IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 14)

Transacciones Distribuidas
¿Cómo replicamos varias operaciones como una?

Temas de la clase

1. Introducción a Transacciones Distribuidas
    a. ¿Qué son?
    b. Desafíos
    c. Propiedades ACID
2. Coordinando transacciones distribuidas
    a. *Commit* de 2 fases
    b. Commit de 3 fases

Introducción a Transacciones Distribuidas

¿Qué son?

Desafíos

Propiedades ACID

Transacciones

- Es una **secuencia de operaciones** que un cliente solicita a un servidor, diseñada para ejecutarse como una **unidad indivisible**.
- Se busca asegurar consistencia y durabilidad de los datos frente a concurrencia y fallos.

Transacciones

- Es una **secuencia de operaciones** que un cliente solicita a un servidor, diseñada para ejecutarse como una **unidad indivisible**.
- Se busca asegurar consistencia y durabilidad de los datos frente a concurrencia y fallos.
- Un ejemplo clásico son las transferencias dentro de un banco.
  - Un usuario a otro implica reducir plata de A y aumentar el dinero de B. Si solo una parte se completa, el sistema queda inconsistente.
  - Si 2 o más usuarios están depositando a B, B debe ver la suma total de transferencia y no que un "suma" se pierda por problemas de concurrencia.

Transacción - Comandos

- *Begin* → Empezar una transacción.
- *Read* → Leer un valor.
- *Write* → Escribir un valor (también se considera actualizar o eliminar).
- *Commit* → Consolidar los cambios permanentemente.
- *Abort* o *Rollback*→ Deshacer todos los cambios de la transacción.

Transacciones Distribuidas

- Es una transacción gestionados por **múltiples servidores** ubicados en diferentes computadoras.
- Se debe asegurar las propiedades de una transacción, pero ahora con los desafíos de un sistema distribuido.
  - Todos los servidores involucrados confirman la transacción o todos la cancelen.
  - Dejar todos los servidores en un estado consistente.

Transacciones Distribuidas

- Es una transacción gestionados por **múltiples servidores** ubicados en diferentes computadoras.
- Se debe asegurar las propiedades de una transacción, pero ahora con los desafíos de un sistema distribuido.
  - Todos los servidores involucrados confirman la transacción o todos la cancelen.
  - Dejar todos los servidores en un estado consistente.
- Un ejemplo clásico es la administración de una mega tienda (por ejemplo, Jumbo)
  - Se está transfiriendo dinero de un servidor a otro que no necesariamente son de la misma red distribuida.
  - Muchos usuarios están paralelamente haciendo compras o solicitando devoluciones.

Desafíos en una transacción distribuida

- Se debe asegurar las propiedades de una transacción, pero ahora con los desafíos de un sistema distribuido.
  - Manejo de Fallas parciales
  - Heterogeneidad de la red (diferentes tipos de componentes interactuando)
  - Latencia de Comunicación
  - Concurrencia
  - Interbloqueos Distribuidos

Desafíos en una transacción distribuida

- Se debe asegurar las propiedades de una transacción, pero ahora con los desafíos de un sistema distribuido.
  - Manejo de Fallas parciales
  - Heterogeneidad de la red (diferentes tipos de componentes interactuando)
  - Latencia de Comunicación
  - **Concurrencia**
  - **Interbloqueos Distribuidos**
- Vamos a explorar un poco más los últimos 2 desafíos: Concurrencia e Interbloqueos Distribuidos.

Desafíos en una transacción distribuida

Concurrencia

- La presencia de múltiples usuarios en un sistema distribuido genera solicitudes concurrentes a sus recursos. Cada recurso debe diseñarse para ser seguro en un entorno concurrente.
- Las transacciones distribuidas deben cumplir con la **serializabilidad de una copia (*one-copy serializability*)**.

