Менеджер томов vinum

Содержание

1. Обзор . . . 1
2. Узкие места доступа . . . 2
3. Целостность данных . . . 3
4. Объекты vinum . . . 5
5. Некоторые примеры . . . 6
6. Именование объектов . . . 13
7. Настройка vinum . . . 15
8. Использование vinum для корневой файловой системы . . . 16

1. Обзор

Независимо от типа дисков, всегда существуют потенциальные проблемы. Диски могут быть слишком малы, слишком медленны или недостаточно надёжны для соответствия требованиям системы. Хотя диски становятся больше, требования к хранению данных также растут. Часто требуется файловая система, размер которой превышает емкость одного диска. Были предложены и реализованы различные решения этих проблем.

Один из методов заключается в использовании нескольких, а иногда и избыточных дисков. Помимо поддержки различных карт и контроллеров для аппаратных систем RAID (Redundant Array of Independent Disks), базовая система FreeBSD включает менеджер томов vinum, драйвер блочных устройств, который реализует виртуальные диски и решает эти три проблемы. vinum обеспечивает большую гибкость, производительность и надёжность по сравнению с традиционными системами хранения данных, а также реализует модели `RAID`-0, `RAID`-1 и `RAID`-5 как по отдельности, так и в комбинации.

Эта глава предоставляет обзор потенциальных проблем с традиционным дисковым хранилищем и введение в менеджер томов vinum.



vinum устарел и отсутствует в FreeBSD 15.0 и более поздних версиях. Пользователям рекомендуется перейти на gconcat(8), gmirror(8), gstripe(8), graid(8) или zfs(8).



Начиная с FreeBSD 5, vinum был переписан для интеграции в архитектуру GEOM, сохраняя при этом оригинальные идеи, терминологию и метаданные на диске. Эта переработанная версия называется *gvinum* (от *GEOM vinum*). Хотя в этой главе используется термин vinum, все команды должны выполняться с помощью `gvinum`. Имя модуля ядра изменилось с оригинального vinum.ko на geom_vinum.ko, а все узлы устройств находятся в

> /dev/gvinum вместо /dev/vinum. Начиная с FreeBSD 6, оригинальная реализация vinum больше не доступна в кодовой базе.

2. Узкие места доступа

Современные системы часто нуждаются в доступе к данным в условиях высокой параллельности. Например, крупные FTP- или HTTP-серверы могут поддерживать тысячи одновременных сеансов и иметь несколько 100 Мбит/с соединений с внешним миром, что значительно превышает устойчивую скорость передачи большинства дисков.

Современные дисковые накопители могут передавать данные последовательно со скоростью до 70 МБ/с, но это значение имеет мало значения в среде, где множество независимых процессов обращаются к накопителю и могут достичь лишь доли этих значений. В таких случаях более интересно рассмотреть проблему с точки зрения дисковой подсистемы. Важным параметром является нагрузка, которую передача данных создаёт на подсистему, или время, в течение которого передача занимает задействованные накопители.

При любом переносе данных на диск сначала необходимо позиционировать головки, дождаться, пока первый сектор окажется под считывающей головкой, а затем выполнить перенос. Эти действия можно считать атомарными, так как их прерывание не имеет смысла.

Рассмотрим типичную передачу около 10 КБ: современные высокопроизводительные диски могут позиционировать головки в среднем за 3,5 мс. Самые быстрые диски вращаются со скоростью 15 000 об/мин, поэтому средняя задержка вращения (половина оборота) составляет 2 мс. При скорости 70 МБ/с сама передача занимает около 150 мкс, что почти ничто по сравнению с временем позиционирования. В таком случае эффективная скорость передачи падает до чуть более 1 МБ/с и явно сильно зависит от размера передачи.

Традиционное и очевидное решение этой проблемы — «больше дисков»: вместо одного большого диска использовать несколько дисков меньшего размера с тем же общим объёмом хранилища. Каждый диск способен позиционироваться и передавать данные независимо, поэтому эффективная пропускная способность увеличивается почти пропорционально количеству используемых дисков.

Фактическое увеличение пропускной способности меньше, чем количество задействованных дисков. Хотя каждый диск способен передавать данные параллельно, нет возможности гарантировать равномерное распределение запросов между дисками. Неизбежно нагрузка на один диск будет выше, чем на другой.

Равномерность нагрузки на диски сильно зависит от способа распределения данных по накопителям. В дальнейшем обсуждении удобно представлять дисковое хранилище как большое количество секторов данных, адресуемых по номерам, подобно страницам в книге. Наиболее очевидный метод — разделить виртуальный диск на группы последовательных секторов размером с отдельные физические диски и хранить их таким образом, как если бы большую книгу разорвали на меньшие разделы. Этот метод называется *объединением* (конкатенацией) и имеет преимущество в том, что диски не требуют каких-либо

определённых соотношений размеров. Он хорошо работает, когда доступ к виртуальному диску равномерно распределён по его адресному пространству. Если доступ сосредоточен на меньшей области, улучшение менее заметно. Организация методом объединения иллюстрирует последовательность выделения блоков хранения в организации методом объединения.

