Tema 1

Tipos de fallos:

- Crash failure
  - fail-stop: detectado por otros.
  - fail-silent: no detectado
- Omission failure
  - receive
  - send
- Timing failure: proceso miente sobre un intervalo temporal
- Response failure
  - value failure: error en el valor de respuesta
  - transition failure: el proceso se desvía del funcionamiento correcto
- Arbitrary (Bizantine) failure: proceso produce mensajes arbitrarios en tiempos arbitrarios, omite commits o steps.

Servers:

- Stateless: no mantienen data del cliente, pueden cambiar estado sin informar.
- Stateful: mantiene el estado del cliente

Tema 2

Tipos de comunicación:

- Space decoupling: sender y receiver no necesitan conocer sus identidades
- Time decoupling: sender y receiver no necesitan existir en el mismo intervalo temporal

- Transient communication: el mensaje se descarta si el receiver no está activo en el momento de delivery
- Persistent communication; el mensaje se guarda por el middleware hasta que el receiver está activo para el delivery

- Synchronous communication: send y receive son operaciones bloqueantes
- Asynchronous communication: send es no bloqueante y receive puede ser ambas

- Discrete communication: intercambio de información independientes
- Continuous communication: intercambio continuo de información

Communication paradigms:

- Remote invocation: transparent, direct, synchronous, transient
- Message passing: direct, synchronous, transient
- Message queuing: asynchronous, persistent
- Publish-subscribe: space decoupling, asynchronous

- Shared data space: persistent, asynchronous
- Shared memory: space decoupling
- stream oriented: direct, transparent, continuous

RPCs semantics and failures:

- Client's request is lost
- Server's reply is lost
- Server crashes after receiving a request
- Client crashes after sending a request

  Handling 1, 2, 3 failures:
  - Retry request message
  - Duplicate filtering: cliente asigna un identificador único a cada request, el server filtra los duplicados para no repetir ejecuciones
  - Retransmission of results: el server mantiene un historial de resultados a reenviar en replies perdidas para no hacer ejecuciones repetidas

  Handling 4:
  - Crash de cliente genera calls huérfanas
  - Las calls huérfanas deben ser eliminadas
    - ExterminationCliente hace un request a otros nodos para matar a sus huérfanos. Cada nodo huérfano mata a los huérfanos y a sus descendientes
    - Expiration: Nodos remotos abortan las RPCs cuando el limite temporal expira
    - Reincarnation: Divide el tiempo en epochs, después de una crash recovery hay una nueva epoch. Los nodos remotos reencarnan y matan los RPCs de epochs anteriores.
    - Gentle reincarnation: Igual que la anterior pero solo matan si el owner no responde.

Publish subscribe systems:

- Topic based: Se identifica por keywords
- Content based: Depende del valor de los atributos
- Type based: Puede basar se en atributos o métodos

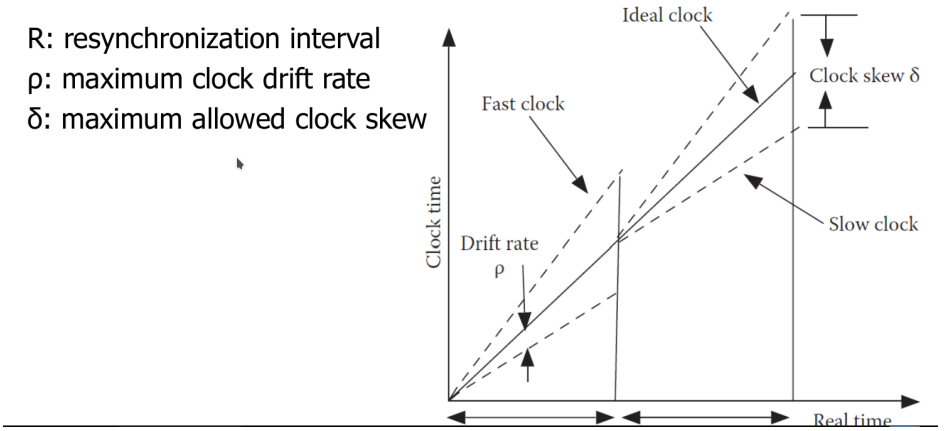
Event routing:

- Flooding: Le envia a todos los nodos y ellos descartan
- Subscription flooding: Hace un flooding a todos los nodos avisando de que eventos quiere recibir para que estos se lo envien cuando publiquen.
- Filtering based: Cada nodo guarda el set de suscripciones que se alcanzan desde sus vecinos, sólo envía a los que tienen a un subscriptor en el path
- Rendezvous-based: Particiona suscripciones y eventos entre los nodos
- Gossiping: Cada nodo escoge a otro random e intercambian eventos.

