evento se ejecuta de forma atómica
- Provoca cambios instantáneos en el estado del sistema

I.6.2. Sistemas Transicionales (cont.)

- Abstracción que modela sistemas cuyos estados cambián en una serie de pasos discretos
- Definición:

$$S= (C,\rightarrow,I)$$

- C = conjunto de configuraciones (con $I \subseteq C$, las iniciales)
- $\rightarrow$ relación binaria de transición sobre C

$$(\gamma,\beta) \in \rightarrow \text{ ó } \quad \gamma \rightarrow \beta$$

I.6.2... Ejecución y alcanzabilidad

- Ejecución
  - Secuencia máxima (infinita, o en configuración terminal)

$$E = (\gamma_0, \gamma_1, \gamma_2, \ldots)$$

$$\gamma_0 \in I$$

$$\gamma_i \rightarrow \gamma_{i+1}$$

- Configuración δ alcanzable desde γ, $\gamma \dashrightarrow \delta$

$$\gamma = \gamma_0, \gamma_1, \gamma_2, \ldots, \gamma_k = \delta, \qquad \gamma_i \rightarrow \gamma_{i+1}$$

  - Configuración δ alcanzable, cuando $\gamma \in I$

I.6.3. Modelo de sistema distribuido asíncrono

- El sistema entero se modela con un sistema transicional
- Cada nodo se modela con un sistema transicional también
  - Es necesario relacionar el modelo global con los modelos de los nodos
  - Debe modelarse un sistema de comunicación
    - Como sistema de comunicación va a utilizarse el paso de mensajes

I.6.3... Paso de mensajes asíncronos

- Algoritmo local de un proceso.
  - Sea M el conjunto de mensajes posibles (codificando destino)
  - $(Z, I, \vdash_i, \vdash_s, \vdash_r)$
    - Z es el conjunto de estados
    - I es el subconjunto de estados iniciales
    - $\vdash_i, \vdash_s, \vdash_r$ relaciones de transición
      - $\forall$ $\vdash_i$ transiciones internas (en $Z \times Z$)
      - $\forall$ $\vdash_s$ transiciones de envío (en $Z \times M \times Z$)
      - $\forall$ $\vdash_r$ transiciones de recepción (en $Z \times M \times Z$)

I.6.3... Paso de mensajes asincrono (cont)

- Relación binaria de transición sobre Z □ Z, $\vdash$
  - c $\vdash$ d sii, (c,d) $\in \vdash^{i}$ ó $\exists$ m $\in$ M :: (c,m,d) $\in (\vdash^{s} \cup \vdash^{r})$
- Un algoritmo distribuido para
  - Un conjunto de procesos P = (p_1,…, p_n)
  - Es una colección de algoritmos locales
  - Cada algoritmo local para cada proceso
  - La colección de mensajes es compartida
    - Una configuración global corresponderá a las configuraciones locales más mensajes en tránsito
    - Inicialmente no hay mensajes en tránsito

I.6.3... Paso de mensajes (cont)

- S= (C,$\rightarrow$,I) a partir de (Z_j,I_j,$\vdash_i^i$, $\vdash_j^s$, $\vdash_j^r$)
  - C = {(c_1,...,c_n, M) :: $c_j \in Z_j$ y M $\subseteq$ M }
  - $\rightarrow$ = $\cup_{j=1,n} \rightarrow_j$, donde $\gamma \rightarrow_k \delta$ sii
    - γ =(c_1,..., c_k,...,c_n, M), δ =(c_1,..., c'_k,...,c_n, M')
      - $c_k \vdash_k^i c'_k$ y M = M' (transición interna)
      - $\forall$ $\exists$ m $\in$ M : (c_k,m,c'_k) $\in \vdash_k^s$ y M' = M $\cup$ {m}
      - $\forall$ $\exists$ m $\in$ M : (c_k,m,c'_k) $\in \vdash_k^r$ y M = M' $\cup$ {m}
  - I = {(c_1,...,c_n, M) :: $c_j \in I_j$ y M = ø}

