Sistemas operativos avanzados

Planificación del procesador

4ª parte: planificación de máquinas virtuales

Requisitos del planificador de MVs

- Reparto de máquina virtuales sobre procesadores disponibles
  - *Virtual CPUs* (*vCPU*) sobre *Physical CPUs* (*pCPU*)
- Proporcionando reparto equitativo
- Permitiendo establecer grados de importancia entre las MVs
  - VMware *shares*; Xen *weights*
- Posibilitando agrupar MVs para asignar recursos
  - p.e. MVs clientes VIP 80% poder de cómputo; MVs resto 20%
- Fijando garantías uso mínimo de recursos (VMware *reservations*)
- Estableciendo uso máximo de recursos (VMware *limits*; Xen *caps*)
- Si máquina anfitriona es multiprocesadora
  - Pudiendo fijar en qué *pCPUs* ejecuta una MV (afinidad estricta)
  - Aprovechando afinidad natural (mejor *vCPU* en misma *pCPU*)
  - Conociendo y aprovechando la topología (SMT, CMP, NUMA)

Planificadores *non-work-conserving (NWC)*

- Planificadores estudiados hasta ahora: *work-conserving*
  - Nunca se queda un recurso sin usar si alguien lo necesita
    - Si sólo está listo proceso mínima prioridad, obtiene 100% UCP
- En ciertas situaciones puede ser mejor que se quede libre; p.ej:
  - Por *tarificación*: sólo consumes lo que has pagado
  - Por *contención*: si te dejo usar esa UCP lógica vas a afectar al proceso ejecutando en la otra UCP lógica de ese *core*
- Planificador *non-work-conserving (NWC)*
  - Puede no asignar recurso disponible aunque alguien lo necesite
- Necesarios para implementar límite máximo de recursos en MVs
  - Alcanzado límite, MV no puede usar + recursos aunque libres

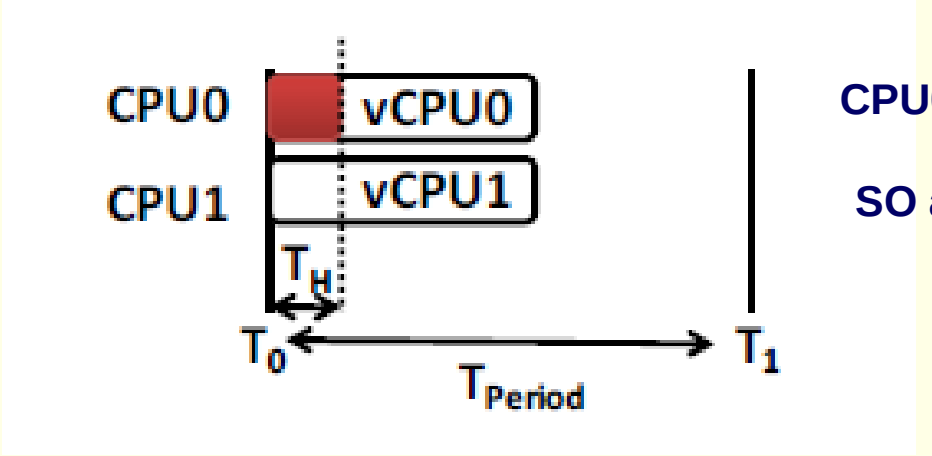
Planificadores de MVs

- No hay que reinventar la rueda:
  - Planificador de procesos → Planificador de *vCPUs*
    - *vCPU* mismos estados que Proceso
- Planificadores *fair-share* adecuados para gestionar MVs
  - *Proportional Share-Based Algorithm* de Vmware
  - *Credit Scheduler* de Xen
- CFS de Linux también para MVs (usado en KVM)
  - Uso de *cgroups* para definir MVs y grupos de MVs
    - *shares* de un *cgroup* = *shares* de VMware
  - Planificación MP cola/UCP aprovecha afinidad de topología
  - Implementación afinidad estricta
  - ¿Qué falta?→ extensión *NWC* de CFS para implementar límites
    - *CPU bandwidth control for CFS*
      http://landley.net/kdocs/ols/2010/ols2010-pages-245-254.pdf

Planificación de multiprocesadores virtuales

- Cada vez más frecuente usar MV paralelas
  - Múltiples *vCPUs* de una misma MV sobre *pCPUs* disponibles
  - Para aprovechar paralelismo de máquina subyacente
  - Incluso de gran escala (*Monster Virtual Machine*)
- Reto más complejo que MV uniprocesador
- SO realiza ciertas asunciones no válidas en MV:
  - En máquina real procesador está siempre ejecutando
    - Pero *vCPU* sólo ejecuta cuando tiene asignada *pCPU*
- Problema con la sincronización basada en espera activa
  - SO usa *spinlocks* para sincro entre llamadas y llamadas e interr.
    - Se estudiarán en detalle en capítulo sobre sincronización
  - SO asume que si procesos tienen asignada sendas UCPs
    - Ejecutan simultáneamente → espera activa de duración acotada
      - **No se cumple en MV**

Competición por *spinlock* en máquina real



CPU0 mantiene *spinlock* durante T_H.