Desafíos en una transacción distribuida

Concurrencia

- La presencia de múltiples usuarios en un sistema distribuido genera solicitudes concurrentes a sus recursos. Cada recurso debe diseñarse para ser seguro en un entorno concurrente.
- Las transacciones distribuidas deben cumplir con la **serializabilidad de una copia (*one-copy serializability*)**.
  - **Serializabilidad**: El resultado final del sistema es el mismo que si las transacciones se hubieran ejecutado una tras otra, en algún orden serial.
  - **...de una copia**: A pesar de que haya varias copias de los datos, el comportamiento de las transacciones debe ser el mismo que si solo existiera una única copia.

Desafíos en una transacción distribuida

Interbloqueos Distribuidos

- Para garantizar serializabilidad de una copia, muchas veces los sistemas establecen mecanismos de bloqueos (*locks*) para ordenar la ejecución de múltiples transacciones.
  - Es posible que diferentes servidores impongan órdenes diferentes en las transacciones, lo que lleva a **dependencias cíclicas** (los famosos *deadlocks*).
- Se suele utilizar el grafo global de espera (*wait-for graph*) para detectar estos *deadlocks*.

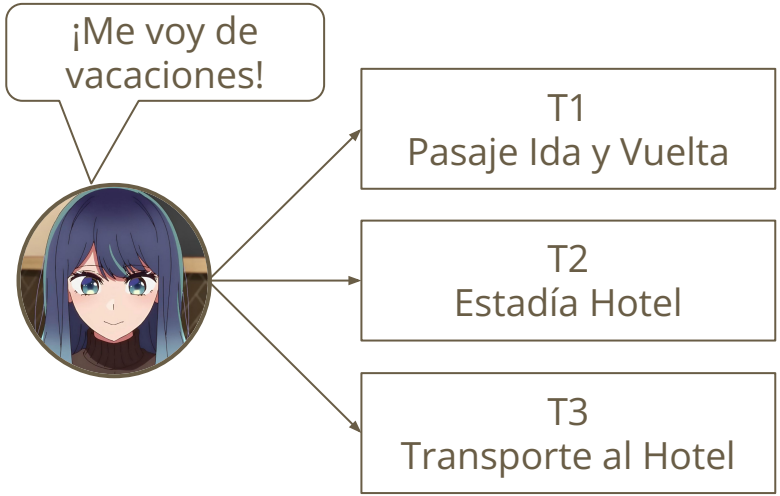
Desafíos en una transacción distribuida

Interbloqueos Distribuidos

- Para garantizar serializabilidad de una copia, muchas veces los sistemas establecen mecanismos de bloqueos (*locks*) para ordenar la ejecución de múltiples transacciones.
  - Es posible que diferentes servidores impongan órdenes diferentes en las transacciones, lo que lleva a **dependencias cíclicas** (los famosos *deadlocks*).
- Se suele utilizar el grafo global de espera (*wait-for graph*) para detectar estos *deadlocks*.

La próxima clase abordaremos con más detalle el desafío de controlar la concurrencia y sus implicancias.

Tipos de Transacción Distribuida



Plana

Tipos de Transacción Distribuida

T1
Pasaje Ida y Vuelta

T2
Estadía Hotel

T3
Transporte al Hotel

Plana

T0
Paquete de viaje

T1
Pasajes

T2
Hotel

T3
Pasaje Ida

T4
Pasaje Vuelta

T5
Puntos

T6
Estadía

T7
Transporte

Anidada

Tipos de Transacción Distribuida

Transacciones anidadas (otro ejemplo)

- Pagar el sueldo de una empresa
  - Nivel 0 (más alto): Pagar el sueldo del mes.
  - Nivel 1 (medio): Pagar a cada "departamento" (finanzas, recursos humanos, etc).
  - Nivel 2 (más bajo): Transferir a cada empleado del departamento.

Tipos de Transacción Distribuida

Transacciones anidadas (beneficios)

- Las sub-transacciones al mismo nivel pueden ejecutarse en paralelo, incluso en servidores diferentes.
- Las sub-transacciones pueden fallar independientemente, y la transacción padre (nivel superior a la sub-transacción) puede optar por una alternativa para completar su tarea, lo que mejora la recuperación y la flexibilidad.