Tema 3

- Synchronize at least every R < δ/2ρ to limit skew between two clocks to less than δ time units

R: resynchronization interval
ρ: maximum clock drift rate
δ: maximum allowed clock skew



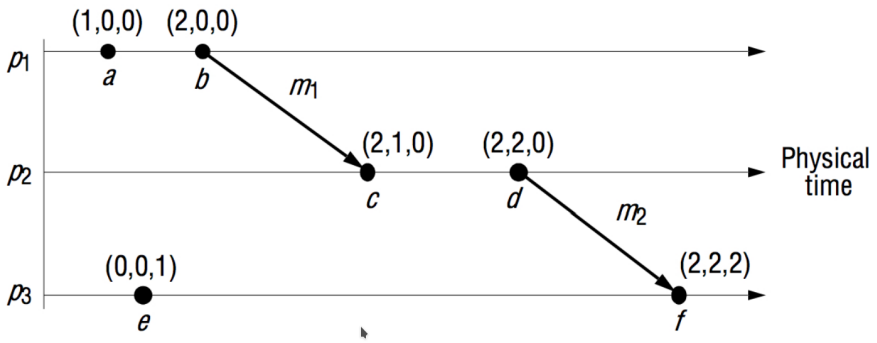
- Christian's algorithm: Sincronizar nodos con un servidor. Cada cliente pregunta el tiempo cada R intervalo.
- Berkeley's algorithm: Los relojes se sincronizan entre ellos. Todos le envian al master su clock, él hace una media y envía recticficaciones.

Lamport's logical clocks:

- Cada proceso mantiene su contador local.
- Cuando hay un evento el contador se incrementa
- Si recibes un mensaje con un clock más avanzado el proceso pasa a tener ese clock+1

Vector clocks:

- Cada proceso tiene un vector de clocks donde el tamaño del VC lo define la cantidad de procesos existentes.
- Cuando hay un evento en el propio proceso se aumenta en uno el valor del proceso en el VC, si se recibe algo también se aumenta. Si se recibe un mensaje donde los valores para las distintas posiciones del VC son mayores se adopta ese reloj, añadiendo uno a la posición de nuestro proceso.



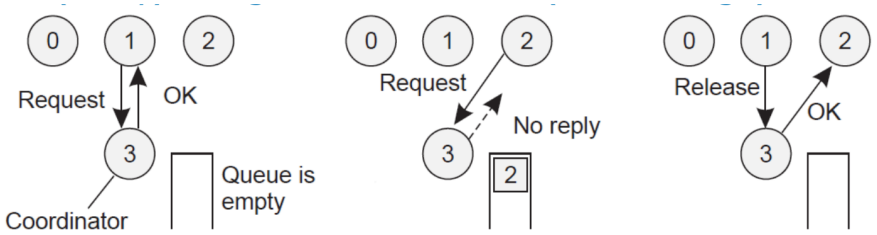
Tema 4

Mutual exclusion:

- Safety: solo debe haber un único proceso en la zona crítica.
- Lieness: los requests para entrar o salir tienen que tener éxito.
- Happened before ordering

Centralized algorithm:

- Para entrar a la zona crítica se envía un request al coordinador
- Si no hay nadie se permite, sino se encola el request
- El proceso que está en la zona envía un release al coordinador, el coordinador da el ok al proceso en espera.

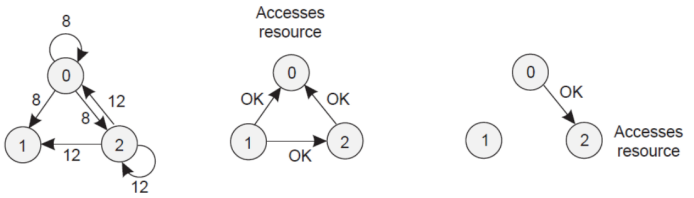


Lin's voting algorithm:

- Para entrar en la zona crítica necesita el ok de la mayoría, M > N/2.
- Envía request a todos los nodos incluyendo su Lamport's clock y process ID.
- Coordinador envía una response diciendo si el request se ha encolado o ok.
- Requester analiza, si tiene M > N/2 entra, sino no hace nada pues está en cola.
- Si nadie obtiene mayoría se envía un yield(release+request) a todos los que han votado por él.
- Los coordinadores borran el voto, encolan el nuevo request, vota por el que esté en el head de la cola y le envía un response, si el voto cambia informa al otro proceso.
- Para salir de la zona se envía un release a todos los coordinadores, cada coordinador borra el voto, vota al head de la queue y envía response.

