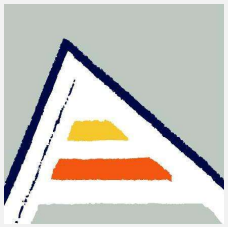


Grado en Ingeniería Informática

Sistemas distribuidos

Sincronización de sistemas distribuidos

Víctor Vives
vvives@dtic.ua.es

Departamento de Tecnología Informática y Computación

2021 - 2022

1. Introducción

Sincronización en sistemas distribuidos

- Para poder ejecutar tareas en entornos distribuidos es importante que el **orden** en las secuencias de las operaciones de procesos sea **estricto** y **universal**.
- Mantener la base del tiempo impone varios problemas tecnológicos importantes.
    - El problema más relevante es la **propagación**.
    - Es el principal problema en sistemas que procesan grandes cantidades de datos a alta velocidad.
    - Los intercambios de alta velocidad pueden afectar a la sincronización.
- La **sincronización** se define como **la forma de forzar un orden parcial o total en cualquier conjunto de eventos** y se utiliza para definir tres problemas distintos pero relacionados entre sí.
    1. La sincronización entre un emisor y un receptor
    2. La especificación y el control de la actividad común entre procesos cooperativos
    3. La serialización de accesos concurrentes a objetos compartidos por múltiples procesos.

1. Introducción

Sincronización de relojes en un sistema distribuido

- La **sincronización de reloje**s en un sistema distribuido consiste en **garantizar que los procesos se ejecuten de forma cronológica y a la vez respetando el orden de los eventos del sistema**.
- Problema derivado de la **necesidad de comunicar distintas máquinas que trabajan en una misma tarea de forma conjunta**.
- **Cada máquina posee un reloj independiente** cuya arquitectura, por muy precisa que sea, hace **imposible** que tenga **tiempos idénticos**.
- Existen varios algoritmos que resuelven este problema a diferentes niveles.
- En todos los casos es necesaria una **comunicación** que establezca la **misma referencia de tiempo para todas las entidades**.

1. Introducción

Sincronización de relojes en un sistema distribuido

- No existe un reloj global al sistema.
- Cada máquina tiene su **propio reloj interno**:
  - Utilizado por procesos locales para la obtención del tiempo actual.
  - Los procesos de una máquina pueden tener marcas de tiempo distintas.
  - Los relojes derivan con respecto al tiempo perfecto y las tasas de deriva también difieren entre ellos.

- **Sesgo**: diferencia de tiempo entre dos relojes en un instante determinado
- **Tasa de deriva**: # segundos / segundo

1. Introducción

Sincronización de relojes en un sistema distribuido

- Se puede calcular el tiempo en el reloj software: $C_i(t) = \alpha H_i(t) + \beta$
- En un instante **t** el SO lee el valor del reloj hardware de la máquina: $H(t)$.
- Se dice que un reloj hardware es correcto si su límite de deriva es conocido ($\rho > 0$). Ej. 10^{-6} segs/seg.
- Por tanto, el error en la medida de dos eventos en t y t' está limitado:

$$(1 - \rho)(t' - t) \leq H(t') - H(t) \leq (1 + \rho)(t' - t) \qquad \text{donde } t' > t$$

$$\text{Monotonicidad: } t' > t \rightarrow C(t') > C(t)$$

- Un **reloj defectuoso** es aquel que no cumple ninguna de las condiciones de corrección.
- Un **fallo de ruptura** de reloj es cuando el reloj se para y no emite tics.
- Un **fallo arbitrario** se considera cualquier otro fallo.

1. Introducción

Sincronización de relojes en un sistema distribuido

Necesidades de sincronización

- Aplicaciones en tiempo real
- Ordenación natural de eventos distribuidos

Concepto de sincronización

- Mantener relojes sincronizados **entre sí**
- Mantener relojes sincronizados **con la realidad**

1. Introducción

Sincronización de relojes en un sistema distribuido

UTC o Tiempo Universal Coordinado

- Transmisión de señal desde centros terrestres o satélites.
- Una o más máquinas del sistema distribuido son receptoras de señal UTC.

```
DateTime localDate = DateTime.Now;
DateTime utcDate = DateTime.UtcNow;

string cultureName = "es-ES";
CultureInfo culture = new CultureInfo(cultureName);

Console.WriteLine("{0}: {1}", cultureName, localDate.ToString(culture));
Console.WriteLine("{0}: {1}", cultureName, utcDate.ToString(culture));
```

```
# Output
es-ES: 23/11/2021 15:00:00
es-ES: 23/11/2021 14:00:00
```

1. Introducción

Sincronización de relojes en un sistema distribuido

UTC o Tiempo Universal Coordinado

- Los relojes se pueden sincronizar con fuentes externas muy precisas.
- Reloj con deriva de 10^{-13} (un segundo cada 300.000 años).
- UTC es un estándar internacional de establecimiento y mantenimiento del tiempo transcurrido.
- Está basado en el tiempo atómico y ocasionalmente ajustado al tiempo astronómico.
- La señal se difunde mediante estaciones de radio por tierra y mediante satélites.
    - Una estación terrestre tiene una precisión entre 0.1 y 10 milisegundos.
    - GPS tiene una precisión de 100 nanosegundos.

1. Introducción

Sincronización de relojes en un sistema distribuido

Existen **dos tipos de relojes**:

- **Físicos**: relacionado con el tiempo real.
- **Lógicos**: relacionado con el orden de los eventos y no con el tiempo en el que ocurren.

En los **relojes físicos** encontramos **dos tipos de sincronización**:

- **Externa**: los relojes se sincronizan con una fuente de tiempo externa fiable.
- **Interna**: los relojes no se sincronizan con una fuente de tiempo externa.

