El Gestor de Volúmenes vinum

Tabla de contenidos

1. Sinopsis

Independientemente del tipo de disco, siempre hay problemas potenciales. Los discos pueden ser muy pequeños, muy lentos, o muy poco fiables para cumplir con los requisitos del sistema. Aunque los discos crecen, también lo hacen los requisitos de almacenamiento. A menudo se necesita un sistema de ficheros que es mayor que la capacidad del disco. Se han propuesto e implementado varias soluciones a estos problemas.

Un método es mediante el uso de múltiples, y a veces redundantes, discos. Además de soportar varias tarjetas y controladores hardware de sistemas de Arrays Redundantes de Discos Independientes `RAID` , el sistema base de FreeBSD incluye el gestor de volúmenes vinum , un controlador de dispositivos de bloques que implementa unidades virtuales de disco y que aborda estos tres problemas. vinum proporciona más flexibilidad, rendimiento, y fiabilidad que el almacenamiento de disco tradicional e implementa los modelos `RAID`-0, `RAID`-1, y `RAID`-5 , tanto individualmente como combinados.

Este capítulo proporciona una visión general de los problemas potenciales del almacenamiento tradicional en disco, y una introducción al gestor de volúmenes vinum.

Comenzando con FreeBSD 5, vinum ha sido reescrito para adaptarlo a la GEOM architecture, a la vez que se mantienen las ideas originales, terminología, y metadata en disco. Esta reescritura se llama *gvinum* (por *GEOM vinum*). Aunque este capítulo utiliza el término vinum, cualquier invocación de comando se debe realizar con `gvinum`. El nombre del módulo del kernel ha cambiado del original vinum.ko a geom_vinum.ko, y todos los nodos de dispositivo residen bajo /dev/gvinum en lugar de /dev/vinum. A partir de FreeBSD 6, la implementación original de vinum no está disponible en el código base.

2. Cuellos de botella de acceso

Los sistemas modernos necesitan frecuentemente acceder a los datos de una forma altamente concurrente. Por ejemplo, grandes servidores de FTP o HTTP pueden mantener miles de sesiones concurrents y tener conexiones de 100 Mbit/s al mundo exterior, mucho más que la tasa de transferencia mantenida de la mayoría de los discos.

Las unidades de disco actuales pueden transferir datos secuencialmente hasta unos 70 MB/s, pero este valor tiene poca importancia en un entorno en el que muchos procesos independientes acceden a un dispositivo, y donde solo pueden conseguir una fracción de esos valores. En tales casos, es más interesante ver el problema desde el punto de vista del subsistema de disco. El parámetro importante es la carga que una transferencia supone en el subsistema, o el tiempo para el cual la transferencia ocupa los dispositivos involucrados en la misma.

En cualquier transferencia de disco, la unidad debe primero posicionar el cabezal, esperar a que el primer sector pase bajo la cabeza de lectura, y después realizar la transferencia. Estas acciones pueden considerarse atómicas y que no tiene sentido interrumpirlas.

Considera una transferencia típica de unos 10 kB: la generación actual de discos de alto rendimiento pueden situar los cabezales en unos 3.5 ms de media. Los discos más rápidos giran a 15,000 rpm, así que la latencia rotacional media (media revolución) es 2 ms. A 70 MB/s, la propia transferencia tarda unos 150 µs, casi nada comparado con el tiempo de posicionamiento. En ese caso, la tasa efectiva de transferencia cae hasta un poco más de 1 MB/s y depende claramente del tamaño de la transferencia.

La solución obvia y tradicional a este cuello de botella es "más agujas": en lugar de usar un gran disco, usar varios discos pequeños con el mismo espacio de almacenamiento agregado. Cada disco es capaz de posicionar y transferir de forma independiente, así que el rendimiento efectivo aumenta en un factor próximo al número de discos utilizados.

La mejora de rendimiento real es menor que el número de discos involucrados. Aunque cada dispositivo es capaz de transferir en paralelo, no hay forma de asegurar que las peticiones se distribuyen de forma equilibrada entre los dispositivos. Inevitablemente la carga en un dispositivo será mayor que en otro.

El reparto equitativo de la carga en los discos depende fuertemente de la forma en la que los datos se comparten entre los dispositivos. En la siguiente discusión, es conveniente pensar en el almacenamiento de disco como un número grande de sectores que son direccionables mediante un número, parecido a las páginas de un libro. El método más obvio es dividir el disco virtual en grupos de sectores consecutivos del tamaño de los discos físicos individuales y almacenarlos de este modo, como coger un libro grande y romperlo en secciones pequeñas. Este método se llama *concatenación* y tiene la ventaja de que los discos no requieren tener ninguna relación específica de tamaño entre ellos. Funciona bien cuando los accesos al disco virtual se reparten equitativamente en su espacio de direcciones. Cuando el acceso se concentra en un área más pequeña, la mejora es menos acentuada. Organización Concatenada ilustra una secuencia en la que las unidades de almacenamiento están asignadas en una organización concatenada.

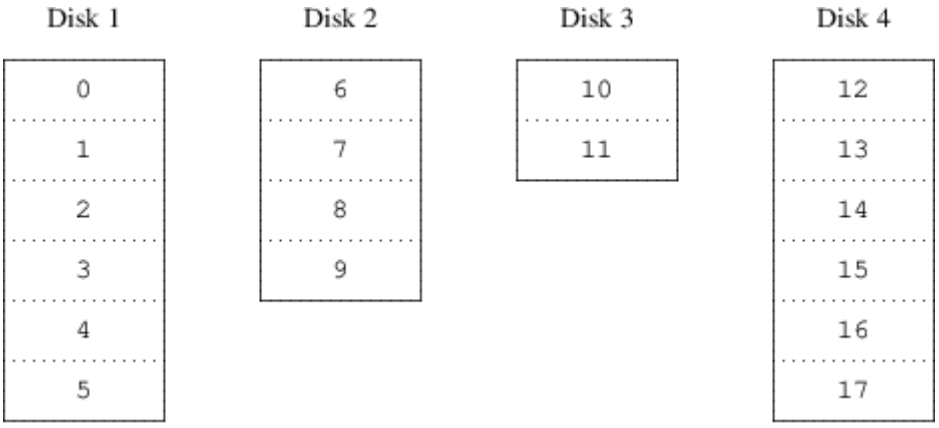

Figura 1. Organización Concatenada

Un mapeo alternativo es dividir el espacio de direcciones en componentes más pequeños del mismos tamaño y almacenarlos secuencialmente en distintos dispositivos. Por ejemplo, los primeros 256 sectores podrían ser almacenados en el primer disco, los 256 sectores siguientes en el siguiente disco, etc. Después de rellenar el último disco, el proceso se repite hasta que los discos están llenos. Este mapeo se llama *segmentado* o `RAID-0`.

