DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Procesamiento de transacciones

© Fernando Berzal, berzal@acm.org

Procesamiento de transacciones

- Transacciones
- ACIDez de las transacciones
- Implementación
  - Logs de transacciones
  - Versiones
- Procesamiento de transacciones distribuidas
  - 2-phase commit
  - 3-phase commit
- El teorema CAP (redux)

Motivación

Los usuarios finales no "ven" los datos directamente:
SQL no es la interfaz adecuada para usuarios finales.

- Los usuarios finales interactúan con aplicaciones:
  Programas con múltiples consultas.

Ejecución de aplicaciones

- Múltiples usuarios simultáneos.
- Cada uno de ellos espera un funcionamiento "correcto"
  … sin tener que esperar indefinidamente.
  … sin verse afectado por errores ajenos.

Motivación

EJEMPLO: Cajero automático

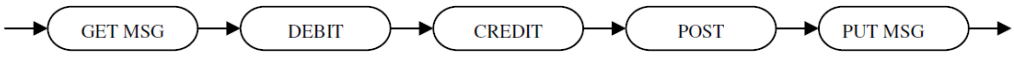
Muchos clientes de un banco realizan
operaciones que han de completarse
simultáneamente: Ejecución entrelazada [interleaving].

- Equidad [fairness]: Cada usuario utiliza el sistema
  como si fuese el único usuario en ese momento.

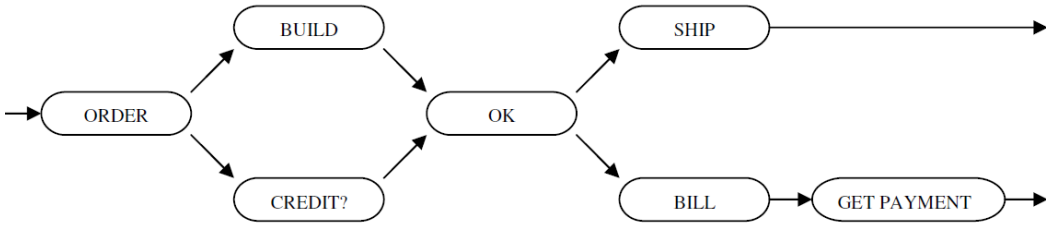
- Utilización eficiente de recursos,
  p.ej. CPU asignada a otros usuarios cuando se espera
  la finalización de operaciones de E/S.

Transacciones

Transacciones simples



Transacciones complejas (paralelismo & anidación)



Transacciones

Definiciones de transacción

- Informal:
  Unidad de cambio en la base de datos.

- Algo más formal:
  Ejecución de un programa sobre la base de datos

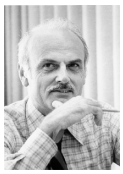
  NOTA: Las aplicaciones son conjuntos de transacciones.

https://en.wikipedia.org/wiki/Database_transaction

Transacciones

¿Quién inventó las transacciones?

- ¿Edgar F. Codd?



https://en.wikipedia.org/wiki/Edgar_F._Codd
"A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13(6):377–387, 1970. DOI 10.1145/362384.362685

- ¿Jim Gray?

https://en.wikipedia.org/wiki/Jim_Gray_(computer_scientist)
"The Transaction Concept: Virtues and Limitations". Proceedings of the 7th International Conference on Very Large Databases, 1981.

Transacciones

Dr. Jim Gray worked at the IBM San Jose Research Laboratory from October 1972 until September 1980. During that time he developed and implemented the foundational techniques that underlie and enable on-line transaction processing. The deployment of on-line transaction processing reduces the cost of business transactions by reducing delays and eliminating paper records. Dr. Gray received the 1998 A.M. Turing Award *"For fundamental contributions to database and transaction processing research and technical leadership in system implementation from research prototypes to commercial products. The transaction is the fundamental abstraction underlying database system concurrency and failure recovery. Gray's work [defined] the key transaction properties: atomicity, consistency, isolation and durability, and his locking and recovery work demonstrated how to build ... systems that exhibit these properties."*

"Jim Gray at IBM: the transaction processing revolution." Bruce G. Lindsay, ACM SIGMOD Record, 37(2). June 2008.

Transacciones

1.1 Historical Perspective

Six thousand years ago, the Sumerians invented writing for transaction processing. The earliest known writing is found on clay tablets recording the royal inventory of taxes, land, grain, cattle, slaves, and gold; scribes evidently kept records of each transaction. This early system had the key aspects of a transaction processing system (see Figure 1.1):

Database. An abstract system state, represented as marks on clay tablets, was maintained. Today, we would call this the *database*.

Transactions. Scribes recorded state changes with new records (clay tablets) in the database. Today, we would call these state changes *transactions*.

The Sumerians' approach allowed the scribes to easily ask questions about the current and past state, while providing a historical record of how the system got to the present state.

The technology of clay-based transaction processing systems evolved over several thousand years through papyrus, parchment, and then paper. For over a thousand years, pa-

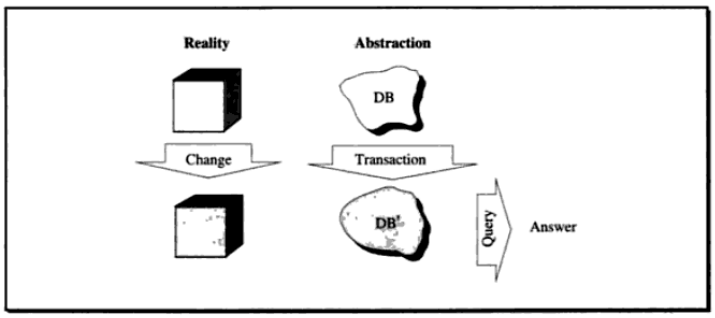
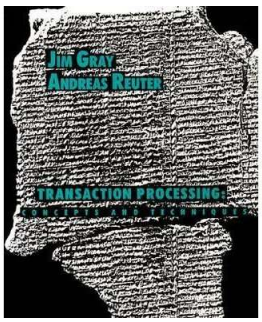


Figure 1.1: The basic abstraction of transaction processing systems. The real state is represented by an abstraction, called the *database*, and the transformation of the real state is mirrored by the execution of a program, called a *transaction*, that transforms the database



ACIDez de las transacciones

ACIDez de las transacciones

Atomicidad [atomicity]:
Las transacciones deben ser atómicas (o todos sus efectos o ninguno).

Consistencia [consistency preservation]
Base de datos consistente antes y después de la transacción (puede que no durante la transacción)

Aislamiento [isolation]

El resultado de la ejecución concurrente de transacciones es el mismo que si se ejecutasen secuencialmente.

Persistencia/durabilidad [durability]

Una vez completada su ejecución, los cambios realizados por una transacción son permanentes.

https://en.wikipedia.org/wiki/ACID

ACIDez de las transacciones

D – Durabilidad/persistencia

Los cambios realizados por una transacción son permanentes: nadie puede cambiar la transacción y el sistema debe garantizar su durabilidad aunque se produzcan fallos.

Fallo de persistencia: En una transferencia bancaria, el usuario cree que la transacción ha terminado pero los datos están en un buffer de disco gestionado por el sistema operativo y falla el suministro eléctrico antes de que los datos se almacenen físicamente en el disco.

ACIDez de las transacciones

I - Aislamiento

Los efectos de una transacción no son visibles para las demás transacciones hasta que termina su ejecución:

La ejecución de una transacción no debe interferir en la ejecución de otras transacciones simultáneas.

Fallo de aislamiento: Dos transferencias simultáneas sobre la misma cuenta acceden en paralelo a su saldo, sin que el sistema fuerce a que la primera transferencia termine antes de comenzar la segunda.

ACIDez de las transacciones

I - Aislamiento

SET TRANSACTION ISOLATION LEVEL…

Garantizar el aislamiento absoluto [**serializable**] puede afectar al rendimiento y no resultar siempre necesario:

- Lecturas "sucias" [**dirty reads**] de datos modificados por transacciones que aún no han finalizado.
- Lecturas "comprometidas" [**committed reads**], sólo de datos modificados por transacciones ya finalizadas.
- Lecturas "repetibles" [**repeatable reads**] si, dentro de una transacción, siempre obtendremos los mismos valores para los mismos datos.

ACIDez de las transacciones

I - Aislamiento

SET TRANSACTION ISOLATION LEVEL…

Nivel de aislamiento	Lecturas sucias	Lecturas no repetibles	"Phantoms"
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

Phantoms: Tuplas recién insertadas (por otras transacciones).

ACIDez de las transacciones

C - Consistencia

La ejecución atómica de una transacción
lleva a la base de datos de un estado consistente
(en el que se satisfacen todas las restricciones)
a otro estado, también consistente.

Que las transacciones mantengan la consistencia del sistema es **responsabilidad del programador**.

ACIDez de las transacciones

A - Atomicidad

Una transacción…

- … o bien se ejecuta completamente [**commit**],
- … o bien deja todo como si nunca hubiese comenzado a ejecutarse [**abort**/**rollback**].

En bases de datos SQL:

Comienzo de la transacción:

- ORACLE: Tras cada COMMIT o ROLLBACK (salvo que activemos AUTOCOMMIT).
- MySQL: START TRANSACTION.

Fin de la transacción: COMMIT o ROLLBACK.

ACIDez de las transacciones

A - Atomicidad

Las operaciones de lectura no causan problemas.

