Bootstrapping Vinum: A Foundation for Reliable Servers

Robert A. Van Valzah

$Id: VinumBootstrap.sgml,v 1.28 2001/10/14 14:08:39 bob Exp bob $

Copyright © 2001 Robert A. Van Valzah

$Date: 2001/10/14 14:08:39 $ GMT

FreeBSD is a registered trademark of the FreeBSD Foundation.
Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the "™" or the "®" symbol.

Table of Contents

In the most abstract sense, these instructions show how to build a pair of disk drives where either one is adequate to keep your server running if the other fails. Life is better if they are both working, but your server will never die unless both disk drives die at once. If you choose ATAPI drives and use a fairly generic kernel, you can be confident that either of these drives can be plugged into most any main board to produce a working server in a pinch. The drives need not be identical. These techniques work equally well with SCSI drives as they do with ATAPI, but I will focus on ATAPI here because main boards with this interface are ubiquitous. After building the foundation of a reliable server as shown here, you can expand to as many disk drives as necessary to build the failure-resilient server of your dreams.

1. Introduction

Any machine that is going to provide reliable service needs to have either redundant components on-line or a pool of off-line spares that can be promptly swapped in. Commodity PC hardware makes it affordable for even small organizations to have some spare parts available that could be pressed into service following the failure of production equipment. In many organizations, a failed power supply, NIC, memory, or main board could easily be swapped with a standby in a matter of minutes and be ready to return to production work.

If a disk drive fails, however, it often has to be restored from a tape backup. This may take many hours. With disk drive capacities rising faster than tape drive capacities, the time needed to restore a failed disk drive seems to increase as technology progresses.

Vinum is a volume manager for FreeBSD that provides a standard block I/O layer interface to the filesystem code just as any hardware device driver would. It works by managing partitions of type `vinum` and allows you to subdivide and group the space in such partitions into logical devices called *volumes* that can be used in the same way as disk partitions. Volumes can be configured for resilience, performance, or both. Experienced system administrators will immediately recognize the benefits of being able to configure each filesystem to match the way it is most often used.

In some ways, **Vinum** is similar to ccd(4), but it is far more flexible and robust in the face of failures. It is only slightly more difficult to set up than ccd(4). ccd(4) may meet your needs if you are only interested in concatenation.

1.1. Terminology

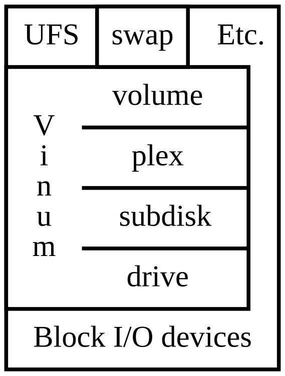
Discussion of storage management can get very tricky simply because of the terminology involved. As we will see below, the terms *disk*, *slice*, *partition*, *subdisk*, and *volume* each refer to different things that present the same interface to a kernel function like swapping. The potential for confusion is compounded because the objects that these terms represent can be nested inside each other.

I will refer to a physical disk drive as a *spindle*. A *partition* here means a BSD partition as maintained by `disklabel`. It does not refer to *slices* or *BIOS partitions* as maintained by `fdisk`.

1.2. Vinum Objects

Vinum defines a hierarchy of four objects that it uses to manage storage (see Figure 1). Different combinations of these objects are used to achieve failure resilience, performance, and/or extra capacity. I will give a whirlwind tour of the objects here--see the Vinum web site (http://www.vinumvm.org/) for a more thorough description.

Figure 1. Vinum Objects and Architecture



The top object, a vinum *volume*, implements a virtual disk that provides a standard block I/O layer interface to other parts of the kernel. The bottom object, a vinum *drive*, uses this same interface to request I/O from physical devices below it.

In between these two (from top to bottom) we have objects called a vinum *plex* and a vinum *subdisk*. As you can probably guess from the name, a vinum subdisk is a contiguous subset of the space available on a vinum drive. It lets you subdivide a vinum drive in much the same way that a disk BSD partition lets you subdivide a BIOS slice.

A plex allows subdisks to be grouped together making the space of all subdisks available as a single object.

A plex can be organized with its constituent subdisks concatenated or striped. Both organizations are useful for spreading I/O requests across spindles since plexes reside on distinct spindles. A striped plex will switch spindles each time a multiple of the stripe size is reached. A concatenated plex will switch spindles only when the end of a subdisk is reached.

An important characteristic of a **Vinum** volume is that it can be made up of more than one plex. In this case, writes go to all plexes and a read may be satisfied by any plex. Configuring two or more plexes on distinct spindles yields a volume that is resilient to failure.

Vinum maintains a *configuration* that defines instances of the above objects and the way they are related to each other. This configuration is automatically written to all spindles under **Vinum** management whenever it changes.

1.3. Vinum Volume/Plex Organization

Although **Vinum** can manage any number of spindles, I will only cover scenarios with two spindles here for simplification. See Table 1 to see how two spindles organized with **Vinum** compare to two spindles without **Vinum**.