2. Relojes físicos

Sincronización interna

Algoritmo de Cristian

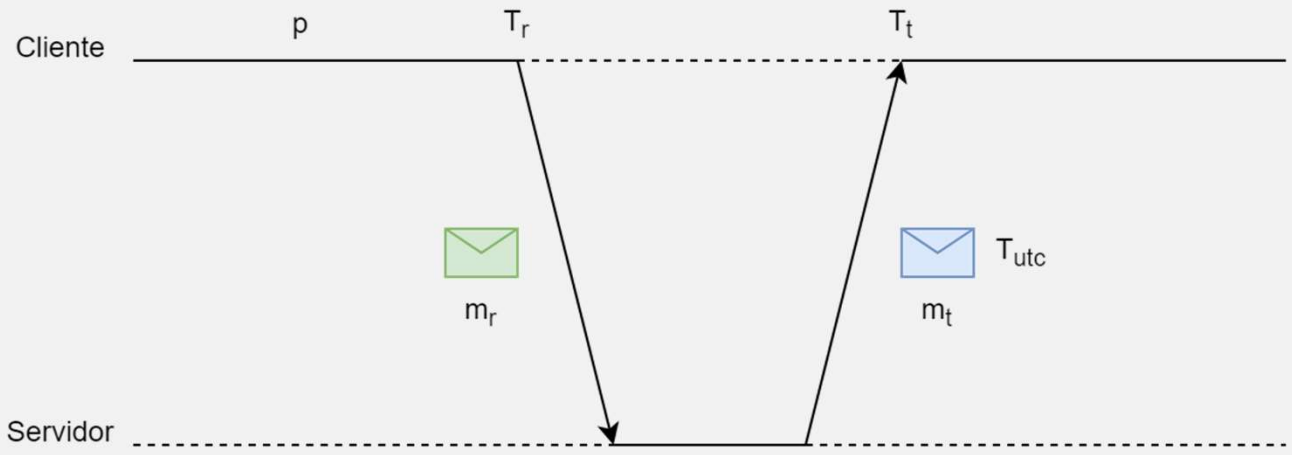
Consiste en un servidor conectado a una fuente de UTC y unos clientes que se sincronizan con él.

1. Un proceso **p** hace una petición de tiempo al servidor en un mensaje m_r
2. Al momento de realizar la petición guarda una marca de tiempo T_r de ese momento
3. El servidor responde con otro mensaje m_t en el que incluye su tiempo T_{utc}
4. El proceso que recibe ese mensaje m_t guarda la marca de tiempo de llegada T_t
5. El proceso realiza una estimación previa antes de actualizar su reloj debido al tiempo de transmisión de la red: $T_{cliente} = T_{utc} + RTT/2$, siendo $RTT = (T_t - T_r) = T_{utc} + (T_t - T_r)/2$
6. El proceso actualiza su reloj con $T_{cliente}$

2. Relojes físicos

Sincronización interna

Algoritmo de Cristian



2. Relojes físicos

Sincronización interna

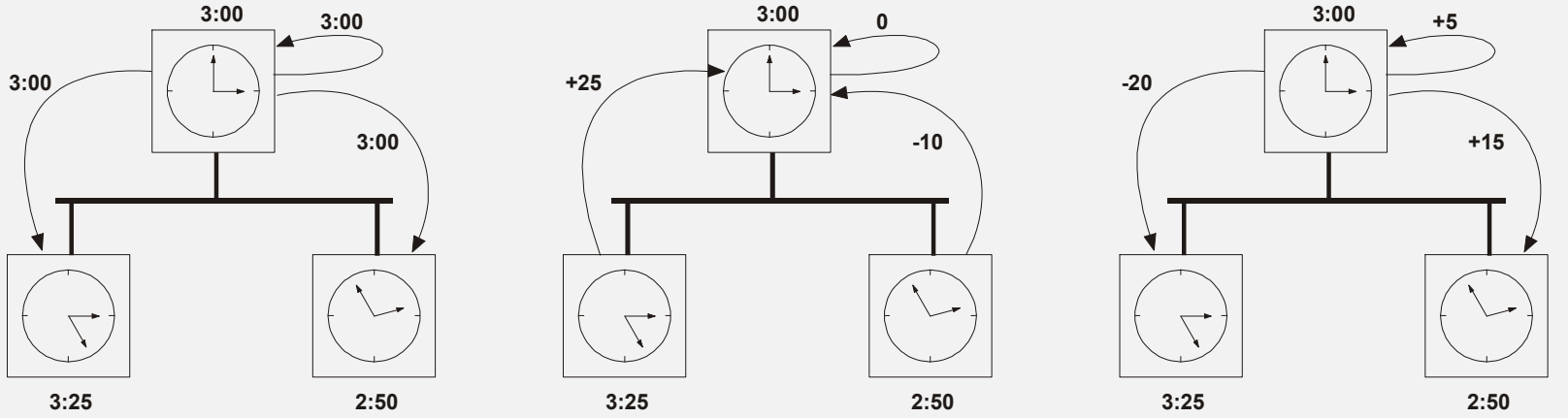
Algoritmo de Berkeley

- Utilizado para **entornos** en los que **no es posible disponer de fuentes de tiempo UTC.**
- Gracias a este algoritmo los relojes del entorno pueden mantenerse sincronizados.
- Es un algoritmo **centralizado**: un maestro y varios esclavos.
- El maestro solicita a los esclavos su hora **periódicamente** para obtener la diferencia promedio.
- El maestro **envía esa diferencia promedio** a todos los esclavos y **todos actualizan sus horas**.
- Existe un **coordinador activo** a diferencia del algoritmo de Cristian que es pasivo.