`RAID` ofrece varias formas de tolerancia a fallos, aunque `RAID-0` es algo engañoso ya que no provee redundancia. El segmentado requiere algo más de esfuerzo para localizar el dato, y puede ocasionar carga de E/S adicional cuando una transferencia está repartida por múltiples discos, pero también puede proporcionar una carga más constante entre los discos. Organización Segmentada ilustra la secuencia en la que las unidades de discos están asignadas en una organización segmentada.

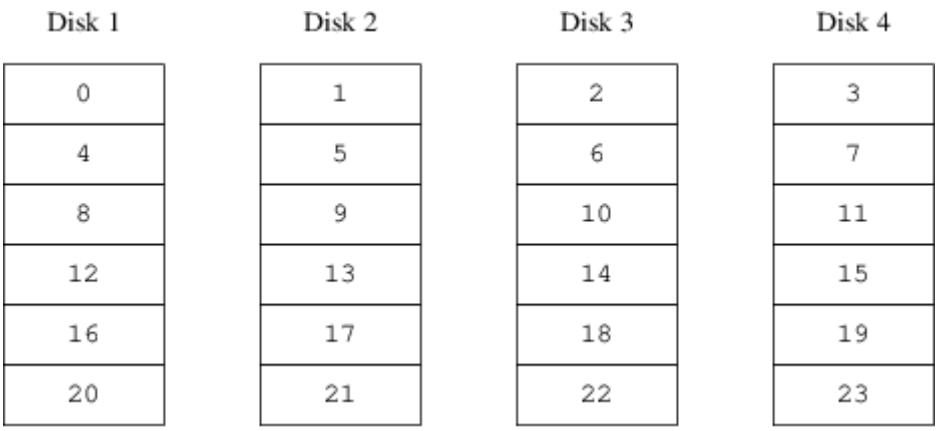

Figura 2. Organización Segmentada

3. Integridad de los Datos

El último problema con los discos es que no son fiables. Aunque la fiabilidad se ha incrementado tremendamente en los últimos años, las unidades de disco todavía son el componente central de un servidor más propensos a fallar. Cuando lo hacen, los resultados pueden ser catastróficas y reemplazar una unidad de disco que ha fallado y restaurar los datos puede resultar en tiempo de parada del servidor.

Una aproximación a este problema es el *mirroring*, o `RAID-1`, que mantiene dos copias de los datos en distinto hardware físico. Cualquier escritura en el volumen escribe en ambos discos; una lectura puede ser resuelta por cualquiera, así que si un disco falla, los datos todavía están disponibles en la

otra unidad.

La configuración en espejo tiene dos problemas:

- Requiere el doble de espacio de almacenamiento que una solución sin redundancia.
- Las escrituras deben realizarse en las dos unidades, de forma que utilizan el doble de ancho de banda que un volumen sin espejo. Las lecturas no sufren penalización en lectura e incluso pueden ser más rápidas.

Una solución alternativa es la *paridad*, implementada en los niveles RAID 2, 3, 4 y 5. De estos, RAID-5 es el más interesante. Como está implementado en vinum, es una variante en una organización segmentada que dedica un bloque en cada segmento para guardar la paridad de los otros bloques. Tal como está implementado en vinum, un RAID-5 plex es similar a un plex segmentado, excepto que implementa RAID-5 al incluir un bloque de paridad en cada segmento. Según lo requerido por RAID-5, la localización de este bloque de paridad cambia de un segmento al siguiente. Los números en los bloques de datos indican los números de bloque relativos.

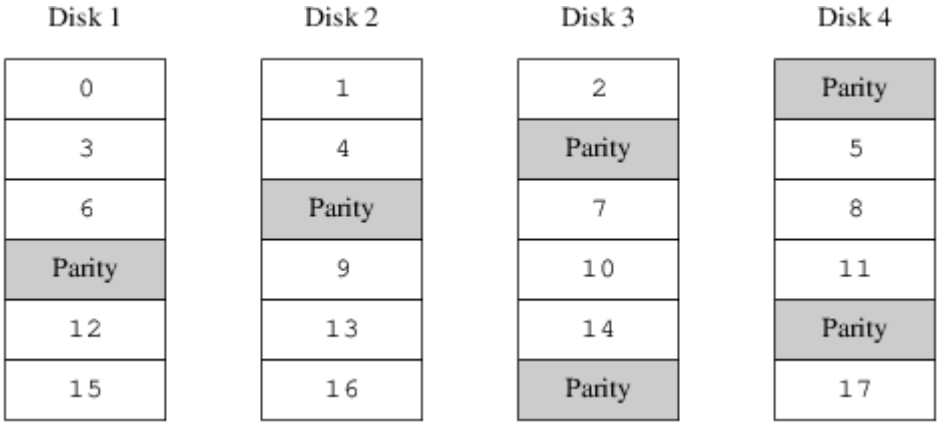

Figura 3. Organización RAID-5

Comparado con la configuración en espejo, RAID-5 tiene la ventaja de que requiere mucho menos espacio de almacenamiento. El acceso de lectura es similar a una organización segmentada, pero el acceso de escritura es significativamente más lento, aproximadamente el 25% del rendimiento de lectura. Si una unidad falla, el array puede seguir operando en un modo degradado donde una lectura de una de las unidades que quedan accesibles continua normalmente, pero una lectura de una unidad que ha fallado es recalculada a partir de los bloques correspondientes del resto de unidades.

4. Objetos vinum

Para abordar estos problemas, vinum implementa una jerarquía de objetos en cuatro niveles:

- El objeto más visible es el disco virtual, llamado *volumen*. Volúmenes tienen esencialmente las mismas propiedades que una unidad de disco UNIX® aunque hay algunas pequeñas diferencias. Una de ellas, que no tienen limitación de tamaño.
- Los volúmenes se componen de *plexes*, cada uno de los cuales representa el espacio de direcciones total de un volumen. Este nivel en la jerarquía proporciona redundancia. Piensa en los plexes como en discos individuales en un array replicado en espejo, cada uno conteniendo los mismos datos.

- Puesto que vinum existe dentro del framework de almacenamiento de disco de UNIX®, sería posible utilizar particiones UNIX como bloques de construcción para plexes multi-disco. En realidad, esto resulta demasiado poco flexible ya que los discos UNIX sólo pueden tener un número limitado de particiones. En su lugar, vinum subdivide una única partición UNIX, la *drive*, en areas contiguas llamadas *subdiscos*, los cuales se utilizan como bloques de construcción de plexes.
- Los subdiscos residen en *unidades* vinum, actualmente particiones UNIX®. Las unidades vinum pueden tener cualquier número de subdiscos. Con la excepción de una pequeña área al comienzo de la unidad, que es utilizada para almacenar información de estado y configuración, la unidad entera está disponible para almacenamiento de datos.