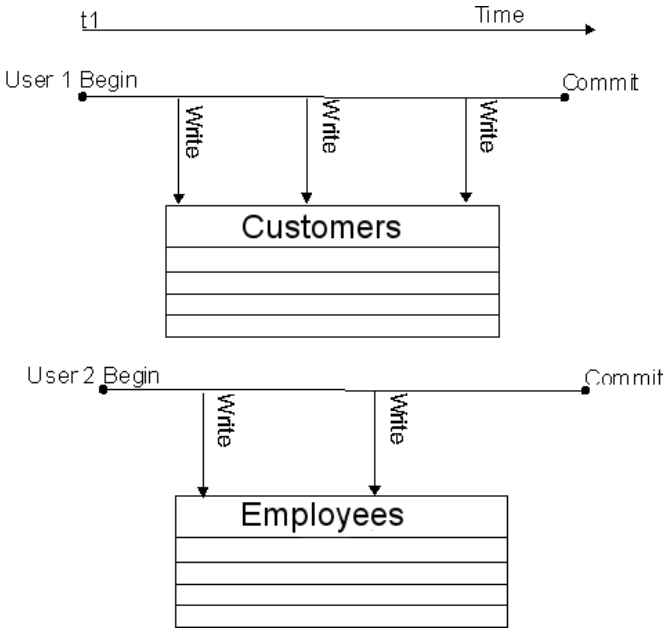
- SET TRANSACTION READ ONLY
  permite realizar optimizaciones.

Las operaciones de escritura hay que gestionarlas:

- **commit**
  para dejar la base de datos en un estado consistente.
- **abort**/**rollback**
  para deshacer los cambios realizados por la transacción.

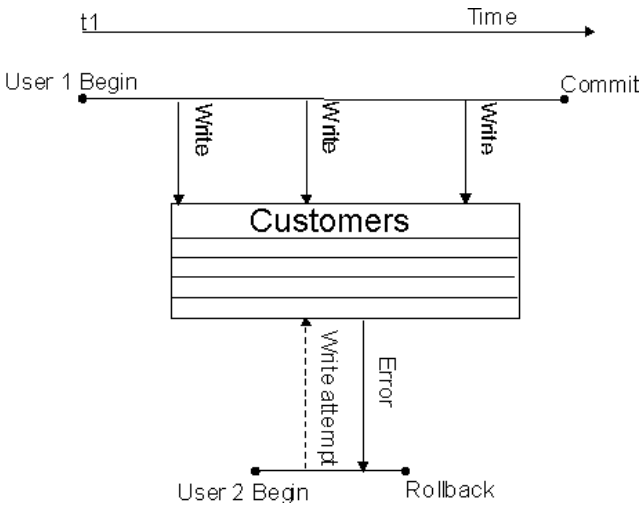
Implementación

Múltiples escritores a la vez… sobre recursos diferentes



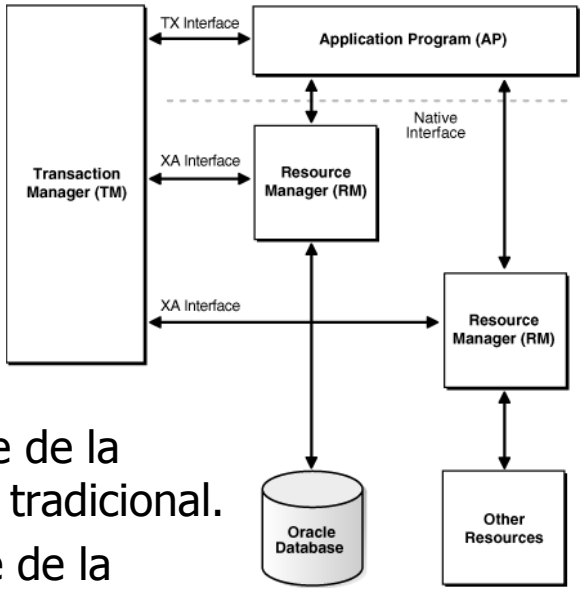
Implementación

Sólo un escritor a la vez… sobre un mismo recurso



Implementación

Gestor de transacciones
/ Monitor de procesamiento de transacciones
Transaction manager
/ TP monitor



DBMS: Componente clave de la arquitectura de un DBMS tradicional.

Middleware: Responsable de la coordinación entre recursos distribuidos.

Implementación

Gestor de transacciones
/ Monitor de procesamiento de transacciones
Transaction manager
/ TP monitor

Responsabilidades:

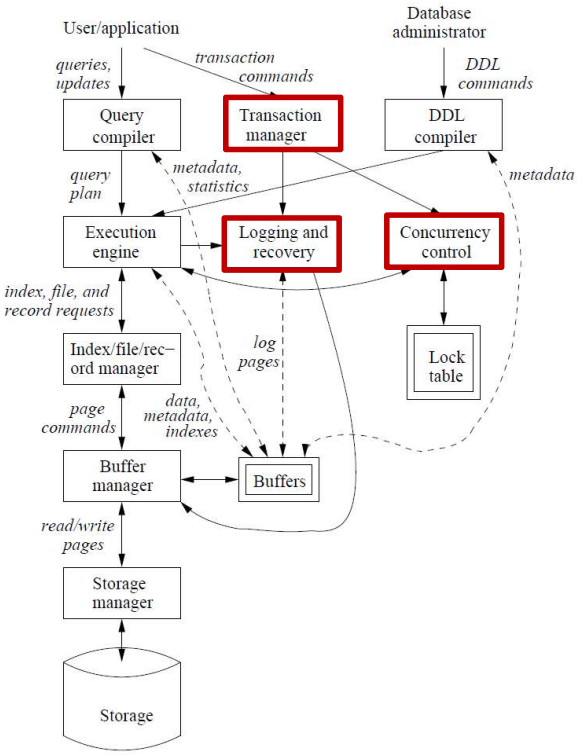
- Demarcación de transacciones (begin/commit/rollback).
- Planificación equitativa de su ejecución [fairness].
- Registro de sus actividades (writes, commits & aborts).
- Detección de conflictos, p.ej. deadlocks.
- Ejecución de tareaas de recuperación [recovery].

Implementación

Arquitectura de un DBMS

"Database Systems: The Complete Book"

Hector Garcia-Molina, Jeffrey D. Ullman & Jennifer Widom



Implementación

Gestor de transacciones
/ Monitor de procesamiento de transacciones
Transaction manager
/ TP monitor

Aspectos esenciales:

- Recuperación [**recovery**]: Realizar tareas que permitan restaurar la base de datos en un estado consistente.
- Control de concurrencia [**concurrency control**]:
  - Evitar que transacciones simultáneas puedan interferir.
  - Factor clave en el rendimiento del sistema (niveles de aislamiento ajustables).

Implementación

Logs de transacciones (a.k.a. journals)

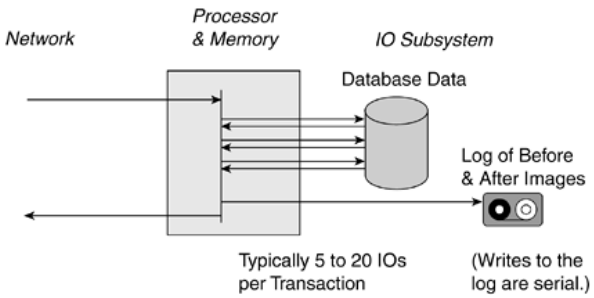
Sirven para mantener un seguimiento de la ejecución de las transacciones (crucial para su recuperación).

¿Qué contienen?

- Inicio/final de las transacciones (commits & aborts).
- Operaciones de escritura
  (imágenes de los datos antes y/o después)
  - **Antes**, para deshacer transacciones [rollback], p.ej. transacciones abortadas.
  - **Después**, para rehacer transacciones [redo], p.ej. recuperación tras un fallo catastrófico.

Implementación

Logs de transacciones (a.k.a. journals)



Network Throughput Requirements

100 transactions per second

200 bytes in
+ 2,000 bytes out
= 2,200 bytes per transaction
= 220,000 bytes per second
Add headers, etc. and convert to bits
= 2.2 MB per second approximately

Disk Throughput Requirements

100 transactions per second

At 8 to 20 IOs per transaction
= 800 to 2,000 IOs per second

At 4,000 bytes per IO
= 3.2 to 8 MBs per second

If average 50 IOs per second per disk
= 16 to 40 disks

Implementación

Logs de transacciones

ABSTRACCIÓN

Base de datos compuesta de elementos.

- Tuplas
- Bloques de disco (lo más usual).
- Relaciones (posibles problemas de rendimiento).