2. Relojes físicos

Sincronización interna

Algoritmo de Berkeley



2. Relojes físicos

Sincronización externa

Network Time Protocol o NTP

- Protocolo de Internet para sincronizar relojes de diferentes sistemas mediante el **enrutamiento de paquetes** en redes con **latencia variable**.
- Utiliza **UDP** como capa de transporte usando el **puerto 123** por defecto.
- Utilizado sobre Internet a diferencia de Cristian o Berkeley que se usan a menudo para intranets.
- Servicio **confiable** que **puede sobrevivir a largas pérdidas de conectividad**.
- Permite que los clientes se sincronicen con la frecuencia suficiente para **evitar la tasa de deriva**.
- Proporciona **protección contra interferencias** con el servicio horario: malicioso o accidental.

2. Relojes físicos

Sincronización externa

El servicio NTP es proporcionado por una **red de servidores** ubicados en Internet.

- **Servidores primarios**: conectados directamente a la fuente de tiempo UTC.
- **Servidores secundarios**: son sincronizados en última instancia con los primarios.

Estos servidores se conectan en una jerarquía lógica llamada **subred de sincronización**.

Los niveles de esta jerarquía se denominan **estratos**.

- Los relojes UTC están en el estrato 0.
- Los servidores primarios ocupan el estrato 1 y son la raíz.
- Los servidores secundarios ocupan el estrato 2 y son sincronizados con los primarios.
- Los servidores hoja o de nivel más bajo se ejecutan en las máquinas de trabajo de los usuarios.

2. Relojes físicos

Sincronización externa

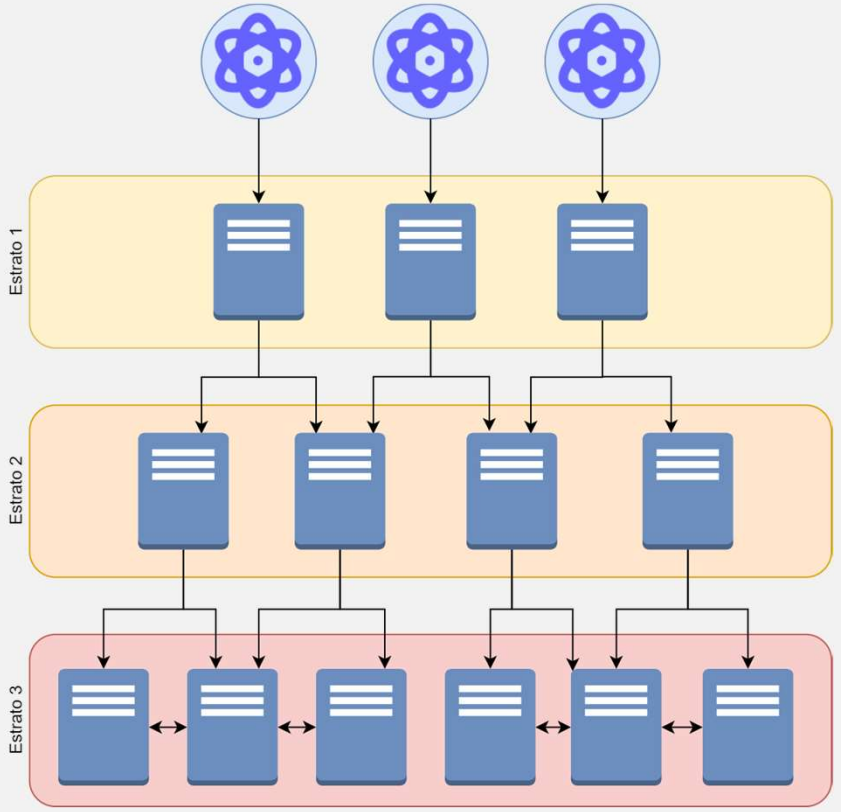
Network Time Protocol o NTP

Modos de sincronización:

- Multidifusión o *multicast*
- Llamada a procedimiento
- Simétrica o intercambio de mensajes

Sincroniza a los clientes con UTC

Todos los nodos utilizan UDP



3. Relojes lógicos

¿Qué es un reloj lógico y para qué sirve?

La idea principal de un reloj lógico consiste en crear **un sistema de convergencia del tiempo** mediante la medición de las **derivas**.

- La noción del tiempo universal se sustituye por la de **tiempo global autoajustable**.
- Los relojes lógicos son útiles para **ordenar eventos** en ausencia de un reloj común.
- Los relojes lógicos solo representan una **relación de orden** parcial.
- Funcionan mediante la alteración del conteo del tiempo para mantener la **sincronización mediante variables** que afectan a la hora de referencia.
- Cada proceso mantiene una **variable entera** que **incrementa en uno al generar un evento**.

3. Relojes lógicos

Lamport

Para la sincronización de relojes lógicos Lamport definió una relación llamada **ocurrencia anterior**.

La expresión **a** → **b** o "**a ocurre antes que b**". La ocurrencia puede darse en dos situaciones:

1. Si **a** y **b** son eventos del mismo proceso, y **a** ocurre antes que **b**, entonces **a** → **b** es verdadero.
2. Si **a** es el evento en el que un proceso envía un mensaje y **b** es el evento de recepción de ese mensaje por otro proceso, entonces **a** → **b** también es verdadero.

Conceptos importantes:

- La ocurrencia anterior es una **relación transitiva**, por lo que si **a** → **b** y **b** → **c**, entonces **a** → **c**.
- Si dos eventos **a** y **b** ocurren en dos procesos diferentes que no intercambian mensajes, entonces **a** → **b** no es verdadero y tampoco **b** → **a**. Se dice entonces que los eventos son **concurrentes**.

3. Relojes lógicos

Lamport

Se necesita una forma de medir la **noción del tiempo** de tal forma que para cada evento **a** se puede asignar un valor de tiempo **C(a)** en el que **todos los procesos coincidan**.

Estos valores deben tener la propiedad de que si **a → b** entonces **C(a) < C(b)**.

El tiempo de reloj **C** siempre **aumenta**, nunca disminuye.