Table 1. Characteristics of Two Spindles Organized with Vinum

Organization	Total Capacity	Failure Resilient	Peak Read Performance	Peak Write Performance
Concatenated Plexes	Unchanged, but appears as a single drive	No	Unchanged	Unchanged
Striped Plexes (RAID-0)	Unchanged, but appears as a single drive	No	2x	2x
Mirrored Volumes (RAID-1)	1/2, appearing as a single drive	Yes	2x	Unchanged

Table 1 shows that striping yields the same capacity and lack of failure resilience as concatenation, but it has better peak read and write performance. Hence we will not be using concatenation in any of the examples here. Mirrored volumes provide the benefits of improved peak read performance and failure resilience--but this comes at a loss in capacity.

> **Note:** Both concatenation and striping bring their benefits over a single spindle at the cost of increased likelihood of failure since more than one spindle is now involved.

When three or more spindles are present, **Vinum** also supports rotated, block-interleaved parity (also called *RAID-5*) that provides better capacity than mirroring (but not quite as good as striping), better read performance than both mirroring and striping, and good failure resilience. There is, however, a substantial decrease in write performance with RAID-5. Most of the benefits become more pronounced with five or more spindles.

The organizations described above may be combined to provide benefits that no single organization can match. For example, mirroring and striping can be combined to provide failure-resilience with very fast read performance.

1.4. Vinum History

Vinum is a standard part of even a "minimum" FreeBSD distribution and it has been standard since 3.0-RELEASE. The official pronunciation of the name is *VEE-noom*.

Vinum was inspired by the Veritas Volume Manager, but was not derived from it. The name is a play on that history and the Latin adage *In Vino Veritas* (*Vino* is the ablative form of *Vinum*). Literally translated, that is “Truth lies in wine” hinting that drunkards have a hard time lying.

I have been using it in production on six different servers for over two years with no data loss. Like the rest of FreeBSD, **Vinum** provides “rock-stable performance.” (On a personal note, I have seen **Vinum** panic when I misconfigured something, but I have never had any trouble in normal operation.) Greg Lehey wrote **Vinum** for FreeBSD, but he is seeking help in porting it to NetBSD and OpenBSD.

> **Warning:** Just like the rest of FreeBSD, **Vinum** is undergoing continuous development. Several subtle, but significant bugs have been fixed in recent releases. It is always best to use the most recent code base that meets your stability requirements.

1.5. Vinum Deployment Strategy

Vinum, coupled with prudent partition management, lets you keep “warm-spare” spindles on-line so that failures are transparent to users. Failed spindles can be replaced during regular maintenance periods or whenever it is convenient. When all spindles are working, the server benefits from increased performance and capacity.

Having redundant copies of your home directory does not help you if the spindle holding root, `/usr`, or swap fails on your server. Hence I focus here on building a simple foundation for a failure-resilient server covering the root, `/usr`, `/home`, and swap partitions.

> **Warning: Vinum** mirroring does not remove the need for making backups! Mirroring cannot help you recover from site disasters or the dreaded `rm -r -f /` command.

1.6. Why Bootstrap Vinum?

It is possible to add **Vinum** to a server configuration after it is already in production use, but this is much harder than designing for it from the start. Ironically, **Vinum** is not supported by `/stand/sysinstall` and hence you cannot install `/usr` right onto a **Vinum** volume.

> **Note: Vinum** currently does not support the root filesystem (this feature is in development).

Hence it is a bit tricky to get started using **Vinum**, but these instructions take you though the process of planning for **Vinum**, installing FreeBSD without it, and then beginning to use it.

I have come to call this whole process “bootstrapping Vinum.” That is, the process of getting **Vinum** initially installed and operating to the point where you have met your resilience or performance goals. My purpose here is to document a **Vinum** bootstrapping method that I have found that works well for me.

1.7. Vinum Benefits

The server foundation scenario I have chosen here allows me to show you examples of configuring for resilience on `/usr` and `/home`. Yet **Vinum** provides benefits other than resilience--namely performance, capacity, and manageability. It can significantly improve disk performance (especially under multi-user loads). **Vinum** can easily concatenate many smaller disks to produce the illusion of a single larger disk (but my server foundation scenario does not allow me to illustrate these benefits here).

For servers with many spindles, **Vinum** provides substantial benefits in volume management, particularly when coupled with hot-pluggable hardware. Data can be moved from spindle to spindle while the system is running without loss of production time. Again, details of this will not be given here, but once you get your feet wet with **Vinum**, other documentation will help you do things like this. See "The Vinum Volume Manager (http://www.vinumvm.org/vinum/vinum.ps)" for a technical introduction to **Vinum**, vinum(8) for a description of the `vinum` command, and vinum(4) for a description of the vinum device driver and the way **Vinum** objects are named.

> **Note:** Breaking up your disk space into smaller and smaller partitions has the benefit of allowing you to "tune" for the most common type of access and tends to keep disk hogs "within their pens." However it also causes some loss in total available disk space due to fragmentation.

1.8. Server Operation in Degraded Mode

Some disk failures in this two-spindle scenario will result in **Vinum** automatically routing all disk I/O to the remaining good spindle. Others will require brief manual intervention on the console to configure the server for degraded mode operation and a quick reboot. Other than actual hardware repairs, most recovery work can be done while the server is running in multi-user degraded mode so there is as little production impact from failures as possible.

I give the instructions in Section 4 needed to configure the server for degraded mode operation in those cases where **Vinum** cannot do it automatically. I also give the instructions needed to return to normal operation once the failed hardware is repaired. You might call these instructions **Vinum** failure recovery techniques.

