Enhanced Network Ram Disk. Un sistema de almacenamiento fiable usando memoria remota

Alejandro Lucero, Pedro Lopez, Jose Duato

La capacidad de los procesadores se dobla cada 18 meses, y las tecnologías en redes de área local están acercándose al rendimiento reservado a sistemas multiprocesadores. Los discos magnéticos aumentan su capacidad de almacenamiento a un ritmo similar, sin embargo, la latencia de acceso a los mismos está limitada por la naturaleza mecánica de estos dispositivos. Esto convierte a los discos magnéticos en el pricipal cuello de botella de los actuales sistemas, siendo un problema que se agudizará en el futuro. Gracias a los avances en los procesadores y sobre todo en las comunicaciones en redes de área local, es posible usar la memoria de nodos remotos como sistema de almacenamiento. La principal desventaja de este tipo de sistemas es su fiabilidad, ya que la probabilidad de fallo aumenta proporcionalmente con el número de nodos, y que el fallo de un nodo afecta a todo el sistema. En este trabajo exponemos un nuevo método para obtener la fiabilidad. Hemos basado nuestro proyecto en un trabajo anterior, consiguiendo mejorar sus prestaciones además de eliminar su principal limitación: su uso en sistemas transaccionales.

Tabla de contenidos

1. Introducción 1
2. Porqué mejorado: Desarrollos anteriores 2
3. Diseño e implementación 3
    3.1. Diseño del nodo backup 4
    3.2. Experiencias en la implementación 5
4. Resultados de las pruebas 7
5. Conclusiones y trabajo futuro 10
6. Bibliografía 11

1. Introducción

La capacidad de los procesadores se dobla cada 18 meses (Ley de Moore), y las tecnologías en redes de área local están alcanzando cotas que hasta hace bien poco eran sólo posibles es sistemas multiprocesadores. Los discos magnéticos

aumentan su capacidad de almacenamiento a un ritmo similar, sin embargo, la latencia de acceso a los mismos está limitada por la naturaleza mecánica de estos dispositivos. Esta limitación convierte a los discos en el principal cuello de botella de los sistemas actuales, siendo este un problema que se agudizará en el futuro.

Gracias a los avances de los procesadores y de las comunicaciones en redes de área local, es factible desde un punto de vista de rendimiento, usar la memoria de nodos remotos como un disco virtual mucho más rápido que los discos magnéticos. Sin embargo, desde un punto de vista de la fiabilidad del sistema, el principal problema de utilizar la memoria de nodos remotos es que la caída de un nodo afecta a todo el conjunto, aumentando la probabilidad de fallo proporcionalmente con el número de nodos utilizados. Para obtener la fiabilidad en este tipo de sistemas hay que hacer tres consideraciones:

1. El trabajo extra introducido en el cliente debe ser mínimo, ya que es un coste a pagar incluso cuando no hay caídas en el sistema.

2. La memoria extra utilizada debe ser mínima, ya que la memoria reservada para obtener la fiabilidad puede ser utilizada por otros procesos o por el propio sistema operativo.

3. El tiempo de recuperación cuando se produce un fallo debe ser bajo.

Flouris y Markatos trabajaron sobre este problema en [1], y diseñaron e implementaron el Network Ram Disk (NRD) para Linux, el cual conseguía excelentes resultados en comparación con el rendimiento de los discos magnéticos. El problema de la fiabilidad lo solucionan creando la técnica Adaptative Parity Caching. Sin embargo, esta técnica contiene ciertas desventajas que limitan sus posibles aplicaciones, como su uso en sistemas transaccionales, además de consumir recursos en el cliente.

Enhanced Network Ram Disk (ENRD) es un proyecto basado en NRD que soluciona los problemas de éste utilizando otro diseño para obtener la fiabilidad, consiguiendo mejorar el rendimiento y disminuyendo la complejidad en el código. ENRD está basado en la idea de utilizar un nodo remoto como almacenamiento de backup: dicho nodo recibirá todos los bloques de escritura y los guardará a disco magnético, siendo su funcionamiento asíncrono respecto al nodo cliente. Mientras que el trabajo original de Flouris y Markatos era un proyecto de investigación, ENRD busca la posibilidad de usar la memoria remota como una utilidad para los clusters Linux, donde su configuración sea rápida y sencilla y posibilite al administrador una herramienta más para ofrecer al usuario de estos sistemas. Una de las aplicaciones del ENRD es como área swap donde puede ayudar a mejorar el tiempo de ejecución de sistemas limitados en memoria RAM. Es frecuente en grupos de investigación científicos ejecutar aplicaciones secuenciales creados por ellos mismos, no estando dispuestos a un coste extra de tiempo y esfuerzo en paralelizar la aplicación. En estos casos, ENRD ayudaría a mejorar el rendimiento de estas aplicaciones, que suelen necesitar enormes cantidades de memoria RAM, y por consiguiente hacen uso del área swap.

