Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

En un sistema multiprocesor de memoria compartida, puede ser más eficiente el programar(*schedule*) una tarea en un procesador en lugar de en otro. Usando esta información de afinidad de procesador en *scheduling* de memoria compartida multiprocesador, puede mejorar el rendimiento particularmente si esta información es poco costosa de obtener y explotar. La afinidad de una tarea específica para un procesador en particular podría surgir de muchas fuentes, por ejemplo: (1) ¿Qué tan rápido una tarea puede correr en un procesador en particular en un ambiente de procesadores heterogéneos? (2) Los recursos asociados a un procesador - cada procesador podría tener un conjunto de recursos disponibles y cada tarea debe ejecutarse en el procesador que está asociado con el conjunto de recursos que requiere. (3) Basado en los contenido de los caché del procesador - podría ser más eficiente en un sistema multiprocesador de memoria compartida programar una tarea en un procesador en particular que en otro si los datos relevantes ya existen en el caché del procesador.

Cuando una tarea se programa para correr en un procesador, inicialmente experimenta un gran número de *cache misses* conforme su conforme su conjunto de datos es traído de la memoria hacie el caché. Al retardo de tiempo *Time delay* de esta ráfaga(*burst*) inicial de intentos fallidos se le llama recarda de caché transitorio (*cache-reload transient*) y al grupo de bloques de caché en uso activo por una tarea se le llama su huella(*footprint*) de caché[1]. Se sugiere que el *cache-reload transient* puede ser un factor significativo. El primer objetivo del análisis es mostrar que el tiempo de recarga del caché en sistemas existentes puede ser significativo. En la primera prueba (un procesador, y huellas de caché referenciadas a través de lecturas) se muestra como el tiempo de recarga del caché causa un incremento del 69 % en el tiempo de ejecución de la tarea. En la segunda prueba (incluye interferencia de bus y escritura de invalidaciones. Mide el tiempo medio de ejecución de múltiples tareas corriendo en un solo procesador con una carga de trabajo equivalente en varios otros procesadores) las tareas que se ejecutaron en un solo procesador se ejecuturon usando el *warm cache* mientras que las tareas que en múltiples procesadores corrían típicamente contra el *cold cache* y durante las mediciones de los tiempos de ejecución se demostró que el tiempo de recarga del caché sigue siendo significativo. Se espera que estos tiempos de recarga sean aún más significativos en arquitecturas más nuevas conforme el tamaño y el costo relativo de los intentos fallidos en caché continuen en aumento.

The Models: *The System Model:* El modelo del comportamiento del sistema las tareas alternan entre entre la ejecución en un procesador y la liberación de este procesador debido a I/O, sincronización, expiración de quantum o *preemption*. El sistema modelado consiste de dos centros de servicio. El primero es un centro de *multiple server queueing* - el cual le brinda servicios a un número fijo de tareas simultáneamente - que representa

[1] *Time delay* y *footprint*, términos de Thiebaut y Stone

al multiprocesador compuesto de M procesadores idénticos. A este centro se le conoce como *ready-queue* porque contiene aquellas tareas que están listas para ser ejecutadas pero estar esperando por un procesador disponible. El segundo centro es un servidor de capacidad infinita (*infinite capacity server*) en - no hay cola de espera porque todas las tareas son atendidas simultáneamente - y representa el tiempo empleado por las tareas mientras no son programadas(*nonschedulable*). Se asume que todas las tareas son idénticas. N denota el número de tareas dentro, la población, del modelo. La política de *scheduling* define la manera en las tareas son eliminadas de la *ready-queue*. Las demandas computacionales de las tareas por visitar el multiprocesador se asumen como independientes e idénticamente distribuidas como variables aletarias exponenciales con media D. Las tareas se convierten en *nonschedulable* por periódos aleatorios que son asumidos como independientes y distribuidos exponencialmente con media Z. *The Cache Model:*. *Bus Interference:*

The Scheduling Policies. *First Come First Served (FCFS):* FCFS ignora por completo la afinidad de una tarea con un procesador en particular. Cuando un procesador se vuelve inactivo, FCFS ejecuta una tarea en la cabeza/tope de la *ready-queue* y cuando la tarea se convierte en *schedulable*, FCFS la coloca al final de la cola. *Fixed Processor (FP):* cada procesador tiene una cola dedicada local y las tareas se asignan permanentemente a estas colas[2]. De esta forma las N tareas son divididas de forma pareja entre M procesadores y trabajan de forma independiente sirviendo N/M tareas de forma FCFS. *Last Processor (LP):* Cuando un procesador está inactivo, busca en la *ready-queue* por la primer tarea que ejecutó por última vez. Si una tarea se encuentra, el procesador la ejecuta sino, el procesador ejecuta la primer tarea del *ready-queue*. *Mininum Intervening (MI):* calcula para cada tarea x en la *ready-queue* el número de tareas ejecutadas desde la última vez que x se ejecutó allí. Luego corre la tarea con el menor valor. Cuando una tarea arriba al *ready-queue* y existen procesadores inactivos, se le asigna al procesador el cual tenga mayor afinidad con la tarea. *Limited Minimun Intervening (LMI):* limita el número de procesadores para el que una tarea mantiene información de afinidad. Una tarea es *asociada* con el procesador si mantiene datos de afinidad para el ese procesador. *Limited Minimun Intervening Routing (LMR):* es similar a LMI excepto que la decisión acerca de en dónde se va a ejecutar una tarea se hace cuando la tarea se convierte *schedulable* en lugar de cuando un procesador se pone inactivo.

