RAID

Introducción

En los últimos años, la mejora en la tecnología de semiconductores ha significado un gran incremento en la velocidad de los procesadores y las memorias principales que, a su vez, exigen que se incremente la velocidad del almacenamiento secundario. Desafortunadamente, mientras las prestaciones de los procesadores RISC han crecido a un ritmo de un 50% al año, los tiempos de acceso de los discos, que dependen de sistemas mecánicos, han mejorado menos de un 10% al año. Los tiempos de transferencia, que siguen las mejoras de los sistemas mecánicos y la densidad de los medios magnéticos, han mejorado aproximadamente un 20% al año, todavía lejos de los procesadores. Esto hace que la diferencia entre la velocidad de los discos y los del procesador se agrande con el tiempo. Además, el incremento de velocidad en los procesadores ha traído nuevas aplicaciones como vídeo, multimedia, etc. que necesitan imágenes grandes. También en la computación científica y el diseño asistido por ordenador se usan cada vez conjuntos de datos mayores.

Una solución a este problema es utilizar matrices de discos (*disk arrays*). Una matriz de discos es un conjunto de varios discos independientes organizados para que el sistema operativo los vea como un solo disco lógico. En 1988 Patterson, Gibson y Katz, un grupo de investigadores de Berkeley, publicaron un artículo [Patterson88] que definió la nomenclatura de las principales organizaciones de matrices de discos y es la referencia básica en este campo. En este artículo, los autores denominaban a las matrices de discos Redundant Arrays of Inexpensive Disks (RAID) para dar idea de que con una matriz de discos económicos se puede conseguir tanto rendimiento (o más) que con un solo disco grande y costoso. Sin embargo, los beneficios del RAID van más allá de simple uso de discos económicos y se ha extendido a todo tipo de discos, por lo que se ha cambiado el significado del término RAID por Redundant Array of **Independent** Disks.

Los sistemas RAID mejoran la velocidad gracias a una técnica denominada *striping*, que consiste en dividir los datos en bandas (*stripes*) que se distribuyen entre los discos que forman la matriz. Si en un momento dado se requieren datos que estén en dos discos distintos, se podrán traer en paralelo, lo que mejorará el tiempo de respuesta.

Esta solución trae sin embargo un nuevo problema: en una matriz sin redundancia, el fallo de un disco significa el fallo de todo el sistema de discos. Por lo tanto, cuanto más discos se tengan más probabilidad hay de que falle el sistema de discos. La solución pasa por utilizar información redundante, de tal forma que aunque falle un disco el sistema pueda seguir funcionando.

Añadir información redundante tiene efectos sobre el rendimiento. Cuando se expliquen las distintas organizaciones de RAID se verán estos efectos. Por otra parte, la información redundante también tiene un coste: los bits que se utilicen para almacenar información redundante no se utilizan para almacenar información útil (datos, desde el punto de vista del usuario). Se denomina sobrecarga (*overhead*) al cociente nº de bytes redundantes / nº de bytes de datos.

Técnicas empleadas

Striping

El *striping* consiste en distribuir los datos transparentemente en varios discos para que parezca un sólo disco rápido. Para realizar la distribución, los datos se agrupan en bandas (*stripes*) que se van distribuyendo por los distintos discos.
El *striping* mejora el rendimiento gracias a que se aprovecha el paralelismo para servir peticiones de E/S. Este paralelismo tiene dos aspectos:

- Se pueden servir peticiones independientes que accedan a distintos discos en paralelo. Esto reduce el tiempo de espera en colas que ven las peticiones de E/S.
- Si los datos de una petición de E/S están distribuidos entre varios discos, se pueden leer varios bloques en paralelo. Esto aumenta las tasas de transferencia.