Transacción Distribuida y Propiedades ACID

Las propiedades ACID son los pilares de la fiabilidad de las transacciones:

- A - *Atomicity* (Atomicidad)
- C - *Consistency* (Consistencia/Coherencia)
- I - *Isolation* (Aislamiento)
- D - *Durability* (Durabilidad)

Transacción Distribuida y Propiedades ACID

A - *Atomicity* (Atomicidad)

- La transacción es una unidad indivisible. Significa que todas las operaciones dentro de la transacción se completan con éxito, o ninguna de ellas lo hace.
- Si algo sale mal (un servidor falla), todos los efectos de la transacción son borrados por completo.
- Si ya se finalizó una transacción (*commit* o *abort*), esa unidad ya queda olvidada. No se puede hacer ninguna operación posterior.
- Dentro de un sistema distribuido, también implica que todos los nodos participantes la aceptan o ninguna.

Transacción Distribuida y Propiedades ACID

C - *Consistency* (Consistencia/Coherencia)

- Una transacción exitosa lleva al sistema de un estado consistente a otro estado consistente.
- La transacción no debe violar ninguna de las reglas o invariantes predefinidas del sistema.
  - Por ejemplo, no aceptar un pago de tarjeta de crédito si este va a superar el cupo asignado al cliente.
  - No sacar dinero de una cuenta que está bloqueada.
- Muchas veces la consistencia es responsabilidad de los programadores de servidores.

Transacción Distribuida y Propiedades ACID

I - *Isolation* (Aislamiento)

- Cada transacción debe ejecutarse sin interferencia de otras transacciones concurrentes.
- Los efectos intermedios de una transacción no deben ser visibles para otras transacciones.
- Evitar que ocurran Lecturas sucias (*Dirty Reads*) y Escrituras prematuras (*Premature Writes*)

Transacción Distribuida y Propiedades ACID

I - *Isolation* (Aislamiento)

- Cada transacción debe ejecutarse sin interferencia de otras transacciones concurrentes.
- Los efectos intermedios de una transacción no deben ser visibles para otras transacciones.
- Solo finalizada la transacción (*commit*), los efectos de la transacción son visibles.
- El aislamiento evita fenómenos como lecturas sucias (*Dirty Reads*) y, junto con los mecanismos de control de escritura, previene escrituras prematuras (*Premature Writes*).

Transacción Distribuida y Propiedades ACID

I - *Isolation* (Aislamiento)

- ***Dirty Reads***: leer una variable que otra transacción escribió, pero que esta todavía no "Commitea".
  - Si la transacción que realizó la escritura se revierte (*abort*), la transacción que realizó la lectura ha accedido a datos "inexistentes" o inválidos.

Transacción Distribuida y Propiedades ACID

I - *Isolation* (Aislamiento)

- ***Dirty Reads***: leer una variable que otra transacción escribió, pero que esta todavía no "Commitea".
  - Si la transacción que realizó la escritura se revierte (*abort*), la transacción que realizó la lectura ha accedido a datos "inexistentes" o inválidos.
- ***Premature Writes***: escribir en la base sin garantizar que se hizo "commit".
  - Esto puede provocar inconsistencias o complicar la recuperación si la transacción falla.
    - Imagina que reduces la plata de una cuenta y no alcanzas a hacer *commit* antes que el sistema falla. La transacción no ocurrió en verdad, pero ya se modificó la base de datos.
  - Incluso dar lugar a un *Dirty Read* si otra transacción llega a leer ese valor no confirmado.

Transacción Distribuida y Propiedades ACID

D - *Durability* (Durabilidad)

- Una vez que una transacción ha completado con éxito (*commit*), todos sus efectos se guardan en almacenamiento permanente.
- Estos cambios deben sobrevivir a cualquier falla del servidor.
- En el caso de Transacciones Anidadas, aunque una sub-transacción hizo *commit*, solo cuando el nivel más superior hace *commit* se garantiza que toda la operación va a perdurar.

Transacción Distribuida y Propiedades ACID

¿Como los comandos pueden fallar las propiedades?

Veamos algunos **casos** donde los comandos *write*, *read*, *commit* o *abort* pueden estar fallando las propiedades ACID.

