Provisión de QoS en servidores de Internet en el contexto de múltiples Servidores Virtuales*

Juan A. Villar, Francisco J. Alfaro, José L. Sánchez
Depto. de Informática. Escuela Politécnica Superior
Universidad de Castilla-La Mancha
02071 - Albacete, España
{juanan, falfaro, jsanchez}@info-ab.uclm.es

José F. Duato
Facultad de Informática
Univ. Politécnica Valencia
46022 - Valencia, España
jduato@disca.upv.es

Resumen

Hoy en día existe una gran cantidad de información y servicios disponibles a través de Internet, generados en su inmensa mayoría por la ejecución de aplicaciones en los muchos servidores que existen en todo el mundo integrados en la infraestructura de las empresas proveedoras de servicios. En buena parte de estas empresas, los recursos disponibles están infrautilizados, pues habitualmente han sido dimensionados para poder garantizar tiempos de respuesta bajo condiciones de carga máxima.

En este trabajo se exponen algunas ideas que pueden conducir a una infraestructura de los proveedores de servicios más eficiente y rentable, y que garantice los requisitos de calidad de servicio demandados por sus clientes. Se proponen mejoras en el uso y gestión de los recursos hardware disponibles. Se trata de conseguir algoritmos de planificación de procesos eficientes y mecanismos de control de admisión capaces, conjuntamente, de dimensionar los servidores de una forma más precisa, garantizando la disponibilidad de los recursos y/o el retardo máximo para dichas aplicaciones.

1. Introducción

La última década ha sido testigo de un fuerte incremento en la variedad de dispositivos de computación, así como en el número de los usuarios de esos dispositivos. Además de las tradicionales estaciones de trabajo, computadores de sobremesa y portátiles, nuevos dispositivos como PCs de bolsillo, PDAs y teléfonos móviles con capacidades multimedia, se utilizan hoy en día masivamente, por personas de todas las edades.

Por otra parte, la conectividad también ha mejorado de una forma importante. El módem tradicional conectado a la línea telefónica ha sido sustituido por los dispositivos cableados que ofrecen mayor ancho de banda, y ya hay disponibles conexiones inalámbricas en congresos, aeropuertos, etc.

Las razones principales del amplio uso de todos estos dispositivos son su disponibilidad y potencia, e incluso, y más importante, la cantidad de información y servicios disponibles a través de Internet. De hecho, la manera tradicional de usar los computadores (habitualmente referida como *Desktop Computing*) ha sido sustituida en gran parte por otra manera que ha venido a llamarse *Internet Computing*. Por otra parte, proporcionar información y servicios a través de Internet está convirtiéndose en una necesidad para las empresas que quieran sobrevivir en un mercado tan competitivo como el actual.

La información y los servicios proporcionados a través de Internet se generan a través de aplicaciones ejecutadas en los muchos servidores que existen en todo el mundo. Muchos de esos servidores estaban basados originalmente en ordenadores personales (PCs), pero el gran

*Este trabajo ha sido parcialmente financiado por la CICYT con el proyecto TIC2003-08154-C06.

incremento en el número de los usuarios ha conducido rápidamente a un incremento drástico en el número de los clientes que tienen acceso concurrentemente a un servidor dado. Como consecuencia, el poder de computación y el ancho de banda de la entrada/salida ofrecida por un único procesador y unos cuantos discos no son suficientes para proporcionar un tiempo de respuesta razonable.

Los clusters de PCs, debido a su buena relación coste/prestaciones, han surgido como plataforma adecuada para ejecutar esas aplicaciones de Internet, y así proporcionar servicio a los centenares o millares de usuarios que concurrentemente solicitan esos servicios. Un ejemplo de este tipo de clusters es el utilizado para dar servicio al popular buscador *Google*, el cual se estima está formado por unos 15,000 ordenadores basados en PCs de sobremesa unidos por una red Ethernet de 100 Mbits [3].

En la mayoría de los servidores actuales de Internet los procesadores están infrautilizados. La razón principal es que los servidores son habitualmente dimensionados para garantizar tiempos de respuesta bajo carga máxima. Una consecuencia directa de este sobredimensionamiento es un coste del servidor significativamente mayor del estrictamente necesario.

