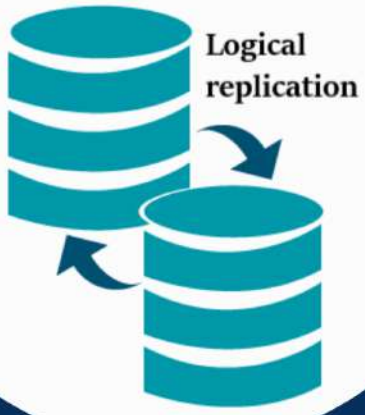
Protocolos de replicación Sección 13.3



Temas e introducción

Los protocolos de replicación gestionan actualizaciones(cambios) en bases de datos distribuidas. Se clasifican según: **método de actualización** (centralizado o distribuido) y **momento de propagación** (eager o lazy), tenemos 4 distintos tipos:

Eager Centralizado	• En este enfoque, un sitio maestro coordina las operaciones de actualización para garantizar consistencia fuerte.
Eager Distribuido	• Aquí, las actualizaciones se distribuyen entre múltiples sitios, y cada transacción debe coordinarse globalmente para que las réplicas permanezcan consistentes.
Lazy Centralizado	• Un único **sitio maestro** coordina las actualizaciones, pero las réplicas se actualizan **de forma diferida**. Esto reduce la carga inmediata en el sistema y mejora el rendimiento en transacciones de solo lectura
Lazy Distribuido	• Las actualizaciones se realizan localmente en varios sitios y se propagan de manera **diferida** a las réplicas. Esto minimiza la latencia y mejora la disponibilidad

Protocolos Eager Centralizados (13.3.1)

Concepto

Tiene un nodo central que realiza las operaciones de actualización de los datos y garantiza la consistencia. Todas las replicas se actualizan a la vez utilizando el protocolo de confirmación de dos fases (2PC), haciendo que las replicas sean mutuamente consistentes, asegurando un historial global serializable (1SR)

Parámetros claves?

Dónde se realizan las actualizaciones:

- **Maestro único:** Un solo maestro gestiona todas las actualizaciones.
- **Copia primaria:** Varios maestros, cada uno encargado de un grupo de datos.

Transparencia de replicación:

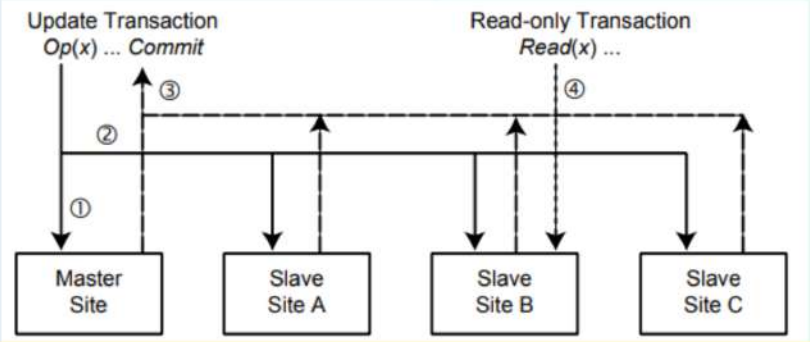
- **Limitada:** Las aplicaciones conocen la ubicación del maestro.
- **Completa:** El gestor de transacciones local determina el maestro.

Maestro único con Transparencia Limitada (13.3.1.1)

Las transacciones de actualización se envían al maestro, donde se aplican bloqueos y las operaciones se ejecutan sobre la copia maestra. Luego, las actualizaciones se propagan a las réplicas esclavas de manera síncrona o diferida, manteniendo el orden de ejecución.

Transacciones de solo lectura: Estas pueden ejecutarse en cualquier réplica esclava, con sincronización opcional al maestro para reducir la carga.

Puntos clave: Este protocolo es simple pero requiere identificar si una transacción es de solo lectura o de actualización, algo que puede definirse al inicio de la transacción.



Maestro único con Transparencia Completa (13.3.1.2)

Este enfoque busca evitar que las aplicaciones tengan que conocer el sitio maestro y reduce parcialmente la sobrecarga del maestro al distribuir ciertas tareas al gestor de transacciones (TM) local donde se ejecuta la aplicación.