I recommend practicing using these instructions by recovering from simulated failures. For each failure scenario, I also give tips below for simulating a failure even when your hardware is working well. Even a minimum **Vinum** system as described in Section 1.10 below can be a good place to experiment with recovery techniques without impacting production equipment.

1.9. Hardware RAID vs. Vinum (Software RAID)

Manual intervention is sometimes required to configure a server for degraded mode because **Vinum** is implemented in software that runs after the FreeBSD kernel is loaded. One disadvantage of such *software RAID* solutions is that there is nothing that can be done to hide spindle failures from the BIOS or the FreeBSD boot sequence. Hence the manual reconfiguration of the server for degraded operation mentioned above just informs the BIOS and boot sequence of failed spindles. *Hardware RAID* solutions generally have an advantage in that they require no such reconfiguration since spindle failures are hidden from the BIOS and boot sequence.

Hardware RAID, however, may have some disadvantages that can be significant in some cases:

- The hardware RAID controller itself may become a single point of failure for the system.

- The data is usually kept in a proprietary format so that a disk drive cannot be simply plugged into another main board and booted.
- You often cannot mix and match drives with different sizes and interfaces.
- You are often limited to the number of drives supported by the hardware RAID controller (often only four or eight).

In other words, **Vinum** may offer advantages in that there is no single point of failure, the drives can boot on most any main board, and you are free to mix and match as many drives using whatever interface you choose.

> **Tip:** Keep your kernel fairly generic (or at least keep `/kernel.GENERIC` around). This will improve the chances that you can come back up on "foreign" hardware more quickly.

The pros and cons discussed above suggest that the root filesystem and swap partition are good candidates for hardware RAID if available. This is especially true for servers where it is difficult for administrators to get console access (recall that this is sometimes required to configure a server for degraded mode operation). A server with only software RAID is well suited to office and home environments where an administrator can be close at hand.

> **Note:** A common myth is that hardware RAID is always faster than software RAID. Since it runs on the host CPU, **Vinum** often has more CPU power and memory available than a dedicated RAID controller would have. If performance is a prime concern, it is best to benchmark your application running on your CPU with your spindles using both hardware and software RAID systems before making a decision.

1.10. Hardware for Vinum

These instructions may be timely since commodity PC hardware can now easily host several hundred gigabytes of reasonably high-performance disk space at a low price. Many disk drive manufactures now sell 7,200 RPM disk drives with quite low seek times and high transfer rates through ATA-100 interfaces, all at very attractive prices. Four such drives, attached to a suitable main board and configured with **Vinum** and prudent partitioning, yields a failure-resilient, high performance disk server at a very reasonable cost.

However, you can indeed get started with **Vinum** very simply. A minimum system can be as simple as an old CPU (even a 486 is fine) and a pair of drives that are 500 MB or more. They need not be the same size or even use the same interface (i.e., it is fine to mix ATAPI and SCSI). So get busy and give this a try today! You will have the foundation of a failure-resilient server running in an hour or so!

2. Bootstrapping Phases

Greg Lehey suggested this bootstrapping method. It uses knowledge of how **Vinum** internally allocates disk space to avoid copying data. Instead, **Vinum** objects are configured so that they occupy the same disk space where `/stand/sysinstall` built filesystems. The filesystems are thus embedded within **Vinum** objects without copying.

There are several distinct phases to the **Vinum** bootstrapping procedure. Each of these phases is presented in a separate section below. The section starts with a general overview of the phase and its goals. It then gives example steps for the two-spindle scenario presented here and advice on how to adapt them for your server. (If you are reading for a general understanding of **Vinum** bootstrapping, the example sections for each phase can safely be skipped.) The remainder of this section gives an overview of the entire bootstrapping process.

Phase 1 involves planning and preparation. We will balance requirements for the server against available resources and make design tradeoffs. We will plan the transition from no **Vinum** to **Vinum** on just one spindle, to **Vinum** on two spindles.

In phase 2, we will install a minimum FreeBSD system on a single spindle using partitions of type `4.2BSD` (regular UFS filesystems).

Phase 3 will embed the non-root filesystems from phase 2 in **Vinum** objects. Note that **Vinum** will be up and running at this point, but it cannot yet provide any resilience since it only has one spindle on which to store data.

Finally in phase 4, we configure **Vinum** on a second spindle and make a backup copy of the root filesystem. This will give us resilience on all filesystems.

2.1. Bootstrapping Phase 1: Planning and Preparation

Our goal in this phase is to define the different partitions we will need and examine their requirements. We will also look at available disk drives and controllers and allocate partitions to them. Finally, we will determine the size of each partition and its use during the bootstrapping process. After this planning is complete, we can optionally prepare to use some tools that will make bootstrapping **Vinum** easier.

Several key questions must be answered in this planning phase:

- What filesystem and partitions will be needed?
- How will they be used?
- How will we name each spindle?
- How will the partitions be ordered for each spindle?
- How will partitions be assigned to the spindles?
- How will partitions be configured? Resilience or performance?
- What technique will be used to achieve resilience?
- What spindles will be used?
- How will they be configured on the available controllers?
- How much space is required for each partition?

2.1.1. Phase 1 Example

In this example, I will assume a scenario where we are building a minimal foundation for a failure-resilient server. Hence we will need at least root, `/usr`, `/home`, and swap partitions. The root, `/usr`, and `/home` filesystems all need resilience since the server will not be much good without them. The swap partition needs performance first and generally does not need resilience since nothing it holds needs to be retained across a reboot.