En sistemas transaccionales de gran volumen se usa Solid State Disks como almacenamiento, que es almacenamiento hardware basado en memoria RAM. Mientras que SSD es una solución profesional y más cara, ENRD ofrece un uso más dinámico y la posibilidad de mejorar proporcionalmente a la velocidad de CPU's y tecnologías de comunicaciones.

2. Porqué mejorado: Desarrollos anteriores

La idea de usar la memoria de nodos remotos no es nueva, habiéndose completado en el pasado varios proyectos donde la implementación de prototipos demostró las posibilidades que aparecen en estos sistemas. En especial se estudió la forma de mejorar el rendimiento cuando se hace uso intensivo de grandes cantidades de memoria RAM, lo que obliga a utilizar área swap degradando el rendimiento. En [2] y [3] se detallan dos prototipos diseñados para Digital Unix. El primero explota la posibilidad de Network Ram en todos los niveles del sistema operativo (paginación de memoria virtual, ficheros mapeados en memoria y buffer del sistema de ficheros), consiguiendo la fiabilidad con escrituras asíncronas a disco. En el segundo, se usa Network Ram solo para paginación, estudiando diversas formas de conseguir la fiabilidad, como escrituras asíncronas a disco, mirroring y parity logging.

Los mismos autores de [3] son los creadores del NRD, donde utilizan la misma técnica para obtener la fiabilidad, pero modificada teniendo en cuenta las distintas características entre el acceso a un área swap y a un sistema de ficheros. Esta técnica denominada Adaptative Parity Caching, consigue obtener un mayor rendimiento en comparación con el uso de discos magnéticos. Sin embargo, el uso de una caché en el cliente limita su aplicación en sistemas transaccionales, ya que no garantiza que los bloques escritos se encuentren en almacenamiento fiable en caso de fallo en el cliente. Además, esta cache consume recursos en el cliente (segunda consideración en la introducción) que podrían ser usados por otros procesos o por el sistema operativo: para un Network Ram Disk de 16Gbytes, la caché usaría 80Mbytes. Finalmente, el cálculo de la paridad de bloques lo debe realizar el cliente (primera consideración en la introducción), que aunque despreciable en una simple operación, deja de serlo cuando se ejecuta millones de veces.

Para evitar estas limitaciones diseñamos e implementamos Enhanced Network Ram Disk(ENRD), consiguiendo incluso mejorar las prestaciones del NRD original, además de liberar al código de la complejidad de implementar Adaptative Parity Caching. Para ello, evitamos el uso de memoria en el cliente para obtener la fiabilidad y el coste extra en el calculo de la paridad. En el nuevo diseño, el cliente enviará los bloques de escritura al nodo remoto y a un nuevo nodo denominado backup, mientras que las peticiones de bloques de lectura las enviará únicamente al nodo remoto. El cliente no trabaja con el backup directamente, sino que lo hace con los servidores, salvo que falle alguno de estos. El funcionamiento del backup es asíncrono respecto al cliente, y utiliza varias técnicas para realizar su tarea eficientemente, como unir bloques secuenciales en la misma llamada al sistema (en el siguiente apartado se explica con detalle su implementación). Para evitar que el cliente envíe el mismo bloque dos veces usamos Multicasting[4]: cada nodo remoto pertenece a un grupo Multicast diferente, mientras el nodo backup pertenece a todos. De esta manera el envío de un bloque a una dirección Multicast será recibido por el nodo remoto y por el nodo backup, liberando al cliente de esta tarea y dejándosela a las tecnologías de interconexión de red. En el caso de fallo en un nodo remoto, el cliente enviará un paquete de RECOVERY al nodo backup, el cual copiará de disco a memoria los bloques pertenecientes al nodo fallido y continuará su ejecución realizando las funciones del servidor.

