Server Operating Systems

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

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

Actualmente existen dos enfoques para la construcciones de servidores: (1) poner el servidor por encima de un sistema operativo (SO) de propósito general. Enfoque de construcción simple pero que compromete seriamente seriamente el rendimiento porque se tiene que usar abstracciones de SO sumamente generales. Estas abstracciones frecuentemente proveen un rendimiento menor para el hardware disponible. (2) Crear un SO específicamente diseñado para una configuración de servidor en particular. Con este enfoque un SO direrente se construye desde cero lo que incrementa el esfuerzo de implementación. Además, debido a que en este enfoque no se multiplexan los recursos entre múltiples servidores, se requiere de que cada servidor tenga una máquina dedicada para él. Este segundo método es costoso y no utiliza los recursos de forma eficiente lo cual compromete el rendimiento.

Server operating Sistems (OSs). Un OSs es un conjunto de abstracciones y de soporte en tiempo de ejecución para aplicaciones de servidor especializada y de alto rendimiento. Un buen OSs debería proveer: (1) herramientas e implementaciones por defecto parametrizables de abstracciones de servidor (protocoles de red, almacenamiento) para la construcción modular de aplicaciones. (2) Total libertad para reemplazar o sobreescribir estas implementaciones por defecto y especializar abstracciones de servidor basado en características específicas de la aplicación. (3) Límites de protección, de tal forma que múltiples aplicaciones pueden compartir tiempo efectivamente en sistema de alto rendimiento.

La construcción de un OSs incluye:

- Implementación por defecto de varias abstracciones útiles para construir aplicaciones de servidor, implementadas de tal forma que pueden ser parametrizadas y combinadas dentro de las aplicaciones.
- Soporte para acceso protegido y directo de recursos de *hardware*, permitiendo a una aplicación de servidor reemplazar por completo cualquiera de sus abstracciones por defecto con su propia elección, la cual se adapta mejor a sus necesidades.
- Soporte para organización basada en eventos de aplicaciones de servidor, la cual evite la gestión de hilos y los problemas de control de concurrencia inherentes a la organización *por hilo*.
- Soporte de compilador de procesamiento de capas integrado (*Integrated Layer Processing – ILP*) dinámico, para mejorar el rendimiento de aplicaciones de *software* específico para red.

A Server Operating System Design. Specialization: Debido a que el rendimiento es crítico para aplicaciones de servidor, el prototipo de OSs soporta especialización directa. Al mismo tiempo, dado el SO extensible (exokernel) usado como base, múltiples aplicaciones pueden coexistir de forma segura en el sistema, incluso cuando algunas de ellas usan políticas de gestion de recursos diferentes. El prototipo provee una variedad de implementaciones parametrizables de abstracciones apropiadas para aplicaciones de algo rendimiento. Direct device-to-device access: La tarea principal de muchos servidores es la de mover datos desde el subsistema de almacenamiento a la red y viceversa. Esto es crítico y debería de funcionar de la mejor forma posible. Para cumplir este ideal, las aplicaciones

de servidor tiene que eliminar *scheduling* y retraso de notificaciones, recorridos de sistema de archivo y de red, y copias de datos redundantes. El prototipo de OSs integra fácilmente el control y los flujos de datos de los dispositivos de *hardware*. El movimiento de datos dispositivo-a-dispositivo se hace más eficiente al permitir a manejadores de interrupciones de red especificos de la aplicación a iniciar actividad en el sistema de disco. Event-driven organization: Por naturaleza las aplicaciones de servidor son reactivas. Muchas implementaciones de servidores usan un hilo separado (o proceso) por petición y entrada/salida convencional. Este enfoque puede incrementar la complejidad y disminuir el rendimiento debido a la creación/borrado de hilos, *thread switching*, datos compartidos y bloqueos. Otro enfoque es usar *non-blocking I/O* y una abstracción de ciclo de evento(*event loop* general, de tal forma que una aplicación de software puede consistir de un conjunto de manejadores de eventos que reaccionan a estimulos externos cuando se inicia acciones adicionales de entrada/salida. Con esta organización orientada a eventos, una aplicación puede explotar el mismo nivel de concurrencia sin los problemas del enfoque por-hilo. Dynamic, compiler-assisted ILP: una técnica para mejorar el rendimiento de *software* de red es ILP, en donde múltiples acciones de protocolos son unidas en un solo paso, miniminzando el *overhead* – se puede reducir el impacto del rendimiento de la memoria del sistema en operaciones de red.

Implementation of a Prototype Server Operating System. El prototipo se construye por encima de exokernel, que está diseñado para proveer *software* de nivel de aplicación con acceso directo y protegido a los recursos de *hardware* al limitar la funcionalidad del kernel con multiplexación de recursos de *hardware* entre aplicaciones. Specialization: para brindar especialización modular, se ha implementado librerías TCP/IP y un sistema de archivos altamente parametrizable y fácil de integrar con otros componentes de la aplicación de servidor. Direct device-to-device access: Aplicaciones puede usar el soporte provisto por las librerías TCP/IP y de sistema de archivos para construir fácilmente una ruta de datos entre la red y el disco la cual implica cero copias memoria-a-memoria, a menos que dicho copiado sea requerido por las implementaciones del controlador del dispositivo. Event-driven organization: Las librerías TCP/IP y de sistema de archivos brindan soporte para organización de aplicaciones de servidor orientadas a eventos al proveer interfases *non-blocking*. Las interfases *non-blocking* progresan tanto como pueden y luego retornan el control a la aplicación indicando qué tanto progreso ha hecho. Dynamic, compiler-assisted ILP: Se usan tuberías(*pipes*) para proveer ILP dinámico. El compilador de tuberías puede integrar varias tuberías dentro de un motor de transferencia de mensaje integrado que está codificado un un ciclo de copiado de datos especializado.

