Sistemas Distribuidos
Módulo 6

Consistencia, Replicación y Memoria Compartida Distribuida

Razones para la Replicación

Hay dos razones principales para la replicación de datos:

- Confiabilidad
  - Continuidad de trabajo ante caída de la réplica
  - Mayor cantidad de copias mejor protección contra la corrupción de datos
- Rendimiento
  - El SD escala en número
  - Escala en área geográfica (disminuye el tiempo de acceso al dato)
  - Consulta simultánea de los mismos datos

Precio a pagar por la replicación de datos:
Problemas de Consistencia

Replicación como Técnica de Escalabilidad

En general lograr escalabilidad va en detrimento del rendimiento.

Como técnicas para facilitar la escalabilidad se utiliza la replicación y el caching.

Ubicar copias de datos u objetos cercanos a los procesos que los usan mejora el rendimiento por la reducción del tiempo de acceso y resuelve el problema de escalabilidad.

Problemas:

La actualización de las réplicas consume más ancho de banda de la red.

Mantener múltiples copias consistentes resulta a su vez un serio problema de escalabilidad y mas en un contexto de consistencia estricta.

La idea es que la actualización se realice con una única operación atómica. Se necesitan sincronizar todas las réplicas.

Dilema: Por un lado la replicación tiende a resolver el problema de la escalabilidad (aumenta el rendimiento); por otro mantener consistentes las copias requiere sincronización global. La cura puede ser peor que la enfermedad.

Modelos de Consistencia

Un modelo de consistencia es esencialmente un contrato entre procesos y el almacenamiento de datos.

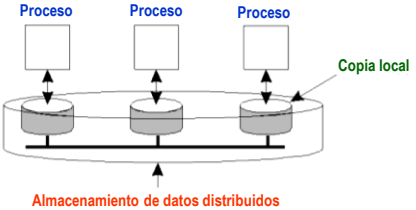
Es decir: si los procesos acuerdan obedecer ciertas reglas, el almacenamiento promete trabajar correctamente.

Normalmente un proceso que realiza una operación de lectura espera que esa operación devuelva un valor que refleje el resultado de la última operación de escritura sobre el dato.

Los modelos de consistencia se presentan divididos en dos conjuntos:

- Modelos de consistencia centrados en los datos.
- Modelos de consistencia centrados en el cliente.

Modelos de Consistencia Centrados en Datos



- Organización general de un almacenamiento lógico de datos, fisicamente distribuidos y replicados a través de múltiples procesos.

Consistencia Estricta

El modelo de consistencia más restrictivo es llamado consistencia estricta y es definido por la siguiente condición:

Cualquier lectura sobre un item de dato *x* retorna un valor correspondiente con la más reciente escritura sobre *x*

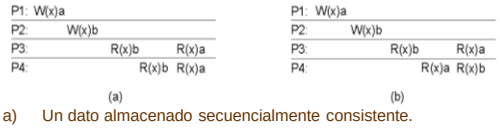
a) Un almacenamiento estrictamente consistente.
b) Un almacenamiento que no es estrictamente consistente.

La definición supone un *tiempo global absoluto*

Consistencia Secuencial

La consistencia secuencial es una forma ligeramente más débil de la consistencia estricta. Satisface la siguiente condición:

El resultado de una ejecución es el mismo si las operaciones (lectura y escritura) de todos los procesos sobre el dato fueron ejecutadas en algún orden secuencial y las operaciones de cada proceso individual aparecen en esta secuencia en el orden especificado por su programa.



a) Un dato almacenado secuencialmente consistente.
b) Un dato almacenado que no es secuencialmente consistente.

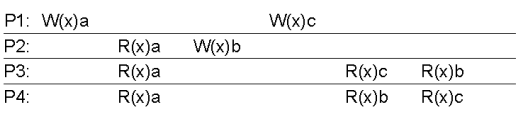
Consistencia Causal

El modelo de consistencia causal (Hutto and Ahamad, 1990) es un debilitamiento de la consistencia secuencial. Se hace una diferenciación entre eventos que están potencialmente relacionados en forma causal y aquellos que no. las operaciones que no están causalmente relacionadas se dicen concurrentes.

