Investigación en sistemas operativos. Un kernel asimétrico para el procesamiento de red, evaluación de resultados preliminares.

Javier Díaz
jdiaz@info.unlp.edu.ar

Matías Zabaljáuregui
matiasz@info.unlp.edu.ar

Laboratorio de Investigación en Nuevas Tecnologías Informáticas
Universidad Nacional de La Plata

Resumen

Este artículo presenta los temas que se están comenzando a explorar en el Laboratorio de Investigación en Nuevas Tecnologías Informáticas como parte de una línea de investigación en sistemas operativos y expone los primeros resultados obtenidos.

La rápida evolución en las tecnologías de hardware, las redes de datos y el diseño del software de usuario contrasta con el hecho de que el diseño del kernel y los mecanismos que implementan la protección y virtualización de los recursos, en los sistemas operativos actuales, no hayan tenido avances significativos en las últimas tres décadas. Esta disparidad se ve reflejada en un overhead significativo de procesamiento, conocido como intrusión del sistema operativo.

Esta línea de investigación se enfoca en identificar, proponer y evaluar nuevas características relacionadas con la performance y escalabilidad del kernel Linux, teniendo en cuenta los nuevos desafíos que presentan las arquitecturas multicore y la computación centrada en las redes a los sistemas operativos de la próxima generación.

1. Introducción

Asumiendo un futuro tecnológico cercano donde la computación centrada en las redes y las arquitecturas paralelas serán características fundamentales, se hace evidente la necesidad de replantear el diseño del software de base teniendo en cuenta los nuevos paradigmas de procesamiento y características del hardware. Por un lado, se espera un gran desarrollo tecnológico basado en las distintas formas de almacenamiento y procesamiento distribuido, soportado por el incesante avance en

los estándares y tecnologías de redes de datos. Por otro lado, las arquitecturas con múltiples núcleos de procesamiento se están posicionando como el estándar actual para la mayoría de las configuraciones de hardware.

En este escenario, se están planteando nuevas características (y adaptando antiguas ideas) que los sistemas operativos deberán incorporar en el corto y mediano plazo. Entre otras, se pueden mencionar el diseño de kernels asimétricos y activos para explotar al máximo el paralelismo en las nuevas arquitecturas, el desarrollo de una capa de llamadas al sistema completamente asincrónicas y basadas en memoria compartida para optimizar las operaciones de entrada/salida que cada vez son más frecuentes, las nuevas tecnologías de virtualización, y las funcionalidades que incorporan soporte para computación Grid en el kernel.

A continuación se describen los temas que forman parte de la línea de investigación en sistemas operativos que se está desarrollando en el Laboratorio de Investigación en Nuevas Tecnologías Informáticas.

2. Línea de Investigación

2.1. Un kernel asimétrico y activo

Generalmente, los sistemas operativos han utilizado a las máquinas SMP de manera simétrica, intentando balancear la carga entre todos los elementos de procesamiento disponibles. En los sistemas ASMP, los procesadores se especializan para tareas específicas. Por ejemplo, es posible que una CPU responda a todas las interrupciones del hardware, mientras el resto del trabajo en el sistema sea distribuido equitativamente entre el resto de las CPUs. O puede restringirse la ejecución del código en modo kernel a sólo un procesador (a uno específico o a un procesador a la vez), mientras el código en espacio de usuario se ejecuta en el resto de las CPUs.

Yendo más allá, la idea de un kernel activo es una evolución del modelo de kernel vertical [4], y propone dedicar CPUs o cores a realizar exclusivamente actividades del kernel. Al ser un agente activo, es posible que el kernel se comunique con el hardware y el espacio de usuario utilizando mecanismos alternativos, menos costosos que los actuales (interrupciones por hardware y software).

Como se detalla en la sección 3, nuestro trabajo se centra en el estudio de las posibilidades que ofrecen estos conceptos en el tratamiento de la entrada/salida de red. Sin embargo, se han identificado posibles aplicaciones en otros dominios (virtualización, tiempo real, etc).

2.2. Llamadas al sistema asincrónicas y basadas en memoria compartida

La API que el kernel ofrece al espacio de usuario define la base de los servicios y operaciones que el sistema operativo provee a los procesos. En particular, define la sintaxis y semántica de las operaciones exportadas por el kernel, conocidas normalmente como llamadas al sistema. Los buffers para entrada/salida de datos aparecen en esta definición como argumentos a las operaciones. La semántica de pasaje de parámetros puede tener un impacto significativo en la eficiencia de la transferencia de datos entre el kernel y la aplicación. Se ha manifestado previamente [2] el costo que implica el modelo tradicional de llamadas al sistema en sistemas operativos que se ejecutan

en máquinas donde se realizan principalmente tareas de entrada/salida.