2.1.1.1. Spindle Naming

The kernel would refer to the master spindle on the primary and secondary ATA controllers as `/dev/ad0` and `/dev/ad2` respectively. [1] But **Vinum** also needs to have a name for each spindle that will stay the same name regardless of how it is attached to the CPU (i.e., if the drive moves, the **Vinum** name moves with the drive).

Some recovery techniques documented below suggest moving a spindle from the secondary ATA controller to the primary ATA controller. (Indeed, the flexibility of making such moves is a key benefit of **Vinum** especially if you are managing a large number of spindles.) After such a drive/controller swap, the kernel will see what used to be `/dev/ad2` as `/dev/ad0` but **Vinum** will still call it by whatever name it had when it was attached to `/dev/ad2` (i.e., when it was "created" or first made known to **Vinum**).

Since connections can change, it is best to give each spindle a unique, abstract name that gives no hint of how it is attached. Avoid names that suggest a manufacturer, model number, physical location, or membership in a sequence (e.g. avoid names like `upper`, `lower`, etc., `alpha`, `beta`, etc., `SCSI1`, `SCSI2`, etc., or `Seagate1`, `Seagate2` etc.). Such names are likely to lose their uniqueness or get out of sequence someday even if they seem like great names today.

> **Tip:** Once you have picked names for your spindles, label them with a permanent marker. If you have hot-swappable hardware, write the names on the sleds in which the spindles are mounted. This will significantly reduce the likelihood of error when you are moving spindles around later as part of failure recovery or routine system management procedures.

In the instructions that follow, **Vinum** will name the root spindle `YouCrazy` and the rootback spindle `UpWindow`. I will only use `/dev/ad0` when I want to refer to whichever of the two spindles is currently attached as `/dev/ad0`.

2.1.1.2. Partition Ordering

Modern disk drives operate with fairly uniform areal density across the surface of the disk. That implies that more data is available under the heads without seeking on the outer cylinders than on the inner cylinders. We will allocate partitions most critical to system performance from these outer cylinders as `/stand/sysinstall` generally does.

The root filesystem is traditionally the outermost, even though it generally is not as critical to system performance as others. (However root can have a larger impact on performance if it contains `/tmp` and `/var` as it does in this example.) The FreeBSD boot loaders assume that the root filesystem lives in the `a` partition. There is no requirement that the `a` partition start on the outermost cylinders, but this convention makes it easier to manage disk labels.

Swap performance is critical so it comes next on our way toward the center. I/O operations here tend to be large and contiguous. Having as much data under the heads as possible avoids seeking while swapping.

With all the smaller partitions out of the way, we finish up the disk with `/home` and `/usr`. Access patterns here tend not to be as intense as for other filesystems (especially if there is an abundant supply of RAM and read cache hit rates are high).

If the pair of spindles you have are large enough to allow for more than `/home` and `/usr`, it is fine to plan for additional filesystems here.

2.1.1.3. Assigning Partitions to Spindles

We will want to assign partitions to these spindles so that either can fail without loss of data on filesystems configured for resilience.

Reliability on `/usr` and `/home` is best achieved using **Vinum** mirroring. Resilience will have to come differently, however, for the root filesystem since **Vinum** is not a part of the FreeBSD boot sequence. Here we will have to settle for two identical partitions with a periodic copy from the primary to the backup secondary.

The kernel already has support for interleaved swap across all available partitions so there is no need for help from **Vinum** here. `/stand/sysinstall` will automatically configure `/etc/fstab` for all swap partitions given.

The **Vinum** bootstrapping method given below requires a pair of spindles that I will call the *root spindle* and the *rootback spindle*.

> **Important:** The rootback spindle must be the same size or larger than the root spindle.

These instructions first allocate all space on the root spindle and then allocate exactly that amount of space on a rootback spindle. (After **Vinum** is bootstrapped, there is nothing special about either of these spindles--they are interchangeable.) You can later use the remaining space on the rootback spindle for other filesystems.

If you have more than two spindles, the `bootvinum` Perl script and the procedure below will help you initialize them for use with **Vinum**. However you will have to figure out how to assign partitions to them on your own.

2.1.1.4. Assigning Space to Partitions

For this example, I will use two spindles: one with 4,124,673 blocks (about 2 GB) on `/dev/ad0` and one with 8,420,769 blocks (about 4 GB) on `/dev/ad2`.

It is best to configure your two spindles on separate controllers so that both can operate in parallel and so that you will have failure resilience in case a controller dies. Note that mirrored volume write performance will be halved in cases where both spindles share a controller that requires they operate serially (as is often the case with ATA controllers). One spindle will be the master on the primary ATA controller and the other will be the master on the secondary ATA controller.

Recall that we will be allocating space on the smaller spindle first and the larger spindle second.

2.1.1.5. Assigning Partitions on the Root Spindle

We will allocate 200,000 blocks (about 93 MB) for a root filesystem on each spindle (`/dev/ad0s1a` and `/dev/ad2s1a`). We will initially allocate 200,265 blocks for a swap partition on each spindle, giving a total of about 186 MB of swap space (`/dev/ad0s1b` and `/dev/ad2s1b`).