Transacción Distribuida y Propiedades ACID

¿Como los comandos pueden fallar las propiedades?

Write

- Falla atomicidad si se hace después *abort* o *commit*.
- Falla consistencia si se está modificando a un valor que no corresponde al sistema.
- Falla aislamiento si modifica la base de datos visible por todos antes de hacer *commit* (escritura prematura)

Read

- Falla atomicidad si se hace después *abort* o *commit*.
- Falla consistencia si se está leyendo una variable que no existe.
- Falla aislamiento, si se lee una variable que existe por otra transacción pero no está consolidada (lectura sucia).

Transacción Distribuida y Propiedades ACID

¿Como los comandos pueden fallar las propiedades?

Commit

- Falla atomicidad, si no logra completar exitosamente la operación. Por ejemplo, no asegurar que todos los nodos involucrados guarden la transacción.
- Falla consistencia si alguna operación dentro de la transacción ya falló en alguna propiedad y el sistema intenta de todos modos consolidarla.

Abort

- Falla durabilidad si intenta abortar una transacción ya consolidada.

Coordinando transacciones distribuidas

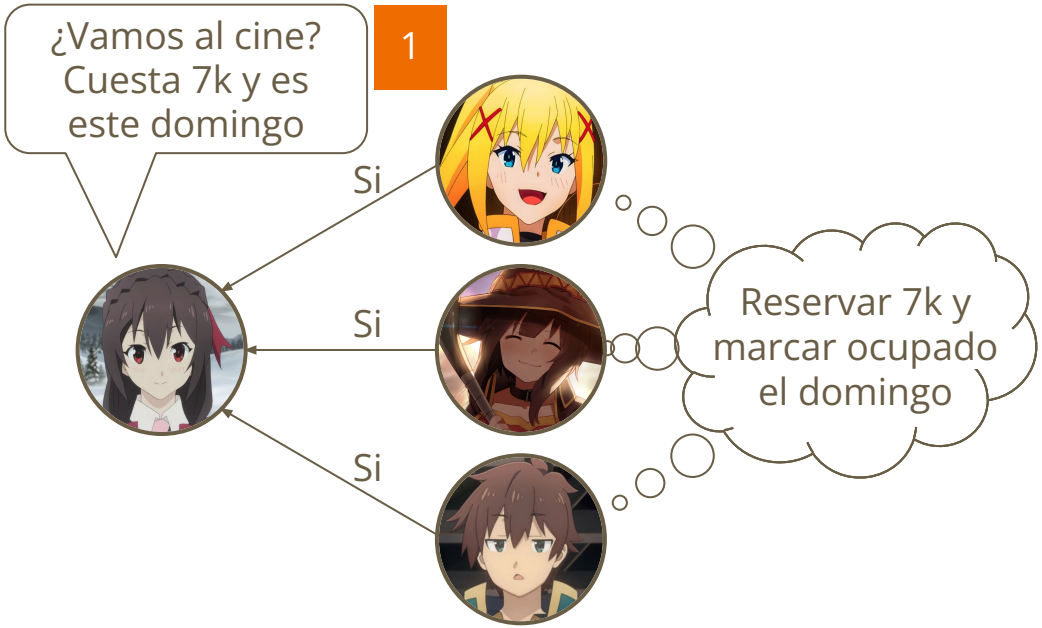
Commit de 2 fases

Commit de 3 fases

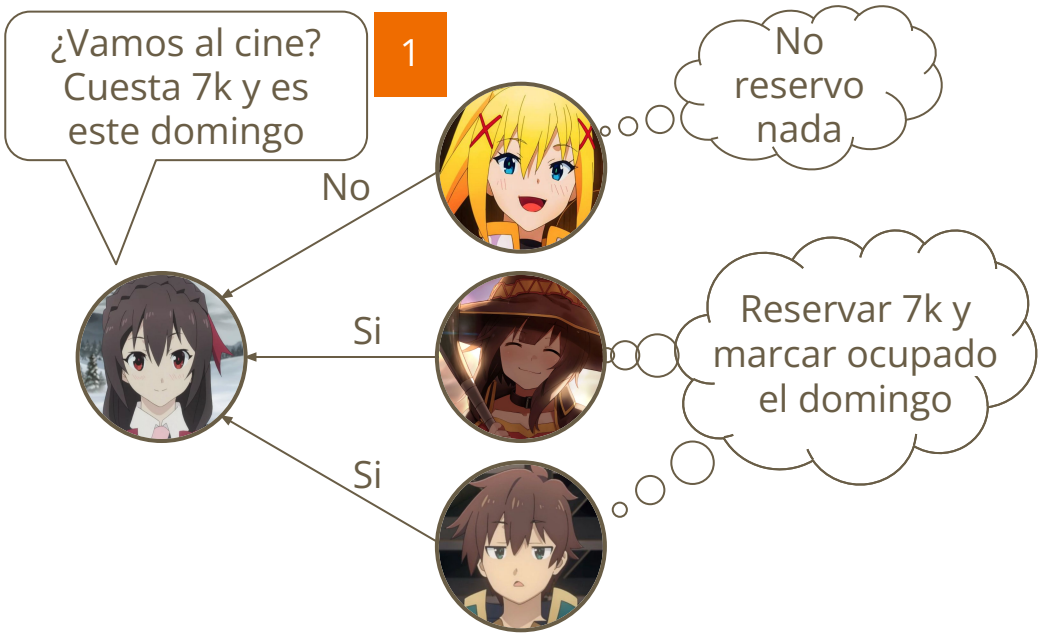
Coordinando transacciones distribuidas

- Dado los desafíos de la red distribuida, se proponen ciertos protocolos para garantizar consistencia.
- Ya estudiamos uno que es el protocolo de replicación activa. Las transacciones pasan solo por 1 réplica y es esta quien la distribuye a los demás sistemas.
- Existen 2 protocolos más que no requieren de una unidad central.