Las secciones siguientes describen el modo en el que estos objetos proporcionan la funcionalidad requerida por vinum.

4.1. Consideraciones sobre el Tamaño del Volumen

Plexes pueden incluir múltiples subdiscos repartidos por todas las unidades en la configuración de vinum.. Como resultado, el tamaño de una unidad individual no limita el tamaño de un plex o un volumen.

4.2. Almacenamiento de Datos Redundante

vinum. implementa configuraciones en espejo adjuntando varios plexes a un volumen. Cada plex es una representación de los datos en un volumen. Un volumen puede contener entre uno y ocho plexes.

Aunque un plex representa los datos completos de un volumen, es posible que algunas partes de esta representación falten físicamente, bien por diseño (no definiendo un subdisco para algunas partes del plex) o por accidente (como resultado del fallo de una unidad). Mientras un plex pueda proveer los datos para el rango completo de direcciones del volumen, este es plenamente funcional.

4.3. ¿Qué Organización Plex?

vinum. implementa tanto concatenación como segmentado a nivel de plex:

- Un *plex concatenado* usa el espacio de direcciones de cada subdisco por turnos. Los plexes concatenados son los más flexibles ya que pueden contener cualquier número de subdiscos, y los subidscos pueden ser de distintas longitudes. El plex se puede extender añadiendo subdiscos adicionales. Requieren menos tiempo de `CPU` que los plexes segmentados, aunque la diferencia en sobrecarga de `CPU` casi no se nota. Por otro lado, son más susceptibles a los puntos calientes, en donde un disco está muy activo y los otros ociosos.
- Un *plex segmentado* reparte los datos entre los distintos subdiscos. Los subdiscos deben tener todos el mismo tamaño y debe haber al menos dos subdiscos para poder distinguirlo de un plex concatenado. La mayor ventaja de los plexes segmentados es que reducen los puntos calientes. Escogiendo un tamaño óptimo de segmento, alrededor de 256 kB, la carga se puede repartir equitativamente entre las unidades. Extender un plex añadiendo nuevos subdiscos es tan complicado que vinum no lo implementa.

vinum Organizaciones Plex resume las ventajas y desventajas de cada organización plex.

Tabla 1. vinum Organizaciones Plex

Tipo Plex	Subdiscos mínimos	Puede añadir subdiscos	Debe tener el mismo tamaño	Aplicación
concatenado	1	sí	no	Gran almacenamiento de datos con máxima flexibilidad de disposición y rendimiento moderado
segmentado	2	no	sí	Alto rendimiento combinado con alto acceso concurrente

5. Algunos Ejemplos

vinum mantiene una *base de datos de configuración* la cual describe los objetos que son conocidos a un sistema individual. Inicialmente, el usuario crea la base de datos de configuración a partir de uno o más ficheros de configuración utilizando gvinum(8). vinum almacena una copia de su base de datos de configuración en cada *dispositivo* de disco bajo su control. Esta base de datos se actualiza con cada cambio de estado, de forma que un reinicio restablece de forma precisa el estado de cada objeto de vinum.

5.1. El Fichero de Configuración

El fichero de configuración describe objetos de vinum individuales. La definición de un volumen sencillo podría ser:

```
drive a device /dev/da3h
volume myvol
  plex org concat
    sd length 512m drive a
```

Este fichero describe cuatro objetos vinum:

- La línea *unidad* describe una partición de disco (*unidad*) y su localización relativa al hardware que hay debajo. Se le da el nombre simbólico *a*. La separación entre los nombres simbólicos y los nombres de dispositivo permite mover discos de un lugar a otro sin confusión.
- La línea *volumen* describe un volumen. El único atributo requerido es el nombre, en este caso *myvol*.

- La línea *plex* define un plex. El único parámetro requerido es la organización, en este caso *concat*. El nombre no es necesario ya que el sistema genera uno automáticamente a partir del nombre del volumen añadiendo el sufijo *.px*, donde *x* es el número del plex en el volumen. Por lo tanto este plex se llamará *myvol.p0*.
- La línea *sd* describe un subdisco. La especificación mínima incluye el nombre de la unidad en la que almacenarlo, y la longitud del subdisco. El nombre no es necesario ya que el sistema asigna un nombre automáticamente derivado del nombre del plex añadiendo el sufijo *.sx*, donde *x* es el número del subdisco en el plex. Por lo tanto vinum asigna a este subdisco el nombre *myvol.p0.s0*.

Después de procesar este fichero, gvinum(8) produce la siguiente salida:

```
#  gvinum -> create config1
Configuration summary
Drives:         1 (4 configured)
Volumes:        1 (4 configured)
Plexes:         1 (8 configured)
Subdisks:       1 (16 configured)

  D a                     State: up       Device /dev/da3h      Avail: 2061/2573 MB
(80%)

  V myvol                 State: up       Plexes:       1 Size:      512 MB

  P myvol.p0            C State: up       Subdisks:     1 Size:      512 MB

  S myvol.p0.s0           State: up       PO:        0  B Size:      512 MB
```

Esta salida muestra el formato de lista abreviada de gvinum(8). Está representado gráficamente en Un Volumen vinum Simple.