SO asegura que proceso no puede ser expulsado.

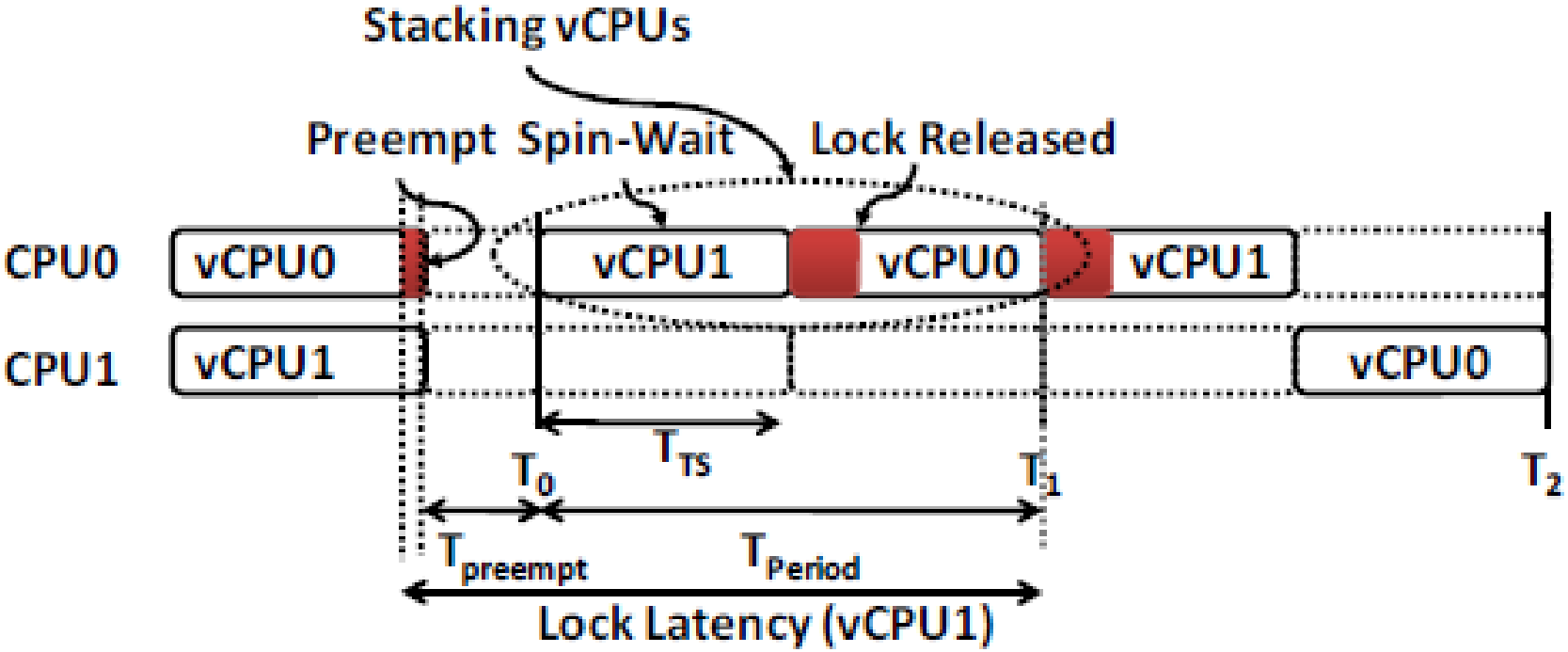
T_H microsegundos

Is Co-scheduling Too Expensive for SMP VMs? **O. Sukwong, H. S. Kim**
http://eurosys2011.cs.uni-salzburg.at/pdf/eurosys2011-sukwong.pdf