Algorithm 13.1: Eager Single Master Modifications to C2PL-TM

```
begin
    ⋮
    if lock request granted then
        if op.Type = W then
            S ← set of all sites that are slaves for the data item
        else
            S ← any one site which has a copy of data item
        DP_S(op)                          {send operation to all sites in set S}
    else
        inform user about the termination of transaction
    ⋮
end
```

Algorithm 13.2: Eager Single Master Modifications to C2PL-LM

```
begin
    ⋮
    switch op.Type do
        case R or W                       {lock request; see if it can be granted}
            find the lock unit lu such that op.arg ⊆ lu ;
            if lu is unlocked or lock mode of lu is compatible with op.Type
            then
                set lock on lu in appropriate mode on behalf of transaction
                op.tid ;
                if op.Type = W then
                    DP_M(op)  {call local DP (M for "master") with operation}
                send "Lock granted" to coordinating TM of transaction
            else
                put op on a queue for lu
    ⋮
end
```

Transferencia de Operaciones vs. Transferencia de Estado:

- **Transferencia de Operaciones:** Las actualizaciones (como expresiones SQL) se envían a los esclavos para ejecutarlas.
- **Transferencia de Estado:** El resultado de la operación en el maestro se envía a los esclavos, simplificando la actualización.

Funcionamiento:

Transparencia Completa:

- Las aplicaciones envían transacciones al TM local.
- El TM local puede actuar como coordinador de las transacciones (lectura o actualización), proporcionando transparencia completa.

Opciones de Implementación:

- **Enfoque 1:** El TM local simplemente reenvía operaciones al sitio maestro, quien ejecuta las operaciones y devuelve los resultados. Este enfoque es simple, pero no resuelve el problema de sobrecarga del maestro.
- **Enfoque 2 (mejorado):** El TM local coordina las transacciones.

  Operaciones de lectura: El TM local obtiene un bloqueo de lectura del maestro y permite que la lectura se realice en un esclavo.

  Operaciones de escritura:
  El maestro realiza la actualización en su copia.

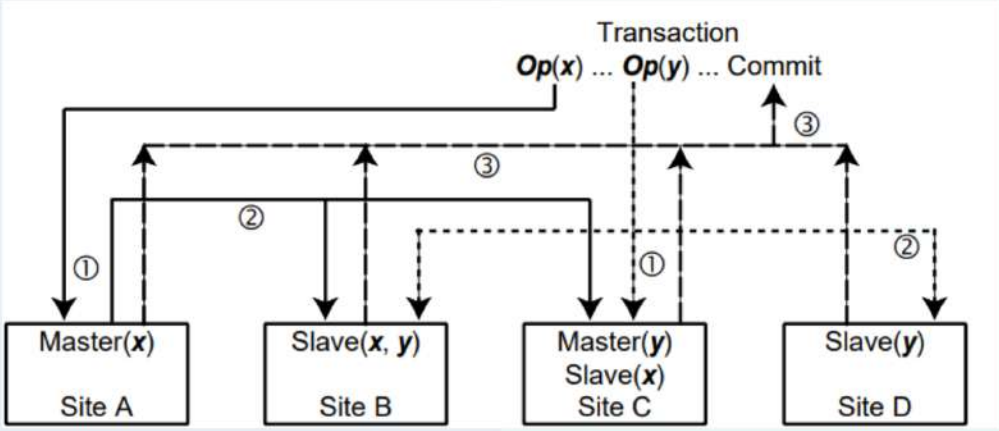
  El TM local propaga las actualizaciones a los esclavos para aplicarlas localmente.

Copia primaria con Transparencia Completa de Replicación (13.3.1.3)

En este enfoque, cada dato replicado tiene una copia designada como **copia primaria**, eliminando la necesidad de un único maestro para todos los datos. Esto distribuye la carga entre los sitios, pero introduce mayor complejidad al manejar el orden de serialización global.

Funcionamiento:
Para bases de datos **totalmente replicadas**, cualquier réplica puede ser la copia primaria.

Para bases de datos parcialmente replicadas, tiene sentido solo si todas las actualizaciones en una transacción afectan datos cuya copia primaria reside en el mismo sitio.



Protocolos Eager Distribuidos (13.3.2)

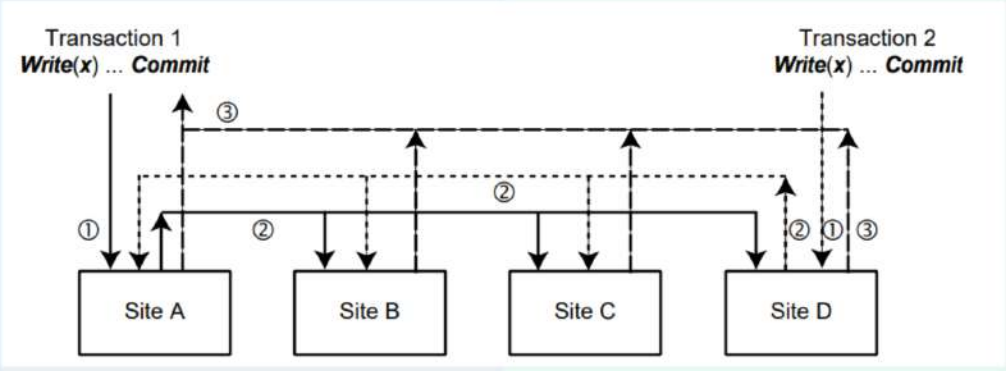
En los protocolos **Eager Distribuidos**, las actualizaciones pueden iniciarse en cualquier sitio y primero se aplican a la réplica local. Luego, estas actualizaciones se **propagan a las demás réplicas** dentro de los límites de la transacción. Al confirmar la transacción, las actualizaciones se hacen permanentes.