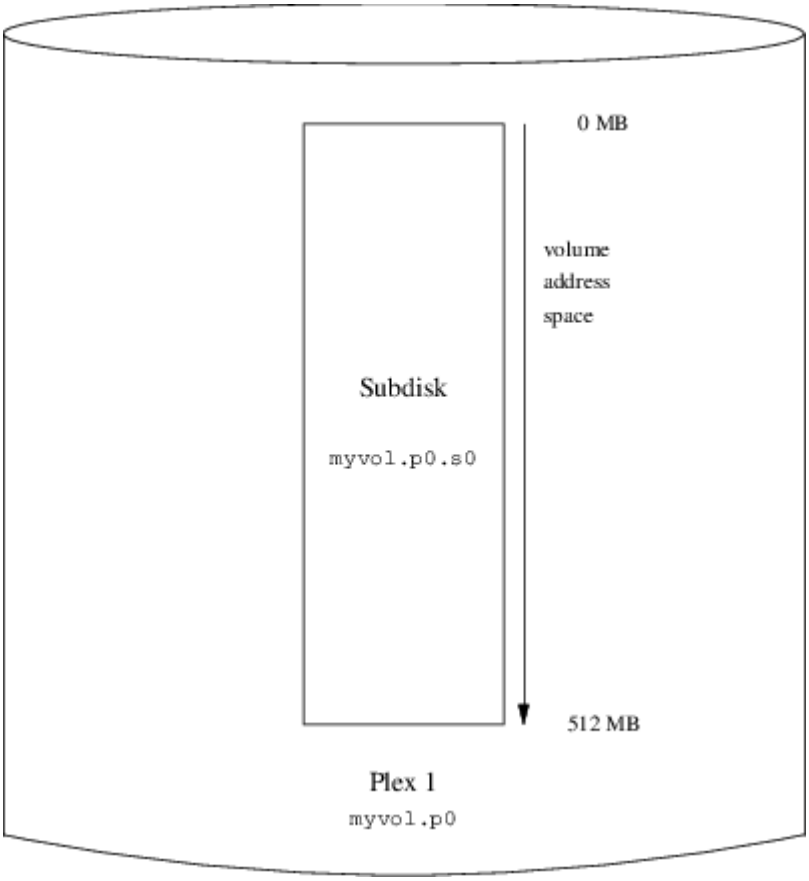

Figura 4. Un Volumen vinum Simple

Esta imagen, y las que siguen, representa un volumen, el cual contiene plexes, que a su vez contienen subdiscos. En este ejemplo, el volumen contiene un plex, y el plex contiene un subdisco.

Este volumen en particular no tiene ninguna ventaja sobre una partición de disco convencional. Contiene un solo plex, así que no es redundante. El plex contiene un solo subdisco, así que no hay diferencia en la asignación de almacenamiento respecto a una partición de disco convencional. Las siguientes secciones ilustran varios métodos de configuración más interesantes.

5.2. Resiliencia Incrementada: Configuración en Espejo

La resiliencia de un volumen puede ser incrementada mediante una configuración en espejo. Cuando se diseña un volumen en espejo, es importante asegurar que los subdiscos de cada plex están en diferentes unidades, de forma que el fallo de una unidad no arrastre ambos plexes. La siguiente configuración crea un espejo de un volumen:

```
drive b device /dev/da4h
volume mirror
  plex org concat
    sd length 512m drive a
  plex org concat
    sd length 512m drive b
```

En este ejemplo, no ha sido necesario especificar la definición de la unidad *a* de nuevo, ya que

vinum hace un seguimiento de todos los objetos en su base de datos de configuración. Después de procesar esta definición, la configuración tiene este aspecto:

```
    Drives:         2 (4 configured)
    Volumes:        2 (4 configured)
    Plexes:         3 (8 configured)
    Subdisks:       3 (16 configured)

    D a                     State: up       Device /dev/da3h       Avail: 1549/2573 MB
(60%)
    D b                     State: up       Device /dev/da4h       Avail: 2061/2573 MB
(80%)

    V myvol                 State: up       Plexes:       1 Size:        512 MB
    V mirror                State: up       Plexes:       2 Size:        512 MB

    P myvol.p0            C State: up       Subdisks:     1 Size:        512 MB
    P mirror.p0           C State: up       Subdisks:     1 Size:        512 MB
    P mirror.p1           C State: initializing     Subdisks:     1 Size:        512
MB

    S myvol.p0.s0           State: up       PO:        0  B Size:        512 MB
    S mirror.p0.s0          State: up       PO:        0  B Size:        512 MB
    S mirror.p1.s0          State: empty    PO:        0  B Size:        512 MB
```

Un Volumen vinum en Espejo muestra la estructura gráficamente.

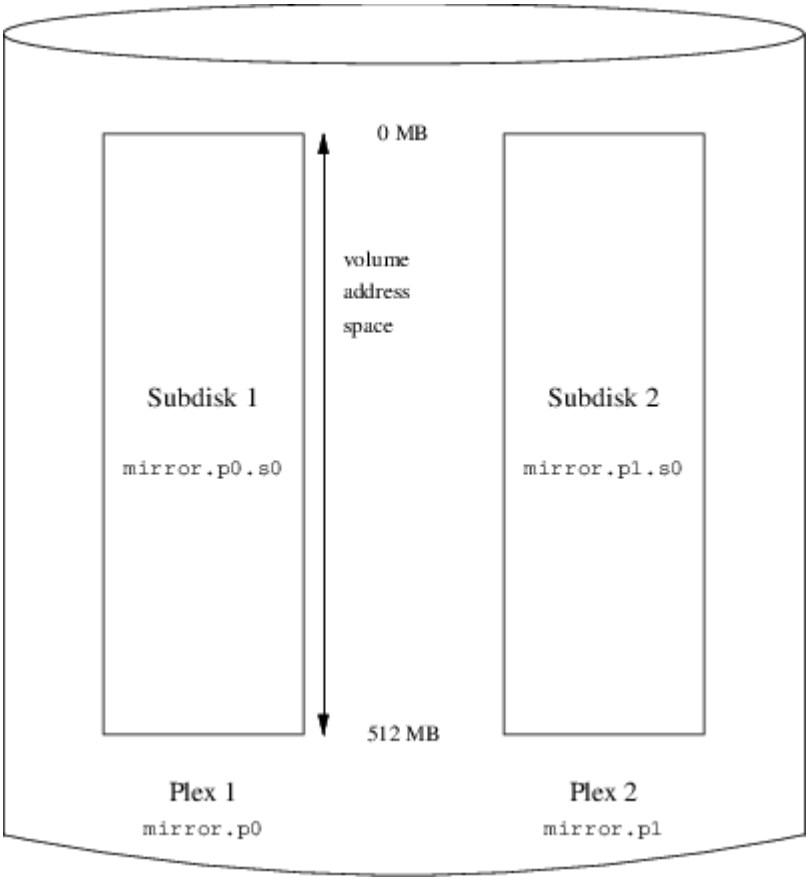

Figura 5. Un Volumen vinum en Espejo

En este ejemplo, cada plex contiene los 512 MB del espacio de direcciones completo. Como en el ejemplo anterior, cada plex contiene solo un subdisco.