La hipótesis de la investigación que se pretende desarrollar es que es posible utilizar los servidores de una manera mucho más rentable mientras se sigue garantizando tiempo de respuesta y productividad bajo carga máxima para aquellas aplicaciones que requieren una cierta cantidad de recursos y/o un tiempo de respuesta máximo.

Para conseguirlo se usará el concepto de Servidor Virtual (SV). Un SV consiste en un conjunto de recursos (procesadores, memoria, discos, interconexión) que se dedican a la ejecución de un grupo de aplicaciones. Usando el concepto de SV, los requisitos de cada aplicación se pueden identificar, asignando así un conjunto correctamente dimensionado de recursos al SV que ejecuta la aplicación.

Por otra parte, la aceptación que tiene Linux [5] como sistema operativo principal para servidores en Internet crece cada día. Evidentemente, no es la única posibilidad, existen otros como Solaris [8] o Windows 2003 [9]. Algunas de las ventajas de Linux son su estabilidad, disponibilidad del código fuente, seguridad, escalabilidad, y abundancia de documentación. Todo esto, y más, convierte a Linux en una herramienta perfecta para desarrollar el trabajo cuyas líneas iniciales se presentan en este artículo. Así, los datos que se aportan en este artículo corresponden con los kernels de las versiones 2.6 de Linux.

Este artículo está organizado de la siguiente manera: la Sección 2 proporciona una breve descripción de los problemas que se identifican en los servidores actuales y cuál puede ser el marco de actuación para solventarlos. La Sección 3 analiza desde un punto de vista general la planificación de tareas en Linux en los kernels 2.6. Tras ello se presenta una propuesta para la implementación del concepto de SV sobre Linux. Para terminar se incluyen las conclusiones del trabajo que prácticamente se pueden ver como el trabajo futuro.

2. Servidores de Internet

Los clusters de PCs se están consolidando en la mayoría de los casos como plataforma para proporcionar servicios de red debido a sus importantes ventajas. La manera tradicional de compartir un cluster entre varias aplicaciones consiste en usar cierto tipo de herramientas o *middleware* para distribuir la carga entre los procesadores en el cluster (por ejemplo, Condor [6]). Sin embargo, estas herramientas no proporcionan soporte para calidad de servicio (QoS) por sí mismas, y así, no pueden garantizar un tiempo de respuesta máximo.

En la mayoría de los servidores de Internet actuales los procesadores están infrautilizados. La razón principal, como ya se ha comentado, es que los servidores habitualmente se dimensionan para garantizar tiempos de respuesta bajo máxima carga. Una consecuencia directa de este dimensionamiento es un coste del servidor significativamente mayor del estrictamente necesario.

2.1. Servidores virtuales

Como ya se ha comentado, un Servidor Virtual (SV) consiste en un conjunto de recursos (procesadores, memoria, discos, interconexión) que se dedican a la ejecución de un grupo de aplicaciones. Esos recursos se corresponden con parte de los dispositivos físicos disponibles de tal forma que varios SVs pueden coexistir en un servidor físico, compartiendo los dispositivos que contiene. Los SVs permiten que las aplicaciones proporcionen más detalles en sus requisitos específicos. De esta forma, una vez identificados los requisitos de cada aplicación, se puede utilizar el concepto de SV para asignar un conjunto de recursos correctamente dimensionado al SV que ejecuta la aplicación.

Por otra parte, las aplicaciones pueden especificar requisitos que varían a lo largo del tiempo. Por ejemplo, un servidor de Internet recibirá probablemente más peticiones durante las horas del día. Así, puede configurarse durante la noche para realizar tareas de mantenimiento, como copias de seguridad. En este caso, el sistema puede reconfigurarse dinámicamente asignando un conjunto de recursos nuevo a cada SV de acuerdo con sus necesidades particulares en cada momento. Obviamente, la reconfiguración no puede ser demasiado frecuente, o la sobrecarga introducida por ésta consumirá una fracción significativa de tiempo de los recursos del sistema.