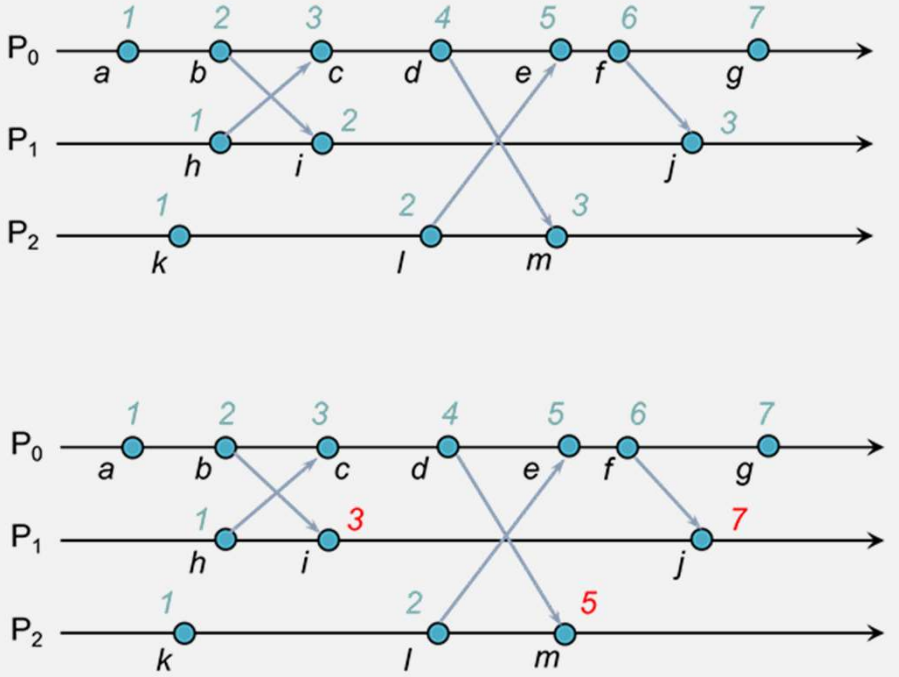
Solo es posible hacer correcciones agregando un valor positivo.

El funcionamiento consiste en **dos momentos**:

1. **Envío del evento**: el proceso debe incrementar su reloj y enviar su tiempo.
2. **Recepción del evento**: el proceso debe comparar entre su tiempo actual y el que recibió. Incrementar el valor máximo entre ambos y tomarlo como su nuevo tiempo.

3. Relojes lógicos

Lamport



3. Relojes lógicos

Relojes vectoriales

La solución de Lamport es buena pero **la ocurrencia anterior no siempre asegura a → b**.

Los relojes de Lamport no capturan la causalidad.

La causalidad sí que puede ser capturada por los **relojes vectoriales**.

La solución consiste en que cada proceso tiene un **contador** o **reloj de tiempo vectorial** de cada proceso en el sistema.

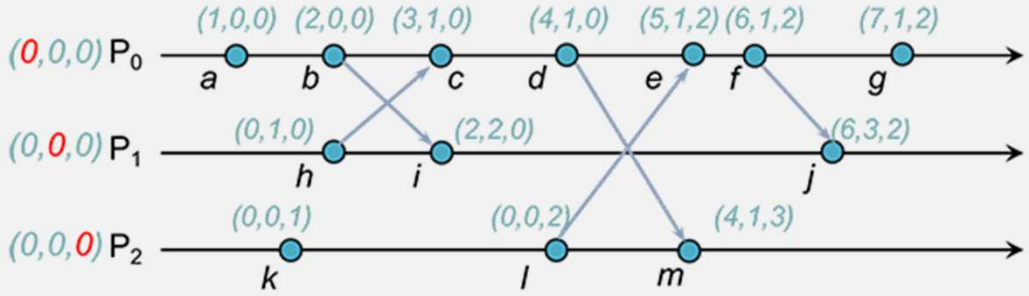
Si tenemos tres procesos, cada proceso tendrá un reloj vectorial con tres posiciones, de forma que **cada proceso sabe el orden de los eventos del resto**.

3. Relojes lógicos

Relojes vectoriales

El funcionamiento también consiste en **dos momentos**:

1. **Envío del evento**: el proceso debe incrementar su propio reloj y enviar el reloj vectorial.
2. **Recepción del evento**: el proceso debe comparar los tiempos actuales con los tiempos que recibió y al valor máximo entre ellos dejarlo en su reloj vectorial e incrementar su propio tiempo.



4. Exclusión mutua

Coordinación distribuida

Dado un conjunto de procesos en un sistema distribuido se necesita:

1. **Coordinar** sus acciones
2. Llegar a un **acuerdo** en uno o más valores

Formas de coordinación y acuerdo:

- **Acceso a recursos**: exclusión mutua distribuida
- **Selección de valores**: algoritmos de elección
- **Comunicación distribuida**: algoritmos de multidifusión
- **Toma de decisiones**: algoritmos de consenso

4. Exclusión mutua

Exclusión mutua distribuida

Conceptos básicos de acceso a recursos:

- **Sección crítica**: porción de código que permite el acceso a un recurso compartido por varios procesos.
- **Exclusión mutua**: el acceso a la sección crítica se regula por medio de semáforos.
- **Exclusión mutua distribuida**: el acceso a la sección crítica se basa en el **paso de mensajes**.

Estructura básica:

1. **Entrar a la sección crítica**: bloque del proceso si la sección crítica está ocupada.
2. **Acceder a los recursos**: uso de los recursos compartidos.
3. **Salir de la sección crítica**: liberación de procesos bloqueados.

4. Exclusión mutua

Exclusión mutua distribuida

Requisitos

- **Seguridad**: sólo un proceso puede estar ejecutándose a la vez en la sección crítica.
- **Pervivencia**: Las peticiones de entrada y salida de la sección crítica son concedidas.
- **Ordenación**: Si una petición de entrada ocurrió antes que otra entonces se garantiza la entrada a la sección crítica en ese orden.