I.6.3.. Paso de mensajes: ejecuciones y eventos

- Una ejecución de un sistema distribuido no es más que una ejecución del sistema transicional inducido anterior
  - Notar que cada transición afecta tan sólo a un proceso (y, posiblemente, al sistema de mensajes)
  - Existe localidad total de acción
    - Los pares $(c,d) \in \vdash_i^i$ son los eventos internos de p_j
    - Los triples $(c,m,d) \in \vdash_i^s$ son los eventos de envío de p_j
    - Los triples $(c,m,d) \in \vdash_i^r$ son los eventos de recepción de p_j

I.6.3... Aplicabilidad de eventos

- Sea *e* de p_k -bien (c,d) ó (c,m,d) -, y sea $\gamma = (c_1,\ldots, c_k,\ldots,c_n, M)$, *e* es aplicable a γ, dando $e(\gamma) = (c_1,\ldots, c'_k,\ldots,c_n, M')$, cuando,
  - $c = c_k$ y $c'_k = d$
  - *e* = (c,d) es interno, o
  - *e* = (c,m,d) es de recepción, y $m \in M$.
    - Entonces $M' = M \backslash \{m\}$
  - *e* = (c,m,d) es de envío.
    - Entonces $M' = M \cup \{m\}$

I.6.3... Aplicabilidad de eventos

- Un evento *e* puede representarse por
  - *e* = (c,x,y,d)
  - x, y $\subseteq$ M, tal que $|x| < 2$ y $|y| < 2$
    - x es el posible mensaje recibido, y el posible mensaje enviado
    - Evento interno x = y = ø
    - Evento de recepción x ={m}, y = ø
    - Evento de envío x = ø, y = {m}
  - Para $\gamma = (c_1,\ldots, c_k=c,\ldots,c_n, M)$, x $\subseteq$ M
    - $e(\gamma) = (c_1,\ldots, c'_k=d,\ldots,c_n, (M\backslash x) \cup y)$

I.6.3... Modelo de envío síncrono

- Un sistema de envío síncrono impone una cota superior al tiempo de tránsito de un mensaje
  - Notar que la recepción del mensaje implica su procesamiento
- Emisión y recepción son un mismo evento
  - Un mensaje no es enviado hasta que el receptor se encuentra listo para recibirlo
    - Quien envía queda bloqueado hasta que tal circunstancia ocurre

I.6.3... Envío síncrono (cont)

- S= (C,→,I) a partir de (Z_j,I_j,$|-_i^i$, $|-_j^s$, $|-_j^r$)
  - C = {(c_1,…,c_n) :: $c_j \in Z_j$}
  - I = {(c_1,…,c_n) :: $c_j \in I_j$}
  - Transiciones internas de algún proceso
    - $\rightarrow_j$
  - Emisiones/recepciones de un mensaje entre un par de procesos
    - $\rightarrow_{j,k}$

I.6.3... Envío síncrono (cont)

- $\rightarrow_k$, donde $\gamma \rightarrow_k \delta$ sii
  - $\gamma = (c_1,\ldots,c_k,\ldots,c_n)$, $\delta = (c_1,\ldots, c'_k,\ldots,c_n)$
    - $c_k \mid\!-_k^i c'_k$ (transición interna)
- $\rightarrow_{j,k}$, donde $\gamma \rightarrow_{j,k} \delta$ sii
  - $\gamma = (c_1,\ldots, c_j,\ldots,c_k,\ldots,c_n)$, $\delta = (c_1,\ldots, c'_j,\ldots,c'_k,\ldots,c_n)$
    - $\exists\, m \in M$:
      - $(c_j,m,c'_j) \in \mid\!-_j^s$
      - $(c_k,m,c'_k) \in \mid\!-_k^r$

I.6.4. Sistemas síncronos

- El conjunto de ejecuciones síncronas es un subconjunto del conjunto de ejecuciones asíncronas.
- El modelo asíncrono es, pues, más general
  - Si algo es posible en sistemas asíncronos, también lo será en sistemas síncronos
- El modelo síncrono introduce una sicronización implícita más estricta
  - Posibilidad de provocar interbloqueos si no se tiene en cuenta