Un ejemplo de software *middleware* que implementa el concepto de SV es LVS (Linux Virtual Server) [4]. El objetivo de este software es distribuir los procesos y los datos de todos los clientes entre los diversos elementos del cluster, maximizando cierta métrica de rendimiento, como por ejemplo la productividad.

Además de permitir una especificación detallada de los requisitos de la aplicación, los SVs proporcionan un marco excelente para incrementar la seguridad y la tolerancia a fallos, y también para proporcionar garantías de QoS. Efectivamente, como cada SV utiliza solamente un conjunto dado de recursos, no requiere accesos al resto de recursos. Así, es posible implementar algunos mecanismos hardware y/o software para prevenir accesos no autorizados a los recursos que no pertenecen a un servidor dado. Esto no es una tarea trivial puesto que muchos recursos pueden compartirse por diversos SVs para utilizar el sistema de una forma más eficiente.

La tolerancia a fallos también se puede aumentar utilizando el concepto de SVs porque el sistema se puede reconfigurar ante la detección de fallos. La reconfiguración reasignará recursos a los diversos SVs de tal forma que no se asigne ningún componente que haya fallado a ningún SV. Una vez más, esto no es una tarea trivial porque los datos críticos necesitan contener cierta redundancia o ser replicados para evitar pérdida de los datos (por ejemplo, usando configuraciones adecuadas de RAIDs).

También, los fallos en los sistemas de interconexión que conectan los nodos del cluster requieren la existencia de rutas alternativas para mantener la conectividad del sistema, y el uso de técnicas adecuadas que sean capaces de usar esas rutas alternativas.

La QoS también puede mejorarse significativamente usando el concepto de SV. Si cada una de las aplicaciones de un determinado conjunto se está ejecutando sobre un SV distinto, es posible especificar los requisitos particulares de cada SV y su variación a lo largo del tiempo. Así, es posible implementar alguna estrategia de control de admisión que acepte o rechace una petición dada para crear un SV cada vez que se reconfigura el sistema. Esta estrategia, combinada con un planificador dinámico que asigne recursos a los SVs según sus necesidades, podrá proporcionar las garantías de QoS a cada aplicación (por ejemplo, tiempo de respuesta limitado para un número máximo de peticiones concurrentes).

Por otra parte, si alguna de las aplicaciones que funcionan en algunos SVs excede la demanda prevista, las aplicaciones que funcionan en los SVs restantes no se verán afectadas por este exceso de demanda. Una vez más, cerciorarse de que cada SV pueda utilizar los recursos solicitados en el tiempo adecuado no es una tarea trivial, porque los SVs no se asignan a un conjunto disjunto de recursos. En este sentido, es necesario un cierto grado de compartición de recursos entre los diversos SVs para maximizar las prestaciones del sistema.

Todos estos problemas (seguridad, tolerancia a fallos y QoS) se consideran extremadamente importantes dado el papel de los servidores en el mundo en el que nos movemos. Sin embargo, para limitar el alcance de esta investigación ésta se centrará solamente en uno de ellos. En concreto, se ha seleccionado el problema de proporcionar garantías de QoS porque es el que el usuario percibe de una forma más rápida y directa, y también debido a su drástico impacto en el coste del sistema.

Para poner en funcionamiento todo lo dicho anteriormente, se hace necesario disponer de un sistema operativo que contemple planificación dinámica. Gracias a toda la comunidad de desarrolladores *open-source* [17], se puede disponer del sistema operativo Linux. Se trata sin duda del mejor lugar posible para el desarrollo de tal planificador.

3. Planificador de tareas de Linux

Cada día, más y más empresas están tomando conciencia de las grandes posibilidades que implica la implantación de sistemas basados en Linux en sus servidores de Internet. A su vez, se están viendo apoyadas por grandes empresas del sector como por ejemplo IBM [14] o HP [13], que ofrecen soporte de Linux en toda su gama de servidores. Aparte están las empresas que basan su modelo de negocio en Linux y el software open-source como Red-Hat [15] o VA Software [16]. Ante este panorama la idea de desarrollar el concepto de SV en el kernel de Linux parece acertada. El primer paso es entrar a valorar cómo está implementado el planificador de Linux actualmente, puesto que se pretende implementar uno propio, para después poder implantar nuestra propuesta en él.

