Sistemas Operativos Distribuidos

6

Memoria compartida distribuida

(DSM, *Distributed Shared Memory*)

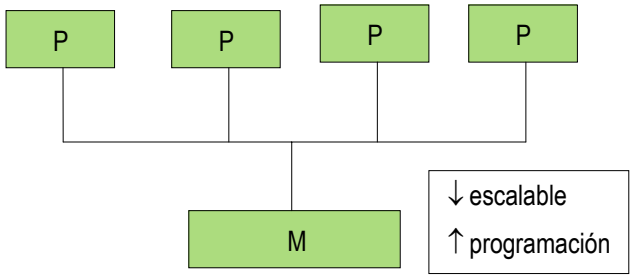
Índice

- Introducción
- Estrategias de implementación
  - Mediante memoria virtual (VM-DSM)
  - Por compilador y entorno en tiempo de ejecución (RT-DSM)
- Aspectos de diseño de DSM
  - Esquema de gestión y política de actualización de copias
  - Sincronización
  - Localización de copias
  - *False-sharing* y *thrashing*
- Modelos de coherencia de memoria

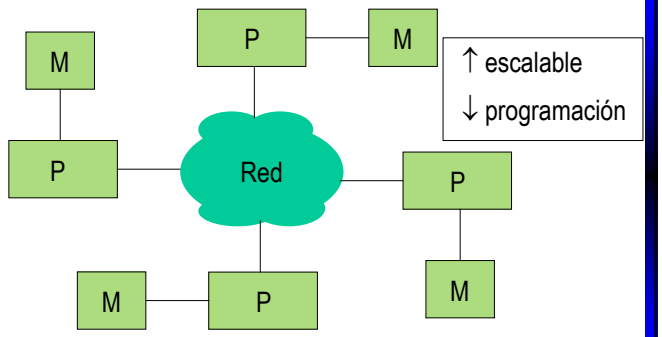
Preámbulo: clasificación de sis. paralelos

- 2 Factores:
  - mem. centralizada *versus* distribuida
  - mem. privada *versus* compartida
- Clasificación:
  - Mem. centralizada y compartida → multiprocesadores
  - Mem. distribuida y privada → sistemas distribuidos
  - Mem. distribuida y compartida → *Distributed Shared Memory* (DSM)
- *Distributed Shared Memory* (DSM)
  - Mediante hardware:
    - Multiprocesadores NUMA (acceso a memoria no uniforme)
  - Mediante software → Objetivo de la presentación (DSM Software)
    - Incluir software en SD para ofrecer DSM

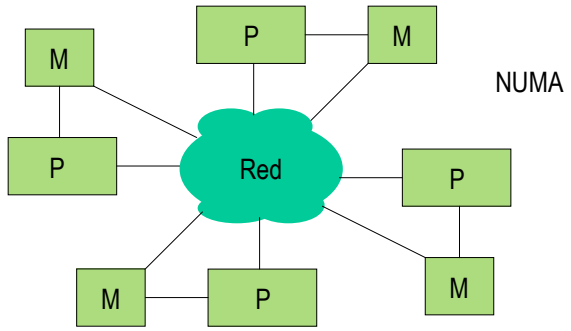
Memoria centralizada y compartida



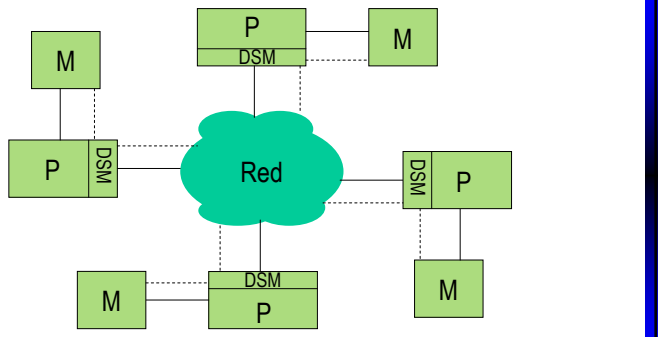
SD: Memoria distribuida y privada



DSM HW: Memoria distribuida y compartida



DSM SW: Memoria distribuida y compartida



Introducción (1/2)