I.6.4... Planificador de eventos

- ¿Quién determina que ocurra una ejecución u otra?
  - Cuando varios eventos están habilitados en γ , ¿quién determina cuál de ellos sucede?
    - Nadie: no determinismo
    - Es útil pensar que existe un planificador en el sistema, (en algunos casos, un “adversario”) que decide el evento que sucede.
- Todo sistema, aparte de los algoritmos locales, dispone de un planificador

I.6.4... Equidad

- Hay problemas irresolubles sin exigir comportamientos específicos del planificador.
  - I.e. Exclusión mutua
    - Requisito: mientras hayan peticiones de sección crítica, deben haber concesiones de sección crítica.
- Típicamente, los requisitos incumplibles tienen que ver con propiedades de viveza (finalmente, esto o aquello sucede…)
  - Ello sucede cuando a un cierto componente del sistema no se le deja intervenir

I.6.4... Equidad

- Ejecución equitativa
  - Intuición: Aquella que en la que no sucede que un evento permanentemente habilitado jamás suceda
  - Ejemplo: cuando un proceso se encuentra en una sección crítica
    - Los pasos de su algoritmo están habilitados
    - Si jamás suceden, el proceso jamás ejecutará el protocolo de salida de la sección crítica
    - Ningún otro proceso podrá entrar en la sección crítica

I.6.4... Equidad

- Ejecución equitativa
  - Equidad débil
    - Ningún evento es aplicable en un número infinito de configuraciones sucesivas sin suceder en la ejecución.
  - Equidad fuerte
    - Ningún evento es aplicable en un número infinito de ejecuciones sin suceder en la ejecución.
  - La diferencia estriba en la permanencia del estímulo requerida en equidad débil
- Planificadores equitativos

I.6.5. Propiedades en sistemas transicionales

- Para demostrar propiedades de un sistema (algoritmo) distribuido
  - Se modela
  - Se hacen corresponder las propiedades en el modelo
  - Se demuestran las propiedades sobre el conjunto de ejecuciones posibles sobre el modelo
    - Se realizan suposiciones acerca del planificador, si ello es necesario

I.6.5. Propiedades (cont)

- Sobre ejecuciones
- De seguridad
  - Una propiedad que debe satisfacerse en cada configuración alcanzable dentro de una ejecución
- De viveza
  - Una propiedad debe satisfacerse en alguna configuración alcanzable en una ejecución

I.6.5... Propiedades: demostración

- Utilización de aseveraciones
  - Aseveración: predicado verdadero en un subconjunto de las configuraciones
- Propiedades de seguridad
  - Aseveración P es cierta en toda configuración alcanzable de toda ejecución del algoritmo
    - P es siempre cierto
  - Técnica 1: Demostración por inducción
  - Técnica 2: Demostrar que P es un invariante

I.6.5.... Invariantes

- P(γ) es verdadera sii P se cumple en γ.
- {P} $\rightarrow$ {Q} sii para toda $\gamma \rightarrow \delta$, si P($\gamma$), entonces, Q($\delta$)
- P es un invariante sii
  - Para todo $\gamma \in$ I, P(γ), y
  - {P} $\rightarrow$ {P}
    - Cierto en toda configuración inicial
    - Conservado en cada transición

I.6.5... Invariantes (cont)

- P invariante, ¿implica P(γ) para todo γ ?
- Existe P, tal que P(γ) en toda configuración alcanzable, γ, pero P no es un invariante (?)
  - La técnica del invariante no siempre sirve para demostrar una propiedad de seguridad
- Si Q invariante, y Q $\Rightarrow$ P, entonces P(γ) en toda γ alcanzable
  - Técnica 3: fortalecimiento de P para encontrar un invariante
    - Pero ¿cómo?

