

# **Consistencia y Transacciones Distribuidas**

**30221 - Sistemas Distribuidos**

**Rafael Tolosana**

**Dpto. Informática e Ing. de Sistemas**

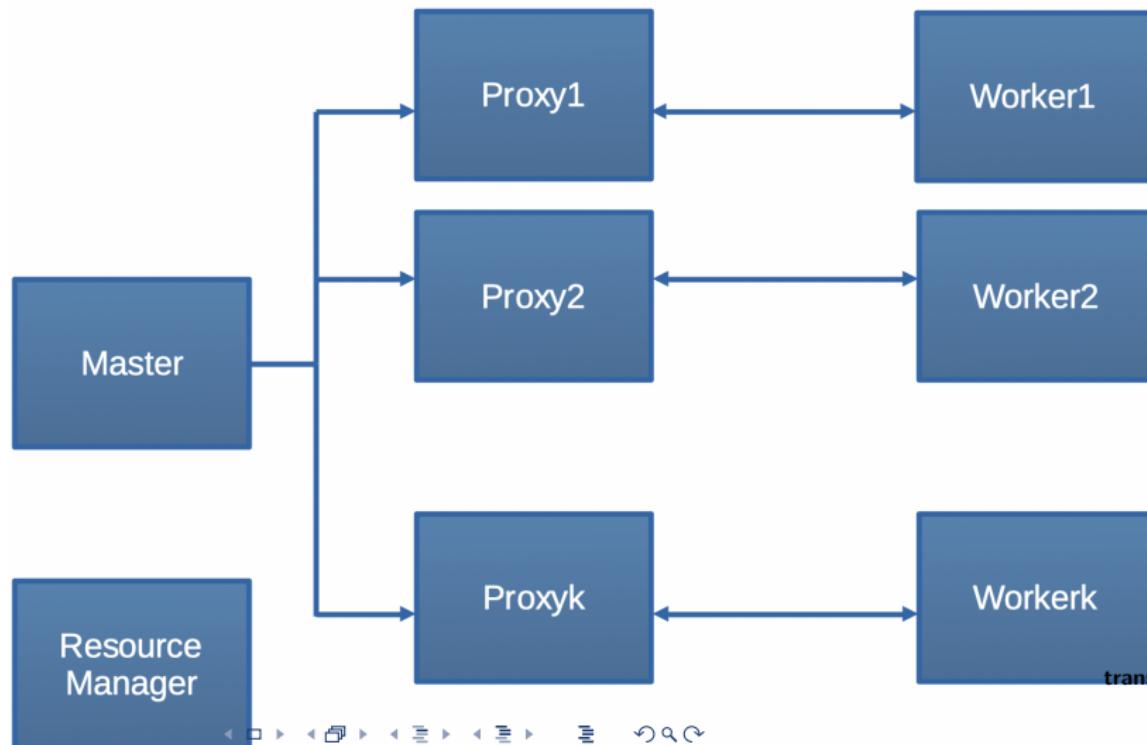
## Lectura Recomendada

- Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5, 2012. Capítulos 16 y 17
- Tanenbaum and Van Steen, Distributed Systems: Principles and Paradigms, 3e, (c) 2017 Prentice- Hall. Chapters 7, 8.5.

# Motivación

# Motivación

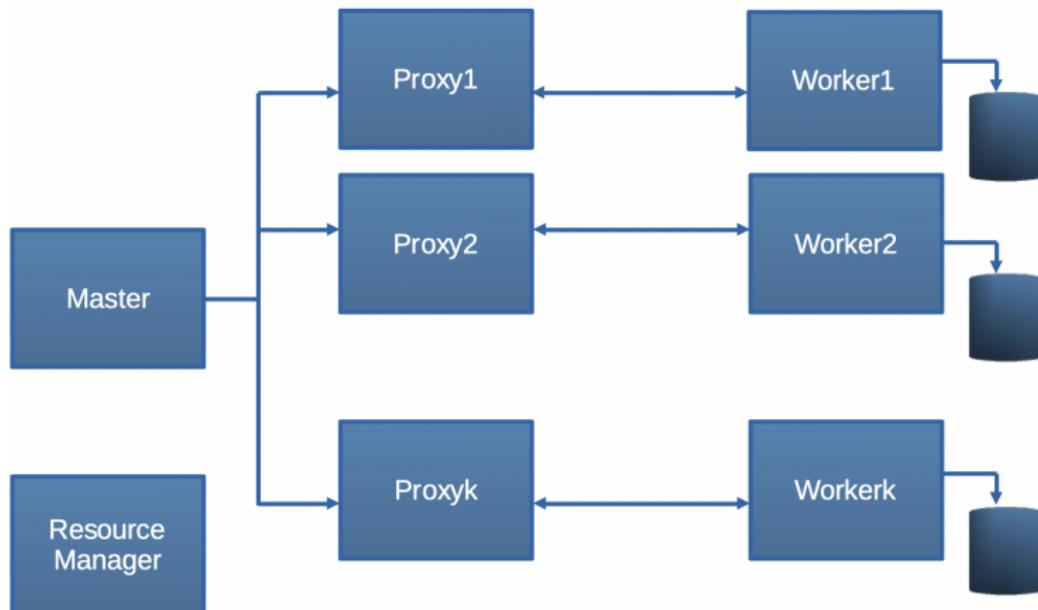
## Escalabilidad arquitectura máster-worker



# Motivación

## Replicación disco-datos

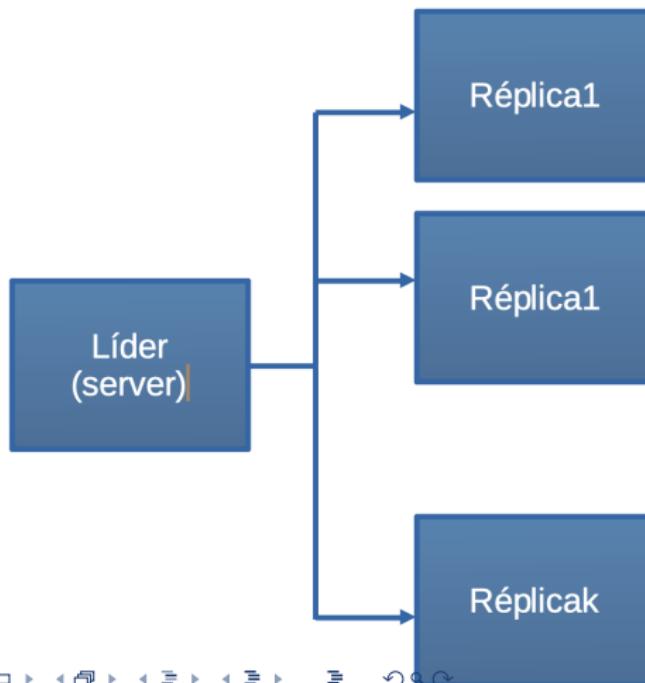
Los workers dan servicio a los clientes



# Motivación

**Raft: tolerancia fallos CON estado**

Las réplicas no dan servicio a los clientes



# Motivación

## Replicación

- Se usa como una estrategia para mejorar la tolerancia a fallos
- Se usa también para mejorar las prestaciones (escalar)