3. Diseño e implementación

En el diseño del NRD el sistema se componía de un nodo cliente y varios nodos remotos (servidores). En el nodo cliente se añadía un módulo al kernel del sistema para poder acceder a un dispositivo de bloque virtual por medio de un driver, el cual se sincronizaba con un daemon(duende) a nivel de usuario. El daemon es el que se encarga de comunicarse con los servidores para la escritura o lectura de bloques. Los servidores escuchaban en dos puertos, uno

de lectura y otro de escritura, y su función era copiar los bloques recibidos a memoria en las operaciones de escritura, y por otra parte enviar los bloques pedidos en las operaciones de lectura. Como ya hemos comentado, la implementación de Adaptative Parity Caching suponía que el daemon consumiera recursos del cliente como la memoria usada para la caché, o tiempo de procesador para calcular la paridad de bloques, añadiendo además cierta complejidad al código.

El diseño de ENRD está basado en el NRD, pero el cliente y los servidores se han liberado de una cierta complejidad al no usar Adaptative Parity Caching, y han sido modificados convenientemente para usar Multicasting. El driver ha sido modificado para evitar posibles condiciones de carrera, utilizando semáforos en vez de las funciones sleep and wake_up para la sincronización entre el driver y el daemon. Se añade un nuevo componente al sistema: el nodo backup. Su función será recibir todos los paquetes de escritura que el cliente envíe a los diferentes grupos Multicast y sustituir a uno de los servidores en caso de caída. El nuevo diseño para conseguir la fiabilidad sitúa la complejidad del código en el nodo backup, aunque en menor medida que las que podían tener el daemon del cliente y los servidores en el NRD cuando se implementaban con la técnica de paridad.

El funcionamiento del daemon en el cliente y de los servidores es sencilla y fácil de implementar, siendo equivalente al NRD original en su versión UNRELIABLE, la cual funcionaba sin garantizar la fiabilidad. Se han añadido las funcionalidades necesarias de recuperación en caso de que el nodo cliente detecte el fallo de un servidor, que funciona mediante un timeout y el envío de un paquete de RECOVERY del cliente al backup. Por otra parte, al usar Multicast nos obliga a usar el protocolo UDP, que implica que no hay garantías en el envío y recepción de los paquetes. Por lo tanto hay que implementar un control de flujo a nivel de aplicación (más adelante explicaremos su implementación). Cabe señalar que existen trabajos sobre Multicast con fiabilidad [5] con el mismo rendimiento que sobre UDP y que podrían ser aplicados con ENRD en el futuro.

A continuación nos centraremos en el funcionamiento del nodo backup, y de las técnicas utilizadas en su diseño, y cómo afectan en él varias limitaciones que encontramos en las pruebas.

3.1. Diseño del nodo backup

Añadir un nuevo componente cuya funcionalidad sea escribir los bloques de escritura a disco magnético es justamente lo que se trata de evitar con el NRD y ENRD, ya que como dijimos anteriormente, la latencia del disco magnético se convierte en el principal escollo a resolver en los actuales y futuros sistemas. Por lo tanto puede parecer una contradicción usarlo dentro de la solución. La clave está en que estas escrituras se realizan asíncronamente respecto al funcionamiento del sistema. Este método es utilizado en [2] pero no de una forma adecuada, ya que son los servidores los que realizan las escrituras a disco. Esto puede ocasionar la pérdida de datos, ya que un fallo puede ocurrir cuando el bloque haya sido copiado a memoria pero aún no haya sido escrito a disco. Además, en una situación de escritura de gran cantidad de bloques, el sistema se vería colapsado, ocasionando que el funcionamiento deje de ser asíncrono. En ENRD al utilizar un nodo de backup nos aseguramos que en caso de caída de un servidor, exista una copia de sus bloques en el backup, lo que garantiza la fiabilidad. Para solucionar el problema de escritura de una gran cantidad de bloques en un corto espacio de tiempo, debemos recordar que para escrituras, mientras un servidor debe copiar el contenido del bloque recibido a memoria, el backup lo debe escribir a disco. Obviamente el backup no lo puede realizar a la misma velocidad (esta es la razón de ser del ENRD) por lo que debe usar varias técnicas que le permitan realizar esta tarea:

1. Unir bloques secuenciales: evitando de esta manera realizar tantas llamadas al sistema como bloques recibidos.

2. Repartir las funciones de recepción de bloques y escritura a disco.

La primera técnica viene determinada por una característica de las operaciones de escritura y que es aprovechada por los discos para mejorar su rendimiento: las escrituras de bloques secuenciales. Si unimos varios bloques cuya posición es consecutiva en el disco, en la misma operación, evitamos que la cabeza del disco tenga que desplazarse y ahorramos la latencia que se produciría. demás, evitamos realizar una llamada al sistema por cada bloque recibido. Para conseguir esto usaremos una cache a nivel de aplicación donde los bloques recibidos serán copiados y donde se intentará unirlos si fuera posible: deben pertenecer al mismo servidor y poseer números de bloques consecutivos. Los bloques serán escritos a disco cuando el número de bloques en la caché del backup lleguen a una cierta cantidad predeterminada. Como ejemplo vamos a tomar una cache de 100 bloques, siendo 10 el límite de bloques en la cache para volcar a disco. Llegan 12 bloques al backup: a6-a7-b3-b8-b9-a1-a2-a3-c6-c7-b1-b2. La letra indica el servidor al que va destinado el bloque y el número es el número de bloque dentro de ese servidor. Cuando llegue el décimo bloque c7, los bloques que estén es ese momento en la caché serán escritos a disco con 5 llamadas al sistema: write(a,-,2) write(b,-,1) write(b,-,2) write(a,-,3) write(c,-,2), donde el primer parámetro es el fichero asociado con el servidor, el segundo en una variable que usará el backup en la implementación y donde se copiará el contenido de los datos de los bloques, y el tercer parámetro es el numero de bloques a escribir(realmente es el número de bytes pero usamos el número de bloques para facilitar la explicación). De esta manera el sistema se evita realizar 5 llamadas al sistema, siendo además seguro que los bloques se escribirán consecutivamente a disco si usamos un sistema de ficheros ext3 o XFS, lo que aumenta el rendimiento total del backup al no tener que desplazarse la cabeza del disco.

La segunda técnica viene dada por el problema de desbordamiento del buffer de comunicaciones que puede ocurrir en determinadas circunstancias. Para ello tenemos que conocer como funcionan las tarjetas de red y la pila de comunicaciones del sistema [6] y [7]. Cuando la tarjeta de red recibe un paquete, lo escribe a unos buffers en memoria aprovechando los canales de DMA (acceso directo a memoria). El sistema operativo se encarga de procesar estos paquetes a través de la pila de comunicaciones, copiando finalmente su contenido a un buffer a nivel de proceso, que es creado cuando se realiza la llamada al sistema para la creación de sockets. Aunque el tamaño de este buffer es configurable y puede ser ampliado, el peligro de desbordamiento sigue existiendo, ya que si llegan paquetes y el buffer está lleno, el sistema los descarta (en este caso el protocolo de transporte de red no tiene nada que hacer, ya que para él los paquetes han llegado sin problemas). Por lo tanto, en caso de que el proceso esté ocupado escribiendo bloques a disco, una llegada masiva de nuevos bloques podría originar el desbordamiento del buffer del socket, perdiendo de esta manera información. La solución está en el famoso ¿divide y vencerás?: usaremos hilos (POSIX threads) para dividir la tarea entre un thread escuchador y un thread escritor. La tarea del escuchador es copiar los nuevos bloques desde el buffer a nivel socket al buffer a nivel de aplicación, y buscar la posibilidad de unir este nuevo bloque con otros que ya estuviesen en el buffer. Mientras, el escritor se sincroniza con el escuchador para escribir los bloques del buffer de aplicación a disco con las mínimas llamadas al sistema, gracias a la labor de unión de bloques del escuchador. Aún en el caso de sistemas monoprocesadores este diseño es eficaz, ya que mientras que el proceso escritor está esperando que finalize la operación de escritura, pueden llegar bloques al sistema, y pueden ser tratados por el escuchador. Si sólo existiera un hilo, sería más fácil que hubiera problemas de desbordamiento del buffer de socket.

3.2. Experiencias en la implementación

Hemos estado hablando del diseño del backup, y de las técnicas usadas para que pueda realizar su tarea sin convertirse en un cuello de botella . Sin embargo, a la hora de la implementación surgieron varias limitaciones que afectaron al diseño descrito.