I.6.5... Invariantes

- Sean P, Q aseveraciones, P es un Q-derivado sii
  - $Q(\gamma) \Rightarrow P(\gamma)$ para todo $\gamma \in I$
  - $\{Q \wedge P\} \rightarrow \{Q \Rightarrow P\}$
- Q invariante y P un Q-derivado, implican $Q \wedge P$ es un invariante
  - Tecnica 3 bis: fortalecer P con otra propiedad de la que sea un derivado

I.6.5... Invariantes

- Debemos demostrar
  - Para todo $\gamma \in I$, $Q(\gamma)$ ^ $P(\gamma)$
    - Q invariante, luego $Q(\gamma)$, $\gamma \in I$
    - P es Q-derivado, luego $Q(\gamma) \Rightarrow P(\gamma)$, $\gamma \in I$
      - Luego $P(\gamma)$, $\gamma \in I$
      - Luego $Q(\gamma)$ ^ $P(\gamma)$, $\gamma \in I$
  - {Q^P} $\rightarrow$ {Q^P}
    - Supongamos $\gamma \rightarrow \delta$
    - Supongamos $Q(\gamma)$ ^ $P(\gamma)$

I.6.5... Invariantes

- Entonces Q(γ)
- Q invariante
  - Luego Q(δ)
- P es Q-derivado
  - Luego {Q^P} $\rightarrow$ {Q $\Rightarrow$ P}
  - Q(γ) ^ P(γ)
    - Luego Q(δ) $\Rightarrow$ P(δ)
    - Pero Q(δ)
      - Luego P(δ)
      - Luego Q(δ) ^ P(δ)

I.6.5... Propiedades de viveza

- Aseveración P es finalmente verdadera
- Técnicas para ejecuciones infinitas:
  - Funciones de norma
  - Finalización apropiada
- Algoritmos con tan sólo ejecuciones finitas pueden utilizar técnicas más simples
  - Conteo de casos posibles
- Predicado ***term*** verdadero en configuraciones terminales

I.6.5... Viveza con respecto a P

- El sistema S termina apropiadamente sii (***term*** ⇒ P) es siempre cierto en S
- Conjuntos bien fundamentados
  - Orden parcial,<
  - No contiene secuencias decrecientes infinitas
  - Ejemplo típico: números naturales
    - Pero no los enteros

I.6.5... Viveza: normas

- Función de Norma f para P
  - W conjunto bien fundamentado
  - f función de C a W
  - Para toda transición $\gamma \rightarrow \delta$, $f(\gamma) > f(\delta)$ o $P(\delta)$
- Si S termina apropiadamente y existe una función de norma para P
  - P es verdadero en alguna configuración de cada ejecución de S
    - Reducción al absurdo y casos
    - Inducción (?)

I.6.5... Uso de normas: demo

- E = ($\gamma_0,\gamma_1,\gamma_2,\ldots$)
  - Si E finita E = ($\gamma_0,\gamma_1,\gamma_2,\ldots,\gamma_m$), con ***term***($\gamma_m$),
    - Luego (terminacion apropiada) P(γ_m), qed.
  - Si infinita, sea E' el prefijo más largo donde ¬P(γ_j), para toda $\gamma_j \in$ E'
    - Si E' finito, E' = ($\gamma_0,\gamma_1,\gamma_2,\ldots,\gamma_l$), y P($\gamma_{l+1}$), qed.
    - Si E' infinito, E' = E,
    - Sea (f(γ_0), f(γ_1), f(γ_2), . . .), dado que ¬P(γ_i) y $\gamma_i \rightarrow \gamma_{i+1}$, para todo i
      - f(γ_i)> f(γ_{i+1}), para todo i
        - (f(γ_0), f(γ_1), f(γ_2), . . .), secuencia infinita descendiente en W

I.6.5... Viveza: consideraciones de equidad

- Una función de norma es extremadamente restrictiva
  - Exige el decremento en cada paso
- Por lo general, resulta difícil encontrar tal función
- Es más fácil encontrar una función que se decremente tan sólo con cierto tipo de transiciones

I.6.5... Funciones de norma y equidad

- Función de norma equitativa para P, f,
  - Conjunto de eventos, E, E = $N_f \oplus D_f$, donde
    - D_f es el conjunto de eventos e, $e(\gamma) = \delta$, tales que $f(\gamma) > f(\delta)$ ó $P(\delta)$
      - Conjunto de eventos con garantías
    - Eventos decrementales
    - Toda ejecución equitativa infinita incluye un número infinito de eventos de D_f
- P se cumple en toda ejecución equitativa…

