TEMA 5

∘ GESTIÓN DE FALLOS

FALLO → Componente incapaz de comportarse como toca

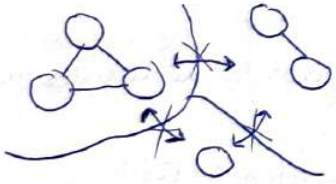
- ∘ DEFECTO ('fault'): Comportamiento / Condición anómala
  → Reintentar, si sigue → ERROR
- ∘ ERROR: Manifestación de un defecto
  → Se sustituye, si no hay réplicas → FALLO
- ∘ FALLO ('failure'): Incapacidad de un elemento para hacer sus funciones VISIBLE PARA EL USUARIO

TRANSPARENCIA / TOLERANCIA A DEFECTOS → Capacidad de un sistema para reestructurarse y seguir funcionando a pesar de los defectos.
Para que un sistema lo sea → todos sus sub-servicios también, con replicación x ej.

SISTEMA DE DETECCIÓN DE DEFECTO → Permite reintentar o rehacer una operación en caso de defecto, si despues de reintentar no se soluciona se considera error. Se intenta sustituir el componente con replicación y si no, se convierte en fallo visible para el usuario.

Cuando se diseñan algoritmos distribuidos asumimos algún ↓

∘ MODELO DE FALLOS



- DETECTABLE: No contesta dentro de un plazo / Respuesta absurda
- NO DETECTABLE: Bizantino

- SIMPLE: Solo afecta a un nodo/componente
- COMPUESTO: Varios fallos simultáneos en nodos distintos

} Nosotros asumimos fallos simples detectables

PARTICIONES / FALLOS DE RED: Cuando en una red fallan las comunicaciones y varios nodos quedan aislados.
Se comunican los nodos particionados pero NO entre particiones.

AP ∘ SISTEMA PARTICIONABLE → Los grupos pueden continuar con un protocolo de reconciliación

CP ∘ MODELO DE PARTICIÓN PRIMARIA → Continúa el grupo con mayoría de nodos
Evita inconsistencias.

TEOREMA CAP → Solo puedes garantizar 2 de entre { Disponibilidad, Consistencia, Particiones } En particiones garantizamos AP / CP

- REPLICACIÓN
  - Mecanismo básico para asegurar la disponibilidad de un componente + facilita la recuperación tras fallo
  - Cada réplica en una máquina distinta → independientes → si falla una las demás NO deben fallar
    - Las operaciones en curso en la réplica caída podrían completarse en otra
    - Las activas se toman como fuente para la reconfiguración de la caída.

MEJORA DE RENDIMIENTO

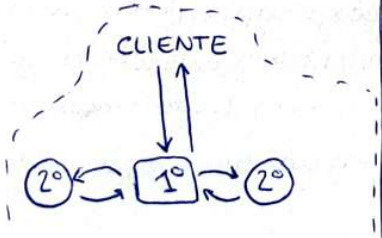
- Operaciones de solo lectura → Idempotentes → El resultado solo depende de los argumentos
  - Ejecutadas por una sola réplica / o todas
  - Facilita la escalabilidad del sistema
  - Evita que las réplicas se actualicen entre sí todo el rato
- Operaciones de escritura → NO idempotentes → dependen del estado (historia) + argumentos
  - Aplicadas en todas las réplicas → retardos
  - Si la operación es sencilla, una réplica realiza la operación y luego la comunica.
  - Los estados pueden divergir → determinar modelo de consistencia

REPLICACIÓN PASIVA

- El cliente envía la petición a la réplica primaria, la misma para todos los clientes y peticiones
- Esta réplica ejecuta la operación y
  - Pesimista (Lento) con garantía → 3. propagar 4. esperar ACK's 5. devolver
  - Optimista (Rápido) sin garantía → 3. devolver 4. propagar 5. esperar ACK's

PROPAGA = cuando cambia el estado

- Caso Fallo → Si es una copia = NADA
  - La 1º: envía un latido cada cierto tiempo, si no se recibe, ha fallado
- Ventajas →
  - Carga mínima
  - Distribuir lecturas
  - Establecer orden operaciones
  - Operaciones NO deterministas
- Inconvenientes →
  - Reconfiguración pesada cuando falla 1º
  - No soporta fallos bizantinos