Cheetah: A Fast HTTP Server. Se inición con una implementación simple y no especializada. Luego se reemplazaron los componentes principales (TCP/IP, sistema de archivos) del SO, uno a la vez. Finalmente se agregaron especializaciones, una a la vez, lo cual fue mejorando el desempeño de Cheetah. Overview: La mayoría de la implementación de Cheetah consiste en la configuración y enlace de las librerías TCP/IP y de sistemas de archivos provistas por el SO. Una implementación de combinada de cache de disco y retransmisión de *buffer* is usada para eliminar copiado de datos y desperdicio de memoria física. Cheetah ejecuta un hilo de control que, luego de la fase de configuración/inicialización, repite un ciclo infinito de consulta por eventos de red o disco y luego le da servicio. Cuando durante el proceso de una petición HTTP alcanza un punto en donde debe esperar por algún estímulo externo, su estado crítico es guardado en la estructura de la petición y el control returon el ciclo de eventos principal. TCP/IP Specialization: Cheetah intenta reducir el costo del establecimiento de conexiones: mantiene un *pool* de estructuras de petición HTTP, usa la memoria que contiene el mensaje de la petición HTTP cuando este arriba, evitando la asignación el copiado de datos en la mayoría de los casos. Cheetah minimiza el número de paquetes de red separados para responder a una petición HTTP (usa la librería *scatter/gather* TCP/IP interfazada con la especificación FIN). Cheetah tambien usa sumas de comprobación *checksums* cuando envía los contenidos de páginas Web. Estas sumas de comprobación son guardadas en disco con el archivo correspondiente y calculadas por el sistema de archivos solo cuando el archivo es modificado.

File system specialization: Se precalcula el encabezado HTTP para cada archivo y se guarda con cada archivo, con excepción de la fecha. Este precalculo reduce el trabajo tiene que ser hecho en la ruta crítica, eliminando tiempo costoso en conversiones, generación y comparación de cadenas. Colocar imágenes adyacente a los archivos HTML leer ambos como una sola unidad mejora el rendimiento.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

La creación de sistemas operativos especializados para servidores los cuales puedan brindar abstracciones y soporte en tiempo de ejecución para aplicaciones de servidor.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Los servidores, que son la base del modelo de computación cliente/servidor, se están volviendo cada vez más relevantes. Si se quiere cumplir la promesa de acceso globa a la información, implementadores de soluciones distruidas y de alto rendimiento deben de construir una variedad de aplicaciones de servidor que puedan dar soporte a un gran número de clientes activos. Idealmente, el desarrollo y operación de estos ambientes deberían conducir a construcción de servidores de forma fácil y modular y así entregar el rendimiento del *hardware* subyacente sin requerir que la máquina esté dedicada a cada servidor. Desafortunadamente el status quo actual se queda corto en este ideal.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Existen numerosas implementaciones de servidores sobre SO como Unix y Windows NT. En estos sistemas el enfoque de acelerar el rendimiento es a partir del uso de *hardware* más rápido. Los ejemplos poco comunes: *sotware* de servidor construido o enlazado dentro un kernel rudimentario:

- Los servidores de alto rendimiento de *Network Appliance* construidos basados en sistemas NFS, los cuales dedican a cada sistema a una aplicación de servidor corriendo por encima de un kernel rudimentario. Este enfoque elimina límites de protección haciendo que compartir la máquina entre múltiples servidores/aplicaciones sea difícil.
- Aspectos del prototipo de OSs expuestos en el artículo han sido propuestos o bien implementados previamente. (ILP, Programación orientada a eventos, soporte dispositivo-a-dispositvo, especialización por medio de SO extendibles – SPIN, exokernel, Synthesis, Cache Kernel, Vino, Scout)

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

La construcción de un prototipo de un SO para servidor como un conjunto de librerías por encima del exokernel Aegis, el cual provee a la aplicaciones de acceso directo y seguro a los recursos de *hardware*. En el diseño e implementación del OSs se vió este soporte como un enfoque claro en la identificación de abstracciones y construcción de librerías que simplifican la construcción de aplicaciones de servidor altamente especializadas que aprovechan el total del rendimiento del *hardware*. Resaltan tres contribuciones:

1. Describir y argumentar que para los sistemas operativos de servidores existe una mejor forma de construir aplicaciones de servidor de alto rendimiento.
2. Identificación de técnicas de diseño para sistemas operativos de servidor y discusión de cómo aplicarlas de una forma modular.
3. La descripción de prototipo de un sistema operativo de servidor y su uso en la construcción de un servidor HTTP de alto rendimiento.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Los resultados muestran que la especialización agresiva que brinda Cheetah mejora el rendimiento en más de un orden de magnitud para tasas altas de peticiones y documentos de tamaño pequeño.

- Documentos servidos por segundo: para documentos de tamaño pequeño (0 bytes, 10 bytes y 100 bytes), Cheetah sirve 8 veces más peticiones que el cache de Harvest[1] que a su vez sirve 2 veces más solicitudes que el servidor NCSA por si solo. Conforme se sirven documentos de tamaños más grandes la diferencia baja pero sigue siendo favorable para Cheetah.
- Documentos de 100 bytes servidos de acuerdo al número de clientes: el rendimiento de ambos NCSA y Harvest es mas o menos independiente del número de clientes. El rendimiento de Cheetah se incrementa con el número de clientes. Con un cliente, Cheetah sirve 261 peticiones por segundo que es aproximadamente 3,4 veces más rápido que Harvest. Con 6 clientes, el desempeño de Cheetah super al de Harvest por un factor de 9.

[1] Harvest httpd-accelerator