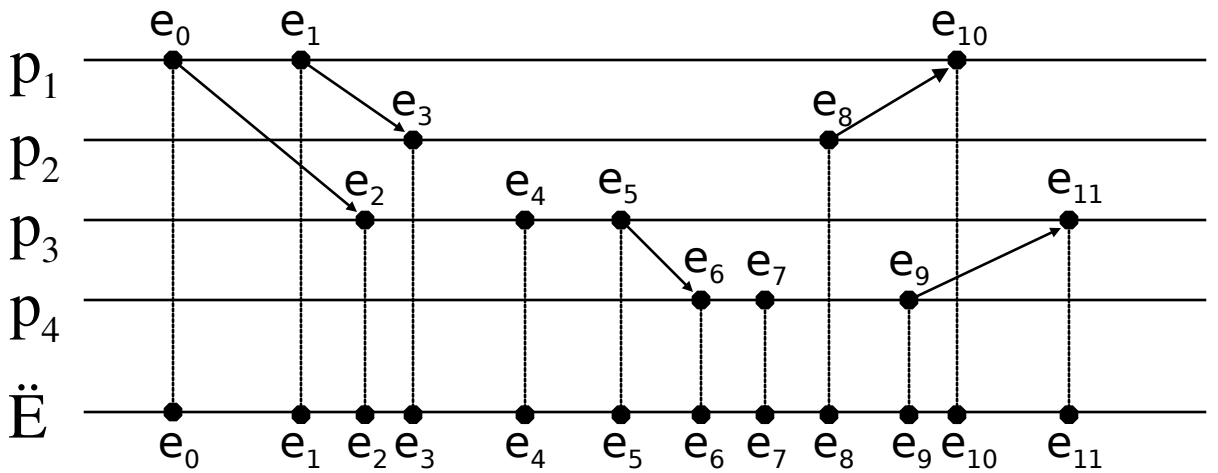
I.6.5... Ejecuciones y ordenación de eventos

- Toda ejecución E = ($\gamma_0,\gamma_1,\gamma_2,\ldots$) induce de forma única una secuencia asociada de eventos
  - Ë = ($e_0, e_1, e_2,\ldots$), donde $e_i(\gamma_i) = \gamma_{i+1}$,
    - Cuando todos los procesos ejecutan algún evento, E también se obtiene de forma única de Ë
- El orden de ejecución de los eventos induce una noción natural de tiempo
  - e_i ocurre antes que e_k para k>j
  - Una ejecución puede ser visualizada en un diagrama espacio-tiempo

I.6.6. Diagramas espacio-tiempo

- Cada proceso tiene su línea temporal
- Si un mensaje, m, es enviado en el evento a, y recibido en el b, una flecha es dibujada yendo de a a b .
  - Los eventos a y b , son llamados eventos correspondientes
- Una línea de ejecución es dibujada, recogiendo las proyecciones verticales de los eventos.