# Motivación

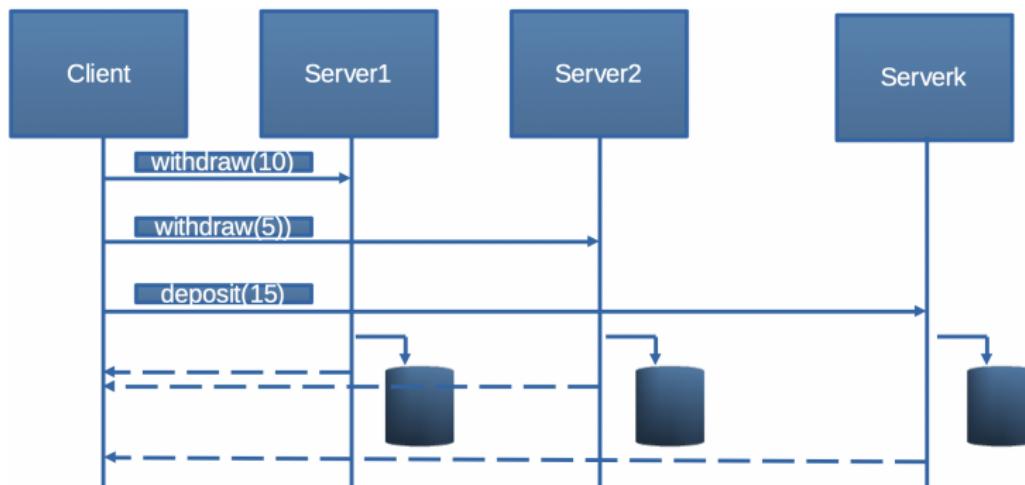
## Replicación

- Se usa como una estrategia para mejorar la tolerancia a fallos
- Se usa también para mejorar las prestaciones (escalar)
- En ambos casos hay que garantizar la consistencia
  - **Copias** de los datos, que sean vistas como “**idénticas**”

# Motivación

**Cliente interactúa con múltiples servidores**

Cambio estado en múltiples servidores independientes

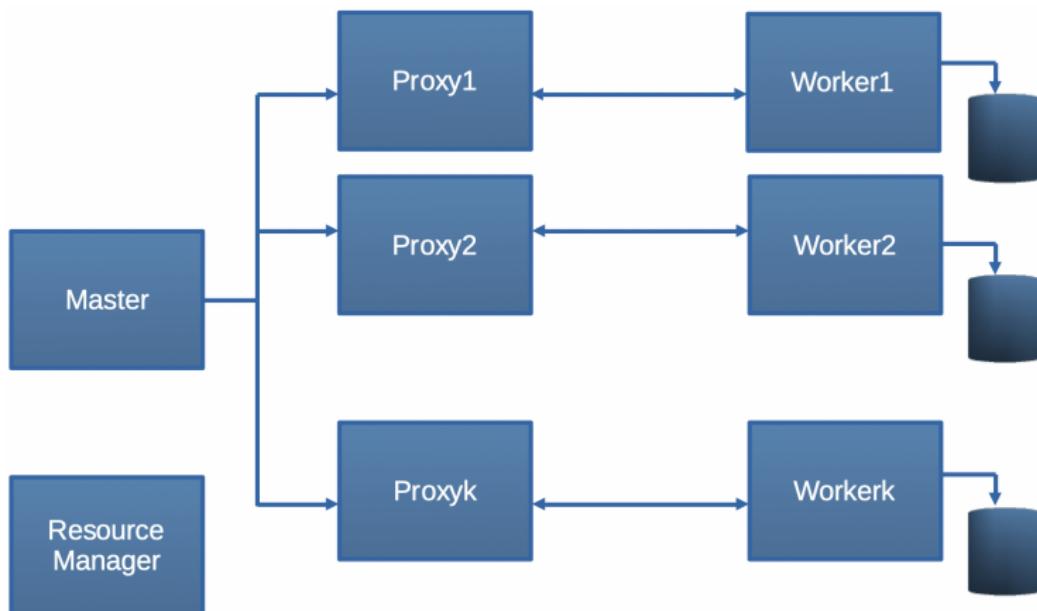


# Consistencia

# Consistencia

Replicación como técnica de escalabilidad

El problema de las escrituras



# Consistencia

## El dilema de la replicación como técnica de escalabilidad

- La replicación puede mejorar las prestaciones
- Sin embargo mantener todas las réplicas consistentes puede suponer un enorme coste en prestaciones
  - Sincronización global

# Consistencia

## El dilema de la replicación como técnica de escalabilidad

- La replicación puede mejorar las prestaciones
- Sin embargo mantener todas las réplicas consistentes puede suponer un enorme coste en prestaciones
  - Sincronización global
- Solución: relajar la consistencia
  - Diferentes modelos
  - En función del modelo de consistencia tendremos unas réplicas consistentes con diferentes características

# Consistencia secuencial

Lamport, 1979

- Cualquier entrelazado read / write es válido, pero el entrelazado será el mismo para todos los procesos

P1: W(x)a

---

P2: W(x)b

---

P3: R(x)b R(x)a

---

P4: R(x)b R(x)a

P1: W(x)a

---

P2: W(x)b

---

P3: R(x)b R(x)a

---

P4: R(x)a R(x)b

<sup>1</sup>From M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.

# Consistencia causal

- Dados dos eventos de escritura  $e_1$  y  $e_2$ , si  $e_2$  está relacionado causalmente con  $e_1$  (*happens-before*), entonces todos los procesos ven el efecto de  $e_1$  y luego el de  $e_2$ .
- Escrituras concurrentes pueden verse en cualquier orden por los distintos nodos

P1:	W(x)a		W(x)c		
P2:	R(x)a	W(x)b			
P3:	R(x)a		R(x)c	R(x)b	
P4:	R(x)a		R(x)b	R(x)c	2

<sup>2</sup>From M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.