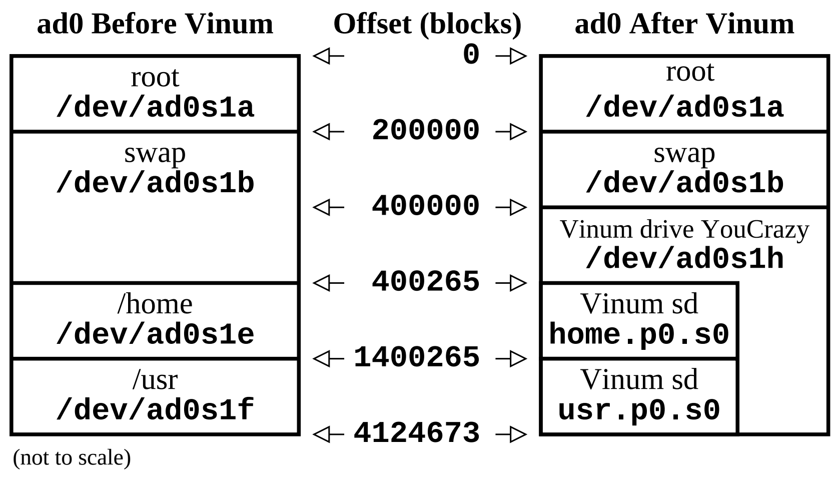
> **Note:** We will lose 265 blocks from each swap partition as part of the bootstrapping process. This is the size of the space used by **Vinum** to store configuration information. The space will be taken from swap and given to a vinum partition but will be unavailable for **Vinum** subdisks.

> **Note:** I have done the partition allocation in nice round numbers of blocks just to emphasize where the 265 blocks go. There is nothing wrong with allocating space in MB if that is more convenient for you.

This leaves 4,124,673 - 200,000 - 200,265 = 3,724,408 blocks (about 1,818 MB) on the root spindle for **Vinum** partitions (`/dev/ad0s1e` and `/dev/ad2s1f`). From this, allocate the 265 blocks for **Vinum** configuration information, 1,000,000 blocks (about 488 MB) for `/home`, and the remaining 2,724,408 blocks (about 1,330 MB) for `/usr`. See Figure 2 below to see this graphically.

The left-hand side of Figure 2 below shows what spindle ad0 will look like at the end of phase 2. The right-hand side shows what it will look like at the end of phase 3.

Figure 2. Spindle ad0 Before and After Vinum

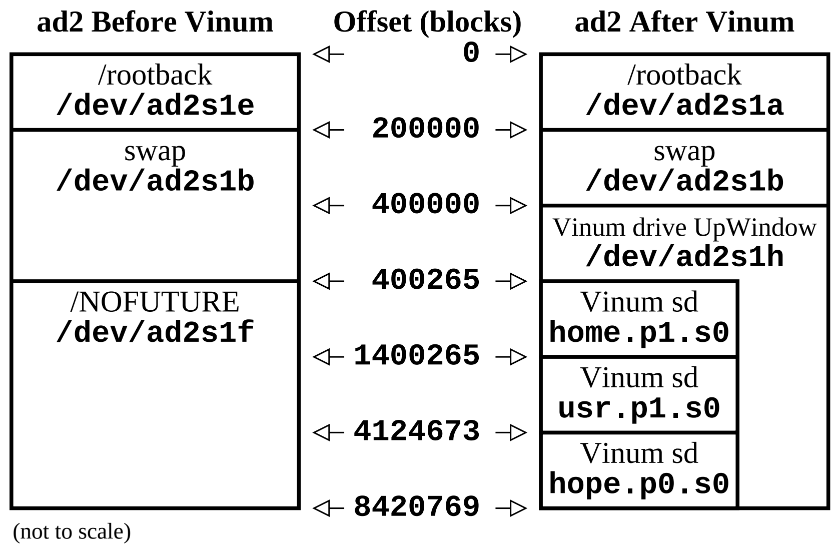


2.1.1.6. Assigning Partitions on the Rootback Spindle

The `/rootback` and swap partition sizes on the rootback spindle must match the root and swap partition sizes on the root spindle. That leaves 8,420,769 - 200,000 - 200,265 = 8,020,504 blocks for the **Vinum** partition. Mirrors of `/home` and `/usr` receive the same allocation as on the root spindle. That will leave an extra 2 GB or so that we can deal with later. See Figure 3 below to see this graphically.

The left-hand side of Figure 3 below shows what spindle ad2 will look like at the beginning of phase 4. The right-hand side shows what it will look like at the end.

Figure 3. Spindle ad2 Before and After Vinum



2.1.1.7. Preparation of Tools

The `bootvinum` Perl script given below in Appendix A will make the **Vinum** bootstrapping process much easier if you can run it on the machine being bootstrapped. It is over 200 lines and you would not want to type it in. At this point, I recommend that you copy it to a floppy or arrange some alternative method of making it readily available so that it can be available later when needed. For example:

```
# fdformat -f 1440 /dev/fd0
# newfs_msdos -f 1440 /dev/fd0
# mount_msdos /dev/fd0 /mnt
# cp /usr/share/examples/vinum/bootvinum /mnt
```

XXX Someday, I would like this script to live in `/usr/share/examples/vinum`. Till then, please use this link (http://www.BGPBook.Com/vinum/bootvinum) to get a copy.

2.2. Bootstrapping Phase 2: Minimal OS Installation

Our goal in this phase is to complete the smallest possible FreeBSD installation in such a way that we can later install **Vinum**. We will use only partitions of type `4.2BSD` (i.e., regular UFS file systems) since that is the only type supported by `/stand/sysinstall`.