  - *Commit de 2 fases.*
  - *Commit de 3 fases.*

Commit de 2 fases (2PC)



Commit de 2 fases (2PC)



Commit de 2 fases (2PC)

Commit de 2 fases (2PC)

- Es un protocolo donde los servidores se comunican entre sí para tomar una decisión conjunta sobre si confirmar o cancelar una transacción.
- Permite que cualquier participante aborte la transacción.
- Se definen 2 roles: Coordinador y Participantes.
  - **Coordinador**: Nodo que inicia y gestiona el protocolo de commit.
  - **Participantes**: Nodos involucrados en la transacción, cada uno verifica que la transacción sea válida dentro de su nodo, por ejemplo, que los recursos de su réplica estén disponibles para usar o que la transacción no genere una inconsistencia.
- Se compone de 2 fases: el *prepare* y el *commit*.

Commit de 2 fases (2PC)

- Se compone de 2 fases: el ***prepare*** y el *commit*.
- Primera fase también conocida como votación. Los pasos son:

1. El coordinador envía un mensaje *`canCommit?`* a todos los participantes en la transacción.
2. Cuando un participante recibe la solicitud, responde con su voto (Sí o No) al coordinador.
3. Antes de responder, valida que la transacción no genere ningún conflicto u ocupen recursos bloqueados.
4. En caso de responder si, bloquea todos los recursos involucrados en la transacción para que ninguna otra transacción pueda hacer *canCommit?* con los mismos recursosmientras espera la fase 2.
    a. Este bloqueo puede ser parcial según el tipo de operación
5. En caso de responder no, el participante aborta la transacción en su réplica y no bloquea ningún recurso. La transacción no existe para dicho nodo.