Criterios de evaluación

- **Retraso de entrada**: cuanto tarda el proceso en entrar a la sección crítica tras solicitarlo
- **Retraso en el relevo**: tiempo de acceso a la sección crítica tras la salida de otro proceso.
- **Ancho de banda**: mensajes enviados en cada operación de entrada y salida de la sección crítica.

4. Exclusión mutua

Exclusión mutua distribuida

Conceptos importantes sobre la pervivencia

Las peticiones de entrada y salida de la sección crítica son concedidas.

Interbloqueo o *deadlock*

El interbloqueo es una situación que se produce cuando cada proceso tiene un recurso y espera a un recurso retenido por otro proceso.

Inanición o *starvation*

La inanición es una situación en la que a un proceso se le niega perpetuamente los recursos necesarios para poder realizar su trabajo.

4. Exclusión mutua

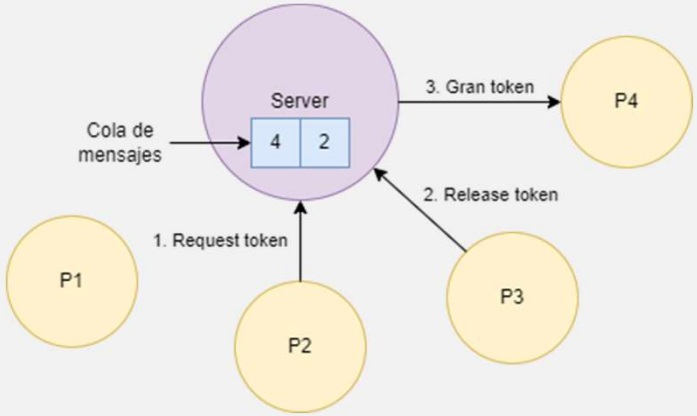
Algoritmo con servidor central

Un servidor concede los permisos para entrar a la sección crítica.

Los permisos se conceden a partir de un token que se envía por mensaje.

Tolerancia a fallos

- La caída del servidor es crítica.
- La caída de los clientes se puede solucionar con temporizadores o *timeouts*.



4. Exclusión mutua

Algoritmo con servidor central

Cumplimiento de requisitos

- Seguridad: Sí, existe un único token que mantiene un único proceso.
- Pervivencia: Sí, el turno de cada proceso siempre llega.
- Ordenación: No, no considera los tiempos locales en los que se enviaron los mensajes.

Rendimiento

- Entrada: dos mensajes, petición y concesión.
- Relevo: dos mensajes, liberación y nueva concesión.
- Ancho de banda: el servidor actúa de cuello de botella: realiza los mensajes de concesión y liberación

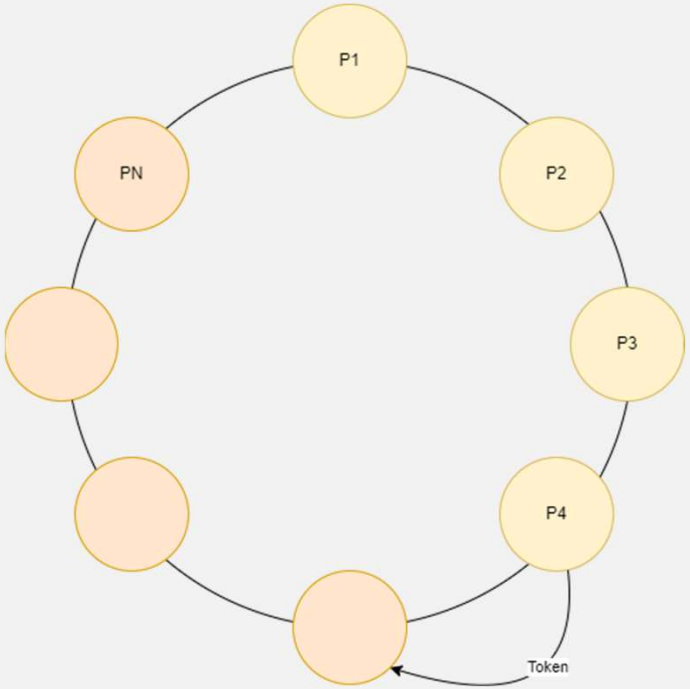
4. Exclusión mutua

Algoritmo basado en anillo

Paso del token sin servidor central.

Se da a cada proceso la dirección de su vecino.

- El token siempre está circulando por el anillo.
- Cuando un proceso recibe el token.
  - Si no quiere entrar en la sección crítica lo envía a su vecino.
  - Si quiere entrar en la sección crítica lo retiene.
- Tolerancia a fallos.
  - Pérdida del token: detección y regeneración
  - Caída de un proceso del anillo: reconfiguración del anillo.



4. Exclusión mutua

Algoritmo basado en anillo

Paso del token sin servidor central.

Cumplimiento de requisitos

- Seguridad: Sí, el servidor se encarga de ello a través de un token único.
- Pervivencia: Sí, todas las peticiones se registran en la cola.
- Ordenación: No, cada proceso recibe el token cuando se lo pasa su vecino.

Rendimiento

- Entrada: de 0 a N mensajes.
- Relevo: de 1 a N – 1 mensajes.
- Ancho de banda: consumo continuo salvo cuando un proceso está en la sección crítica.

4. Exclusión mutua

Algoritmo basado en anillo

Paso del token sin servidor central.

Problemas

- Carga de la red incluso si ningún proceso entra en la sección crítica.
- Si un proceso cae se necesita reconfiguración
    - Si además tenía el token se necesita un algoritmo de elección para regenerar el testigo.
- Para asegurarse de que el sistema está caído es necesario generar varios tokens.
- Una desconexión o ruptura de la red invalida el algoritmo.