El tamaño de las bandas es determinante. Hay dos enfoques básicos:

- Grano fino: cada unidad de datos entrelazada es pequeña. Esto hace que una petición de E/S, independientemente de su tamaño, acceda a todos (o casi todos) los discos. Como consecuencia, todas las peticiones de E/S ven mejoradas sus tasas de transferencia. El lado negativo de este enfoque es que no se puede servir más de un petición a la vez y, además, todos los discos pierden tiempo posicionándose.
- Grado grueso: cada unidad de datos entrelazada es relativamente grande. Esto hace que las peticiones de E/S pequeñas accedan sólo a un disco (o a un subconjunto pequeño), mientras que las peticiones grandes siguen accediendo a todos los discos. La ventaja es que se permiten múltiples peticiones pequeñas en paralelo (con tasas de transferencia menores que en el caso anterior) a la vez que para las peticiones grandes se siguen obteniendo elevadas tasas de transferencia.

Redundancia

La redundancia permite aumentar la fiabilidad del sistema de discos. A la hora de seleccionar cómo se introduce la información redundante hay dos problemas que solucionar:

- Elegir el método para calcular la información redundante. El *mirroring* simplemente copia la información según está[1]. La forma más común es utilizar paridad calculada mediante XOR, aunque algunas organizaciones de RAID utilizan códigos Hamming o Reed-Solomon.
- Elegir la forma de distribuir la información redundante en el array. Hay muchos esquemas, pero simplificando se pueden clasificar en dos: los que distribuyen la información redundante en pocos discos (típicamente sólo uno) y los que la distribuyen uniformemente entre todos los discos. Estos segundos suelen ser mejores ya que evitan cuellos de botella y problemas de balanceo de carga.

[1] También se habla de una técnica denominada *duplexing* que es igual que el *mirroring* pero replicando además de los discos las controladoras.

Niveles RAID

En el [Patterson98] se definían 5 formas distintas de organizar las matrices de disco que se denominaron "niveles RAID", del 1 al 5. Hay que tener en cuenta que a pesar de que el uso de la palabra "nivel" parece implicar una jerarquía, esto no es así; es decir, el nivel 5 no es mejor que el 1: tiene ventajas en algunos aspectos y desventajas en otros. Por lo tanto, la elección de un nivel u otro involucrará un **compromiso** entre las ventajas y las desventajas, teniendo que elegir, por ejemplo, entre mayor fiabilidad y menor rendimiento.
Tras la definición de estos 5 niveles, se ha extendido el uso de otros dos: el 0 y el 6. Además se han desarrollado organizaciones que usan varios niveles combinados.

RAID 0: No redudante

Este tipo no es estrictamente RAID, ya que no incorpora información redundante, sino solamente *striping*; pero como es la base de los otros sistemas resulta interesante estudiarlo. Simplemente, distribuye los datos en bandas a lo largo de varios discos. Como ventajas, es el sistema más barato (ya que tiene sobrecarga cero) y tiene un rendimiento muy elevado, al no necesitar ni leer ni escribir la información redundante en cada petición de E/S. La gran desventaja de este sistema es su baja fiabilidad:

$$MTTF_{array} = \frac{MTTF_{disco}}{N}$$

Siendo N el número de discos del array. Por ejemplo, si se tiene un $MTTF_{disco}$ de 10 000 horas (es decir, 1.14 años), en un array de 6 discos el $MTTF_{array}$ será de sólo 1 666.67 horas (0.19 años).

RAID 1: En espejo

En esta organización, cada vez que se escribe un bloque en un disco de datos, también se escribe en un disco redundante, con lo que se tienen dos copias de toda la información. Por lo tanto, la sobrecarga es del 100%, la más alta de todas las configuraciones. Como ventaja, es muy fácil reconstruir el array si falla un disco: simplemente hay que copiar la información de su disco espejo.
La velocidad de RAID 1 en lectura puede ser mayor que la de RAID 0 ya que se puede leer la información de dos discos a la vez y escoger el que menos tarde o se pueden dividir peticiones que vayan al mismo disco entre él y su disco espejo. En escrituras el tiempo será un poco mayor porque hay que hacer dos escrituras, que aunque vayan en paralelo requerirán el doble de tiempo en la controladora (suponiendo que sea única). Sin embargo, este tiempo es despreciable comparado con el de posicionamiento y transferencia.
El MTTF es:

$$MTTF_{array} = \frac{MTTF_{disco}^2}{2N * MTTR_{disco}}$$

con N el número de discos (sin incluir los de espejo). El MTTR es el tiempo medio de reparación. Por ejemplo, si se tiene, como antes, un $MTTF_{disco}$ de 10 000 horas, un MTTR de 5 horas y 6 discos, tendremos un N=3 y un $MTTF_{array}$ de 3 333 333.33 horas (380.52 años).