Commit de 2 fases (2PC)

- Se compone de 2 fases: el *prepare* y el ***commit***.
- Segunda fase también conocida como de consolidación. Los pasos son:

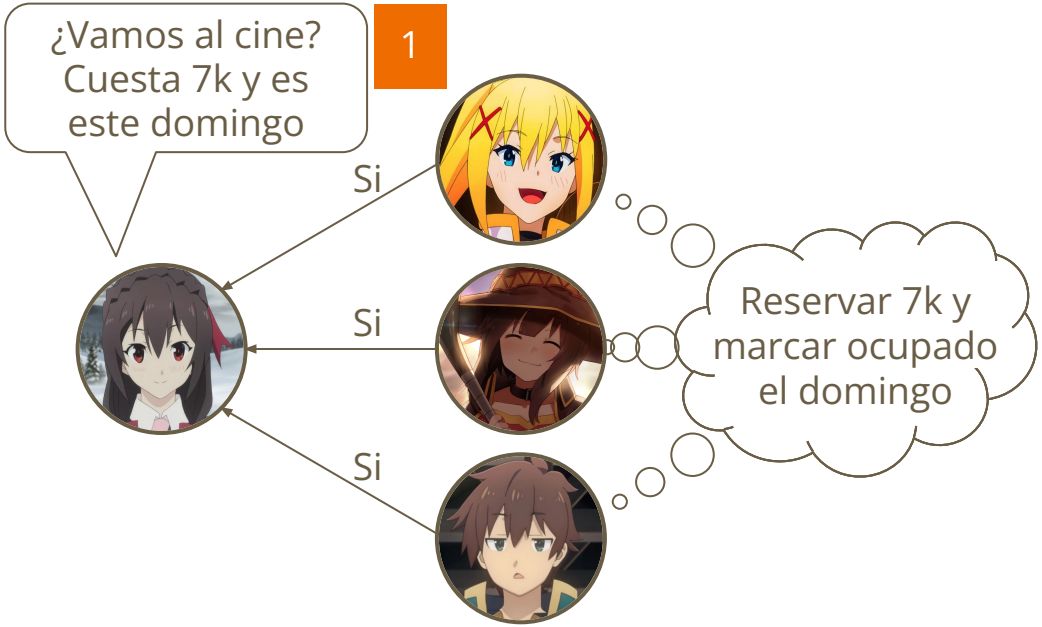
1. El coordinador recopila todos los votos (incluyendo el suyo si también es participante).
2. Si todos los votos son "Sí", el coordinador decide confirmar la transacción y envía un mensaje *`doCommit`* a todos los participantes.
3. Si algún voto es "No" (o si un participante falla), el coordinador decide abortar la transacción y envía un mensaje *`doAbort`* a todos los participantes.
4. Los participantes, al recibir *`doCommit`* o *`doAbort`*, aplican la decisión en su réplica y liberan los recursos bloqueados.

Commit de 2 fases (2PC)

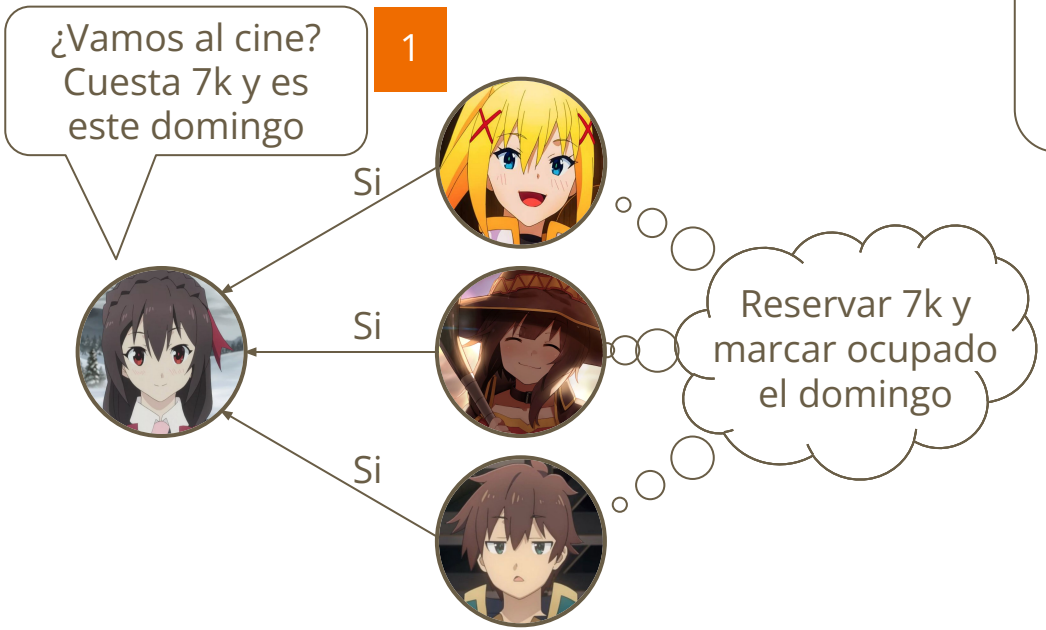
⚠️ **Problema** ⚠️ **¿Qué pasa si coordinador falla después de la fase 1?**