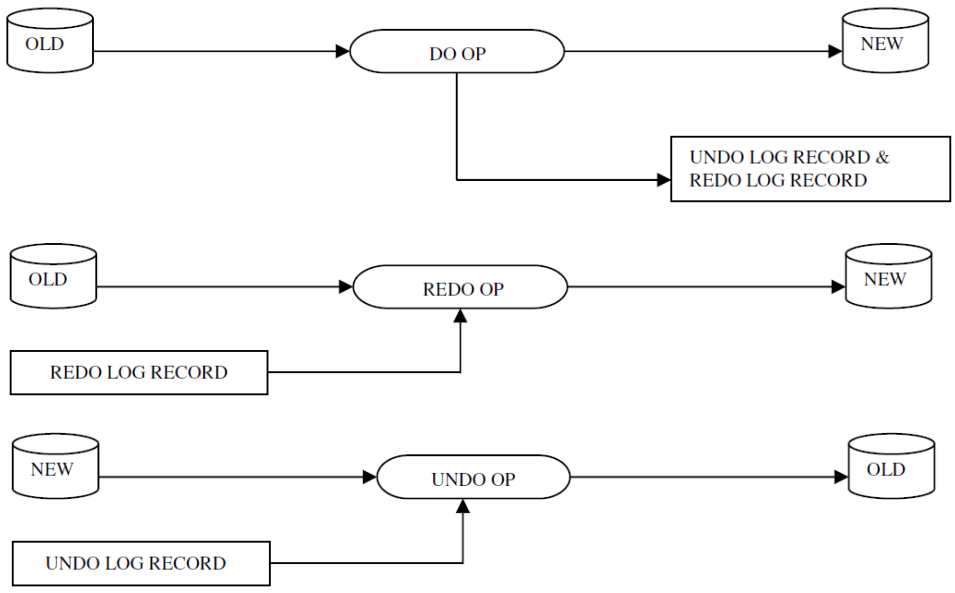
Cada transacción lee/escribe algunos elementos.

En el log, un fichero "append-only", se registran las operaciones realizadas por las distintas transacciones…

Implementación

Logs de transacciones

DO / UNDO / REDO



Implementación

Logs de transacciones

UNDO LOG

Registros del log:

- <START T> Comienzo de transacción
- <COMMIT T> Transacción finalizada con éxito
- <ABORT T> Transacción abortada
- <T,X,v> T actualiza el valor de X
  v era el antiguo valor de X
  (por si hay que deshacer T)

Implementación

Logs de transacciones

UNDO LOG

Reglas de funcionamiento:

- Si T modifica X, la entrada <T,X,v> debe escribirse en el log antes de X se escriba en disco.
- Si T finaliza con commit, <COMMIT T> debe escribirse en el log sólo después de que todos los cambios de T se hayan guardado en disco.

Las escrituras se hacen **pronto** (antes del commit).

Implementación

Logs de transacciones

UNDO LOG

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						<COMMIT T>

Implementación

Logs de transacciones

UNDO LOG RECOVERY

Reglas de recuperación tras un fallo del sistema:

- Decidir, para cada transacción, si se completó o no:
  - <START T>…<COMMIT T> OK
  - <START T>…<ABORT T> OK
  - <START T>… error
- Deshacer todas las modificaciones efectuadas por las transacciones no completadas.

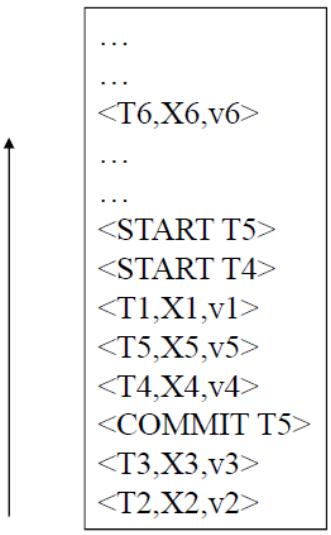
Implementación

Logs de transacciones

UNDO LOG RECOVERY

Se lee el log desde el final:

- <COMMIT T> o <ABORT T>:
  Marcar T como completada.
- <T,X,v>:
  Si T no se ha completado,
  escribir X=v en disco.



Operaciones idempotentes (si se repiten una segunda vez, no pasa nada, p.ej. si falla el sistema durante el proceso de recuperación).

Implementación

Logs de transacciones

REDO LOG

Registros del log:

- <START T> Comienzo de transacción
- <COMMIT T> Transacción finalizada con éxito
- <ABORT T> Transacción abortada
- <T,X,v> T actualiza el valor de X
  v es el nuevo valor de X
  (por si hay que rehacer T)

Implementación

Logs de transacciones

REDO LOG

Regla de funcionamiento:

Si T modifica X,
tanto la entrada <T,X,v> como <COMMIT T>
deben escribirse en el log
antes de X se escriba en disco.

Las escrituras hacen **tarde** (después del commit).

Implementación

Logs de transacciones

REDO LOG

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Implementación

Logs de transacciones

REDO LOG RECOVERY

Reglas de recuperación tras un fallo del sistema:

- Decidir, para cada transacción, si se completó o no:
  - <START T>…<COMMIT T> OK
  - <START T>…<ABORT T> OK
  - <START T>… error
- Rehacer todas las modificaciones efectuadas por las transacciones que se completaron con un **commit**.

Implementación

Logs de transacciones

REDO LOG RECOVERY

Se lee el log desde el principio:

- <T,X,v>:
  Si T se completó con **commit** se escribe X=v en disco.

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…

Implementación

Logs de transacciones

UNDO vs. REDO

Undo logging

- Escritura en disco antes del COMMIT en el log.
- Al encontrarnos <COMMIT T>, sabemos que todos los datos modificados por T están en disco (no hay que deshacer nada).
- No se puede actualizar una copia de seguridad de la BD!!!

Redo logging

- Escritura en disco después del COMMIT en el log.
- Si no nos encontramos <COMMIT T>, T no ha escrito ningún dato en disco ["no dirty data"]: hay que mantener los bloques en memoria hasta el commit.

Implementación

Logs de transacciones

UNDO/REDO LOG

Registros de las modificaciones en el log: <T,X,u,v>, guardando tanto el valor antiguo (u) como el nuevo (v).

Regla de funcionamiento:
Si T modifica X, <T,X,u,v> debe registrarse en el log antes de que X se escriba en disco.

Ventaja:
Da igual que escribamos antes o después del COMMIT.

Implementación

Logs de transacciones

UNDO/REDO LOG

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
OUTPUT(A)	16	16	16	16	8	
						<COMMIT T>
OUTPUT(B)	16	16	16	16	16	

Implementación

Logs de transacciones

UNDO/REDO LOG RECOVERY

Tras un fallo del sistema:

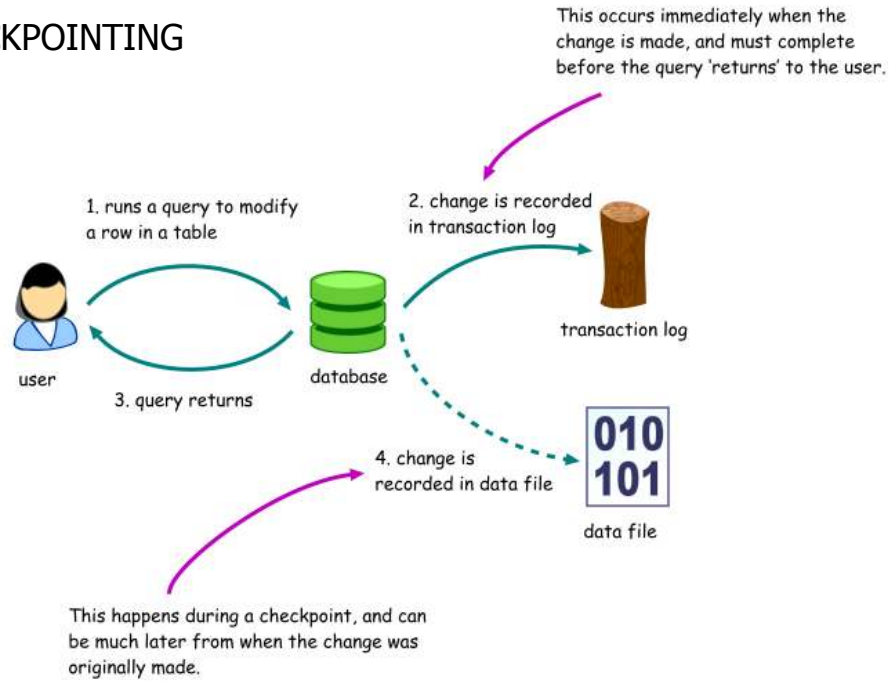
1. Rehacer todas las transacciones finalizadas con un commit (hacia adelante).
2. Deshacer todas las transacciones no finalizadas con éxito (hacia atrás).

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…

Implementación

Logs de transacciones

CHECKPOINTING



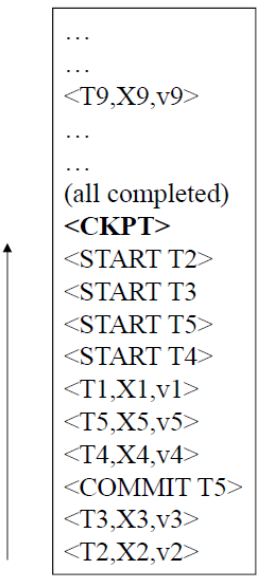
Implementación

Logs de transacciones

CHECKPOINTING

Para no tener que rehacer/deshacer el log completo en caso de fallo, periódicamente:

- Se dejan de aceptar nuevas transacciones
- Se espera a que se completen todas las transacciones actuales.
- Se añade un registro <CKPT> al log.
- Se reanudan las transacciones.