La condición a cumplir para que un datos sean causalmente consistentes es:

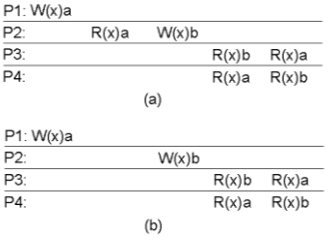
Escrituras que están potencialmente relacionadas en forma causal deben ser vistas por todos los procesos en el mismo orden. Escrituras concurrentes pueden ser vistas en un orden diferente sobre diferentes máquinas.

Consistencia Causal (cont)



- Esta secuencia es permitida con un almacenamiento causalmente consistente, pero no con un almacenamiento secuencialmente consistente o con un almacenamiento consistente en forma estricta.

Consistencia Causal (cont)



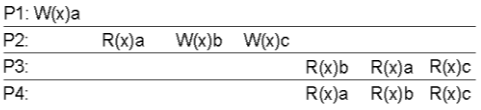
a) Una violación de una consistencia causal.
b) Una correcta secuencia de eventos en una consistencia causal.

Consistencia First Input First Output

- Condición Necesaria :

Escrituras realizadas por un proceso único son vistas por los otros procesos en el orden en que son hechas, pero escrituras desde diferentes procesos pueden ser vistas en diferente orden por diferentes procesos.

Consistencia First Input First Output (cont)



- Una secuencia válida de eventos de una consistencia First Input First Output.

Consistencias atendidas por el programador

En los siguientes modelos de consistencia, que solo se enumeran, es necesaria la intervención del programador para mantener los protocolos respectivos:

- Consistencia Débil
- Consistencia Relajada
- Consistencia Entry

Referencias:

[1] Distributed Operating Systems, Tanenbaum, items 6.3.5, 6.3.6 y 6.3.7

[2] Distributed Systems: Principles and Paradigms, 1ra ed, Tanenbaum et ál, items 6.2.5, 6.2.6 y 6.2.7

[3] Distributed Operating Systems:Concepts and Design, Sinha, sec 5.6

Sumario de Modelos de Consistencia

Consistencia	Descripción
Estricta	Ordenamiento en tiempo absoluto de todos los accesos compartidos.
Linealizabilidad	Todos los procesos deben ver todos los accesos compartidos en el mismo orden. Los accesos son ordenados de acuerdo a una estampilla de tiempo global.
Secuencial	Todos los procesos ven todos los accesos compartidos en el mismo orden. Los accesos no están ordenados en el tiempo.
Causal	Todos los procesos ven los accesos compartidos causalmente relacionados en el mismo orden.
FIFO	Todos los procesos ven las escrituras de cada uno en el orden en que fueron hechas. Escrituras de diferentes procesos pueden no ser vistas en el mismo orden.

(a) Modelos de consistencia que no usan operaciones de sincronización

Consistencia	Descripción
Débil	Datos compartidos pueden ser contados como consistentes luego de que se haya hecho la sincronización.
Relajada	Datos compartidos son hechos consistentes cuando la región crítica es abandonada.
Entry	Datos compartidos pertenecientes a una región crítica son hechos consistentes cuando se entra a la región crítica.

(b) Modelos con operaciones de sincronización

Modelos de Consistencia Centrados en el Cliente

Este tipo de modelos trata una clase especial de almacenamiento de datos distribuidos.