- **Lecturas y escrituras:**
  - **Lecturas:** Se pueden realizar en cualquier réplica.
  - **Escrituras:** Deben realizarse en todas las réplicas para mantener la consistencia.

Funcionamiento:

Origen de las actualizaciones:
- Si una réplica local existe en el sitio donde inicia la actualización, esta se aplica directamente.
- Si no existe, la actualización se reenvía a un sitio que posea una réplica del dato, el cual coordina la ejecución.

Propagación de actualizaciones:
- Las actualizaciones se propagan a todas las réplicas dentro de los límites de la transacción, utilizando protocolos como ROWA (Read-One-Write-All).



Protocolos Lazy Centralizados (13.3.3)

Los protocolos de replicación Lazy Centralizados son similares a los Eager Centralizados, pero con una diferencia clave: las actualizaciones primero se aplican a la réplica maestra, y su propagación a las réplicas esclavas se realiza después de confirmar la transacción, como una transacción de refresco independiente.

- **Aplicaciones:**
  Este enfoque es adecuado para sistemas distribuidos donde:
  - Las lecturas son más frecuentes que las actualizaciones.
  - La consistencia eventual es aceptable en lugar de consistencia fuerte.

Funcionamiento:
Aplicación de actualizaciones:

- **Las actualizaciones se ejecutan en la réplica maestra.**
- Una vez que la transacción se confirma (commit), las actualizaciones se propagan a las réplicas esclavas.

- **Lecturas:**
  Las operaciones de lectura en las réplicas esclavas pueden acceder a datos desactualizados (stale), ya que las actualizaciones realizadas en el maestro aún no se han propagado.

Maestro Único con Transparencia Limitada (13.3.3.1)

En este enfoque, las transacciones de actualización se **ejecutan directamente en el maestro**, y las actualizaciones se **propagan a las réplicas esclavas después de confirmar la transacción como una transacción de refresco**. Esto puede causar que las réplicas esclavas lean datos desactualizados.

Lectura:

- **Las lecturas en las réplicas esclavas acceden a datos locales.**
- Estos datos pueden no estar actualizados si las transacciones de refresco aún no se han ejecutado.

Desafíos

Orden de ejecución:

- **En bases de datos parcialmente replicadas, las réplicas pueden recibir transacciones de refresco de múltiples maestros.**
- Se deben asegurar órdenes consistentes entre las réplicas para que los estados converjan.

Uso de un grafo de replicación:

- Este grafo registra relaciones entre transacciones y sitios para detectar ciclos que puedan causar inconsistencias.
- Si un ciclo es detectado, la transacción actual se aborta y se reinicia más tarde para preservar historiales globalmente serializables (1SR).

Funcionamiento:

Actualización:

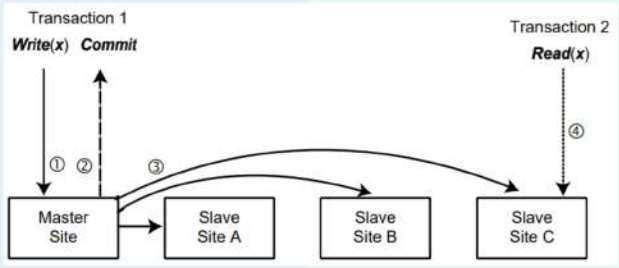
- **Una transacción de actualización se ejecuta primero en la réplica maestra.**
- Al confirmarse, se envía una **transacción de refresco** a las réplicas esclavas.

- **Escritura en esclavos:**
  Las escrituras en réplicas esclavas son rechazadas, ya que deben realizarse en el maestro.
-
- **Propagación ordenada:**
  - Las transacciones de refresco deben ejecutarse en el mismo orden en todas las réplicas para mantener la consistencia global. Esto se logra con marcas de tiempo asignadas por el maestro durante la confirmación de la transacción.

Conflictos y reconciliación:

- Si las transacciones de refresco llegan fuera de orden, pueden causar conflictos.
- Soluciones propuestas:
- **Compensación:** Ejecutar transacciones que reviertan los efectos de las inconsistencias.
- **Reconstrucción del estado:** Reconciliar los datos manualmente o con algoritmos específicos.

Mantenimiento distribuido:
Si el grafo se mantiene centralizado, el algoritmo pierde su naturaleza distribuida. Se requieren mecanismos distribuidos para su construcción y mantenimiento.