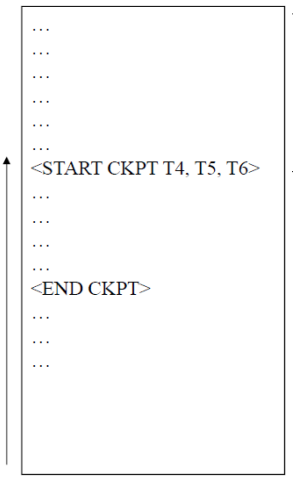


Problema: Se bloquea la BD durante el checkpoint…

Implementación

Logs de transacciones

NONQUIESCENT CHECKPOINTING



Solución:

Checkpoint no quiescente

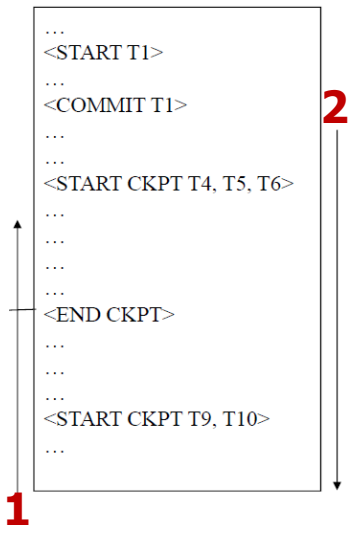
UNDO LOG CHECKPOINT

- Registro <START CKPT(T1,..,Tk)> donde T1..Tk son las transacciones activas.
- ... continúa el funcionamiento normal ...
- Registro <END CKTP> cuando se completan todas las transacciones activas al comenzar el checkpoint.

Implementación

Logs de transacciones

NONQUIESCENT CHECKPOINTING



REDO LOG CHECKPOINT

- Registro <START CKPT(T1,..,Tk)> donde T1..Tk son las transacciones activas.
- ... continúa el funcionamiento normal ... mientras se escriben en disco todos los bloques de las transacciones finalizadas con **commit** [dirty blocks].
- Registro <END CKTP> cuando se completan todas las transacciones activas al comenzar el checkpoint.

Implementación

Logs de transacciones

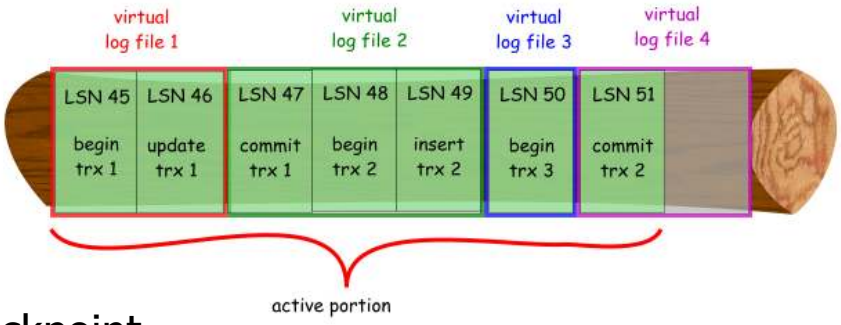
* trx = transaction

LSN = Log Sequence Number

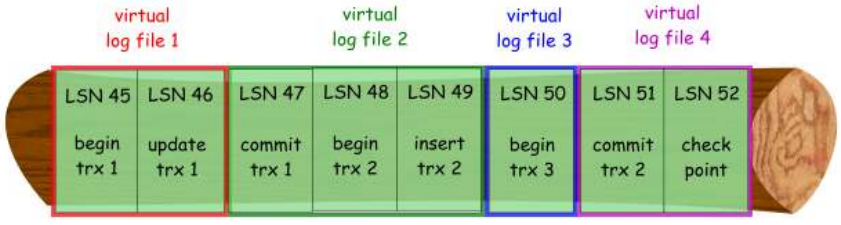
Implementación

Logs de transacciones

Sección activa del log [full database recovery]:



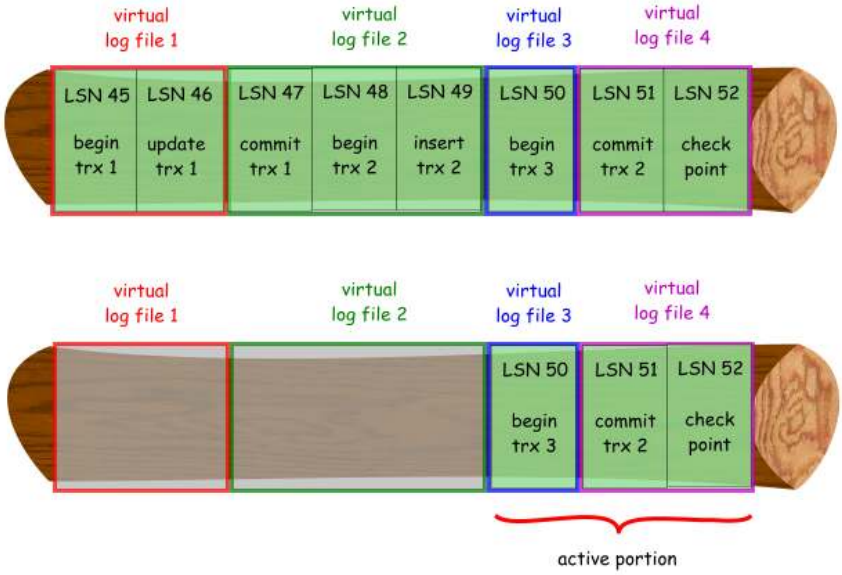
Checkpoint



Implementación

Logs de transacciones

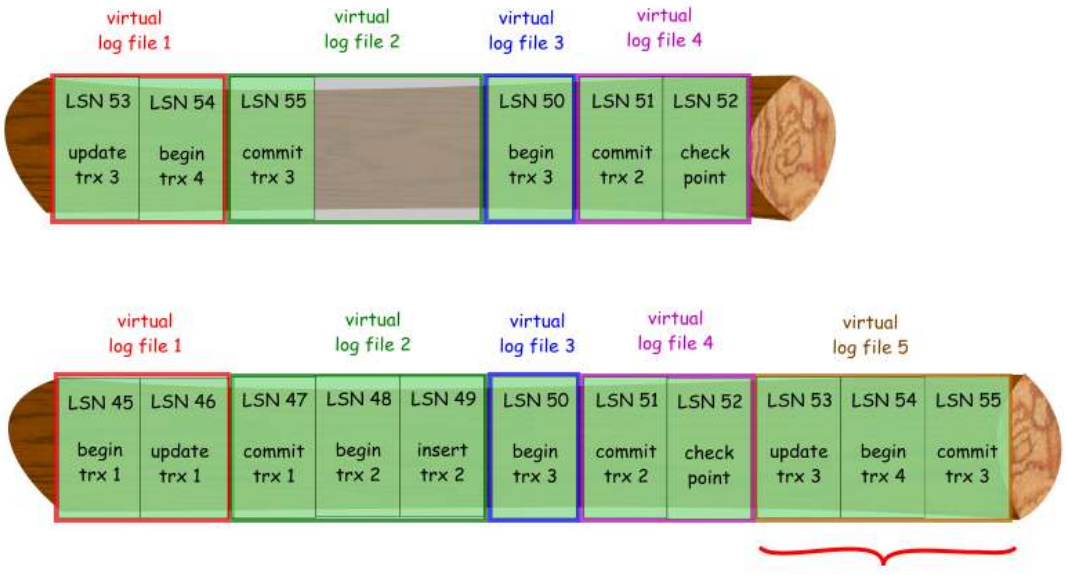
Checkpoint



Implementación

Logs de transacciones

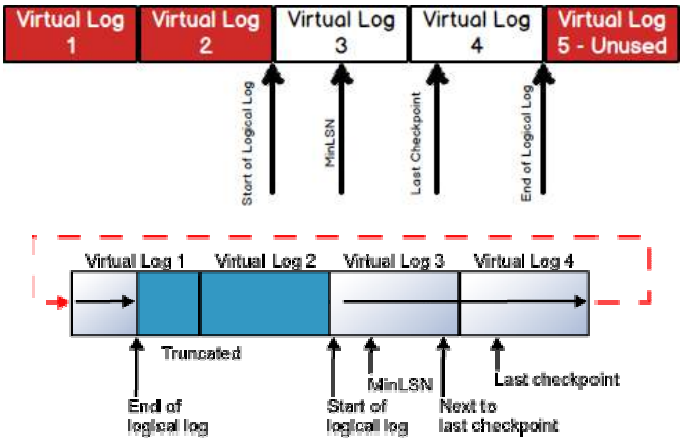
Reutilización & crecimiento del log

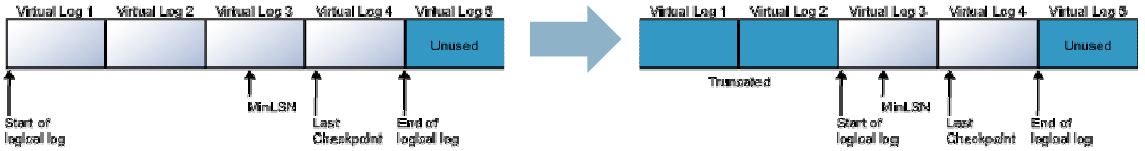


Implementación

Logs de transacciones

SQL SERVER

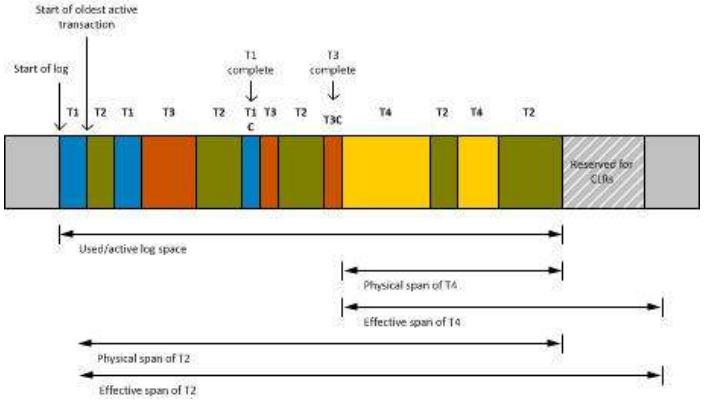




Implementación

Logs de transacciones

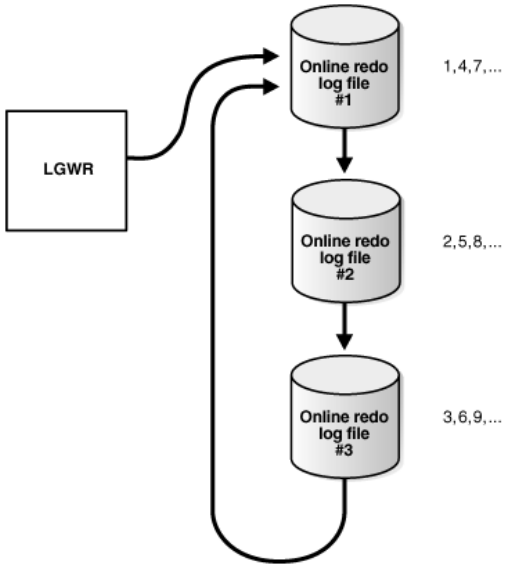
SYBASE

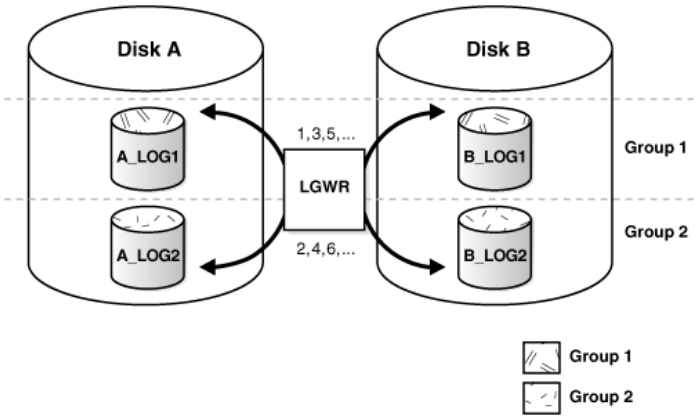


Implementación

Logs de transacciones

ORACLE "REDO" LOGS



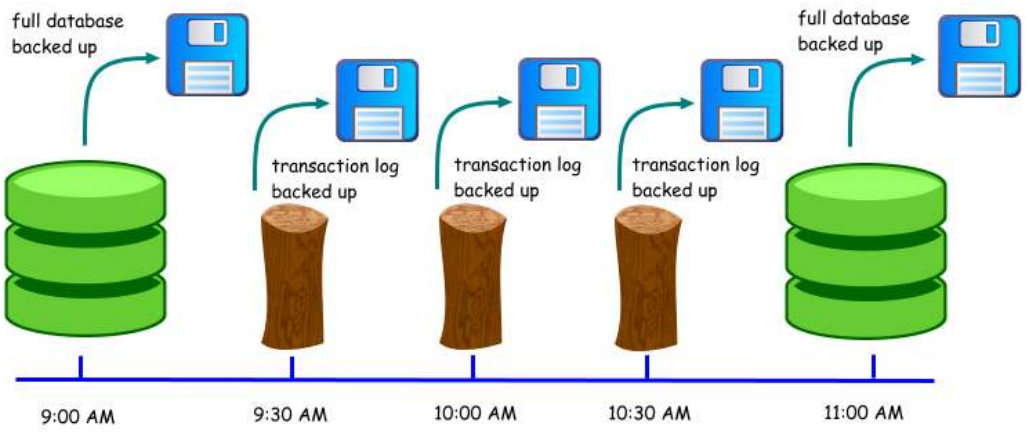


Logs multiplexados (múltiples copias)

Implementación

Logs de transacciones

Realización de copias de seguridad "diferenciales"

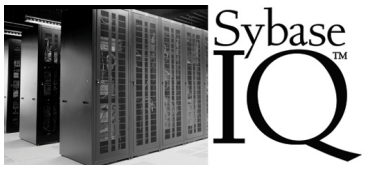


Implementación

Logs vs. Versioning

Una solución alternativa:

Los datos nunca se modifican, sino que se crean distintas versiones de los mismos.

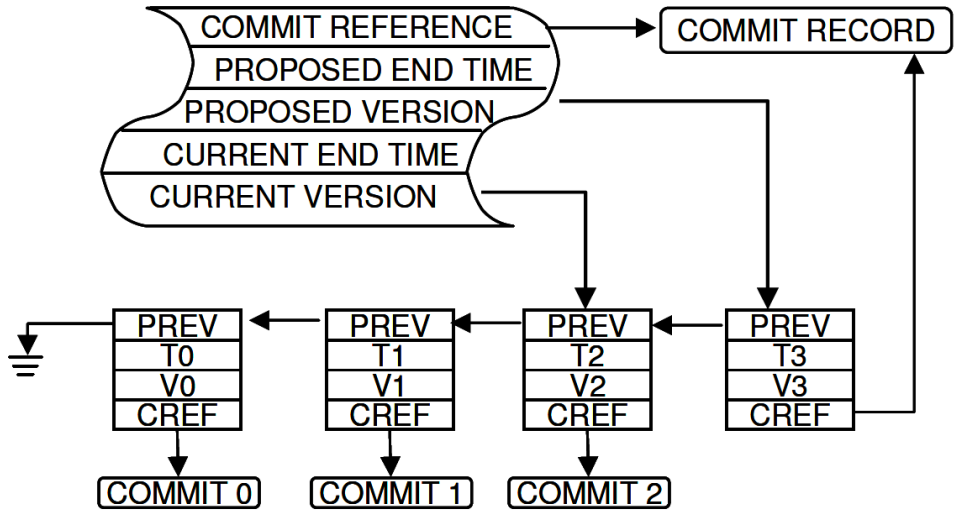


Modificar un dato de un objeto se transforma en crear un nuevo valor y asociárselo al objeto como valor actual.

Los valores antiguos siguen existiendo y se puede acceder a ellos especificando un instante de tiempo del intervalo durante el que eran los valores "actuales".

Implementación

Versioning: "version-oriented systems"

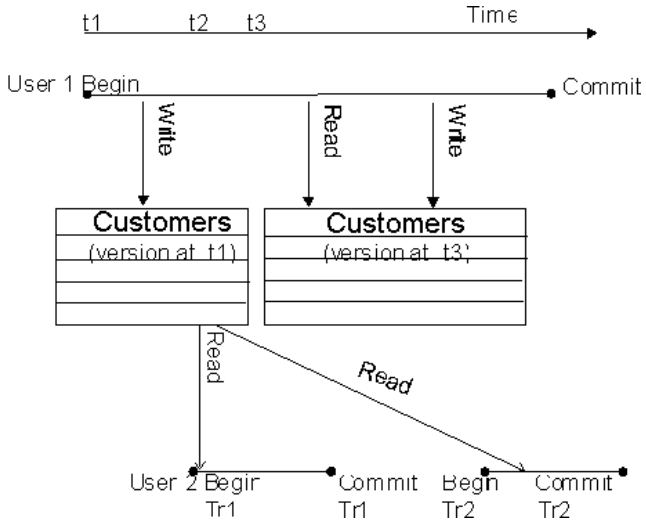


a.k.a. "time-domain addressing"
a.k.a. "immutable object systems"

Implementación

Versioning

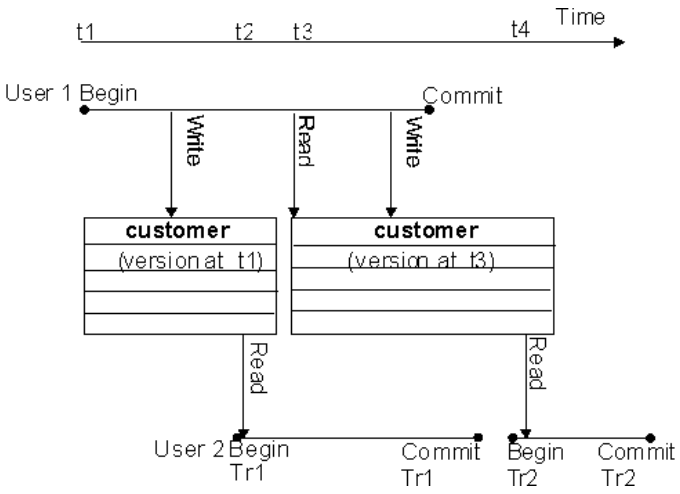
Cada transacción utiliza la última versión para la que se ha realizado un commit:



Implementación

Versioning

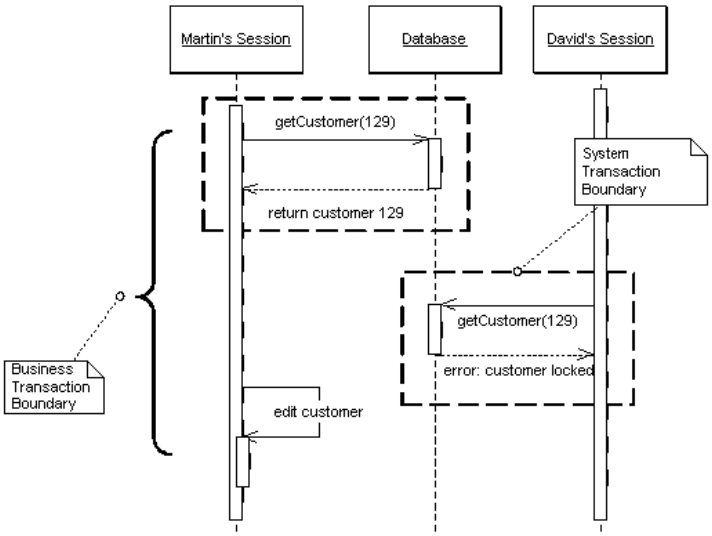
Los commits de las transacciones de lectura tienen implicaciones para otras transacciones:



Implementación

Bloqueo pesimista vs. Bloqueo optimista

Pessimistic lock

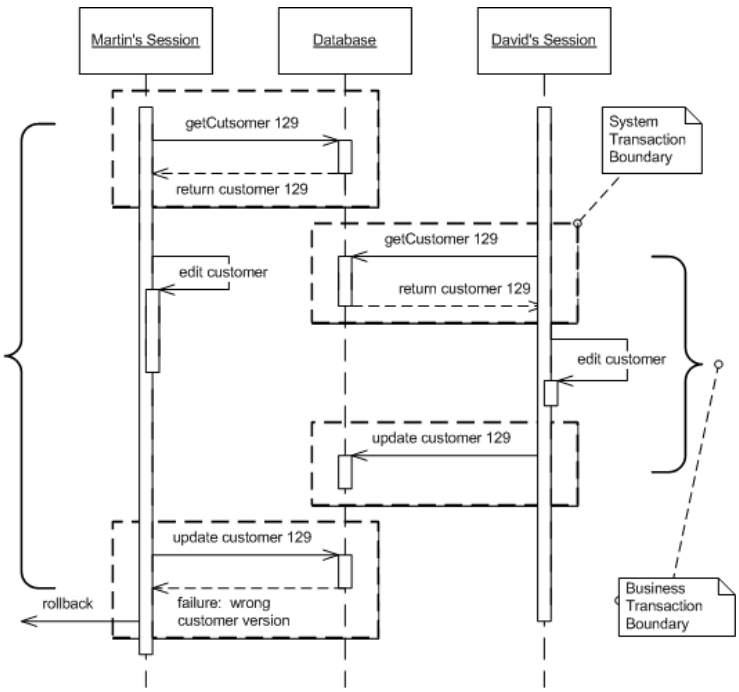


Evita el conflicto entre transacciones permitiendo que sólo una de ellas acceda a los datos.

Implementación

Bloqueo pesimista vs. Bloqueo optimista

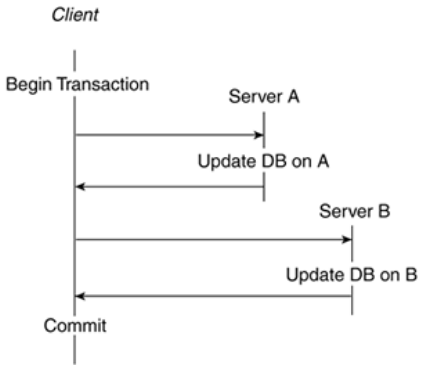
Optimistic lock



Previene las interferencias detectando el conflicto y deshaciendo la transacción [rollback].

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

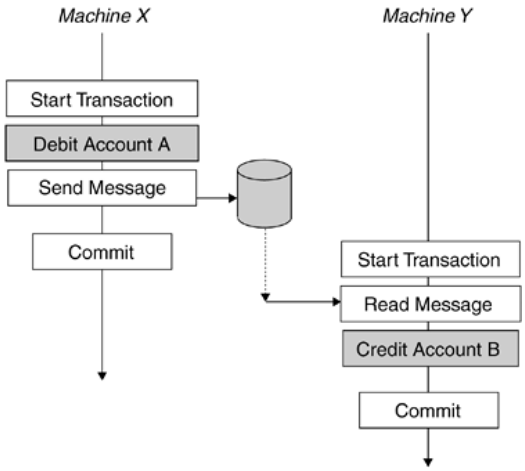


Procesamiento de transacciones

Procesamiento de transacciones distribuidas

USO DE COLAS DE MENSAJES

[implementación incorrecta]

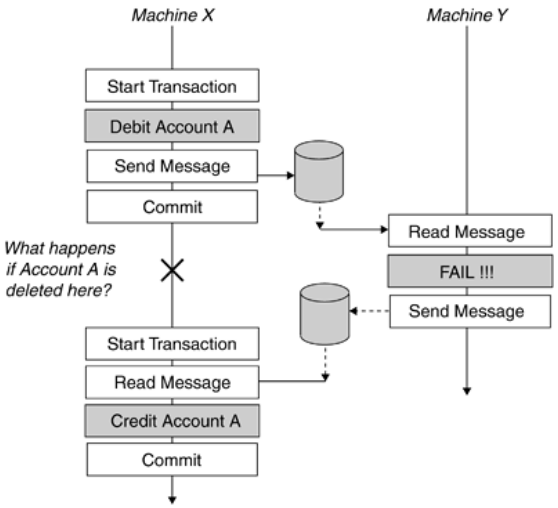


Procesamiento de transacciones

Procesamiento de transacciones distribuidas

USO DE COLAS DE MENSAJES

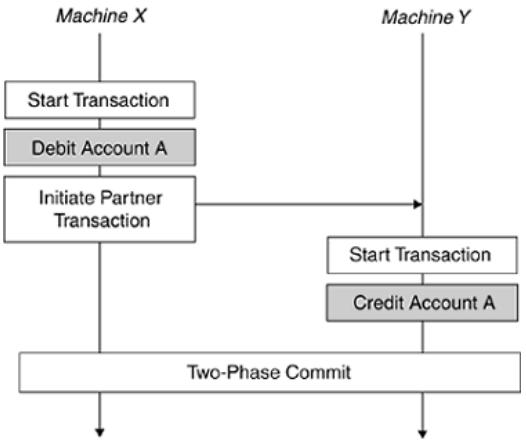
Transacción de revocación [reversal transaction]