- Multiprocesadores con mem. compartida vs. s. distribuidos:
  - HW más complejo y difícilmente ampliable
  - SW más sencillo y mejor conocido
- Modelo de programación en sistemas con mem. compartida
  - Llamadas a procedimiento (o invocación de métodos)
  - Comunicación mediante datos compartidos
  - Sincronización mediante semáforos o mecanismos equivalentes
- Modelo de programación tradicional en s. distribuidos
  - Paso de mensajes para comunicación y sincronización
- Querría ejecutar en SD aplicación paralela basada en m.comp.
- Nuevo modelo de programación en s. distribuidos
  - RPC (o RMI) + DSM

Introducción (2/2)

- Memoria compartida distribuida (DSM)
  - Memoria compartida implementada por software
  - Simulada mediante paso de mensajes
  - Comunicación con datos compartidos y sincronización con semáforos
- Objetivo:
  - Sistemas fáciles de construir y programar
  - Especialmente adecuada para aplicaciones paralelas
    - Programa concurrente con estructuras de datos compartidas
  - Limitada aplicación para cliente-servidor
    - Proveedor y consumidor de servicios no deberían compartir memoria
- Problemas:
  - Rendimiento aceptable requiere múltiples copias en distintos nodos
    - Problema de coherencia: Similar a multiprocesadores pero por SW
  - Difícil manejar heterogeneidad

Implementación VM-DSM

- Extensión de m. virtual que permite solicitar páginas remotas
  - Accesos a DSM convencionales (LOAD/STORE)
  - Fallo de página no local, se solicita a máquina remota
- Normalmente implementada en modo usuario
  - Se impide acceso a página no presente en máquina local (mprotect)
  - En tratamiento de excepción (SEGV), solicita página remota
  - Se "instala" la página en mapa de proceso con permiso de acceso
  - Proceso reanuda ejecución de forma transparente
- Características:
  - Dificulta tratamiento de heterogeneidad
  - No sobrecarga en accesos pero sí por tratamiento de fallos
  - *False sharing*
- Ejemplos:
  - Ivy, TreadMarks, Munin

Implementación RT-DSM

- Extensión del compilador + biblioteca DSM en t. ejecucíón
- Acceso dato compartido, compilador traduce en LOAD/STORE
  - Pero además en llamada a DSM
- Por cada dato compartido, el compilador asigna espacio extra:
  - Por ejemplo, bits de modificado y validez, nº de versión, *timestamp*, ...
- Características:
  - Facilita tratamiento de heterogeneidad
  - Sobrecarga en cada acceso
  - No *False sharing*
- Ejemplos:
  - Midway

Gestión y actualización de copias

- Alternativas:
  - *Write-invalidate*:
    - Actualización genera invalidación de copias
    - Múltiples copias de lectura (*read-replication*)
      - Múltiples lectores/único escritor
    - Ivy, Treadmarks, Munin (configurable)
  - *Write-update*:
    - Actualización se propaga a copias
    - Múltiples copias tanto de lectura como de escritura (*full-replication*)
      - Múltiples lectores/múltiples escritores
    - Midway, Munin (configurable)
  - En ambas opciones puede usarse *multicast*

Sincronización y localización de copias

- Sincronización
  - Aplicaciones usan mecanismos "tradicionales"
    - cerrojos, barreras, semáforos, ...
  - Factible pero ineficiente implementación basada en *TestAndSet*
    - Genera mucho tráfico de accesos e invalidaciones superfluas
  - Implementación basada directamente en mensajes explícitos
- Localización de copias
  - Mediante *broadcast*
    - Poco *escalable*
  - Gestor mantiene información sobre copias de todos los datos
    - "Cuello de botella", falta de *escalabilidad* y punto único de fallo
  - Varios gestores: cada uno guarda info. sobre subconjunto de datos
    - DSM basada en páginas: Reparto dependiendo de número de página
  - Manteniendo cadena de dueños probables