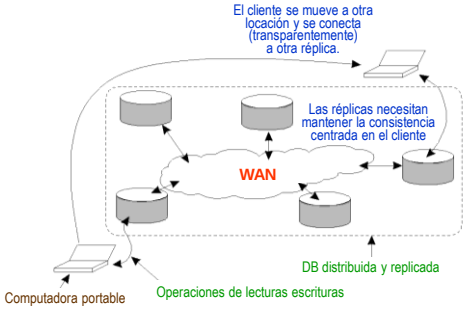
Los almacenamiento de datos referidos están caracterizados por una falta de actualizaciones simultáneas, o cuando dichas actualizaciones ocurren, pueden ser fácilmente resueltas.

La mayoría de las operaciones son de lectura.

La introducción de modelos de consistencia centrados en el cliente permiten esconder muchas inconsistencias de manera relativamente fácil.

En esencia la consistencia centrada en el cliente provee garantías *para un único cliente* concerniente a la consistencia de accesos a los datos de ese cliente. No se dan garantías para accesos concurrentes por diferentes clientes.

Consistencia Eventual



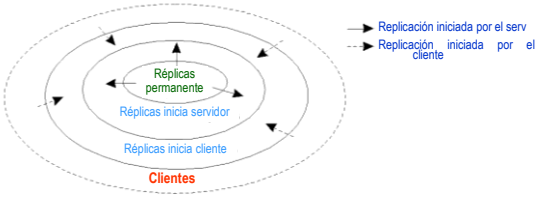
Consistencias orientadas al cliente

- Lecturas Monotónicas
- Escrituras Monotónicas
- Lea sus Escrituras
- Escrituras seguidas de Lecturas

Referencias:

[1] Distributed Systems: Principles and Paradigms, 1ra ed, Tanenbaum et ál, items 6.3.2, 6.3.3, 6.3.4 y 6.3.5

Ubicación de Réplicas



La organización lógica de diferentes clases de copias de datos almacenados en tres anillos concéntricos

Protocolos de Consistencia

Un *protocolo de consistencia* describe una implementación de un modelo específico de consistencia.

Lejos, los modelos de consistencia en los cuales las operaciones están globalmente seriadas son los modelos más importantes y los más ampliamente aplicados.

Estos modelo incluyen consistencia secuencial, consistencia débil con variables de sincronización y transacciones atómicas.

Los protocolos pueden ser:

- Protocolos basados en el primario
- Protocolos de escritura replicada

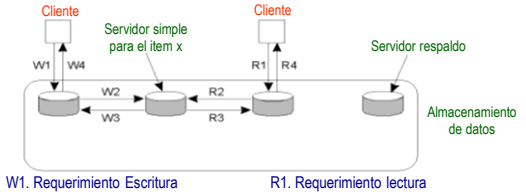
Protocolos Basados en el Primario

Cada item de dato x tiene en el almacenamiento de datos un primario asociado, el cual es responsable de coordinar las operaciones de escritura sobre x.

Se debe hacer una distinción si el primario está fijo en un servidor remoto o si las operaciones de escritura pueden ser llevados localmente luego de mover el primario a donde reside el proceso que inició la operación de escritura.

Protocolos Escritura Remota (1)

Protocolo de escritura remota basado en primario con servidor fijo al cual las lecturas y escrituras son encaminadas.

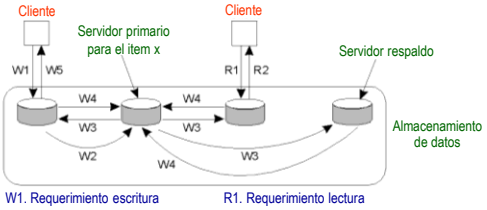


W1. Requerimiento Escritura
W2. Req. encaminado al serv. por x
W3. ACK escritura completa
W4. ACK escritura completa

R1. Requerimiento lectura
R2. Req. encaminado al serv. por x
R3. Retorno respuesta
R4. Retorno. respuesta

Protocolos Escritura Remota (2)

El principio de protocolo de respaldo primario.