La primera de ellas es importante aunque no obliga a rediseñar el sistema. Surge por la tecnología de interconexión usada. En las pruebas realizadas usamos un switch Ovislink Fast Ethernet sin soporte real de Multicasting y un switch 3COM que si lo soportaba, ambos de 24 puertos. Para nuestra sorpresa, el 3COM era sumamente lento cuando tenía activada la funcionalidad de Multicast, lo que degradaba el funcionamiento del sistema hasta el punto de ser peor que cuando usamos el disco magnético. El soporte Multicast se basa en almacenar en una tablas internas qué puertos del switch han enviado paquetes de pertenencia a alguna dirección de Multicast. Así, cuando llegue un paquete con este protocolo, el switch usa estas tablas para reenviar el paquete sólo a los puertos adecuados, en vez de enviarlo a cada puerto, que es lo que realiza cuando no tiene dicho soporte. Parece ser, por lo menos en nuestras pruebas, que es más eficiente para el switch enviar el paquete al buffer de salida de cada puerto que leer las tablas y enviarlo sólo a los miembros del grupo dado. Así, tanto el 3COM sin funcionalidad Multicast como el Ovislink conseguían excelentes rendimientos para un número de 7 nodos. A pesar de usar un diseño basado en Multicast en tecnologías sin soporte para ello los resultados son satisfactorios, aunque habría que estudiar el comportamiento con un mayor número de nodos, donde la política de reenvío a cada puerto pueda soponer una sobrecarga evidente.

Una segunda limitación es el comportamiento del switch para escrituras de un gran número de bloques consecutivamente. Si el cliente envía un gran número de peticiones de escritura seguidas, el switch se ve incapaz de manejarlas, lo que origina la pérdida de paquetes dentro del switch. Esto obliga a realizar un control de flujo a nivel de aplicación, cuya implementación era obligado debido al protocolo UDP. El control de flujo implementado esta pensado para evitar la limitación del switch en cuanto al número de paquetes en fila o consecutivos y no como un completo y eficiente control de flujo. Durante los tests efectuados no se perdió ningún paquete, por lo que podría ser suficiente para uso en redes locales (aunque pendiente para un trabajo futuro). La implementación del control de flujo se limita a una labor de sincronismo cada 100 bloques enviados (50 paquetes), consistente en el envío de un paquete SYNC del cliente a cada nodo y la contestación de cada nodo al cliente. Actualmente el paquete de contestación es siempre el mismo, un paquete de OK. Para un control de flujo más eficiente se usaría una pequeña cache en el cliente (unos 100Kbytes), donde el paquete de contestación fuera más complejo, indicando si algún bloque no se recibió y cuál o cuáles de ellos fallaron.

Por último, las técnicas utilizadas en el diseño del backup no son suficientes, perdiendo el backup bloques en situaciones de escrituras masivas. El problema no está en el diseño, sino en la implementación. El thread escuchador, cada vez que llega un nuevo paquete debe buscar que ese bloque no se encuentre ya en el buffer, recorriendo la lista de bloques en el buffer de una forma secuencial. Cuando el tamaño de una lista es mayor que unos pocos elementos, está demostrado que existen otras estructuras de datos más adecuadas, como árboles binarios balanceados AVL[8].Por este motivo, cuando el número de bloques en el buffer es elevado, el escuchador tarda en realizar esta tarea, lo que origina que nuevos bloques lleguen al buffer a nivel socket con el peligro de desbordamiento. Para evitar la pérdida de bloques introducimos un paquete de sincronismo con el backup junto con los paquetes de sincronismo que se envían a los servidores, lo que conlleva a que el funcionamiento del backup respecto al cliente no sea realmente asíncrono, ya que en caso de escritura de un gran número de bloques el paquete de OK mandado desde el backup al cliente sufre un pequeño retraso, introduciendo una pequeña sobrecarga en los resultados. Sin embargo, esta sobrecarga es lo

suficientemente pequeña como para conseguir una eficiecia mejor que el NRD original como veremos en las gráficas de resultados de las pruebas. El coste de búsqueda y de inserción en el buffer puede ser mejorado sustancialmente usando árboles binarios como estructura de datos cuando el número de bloques alcanze un determinado tamaño, al igual que se usa en el kernel de Linux para recorrer el array de páginas.