False-sharing y *thrashing*

- *False sharing*
  - Puede ocurrir si tamaño(unidad de transferencia) > tamaño(dato)
    - Sí en VM-DSM; No en RT-DSM
    - Página puede contener datos independientes que se escriben a la vez
    - Se puede reducir con asignación "inteligente" de datos
      - Solución extrema: dato ocupa nº entero de páginas (fragmentación)
  - Mayor tráfico en la red
  - *Write-invalidate* invalidaciones innecesarias
  - *Write-update* posible sobrescritura errónea de datos
  - Uso de "twins" y "diffs" para paliar problemas (TreadMarks, Munin)
- *Thrashing*
  - Procesos compitiendo por dato (realmente o por *false-sharing*)
  - Página "viaja" continuamente entre procesadores ("ping-pong")
    - Suele existir un intervalo mínimo de permanencia en un nodo

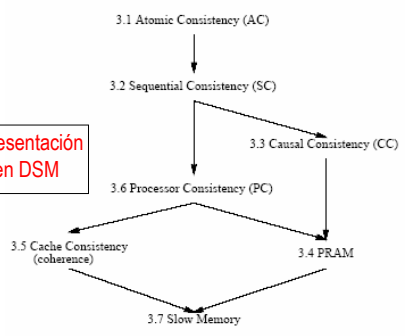
Modelo de coherencia de memoria

- Compromiso del s. memoria respecto a accesos de programa
  - Qué garantiza s. memoria compartida HW o SW a las aplicaciones
  - Qué valor devuelve cada acceso de lectura
- Obviamente, programador querría coherencia estricta:
  - Lectura devuelve valor de escritura "más reciente"
- Ineficiente incluso en sistemas de memoria compartida
  - Además, aunque HW de memoria la asegure hay otros factores:
    - Procesador y compilador reordenan instrucciones
    - Compilador guarda variables en registros
  - Modelo de memoria de Java no es estricto
- Modelo de memoria: compromiso coherencia y rendimiento
  - Garantías de coherencia menos estricta → mayor rendimiento
  - Permiten optimizaciones (p.e. retardar propagación de cambios)
  - Pero ofrecen modelo de programación menos intuitivo

Modelos de coherencia uniformes

- Coherencia secuencial (Lamport 1979):
  - "Resultado equivalente a que los accesos a memoria de procesadores se ejecutaran en algún orden secuencial apareciendo las operaciones de cada uno en el orden especificado por su programa".
  - Definición similar a la "seriabilidad" de las transacciones
  - Todos los procesadores ven accesos a memoria en el mismo orden
    - Aunque puede ser ≠ del tiempo real exacto de emisión de los accesos
- Coherencia secuencial permite muy pocas optimizaciones
  - Escritura debe esperar hasta que se complete propagación
  - No escalable
- Uso de modelos de coherencia más "relajados" (véase figura)
- Modelos presentados hasta ahora son de tipo uniforme
  - Tratan igual todos los accesos a memoria
  - No suficientemente relajados para DSM

Modelos de coherencia más relajados



Obviados en la presentación por no usarse en DSM

Memory Consistency Models . David Mosberger, 1993

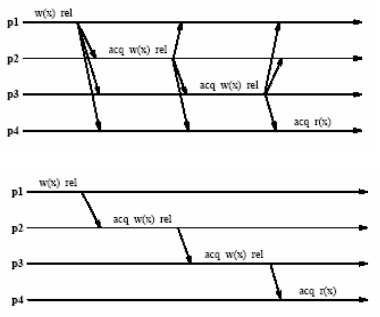
Modelos de coherencia híbridos

- Programa bien construido → no condiciones de carrera
  - Accesos conflictivos a datos deben usar sección crítica (SC)
- Modelos de coherencia híbridos
  - Distinguen operaciones de accesos a datos y de sincronización
  - Tienen en cuenta sincronización para mantener coherencia de datos
  - No se requiere propagar escrituras sobre datos inmediatamente
  - Necesario gestionar un *bit de modificado*
    - Por página en VM-DSM; Por variable en RT-DSM
  - Si no condiciones de carrera → equivalencia a c. secuencial
  - Accesos conflictivos sin sincronización: resultado impredecible
  - Varios modelos: de liberación, de entrada, de ámbito, ...
  - Distinguir dos tipos de operaciones de sincronización:
    - Entrada en SC (*acquire*)
    - Salida de SC (*release*)