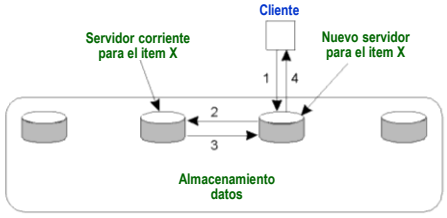


W1. Requerimiento escritura
W2. Req. encaminado al primario
W3. Avisa a respaldos actualizar
W4. ACK actualización
W5. ACK escritura completada

R1. Requerimiento lectura
R2. Respuesta a lectura

Protocolos de Escrituras Locales (1)

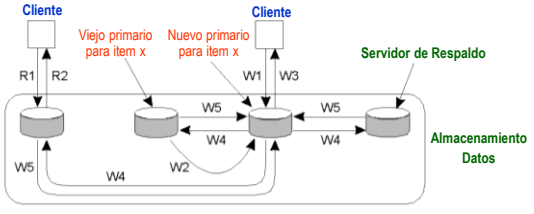
Protocolo escritura local basado en primario en el cual una única copia es migrada entre procesos.



1.Solicitud de lectura o escritura.
2.Continua solicitud al servidor corriente de x.
3.Se mueve el item x al servidor del cliente.
4.Retorna el resultado de la operación sobre el servidor del cliente.

Protocolos de Escritura Local (2)

Protocolo de respaldo primario en el cual el primario migra hacia el proceso que espera realizar una actualización.



W1.Solicitud escritura
W2.Mueve item x a un nuevo primario
W3.ACK escritura completada
W4.Actualiza los respaldos
W5.ACK actualización

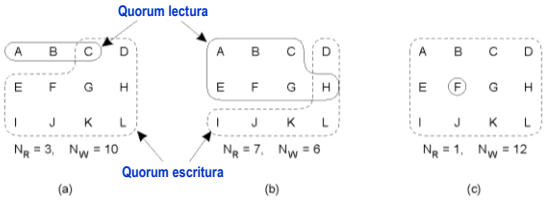
R1. Solicitud de lectura
R2. Respuesta a la lectura

Protocolos de Escritura Replicada

En los protocolos de escrituras replicadas, las operaciones pueden ser llevadas a cabo sobre múltiples réplicas en vez de una sola como en el caso de las réplicas basadas en el primario.

- Replicación Activa
- Protocolos basados en quorum.

Protocolos Basados en Quorum



Tres ejemplos del algoritmo de votación:

a) Una correcta elección de conjuntos de lectura y escritura.
b) Una elección que puede llevar a conflictos escritura-escritura.
c) Una elección correcta, conocida como ROWA (read one, write all)

Memoria Compartida Distribuida

Comunicación entre procesos:

- Pasaje de Mensajes
- Memoria Compartida

Memoria Compartida Distribuida (MCD):

La capa abstracta se implementa en:

- Kernel
- Rutinas en tiempo de corrida

Memoria Compartida Distribuida

La memoria se parte en “bloques” fijos.

El “caching” evita la *latencia de acceso*.

Puede soportar: migración y/o replicación

Memoria Compartida Distribuida

Diseño e Implementación de MCD

Aspectos que deben considerarse en el diseño de una MCD.

- Granularidad
- Estructura del espacio de memoria compartida
- Coherencia de memoria y sincronización de acceso
- Localización de datos y accesos
- Estrategias de reemplazo
- Thrashing
- Heterogeneidad

Memoria Compartida Distribuida

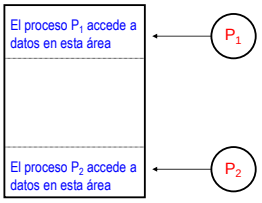
Granularidad

Problemas que se presentan con el tamaño del bloque

- Sobrecarga de paginado
- Tamaño de directorio
- Thrashing
- Falso compartir

Memoria Compartida Distribuida

Falso compartir



Memoria Compartida Distribuida

Ventajas de usar un tamaño de bloque igual a la página de memoria local:

⇒Permite el uso de sistemas existentes de falta de páginas

⇒Permite el control de derechos de acceso

⇒Si las páginas pueden colocarse en un *paquete* no impone sobrecarga en la red

⇒El tamaño es adecuado con respecto a la contención de memoria

Memoria Compartida Distribuida

Estructura del espacio de memoria compartida

La estructura define la abstracción de la vista a los programadores de aplicaciones en un sistema MCD.