4. Resultados de las pruebas

ENRD está basado en un trabajo anterior, en el cual se realizaron varias pruebas para comparar su rendimiento respecto al disco. En nuestro trabajo conseguimos mejorar las prestaciones del NRD, haciendo su uso posible en sistemas transaccionales. Por lo tanto, nuestras pruebas están orientadas a comparar el ENRD y el NRD, que obviamente demostrará que ENRD es mucho más eficiente que el disco. Utilizamos la versión más rápida del NRD en la cual no existe fiabilidad, y se comparará al ENRD con fiabilidad activada (el nodo backup funcionando) y al ENRD sin backup. De esta manera veremos cual es la sobercarga que introduce el sincronismo del backup explicado en el apartado anterior. Veremos también cual es el efecto de las técnicas empleadas en el backup que permite al sistema el ahorro de un gran número de llamadas al sistema.

Para la realización de las pruebas se utilizó un programa llamado lutherbench, creado por nosotros. Dicho programa utiliza varios parámetros: número de threads n, tamaño de fichero fs, tamaño de registro rs y número de escrituras wn. Cada thread escribirá un fichero de fs tamaño en bloques de rs bytes un total de wn veces.

Se definieron distintas pruebas con varios tamaños de ficheros y varios tamaños de registro de escritura, lanzando 1, 5, 10 y 20 threads. Cada uno de estos threads escribía 10000 veces cada fichero. Los ficheros son creados con el flag O_SYNC, lo que obliga al sistema a escribir a disco antes de finalizar la llamada de escritura (por eficiencia, sin este flag el sistema escribe en el Buffer Cache del sistema y posteriormente lo hace a disco). De esta manera es como funcionan los sistemas transaccionales.

Se utilizaron 7 nodos biprocesadores Pentium III 1Ghz, 512Mbytes de RAM, tarjetas Intel Ethernet Pro 100, Discos IDE y switch Ovislink 124TH+. En cada uno de los 5 nodos servidores se usó 200Mbytes como parte del ENRD, y se usó el mismo tamaño en el buffer del nodo backup.

- 1 thread, escribiendo ficheros de diferentes tamaños en bloques de 1024 bytes 10000 veces. Los resultados son los tiempos medios de escritura por fichero en milisegundos.

```
Tamaño fichero  Disco NRD ENRD(backup) ENRD


1024  21 0 0  0
2048  41 1 0  0
4096  83 2 0  0
8192  164 5 1  1
16384  328 11 3  3
32768  648 26 8  8
65536   56 18  18
```

Únicamente en esta tabla mostramos el comportamiento del disco, que ya es suficientemente significativa. Vemos como el backup no produce ninguna sobrecarga con tan poca carga.

- 2 thread, escribiendo ficheros de diferentes tamaños en bloques de 1024 bytes 10000 veces. Los resultados son los tiempos medios de escritura por fichero en milisegundos.

```
Tamaño fichero  NRD  ENRD(backup)  ENRD


1024  1  0   0
2048  2  0   0
4096  4  1   1
8192  8  4   2
16384  18  6   6
32768  45  16   15
65536  95  35   33
```

La sobrecarga del backup es pequeño, pero ya afecta al rendimiento del sistema.

- 5 thread, escribiendo ficheros de diferentes tamaños en bloques de 1024 bytes 10000 veces. Los resultados son los tiempos medios de escritura por fichero en milisegundos.

```
Tamaño fichero  NRD  ENRD(backup)  ENRD


1024  3  1   1
2048  5  2   2
4096  9  4   3
8192  19  7   6
16384  43  18   14
32768  108  39   35
65536  238  82   77
```

- 10 thread, escribiendo ficheros de diferentes tamaños en bloques de 1024 bytes 10000 veces. Los resultados son los tiempos medios de escritura por fichero en milisegundos.

```
Tamaño fichero  NRD  ENRD(backup)  ENRD


1024  8  7   7
2048  14  9   8
4096  22  11   10
8192  37  17   15
16384  78  45   29
32768  186  78   67
65536  656  162   146
```

- 20 thread, escribiendo ficheros de diferentes tamaños en bloques de 1024 bytes 10000 veces. Los resultados son los tiempos medios de escritura por fichero en milisegundos.

Tamaño fichero NRD ENRD(backup) ENRD

```
2048  50  40   38
4096  68  45   43
8192  103  54   52
16384  249  145   77
32768  790  211   147
65536  1780  502   272
```

Durante las pruebas, el porcentaje de buffer de aplicación en el backup utilizado no superó nunca el 20% , por lo que se puede ajustar el tamaño del buffer en lugar de usar el mismo tamaño de memoria que en los servidores.