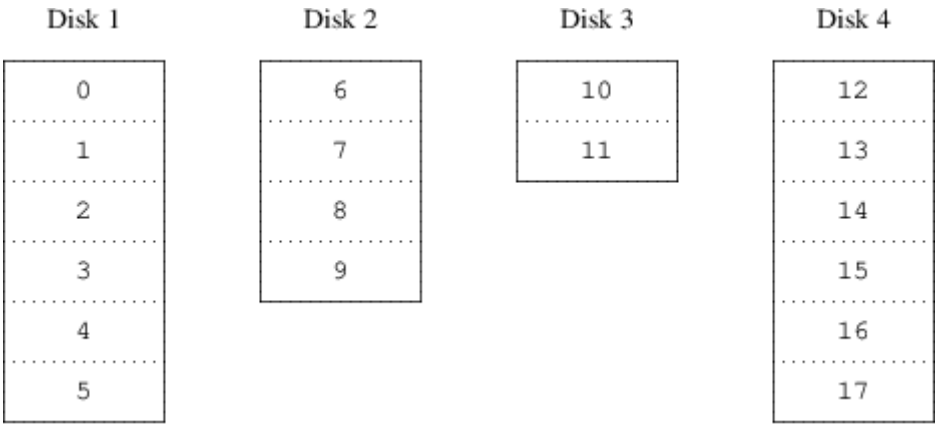


Рисунок 1. Организация методом объединения

Альтернативный метод распределения заключается в разделении адресного пространства на меньшие равные по размеру компоненты и их последовательном хранении на разных устройствах. Например, первые 256 секторов могут храниться на первом диске, следующие 256 секторов — на следующем диске и так далее. После заполнения последнего диска процесс повторяется, пока все диски не будут заполнены. Такой метод называется *чередованием (striping)* или RAID-0.

RAID предлагает различные формы отказоустойчивости, хотя RAID-0 несколько вводит в заблуждение, так как не обеспечивает избыточности. Разделение данных требует несколько больше усилий для их поиска и может создавать дополнительную нагрузку ввода-вывода, когда передача распределяется по нескольким дискам, но также может обеспечить более равномерную нагрузку на диски. Организация методом чередования иллюстрирует последовательность, в которой организуется распределение блоков хранения с чередованием.

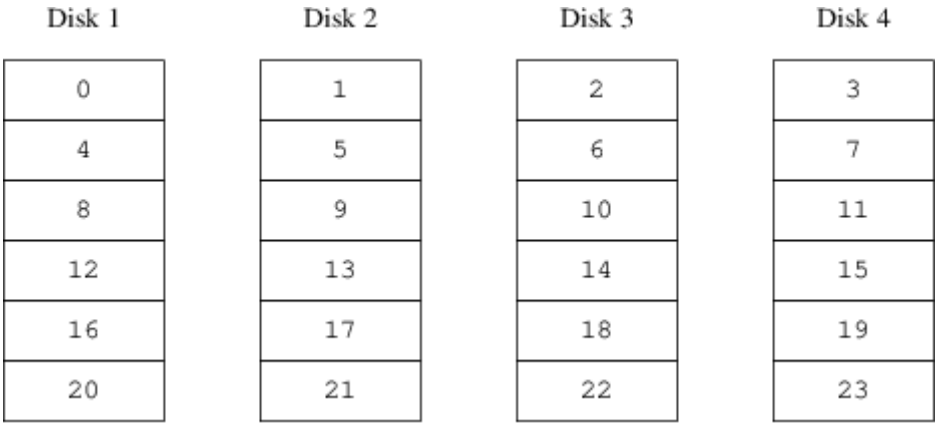


Рисунок 2. Организация методом чередования

3. Целостность данных

Последняя проблема с дисками заключается в их ненадёжности. Хотя надёжность

значительно повысилась за последние годы, дисковые накопители остаются наиболее вероятным компонентом сервера, который может выйти из строя. Когда это происходит, последствия могут быть катастрофическими, а замена вышедшего из строя диска и восстановление данных могут привести к простою сервера.

Один из подходов к этой проблеме — *зеркалирование*, или RAID-1, при котором данные хранятся в двух экземплярах на разных физических носителях. Любая запись на том записывается на оба диска; чтение может выполняться с любого из них, поэтому при отказе одного диска данные остаются доступны на другом.

Зеркалирование имеет две проблемы:

- Требуется в два раза больше дискового пространства, чем для решения без избыточности.
- Запись должна выполняться на оба диска, поэтому она занимает в два раза больше пропускной способности, чем в незеркалированном томе. Чтение не страдает от потери производительности и может быть даже быстрее.

Альтернативным решением является *чётность*, реализованная в уровнях RAID 2, 3, 4 и 5. Из них RAID-5 представляет наибольший интерес. В реализации vinum это вариант организации с чередованием, где один блок каждой полосы выделяется под чётность одного из других блоков. В реализации vinum плекс RAID-5 аналогичен плексу с чередованием, за исключением того, что он реализует RAID-5, включая блок чётности в каждую полосу. Как требуется в RAID-5, расположение этого блока чётности меняется от одной полосы к другой. Числа в блоках данных обозначают относительные номера блоков.