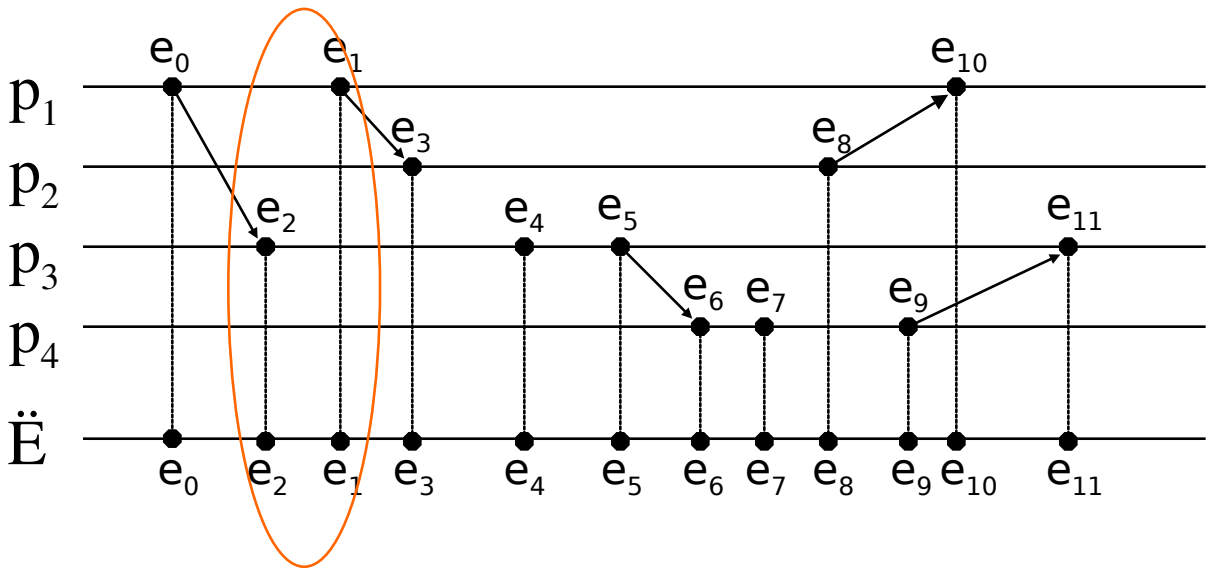
I.6.6. Diagramas de espacio-tiempo



I.6.7. Conmutabilidad de eventos

- El orden dado por la ejecución anterior es algo arbitrario
  - Depende del no determinismo presente en el orden de ejecución de cada proceso
  - En principio podíamos pensar que si cada proceso va a su propia velocidad, evento e_1 podría suceder después que evento e_2
    - Tan solo hay que “estirar” un poco la línea de p_1

I.6.7.... Diagrama revisado



I.6.7... Conmutabilidad de eventos

- Formalizar la intuición anterior del siguiente modo
- e_p y e_q , con p ≠ q, son conmutables sii
  - $e_p(\gamma)$ y $e_q(\gamma)$
- Además
  - $e_q(e_p(\gamma)) = e_p(e_q(\gamma))$

I.6.7... Conmutación de eventos (cont)

- El resultado anterior aplicable a cualquier par de eventos consecutivos de una ejecución:
  - Sobre procesos diferentes
- Cuando un par de eventos no puede ser conmutado decimos que están relacionados causalmente

I.6.7... Causación directa

- Toda ejecución define una relación sobre sus eventos, la *causación directa*, □
- e □ e' cuando
  - e y e' son dos eventos consecutivos de un mismo proceso, p, y e aparece antes que e'.
  - e es un evento de envío y e' es su correspondiente evento de recepción

I.6.7... Orden causal

- Toda ejecución E define una relación sobre sus eventos llamado *orden causal,* <
- El orden causal se deriva de la causación directa
  - < es el cierre transitivo de
    - I.e. La menor relación conteniendo a y siendo transitiva
- Expresamos e≤e' , cuando bien e<e', ó e=e' .
  - Si ¬(e≤e' , ni e'≤e), e y e' son concurrentes
    - e || e'

I.6.7.... Consecuencias

- Dado evento e, se define su conjunto de causación como
  - $C(e) = \{e' \mid e' \leq e\}$
    - Claramente $e' \leq e$ sii $C(e') \subseteq C(e)$
  - $C_p(e) = \{e' \mid e' \leq e \text{ y } e' \text{ de } p\}$
- e<e' implica la existencia de una cadena causal
  - $e = e_0, e_1, e_2, \ldots, e_k = e'$
  - Donde para todo i, e_i □ e_{i+1}

I.6.7... Consecuencias

- Dado e, existen múltiples cadenas descendientes a partir de e, l = $(e_0, e_1, e_2, \ldots, e_k = e)$
  - Una cadena descendiente es máxima sii $C(e_0) = \{e_0\}$
    - Cadena máxima causal de e
  - Sea L = {l : l es una cadena máxima causal de e}
    - Existe una cota superior para |l| (?)
  - ***orden***(e) = max{|l| : l $\in$ L}