RAID 2: ECC estilo memoria

Los sistemas de memoria tienen sistemas de recuperación muchos más baratos (es decir, con menos sobrecarga) que el *mirroring* gracias a los códigos Hamming. Estos códigos tienen información de paridad para distintos grupos de datos que se sobreponen. En un esquema RAID que emplea esta característica se necesitan 3 discos redundantes para 4 discos de datos, uno menos que en RAID 1. El número de discos redundantes es proporcional al logaritmo del número de discos totales en el sistema, con lo que a mayor número de discos, disminuye la sobrecarga.
La gran ventaja de los códigos Hamming sobre almacenar simplemente un bit de paridad es que, además de detectar fallos de 1 bit, permiten arreglarlos. En memorias, un esquema con un sólo bit de paridad permite detectar fallos pero no arreglarlos, ya que no se sabe qué bit es el incorrecto. En matrices de discos, habitualmente cuando falla un disco habitualmente se sabe qué disco es y, por lo tanto, sólo con la paridad se puede corregir el error. Esta razón, junto con la constatación de que los discos ya incluyen internamente ECC, hace que no existan implementaciones comerciales de este nivel.

RAID 3: Paridad entrelazada a nivel de bit

Aprovechando que en los sistemas de discos se puede saber cuándo un disco ha fallado, este nivel emplea un sólo bit de paridad para almacenar la información redundante. Las bandas en RAID 3 tienen tamaño 1 bit y en un disco extra se almacenan los bits de paridad para cada banda. Con este esquema se puede tolerar el fallo de un disco.
Las peticiones de E/S acceden a todos los discos, así que no se pueden servir varias peticiones a la vez.

RAID 4: Paridad entrelazada a nivel de bloque

Este esquema es similar al anterior excepto en que el tamaño de las bandas ya no es de 1 bit, sino de uno o más bloques (siendo el bloque la unidad básica de acceso al disco). Las peticiones de lectura con tamaño inferior al bloque sólo accederán a un disco, con lo que se podrán atender varias a la vez. Las peticiones de escritura, además de acceder a los bloques escritos deben calcular el bit de paridad y acceder al disco de paridad, que se convierte fácilmente en un cuello de botella. Es por ello que no se suele utilizar porque no tiene ninguna ventaja sobre RAID 5.

RAID 5: Paridad entrelazada a nivel de bloque y distribuida

Este sistema es similar al anterior pero, en lugar de agrupar todos los bloques de paridad en un solo disco, los distribuye entre todos los discos. Además de eliminar el cuello de botella, esta organización tiene una ventaja extra: en un lectura participará un disco más (el que antes se dedicaba a paridad que ahora también contiene datos).
La forma en que se distribuya la paridad afecta al rendimiento. La mejor estudiada hasta el momento es la simétrica a la izquierda (*left-simetric*).
Este nivel tiene el mejor rendimiento en lecturas pequeñas y grandes y escrituras grandes de todos los niveles redundantes. Sin embargo, en escrituras pequeñas tiene un rendimiento menor que otras organizaciones. Esto es debido a lo siguiente: en una escritura de una petición grande que acceda a bloques en todos los discos, para calcular la paridad simplemente se hace el XOR de todos los bloques nuevos. Sin embargo, en una lectura pequeña que sólo acceda a un bloque, para calcular la paridad habría que leer todos los bloques viejos para hacer el XOR, con lo cual una petición de escritura de 1 bloque se estaría convirtiendo en una petición de lectura de muchos bloques. Una solución mejor es la siguiente:

1) Leer el bloque viejo. Comparar los datos viejos y los nuevos para determinar qué bits cambian.
2) Leer la paridad vieja. Con esto y la información del paso anterior se puede saber cuál debe ser la nueva paridad.
3) Escribir la paridad vieja.
4) Escribir los bloques nuevos.