El planificador de tareas de Linux es el componente que selecciona la siguiente tarea en ejecutarse en el procesador. Dicho componente se considera todo un subsistema del kernel[1] que divide en el tiempo el uso del recurso procesador entre las tareas que están listas para ejecutarse. Por tanto, de las decisiones que tome el planificador de tareas dependerá directamente el rendimiento total del sistema. Es por esto que el planificador se considera la piedra angular en todos los sistemas operativos multitarea, como es Linux.

La responsabilidad del planificador es maximizar la productividad del procesador, mientras se minimiza el tiempo de respuesta de las aplicaciones, y a su vez, realizar un reparto justo del procesador entre las tareas. Además, al usuario del sistema le tiene que dar la impresión de que todas las aplicaciones se están ejecutando simultáneamente.

Para alcanzar los objetivos anteriores, el planificador clasifica las tareas en dos grupos: las tareas *CPU-Bound* son aquellas que hacen un uso alto del procesador. Por contra, las tareas *I/O-Bound* son tareas que pasan mucho tiempo esperando eventos del subsistema de entrada/salida.

Las primeras encajan en el perfil de aplicaciones de alto cómputo y poca interacción con el usuario. Por contra, las segundas se identifican con tareas interactivas, donde los eventos que esperan son las órdenes (por ejemplo ratón o teclado) que emite el usuario. El usuario siempre introduce entre los eventos que emite grandes esperas de varios órdenes de magnitud mayores, respecto a las esperas provocadas por dispositivos hardware (p.e. discos, NICs, etc.). Además, el usuario siempre espera que el sistema reaccione de forma inmediata a sus eventos. Aparte están las tareas con restricciones temporales que soporta Linux, llamadas tareas de tiempo real o *soft real time tasks*. Estas reciben un tratamiento diferente respecto a las tareas de usuario.

A continuación, se describen algunos detalles relacionados con el funcionamiento del planificador de tareas de Linux.

3.1. Arrays de prioridad

Linux utiliza un mecanismo basado en prioridades para planificar las tareas [1]. Por defecto, se declaran 140 niveles de prioridad que se reparten del siguiente modo: los 100 primeros para tareas de tiempo real y los 40 restantes, menos prioritarios, para tareas de usuario.

[1] Como ya se ha comentado, todo el estudio que se ha realizado está centrado en los kernels de la serie 2.6

El valor de prioridad que tiene asociado una tarea en el sistema se calcula dinámicamente en base a un valor de prioridad estática o *nice* establecido por el usuario propietario de dicha tarea, más un valor dinámico entre [-5,+5] que el planificador le calcula en función del uso que haga del procesador (+5), o de la E/S (-5). De este modo se intenta priorizar las tareas interactivas en contra de tareas de cómputo. A las tareas de tiempo real sólo se les considera el valor de nice.

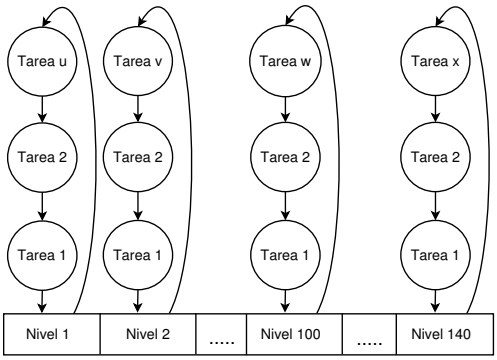

Figura 1: Array de prioridad.

Las tareas se clasifican en alguno de los niveles de prioridad, y deben ejecutarse antes que el resto de tareas menos prioritarias. El tiempo que a cada tarea se le va a permitir ejecutarse se llama *timeslice* y si por alguna razón no se completa ese tiempo la tarea vuelve a la lista a esperar, para posteriormente consumirlo cuando pueda. En esta lista se aplica un política *round-robin* en el orden de llegada. En la Figura 1 se muestra el esquema de un array de prioridad utilizado por el kernel donde se pueden apreciar los diferentes niveles y las listas de tareas, donde u,v,w,x ≥ 0.