4. Exclusión mutua

Algoritmo de Ricart y Agrawala

Algoritmo basado en **relojes lógicos** y **multidifusión**: asegura seguridad, pervivencia y ordenación.

Algoritmo **descentralizado**: evita cuellos de botella.

Basado en el algoritmo de Lamport.

- Cada proceso conoce la dirección de los demás procesos.
- Cada proceso posee un reloj lógico.

Concepto general del algoritmo

1. Cuando un proceso quiere entrar en la sección crítica pregunta a los demás si puede entrar.
2. Cuando todos los demás procesos contestan entonces entra.

4. Exclusión mutua

Algoritmo de Ricart y Agrawala

El acceso se obtiene a través de un token.

Cada proceso guarda el estado en relación a la sección crítica: **liberada**, **buscada** o **tomada**.

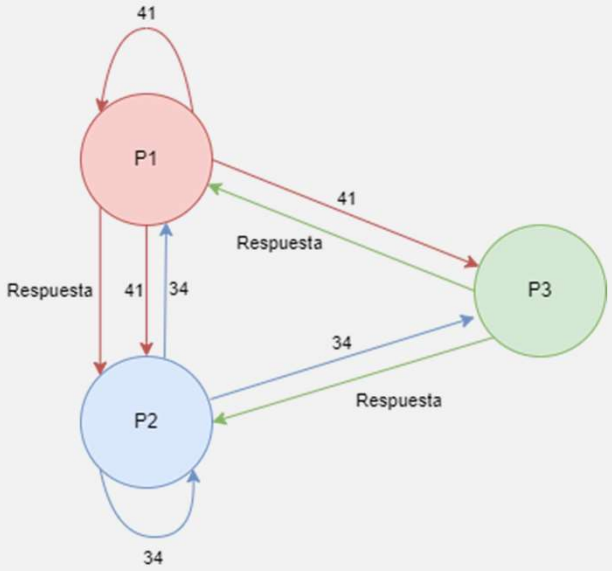
Cada proceso tiene disponible la cola de solicitudes.

Rendimiento

- Entrada: N - 1 mensajes de petición y N – 1 mensajes de concesión.
- Relevo: Un mensaje (el del proceso en la sección crítica)
- Ancho de banda: 2N – 2

Problemas

- Costoso, el fallo de cualquier proceso bloquea el sistema, congestión.



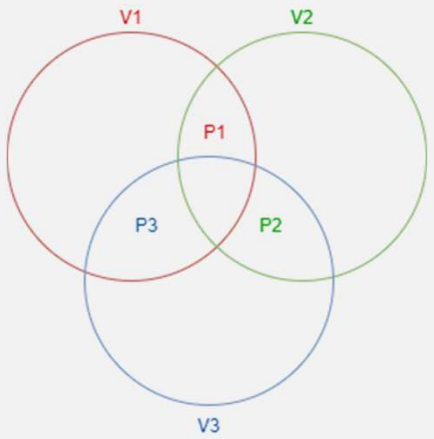
4. Exclusión mutua

Algoritmo de Maekawa

Análogo del algoritmo anterior pero reduciendo el número de mensajes.

- No se necesita que todos los procesos permitan el acceso.
- Basta con obtener el permiso de un subconjunto K de procesos.
- Siempre que los subconjuntos usados por un par de procesos se solapen con ciertos criterios.

N = 3	Número de procesos
K = 2	Procesos por conjunto de voto
M = 2	Conjunto de voto por proceso



4. Exclusión mutua

Algoritmo de Maekawa

Requisitos

- Cumple con la seguridad
- No cumple con la pervivencia pues puede producir interbloqueos a partir de 3 procesos.

Solución mejorada por **Saunders** mediante el uso de la **ocurrencia anterior**: se cumple pervivencia y ordenación.

Esta solución con conjunto de votos solapados se utiliza para direccionamiento P2P.

Rendimiento

- Entrada: $\sqrt{N}$ mensajes de petición y $\sqrt{N}$ mensajes de concesión.
- Relevo: $\sqrt{N}$
- Ancho de banda: $3\sqrt{N}$

5. Elección y consenso

Elección y consenso en sistemas distribuidos

Elegir un **proceso único** para que tome un determinado **rol** o para decidir una determinada **acción**.

Aplicaciones

- Elegir un nuevo servidor si se cae el actual.
- Elegir un nuevo proceso para entrar en una sección crítica.
- Elegir el proceso menos activo (balanceo de carga).
- Elegir el proceso con la copia más reciente (réplicas).

5. Elección y consenso

Elección y consenso en sistemas distribuidos

Un proceso convoca **elecciones** cuando lleva a cabo una acción que inicia el algoritmo de elección.

Puede haber N elecciones **concurrentes**.

Un proceso siempre toma uno de estos **roles**.

- **Participante**: comprometido en una ejecución del algoritmo.
- **No participante**: no comprometido con ninguna elección.

El proceso elegido debe ser **único** incluso en elecciones concurrentes.

Todos los procesos tienen un **identificador único** para el conjunto y totalmente **ordenados**.

El proceso elegido es aquel de **mayor identificador**.

5. Elección y consenso

Elección y consenso en sistemas distribuidos

Cada proceso p_i mantiene una variable e_i que contiene el identificador del proceso elegido.

Cuando el proceso se convierta en participante fija la variable al valor especial $\perp$ indicando que no hay consenso aún.

Requisitos

- Seguridad: un proceso participante p_i tiene $e_i = \perp$ o $e_i = P$ donde P es el proceso elegido con mayor identificador.
- Pervivencia: Todos los procesos p_i participan y al final fijan e_i != P, independientemente de los procesos que caigan.