REPLICACIÓN ACTIVA "maquina estados"

- El cliente envía su petición a TODAS las réplicas del servidor
- Cada réplica ejecuta la operación y la devuelve al cliente
- Los msg. de los clientes deben secuenciarse para que lleguen igual.
- Se deben usar mecanismos de concurrencia para acceder a recursos compartidos.
- Ventajas →
  - Reconfiguración Trivial, ignoras la réplica fallida
  - Soporta fallos bizantinos
- Inconvenientes →
  - Consistencia Fuerte
  - No operaciones NO deterministas
  - Mecanismo para filtrar solicitudes duplicadas

CONSISTENCIA

- Cuando se replica información en múltiples nodos → especifica QUÉ divergencias se admiten entre los valores de las réplicas.
- Cuando un cliente / proceso realiza escritura → propaga los cambios a las demás réplicas
  - ↳ La consistencia obtenida depende de { retardo de la propagación; esperas que el retardo genere en otros procesos }
- 2 enfoques {
  - RÁPIDO: { Al realizar la modificación se publica sin esperar a que se sincronicen todos los nodos. Rápido, poco fiable, puede haber discrepancias }
  - LENTO: { Primero se difunde la modificación, se verifica la consistencia Lento, con garantías de consistencia. }

MODELOS DE CONSISTENCIA

+ LENTO

Estricta (L) → Causal (R) → FIFO (R) → +RÁPIDO

Procesador (L) → Caché (L)

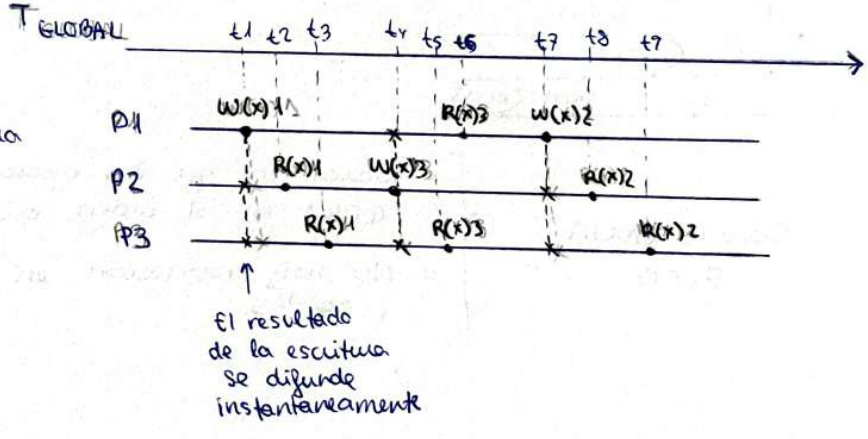
CONSISTENCIA FINAL

"Eventual consistency"

- Sistemas que admitan particiones
- Teorema CAP → Si priorizo la Disponibilidad NO puedo garantizar la Consistencia en todas las particiones.
- Aquí, cada partición seguirá realizando sus operaciones hasta que se decida un nuevo estado conjunto que refleje todos los cambios realizados, cuando se resuelvan las particiones.

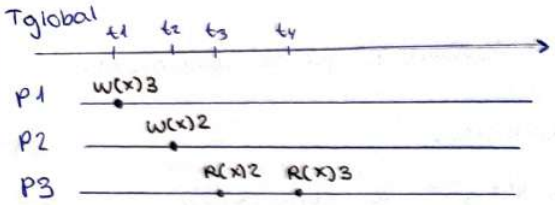
CONSISTENCIA ESTRICTA

- Como trabaja una consistencia fuerte
- Asume un reloj global que etiqueta cada evento
  - → NO pueden suceder 2 escrituras a la vez
  - → Propagación inmediata : Latencia 0
  - → Siempre se lee el último valor de la última escritura
- IMPOSIBLE DE IMPLEMENTAR
  - ↓
  - En un sist. distribuido no podemos hablar de reloj global



CONSISTENCIA SECUENCIAL

- Todos los nodos VEN el mismo orden pero NO siempre es el valor de la última escritura.
  - ↓
  - Llegan a un acuerdo sobre el orden en el que se ejecutarán o verán los cambios.
  - Todos aceptan el orden pero van a su ritmo
  - Las modificaciones de un mismo nodo llegan en orden a todas las réplicas