Por lo tanto, sólo se necesitan 4 accesos, aunque sigue siendo peor que RAID 1.
Otra desventaja de este nivel con respecto al RAID 1 es que es más difícil reconstruir el RAID cuando se ha producido un fallo.
El MTTF es:

$$MTTF_{array} = \frac{MTTF_{disco}^2}{N * (N+1) * MTTR_{disco}}$$

Con N el número de discos de datos. Por ejemplo, si se tiene, como antes, un $MTTF_{disco}$ de 10 000 horas, un MTTR de 5 horas y 6 discos (5 con datos y 1 con información de redundancia, aunque en realidad los 6 discos tienen datos e información de redundancia), tendremos un $MTTF_{array}$ de 666 666.67 horas (76.1 años).

RAID 6: Paridad P+Q

La redundancia mediante paridad es capaz de corregir fallos de 1 disco. Si el número de discos en un RAID es grande, empieza a crecer la probabilidad de que fallen dos discos (o más) a la vez. Por ejemplo, reconstruir un RAID 5 puede ser una operación que lleve tiempo y puede ocurrir que durante ese tiempo falle otro disco. Si el fallo se diese durante la operación de reconstrucción, se perderían los datos. Por eso se ha desarrollado este nivel, que tiene niveles de corrección más elevados y que permite el fallo simultáneo de dos discos. Este esquema utiliza el código Reed-Solomon, que consigue el objetivo anterior utilizando sólo un disco redundante más que el RAID 5. El rendimiento, sin embargo, es peor.

RAID de varios niveles

Se han desarrollado soluciones que utilizan una combinación de varios niveles de RAID, intentado combinar las ventajas de cada uno de los niveles empleados. Las más comunes[2] son RAID 0+1 y RAID 1+0.
RAID 0+1 utiliza primero RAID 0 para crear un array con un rendimiento muy elevado pero sin redundancia. A continuación crea un espejo de este array, es decir, hace RAID 1 para obtener redundancia. Por ejemplo, con 8 discos, se podrían hacer dos RAID 0 de 4 discos, y el conjunto de estos dos RAID 0 funcionaría como un RAID 1. En RAID 0+1, si falla uno de los discos de un RAID 0, ese array se pierde, pero el otro sigue funcionando y el array RAID 0+1 no fallará mientras no ocurra un nuevo fallo en el segundo RAID 0.
RAID 1+0 aplica primero RAID 1 y luego RAID 0. Por ejemplo, con 8 discos, se podrían hacer 4 arrays RAID 1, es decir, 4 conjuntos de dos discos haciendo de espejo. A continuación, se distribuyen los datos mediante *stripping* entre estos RAID 1. Esta combinación tiene una mayor tolerancia a fallos que RAID 0+1: mientras funcione uno de los discos en cada array RAID 1, el RAID 1+0 estará funcionando.

[2] A veces los productos comerciales no utilizan de manera consistente estas denominaciones. Incluso a veces se usa la denominación RAID 10.

Resumen

Nivel	**Rendimiento**	**MTTF**	**Nº mínimo de discos**	**Overhead**
RAID 0	El mejor en escr.	$\frac{MTTF_{disco}}{N}$	2	0%
RAID 1	El mejor en lect.	$\frac{MTTF_{disco}^2}{2N * MTTR_{disco}}$	2	100 %
RAID 2	No existe comercialmente			
RAID 3	Alto para entornos monousuario		3	N/N+1
RAID 4			3	N/N+1
RAID 5	Alta en lect. Media en escr. Baja mucho cuando hay fallo	$\frac{MTTF_{disco}^2}{N * (N+1) * MTTR_{disco}}$	3	N/N+1
RAID 6			4	N/N+2
RAID 0+1			4	100 %
RAID 1+0			4	100 %

Siendo N el número de discos de datos del RAID.