I.6.7... Visión local de la ejecución

- Cada proceso tiene tan sólo una visión parcial de la ejecución global
  - Los eventos que le afectan
  - Internos, recepción o envío
- La conmutación de eventos no relacionados causalmente no modifica la visión local de ningún proceso
  - La ejecución resultante será “equivalente” desde el punto de vista de cada proceso

I.6.7... Secuencias de eventos

- Sea f = (f_0, f_1, f_2, ...), como sigue
  - Existe $\delta_0 \in I$, tal que $\delta_1 = f_0(\delta_0)$
  - Para i > 0, $\delta_i = f_{i-1}(\delta_{i-1})$, está bien definido
- f se corresponde con la ejecución A
  - A = (δ_0, δ_1,...), y f = Ä

I.6.7... Permutaciones de ejecuciones

- E = ($\gamma_0, \gamma_1, \gamma_2, \ldots$), Ë = ($e_0, e_1, e_2, \ldots$)
- Sea f = ($f_0, f_1, f_2, \ldots$) una permutación de Ë
  - σ biyección sobre N (infinita), [0,…,k-1] (finita)
    - $f_i = e_{\sigma(i)}$
- f es consistente con el orden causal de Ë, sii
  - $f_i \leq f_j$ implica que $i < j$

I.6.7... Equivalencia de permutaciones

- E = ($\gamma_0,\gamma_1,\gamma_2$,. . .), Ë = ($e_0$, e_1, e_2,. . .), y
  f = (f_0, f_1, f_2, ...) consistente. Propiedades:
  - f es una ejecución Ä, partiendo de γ_0
  - Si E es finita, A tiene la misma longitud, k
    - Mismas configuraciones finales: $\gamma_k = \delta_k$
  - Si E es infinita, A también lo es

I.6.7... Computaciones

- E $\equiv$ A, con E = (γ_0,γ_1,γ_2,. . .), A = (δ_0,δ_1,δ_2,. . .), sii
  - $\gamma_0 = \delta_0$
  - Ë es una permutación consistente de Ä
  - $\equiv$ es una relación de equivalencia (?)
- Cada clase es una computación
  - El conjunto cociente es el de las computaciones
  - La vista local de un proceso en cualquier ejecución de una misma computación no varía
    - Los procesos no pueden distinguir entre ejecuciones equivalentes
    - Las computaciones son las entidades que tienen sentido “físico”

I.6.7... Visión local (revis.)

- Sea C una computación, y sea Ë una ejecución de C (Ë ⊆ C)
- La ejecución local sobre p de Ë, $Ë_p$,
  - Subsecuencia de eventos de Ë, conteniendo tan sólo los eventos del proceso p
  - En el mismo orden relativo que en Ë
- También: Proyección sobre p de Ë, $Ë|_p$
- ***orden*** $_p$(e), orden de e en $Ë_p$

I.6.8 Relojes lógicos

- Motivación
  - Emular el poder expresivo de los relojes físicos
  - Algunos problemas pueden ser resueltos cuando se dispone de una ordenación (parcial o total) de los eventos
    - La ordenación debe ser consistente con la relación de causalidad
    - Los relojes lógicos siempre se refieren a computaciones particulares
      - Hablaremos del conjunto de eventos de una computación

I.6.8... Reloj

- Sobre una computación C
  - Sobre su conjunto de eventos
- Función, Θ, del conjunto de eventos a un conjunto ordenado
  - Θ: events(C) $\longrightarrow$ W
  - a <b implica $\Theta(a) < \Theta(b)$
- Interesa que pueda calcularse/observarse dentro del sistema

I.6.8... Reloj: ejemplos

- Ordenación en una ejecución
  - Si Ë = (e_0, e_1, e_2,. . .),
    - $\Theta(e_i) = i$
  - En el hipotético caso de que existiese un observador global, esto podría ser usado por él
  - Este orden no puede ser observado desde dentro del sistema
    - Ningún proceso puede observar este reloj
    - El reloj no está bien definido para las computaciones

I.6.8... Reloj: ejemplos

- Reloj de tiempo real
  - Todo proceso recibe un reloj hardware
  - Cuando sucede un evento, el proceso lo asocia con el valor del reloj en dicho momento
- Obtenemos un modelo diferente
  - Sincronización de cambios de estado a nivel global
  - No conmutabilidad de eventos!