5.3. Optimizando el Rendimiento

El volumen en espejo del ejemplo anterior es más resistente a fallos que un volumen que no esté en espejo, pero su rendimiento es menor ya que cada escritura en el volumen requiere una escritura en ambas unidades, usando una mayor porción del ancho de banda total del disco. Las consideraciones de rendimiento requieren una aproximación diferente: en lugar de replicar en espejo, los datos son segmentados en tantas unidades de disco como sea posible. La siguiente configuración muestra un volumen con un plex segmentado en cuatro discos:

```
    drive c device /dev/da5h
drive d device /dev/da6h
volume stripe
plex org striped 512k
  sd length 128m drive a
  sd length 128m drive b
  sd length 128m drive c
  sd length 128m drive d
```

Como antes, no es necesario definir las unidades que ya son conocidas por vinum. Después de procesar esta definición, la configuración tiene este aspecto:

```
    Drives:         4 (4 configured)
    Volumes:        3 (4 configured)
    Plexes:         4 (8 configured)
    Subdisks:       7 (16 configured)

    D a                     State: up       Device /dev/da3h        Avail: 1421/2573
MB (55%)
    D b                     State: up       Device /dev/da4h        Avail: 1933/2573
MB (75%)
    D c                     State: up       Device /dev/da5h        Avail: 2445/2573
MB (95%)
    D d                     State: up       Device /dev/da6h        Avail: 2445/2573
MB (95%)

    V myvol                 State: up       Plexes:       1 Size:        512 MB
    V mirror                State: up       Plexes:       2 Size:        512 MB
    V striped               State: up       Plexes:       1 Size:        512 MB

    P myvol.p0            C State: up       Subdisks:     1 Size:        512 MB
    P mirror.p0           C State: up       Subdisks:     1 Size:        512 MB
    P mirror.p1           C State: initializing     Subdisks:     1 Size:        512
MB
    P striped.p1            State: up       Subdisks:     1 Size:        512 MB

    S myvol.p0.s0           State: up       PO:        0  B Size:        512 MB
    S mirror.p0.s0          State: up       PO:        0  B Size:        512 MB
    S mirror.p1.s0          State: empty    PO:        0  B Size:        512 MB
    S striped.p0.s0         State: up       PO:        0  B Size:        128 MB
    S striped.p0.s1         State: up       PO:      512 kB Size:        128 MB
    S striped.p0.s2         State: up       PO:     1024 kB Size:        128 MB
    S striped.p0.s3         State: up       PO:     1536 kB Size:        128 MB
```

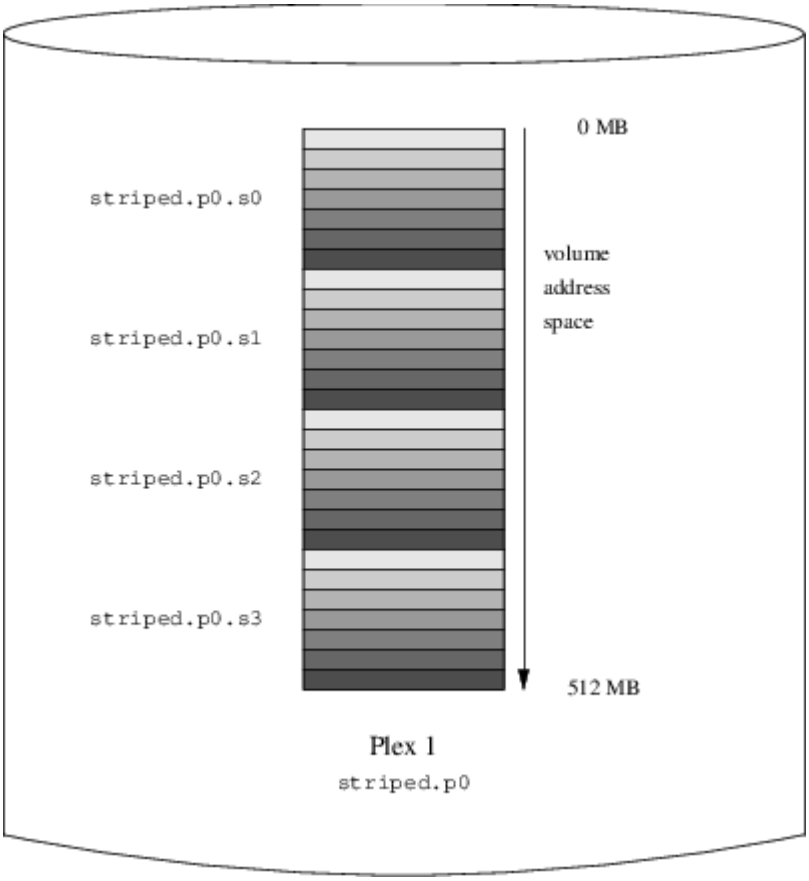

Figura 6. Un Volumen vinum Segmentado

Este volumen está representado en Un Volumen vinum Segmentado. El color oscuro de los segmentos indican la posición dentro del espacio de direcciones del plex, donde los segmentos más claros aparecen primero y los más oscuros últimos.

5.4. Resiliencia y Rendimiento

Con suficiente hardware, es posible construir volúmenes que muestren resiliencia aumentada y rendimiento aumentado comparado con particiones UNIX® estándar. Una configuración típica podría ser:

```
volume raid10
  plex org striped 512k
    sd length 102480k drive a
    sd length 102480k drive b
    sd length 102480k drive c
    sd length 102480k drive d
    sd length 102480k drive e
  plex org striped 512k
    sd length 102480k drive c
    sd length 102480k drive d
    sd length 102480k drive e
    sd length 102480k drive a
    sd length 102480k drive b
```

Los subdiscos del segundo plex están desplazados en dos unidades respecto a aquellos en el primer

flex. Esto ayuda a garantizar que las escrituras no van al mismo subdisco incluso si la transferencia se realiza sobre dos unidades.

Un Volumen vinum Segmentado en Espejo representa la estructura de este volumen.