# Consistencia eventual

- Todas las escrituras se propagarán con el tiempo. Si no hay escrituras durante un largo periodo de tiempo, todas las réplicas llegarán, gradualmente a un estado consistente.

# Consistencia eventual

- Todas las escrituras se propagarán con el tiempo. Si no hay escrituras durante un largo periodo de tiempo, todas las réplicas llegarán, gradualmente a un estado consistente.
- Ejemplo DNS

# Consistencia eventual

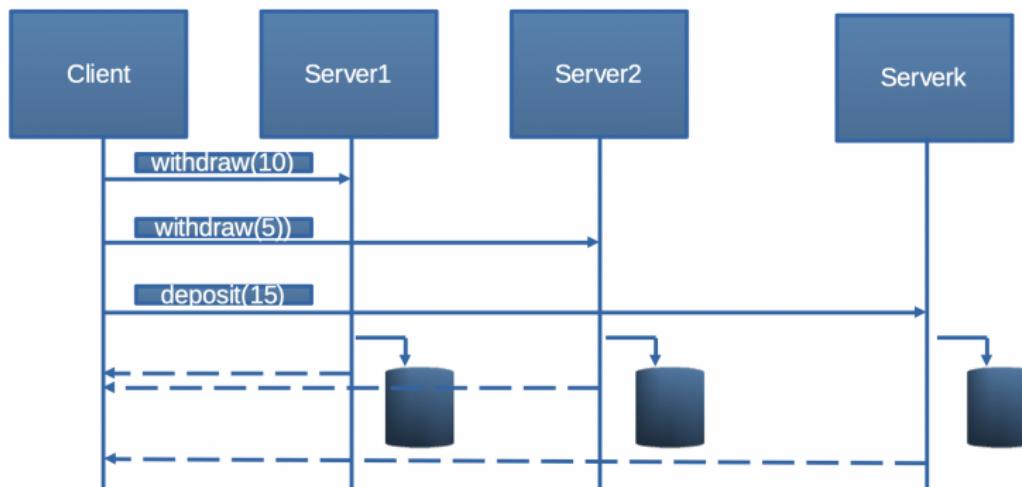
- Todas las escrituras se propagarán con el tiempo. Si no hay escrituras durante un largo periodo de tiempo, todas las réplicas llegarán, gradualmente a un estado consistente.
- Ejemplo DNS
- Si no hay muchas escrituras, con el tiempo todas las réplicas serán iguales

# Transacciones Distribuidas

# Transacciones Distribuidas

**Cliente interactúa con múltiples servidores**

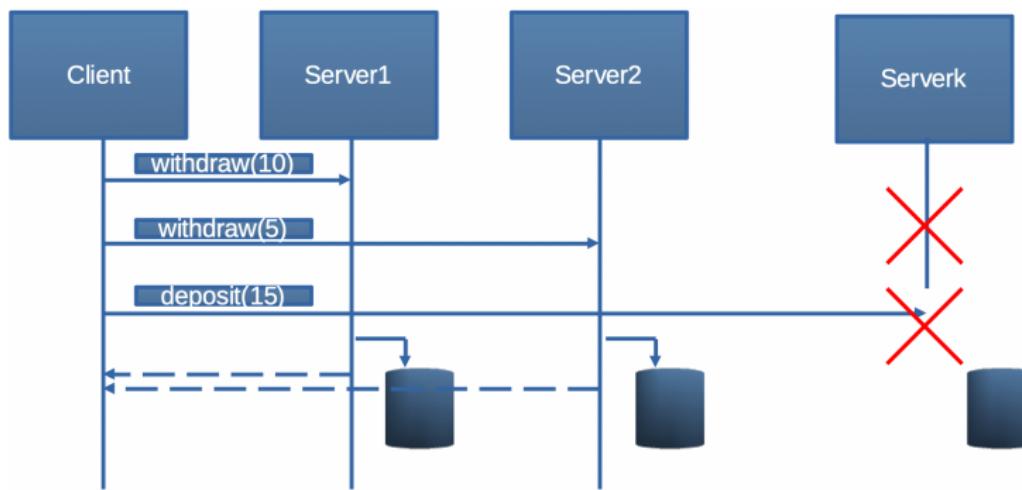
A veces se cambia el estado



# Transacciones Distribuidas

**Cliente interactúa con múltiples servidores**

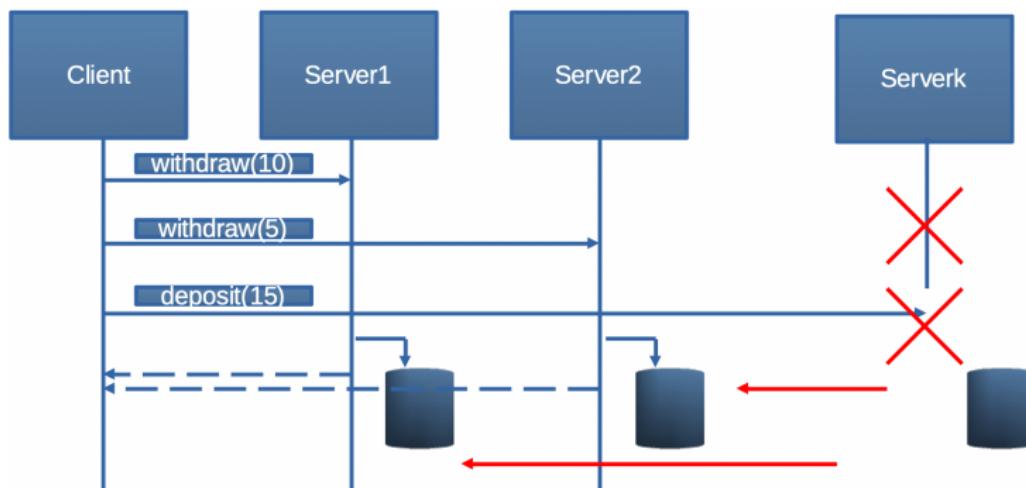
A veces se cambia el estado



# Transacciones Distribuidas

**Cliente interactúa con múltiples servidores**

A veces se cambia el estado



# Transacciones Distribuidas

## Definición: Transacción

- Una *transacción distribuida* define una secuencia de operaciones en un conjunto de servidores. Se garantiza que la secuencia es atómica, o se ejecuta completamente o no se ejecuta. Además, pueden coexistir múltiples clientes y múltiples servidores y todos ellos pueden fallar.