Puede ser vista para unos como palabra y para otros como objetos.

La estructura y la granularidad está relacionadas.

Memoria Compartida Distribuida

No estructurada**:** es un arreglo de palabras.

Por tipo de datos: colección de objetos (Clouds y Orca) o colección variables en lenguaje fuente (Munin y Midway). La granularidad es la variable o el objeto.

Como base de datos: implica una memoria asociativa.

Memoria Compartida Distribuida

Ejemplo de Implementación

La implementación mas común es el modelo de *consistencia secuencial*, porque:

- ✓ Es realizable
- ✓ Soporta semántica mas intuitiva para la coherencia de memoria.
- ✓ No impone nada extra al programador.

Desventaja: baja concurrencia

La consistencia débil y de liberación con sus variables implícitas proveen mejor concurrencia y soportan una semántica mas intuitiva.

Memoria Compartida Distribuida

Implementación del modelo de consistencia secuencial para una memoria compartida distribuida

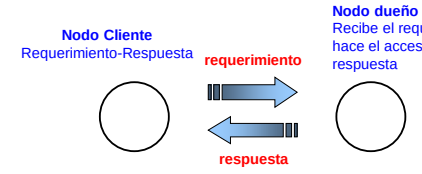
Protocolos: dependen de la estrategia elegida

Estrategias:

- ❑**No réplica, no migratoria (NRNMB)**
- ❑**No réplica, migratoria (NRMB)**
- ❑**Réplica, migratoria (RMB)**
- ❑**Réplica, no migratoria (RNMB)**

Memoria Compartida Distribuida

No réplica, no migratoria (NRNMB)



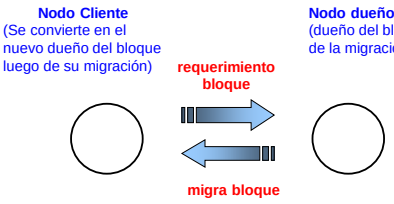
Desventajas:

✓La serialización de los requerimientos crea un *cuello de botella.*

✓No es posible paralelismo

Memoria Compartida Distribuida

No réplica, migratoria (NRMB)



Memoria Compartida Distribuida

Ventajas:

✓ No se incurre en costos de comunicación cuando un proceso accede a un dato local

✓ Permite tomar ventajas de la localidad de accesos a los datos

Desventajas:

✓ Problemas de *thrashing*

✓ No hay paralelismo

Memoria Compartida Distribuida

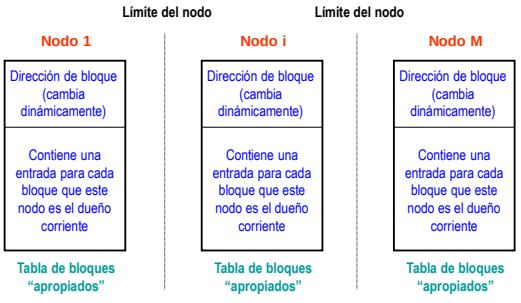
Localización de los bloques

1.Broadcasting

✓ No escala bien. El sistema de comunicaciones es *cuello de botella*.

✓ La latencia de la red es grande

Memoria Compartida Distribuida



Memoria Compartida Distribuida

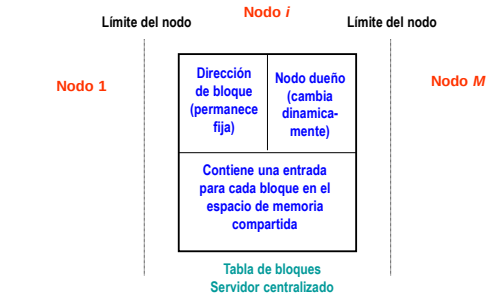
2.Algoritmo servidor centralizado

Todos los nodos conocen su dirección.