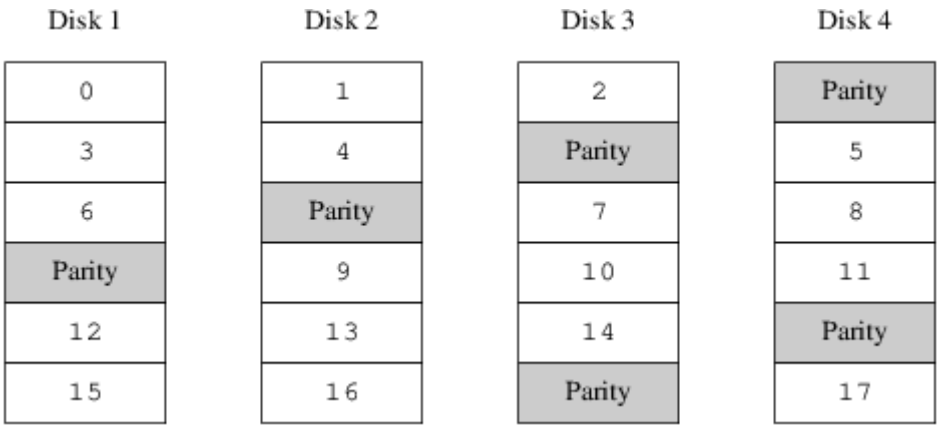


Рисунок 3. Организация RAID-5

По сравнению с зеркалированием, RAID-5 имеет преимущество в виде значительно меньшего требуемого объёма хранилища. Скорость чтения аналогична таковой при чередующейся организации, но скорость записи значительно ниже — примерно 25% от скорости чтения. Если один диск выходит из строя, массив может продолжать работу в деградировавшем режиме, при котором чтение с оставшихся доступных дисков продолжается в обычном режиме, а чтение с отказавшего диска пересчитывается из соответствующих блоков всех оставшихся дисков.

4. Объекты vinum

Для решения этих проблем vinum реализует четырёхуровневую иерархию объектов:

- Наиболее заметным объектом является виртуальный диск, называемый *томом*. Том обладает практически теми же свойствами, что и UNIX® дисковый накопитель, хотя есть некоторые незначительные отличия. Например, у тома нет ограничений по размеру.
- Тома состоят из *плексов*, каждый из которых представляет полное адресное пространство тома. Этот уровень в иерархии обеспечивает избыточность. Можно представить плексы как отдельные диски в зеркальном массиве, каждый из которых содержит одинаковые данные.
- Поскольку vinum существует в рамках системы хранения данных UNIX®, можно было бы использовать разделы UNIX® в качестве строительных блоков для многодисковых plexes. Однако на практике это оказывается слишком негибким, так как диски UNIX® могут иметь только ограниченное количество разделов. Вместо этого vinum разбивает единственный раздел UNIX®, называемый *дисковый раздел (drive)*, на непрерывные области, называемые *поддисками (subdisk)*, которые используются как строительные блоки для плексов.
- Поддиски располагаются на *дисковых разделах* vinum, в настоящее время это разделы UNIX®. Разделы vinum могут содержать любое количество поддисков. За исключением небольшой области в начале раздела, которая используется для хранения конфигурации и состояния, весь раздел доступен для хранения данных.

Следующие разделы статьи описывают, каким образом эти объекты обеспечивают функциональность, требуемую для vinum.

4.1. Учёт размера томов

Плексы могут включать несколько поддисков, распределённых по всем дискам в конфигурации vinum. В результате, размер отдельного диска не ограничивает размер плекса или тома.

4.2. Избыточное хранение данных

vinum реализует зеркалирование путем присоединения нескольких плекс к тому. Каждый плекс представляет данные в томе. Том может содержать от одного до восьми плексов.

Хотя плекс представляет полные данные тома, возможно, что некоторые части представления физически отсутствуют — либо по замыслу (если поддиск для частей плекса не определён), либо случайно (в результате выхода диска из строя). До тех пор, пока хотя бы один плекс может предоставить данные для полного адресного пространства тома, том остаётся полностью работоспособным.

4.3. Какую организацию плексов выбрать?

vinum реализует как объединение, так и чередование на уровне плекс:

- *Плекс с объединением* использует адресное пространство каждого поддиска по очереди. Объединённые плексы являются наиболее гибкими, так как могут содержать любое количество поддисков, а поддиски могут быть разной длины. Плекс может быть расширен путём добавления дополнительных поддисков. Они требуют меньше процессорного времени, чем чередующиеся плексы, хотя разница в нагрузке на процессор незначительна. С другой стороны, они наиболее подвержены "горячим точкам", когда один диск очень активен, а другие простаивают.
- *Плекс с чередованием* распределяет данные по каждому поддиску. Поддиски должны быть одного размера, и их должно быть как минимум два, чтобы отличить такой плекс от объединенного. Главное преимущество чередующихся плексов в том, что они уменьшают вероятность появления "горячих точек". Выбрав оптимальный размер полосы (около 256 КБ), можно равномерно распределить нагрузку на диски – компоненты системы. Расширение плекса путем добавления новых поддисков настолько сложно, что vinum не реализует эту возможность.

Организации плексов в vinum обобщает преимущества и недостатки каждой организации плексов.

Таблица 1. Организации плексов в vinum

Тип плекса	Минимальное количество поддисков	Может добавлять поддиски	Должен быть равного размера	Приложение
объединённый	1	да	no	Крупное хранилище данных с максимальной гибкостью размещения и умеренной производительностью
чередуемый	2	no	да	Высокая производительность в сочетании с высокопараллельным доступом

5. Некоторые примеры

vinum поддерживает *базу данных конфигурации*, которая описывает объекты, известные конкретной системе. Первоначально пользователь создаёт базу данных конфигурации из

одного или нескольких конфигурационных файлов с помощью gvinum(8). vinum хранит копию своей базы данных конфигурации на каждом *устройстве* диска, находящемся под его управлением. Эта база данных обновляется при каждом изменении состояния, так что перезапуск точно восстанавливает состояние каждого объекта vinum.