3.2. Colas de ejecución

El planificador crea una cola de ejecución o *runqueue* para cada procesador, donde mantiene toda la información necesaria para planificar las tareas. Particularmente, dentro de cada *runqueue* el planificador utiliza dos arrays de prioridad. En el array de tareas activas, llamado *Active*, se mantienen las tareas que no han consumido su timeslice. Análogamente en el array de tareas expiradas, *Expired*, se mantienen las tareas que han consumido su timeslice. En la Figura 2 se muestra la distribución de colas de ejecución entre los procesadores.

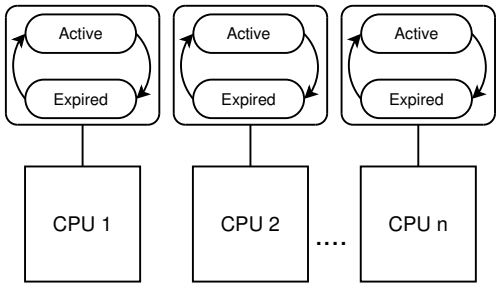

Figura 2: Ejemplo de runqueues en un sistema con n procesadores. Sólo hay una runqueue por procesador.

La duración del timeslice asignada a cada tarea se escala en el rango [5 ms, 800 ms], aunque esto se controla desde macros a la hora de compilar el kernel.

Básicamente, cuando una tarea ha consumido su timeslice, el planificador la saca del array Active, le calcula el siguiente timeslice y la guarda en el array Expired. Cuando el array Active se ha vaciado entonces el planificador intercambia los punteros de los arrays Active y Expired. A esto se le llama ciclo de planificación y fue introducido en la serie 2.6, siendo la característica más importante del planificador.

3.3. Ciclo de planificación

Cada vez que el planificador de Linux vacía el array de prioridad Active y lo intercambia con Expired, se dice que ha completado un ciclo de planificación. En las versiones anteriores a los kernels 2.6, el tiempo que se empleaba en un ciclo dependía proporcionalmente del número de tareas existentes (complejidad O(n) siendo n el número de tareas) puesto que cuando todas las tareas habían terminado su timeslice, era entonces cuando el planificador calculaba para cada tarea el siguiente timeslice. Ahora en los kernels 2.6, cuando se llega a ese punto,

simplemente se intercambian los punteros de los arrays Active y Expired, ya que antes de insertar cada tarea en el array Expired ya calcula el timeslice. Esta operación consume un tiempo constante e independiente del número de tareas que existen en el array Expired (complejidad O(1)). Al final, esta característica ha sido la que ha bautizado al planificador como "Planificador O(1)".

En cada procesador, el subsistema de planificación realiza, en un bucle infinito, los siguientes pasos que corresponden a un ciclo de planificación:

- Tomar una tarea del array Active y cederle el procesador para que se ejecute.
- Cuando la tarea abandone el procesador se mira si ha consumido su timeslice. Si lo hizo se le calcula uno nuevo y se manda al array Expired, de lo contrario se le devuelve al array Active con la parte que no ha consumido.
- Cuando el array Active se vacía, entonces se intercambian los arrays y se vuelve a comenzar otro ciclo de planificación.

4. QoS usando Servidores Virtuales

La idea sobre la que se va a trabajar consiste en utilizar los servidores de una manera mucho más rentable mientras se siguen garantizando tiempo de respuesta y productividad bajo carga máxima, para aquellas aplicaciones que requieren una cierta cantidad de recursos y/o un tiempo de respuesta máximo al procesar peticiones de los clientes. En particular, se cree que esto es posible en el contexto de varios SVs que comparten un único servidor físico. Y para ello se han establecido los siguientes supuestos:

1. Las aplicaciones que se ejecutan en diferentes SVs tienen generalmente necesidades de recursos que varían a lo largo del tiempo. Por lo tanto, considerando cuidadosamente la distribución de los requisitos de recursos a lo largo del tiempo, es posible asociar varias aplicaciones (o SVs) en un único servidor físico sin solicitar una potencia máxima de cómputo tan alta como la suma de las cargas pico de todas las aplicaciones (o SVs).
2. No todas las aplicaciones que se ejecutan en un servidor en un instante dado en el tiempo requieren tiempo de respuesta máximo. Por lo tanto, es posible diseñar el planificador del procesador de tal forma que diferentes aplicaciones reciban diferente tratamiento. Como consecuencia de esto, no es necesario dimensionar un servidor para la carga máxima, existiendo maneras más rentables de compartir recursos entre aplicaciones a la vez que se sigue garantizando el tiempo de respuesta para aquellas aplicaciones que tienen tal requisito.
3. No todas las aplicaciones que tienen requisitos de retardo máximo necesitan proporcionar la misma QoS a sus clientes, es decir: diferentes aplicaciones tienen diferentes requisitos. Por lo tanto, un planificador adecuado, capaz de planificar los procesos según sus requisitos precisos, será capaz de utilizar los recursos de una forma más rentable.

En particular, se plantea desarrollar toda la infraestructura necesaria para conseguir implementar una política round-robin con pesos con múltiples niveles de prioridad dentro del planificador de Linux. Dicho mecanismo está inspirado en el mecanismo de arbitraje de canales virtuales definido para InfiniBand [18]. En concreto, esta estrategia está basada en la reciente investigación que se ha desarrollado en el contexto de InfiniBand para proporcionar QoS en este tipo de redes [19]. Para ello se tendrá que modificar todo el mecanismo de gestión de los arrays de prioridades que implementa actualmente el kernel, y definir otro mecanismo para el cálculo de las prioridades de las tareas. Estas estrategias deben considerar todas las relaciones existentes entre las tareas, hilos y procesos que conforman una aplicación concreta.

Las aplicaciones (o SVs) serán clasificadas en varias categorías, dependiendo de sus requisitos de QoS. En particular, se ha pensado

implementar soporte para las siguientes categorías:

- Recursos dedicados sensibles al retardo (DRTS): Aplicaciones que requieran un conjunto garantizado de recursos (virtuales) y un retardo máximo garantizado. Un ejemplo de una aplicación que encaja en esta categoría es un servidor de vídeo bajo demanda.

- Recursos dedicados (DR): Aplicaciones que necesitan un conjunto garantizado de recursos (virtuales) pero no requieren retardo máximo. Un ejemplo de una aplicación sería un servidor web. Obsérvese que, en promedio, el tiempo de respuesta para la categoría DR será tan buena como para la categoría DRTS porque se garantizan suficientes recursos para esta categoría. Sin embargo, la principal diferencia con respecto a la categoría DRTS es que el planificador no proporciona ninguna garantía de retardo para la categoría DR.

- Best Effort (BE): Aplicaciones que no necesitan ni un conjunto garantizado de recursos ni un retardo máximo, pero requieren una cierta cantidad de poder computacional, y así, requieren toda la capacidad de cómputo disponible en cierto instante de tiempo. Un ejemplo de aplicación sería cualquiera que requiera computación numérica a gran escala.

- Challenge (CH): Aplicaciones que sólo serán ejecutadas cuando la carga total está por debajo de cierto umbral o será periódicamente planificada cuando la carga es baja. Un ejemplo de aplicación sería las encargadas de hacer copias de seguridad.

Además de un algoritmo de planificación, se propone utilizar un algoritmo de control de admisión para determinar si un SV dado con ciertos requisitos se puede admitir o no. Dados los requisitos del SV y su distribución a lo largo del tiempo, este algoritmo calculará el nivel de prioridad adecuado para las aplicaciones más exigentes que se ejecutan en ese SV. Estos parámetros se calcularán para todos los intervalos de tiempo que presenten diferentes requisitos, y el SV se admitirá solamente si es posible mantener los requisitos de los SVs previamente establecidos.

