BSD – Berkeley Software Distribution

En el fs de Unix con el cual se venía trabajando, cada drive esta dividido en uno o mas particiones, cada una de estas particiones podían contener un file system. Pero un fs no se expande a traves de varias particiones. Un fs esta descripto por el Superblock, que contiene informacion basica del mismo.

Dentro del fs encontramos archivos, y el acceso a los mismos normalmente incurre en una larga busqueda desde el inodo de un archivo hasta los datos del mismo. Los archivos de un directorio no estan asignados en bloques consecutivos de inodos, causando que muchos bloques que no estan contiguos deban ser accedidos cuando se ejecutan operaciones en inodos de muchos archivos en un mismo directorio.

La asignacion de bloques a los archivos es causante de un limitado throughput del fs. El fs no transmite mas de 512 bytes por transaccion y suele encontrarse con que la proxima lectura no se encontrará en el próximo cilindro, forzando a la realización de seeks.

Para mejorar esto se cambio el tamaño del bloque, haciendo que la transferencia de datos sea el doble y que muchos archivos pudiesen ser accedidos sin la necesidad de leer bloques de indireccion.

Un problema que aun persistía era que la lista de bloques libres pronto se convertía en una lista con bloques dados en forma random, haciendo que los bloques de un archivo no estuviesen hubicados en forma estratégica para mejorar las lecturas.

En el fs llamado **BSD**, que es una mejora al System V, cada drive contiene uno o mas fs. Los fs son descriptos por los superblocks, localizados al comienzo de cada particion de los fs. Debido a que los superblocks contienen informacion critica, se replican para proteccion en caso de perdidas o fallas. Esto se hace cuando el fs se crea.

Para asegurar la posibilidad de crear archivos grandes sin tener que utilizar mas de dos niveles de indireccion, el minimo tamaño del bloque de un fs es 4096 bytes (o >= 4096 bytes en potencia de dos). El tamaño del bloque de un fs se graba en el superblock, entonces es posible que fs con tamaños distintos de bloques puedan ser accedidos en el mismo sistema. El tamaño solo puede ser decidido al crear el fs y no se puede modificar.

Este nuevo fs divide la particion en una o mas áreas llamadas **grupos de cilindros**. Un grupo de cilindro es un numero de cilindros consecutivos en el disco. Asociado a cada grupo de cilindro encontramos informacion que incluye espacio para inodos, un bit map describiendo bloques dentro del cilindro disponibles, e informacion acerca del uso de los bloques dentro del cilindro, ademas de una copia del superblock. El bit map de bloques disponibles dentro del grupo de cilindros reemplaza la tradicional lista de bloques libres del superblock. Para cada grupo de cilindros un numero definido de inodos se asigna en la creacion del fs. La política que se sigue determina alocar un inodo por cada 2048 bytes de espacio en el grupo.

La informacion de cada grupo de cilindros podría ser guardada siempre al comienzo de cada uno. Sin embargo, si se siguiese esta idea, toda la informacion crucial, quedaria siempre guardada en el primer plato, y si este sufriese algun daño, se perderìa toda la informaciòn. Por eso, la informacion se guarda en distintos offsets dentro de cada

grupo, en general se guarda como en espiral con respecto de todos los grupos de cilindros.

Se presenta un problema con respecto al tamaño de los bloques (4096 bytes): la mayoría de los archivos en UNIX son archivos pequeños. De esta manera, al asignar un bloque de 4096 bytes a un archivo pequeño, se esta desperdiciando mucho espacio.

Entonces para poder seguir utilizando bloques grandes que mejoran la transferencia de datos y aceleran los tiempos de lectura, sin sufrir las consecuencias del desperdicio de espacio, se divide al bloque en **fragmentos** (cada uno de los cuales es direccionable). El mapa de bloques asociado a cada grupo de cilindros guarda el espacio disponible basandose en fragmentos. Para decidir si un bloque esta disponible, se analizan sus fragmentos.

Entonces es posible guardar dentro de un bloque, fragmentos de distintos archivos, pero un archivo no puede tener fragmentos en distintos bloques.