La línea de trabajo se centra en ideas alternativas orientadas a incrementar la eficiencia del kernel Linux en el manejo de entrada/salida, relacionadas principalmente con la posibilidad de realizar operaciones vectorizadas (utilizando más de un buffer de datos por invocación, lo que favorece las operaciones scatter-gather) y utilizando memoria compartida entre el kernel y el espacio de usuario, para evitar las copias físicas en memoria de un dominio de protección al otro. Además, para lograr mayor escalabilidad, evitando el overhead que implica el scheduling de grandes cantidades de threads [6], se están proponiendo APIs que poseen una semántica de entrada/salida asincrónica para operaciones de sockets y archivos. Las aplicaciones realizan operaciones sin ser bloqueadas y sin verificar previamente el estado del descriptor de socket o archivo, luego reciben eventos que señalizan la terminación de una operación previamente invocada. Este modelo permite que múltiples operaciones entrada/salida estén realizándose concurrentemente sin que el proceso o thread sea bloqueado por el sistema operativo.

2.3. Nuevas ténicas de virtualización

Recientemente, dos nuevas técnicas de virtualización han logrado un nivel de performance sorprendente con respecto a la performance en la ejecución nativa. La paravirtualización y la virtualización asistida por hardware, por separado o combinadas (virtualización cooperativa), se posicionan como una nueva capa de abstracción estándar que revolucionará la estructura de los centros de cómputos gracias a la consolidación del hardware, un mayor nivel de seguridad y redundancia, la reducción en el uso de energía, la migración de hosts virtuales sin interrumpir los servicios, etc.

Hasta ahora, la solución basada en un *hypervisor* era la manera estándar de implementar las técnicas mencionadas de virtualización. Sin embargo, en este caso es necesario duplicar ciertas funcionalidades que ya existen en un kernel. Por lo tanto, se están planteando ciertas ventajas al incorporar capacidades de virtualización al propio kernel.

La línea de trabajo se enfoca en comparar las técnicas alternativas, evaluar la integración de esta funcionalidad en el kernel y detectar las opciones que ofrecen las arquitecturas multicore a éstas tecnologías.

3. Un prototipo y resultados preliminares

Como se manifestó en un trabajo previo [2], los costos asociados a ciertos mecanismos del sistema operativo, agrupados bajo el nombre de intrusión del sistema operativo, pueden llegar a ser significativos. Esta sección presenta una modificación al kernel Linux 2.6 basada en la idea de un kernel activo, que reduce los costos relacionados con el manejo de interrupciones y la sincronización del subsistema de red en una máquina SMP. Además se exponen algunas mediciones realizadas, comparando una configuración del kernel estándar y un kernel modificado.

Con esta modificación se logra desacoplar el procesamiento de las aplicaciones de usuario del procesamiento de red del kernel. El procesamiento TCP/IP es aislado de las CPUs

que ejecutan el sistema operativo y los procesos de usuario (CPUs-HOST), y se delega a un procesador o subconjunto de procesadores dedicados a las comunicaciones (CPUs-NET). Las CPUs-HOST deben comunicarse con las CPUs-NET usando un medio de comunicación de bajo costo y no intrusivo, como la memoria compartida.

Una CPU-NET no necesita ser multiplexada entre otras aplicaciones y/o funciones del kernel, con lo cual puede ser optimizada para lograr la mayor eficiencia posible en su función. Por ejemplo, es posible evitar el costo de las interrupciones de hardware si las CPUs-NET consultan, con una frecuencia suficiente, el estado de las NICs (Network Interface Controller) para saber si necesitan ser atendidas. También se reduce la contención por datos compartidos y el costo de sincronización dado que el procesamiento de red ya no se realiza de manera asincrónica y descontrolada entre todas las CPUs de la máquina (el kernel Linux sigue el paradigma de paralelismo por mensaje [5]), sino que se ejecuta sincrónicamente en un subconjunto reducido de CPUs (posiblemente sólo una CPU, dependiendo del tráfico a procesar).