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

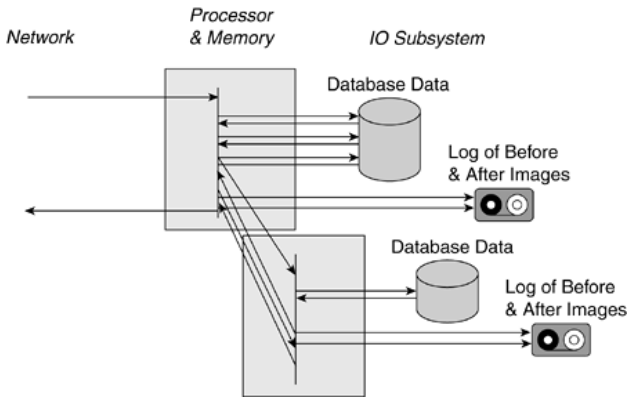
2-PHASE COMMIT



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

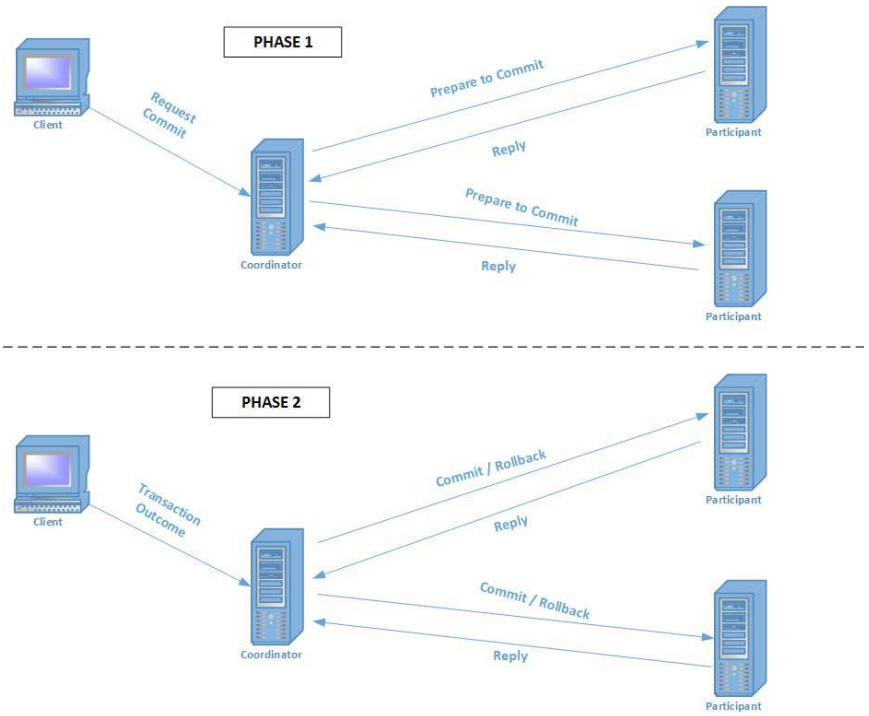
2-PHASE COMMIT



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

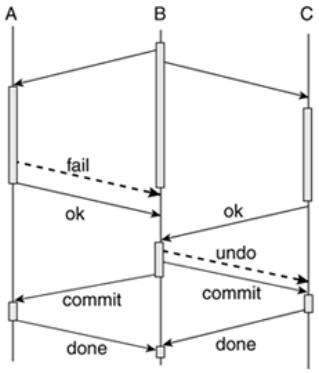
2-PHASE COMMIT



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

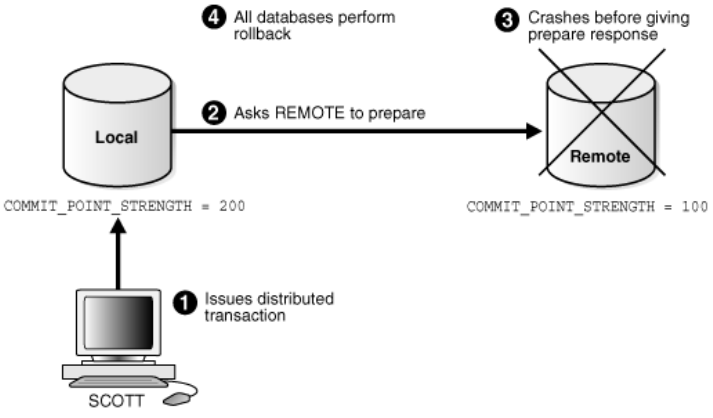
2-PHASE COMMIT



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

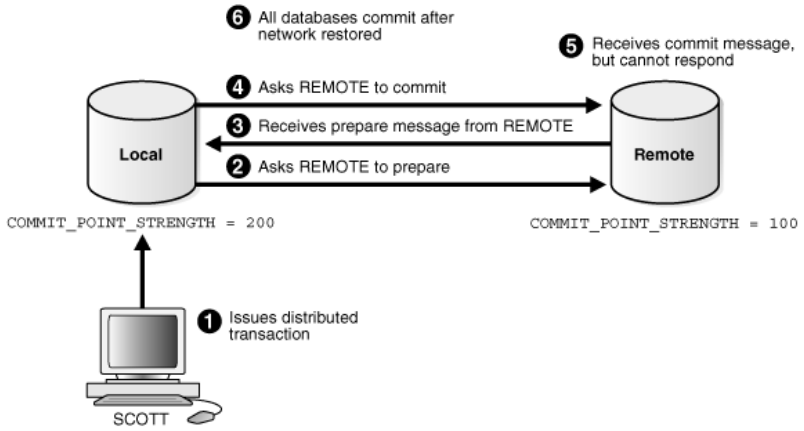


Fallo durante la fase de preparación

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT



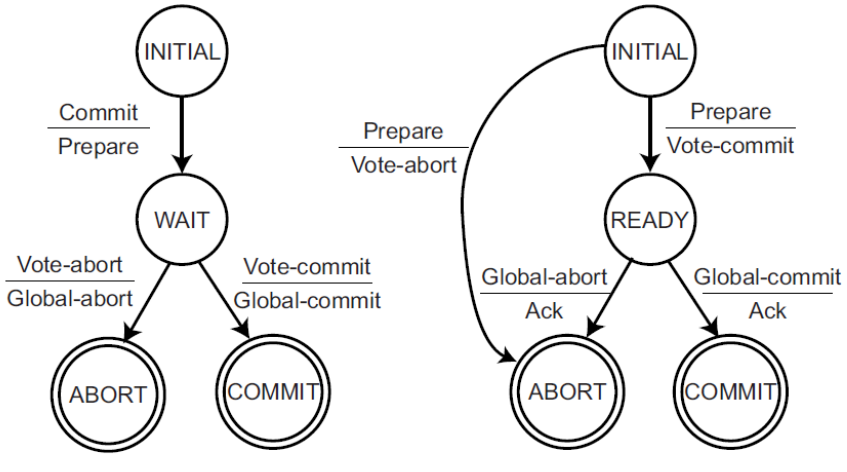
Fallo durante la fase de commit

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Protocolo



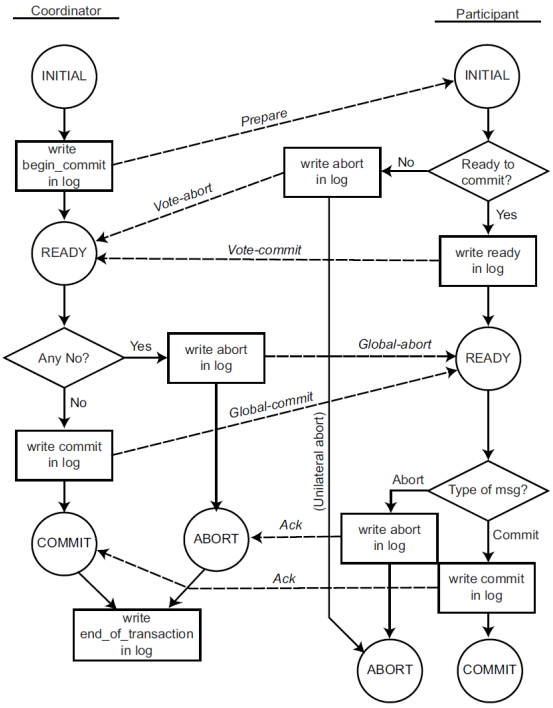
(a) Coordinator (b) Participants

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Protocolo

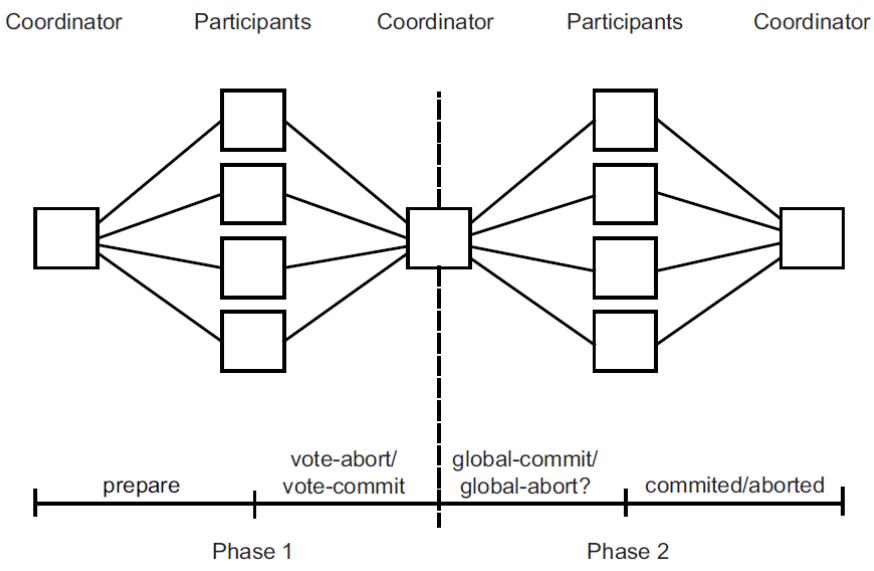


Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Protocolo centralizado

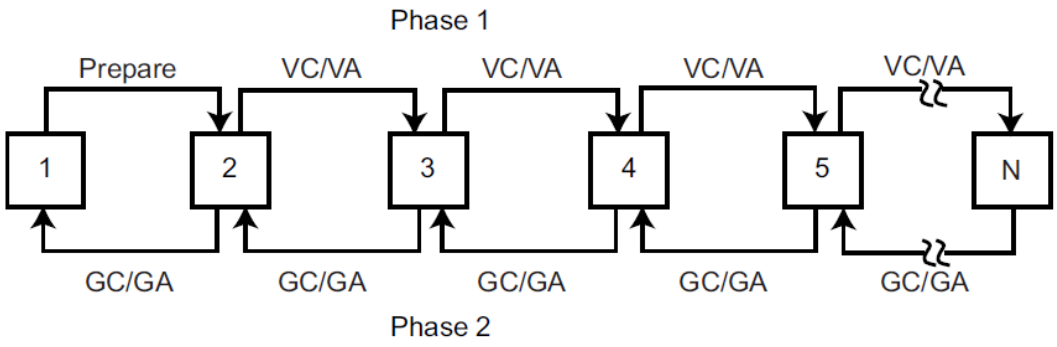


Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Protocolo lineal = Nested 2PC



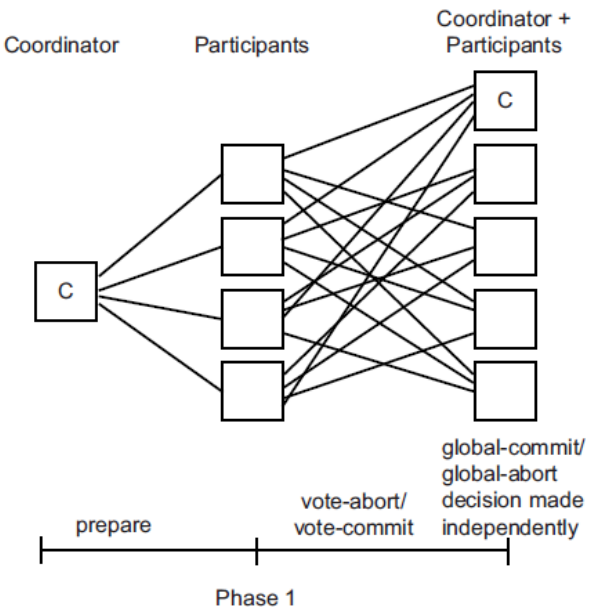
VC = Vote commit
VA = Vote abort
GC = Global commit
GA = Global abort

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Protocolo distribuido = Distributed 2PC



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

Optimizaciones para mejorar su rendimiento…

- **Presumed abort 2PC**
- **Presumed commit 2PC**

… reducen el número de mensajes transmitidos.