Rendimiento

- Ancho de banda: proporcional al número de mensajes enviados

5. Elección y consenso

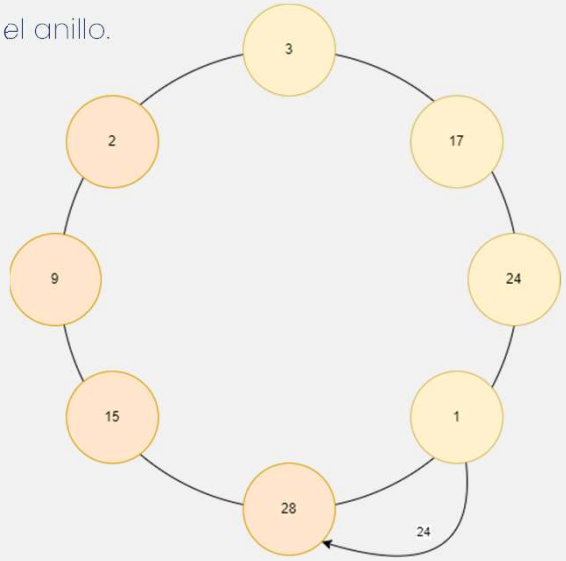
Algoritmo en anillo (Chang y Roberts, 1979)

Los procesos se consideran dispuestos en un anillo lógico.

Un proceso convoca **elecciones** y pasa su identificador al siguiente proceso en el anillo.

- Si el identificador del proceso es mayor que el recibido lo cambia.
- Cuando el mensaje llega al proceso convocante finaliza la elección

y se transmite siguiendo el mismo orden el identificador elegido.



5. Elección y consenso

Algoritmo en anillo (Chang y Roberts, 1979)

Los procesos se consideran dispuestos en un anillo lógico.

1. Cada proceso se etiqueta como no participante.
2. Cualquier proceso convoca elecciones.
    - Se marca a sí mismo como participante y pone $e_i = \perp$.
    - Pone su identificador en un mensaje elección y lo envía al vecino.
3. Un proceso recibe un mensaje elección con identificador.
    - Lo cambia si es mayor que el suyo.
    - Si es igual se convierte en coordinador, se marca como no participante y envía el mensaje elegido al vecino.
4. Un proceso recibe un mensaje elegido.
    - Se marca como no participante y fija e_i al valor del identificador del mensaje.
    - Envía su mensaje elegido al siguiente vecino a no ser que sea coordinador.

5. Elección y consenso

Algoritmo en anillo (Chang y Roberts, 1979)

Requisitos

- **Seguridad**: Sí, tras la primera vuelta se obtiene el proceso activo con identificador más alto.
- **Pervivencia**: Sí, la última vuelta asegura que todos los procesos activos conocen el resultado de la elección.

Rendimiento

- Peor caso: el nuevo elegido es el vecino antihorario del convocante.
- Ancho de banda: 3N – 1 mensajes
    - N – 1 mensajes de elección hasta alcanzar al vecino antihorario.
    - Otros N mensajes de elección con identificador del vecino antihorario.
    - N mensajes de elegido.

5. Elección y consenso

Algoritmo del matón o *bully*

Todos los procesos deben conocer las identidades y direcciones del resto de procesos.

Se presupone una comunicación fiable.

El algoritmo selecciona al proceso superviviente con mayor identificador.

Permite la caída de procesos durante la elección: utiliza *timeouts* para detectar fallos de procesos.

Existen 3 tipos de mensajes:

- Mensaje de **elección**: para anunciar una elección.
- Mensaje de **respuesta**: respuesta a un mensaje de elección.
- Mensaje de **coordinador**: para anunciar la identidad de un nuevo coordinador.

5. Elección y consenso

Algoritmo del matón o *bully*

1. El convocante envía mensajes elección al proceso o procesos con mayor identificador.
2. Si ninguno le responde multidifunde que es el nuevo coordinador.
3. Si alguno le responde

- El convocante inicial queda en espera
- Los procesos que responden inician un nuevo proceso de elección como convocantes (vuelta a paso 1).

La elección comienza cuando un proceso se da cuenta (por los timeouts) de que el coordinador ha fallado.

Varios procesos pueden descubrir el fallo de forma concurrente.

5. Elección y consenso

Algoritmo del matón o *bully*

1. Si un proceso sabe que tiene el identificador no fallido más alto se elige a sí mismo coordinador enviando el mensaje coordinador a todos los de identificador más bajo (proceso de matón).
2. Si no tiene el identificador no fallido más alto.
    - Si tras un tiempo no recibe ningún mensaje se vuelve al paso 1.
    - Si recibe un mensaje respuesta entonces espera al mensaje del coordinador.
        - Si recibe el mensaje coordinador fija su variable e_i al identificador del mensaje.
        - Si no recibe mensaje coordinador comienza otra nueva elección.

Si un mismo proceso se recupera o se lanza un nuevo proceso sustituto se comienza una nueva elección aunque el coordinador actual esté funcionando.

5. Elección y consenso

Algoritmo del matón o *bully*

Requisitos

- **Pervivencia**: todos fijan e_i mediante la multidifusión final.
- **Seguridad**: tiene problemas de seguridad como la recepción de dos mensajes coordinador con identificadores distintos o que los valores de los timeout no sean precisos (el sistema no es síncrono).

Rendimiento

- Mejor caso: el proceso con el segundo identificador más alto detecta el fallo del coordinador N – 2 mensajes.
- Peor caso: el proceso con identificador más bajo detecta el fallo del coordinador $O(N^2)$.

5. Elección y consenso

Algoritmo de invitación

Problemática de los algoritmos anteriores: **la presencia de errores en entornos asíncronos**.

- Basados en timeouts: los retrasos de transmisión pueden causar la elección de múltiples líderes.
- La pérdida de conexión entre dos grupos de procesadores puede aislar permanentemente los procesadores.

Resolución

- Definición de grupos de procesadores con líder único.
- Detección y agregación de grupos.
- Reconocimiento por parte del líder de los miembros del grupo.