En las placas de red se desactivan las interrupciones y se verifica el estado por *polling*. El resto de las interrupciones del sistema son enrutadas a la CPU-HOST usando los mecanismos de enrutamiento ofrecidos por el Controlador Programable Avanzado de Interrupciones de E/S (I/O APIC) [1]. De esta forma, la CPU-NET no debe manejar eventos asincrónicos no relacionados con la red. Sin embargo, la interrupción del timer del APIC local se mantiene habilitada en la CPU-NET, ya que se utiliza para implementar los timers por software del kernel Linux (muy utilizados por TCP), pero además deja abierta la posibilidad de disparar eventos periódicos, como pueden ser evaluar la situación y realizar una reconfiguración automática del subconjunto de CPUs-NET.

Se modificó el kernel Linux 2.6.17.8 para implementar la arquitectura separada de CPU-HOST y CPU-NET. Las pruebas se realizaron sobre máquinas con dos procesadores AMD Opteron de 64 bits, Serie 200 DE 2 GHz (denominadas CPU0 y CPU1).

En la prueba A, se configuró el kernel para lograr afinidad total [3] del procesamiento de dos de las NICs a la CPU0 y el procesamiento de la tercer NIC se asoció a la CPU1. Esta es la configuración que logró mayor eficiencia sin modificar el código fuente. En la prueba B se utilizó el prototipo implementado. En ambas pruebas se logró un rendimiento de red cercano a 1,5 Gbps consumiendo aproximadamente el 70 % de la capacidad de cómputo de la máquina. El cuadro 1 muestra los cambios de contexto (cswch/s) e interrupciones (intr/s) por segundo durante ambas pruebas. Los cuadros 2 y 3 presentan los resultados del perfilamiento estadístico del kernel durante cada prueba, es decir, indican las funciones del kernel que utilizaron la CPU por más tiempo.

Cuadro 1: Comparación de pruebas A y B

medidas\pruebas	A	B
cswch/s	157186,17	99275,49
intr/s	131363	0

Si se comparan las pruebas A y B, se observa que la segunda prueba reduce en más del 30 % la cantidad de cambios de contexto por segundo y lleva a cero la cantidad de interrupciones de hardware. Los costos de sincronización (la función _spin_lock()) bajan notablemente y el subsistema de interrupciones (las

Cuadro 2: Perfilamiento del kernel durante la prueba A

muestras	%	símbolo
154270	13.1928	__copy_to_user_ll
153381	13.1167	_spin_lock
94636	8.0930	handle_IRQ_event
58272	4.9833	__do_IRQ
45311	3.8749	schedule
39026	3.3374	tcp_v4_rcv
30196	2.5823	qdisc_restart
26420	2.2594	__switch_to
24781	2.1192	default_idle

funciones handle_IRQ_event() y __do_IRQ()) ya no genera costos importantes.

Cuadro 3: Perfilamiento del kernel durante la prueba B

muestras	%	símbolo
148053	10.3428	__copy_to_user_ll
120776	8.4372	net_rx_action
95700	6.6855	desencolar
68739	4.8020	qdisc_restart
65176	4.5531	tcp_v4_rcv
59659	4.1677	schedule
50669	3.5397	try_to_wake_up
48140	3.3630	tcp_rcv_established
43827	3.0617	kfree
36767	2.5685	__tcp_select_window
30361	2.1210	local_bh_enable
28993	2.0254	eth_type_trans

Esta prueba de concepto hace suponer que es posible reducir la intrusión del sistema operativo en el procesamiento de red. En aquellas máquinas cuya principal función es el movimiento de datos, como es el caso con un *Storage Element* en una infraestructura Grid, puede ser beneficial dedicar una CPU o core de manera exclusiva al procesamiento de red. Actualmente se está adaptando esta idea a una arquitectura multicore y se están realizando varias optimizaciones a la implementación que resultarán en una mejora importante en la eficiencia de procesamiento.

Referencias

[1] Daniel P. Bonet and Marco Cesati. *Understanding Linux Kernel 3rd Edition*. O'Reilly, 2005.

[2] Javier Díaz and Matías Zabaljáuregui. Modificaciones al kernel Linux para redes de alta velocidad. WICC 2006.

[3] A. Foong, J. Fung, and D.Ñewell. Improved Linux SMP Scaling: User-directed Processor Affinity. INTEL.

[4] S. J. Muir. Piglet: An Operating System for Network Appliances.

[5] D. Schmidt and T. Suda. Measuring the Impact of Alternative Parallel Process Architectures on Communication Subsystem Performance. Department of Information and Computer Science. University of California.

[6] Y. Turner, T. Brencht, G. Regnier, V. Saletore, G. Janakiraman, and B. Lynn. Scalable Networking for Next-Generation Computing Platforms. Proceedings of the Third Annual Workshop on System Area Networks, Madrid, Spain, 2004.