… reducen el número de escrituras en el log.

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

2-PHASE COMMIT

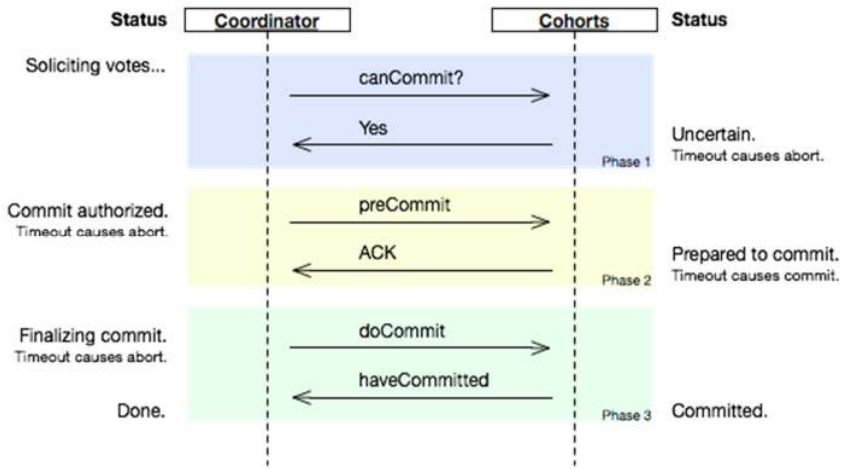
Inconveniente

¿Qué sucede si el coordinador falla de forma permanente después de que algunos participantes hayan pasado de la fase de preparación/votación a la fase de commit?

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

3-PHASE COMMIT



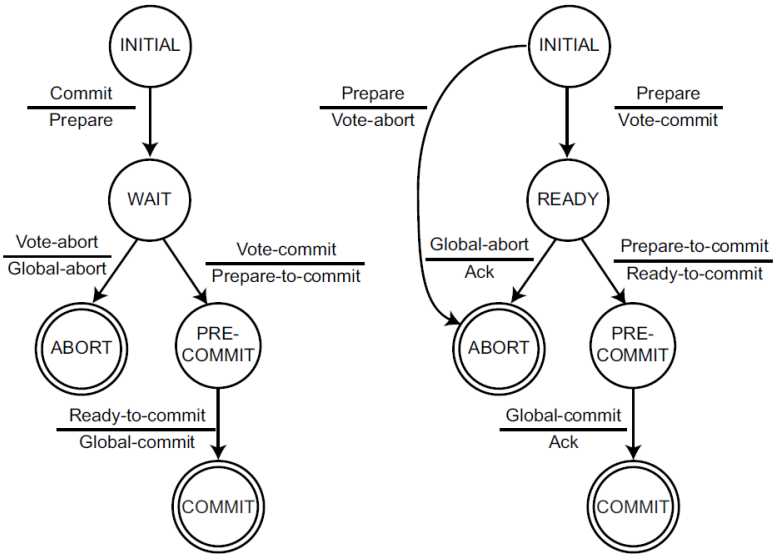
https://en.wikipedia.org/wiki/Three-phase_commit_protocol

Procesamiento de transacciones

Procesamiento de transacciones distribuidas

3-PHASE COMMIT

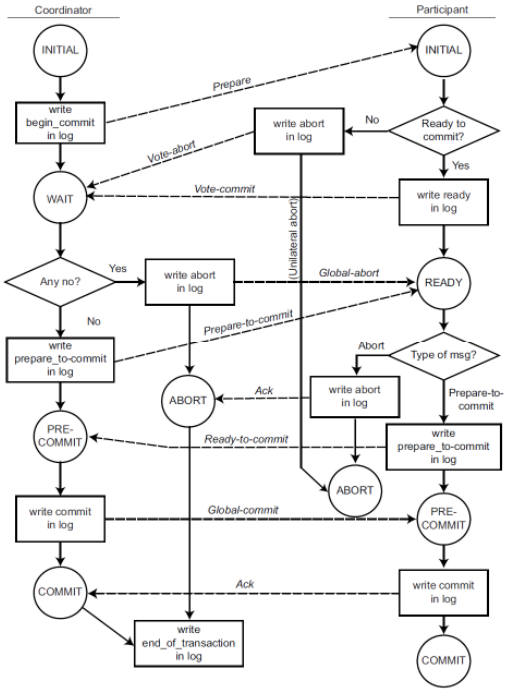
Protocolo



Procesamiento de transacciones

Procesamiento de transacciones distribuidas

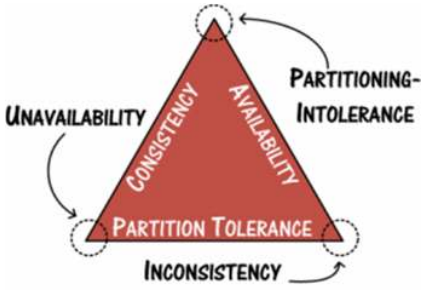
3-PHASE COMMIT Protocolo



El teorema CAP

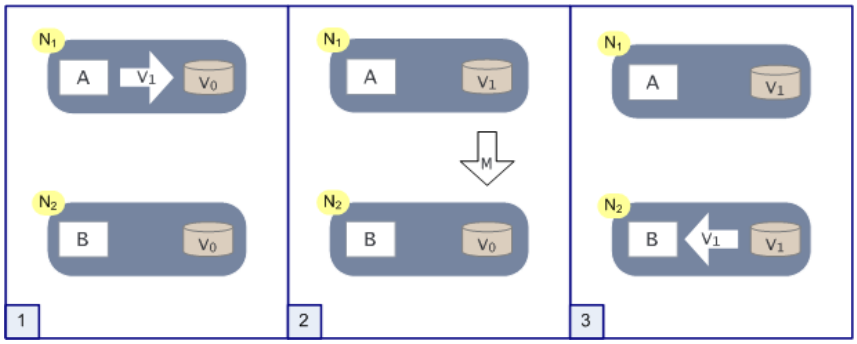
Tres requisitos de las aplicaciones distribuidas:

- Consistency [consistencia].
- Availability [disponibilidad].
- Partition Tolerance [tolerancia a particiones].



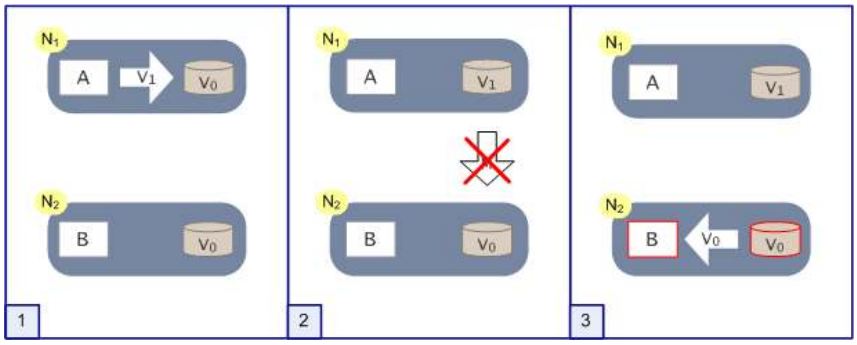
El teorema CAP

ESCENARIO BASE



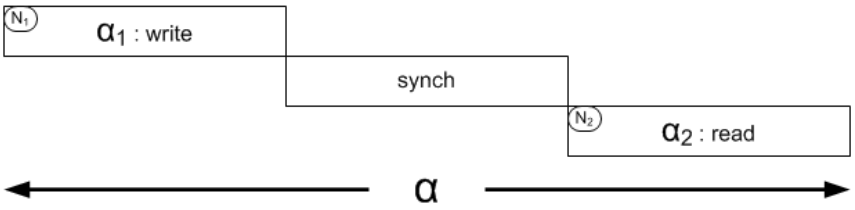
El teorema CAP

PARTICIÓN DE LA RED



El teorema CAP

Desde el punto de vista transaccional…



El teorema CAP

RESULTADO

- **CA (no P):** Se elimina la posibilidad de que la red se parta, lo que puede limitar la escalabilidad del sistema (p.ej. todo en una sola máquina), o bien…
- **CP (no A):** Se limita la disponibilidad (mientras la red esté partida, los servicios tendrán que esperar hasta garantizar la consistencia de los datos), o bien…
- **AP (no C):** Se admite la posibilidad de que existan inconsistencias en los datos ➔ **BASE**

El teorema CAP

BASE

Basically **A**vailable
Soft state
Eventually Consistent

ACID

Atomicity
Consistency
Isolation
Durability

Bibliografía recomendada

- M. Tamer Özsu & Patrick Valduriez:
  Principles of Distributed Database Systems.
  Springer, 3rd edition, 2011.
  ISBN 1441988335

  Chapter 10
  Transaction management

  Chapter 11
  Distributed concurrency control

  Chapter 12
  Distributed DBMS reliability