- Un participante que ha votado "Sí" pero aún no ha recibido la decisión final del coordinador se encuentra en un **estado incierto**.
- No puede decidir unilateralmente y no puede liberar los objetos para otras transacciones porque puede ser solo un retraso en la comunicación.
- Todo esto provoca retrasos extensos en las demás transacciones.

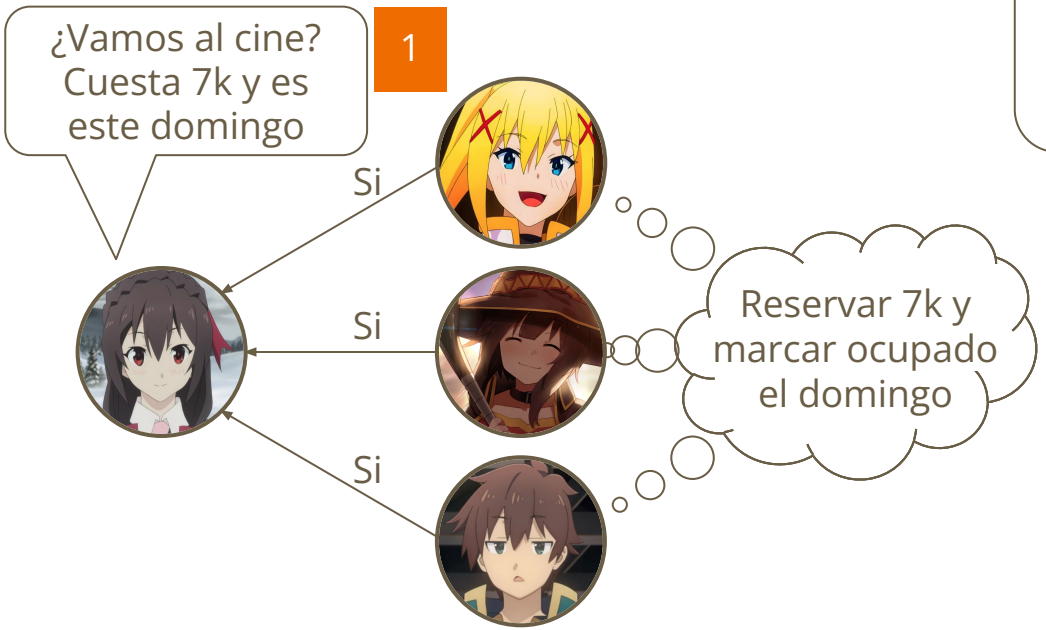
Commit de 3 fases (3PC)



Commit de 3 fases (3PC)



Commit de 3 fases (3PC)



Commit de 3 fases (3PC)

- Es un protocolo diseñado para mitigar los retrasos asociados con el estado "incierto" en 2PC (*2-phase-commit*), al evitar que los participantes se queden bloqueados indefinidamente si el coordinador falla.
- Con la adición de una fase de confirmación y la capacidad que los participantes se comuniquen entre ellos. Antes de ir a estado incierto, se coordinan entre ellos para tomar una decisión.
- Mantiene los 2 roles del protocolo anterior: Coordinador y participantes.
- Se compone de 3 fases: el *vote request*, *pre-commit* y el *do-commit*.

Commit de 3 fases (3PC)

- Se compone de 3 fases: el ***vote request***, *pre-commit* y el *do-commit*.
- El ***vote requests*** es Idéntica a la fase de preparación del protocolo de 2 fases.
  - El coordinador envía `canCommit?`. Los participantes votan y reservan los recursos en caso de votar "Sí".