5.1. Файл конфигурации

Файл конфигурации описывает отдельные объекты vinum. Определение простого тома может выглядеть следующим образом:

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

Этот файл описывает четыре объекта vinum:

- Строка *drive* описывает раздел диска (*drive*) и его расположение относительно оборудования, на котором он расположен. Ему присваивается символическое имя *a*. Такое разделение символических имён от имён устройств позволяет перемещать диски из одного места в другое без путаницы.
- Строка *volume* описывает том. Единственный обязательный атрибут — это имя, в данном случае *myvol*.
- Строка *plex* определяет плекс. Единственный обязательный параметр — это организация, в данном случае *concat*. Имя не требуется, так как система автоматически генерирует его из имени тома, добавляя суффикс *.px*, где *x* — номер плекса в томе. Таким образом, этот плекс будет называться *myvol.p0*.
- Строка *sd* описывает поддиск. Минимальные требования — это имя диска для его хранения и длина поддиска. Имя не обязательно, так как система автоматически назначает имена, производные от имени плекса, добавляя суффикс *.sx*, где *x* — номер поддиска в плексе. Таким образом, vinum присваивает этому поддиску имя *myvol.p0.s0*.

После обработки этого файла команда gvinum(8) выводит следующий результат:

```
#  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
```

Этот вывод показывает краткий формат списка gvinum(8). Он представлен графически в Простой том vinum.

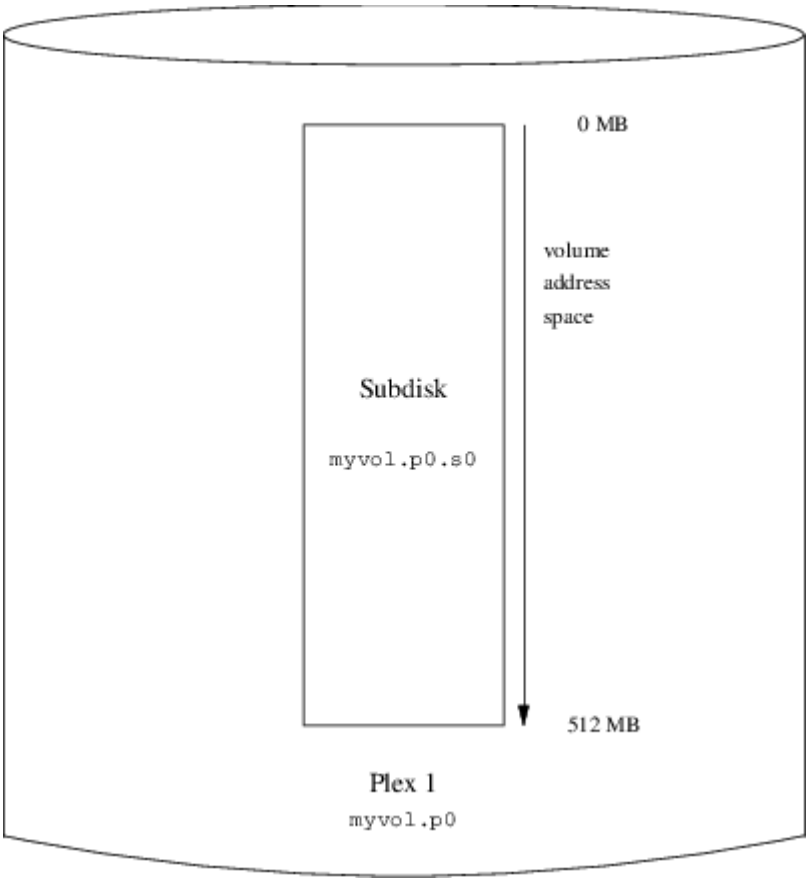


Рисунок 4. Простой том vinum

Этот рисунок и следующие представляют том, который содержит плексы, которые, в свою очередь, содержат поддиски. В этом примере том содержит один плекс, а плекс содержит один поддиск.

Этот конкретный том не имеет особых преимуществ по сравнению с обычным разделом диска. Он содержит один плекс, поэтому не является избыточным. Плекс содержит один поддиск, поэтому нет различий в распределении хранилища по сравнению с обычным разделом диска. В следующих разделах показаны различные более интересные методы конфигурации.

5.2. Увеличенная отказоустойчивость: зеркалирование

Устойчивость тома может быть повышена за счёт зеркалирования. При создании зеркального тома важно убедиться, что поддиски каждого плекса находятся на разных дисках, чтобы выход из строя одного диска не затронул оба плекса. Следующая конфигурация создаёт зеркальный том:

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

В этом примере не потребовалось снова указывать определение диска *a*, поскольку vinum отслеживает все объекты в своей базе данных конфигурации. После обработки этого определения конфигурация выглядит следующим образом:

```
    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
```

Зеркальный том vinum графически отображает структуру.