Results. *Comparison of Scheduling Policies:* Los resultados de las pruebas muestran la importancia de incluir afinidad de caché de procesador en decisiones de *scheduling* especialmente en un sistema con cargas de trabajo pesadas. Bajo cargas de trabajo livianas, cuando el tiempo de recarga de la huella de caché(denotado con C) no es grande. FCFS super a FP, sin embargo conforme C aumenta, la penalización por programar*scheduling* una tarea en un procesador en donde hay poco o ninguna afinidad supera al costo del desequilibrio de la carga, FP supera a FCFS. Bajo cargas pesadas de trabajo, FP supera a FCFS en todo menos en un pequeño grupo de valores de C. Explotar la afinidad simple de procesador de una forma simple e inteligente trae beneficios. Bajo cargas livianas, LP tiene un rendimiento idéntico a FCFS. Bajo cargas pesadas, LP rinde de manera similar a FP. En cuanto MI, un algoritmo voraz, este protocolo hace decisiones locales óptimas lo cual le brinda beneficios y limitaciones. Bajo cargas liviana MI supera a LP, pero la diferencia se ensancha conforme aumenta C. Cuando la carga es pesada, MI ofrece una modesta ventaja sobre LP. La desventaja de MI es que tiene que mantener grandes cantidades de información de afinidad para cada tarea, por esa razón se prueba LMI como alternativa y como se esperaba el rendimiento de LMI se acerca a MI dependiendo si el número de asociaciones de procesador (denotado con A) se acerca al número de procesadores(denotado con M). Bajo cargas livianas LMI es muy similar a MI. La convergencia es más rapida en LMI que en MI bajo cargas pesadas de trabajo. LMR comparado con LMI: en general LMR provee bajo tiempo de respuesta de la tarea en segundo momento(denotado con $\bar{T^2}$). Bajo cargas livianas LMR y LMI tiene un rendimiento similar. Bajo cargas pesadas LMR rinde un poco peor que LMI

[2]Las tareas no migran hacia otro procesador

y esta diferencia aumenta conforma se incrementa C. Conforme la carga del sistema continua en aumento los resultados muestran una degradación significativa en las políticas de *scheduling*. Cuando la carga es muy alta un gran número de tareas son ejecutadas en cada procesador antes de que una tarea regrese al procesador y por lo tanto cada tarea experimenta una penalización grande en la recarga transitoria. *Inserting Idle Time:* Un mecanismo de pausa es necesario para determinar, en función de la carga del sistema y el tiempo de recarga del caché, si el procesador debe o no permanecer inactivo luego de completar su última tarea coincidente(*matching*) en la *ready-queue*. Al aplicar una heurística de pausa se nota que: (1) LP es equivalante a FP una vez que los procesadores empiezar a pausar. (2) El rendimiento de LMI se acerca más rápido al de MI bajo cargas livianas. (3) En cargas pesadas LMI supera a MI y FP. (4) Cuando los procesadores empiezan a pausar LMR rinde un poco peor que LMI para valores iguales de A. *Bus Interference:* Los resultados anteriores no toman en cuenta el incremento en el tráfico en el bus debido a la recarga de caché. En las mediciones se mostró que los efectos de la interferencia de bus degradan el rendimiento.

Implementation Issues. La discusión sobre estas políticas de *scheduling* se divide en dos categorías: (1) *queue-based:* considera la organización y uso de las estructuras de datos para incorporar afinidad de caché de procesador en decisiones de *scheduling*. La forma más simple de una política de *queue-based scheduling* es una variación de la implementación de la política LP: muchos multiprocesadores con memoria compartida operando en cola. Esta cola es usada para almacenar tareas sin afinidad hacia un procesador en particular tal y como tareas que no se han ejecutado previamente y tareas que han perdido su afinidad hacia un procesador debido al reemplazo de su huella(*footprint*) o por la transición hacia una huella diferente. (2) *priority-based:* considera el aumento de la disciplina de prioridades del sistema con información de afinidad de caché. El incluir esta información en el calculo de le prioridad le permite al *scheduler* balancear la afinidad de la tarea con otros criterios de planificación. Existen dos claes de políticas basadas en prioridades: dependientes y no-dependientes de tiempo. La última es bastante estática y se basa típicamente en el tipo de tarea y su estado actual. La primera provee grados de libertad adicionales en la disciplina de *priority queueing* a través de un conjunto de parámetros variales los cuales están a la disposición del diseñador para ajustaar tiempos de espera relativos a la tarea.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?
2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?
3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?
4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?
5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?