I.6.8... Reloj lógico de Lamport

- Clásico ejemplo
  - A un evento a, le asigna la longitud de la máxima cadena causal decreciente a partir de a
    - I.e. $\Theta_L(a)$ = ***orden***(a)
  - Notar: este reloj está definido sobre la computación
    - Todas las ejecuciones de una computación definen el mismo orden causal
  - Si a <b entonces ***orden***(a)< ***orden***(b)
    - $\Theta_L(a) < \Theta_L(b)$

I.6.8... Cálculo de Θ_L

- Algoritmo distribuido
  - Proceso p tiene un contador t_p, inicialmente 0
  - Sea a el evento. Primero se recalcula t_p así
    - Si interno o de envío t_p++.
    - En todo mensaje se envía t_p, que es el valor de reloj del evento de envío
    - Si a recepción, sea τ_m el valor de reloj enviado en el mensaje m recibido, $t_p := \max(t_p, \tau_m) + 1$
  - Finalmente, $\Theta_L(a) = t_p$

I.6.8... Cálculo de Θ_L: demo

- Por inducción en el orden de los eventos de una computación
  - Base: eventos de orden 0
    - Internos o de envío
    - Primeros en su proceso
  - HI: $\Theta_L(a)$ = ***orden***(a), para todo a, tal que ***orden***(a)<n (n>0)

I.6.8... Modularidad

- Notar que el algoritmo de cálculo del reloj lógico de Lamport es independiente del algoritmo principal que se esté ejecutando
  - Estructuración de sistemas por capas
  - El algoritmo de Lamport estaría en una capa inferior
  - El algoritmo principal se asentaría sobre dicha capa

I.6.8... Relojes causales

- El reloj de Lamport pierde información de la relación causal (?)
  - Hay casos en que interesa retener dicha información
    - Para conocer cuando dos eventos diferentes son concurrentes
  - Para ello necesitamos una condición más fuerte
    - a <b **si y sólo si** $\Theta(a) < \Theta(b)$
  - Necesitamos un dominio imagen parcialmente ordenado. Usaremos vectores.

I.6.8... Reloj vector de Mattern, Θ_v

- $W = N^n$
  - $\Theta_v(a)$ es un vector de longitud n (número de procesos)
  - Orden de vectores
    - $(a_0, a_1, ..., a_n) \leq (b_0, b_1, ..., b_n)$ sii $\forall i: a_i \leq b_i$
      - Orden parcial
  - $\Theta_v(a) = (t_1, t_2, \ldots, t_n)$, donde $t_p = |C_p(a)|$

I.6.8... Ejercicio

- Proponer un algoritmo que lo implemente
- Proponer intuición de demostración

I.6.9. Complejidad

- Comunicación
  - Número de mensajes enviados/recibidos
    - Pero no todos los mensajes tienen el mismo tamaño
    - Constituye una primera aproximación
    - Válida cuando están acotados
  - Cantidad de información enviada
    - Número de bits
    - Refleja mejor la utilización de los canales de comunicación

I.6.9. Complejidad (cont)

- Temporal (latencia)
  - Dificultad. Modelo asíncrono
    - No hay noción de tiempo que se tarda en hacer algo.
  - Idealización:
    - Ejecutar eventos no cuesta nada: carga temporal 0
    - Es el envío/recepción de mensajes lo que cuesta
      - Notar que un mensaje se considera recibido cuando ha sido procesado por el receptor
      - Coste uniforme: 1
  - Coste temporal de una computación = número de recepciones de la cadena máxima causal con mayor número de recepciones.

I.6.9. Complejidad (cont)

- Espacial
  - Cantidad de memoria necesaria en cada proceso para ejecutar un algoritmo
  - Tamaño máximo de los mensajes necesarios
    - Tamaño máximo de la memoria compartida necesaria