La principal ventaja de la estrategia propuesta es que es posible garantizar recursos dedicados y retardos máximos a los SVs independientemente de los SVs que sean aceptados posteriormente, e independientemente de si otros SVs exceden sus requisitos en tiempo de ejecución, con tal que el algoritmo de control de admisión se ejecute cada vez que un nuevo SV necesita ser asignado. Obviamente, si un SV excede sus requisitos, no será posible garantizar retardo máximo a las aplicaciones que se ejecutan en él, pero el hecho importante es que esto no afectará a otros SVs.

Finalmente, hay que mencionar que se comenzará a desarrollar la propuesta en sistemas monoprocesador, pero con la mente puesta en las últimas tecnologías en arquitectura de procesadores como, por ejemplo, Hyper-Threading [10] o Multi-Core [11, 12], y cómo no, terminar dando soporte en sistemas basados en clusters puesto que son estos los ámbitos más importantes para las aplicaciones de Internet.

5. Conclusiones

En este trabajo, se ha realizado una breve introducción del concepto de Servidor Virtual, mostrando cómo se puede aplicar en el ámbito de los servidores en Internet para proporcionar calidad de servicio, además de garantizar tiempo de respuesta y productividad bajo carga máxima a un conjunto de aplicaciones, sin necesidad de sobredimensionar los servidores físicos, y por tanto, consiguiendo reducir la inversión económica.

Igualmente, se ha presentado el planificador de tareas que implementa Linux 2.6, haciendo especial hincapié en su mecanismo de prioridades, en los arrays de prioridades y las colas de ejecución de los procesadores. Ambas estructuras de datos sirven para la toma de decisiones de planificación.

Para finalizar, se ha esbozado una propuesta para desarrollar las modificaciones necesarias en Linux con el objetivo de implementar el concepto de servidor virtual. El objetivo principal es conseguir un algoritmo de planificación eficiente apoyado en la base teórica ya desarrollada para la gestión de las tablas de arbitraje de los puertos en InfiniBand, y paralelamente desarrollar un mecanismo de control de admisión de nuevos servidores virtuales. El diseño adecuado de la propuesta permitirá utilizar los servidores de una manera mucho más rentable mientras se sigue garantizando tiempo de respuesta y productividad bajo carga máxima, para aquellas aplicaciones que requieren una cierta cantidad de recursos y/o un tiempo de respuesta máximo al procesar peticiones de los clientes en Internet.

Referencias

[1] Robert Love, *Linux Kernel Development, Second Edition*, Novell Press, 2005.

[2] Josh Aas, *Understanding the Linux 2.6.8.1 CPU Scheduler*, `http://josh.trancesoftware.com/linux`, 2005.

[3] Luiz A. Barroso, Jeffrey Dean, Urs Hölzle, *Web Search For A Planet: The Google Cluster Architecture*, `http://www.computer.org/micro/mi2003/m2022.pdf`

[4] *Linux Virtual Server Project*, `http://www.linuxvirtualserver.org`

[5] *The Linux Kernel Archives*, `http://www.kernel.org`

[6] *Condor Project Homepage*, `http://www.cs.wisc.edu/condor`

[7] Daniel P. Bovet, Marco Cesati, *Understanding the Linux Kernel, Second Edition*, Oreilly, 2002.

[8] Sun Microsystems Inc., *Solaris*, `http://www.sun.com/software/solaris`

[9] Microsoft Co., *Windows 2003*, `http://www.microsoft.com/windowsserver2003`

[10] Intel Co., *Intel Hyper-Threading Technology*, `http://www.intel.com`

[11] Intel Co., *Intel Multi-Core Technology*, `http://www.intel.com`

[12] AMD Inc., *AMD Multi-Core Technology*, `http://multicore.amd.com`

[13] HP Services & Linux, `http://www.hp.com/linux`

[14] IBM Linux, `http://www.ibm.com/linux`

[15] Red-Hat Inc., `http://www.redhat.com`

[16] VA Software Co., `http://www.vasoftware.com`

[17] The Free Software Foundation, `http://www.fsf.org`

[18] InfiniBand Trade Association, *InfiniBand Architecture Specification*, Octubre 2000.

[19] Francisco J. Alfaro, *Una sencilla metodología para garantizar calidad de servicio en subredes InfiniBand*, Tesis Doctoral ISBN:8484273202, 2004.