En http://torus.gap.upv.es:8080/enrd/english/results.html se pueden ver tablas para otros tamaños de registro de escritura, así como el resultado de utilizar ENRD con la base de datos Postgresql, simulando transacciones.

5. Conclusiones y trabajo futuro

La limitación de mejora en el rendimiento de los discos magnéticos debido a su naturaleza mecánica los convierten en el principal cuello de botella de los actuales sistemas, y considerando que las velocidades de los procesadores y de las comunicaciones en las redes de área local continuarán aumentando en el futuro, las limitaciones de los disco magnéticos se verán acentuadas los próximos años. Utilizando parte de las causas de este problema, los procesadores y las comunicaciones, podemos aumentar el rendimiento de las aplicaciones que hacen uso intensivo de la entrada/salida a disco, usando para ello la memoria de nodos remotos. De esta manera aprovechamos los ciclos de computación de los procesadores que quedaban sin utilizarse debido a la diferencia de rendimiento entre los distintos componentes del sistema.

En NRD se lograba alcanzar los objetivos descritos, pero con ciertas limitaciones que lo hacían inapropiado para sistemas transaccionales. Además utilizaba demasiados recursos del cliente que podían ser empleados para otros propósitos.

Con ENRD evitamos estas limitaciones al mismo tiempo que conseguimos un mejor rendimiento incluso respecto al NRD sin fiabilidad. Para esto nos basamos en un nodo backup trabajando asíncronamente respecto al cliente, el cual guarda todos los bloques de escritura a disco magnético, utilizando varias ténicas de programación para realizar su tarea eficientemente.

En un trabajo futuro se puede mejorar el rendimiento del sistema ENRD para conseguir una verdadera asincronía entre el cliente y el backup. Su pricipal limitación es la forma de recorrer la lista de bloques en la cache del backup, lo que en estos momentos se realiza de una manera secuencial. Utilizar otras estructuras de datos más apropiadas, como árboles binarios balanceados, cuando el tamaño de una lista es relativamente grande, podría mejorar el rendimiento al disminuir la sobrecarga del backup respecto al cliente. Otro punto de mejora es el control de flujo, el cual no es difícil de implementar y se requeriría para poder usar ENRD en sistemas comerciales. Como hemos comentado, la

posibilidad de usar una versión fiable de Multicast evitaría dicha implementación a nivel de aplicación. Modificar el diseño del ENRD para que sea el propio driver en vez del daemon el que se comunique con los servidores remotos sería interesante y mejoraría el rendimiento (esta idea se ha utilizado en el kernel httpd server).

Por último, un estudio de las posibilidades de ENRD con tecnologías de red más avanzadas sería conveniente, o simplemente con librerías de comunicaciones que eviten la sobrecarga de usar la pila TCP/IP como GAMMA[9].

6. Bibliografía

1. Michael D. Flouris y Evangelos P. Markatos. The Network Ram Disk: using remote memory on heterogeneous NOW's. Cluster Computing: The Journal on Networks, Software and Applications, 2 (4), pp. 281-293,1999, Baltzer Science Publishers.

2. M.J. Feeley, W.E. Morgan, F.H. Pighin, A.R. Karlin, H.M. Levy y C.A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. En Proceedings of the 15th Symposium on Operating Systems Principles, pp. 201-212, diciembre 1995

3. E.P. Markatos y G. Dramitinos. Implementation of a Reliable Remote Memory Pager. En Proceedings of de 1996 Usenix Technical Conference, pp. 177-190, enero 1996.

4. R. Braudes y S. Zabele. Requeriments for Multicast Protocols, FRC 1458, mayo 1993.

5. Alex Koifman y Stephen Zabele. RAMP: A Reliable Adaptative Multicast Protocol. En Proceedings of IEEE INFOCOM, pp. 1442-1451, marzo 1996.

6. Glenn Herrin. Linux IP Networking: A Guide to the Implementation and Modification of the Linux Protocol Stack. http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html

7. http://www.3com.com 3c90xc NICs Technical Reference (3COM). Part Number: 89-0931-000. Publicado en septiembre de 1999.

8. http://www.gnu.org/directory/Software_Libraries/C_libraries/libavl.html

9. http://www.disi.unige.it/project/gamma/