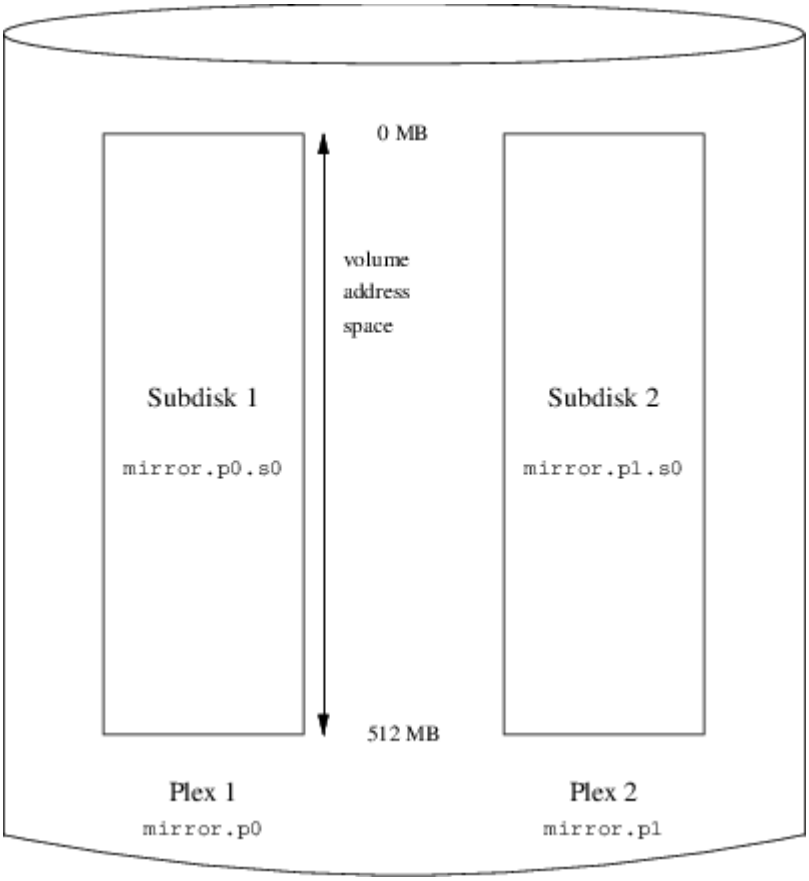


Рисунок 5. Зеркальный том vinum

В этом примере каждый плекс содержит полные 512 МБ адресного пространства. Как и в предыдущем примере, каждый плекс содержит только один поддиск.

5.3. Оптимизация производительности

Зеркальный том в предыдущем примере более устойчив к сбоям, чем незеркальный том, но его производительность ниже, так как каждая запись в том требует записи на оба диска, используя большую часть общей пропускной способности дисков. Соображения производительности требуют другого подхода: вместо зеркалирования данные распределяются по полосам на максимально возможное количество дисков. Следующая конфигурация показывает том с плексом, распределённым по полосам на четырёх дисках:

```
    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
```

Как и ранее, не нужно определять диски, которые уже известны vinum. После обработки этого определения конфигурация выглядит следующим образом:

```
    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
```

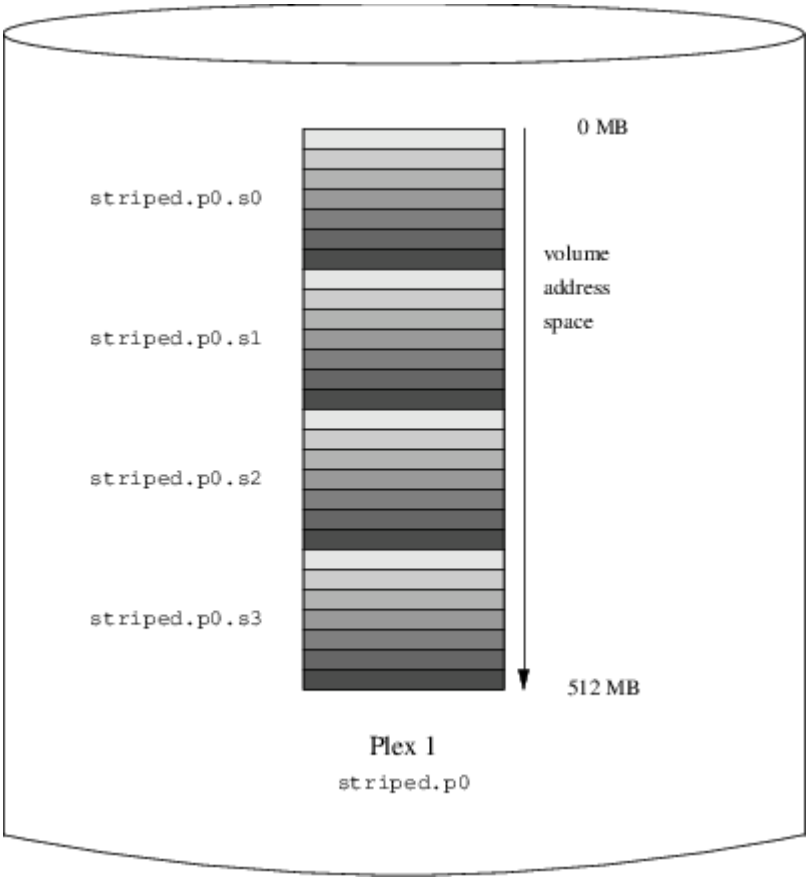


Рисунок 6. Том vinum с чередованием

Этот том представлен на схеме Том vinum с чередованием. Темнота полос указывает на позицию в адресном пространстве плекса, где самые светлые полосы идут первыми, а самые темные — последними.

5.4. Устойчивость и производительность

При достаточном аппаратном обеспечении можно создать тома, которые демонстрируют как повышенную отказоустойчивость, так и увеличенную производительность по сравнению со стандартными разделами UNIX®. Типичный конфигурационный файл может выглядеть так:

```
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
```

Поддиски второго плекса смещены на два диска относительно поддисков первого плекса. Это помогает гарантировать, что записи не будут направляться на одни и те же поддиски, даже если передача затронет два диска.