2.2.1. Phase 2 Example

1. Start up the FreeBSD installation process by running `/stand/sysinstall` from installation media as you normally would.
2. Fdisk partition all spindles as needed.

   Important: Make sure to select BootMgr for all spindles.

3. Partition the root spindle with appropriate block allocations as described above in Section 2.1.1.5. For this example on a 2 GB spindle, I will use 200,000 blocks for root, 200,265 blocks for swap, 1,000,000 blocks for `/home`, and the rest of the spindle (2,724,408 blocks) for `/usr`. (`/stand/sysinstall` should automatically assign these to `/dev/ad0s1a`, `/dev/ad0s1b`, `/dev/ad0s1e`, and `/dev/ad0s1f` by default.)

   Note: If you prefer Soft Updates as I do and you are using 4.4-RELEASE or better, this is a good time to enable them.

4. Partition the rootback spindle with the appropriate block allocations as described above in Section 2.1.1.6. For this example on a 4 GB spindle, I will use 200,000 blocks for `/rootback`, 200,265 blocks for swap, and the rest of the spindle (8,020,504 blocks) for `/NOFUTURE`. (`/stand/sysinstall` should automatically assign these to `/dev/ad2s1e`, `/dev/ad2s1b`, and `/dev/ad2s1f` by default.)

   Note: We do not really want to have a `/NOFUTURE` UFS filesystem (we want a vinum partition instead), but that is the best choice we have for the space given the limitations of `/stand/sysinstall`. Mount point names beginning with `NOFUTURE` and `rootback` serve as sentinels to the bootstrapping script presented in Appendix A below.

5. Partition any other spindles with swap if desired and a single `/NOFUTURExx` filesystem.
6. Select a minimum system install for now even if you want to end up with more distributions loaded later.

   Tip: Do not worry about system configuration options at this point--get **Vinum** set up and get the partitions in the right places first.

7. Exit `/stand/sysinstall` and reboot. Do a quick test to verify that the minimum installation was successful.

The left-hand side of Figure 2 above and the left-hand side of Figure 3 above show how the disks will look at this point.

2.3. Bootstrapping Phase 3: Root Spindle Setup

Our goal in this phase is get **Vinum** set up and running on the root spindle. We will embed the existing `/usr` and `/home` filesystems in a **Vinum** partition. Note that the **Vinum** volumes created will not yet be failure-resilient since we have only one underlying **Vinum** drive to hold them. The resulting system will automatically start **Vinum** as it boots to multi-user mode.

2.3.1. Phase 3 Example

1. Login as root.
2. We will need a directory in the root filesystem in which to keep a few files that will be used in the **Vinum** bootstrapping process.

```
# mkdir /bootvinum
# cd /bootvinum
```

3. Several files need to be prepared for use in bootstrapping. I have written a Perl script that makes all the required files for you. Copy this script to `/bootvinum` by floppy disk, tape, network, or any convenient means and then run it. (If you cannot get this script copied onto the machine being bootstrapped, then see Appendix B below for a manual alternative.)

```
# cp /mnt/bootvinum .
# ./bootvinum
```

   Note: `bootvinum` produces no output when run successfully. If you get any errors, something may have gone wrong when you were creating partitions with `/stand/sysinstall` above.

   Running `bootvinum` will:

   - Create `/etc/fstab.vinum` based on what it finds in your existing `/etc/fstab`
   - Create new disk labels for each spindle mentioned in `/etc/fstab` and keep copies of the current disk labels

- Create files needed as input to `vinum create` for building **Vinum** objects on each spindle
- Create many alternates to `/etc/fstab.vinum` that might come in handy should a spindle fail

You may want to take a look at these files to learn more about the disk partitioning required for **Vinum** or to learn more about the commands needed to create **Vinum** objects.

4. We now need to install new spindle partitioning for `/dev/ad0`. This requires that `/dev/ad0s1b` not be in use for swapping so we have to reboot in single-user mode.

    a. First, reboot the system.

    ```
    # reboot
    ```

    b. Next, enter single-user mode.

    ```
    Hit [Enter] to boot immediately, or any other key for command prompt.
    Booting [kernel] in 8 seconds...

    Type '?' for a list of commands, 'help' for more detailed help.
    ok boot -s
    ```

5. In single-user mode, install the new partitioning created above.

    ```
    # cd /bootvinum
    # disklabel -R ad0s1 disklabel.ad0s1
    # disklabel -R ad2s1 disklabel.ad2s1
    ```

    Note: If you have additional spindles, repeat the above commands as appropriate for them.

6. We are about to start **Vinum** for the first time. It is going to want to create several device nodes under `/dev/vinum` so we will need to mount the root filesystem for read/write access.

    ```
    # fsck -p /
    # mount /
    ```

7. Now it is time to create the **Vinum** objects that will embed the existing non-root filesystems on the root spindle in a **Vinum** partition. This will load the **Vinum** kernel module and start **Vinum** as a side effect.