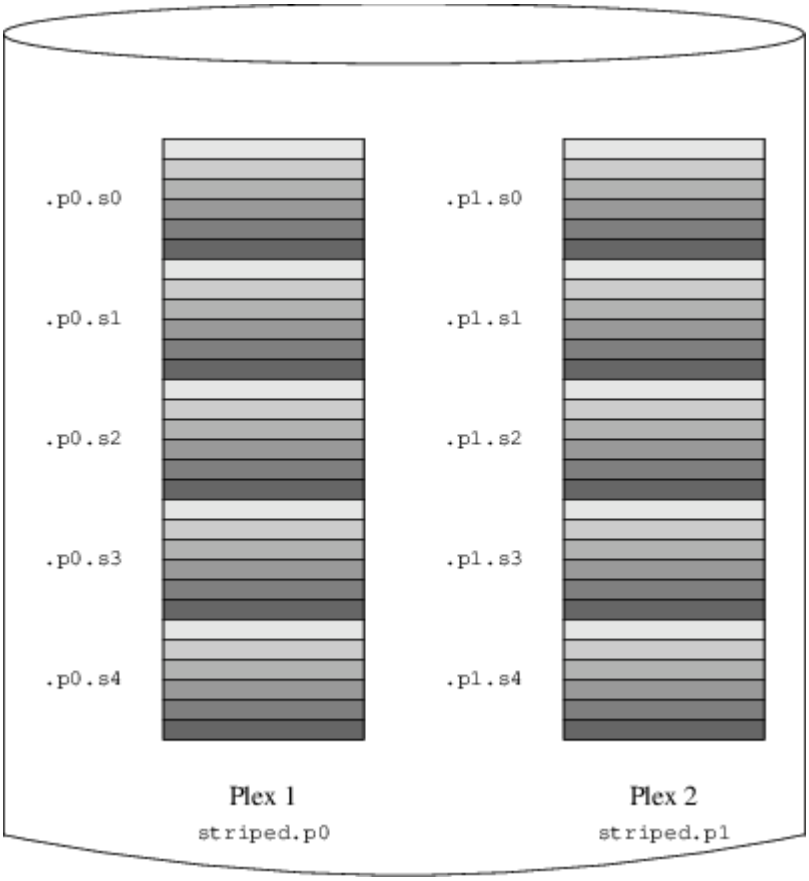

Figura 7. Un Volumen vinum Segmentado en Espejo

6. Nombrado de Objetos

vinum asigna nombres por defecto a los plexes y los subdiscos, aunque pueden ser modificados. Modificar los nombres por defecto no está recomendado ya que no proporciona una ventaja significativa y puede provocar confusión.

Los nombres pueden contener cualquier carácter que no sea blanco, pero se recomienda restringirlos a letras, dígitos y al carácter subrayado. Los nombres de los volúmenes, plexes y subdiscos pueden tener hasta 64 caracteres de longitud, y los nombres de unidades pueden ser de hasta 32 caracteres de longitud.

A los objetos vinum se les asignan nodos de dispositivo en la jerarquía /dev/gvinum. La configuración que se muestra arriba haría que vinum creara los siguientes nodos de dispositivo:

- Entradas de dispositivo para cada volumen. Estos son los dispositivos principales utilizados por vinum. La configuración anterior incluiría los dispositivos /dev/gvinum/myvol, /dev/gvinum/mirror, /dev/gvinum/striped, /dev/gvinum/raid5 y /dev/gvinum/raid10.
- Todos los volúmenes tienen entradas directas bajo /dev/gvinum/.
- Los directorios /dev/gvinum/plex, y /dev/gvinum/sd, los cuales contienen nodos de dispositivo

para cada plex y para cada subdisco, respectivamente.

Por ejemplo, considera el siguiente fichero de configuración:

```
    drive drive1 device /dev/sd1h
    drive drive2 device /dev/sd2h
    drive drive3 device /dev/sd3h
    drive drive4 device /dev/sd4h
    volume s64 setupstate
      plex org striped 64k
        sd length 100m drive drive1
        sd length 100m drive drive2
        sd length 100m drive drive3
        sd length 100m drive drive4
```

Después de procesar este fichero, gvinum(8) crea la siguiente estructura en /dev/gvinum:

```
    drwxr-xr-x  2 root  wheel       512 Apr 13
16:46 plex
    crwxr-xr--  1 root  wheel   91,   2 Apr 13 16:46 s64
    drwxr-xr-x  2 root  wheel       512 Apr 13 16:46 sd

    /dev/vinum/plex:
    total 0
    crwxr-xr--  1 root  wheel   25, 0x10000002 Apr 13 16:46 s64.p0

    /dev/vinum/sd:
    total 0
    crwxr-xr--  1 root  wheel   91, 0x20000002 Apr 13 16:46 s64.p0.s0
    crwxr-xr--  1 root  wheel   91, 0x20100002 Apr 13 16:46 s64.p0.s1
    crwxr-xr--  1 root  wheel   91, 0x20200002 Apr 13 16:46 s64.p0.s2
    crwxr-xr--  1 root  wheel   91, 0x20300002 Apr 13 16:46 s64.p0.s3
```

Aunque se recomienda que los plexes y subdiscos no tengan asignados nombres específicos, las unidades de vinum deben tener nombre. Esto hace posible mover una unidad a una localización diferente y que sea reconocida automáticamente. Los nombres de unidad pueden tener una longitud de hasta 32 caracteres.

6.1. Creando Sistemas de Ficheros

En el sistema los volúmenes parecen idénticos a los discos, con una excepción. A diferencia de las unidades UNIX®, vinum no particiona los volúmenes, por lo que estos no tienen tabla de particiones. Esto ha requerido modificaciones en algunas utilidades de disco, notablemente de newfs(8), de forma que no intente interpretar la última letra de un nombre de volumen vinum. Por ejemplo, una unidad de disco podría tener un nombre como /dev/ad0a# o /dev/da2h#. Estos nombres representan la primera partición (a) en el primer (0) disco IDE (ad) y la octava partición (h) en el tercer (2) disco SCSI (da) respectivamente. Por el contrario, un volumen vinum podría

llamarse /dev/gvinum/concat#, que no guarda relación con un nombre de partición.

Para crear un sistema de ficheros en este volumen, usa newfs(8):

```
# newfs /dev/gvinum/concat
```

7. Configurando vinum

El kernel GENERIC no contiene vinum. Es posible construir un kernel a medida que incluya vinum, pero no se recomienda. La forma estándar de arrancar vinum es como un módulo del kernel. kldload(8) no es necesario porque cuando gvinum(8) arranca, comprueba si el módulo ha sido cargado, y si no es así, lo carga automáticamente.

7.1. Arranque

vinum almacena información de configuración en las porciones de disco de forma esencialmente igual a como guarda los ficheros de configuración. Cuando lee de la base de datos de configuración, vinum reconoce un número de palabras clave que no están permitidas en los ficheros de configuración. Por ejemplo, una configuración de disco podría contener el siguiente texto:

```
volume myvol state up
volume bigraid state down
plex name myvol.p0 state up org concat vol myvol
plex name myvol.p1 state up org concat vol myvol
plex name myvol.p2 state init org striped 512b vol myvol
plex name bigraid.p0 state initializing org raid5 512b vol bigraid
sd name myvol.p0.s0 drive a plex myvol.p0 state up len 1048576b driveoffset 265b
plexoffset 0b
sd name myvol.p0.s1 drive b plex myvol.p0 state up len 1048576b driveoffset 265b
plexoffset 1048576b
sd name myvol.p1.s0 drive c plex myvol.p1 state up len 1048576b driveoffset 265b
plexoffset 0b
sd name myvol.p1.s1 drive d plex myvol.p1 state up len 1048576b driveoffset 265b
plexoffset 1048576b
sd name myvol.p2.s0 drive a plex myvol.p2 state init len 524288b driveoffset 1048841b
plexoffset 0b
sd name myvol.p2.s1 drive b plex myvol.p2 state init len 524288b driveoffset 1048841b
plexoffset 524288b
sd name myvol.p2.s2 drive c plex myvol.p2 state init len 524288b driveoffset 1048841b
plexoffset 1048576b
sd name myvol.p2.s3 drive d plex myvol.p2 state init len 524288b driveoffset 1048841b
plexoffset 1572864b
sd name bigraid.p0.s0 drive a plex bigraid.p0 state initializing len 4194304b driveoff
set 1573129b plexoffset 0b
sd name bigraid.p0.s1 drive b plex bigraid.p0 state initializing len 4194304b driveoff
set 1573129b plexoffset 4194304b
sd name bigraid.p0.s2 drive c plex bigraid.p0 state initializing len 4194304b driveoff
```

```
set 1573129b plexoffset 8388608b
sd name bigraid.p0.s3 drive d plex bigraid.p0 state initializing len 4194304b driveoff
set 1573129b plexoffset 12582912b
sd name bigraid.p0.s4 drive e plex bigraid.p0 state initializing len 4194304b driveoff
set 1573129b plexoffset 16777216b
```