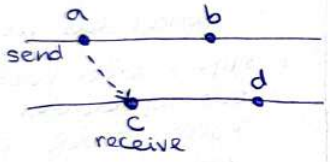


Todas las réplicas recibirán antes $x=3$ que $x=2$

Implementación ↑ garantías de consistencia →
- Todos los nodos tienen un socket PUSH y SUB
- PUSH → Envían el msg a un secuenciador
- Secuenciador envía cada cambio a los nodos. (PUSH y PUB)

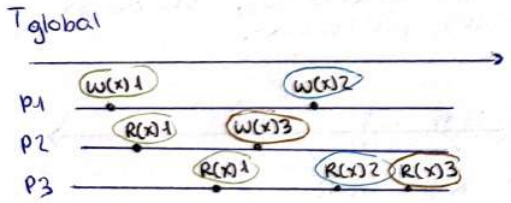
CONSISTENCIA CAUSAL

- Asegura que los eventos dentro de cada nodo ocurren en orden → El evento de "envío" siempre ocurre antes que "llegada".
- Establece un orden utilizando la propiedad transitiva $a \rightarrow b$, si NO → son concurrentes

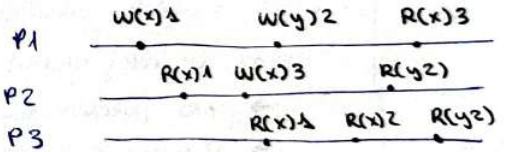


$a \rightarrow c \rightarrow d$ Pero b y d son concurrentes.

Lamport



R(x)1 siempre precede a R(y)2



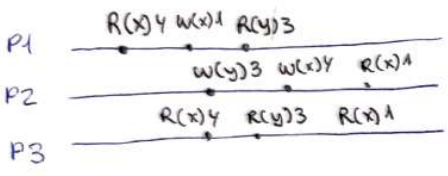
R(x)3 puede obtenerse cuando sea

CONSISTENCIA FIFO

- Garantiza que las operaciones de escritura realizadas por un MISMO proceso sean leídas en el orden que fueron escritas por todos los demás procesos
- No hay restricción en el orden en el que se leen las operaciones de escritura en diferentes procesos.

CONSISTENCIA CACHÉ

- Las escrituras realizadas sobre una misma variable sean vistas en el mismo orden por todos los procesos
- Sin restricción sobre lo que se haga en diferentes variables.
- Se usa un secuenciador por variable, pero NO hay uno global



El orden para la x es 1° 4 y luego 1
Mientras que el valor de y se puede ir mezclando.

Modelo de Consistencia	Definición Breve	Diferencias Relevantes	Características Clave
Estricta	Todas las operaciones de escritura son instantáneamente visibles para todas las réplicas.	Es un modelo teórico e ideal que no se puede implementar en la práctica.	Asume un reloj global, no pueden suceder dos escrituras a la vez en todo el sistema, se asume una propagación inmediata de las escrituras y una latencia cero.
Secuencial	Las operaciones de escritura se propagan a todas las réplicas en un orden específico.	Permite cierto retraso antes de que las operaciones de escritura sean visibles para las réplicas.	Todos los nodos llegan a un acuerdo sobre el orden en el que se ejecutarán o verán los cambios. Todos aceptarán este orden, pero cada uno avanzará a su propio ritmo.
Causal	Si una operación de escritura es visible para una réplica, entonces todas las operaciones de escritura que causaron esa operación también son visibles.	Mantiene el orden de las operaciones de escritura que están causalmente relacionadas.	Utiliza la propiedad transitiva para establecer un orden entre los eventos. Solo se establece un orden si se puede extrapolar con la propiedad transitiva.
FIFO	Las operaciones de escritura realizadas por un mismo proceso son leídas en el orden en que fueron escritas por todos los demás procesos.	No impone ninguna restricción en el orden en que se leen las operaciones de escritura realizadas por diferentes procesos.	Se ocupa del orden de las operaciones de escritura realizadas por un mismo proceso.
Caché	Las operaciones sobre una misma variable siempre se realizan en el mismo orden.	No hay un secuenciador global que garantice un orden específico para las operaciones entre diferentes variables.	Cada variable tiene su propio secuenciador que garantiza que las operaciones sobre esa variable se realizan en el mismo orden.