✓ Reduce paralelismo (serializa los requerimien-tos).

✓ Una falla hace caer el sistema.

Memoria Compartida Distribuida

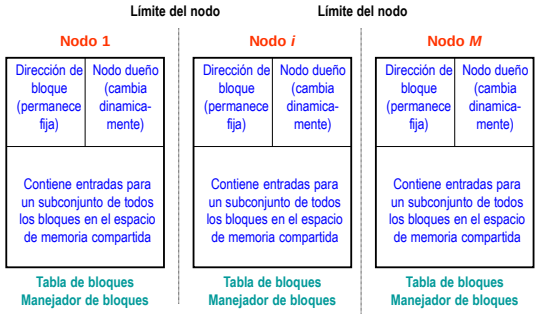


Memoria Compartida Distribuida

3. Algoritmo servidor distribuido fijo

- ✓ Distribuye el rol del servidor.
- ✓ Manejan bloques en varios nodos.
- ✓ Cada uno maneja un subconjunto determinado de bloques de datos.

Memoria Compartida Distribuida

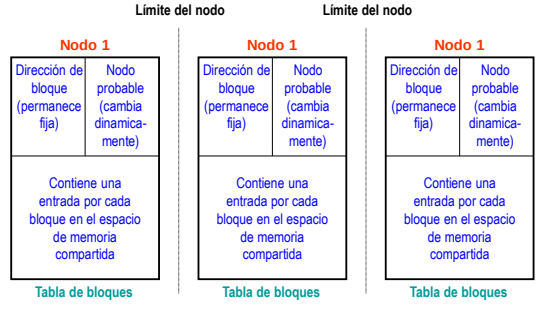


Memoria Compartida Distribuida

4. Algoritmo servidor distribuido dinámico

- ✓ Cada nodo tiene su propia tabla de información de los bloques de la MCD.
- ✓ No siempre la información de esta tabla es correcta.

Memoria Compartida Distribuida



Memoria Compartida Distribuida

Réplica, Migratorio (RMB)

Ataca la desventaja principal de las estrategias no replicantes: la reducción del paralelismo.

Se incrementa el costo de las operaciones de escritura.

Se debe mantener la **consistencia**

La replicación complica el protocolo de coherencia de memoria.

Memoria Compartida Distribuida

Protocolos:

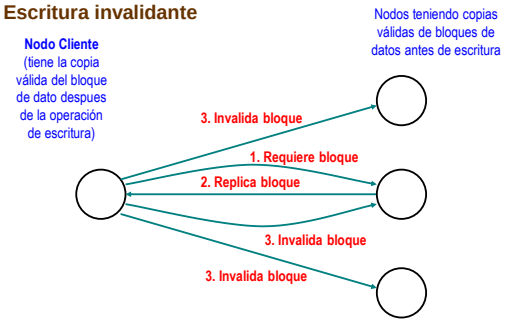
- Escritura invalidante
- Escritura actualizada

Este esquema requiere un ordenamiento total de las operaciones de escritura para lograr la consistencia secuencial.

Se utiliza un secuenciador global.