    ```
    # vinum create create.YouCrazy
    ```

    You should see a list of **Vinum** objects created that looks like the following:

    ```
    1 drives:
    D YouCrazy              State: up Device /dev/ad0s1h Avail: 0/1818 MB (0%)

    2 volumes:
    V home                  State: up Plexes:       1 Size:        488 MB
    V usr                   State: up Plexes:       1 Size:       1330 MB

    2 plexes:
    P home.p0             C State: up Subdisks:     1 Size:        488 MB
    P usr.p0              C State: up Subdisks:     1 Size:       1330 MB

    2 subdisks:
    S home.p0.s0            State: up PO:        0  B Size:        488 MB
    S usr.p0.s0             State: up PO:        0  B Size:       1330 MB
    ```

You should also see several kernel messages which state that the **Vinum** objects you have created are now `up`.

8. Our non-root filesystems should now be embedded in a **Vinum** partition and hence available through **Vinum** volumes. It is important to test that this embedding worked.

   ```
   # fsck -n /dev/vinum/home
   # fsck -n /dev/vinum/usr
   ```

   This should produce no errors. If it does produce errors *do not fix them*. Instead, go back and examine the root spindle partition tables before and after **Vinum** to see if you can spot the error. You can back out the partition table changes by using `disklabel -R` with the `disklabel.*.b4vinum` files.

9. While we have the root filesystem mounted read/write, this is a good time to install `/etc/fstab`.

   ```
   # mv /etc/fstab /etc/fstab.b4vinum
   # cp /etc/fstab.vinum /etc/fstab
   ```

10. We are now done with tasks requiring single-user mode, so it is safe to go multi-user from here on.

    ```
    # ^D
    ```

11. Login as root.

12. Edit `/etc/rc.conf` and add this line:

    ```
    start_vinum="YES"
    ```

2.4. Bootstrapping Phase 4: Rootback Spindle Setup

Our goal in this phase is to get redundant copies of all data from the root spindle to the rootback spindle. We will first create the necessary **Vinum** objects on the rootback spindle. Then we will ask **Vinum** to copy the data from the root spindle to the rootback spindle. Finally, we use `dump` and `restore` to copy the root filesystem.

2.4.1. Phase 4 Example

1. Now that **Vinum** is running on the root spindle, we can bring it up on the rootback spindle so that our **Vinum** volumes can become failure-resilient.

   ```
   # cd /bootvinum
   # vinum create create.UpWindow
   ```

   You should see a list of **Vinum** objects created that looks like the following:

   ```
   2 drives:
   D YouCrazy              State: up       Device /dev/ad0s1h      Avail: 0/1818 MB (0%)
   D UpWindow              State: up       Device /dev/ad2s1h      Avail: 2096/3915 MB (53%)

   2 volumes:
   V home                  State: up       Plexes:       2 Size:        488 MB
   V usr                   State: up       Plexes:       2 Size:       1330 MB

   4 plexes:
   P home.p0             C State: up       Subdisks:     1 Size:        488 MB
   P usr.p0              C State: up       Subdisks:     1 Size:       1330 MB
   P home.p1             C State: faulty   Subdisks:     1 Size:        488 MB
   P usr.p1              C State: faulty   Subdisks:     1 Size:       1330 MB
   ```

```
4 subdisks:
S home.p0.s0            State: up       PO:         0  B Size:         488 MB
S usr.p0.s0             State: up       PO:         0  B Size:        1330 MB
S home.p1.s0            State: stale    PO:         0  B Size:         488 MB
S usr.p1.s0             State: stale    PO:         0  B Size:        1330 MB
```

You should also see several kernel messages which state that some of the **Vinum** objects you have created are now `up` while others are `faulty` or `stale`.

2. Now we ask **Vinum** to copy each of the subdisks on drive `YouCrazy` to drive `UpWindow`. This will change the state of the newly created **Vinum** subdisks from `stale` to `up`. It will also change the state of the newly created **Vinum** plexes from `faulty` to `up`.

   First, we do the new subdisk we added to `/home`.

```
# vinum start -w home.p1.s0
reviving home.p1.s0
(time passes . . . )
home.p1.s0 is up by force
home.p1 is up
home.p1.s0 is up
```

   Note: My 5,400 RPM EIDE spindles copied at about 3.5 MBytes/sec. Your mileage may vary.

3. Next we do the new subdisk we added to `/usr`.

```
# vinum start -w usr.p1.s0
reviving usr.p1.s0
(time passes . . . )
usr.p1.s0 is up by force
usr.p1 is up
usr.p1.s0 is up
```

   All **Vinum** objects should be in state `up` at this point. The output of `vinum list` should look like the following:

```
2 drives:
D YouCrazy              State: up Device /dev/ad0s1h Avail: 0/1818 MB (0%)
D UpWindow              State: up Device /dev/ad2s1h Avail: 2096/3915 MB (53%)

2 volumes:
V home                  State: up Plexes:       2 Size:        488 MB
V usr                   State: up Plexes:       2 Size:       1330 MB

4 plexes:
P home.p0             C State: up Subdisks:     1 Size:        488 MB
P usr.p0              C State: up Subdisks:     1 Size:       1330 MB
P home.p1             C State: up Subdisks:     1 Size:        488 MB
P usr.p1              C State: up Subdisks:     1 Size:       1330 MB

4 subdisks:
S home.p0.s0            State: up PO:        0  B Size:        488 MB
S usr.p0.s0             State: up PO:        0  B Size:       1330 MB
S home.p1.s0            State: up PO:        0  B Size:        488 MB
```

```
S usr.p1.s0              State: up PO:        0  B Size:       1330 MB
```

4. Copy the root filesystem so that you will have a backup.

```
# cd /rootback
# dump 0f - / | restore rf -
# rm restoresymtable
# cd /
```

> **Note:** You may see errors like this:
>
> ```
> ./tmp/rstdir1001216411: (inode 558) not found on tape
> cannot find directory inode 265
> abort? [yn] n
> expected next file 492, got 491
> ```
>
> They seem to cause no harm. I suspect they are a consequence of dumping the filesystem containing `/tmp` and/or the pipe connecting `dump` and `restore`.