Aquí las diferencias obvias son la presencia de información explícita de localización y nombrado, ambas permitidas pero no recomendadas, y la información de los estados. vinum no almacena información sobre las unidades en la información de configuración. Encuentra las unidades escaneando las unidades de disco configuradas en busca de particiones con una etiqueta vinum. Esto posibilita que vinum identifique las unidades correctamente incluso si se les han asignado identificadores de unidad UNIX® diferentes.

7.1.1. Arranque Automático

Gvinum siempre realiza un arranque automático una vez que el módulo del kernel ha sido cargado, via loader.conf(5). Para cargar el módulo de *Gvinum* en el arranque, añade `geom_vinum_load="YES"` a /boot/loader.conf.

Cuando vinum es arrancado con `gvinum start`, vinum lee la base de datos de configuración de una de las unidades vinum. En condiciones normales, cada unidad contiene una copia idéntica de la base de datos de configuración, de forma que no importa de qué unidad se lea. Después de una caída, sin embargo, vinum debe determinar qué unidad fue actualizada más recientemente y leer la configuración de esta unidad. Después actualiza la configuración progresivamente, si es necesario, de unidades más antiguas.

8. Usando vinum para el Sistema de Ficheros Raíz

Para una máquina que tiene sistemas de ficheros completamente en espejo usando vinum, es deseable también configurar en espejo el sistema de ficheros raíz. Establecer dicha configuración es menos trivial que configurar en espejo un sistema de ficheros arbitrario porque:

- El sistema de ficheros raíz debe estar disponible muy temprano durante el proceso de arranque, por lo que la infraestructura vinum debe estar disponible en ese momento.
- El volumen que contiene el sistema de ficheros raíz, también contiene el sistema de arranque y el kernel. Estos deben ser leídos usando las utilidades nativas del sistema hospedador, como la BIOS, que frecuentemente no puede ser informada de los detalles de vinum.

En las secciones siguientes, el término "volumen raíz" se utiliza para describir el volumen vinum que contiene el sistema de ficheros raíz.

8.1. Arrancando vinum Suficientemente Pronto para el Sistema de Ficheros Raíz

vinum debe estar disponible pronto en el arranque del sistema puesto que loader(8) debe ser capaz de cargar el módulo del kernel de vinum antes de arrancar el kernel. Esto se puede conseguir poniendo la siguiente línea en /boot/loader.conf:

```
geom_vinum_load="YES"
```

8.2. Haciendo Accesible un Volumen Raíz basado en vinum en el Código de Arranque

El código de arranque de FreeBSD ocupa sólo 7.5 KB y no entiende las estructuras internas de vinum. Esto significa que no puede parsear los datos de configuración de vinum o resolver los elementos de un volumen de arranque. Por lo tanto, se necesitan algunas soluciones alternativas para proporcionar al código de arranque con el espejismo de una partición estándar `a` que contiene el sistema de ficheros raíz.

Para que esto sea posible, el volumen raíz debe cumplir los siguientes requisitos:

- El volumen raíz no debe ser un segmento o `RAID`-5.
- El volumen raíz no debe contener más de un subdisco concatenado por plex.

Ten en cuenta que es deseable y posible utilizar múltiples plexes, cada uno conteniendo una réplica del sistema de ficheros raíz. El proceso de arranque sólo utilizará una réplica para encontrar el sistema de arranca y todos los ficheros de arranque, hasta que el kernel monta el sistema de ficheros raíz. Cada subdisco en estos plexes necesita su propio espejismo de partición `a`, para que el dispositivo respectivo sea arrancable. No es estrictamente necesario que cada una de estas particiones `a` simuladas estén localizadas en el mismo desplazamiento en el dispositivo, comparado con otros dispositivos que contienen plexes del volumen raíz. Sin embargo, probablemente sea buena idea crear volúmenes vinum de ese modo de forma que los dispositivos en espejo sean simétricos, para evitar confusión.

Para establecer estas particiones `a` para cada dispositivo que contiene una parte del volumen raíz, se necesite lo siguiente:

1. La localización, desplazamiento desde el principio del dispositivo, y el tamaño del subdisco del dispositivo que forma parte del volumen raíz necesita ser examinado, usando el siguiente comando:

   ```
   # gvinum l -rv root
   ```

   Los desplazamientos y tamaños en vinum se miden en bytes. Estos deben ser divididos entre 512 para obtener los números de bloque que van a ser usados por `bsdlabel`.

2. Ejecuta este comando para cada dispositivo que participa en el volumen raíz:

   ```
   # bsdlabel -e devname
   ```

   devname debe ser o el nombre del disco, como da0 para discos sin una tabla de rebanadas, or el nombre de la rebanada, como ad0s1.

   Si ya hay en el dispositivo una partición `a` de un sistema raíz pre-vinum>vinum#>, se debería renombrar a algo diferente de forma que se mantenga accesible (por si acaso), pero ya no será utilizado como arranque por defecto del sistema. Un sistema de ficheros raíz que está actualmente montado no puede ser renombrado, así que esto se debe ejecutar arrancando desde un medio "Fixit", o en un proceso de dos pasos donde, en una configuración en espejo, el disco que no está siendo arrancando es manipulado primero.

   El desplazamiento de la partición vinum en este dispositivo (si lo hay) se debe añadir al desplazamiento del subdisco del volumen raíz respectivo en este dispositivo.El valor resultante será el valor del `offset` para la nueva partición `a`. El valor de `size` para esta partición se tomará de forma literal del cálculo anterior. El `fstype` debería ser `4.2BSD`. Los valores `fsize`, `bsize`, y `cpg` deberían ser escogidos para que coincidan con el sistema de ficheros real, aunque no son realmente importantes en este contexto.

   De ese modo, se establecerá una nueva partición `a` que solapa la partición vinum en este dispositivo.`bsdlabel` solo permitirá este solapamiento si la partición vinum ha sido adecuadamente marcada utilizando el fstype `vinum`.