# Transacciones Distribuidas

## Definición: Transacción

- Una *transacción distribuida* define una secuencia de operaciones en un conjunto de servidores. Se garantiza que la secuencia es atómica, o se ejecuta completamente o no se ejecuta. Además, pueden coexistir múltiples clientes y múltiples servidores y todos ellos pueden fallar.
- Una transacción o bien termina comprometida (*committed*, sus efectos son visibles) o se aborta (*aborted*, sus efectos no son visibles)
- Las *transacciones anidadas* se estructuran a partir de conjuntos de otras transacciones. Son especialmente útiles en los sistemas distribuidos porque permiten una concurrencia adicional.

# Transacciones Distribuidas

## Transacciones

- Los clientes requieren que una secuencia de peticiones separadas a un servidor sean atómicas:
  - Estén libres de interferencias por operaciones que se estén realizando por otros clientes concurrentes

# Transacciones Distribuidas

## Transacciones

- Los clientes requieren que una secuencia de peticiones separadas a un servidor sean atómicas:
  - Estén libres de interferencias por operaciones que se estén realizando por otros clientes concurrentes
  - O bien *todas las operaciones deben completarse con éxito*, en cuyo caso todos los servidores guardan persistentemente el efecto de la operación
  - o todas abortan y no deben tener ningún efecto (por ejemplo en caso de caída del servidor), de manera que todos los servidores tienen que deshacer los efectos de las operaciones.
  - **Parece similar al consenso... servidores: all commit or all abort - Atomic commit**

# Transacciones Distribuidas

## Consenso vs Transacción (atomic commit)

### Diferencia 1

- Consenso: 1 o más nodos proponen un valor
- Atomic commit: todos los nodos votan commit or abort

# Transacciones Distribuidas

## Consenso vs Transacción (atomic commit)

### Diferencia 1

- Consenso: 1 o más nodos proponen un valor
- Atomic commit: todos los nodos votan commit or abort

### Diferencia 2

- Consenso: se decide sobre alguno de los valores propuestos
- Atomic commit: commit si todos votan commit, abort si al menos uno vota abort

### Diferencia 3

- Consenso: se toleran fallos mientras haya quorum suficiente
- Atomic commit: se aborta si algún nodo falla

# Transacciones en 1 servidor

## Propiedades ACID

- Atómica
  - Una transacción o se ejecuta completamente o no se ejecuta

# Transacciones en 1 servidor

## Propiedades ACID

- Atómica
  - Una transacción o se ejecuta completamente o no se ejecuta
- Consistente
  - Una transacción lleva al sistema de un estado consistente a otro estado consistente
  - Típicamente es responsabilidad del programador de los servidores y de los clientes dejar los contenidos de las bases de datos consistentes (dependientes de la aplicación, no del protocolo de transacción)
  - El término consistente tiene múltiples connotaciones dependientes del contexto.
    - Rélicas consistentes indican que son “iguales”

# Transacciones en 1 servidor

## Propiedades ACID

- Isolation
  - Al ejecutar una transacción concurrentemente con otras transacciones no hay condiciones de carrera

# Transacciones en 1 servidor

## Propiedades ACID

- Isolation
  - Al ejecutar una transacción concurrentemente con otras transacciones no hay condiciones de carrera
- Durabilidad
  - Despues de que una transacción se haya completado con éxito, todos sus efectos se guardan en el almacenamiento permanente.

# Transacciones en 1 servidor

## Control de la Concurrencia en Transacciones

- En general conocéis los problemas derivados de las condiciones de carrera

# Transacciones en 1 servidor

## Control de la Concurrencia en Transacciones

- En general conocéis los problemas derivados de las condiciones de carrera
- En el contexto de las transacciones hay dos problemas conocidos:
  - La actualización perdida
  - Recuperaciones inconsistentes

# Transacciones en 1 servidor

## Ejemplo Banca: operaciones

- cuenta.withdraw(n)
  - retira n unidades monetarias de la cuenta

# Transacciones en 1 servidor

## Ejemplo Banca: operaciones

- cuenta.withdraw(n)
  - retira n unidades monetarias de la cuenta
- cuenta.deposit(n):
  - Introduce n u.m. en la cuenta
- cuenta.getBalance()
  - Devuelve el saldo de la cuenta

# Transacciones en 1 servidor

## La actualización perdida

saldos iniciales de a y b: USD200

The lost update problem

Transaction T:	Transaction U:
$balance = b.getBalance();$	$balance = b.getBalance();$
$b.setBalance(balance * 1.1);$	$b.setBalance(balance * 1.1);$
$a.withdraw(balance / 10)$	$c.withdraw(balance / 10)$
$balance = b.getBalance(); \quad \$200$	$balance = b.getBalance(); \quad \$200$
$b.setBalance(balance * 1.1); \quad \$220$	$b.setBalance(balance * 1.1); \quad \$220$
$a.withdraw(balance / 10) \quad \$80$	$c.withdraw(balance / 10) \quad \$280$
	3

<sup>3</sup>Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design  
Edn. 5 © Pearson Education 2012

# Transacciones en 1 servidor

## Recuperaciones inconsistentes

saldos iniciales de a y b: USD200

The inconsistent retrievals problem

Transaction V:		Transaction W:
<i>a.withdraw(100)</i>		<i>aBranch.branchTotal()</i>
<i>b.deposit(100)</i>		
<i>a.withdraw(100);</i>	\$100	
		<i>total = a.getBalance()</i> \$100
<i>b.deposit(100)</i>	\$300	<i>total = total + b.getBalance()</i> \$300
		<i>total = total + c.getBalance()</i>
	•	•
		4

<sup>4</sup>Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012

# Transacciones en 1 servidor

## Equivalencia serial

### Definición:

- Se dice que un entrelazado de dos transacciones V y W es un *equivalente serial*, si el resultado de ejecutar V y W con ese entrelazado es el mismo que ejecutarlas de forma aislada o secuencialmente.

# Transacciones en 1 servidor

## Equivalencia serial

### Definición:

- Se dice que un entrelazado de dos transacciones V y W es un *equivalente serial*, si el resultado de ejecutar V y W con ese entrelazado es el mismo que ejecutarlas de forma aislada o secuencialmente.
- ¿Para qué buscan estos entrelazados?

# Transacciones en 1 servidor

## Equivalencia serial

### Definición:

- Se dice que un entrelazado de dos transacciones V y W es un *equivalente serial*, si el resultado de ejecutar V y W con ese entrelazado es el mismo que ejecutarlas de forma aislada o secuencialmente.
- ¿Para qué buscan estos entrelazados?
- ¿Qué relación hay entre eso y los semáforos / monitores?