5. Elección y consenso

Algoritmo de invitación

Procedimiento

1. Si un procesador detecta la pérdida del líder, entonces se declara líder y forma su propio grupo.
2. Periódicamente el líder de cada grupo busca otros líderes de otros grupos.
3. Dos grupos se unen por medio de mensajes de aceptación.
    - Como respuesta a mensajes de aceptación.
    - De forma explícita.

5. Elección y consenso

Algoritmo de invitación

Procedimiento

1. Si un procesador detecta la pérdida del líder, entonces se declara líder y forma su propio grupo.
2. Periódicamente el líder de cada grupo busca otros líderes de otros grupos.
3. Dos grupos se unen por medio de mensajes de aceptación.
    - Como respuesta a mensajes de aceptación.
    - De forma explícita.

5. Elección y consenso

Consenso distribuido

En un entorno distribuido los procesos tienen dificultades para ponerse de acuerdo en un valor cuando dos o más procesos han propuesto valores distintos.

Protocolos de acuerdo específicos

- Exclusión mutua: consenso en quién entra en la sección crítica.
- Elección: consenso en quién es el nuevo coordinador.
- Multidifusión: consenso en el orden de entrega de los mensajes.

El **consenso** llega por un **protocolo específico prefijado**.

5. Elección y consenso

Consenso distribuido: modelo del sistema

- Sean N procesos p_i que se comunican por paso de mensajes.
- La comunicación es fiable pero los procesos pueden fallar.
  - Por caída de los procesos
  - Por fallos arbitrarios (bizantinos)
- Hasta *f* de los N procesos pueden fallar.
  - Se debe llegar a un consenso incluso en este caso.

5. Elección y consenso

Consenso distribuido: definición

- Cada proceso p_i comienza en el estado no decidido.
- Cada proceso p_i propone un único valor vi de un conjunto de posibles valores E.
  - Es el valor que quiere que tenga la variable e_i.
- Los procesos se comunican entre sí intercambiando valores.
  - Buscan un valor de consenso por diferentes criterios: mayoría, mínimo, máximo, etc.
- Cada proceso fija el valor de la variable e_i sobre la que se busca consenso
  - El estado pasa a decidido.
  - En el estado decidido ya no se puede cambiar el valor e_i.

5. Elección y consenso

Consenso distribuido: condiciones

Terminación

Cada proceso correcto ha de fijar su variable de decisión.

Acuerdo

El valor de decisión de los procesos correctos es el mismo.

Integridad

Todos los procesos correctos han propuesto el mismo valor.

5. Elección y consenso

Consenso distribuido: el problema de los generales bizantinos

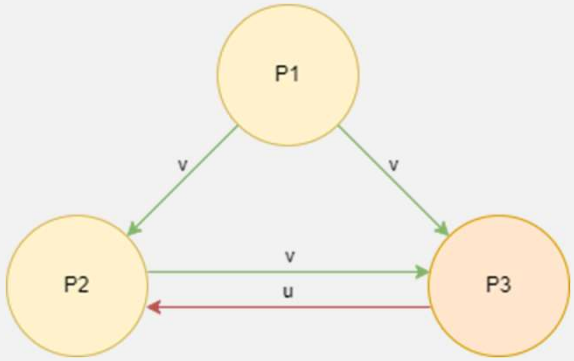
¿Qué ocurre su un proceso propone un valor **distinto** al resto?

- N generales deben acordar si atacan o se retiran.
- Uno de los dos generales (el comandante) da la orden.
- **El comandante define el valor de consenso.**
- Puede que ese valor haya sido corrompido (fallo arbitrario *f*) por el comandante u otros generales.
- El problema tiene solución si N >= 3*f* + 1.

5. Elección y consenso

Consenso distribuido: el problema de los generales bizantinos

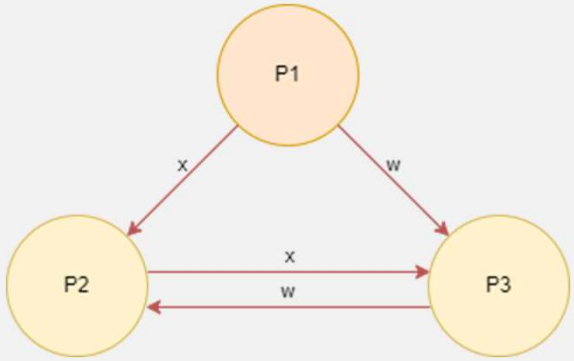
1. El comandante P1 manda el valor correcto (v) pero P3 falla y lo corrompe (u).
   - Tras dos rondas P2 no puede determinar qué valor es correcto.



5. Elección y consenso

Consenso distribuido: el problema de los generales bizantinos

2. El comandante P1 falla y envía mensajes contradictorios.
   - De nuevo P2 no puede determinar qué valor es el correcto ni qué proceso es el que ha fallado.



5. Elección y consenso

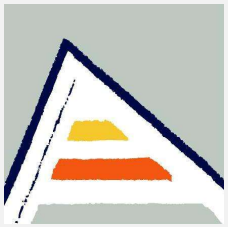
Consenso distribuido: conclusiones

¿Imposibilidad de consenso en sistemas asíncronos?

Fischer (1985) probó que en sistemas totalmente asíncronos donde uno o varios procesos pueden fallar, un algoritmo nunca puede garantizar el consenso en todos los casos.

No se pueden hacer asunciones de los tiempos de procesamiento o de los retardos en los envíos de mensajes.

Fischer no indica que el consenso no se alcance nunca si no que existe la posibilidad de que no se alcance.



Grado en Ingeniería Informática

Sistemas distribuidos

Sincronización de sistemas distribuidos

Víctor Vives
vvives@dtic.ua.es

Departamento de Tecnología Informática y Computación

2021 - 2022