Problema y solución: *Co-Scheduling*

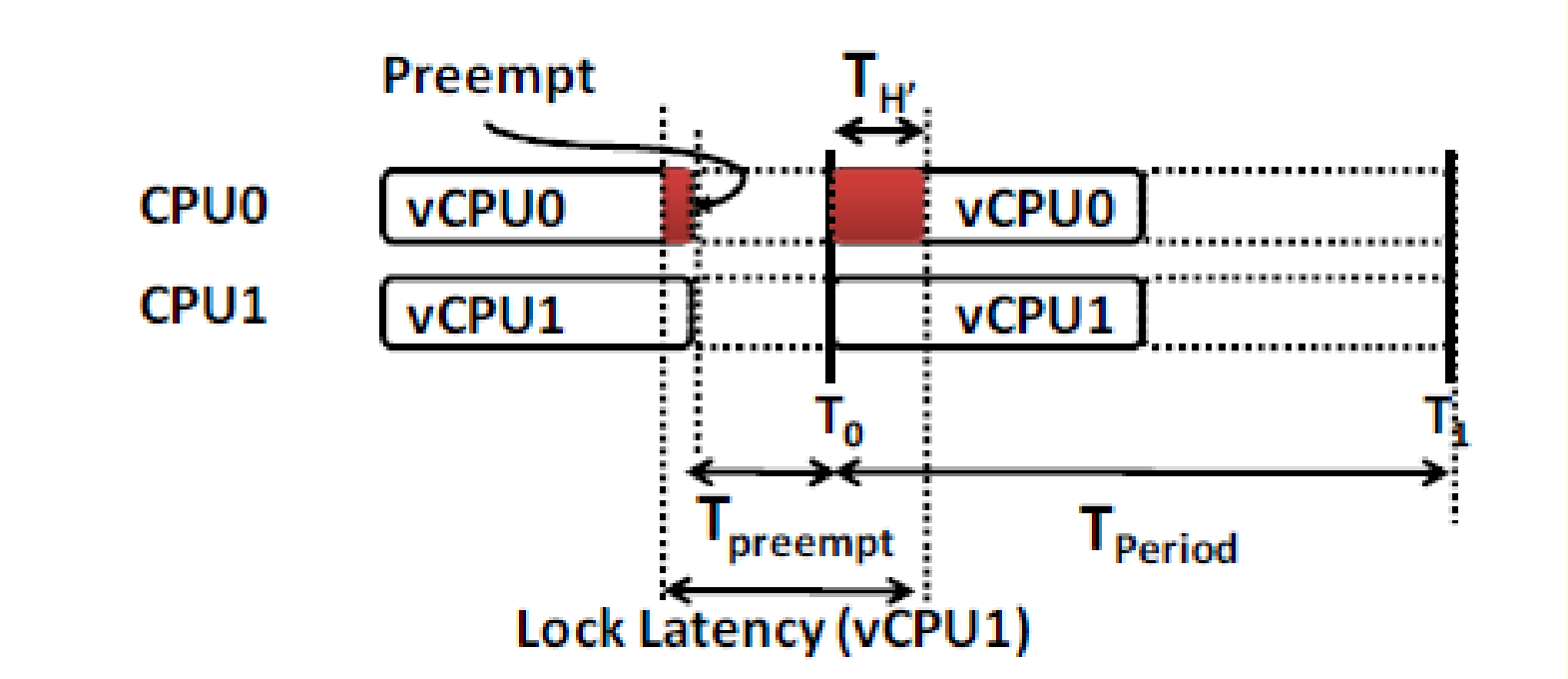
- SO no expulsa proceso poseedor de *spin*
  - Pero hipervisor puede expulsar *vCPU* donde ejecuta
- Latencia del *spin* muy elevada (milisegundos o peor)
  - Desperdicio de recursos (en gráfica T_{TS} espera activa inútil)
- Posible solución: *Co-Scheduling* (**aka *Gang Scheduling***)
  - Planificar simultáneamente todas *vCPUs* de una MV
  - Problema de fragmentación de UCPs: ejemplo
    - 4 *pCPUs*: MV1 3 *vCPUs*; MV2 2 *vCPUs*
      - Siempre se desperdicia al menos 1 *pCPU*
  - nº *vCPUs* de MV ≤ nº *pCPUs* de máquina real
- Refinamiento: *Co-Scheduling* relajado (VMware versión≥ESX3.3)
  - No ejecución simultánea de todas *vCPUs* de una MV
    - Si el desfase entre *vCPU* más adelantada y más retrasada > umbral
      - Se detiene ejecución de *vCPU* más adelantada

Competición por *spinlock* en máquina virtual



Is Co-scheduling Too Expensive for SMP VMs? **O. Sukwong, H. S. Kim**
http://eurosys2011.cs.uni-salzburg.at/pdf/eurosys2011-sukwong.pdf

Competición por *spinlock* en MV con *co-scheduling*



Is Co-scheduling Too Expensive for SMP VMs? **O. Sukwong, H. S. Kim**
http://eurosys2011.cs.uni-salzburg.at/pdf/eurosys2011-sukwong.pdf

Influencia topología MP real sobre MP virtual

- SO en MP real debe conocer su topología (SMT, CMP, NUMA)
- SO en MP virtual también debería conocerla para optimizar
  - SO debería conocer y explotar características de sus *vCPUs*
    - Pe. si 2 de sus *vCPUs* asociadas a 2 *pCPUS* de mismo paquete
    - Aunque puede dificultar migración MV a otro equipo
  - Pero en MV esta asignación es dinámica
    - P.e. una de las *vCPUs* migra a *pCPU* de otro paquete
    - Tema abierto: ¿cómo SO lo descubre y gestiona este dinamismo?
- Alternativa: ofrecer a SO alojado visión MP simétrico
  - No optimización pero facilita gestión y migración MVs
  - No tolerable en NUMA: SO debe saber que 2 vCPUs ≠ nodo
    - *vNUMA*: SO es informado de a qué nodos NUMA ∈ sus *vCPUs*
    - Planificador de MVs nunca migra *vCPU* de nodo

Otros aspectos problemáticos de MPs virtuales

- SO asume configuración de procesadores es fija
  - En MP virtual nº y características de procesadores cambiante
  - SO alojado debería gestionar *CPU-HotPlug*
    - Permite añadir/eliminar UCPs con máquina arrancada
  - ¿Aplicable a cambios dinámicos en topología?

- Linux: desactivando CPU 4

```
echo 0 > /sys/devices/system/cpu/cpu4/online
```