Commit de 3 fases (3PC)

- Se compone de 3 fases: el *vote request*, ***pre-commit*** y el *do-commit*.
- En el ***pre-commit***, el coordinador envía una solicitud `preCommit` a todos los participantes si es que todos los votos fueron "Sí". Los participantes que votaron "Sí" esperan esta solicitud, la acusan de recibo, pero no consolidan nada.
  - Si el coordinador no manda el *pre-commit*, entre los participantes que votaron que "Sí" verifican la situación.
  - Si al menos 1 de los participantes tiene *pre-commit*, se entienden que todos votaron que "Sí". Por lo que se aseguran que todos sepan la existencia de ese *pre-commit* y aplican consolidación.
  - Si nadie tiene *pre-commit* se asume que algo salió mal y abortan la transacción.

Commit de 3 fases (3PC)

- Se compone de 3 fases: el *vote request*, *pre-commit* y el ***do-commit***.
- En el ***do-commit***, si el coordinador recibe todos los acuses de recibo de *preCommit*, envía una solicitud `doCommit` a los participantes. Los participantes, al recibirla, consolidan la transacción (aplican *commit*).
  - Si el coordinador no manda el *doCommit*, igual a la fase anterior, los participantes conversan entre ellos. Dado que al menos 1 tendrá *pre-commit*, entonces llegan al consenso de consolidar la transacción.

Commit de 3 fases (3PC)

Comparación con 2PC

- En 3PC, luego de consultar, hay una fase de "avisar" pero sin comprometer. Permite que los participantes tengan instancia para ponerse de acuerdo y decidir si abortan o prosiguen la transacción.
- En 2PC, una vez que el coordinador obtiene todos los "Sí", va directo a la consolidación. Por lo que no da instancia a los participantes de conversar y decidir si efectivamente se consolida o mejor abortar.
- La principal desventaja de 3PC es el costo en términos del número de mensajes y el número de rondas requeridas en el caso normal (sin fallos) en comparación con el 2PC.

Poniendo a prueba lo que hemos aprendido 👀

¿Cuáles de las siguientes afirmaciones son **correctas**?

I. El protocolo 2PC puede llevar a una violación del aislamiento si múltiples participantes quedan bloqueados esperando una decisión.

II. El protocolo 3PC reduce el riesgo de bloqueo indefinido pero incrementa la sobrecarga de mensajes en caso de fallas.

III. En una transacción anidada, una sub-transacción no viola la durabilidad si consolida sus cambios sólo cuando las demás sub-tracciones de su mismo nivel ya aceptaron la transacción.

IV. El protocolo 3PC garantiza la atomicidad en las transacciones.

A) Solo II

B) I y III

C) II y IV

D) II, III y IV

E) I, II y III

Poniendo a prueba lo que hemos aprendido 👀

¿Cuáles de las siguientes afirmaciones son **correctas**?

I. El protocolo 2PC puede llevar a una violación del aislamiento si múltiples participantes quedan bloqueados esperando una decisión.

II. El protocolo 3PC reduce el riesgo de bloqueo indefinido pero incrementa la sobrecarga de mensajes en caso de fallas.

III. En una transacción anidada, una sub-transacción no viola la durabilidad si consolida sus cambios sólo cuando las demás sub-tracciones de su mismo nivel ya aceptaron la transacción.

IV. El protocolo 3PC garantiza la atomicidad en las transacciones.

A) Solo II

C) II y IV

E) I, II y III

B) I y III

D) II, III y IV

Próximos eventos

Próxima clase

- Control de Concurrencia
- ¿Qué hacemos cuando hay concurrencia de escrituras sobre datos? ¿será la exclusión mutua la única opción?

Evaluación

- Hoy finaliza el plazo oficial de la T2. Luego tienen 2 días máximo para entregas atrasadas.
- *Spoiler T3*: vamos a simular transacciones distribuidas para ver cómo queda la base de datos al final.

IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 14)

Créditos (animes utilizados)

[Oshi no Ko]

Kono Subarashii Sekai ni Shukufuku wo! (Konosuba)