# Transacciones en 1 servidor

## Operaciones Conflictivas

- Se dice que dos operaciones op1 y op2 **son conflictivas** si el resultado final de su ejecución depende del orden en el que se ejecutaron.

# Transacciones en 1 servidor

## Operaciones Conflictivas

- Se dice que dos operaciones op1 y op2 **son conflictivas** si el resultado final de su ejecución depende del orden en el que se ejecutaron.
- Por ejemplo: read y write sobre la misma variable:  
read(a); write(a) vs write(a); read(a)

# Transacciones en 1 servidor

## Operaciones Conflictivas

- Se dice que dos operaciones op1 y op2 **son conflictivas** si el resultado final de su ejecución depende del orden en el que se ejecutaron.
- Por ejemplo: read y write sobre la misma variable:  
read(a); write(a) vs write(a); read(a)
- Para cualquier par de transacciones, es posible determinar el orden de los pares de operaciones conflictivas sobre los objetos a los que ambos acceden.
- Para que dos transacciones sean equivalentes en serie, es necesario y suficiente que todos los pares de operaciones conflictivas de las dos transacciones se ejecuten en el mismo orden.

# Transacciones en 1 servidor

Existen también locks (mutex)

- Por tanto, pueden aparecer bloqueos
- Hay técnicas de detección de bloqueos
- Hay técnicas de prevención de bloqueos

Una alternativa al locking es el Control Optimista

- Se asume que probablemente no habrá operaciones en conflicto

# Transacciones en 1 servidor

Existen también locks (mutex)

- Por tanto, pueden aparecer bloqueos
- Hay técnicas de detección de bloqueos
- Hay técnicas de prevención de bloqueos

Una alternativa al locking es el Control Optimista

- Se asume que probablemente no habrá operaciones en conflicto
- El control optimista de la concurrencia permite que las transacciones sigan adelante hasta que estén listas para comprometerse (*commit*), tras lo cual se realiza una comprobación para ver si han realizado operaciones conflictivas en los objetos.

# Two-phase Commit

**Two-phase commit, Jim Gray 1978**

- Es una solución clásica al problema atomic commit

# Two-phase Commit

## Two-phase commit, Jim Gray 1978

- Es una solución clásica al problema atomic commit
- Si un servidor falla, todos abortan y vuelven al estado anterior

# Two-phase Commit

## Two-phase commit, Jim Gray 1978

- Es una solución clásica al problema atomic commit
- Si un servidor falla, todos abortan y vuelven al estado anterior
- Si todos los servidores terminan correctamente, todos comprometen el resultado.

# Two-phase Commit

## Two-phase commit, Jim Gray 1978

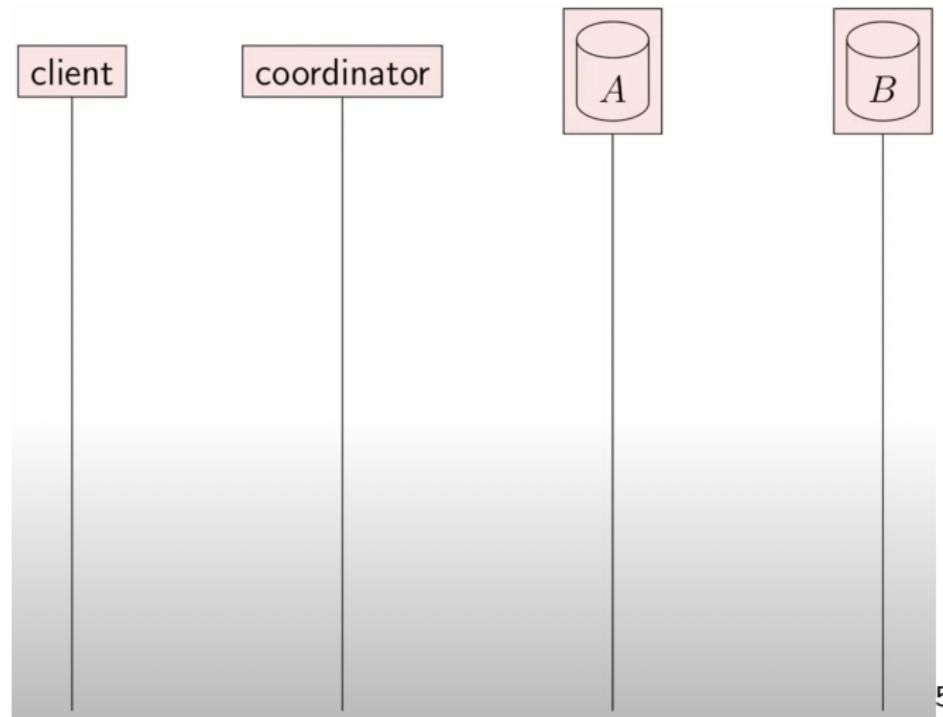
- Es una solución clásica al problema atomic commit
- Si un servidor falla, todos abortan y vuelven al estado anterior
- Si todos los servidores terminan correctamente, todos comprometen el resultado.
- Es el protocolo de interacción que habitualmente se utiliza para implementar transacciones distribuidas

# Two-phase Commit

## Two-phase commit, Jim Gray 1978

- Es una solución clásica al problema atomic commit
- Si un servidor falla, todos abortan y vuelven al estado anterior
- Si todos los servidores terminan correctamente, todos comprometen el resultado.
- Es el protocolo de interacción que habitualmente se utiliza para implementar transacciones distribuidas
- Actores:
  - el proceso cliente
  - los procesos servidores
  - un proceso coordinador de la transacción