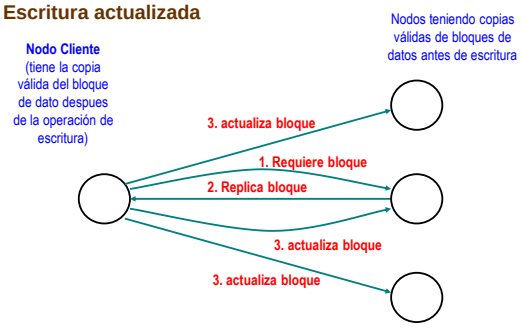
Memoria Compartida Distribuida

Escritura invalidante



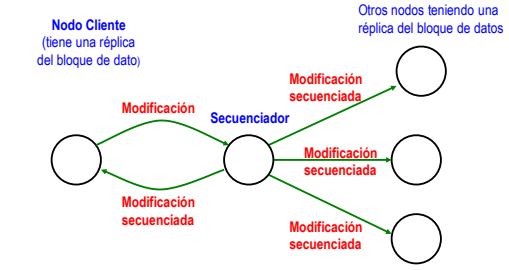
Memoria Compartida Distribuida

Escritura actualizada



Memoria Compartida Distribuida

Mecanismo secuenciador



Memoria Compartida Distribuida

Localización de datos en RMB

Necesita:

1. Localizar al dueño de un bloque.
2. Tener la pista de los nodos que corrientemente tienen una copia válida del bloque.

Algoritmos usados:

- Broadcasting.
- Algoritmo servidor centralizado.
- Algoritmo servidor distribuido fijo.
- Algoritmo servidor distribuido dinámico.

Memoria Compartida Distribuida

Para reducir la cadena de nodos a atravesar para alcanzar el verdadero dueño del bloque, la tabla de bloques del nodo es adaptada de la siguiente forma:

a) Siempre que el nodo recibe un requerimiento de invalidación.
b) Siempre que el nodo pasa la propiedad.
c) Siempre que el nodo pasa un requerimiento que falla.

Réplica, no migratoria (RNMB)

La ubicación de cada réplica es fija.

Siempre que se escribe se actualiza.

Localización de datos en RNMB

a) Las localizaciones de las réplicas nunca cambian.
b) Todas las réplicas se mantienen consistentes.
c) Sólo un requerimiento de lectura puede ser directamente enviado a uno de los nodos teniendo una réplica del bloque y todos los requerimientos de escritura deben primero ser enviados al secuenciador.

Estructura:

- ✓ Una tabla de bloques por cada nodo.
- ✓ Una tabla de secuencias en el secuenciador.

La tabla del secuenciador tiene los siguientes campos:

a) Dirección del bloque.
b) Conjunto de réplicas.
c) Número de secuencia

Estrategias de reemplazo

a) Qué bloque tiene que ser reemplazado.
b) Dónde debe ubicarse el bloque reemplazado.

¿Qué bloque tiene que ser reemplazado?

a) Usado vs. no usado (LRU).
b) Espacio fijo vs. espacio variable.

En algunos sistemas cada bloque es clasificado:

- No usado
- Nil (invalidado)
- Read-only
- Read-owned: RO y el nodo es dueño
- Escribible

De acuerdo a ésto, la prioridad de reemplazo sería:

1. No usado y Nil
2. Read-only
3. Read-owned
4. Escribible

Ubicación el bloque reemplazado

- ➢ Usar almacenamiento secundario.
- ➢ Usar memoria de otros nodos.

Thrashing

Cuando hay migración y se producen las siguientes situaciones:

- ➢ Datos entrelazados entre distintos nodos.
- ➢ Bloques con permiso RO son repetidamente invalidados inmediatamente que son replicados.

Implica poca **localidad**

Memoria Compartida Distribuida

Soluciones:

1. Proveer locks de aplicación controlada: fija el bloque por un tiempo.
2. No permitir que quiten, por un tiempo *t*, un bloque del nodo (*t* está dado por estadística o por accesos).
3. Ajustar el algoritmo de coherencia para usar modelos de datos compartidos.

Memoria Compartida Distribuida

Ventajas de la Memoria Compartida Distribuida

- Abstracción mas simple.
- Mejor portabilidad de programas de aplicación distribuidos.
- Mejor desempeño de algunas aplicaciones. Lo hace posible la *localidad de los datos*, el *movimiento de datos por demanda* y *espacio de direcciones mas grande*.
- Ambiente de comunicación mas flexible.
- Facilidad para la migración de procesos.