Ricart & Agrawala's algorithm:

- Proceso envía request a todos los procesos con su Lamport's clock e ID. Todos los procesos deben conocerse.
- Cuando el requester reciba ok de todos los procesos puede acceder a la zona.
- Cuando un proceso recibe un request si no está intentando acceder envía ok, si está accediendo actualmente encola el request, si quiere acceder pero aún no lo ha hecho envía ok solo si el request que le llega tiene un tiempo lógico más bajo, sino no responde.
- Para salir de la zona crítica el proceso envía un ok al resto de procesos en la cola.

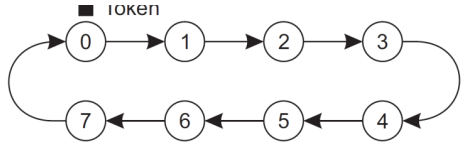


Maekawa's algorithm:

- El permiso se obtiene de parte de un subset, todos los subsets tienen igual tamaño.
- Cada proceso pertenece a varios subsets.
- La intersección de dos subsets es no vacía.
- El proceso envía un request a todos los componentes del subset, para acceder necesita que todos devuelvan ok.
- Cuando un proceso recibe el request si no está accediendo o no ha votado por otro proceso da ok, sino encola el request.
- Para salir se envía release a todos los procesos del subset y estos envían un ok al head de la queue.

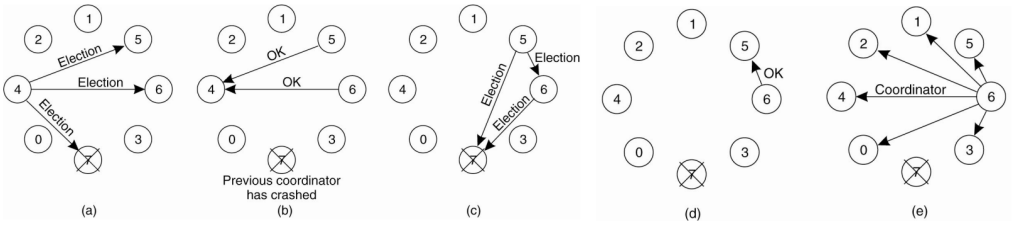
Token ring algorithm:

- Los procesos se ordenan formando un anillo unidireccional.
- Solo el proceso que tenga el token puede entrar.



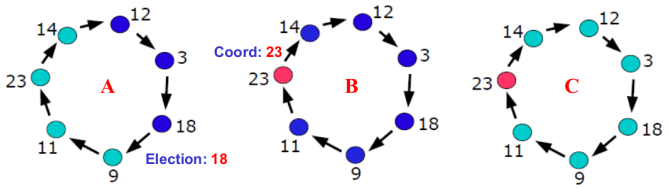
Bully algorithm:

- Proceso envía un mensaje de elección a todos los procesos con ID mayor.
- Si un proceso recibe un mensaje de elección devuelve ok e inicia una nueva selección si aún no lo ha hecho.
- Si el proceso recibe un ok entiende que no va a ser coordinador y espera mensajes del coordinador.
- Si el proceso no recibe ningún ok en un límite de tiempo se convierte en coordinador y envía mensajes de coordinador al resto.



Chang and Robert's ring algorithm:

- Los procesos se organizan en un anillo unidireccional organizado por IDs.
- El proceso envía un mensaje de elección a su sucesor con su ID y se convierte en participante.
- Si el sucesor aún no era participante le envía a su sucesor el ID más grande entre el que le ha llegado y el suyo, si ya era participante y es más grande lo envía, sino no.
- Si a un proceso le llega su propio ID se convierte en coordinador y envía un mensaje de coordinador.



Enhanced ring algorithm:

- El proceso le envía un mensaje de elección a su sucesor vivo más cercano.
- Cada proceso va añadiendo su ID.
- Cuando llega de vuelta al primer proceso el de ID más alto es el coordinador.

Ordered Multicast:

- FIFO
- CAUSAL: garantiza FIFO
- TOTAL: todos los procesos tienen el mismo orden

Tema 5

Linearizability:

- Mismo objetivo que strong consistency(exclusive ordered writes + up to date monotonic reads) pero aceptando que pueden haber overlaps

Overlapping:

- Puede producirse en cualquier orden pero tiene que ser visible para todo el mundo

Quorum based protocols:

- Los clientes deben obtener el permiso de un quorum para poder escribir o leer
- Nr + Nw > N
- Nw > N/2 para evitar conflictos de escritura
- Puede ser linearizable pero hay que añadir pasos adicionales
    - Usar vectores de versiones
    - Usar read-repair

Replicated write protocol:

- Active replication
    - k+1 procesos pueden tolerar k crash/omission failures
    - 2k + 1 procesos pueden tolerar Byzantine failures
- Quorum based
    - 2k + 1 procesos pueden tolerar crash/omission failures
    - 3k + 1 procesos pueden tolerar Byzantine failures