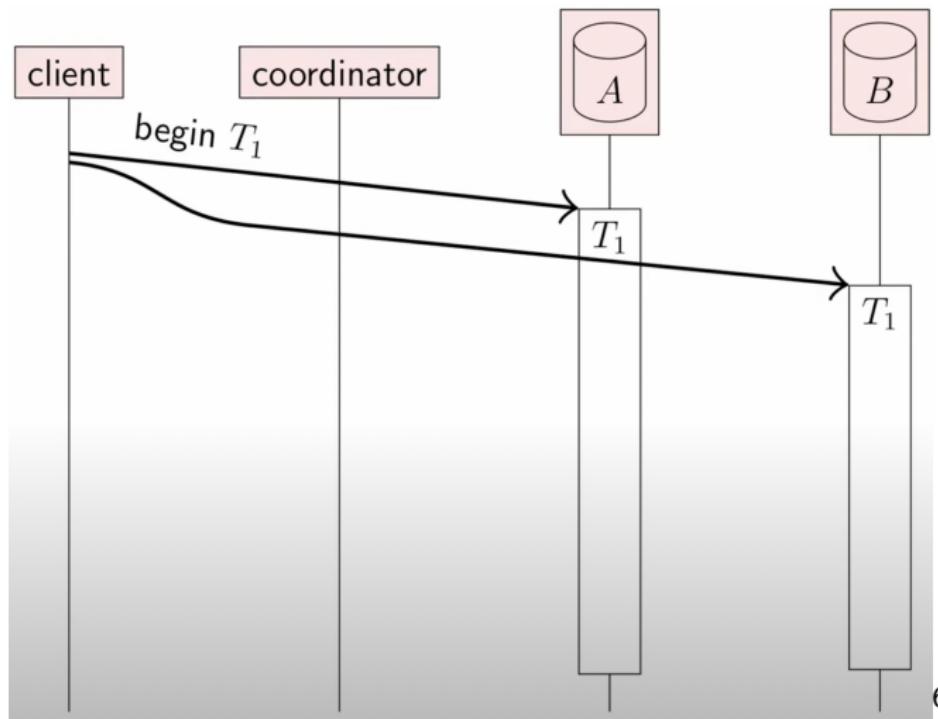
# Two-phase Commit



5

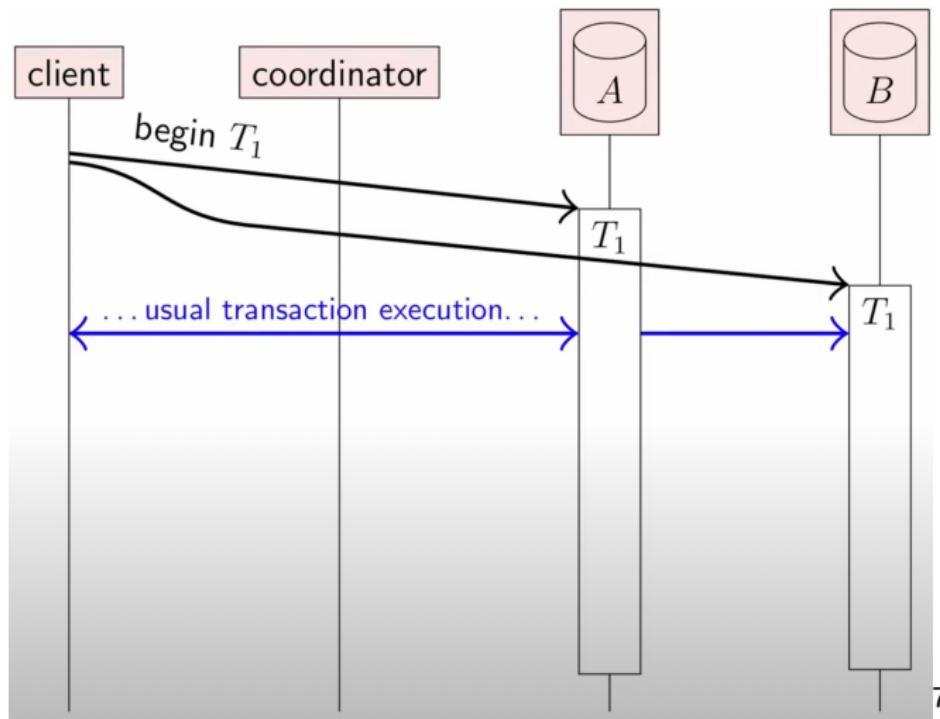
<sup>5</sup>From Dr Martin Kleppmann

# Two-phase Commit



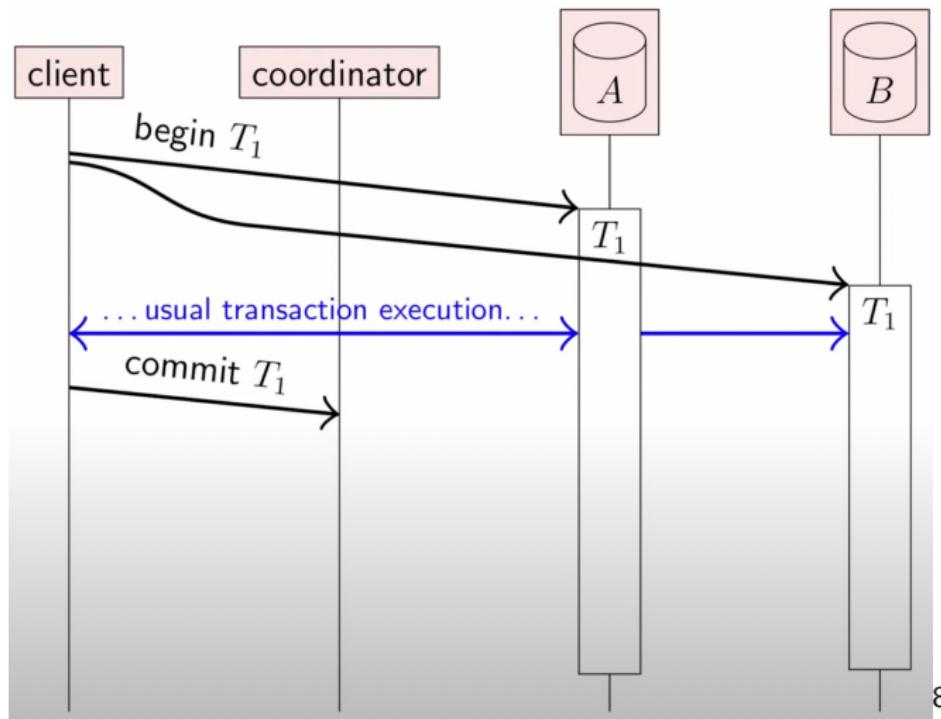
<sup>6</sup>From Dr Martin Kleppmann

# Two-phase Commit



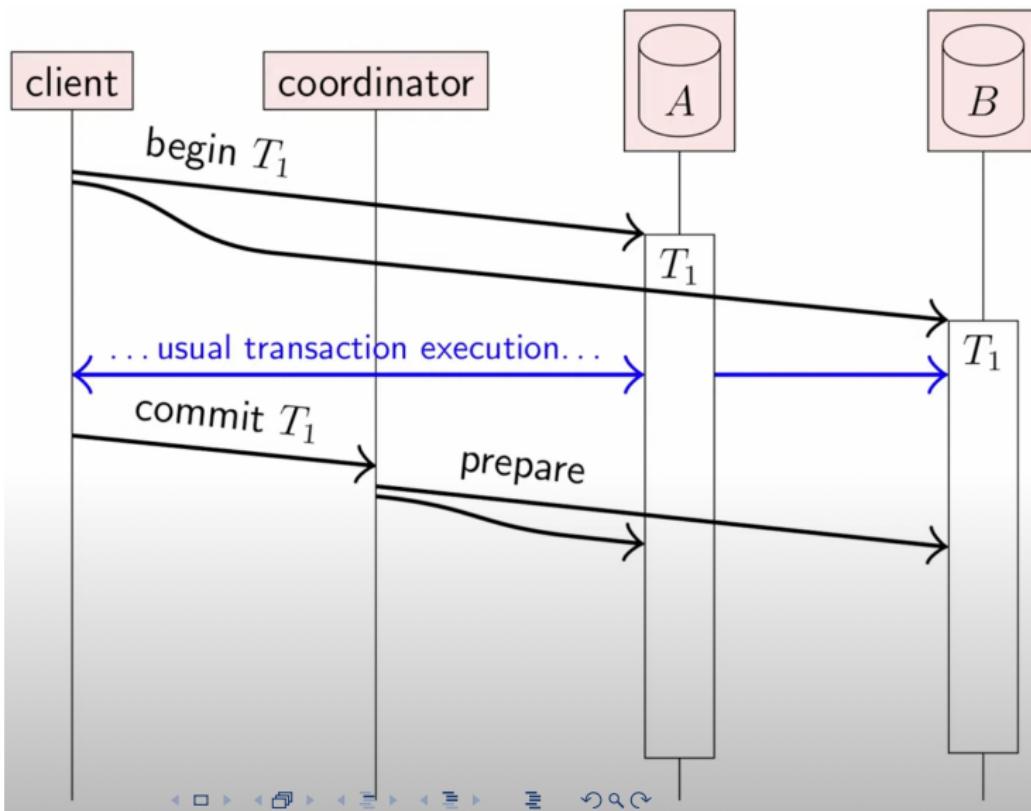
<sup>7</sup>From Dr Martin Kleppmann

# Two-phase Commit

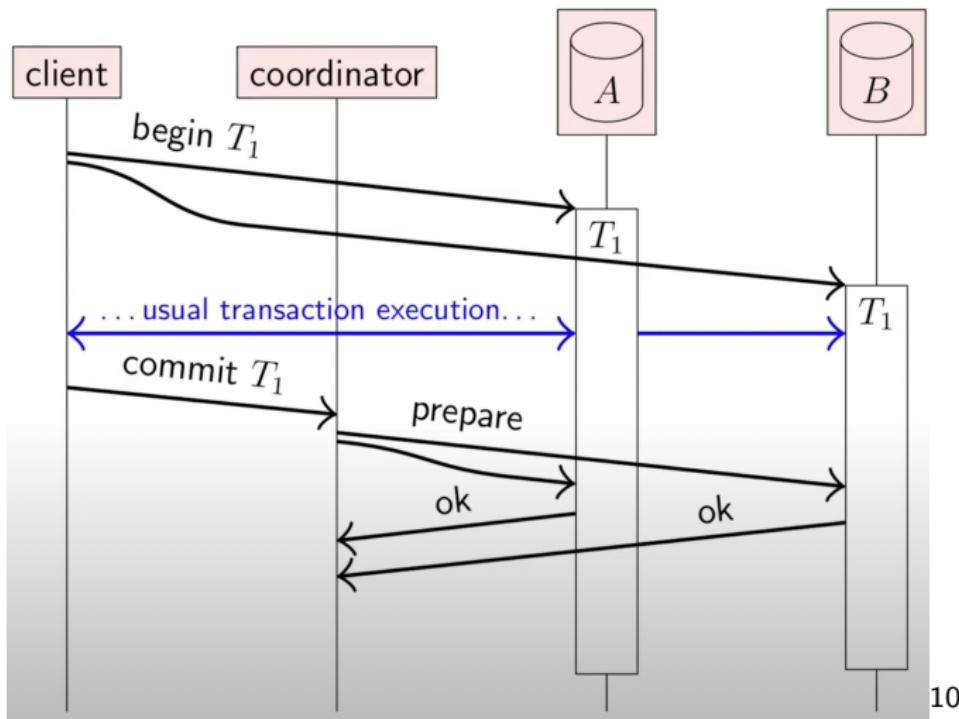


<sup>8</sup>From Dr Martin Kleppmann

# Two-phase Commit



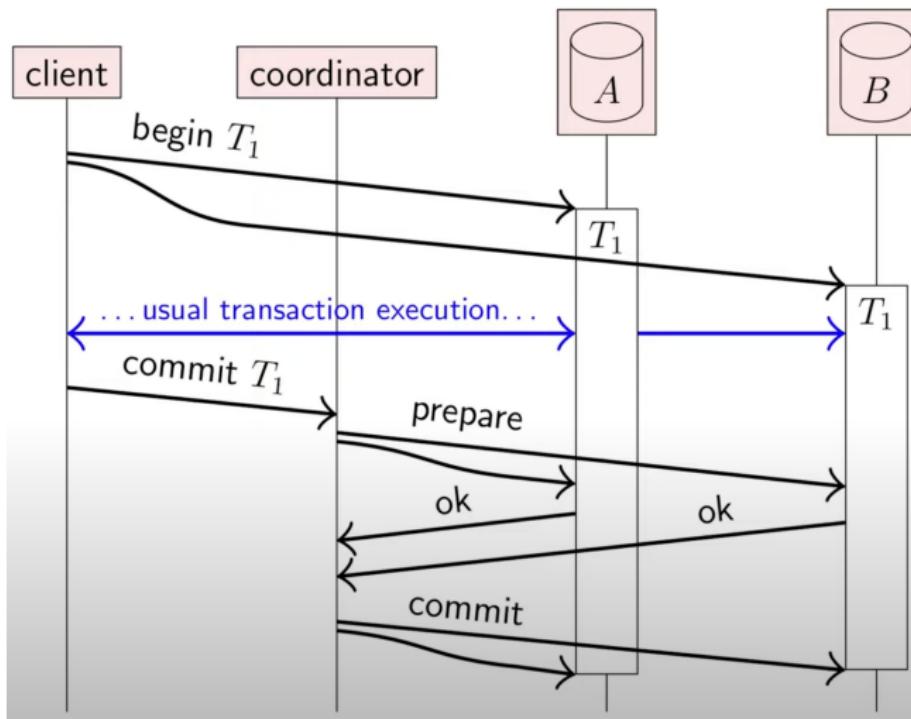
# Two-phase Commit



10

<sup>10</sup>From Dr Martin Kleppmann

# Two-phase Commit



11

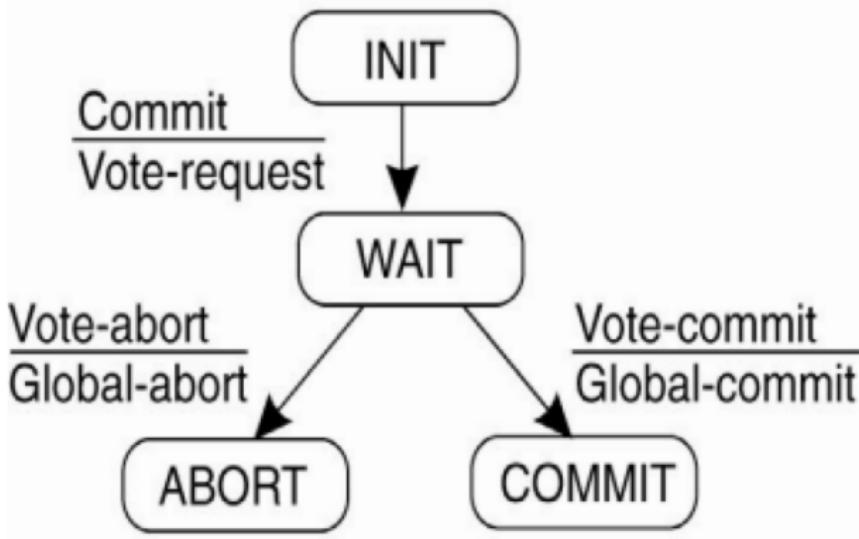
<sup>11</sup>From Dr Martin Kleppmann

# Two-phase Commit

- Fase inicial: el cliente envía mensaje de comienzo de la transacción a los servidores
- Fase operacional: el cliente interactúa con los servidores y realiza las operaciones correspondientes
- Fase 1a: el cliente le indica al coordinador el fin de la transacción
- Fase 1b: el coordinador envía el mensaje (prepare) a todos los servidores para que indiquen si voto: commit o abort