Cada vez que se escriben datos a un archivo, el sistema chequea si el tamaño del archivo ha crecido. Si el archivo necesita ser expandido para albergar mas informacion, una de estas tres condiciones existe:

1) Hay suficiente espacio en el bloque o fragmento asignado para albergar la nueva información. Esta es guardada en el espacio disponible.
2) El archivo no contiene fragmentos y el ultimo bloque del mismo no tiene suficiente espacio para albergar la nueva informacion. Entonces se completa el bloque con parte de la nueva informacion, si lo que queda por almacenar alcanza para llenar un bloque entero, entonces se aloca un bloque y se almacena la informacion. Se continua asi hasta que la informacion ocupe menos de un bloque. Entonces un bloque con los fragmentos necesarios se asigna y se guarda la informacion.
3) El archivo contiene uno o mas fragmentos y estos no contienen suficiente espacio para albergar la nueva informacion. Si el tamaño de los datos en los fragmentos mas el tamaño de la nueva informacion superan la capacidad de un bloque entonces se asigna un bloque y se guardan alli los datos de los fragmentos mas la nueva informacion. Y luego se continua como en el punto 2). Si la suma de informacion no ocupa mas de un bloque, entonces se asigna un bloque y se almacenan los fragmentos necesarios para albergar la informacion.

Para poder realizar esto, se necesita que el fs no se llene. Para cada fs existe un parametro que determina el porcentaje minimo de bloques libres que deben existir. Si este minimo no existiese entonces el administrador es el unico capaz de poder continuar asignando bloques. Este porcentaje puede ser modificado.

Un logro de este nuevo fs es la capacidad que tiene de poder parametrizar las aptitudes y caracteristicas de almacenamiento para que los bloques puedan ser asignados en una configuracion optima. (velocidad del procesador, capacidad de transferencia de datos, etc).

Este fs trata de asignar nuevos bloques en el mismo cilindro del bloque anterior de un archivo. Y gracias a los parametros del hardware, se ubicarán convenientemente según el tiempo rotacional. Se trata tambien de distribuir informacion que no se relaciona a traves de los diferentes grupos de cilindros.

Los inodos de archivos dentro de un mismo directorio son frecuentemente accedidos en forma grupal. Se trata entonces de ubicar los inodos de archivos de un mismo directorio en el mismo grupo de cilindro. Para asegurarse de que los archivos sean distribuidos a traves del disco, una politica diferente se sigue. Un nuevo directorio se va a ubicar en el grupo de cilindros que tenga mas cantidad de inodos libres y la menor cantidad de directorio en el.

Como los bloques de un mismo archivo en general se acceden en forma conjunta, la policita establece que trate de ubicar los bloques del archivo en el mismo grupo de cilindros, preferentemente en una posicion rotacional conveniente en el mismo cilindro. El problema que se presenta es que los archivos grandes van a ocupar rapidamente el espacio en un grupo de cilindros, y esto hará que tanto el archivo como otros archivos tengan que ubicar sus datos en otros grupos. Para evitar esto, y hacer que los grupos de cilindros traten de nunca estar llenos, se sigue la siguiente politica: una vez que un archivo exceda los 48 bytes, se cambiarà de grupo de cilindros y se volverá a cambiar una vez que llegue al mb de datos. El nuevo grupo de cilindros será elegido de aquellos que tengan el mayor numero de bloques libres.

A la hora de asignar un bloque a un archivo y no encontrar el proximo bloque disponible, se pueden dar las siguientes situaciones:

1) Se elige el proximo bloque rotacionalmente cercano y disponible en el mismo cilindro.
2) Si no hay bloques disponibles en el mismo cilindro, se usa un bloque dentro del mismo grupo de cilindros.
3) Si el grupo de cilindros esta completamente lleno, se hashea el nro de cilindro para elegir otro grupo de cilindros y buscar un bloque libre.
4) Finalmente, si el hash falla, se aplica una busqueda exhaustiva en todos los grupos de cilindros.

Simbolic links

Se implementa como un archivo que contiene un path. Cuando el sistema encuentra un symbolic link mientras esta interpretando un componente de un pathname, el contenido del symbolic link es anexado al resto del path, y este nombre es interpretado para obtener el path resultante.
Con la incorporacion de los symbolic links, se resuelve el problema de referenciar inodos que se encuentran en otros fs, por ejemplo.