Том <span class= представляет структуру этого тома.

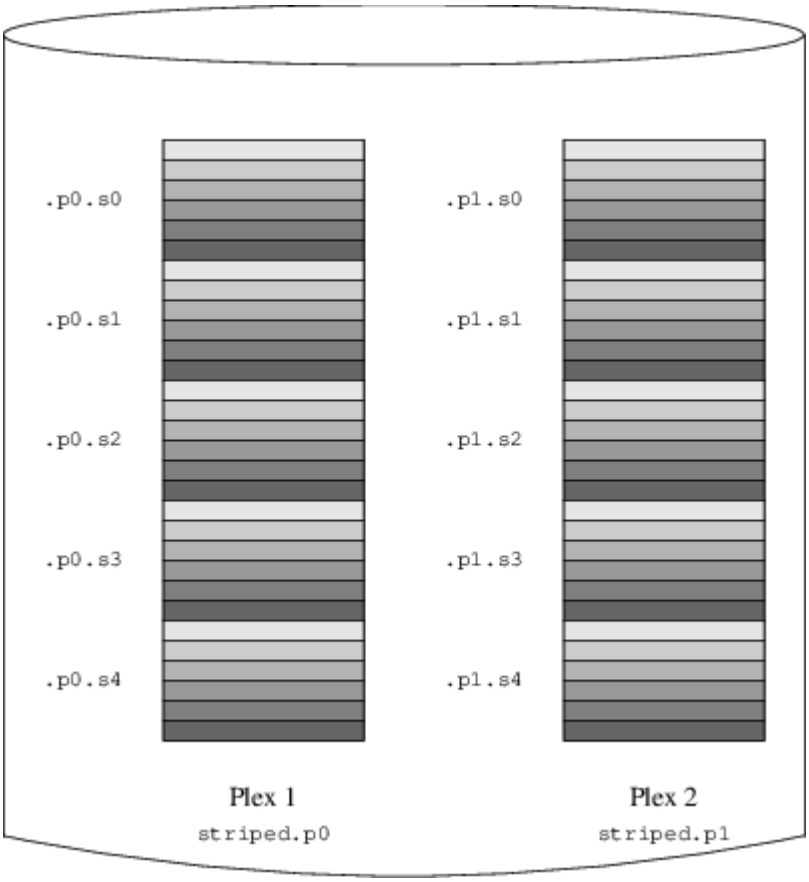


Рисунок 7. Том vinum с зеркалированием и чередованием

6. Именование объектов

vinum назначает стандартные имена для плексов и поддисков, хотя их можно изменить. Не рекомендуется изменять стандартные имена, так как это не даёт значительных преимуществ и может вызвать путаницу.

Имена могут содержать любые непустые символы, но рекомендуется ограничиваться буквами, цифрами и символами подчёркивания. Имена томов, плексов и поддисков могут быть длиной до 64 символов, а имена дисков — до 32 символов.

Объектам vinum назначаются узлы устройств в иерархии /dev/gvinum. Приведённая выше конфигурация приведёт к тому, что vinum создаст следующие узлы устройств:

- Записи устройств для каждого тома. Это основные устройства, используемые vinum. Приведённая конфигурация включает устройства /dev/gvinum/myvol, /dev/gvinum/mirror, /dev/gvinum/striped, /dev/gvinum/raid5 и /dev/gvinum/raid10.
- Все тома получают собственные записи в /dev/gvinum/.
- Каталоги /dev/gvinum/plex и /dev/gvinum/sd, которые содержат узлы устройств для

каждого плекса и каждого субдиска соответственно.

Например, рассмотрим следующий конфигурационный файл:

```
    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
```

После обработки этого файла gvinum(8) создаёт следующую структуру в /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
```

Хотя рекомендуется не назначать конкретные имена плексам и поддискам, диски vinum должны быть именованными. Это позволяет переместить диск в другое место и по-прежнему автоматически его распознавать. Имена дисков могут быть длиной до 32 символов.

6.1. Создание файловых систем

Тома для системы выглядят идентично дискам, за одним исключением. В отличие от дисков UNIX®, vinum не разбивает тома на разделы, поэтому они не содержат таблицы разделов. Это потребовало внесения изменений в некоторые утилиты для работы с дисками, в частности, в newfs(8), чтобы они не пытались интерпретировать последнюю букву имени тома vinum как идентификатор раздела. Например, имя диска может выглядеть как /dev/ad0a или /dev/da2h. Эти имена обозначают первый раздел (a) на первом (0) IDE-диске (ad) и восьмой раздел (h) на третьем (2) SCSI-диске (da), соответственно. В отличие от этого,

том vinum может называться /dev/gvinum/concat, что не имеет отношения к имени раздела.

Чтобы создать файловую систему на этом томе, используйте newfs(8):

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

7. Настройка vinum

Ядро GENERIC не содержит vinum. Можно собрать пользовательское ядро с включённым vinum, но это не рекомендуется. Стандартный способ запуска vinum — в качестве модуля ядра. Команда kldload(8) не требуется, так как при запуске gvinum(8) проверяет, загружен ли модуль, и если нет, загружает его автоматически.

7.1. Запуск

vinum хранит конфигурационную информацию на дисковых слайсах практически в той же форме, что и в конфигурационных файлах. При чтении из базы данных конфигурации vinum распознаёт ряд ключевых слов, которые не допускаются в конфигурационных файлах. Например, конфигурация диска может содержать следующий текст:

```
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
```

Очевидные различия здесь — наличие явной информации о местоположении и именования, что разрешено, но не рекомендуется, а также информация о состояниях. vinum не хранит сведения о дисках в конфигурационной информации. Он находит диски, сканируя настроенные дисковые накопители на наличие разделов с меткой vinum. Это позволяет vinum корректно идентифицировать диски, даже если им были присвоены разные идентификаторы дисков UNIX®.

7.1.1. Автоматический запуск

Gvinum всегда запускается автоматически после загрузки модуля ядра через loader.conf(5). Чтобы загрузить модуль *Gvinum* при загрузке системы, добавьте `geom_vinum_load="YES"` в /boot/loader.conf.

Когда vinum запускается командой `gvinum start`, vinum читает конфигурационную базу данных с одного из дисков vinum. В нормальных условиях каждый диск содержит идентичную копию конфигурационной базы данных, поэтому не имеет значения, с какого диска читать. Однако после сбоя vinum должен определить, какой диск был обновлён последним, и прочитать конфигурацию с этого диска. Затем, если необходимо, он обновляет конфигурацию последовательно с более старых дисков.

8. Использование vinum для корневой файловой системы

Для машины с полностью зеркалированными файловыми системами с использованием vinum, желательно также зеркалировать корневую файловую систему. Настройка такой конфигурации менее тривиальна, чем зеркалирование произвольной файловой системы, потому что:

- Корневая файловая система должна быть доступна очень рано в процессе загрузки, поэтому инфраструктура vinum должна быть уже доступна на этом этапе.
- Том, содержащий корневую файловую систему, также включает системный загрузчик и ядро. Они должны быть прочитаны с использованием родных утилит хостовой системы, таких как BIOS, который зачастую нельзя обучить работе с деталями vinum.

В следующих разделах термин "корневой том" обычно используется для описания тома vinum, который содержит корневую файловую систему.

8.1. Запуск vinum на раннем этапе для обеспечения доступа к корневой файловой системе

Файл `vinum` должен быть доступен на раннем этапе загрузки системы, так как `loader(8)` должен загрузить модуль ядра `vinum` перед запуском ядра. Это можно сделать, добавив следующую строку в `/boot/loader.conf`:

```
geom_vinum_load="YES"
```

8.2. Создание корневого тома на основе vinum, доступного для загрузчика

Текущая загрузочная запись FreeBSD занимает всего 7,5 КБ кода и не понимает внутренние структуры vinum. Это означает, что она не может разобрать конфигурационные данные vinum или определить элементы загрузочного тома. Таким образом, необходимы некоторые обходные решения, чтобы предоставить загрузочному коду иллюзию стандартного раздела `a`, содержащего корневую файловую систему.

Для этого должны быть выполнены следующие требования к корневому тому:

- Корневой том не должен быть чередующимся или `RAID`-5.
- Корневой том не должен содержать более одного объединённого поддиска на плекс.

Обратите внимание, что желательно и возможно использовать несколько плексов, каждый из которых содержит одну реплику корневой файловой системы. Процесс начальной загрузки будет использовать только одну реплику для поиска загрузчика и всех загрузочных файлов, пока ядро не смонтирует корневую файловую систему. Каждый отдельный поддиск в этих плексах должен иметь свою собственную иллюзию раздела `a`, чтобы соответствующее устройство было загрузочным. Не строго необходимо, чтобы каждый из этих фальшивых разделов `a` находился на том же смещении внутри своего устройства по сравнению с другими устройствами, содержащими плекс корневого тома. Однако, вероятно, хорошей идеей будет создавать тома vinum таким образом, чтобы результирующие зеркальные устройства были симметричными, чтобы избежать путаницы.

Для настройки этих разделов `a` на каждом устройстве, содержащем часть корневого тома, требуется следующее:

1. Местоположение, смещение от начала устройства и размер подобласти этого устройства, которая является частью корневого тома, необходимо проверить с помощью команды:

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

Смещения и размеры в vinum измеряются в байтах. Их необходимо разделить на 512, чтобы получить номера блоков, которые будут использоваться в `bsdlabel`.

2. Выполните эту команду для каждого устройства, участвующего в корневом томе:

```
# bsdlabel -e devname
```

`devname` должен быть либо именем диска, например, da0 для дисков без таблицы разделов, либо именем раздела, например, ad0s1.

Если на устройстве уже существует раздел `a` из корневой файловой системы до vinum, его следует переименовать во что-то другое, чтобы он оставался доступным (на всякий случай), но больше не использовался по умолчанию для загрузки системы. Текущий смонтированный корневой файловой системы нельзя переименовать, поэтому это должно выполняться либо при загрузке с "Fixit" носителя, либо в два этапа, когда в зеркале сначала обрабатывается диск, с которого в данный момент не загружаются.

Смещение раздела vinum на этом устройстве (если есть) должно быть добавлено к смещению соответствующего поддиска корневого тома на этом устройстве. Полученное значение станет значением `offset` для нового раздела `a`. Значение `size` для этого раздела можно взять дословно из приведённых выше расчётов. Для `fstype` следует указать `4.2BSD`. Значения `fsize`, `bsize` и `cpg` должны быть выбраны в соответствии с реальной файловой системой, хотя в данном контексте они не слишком важны.

Таким образом, будет создан новый раздел `a`, который перекрывает раздел vinum на этом устройстве. `bsdlabel` разрешит это перекрытие только в том случае, если раздел vinum был правильно помечен с использованием типа файловой системы `vinum`.

3. Поддельный раздел `a` теперь существует на каждом устройстве, имеющем одну реплику корневого тома. Настоятельно рекомендуется проверить результат с помощью команды, например:

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

Следует помнить, что все файлы, содержащие управляющую информацию, должны быть относительны к корневой файловой системе в томе vinum, которая при настройке нового корневого тома vinum может не совпадать с текущей активной корневой файловой системой. Поэтому, в частности, необходимо позаботиться о /etc/fstab и /boot/loader.conf.

При следующей перезагрузке загрузчик должен определить соответствующую управляющую информацию из новой корневой файловой системы на основе vinum и действовать соответствующим образом. В конце процесса инициализации ядра, после объявления всех устройств, явным признаком успешного завершения настройки будет сообщение вида:

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

8.3. Пример настройки корневой системы на основе vinum

После настройки корневого тома vinum, вывод команды `gvinum l -rv root` может выглядеть следующим образом:

```
...
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)
```

Значения, на которые следует обратить внимание: `135680` для смещения, относительного к разделу /dev/da0h. Это соответствует 265 блокам диска по 512 байт в терминах `bsdlabel`. Аналогично, размер этого корневого тома составляет 245760 блоков по 512 байт. /dev/da1h, содержащий вторую реплику этого корневого тома, имеет симметричную конфигурацию.

Метка bsdlabel для этих устройств может выглядеть следующим образом:

```
...
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*)
```

Можно заметить, что параметр `size` для поддельного раздела `a` совпадает с указанным выше значением, в то время как параметр `offset` представляет собой сумму смещения внутри раздела vinum `h` и смещения этого раздела в устройстве или слайсе. Это стандартная настройка, необходимая для избежания проблемы, описанной в Ничего не загружается, паника при загрузке. Весь раздел `a` полностью находится внутри раздела `h`, содержащего все данные vinum для этого устройства.

В приведённом выше примере все устройство выделено под vinum, и не осталось корневого раздела, существовавшего до vinum.

8.4. Устранение неполадок

Следующий список содержит несколько известных подводных камней и их решения.

8.4.1. Загрузчик системы загружается, но система не запускается

Если по какой-либо причине система не продолжает загрузку, процесс можно прервать, нажав `space` при появлении 10-секундного предупреждения. Переменную загрузчика `vinum.autostart` можно проверить, введя команду `show`, и изменить с помощью `set` или `unset`.

Если модуль ядра vinum ещё не был в списке модулей для автоматической загрузки, введите `load geom_vinum`.

Когда всё готово, процесс загрузки можно продолжить, введя `boot -as`, где `-as` указывает ядру запросить корневую файловую систему для монтирования (`-a`) и остановить процесс загрузки в однопользовательском режиме (`-s`), при этом корневая файловая система монтируется в режиме только для чтения. Таким образом, даже если смонтирован только один слой многокомпонентного тома, не возникает риска несогласованности данных между слоями.

На запрос о корневой файловой системе для монтирования можно ввести любое устройство, содержащее действительную корневую файловую систему. Если /etc/fstab настроен правильно, по умолчанию должно быть что-то вроде `ufs:/dev/gvinum/root`. Типичным альтернативным выбором может быть что-то вроде `ufs:da0d`, что может быть гипотетическим разделом, содержащим корневую файловую систему до vinum. Следует быть осторожным, если здесь вводится один из псевдонимов `a` разделов, чтобы он действительно ссылался на поддиски устройства vinum root, потому что в зеркальной настройке это приведёт к монтированию только одной части зеркального корневого устройства. Если эта файловая система будет позже смонтирована в режиме чтения-записи, необходимо удалить другие плексы тома vinum root, так как в противном случае эти плексы будут содержать несогласованные данные.

8.4.2. Только первичная загрузка Bootstrap

Если /boot/loader не загружается, но первичная загрузка всё ещё работает (это видно по одному дефису в левой части экрана сразу после начала процесса загрузки), можно попытаться прервать первичную загрузку, нажав `space`. Это остановит загрузку на втором этапе. Здесь можно попытаться загрузиться с альтернативного раздела, например, с раздела, содержащего предыдущую корневую файловую систему, которая была перемещена из `a`.

8.4.3. Ничего не загружается, паника при загрузке

Такая ситуация произойдет, если загрузчик был уничтожен при установке vinum. К сожалению, vinum случайно оставляет свободными только первые 4 КБ в начале своего раздела перед записью заголовочной информации vinum. Однако, первая и вторая стадии загрузчика вместе с bsdlabel требуют 8 КБ. Поэтому, если раздел vinum начинается со смещения 0 внутри слайса или диска, который должен быть загрузочным, установка vinum

повредит загрузчик.

Аналогично, если описанная выше ситуация была исправлена загрузкой с "Fixit"-носителя, и загрузчик был переустановлен с помощью `bsdlabel -B`, как описано в этапе два, загрузчик повредит заголовок vinum, и vinum больше не сможет найти свои диски. Хотя фактические данные конфигурации vinum или данные в томах vinum не будут повреждены, и можно восстановить все данные, введя точно такую же конфигурацию vinum снова, исправить ситуацию сложно. Необходимо переместить весь раздел vinum как минимум на 4 КБ, чтобы заголовок vinum и системный загрузчик больше не конфликтовали.