Maestro Único o Copia Primaria con Transparencia Completa (13.3.3.2)

Este enfoque permite que tanto las transacciones de lectura como de escritura se inicien en cualquier sitio, y estas operaciones se reenvían al maestro único o al sitio primario correspondiente. Aunque proporciona transparencia completa, introduce dos problemas principales:

- Historias no serializables globalmente (no-1SR).
- Transacciones que no ven sus propias escrituras.

Soluciones Propuestas

Pruebas de Validez en el Maestro (Bernstein et al., 2006):

- Se asigna un timestamp a cada transacción al momento de confirmar en el maestro.
- Reglas para el timestamp:

Debe ser mayor a los timestamps emitidos anteriormente.

Debe ser menor que los timestamps de las últimas modificaciones de los datos accedidos.

- Si no se puede cumplir esta regla, la transacción es abortada.
- **Ventaja:** Garantiza que las operaciones de lectura accedan a los valores correctos.

Listas de Actualizaciones:

- El maestro mantiene un listado de todas las actualizaciones realizadas por una transacción.
- Al ejecutar una lectura, esta consulta la lista para incluir los valores actualizados por la misma transacción.
- Limitación: Todas las lecturas y escrituras deben ejecutarse en el maestro, eliminando la ventaja de transparencia completa.

Problemas Identificados

Historias No-1SR (Ejemplo 13.4):

- Escenario: Dos transacciones (T1 y T2) se ejecutan en diferentes sitios (esclavo y maestro, respectivamente).
- Secuencia: Las operaciones de T1 y T2 se ejecutan y se propagan de manera desordenada entre el maestro y el esclavo.
- Resultado: Los historiales generados en los sitios no son serializables globalmente, ya que las operaciones no respetan un orden coherente en todos los sitios.

Transacciones que no ven sus propias escrituras (Ejemplo 13.5):

- Escenario: Una transacción (T3) escribe en un dato x y luego intenta leerlo desde el mismo sitio esclavo.
- Problema: Al leer x en el esclavo, la transacción obtiene un valor desactualizado porque la actualización aún no se ha propagado desde el maestro.
- Resultado: La transacción no puede ver el valor actualizado de su propia escritura.

Consideraciones y Limitaciones

- Transparencia Completa:

Es deseable, pero los problemas de consistencia y serialización dificultan su implementación práctica en protocolos lazy.

- Complejidad:

Mecanismos como el uso de timestamps o listas de actualizaciones aumentan la carga computacional en el maestro.

- Conflictos:

El manejo de inconsistencias (por ejemplo, abortar transacciones) es esencial para mantener un historial global serializable.

Protocolos Lazy Distribuidos (13.3.4)

En los protocolos de replicación Lazy Distribuidos, las actualizaciones pueden realizarse en cualquier réplica, y se propagan a otras réplicas de manera diferida mediante transacciones de refresco. Este enfoque es el más complejo debido a la concurrencia y los conflictos que surgen entre actualizaciones realizadas en diferentes réplicas.

Funcionamiento:

En el sitio local:

- Las operaciones de lectura (Read) y escritura (Write) se realizan en la copia local.
- La transacción se confirma (commit) localmente.
- Las actualizaciones se propagan posteriormente a otras réplicas mediante transacciones de refresco.

En los otros sitios:

- Las transacciones de refresco se procesan utilizando el mecanismo local de control de concurrencia.
-
- Se debe garantizar un orden adecuado para estas actualizaciones, lo que introduce complicaciones debido a los posibles conflictos entre actualizaciones concurrentes realizadas en diferentes sitios.

Desafíos

Conflictos de Actualización:

- Múltiples transacciones pueden actualizar diferentes copias del mismo dato en sitios distintos, generando conflictos.
- Estas diferencias deben reconciliarse antes de aplicar las actualizaciones.

Reconciliación:

- Es el proceso crítico para resolver conflictos y determinar el orden de ejecución de las transacciones de refresco.
- Ejemplo de estrategias de reconciliación:

Orden por timestamps: Se da preferencia a actualizaciones con marcas de tiempo más recientes.

Prioridad por sitio: Se priorizan actualizaciones de ciertos sitios más "importantes".

- Sin embargo, estas estrategias son ad hoc y dependen de los semánticos de la aplicación.

Sincronización de Relojes:

- Los métodos basados en timestamps requieren relojes sincronizados en todos los sitios, lo cual es difícil de lograr en sistemas distribuidos grandes.
- Si los relojes no están sincronizados, las decisiones de orden pueden ser arbitrarias y no reflejar la lógica de la aplicación.

Pérdida de Actualizaciones:

- Como resultado del proceso de reconciliación, algunas actualizaciones pueden ser descartadas.