3. Ahora existe una falsa partición `a` en cada dispositivo que tiene una réplica del volumen raíz. Es altamente recomendable verificar el resultado usando un comando como:

   ```
   # fsck -n /dev/devnamea
   ```

Debería recordarse que todos los ficheros que contienen información de control deben ser relativos al sistema de ficheros raíz en el volumen vinum que, cuando se establece un nuevo volumen raíz vinum, podría no coincidir con el sistema de ficheros raíz que está actualmente activo. Así que en concreto, hay que tener cuidado con /etc/fstab y /boot/loader.conf.

En el siguiente reinicio, el código de arranque debería averiguar la información de control apropiada en el nuevo sistema de ficheros raíz vinum, y actuar en consecuencia. Al final del proceso de inicialización del kernel, después de que todos los dispositivos han sido anunciados, el aviso destacado que muestra el éxito de esta configuración es un mensaje como este:

```
Mounting root from ufs:/dev/gvinum/root
```

8.3. Ejemplo de una Configuración Raíz basada en vinum

Después de que el volumen raíz vinum ha sido configurado, la salida de `gvinum l -rv root` podría ser como:

```
...
Subdisk root.p0.s0:
        Size:        125829120 bytes (120 MB)
        State: up
        Plex root.p0 at offset 0 (0  B)
        Drive disk0 (/dev/da0h) at offset 135680 (132 kB)

Subdisk root.p1.s0:
        Size:        125829120 bytes (120 MB)
        State: up
        Plex root.p1 at offset 0 (0  B)
        Drive disk1 (/dev/da1h) at offset 135680 (132 kB)
```

Los valores en los que fijarse son `135680` para el offset, en relación a la partición /dev/da0h#. Esto se traduce en 256 bloques de disco de 512 bytes en términos de `bsdlabel`. De igual modo, el tamaño de este volumen raíz es 245760 bloques de 512 bytes. /dev/da1h#, al contener la segunda réplica de este volumen raíz, tiene una configuración asimétrica.

El bsdlabel para estos dispositivos podría parecerse a:

```
...
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:   245760      281    4.2BSD     2048 16384     0  # (Cyl.    0*- 15*)
  c: 71771688        0    unused        0     0        # (Cyl.    0 - 4467*)
  h: 71771672       16     vinum                      # (Cyl.    0*- 4467*)
```

Se puede observar que el parámetro `size` para la partición falsa `a` coincide con el valor esbozado arriba, mientras que el parámetro `offset` es la suma del desplazamiento dentro de la partición vinum `h`, y el desplazamiento de esta partición dentro del dispositivo o rebanada. Esta es una configuración típica que es necesaria para evitar el problema descrito en Nada Arranca, el Arranque entra en Pánico. La partición `a` entera está completamente dentro de la partición `h` que contiene todos los datos vinum para este dispositivo.

En el ejemplo de arriba, el dispositivo entero está dedicado a vinum y no hay una partición raíz pre-vinum restante.

8.4. Solución de problemas

La siguiente lista contiene unos pocos problemas conocidos y sus soluciones.

8.4.1. El Sistema de Arranque Carga, pero el Sistema No Arranca

Si por algún motivo el sistema con continúa con el arranque, el proceso de arranque se puede interrumpir presionando `espacio` durante el aviso de 10 segundos. La variable del cargador `vinum.autostart` puede examinarse tecleando `show` y manipularse usando `set` o `unset`.

Si el módulo del kernel de vinum no estaba en la lista de módulos que son cargados automáticamente, teclea `load geom_vinum`.

Cuando está listo, el proceso de arranque puede continuar tecleando `boot -as` cuyo parámetro `-as` solicita al kernel que pregunte por el sistema de ficheros raíz a montar (`-a`) y hacer que el proceso de arranque pare en modo de usuario único (`-s`), donde el sistema de ficheros raíz está montado como solo lectura. De ese modo, incluso si sólo ha sido montado un plex de un volumen multi-plex, no hay riesgo de inconsistencia de datos entre plexes.

En el prompt en el que se pregunta por el sistema de ficheros raíz a montar, se puede introducir cualquier dispositivo que contiene un sistema de ficheros raíz válido. Si /etc/fstab está configurado correctamente, el valor por defecto debería ser algo como `ufs:/dev/gvinum/root`. Una opción alternativa típica sería algo como `ufs:da0d` que podría contener una hipotética partición de un sistema de ficheros raíz pre-vinum. Ha que tener cuidado si se introduce uno de los alias de las particiones `a`, que en realidad referencia los subdiscos del dispositivo raíz vinum, porque en una configuración en espejo, esto sólo montaría un trozo del dispositivo raíz en espejo. Si este sistema de ficheros se va a montar como lectura-escritura posteriormente, es necesario eliminar el otro plex(es) del volumen raíz vinum puesto que de otro modo los plexes tendrían datos inconsistentes.

8.4.2. Sólo Arranca la Configuración de Arranque Primaria

Si /boot/loader falla al cargar, pero la configuración de arranque primaria todavía carga (visible mediante un sólo guión en la columna de la izquierda de la pantalla justo después de que comience el proceso de arranque), se puede intentar interrumpir el arranque primario presionando `espacio`. Esto hará que el proceso de arranque se pare en stage two. Aquí se puede intentar arrancar desde una partición alternativa, como la partición que contiene sl sistema de ficheros raíz anterior que ha sido movido desde `a`.

8.4.3. Nada Arranca, el Proceso de Arranque entra en Pánico

Esta situación ocurrirá si el código de arranque ha sido destruido por la instalación de vinum. Desafortunadamente, vinum deja por accidente sólo 4KB libres al comienzo de su partición antes de empezar a escribir la información de cabecera de vinum. Sin embargo, los códigos de arranque de las fases uno y dos más bsdlabel requieren 8 KB. Así que si una partición vinum empezó en un offset 0 dentro de una rebanada o disco que se pretendía que fuera arrancable, la configuración de vinum se llevará por delante el código de arranque.

De forma similar, si se ha recuperado de la situación anterior, arrancando desde un medio "Fixit", y el código de arranque ha sido reinstalado utilizando `bsdlabel -B` como se describe en stage two, el código de arranque destruirá la cabecera vinum, y vinum ya no podrá encontrar su(s) disco(s). Aunque ningún volumen vinum o datos de configuración de vinum serán destruidos, y sería posible recuperar todos los datos introduciendo de nuevo exactamente los mismos datos de configuración de vinum la situación es difícil de arreglar. Es necesario mover la partición vinum

entera al menos 4 KB, para que la cabecera vinum y el código de arranque del sistema ya no colisionen.