- Fase 1c: los servidores envían el voto al coordinador
- Fase 2a: el coordinador recoge todos los votos: si alguno es abort, el resultado final es abort. Si todos son commit, el resultado es commit.
- Fase 2c: cada servidor espera el mensaje de confirmación: commit o abort.

# Two-phase Commit

Máquina de estados del coordinador



Coordinator

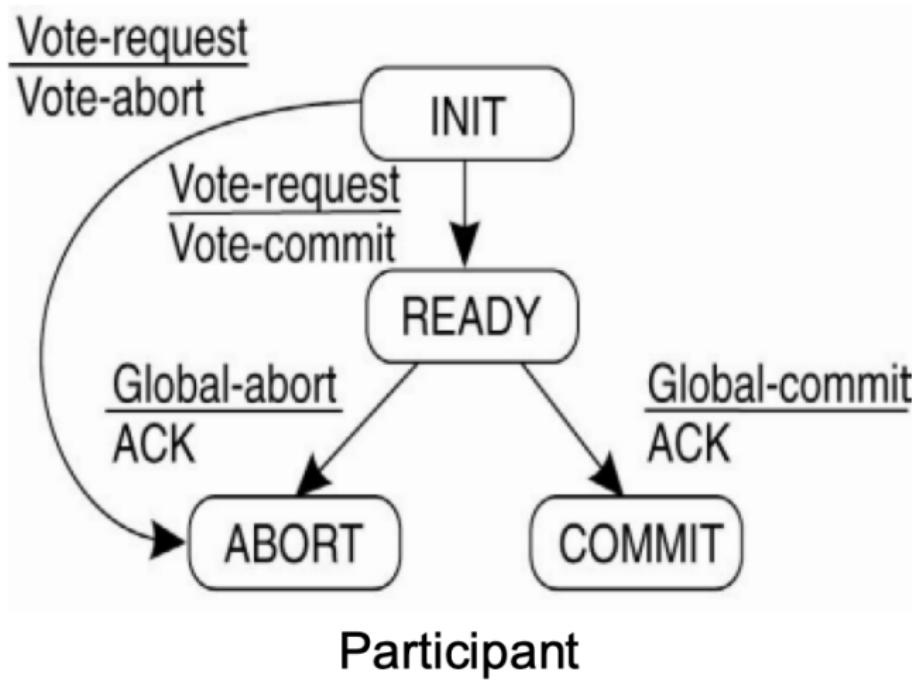
12

transacciones  
**37/40**

<sup>12</sup>From M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.

# Two-phase Commit

Máquina de estados del servidor



# Two-phase Commit

## ¿Y si el coordinador falla?

- Podría escribir su decisión en disco y al recuperarse, continuar

# Two-phase Commit

## ¿Y si el coordinador falla?

- Podría escribir su decisión en disco y al recuperarse, continuar
- ¿Pero y si el coordinador falla justo después de haber solicitado los votos a los servidores (prepare)?
  - Todos los servidores se quedan a la espera de qué hacer: commit or abort

# Two-phase Commit

## ¿Y si el coordinador falla?

- Podría escribir su decisión en disco y al recuperarse, continuar
- ¿Pero y si el coordinador falla justo después de haber solicitado los votos a los servidores (prepare)?
  - Todos los servidores se quedan a la espera de qué hacer: commit or abort
- Una solución: replicación y consenso (por ejemplo Raft) para el coordinador

# Conclusiones

# Conclusiones

## Consistencia y Transacciones Distribuidas

- La consistencia puede utilizarse para mejorar las prestaciones o para incrementar la tolerancia a fallos
- La consistencia para incrementar transacciones define diferentes modelos de cómo y cuándo las réplicas tienen que ser “consistentes”
- Las transacciones distribuidas surgen como necesidad de un cliente de interactuar con múltiples servidores (con estado) simultáneamente. De manera que o se ejecutan todos las operaciones o ninguna
- Las transacciones distribuidas guardan cierta similitud con el problema de consenso, aunque presentan 3 grandes diferencias
- Como ejemplo representativo de transacciones distribuidas estudiamos el protocolo commit en dos fases

# **Consistencia y Transacciones Distribuidas**

**30221 - Sistemas Distribuidos**

**Rafael Tolosana**

**Dpto. Informática e Ing. de Sistemas**