M. coherencia híbridos: ERC | LRC | EC

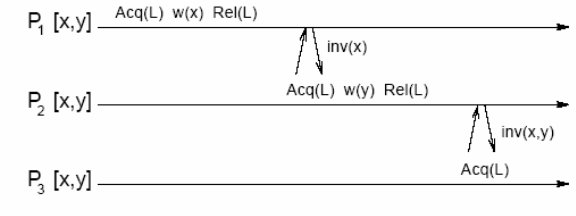
- Coherencia de liberación impaciente (ERC): Munin
  - Al salir SC: Cambios realizados durante SC se propagan a copias
    - Se invalidan (si *write-invalidate*) o se envían cambios (si *write-update*)
- Coherencia de liberación perezosa (LRC): TreadMarks
  - No propagar cambios hasta que otro proceso los necesite
  - P entra SC cerrojo C: obtiene cambios de último proceso Q que tuvo C
    - P debe ver cambios hechos por Q y previos que tengan relación causal
      - Sucesión de SC sobre un cerrojo establecen un orden causal
- Coherencia de entrada (EC): Midway
  - Todo dato compartido asociado al menos a una v. sincro (cerrojo)
  - P entra SC cerrojo C: pide cambios a último proceso Q que tuvo C
    - P debe ver cambios hechos por Q y previos que tengan relación causal
    - Pero sólo cambios de los datos asociados a C

ERC vs. LRC



Lazy Release Consistency for Distributed Shared Memory. Tesis de Peter Keleher, 1995

Orden causal en LRC

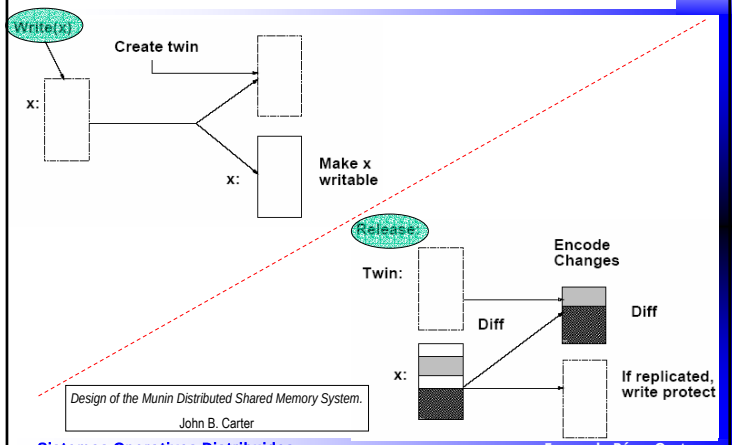


Lazy Release Consistency for Distributed Shared Memory.
Tesis de Peter Keleher, 1995

False sharing en VM-DSM con m. híbridos

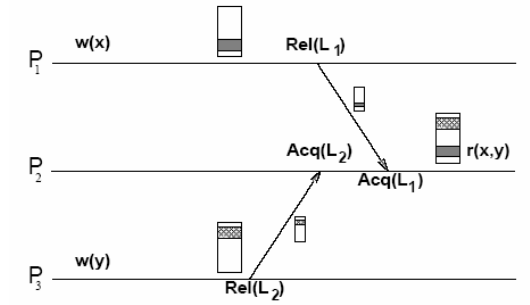
- Procesos pueden acceder a datos disjuntos en misma página
  - No necesitan SC entre ellos ya que no hay condición de carrera
- Al propagar copias se sobrescriben cambios
- Solución: "twins" y "diffs"
  - Se prohíbe modificar la página
  - Cuando se produce una excepción (STORE)
    - Se hace una copia de trabajo (*twin*)
  - Al salir SC (Munin) o cuando otro proc. pida cambios (ThreadMarks)
    - Se calcula diferencias (*diff*)
    - Se propaga (mediante invalidación o actualización) sólo *diff*
    - El receptor mezcla *diffs* con su copia de la página
  - Elimina problema de sobrescritura y reduce tráfico
    - Coste de copias y cálculo de diferencias

Twins y *Diffs* en Munin



Design of the Munin Distributed Shared Memory System.
John B. Carter

Múltiples escritores sobre misma página



Lazy Release Consistency for Distributed Shared Memory.
Tesis de Peter Keleher, 1995