5. Make a directory on which we can mount a damaged root filesystem during the recovery process.

```
# mkdir /rootbad
```

6. Remove sentinel mount points that are now unused.

```
# rmdir /NOFUTURE*
```

7. Create empty **Vinum** drives on remaining spindles.

```
# vinum create create.ThruBank
# ...
```

At this point, the reliable server foundation is complete. The right-hand side of Figure 2 above and the right-hand side of Figure 3 above show how the disks will look.

You may want to do a quick reboot to multi-user and give it a quick test drive. This is also a good point to complete installation of other distributions beyond the minimal install. Add packages, ports, and users as required. Configure `/etc/rc.conf` as required.

> **Tip:** After you have completed your server configuration, remember to do one more copy of root to `/rootback` as shown above before placing the server into production.

> **Tip:** Make a schedule to refresh `/rootback` periodically.

> **Tip:** It may be a good idea to mount `/rootback` read-only for normal operation of the server. This does, however, complicate the periodic refresh a bit.

> **Tip:** Do not forget to watch `/var/log/messages` carefully for errors. **Vinum** may automatically avoid failed hardware in a way that users do not notice. You must watch for such failures and get them repaired before a second failure results in data loss. You may see **Vinum** noting damaged objects at server boot time.

3. Where to Go from Here?

Now that you have established the foundation of a reliable server, there are several things you might want to try next.

3.1. Make a Vinum Volume with Remaining Space

Following are the steps to create another **Vinum** volume with space remaining on the rootback spindle.

> **Note:** This volume will not be resilient to spindle failure since it has only one plex on a single spindle.

1. Create a file with the following contents:

   ```
   volume hope
     plex name hope.p0 org concat volume hope
       sd name hope.p0.s0 drive UpWindow plex hope.p0 len 0
   ```

   > **Note:** Specifying a length of `0` for the `hope.p0.s0` subdisk asks **Vinum** to use whatever space is left available on the underlying drive.

2. Feed these commands into `vinum create`.

   ```
   # vinum create filename
   ```

3. Now we `newfs` the volume and `mount` it.

   ```
   # newfs -v /dev/vinum/hope
   # mkdir /hope
   # mount /dev/vinum/hope /hope
   ```

4. Edit `/etc/fstab` if you want `/hope` mounted at boot time.

3.2. Try Out More Vinum Commands

You might already be familiar with `vinum list` to get a list of all **Vinum** objects. Try `-v` following it to see more detail.

If you have more spindles and you want to bring them up as concatenated, mirrored, or striped volumes, then give `vinum concat` *`drivelist`*, `vinum mirror` *`drivelist`*, or `vinum stripe` *`drivelist`* a try.

See vinum(8) for sample configurations and important performance considerations before settling on a final organization for your additional spindles.

The failure recovery instructions below will also give you some experience using more **Vinum** commands.

4. Failure Scenarios

This section contains descriptions of various failure scenarios. For each scenario, there is a subsection on how to configure your server for degraded mode operation, how to recover from the failure, how to exit degraded mode, and how to simulate the failure.

> **Tip:** Make a hard copy of these instructions and leave them inside the CPU case, being careful not to interfere with ventilation.

4.1. Root filesystem on ad0 unusable, rest of drive ok

> **Note:** We assume here that the boot blocks and disk label on `/dev/ad0` are ok. If your BIOS can boot from a drive other than `C:`, you may be able to get around this limitation.

4.1.1. Configure Server for Degraded Mode

1. Use BootMgr to load kernel from `/dev/ad2s1a`.
   a. Hit **F5** in BootMgr to select `Drive 1`.
   b. Hit **F1** to select `FreeBSD`.
2. After the kernel is loaded, hit any key but enter to interrupt the boot sequence. Boot into single-user mode and allow explicit entry of a root filesystem.

```
Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [kernel] in 8 seconds...

Type '?' for a list of commands, 'help' for more detailed help.
ok boot -as
```

3. Select `/rootback` as your root filesystem.

```
Manual root filesystem specification:
    <fstype>:<device>  Mount <device> using filesystem <fstype>
    e.g. ufs:/dev/da0s1a
    ?                  List valid disk boot devices
    <empty line>       Abort manual input

  mountroot> ufs:/dev/ad2s1a
```

4. Now that you are in single-user mode, change `/etc/fstab` to avoid the bad root filesystem.

> **Tip:** If you used the `bootvinum` Perl script from Appendix A below, then these commands should configure your server for degraded mode.
>
> ```
> # fsck -p /
> # mount /
> # cd /etc
> # mv fstab fstab.bak
> # cp fstab_ad0s1_root_bad fstab
> ```

```
# cd /
# mount -o ro /
# vinum start
# fsck -p
# ^D
```

4.1.2. Recovery

1. Restore `/dev/ad0s1a` from backups or copy `/rootback` to it with these commands:

   ```
   # umount /rootbad
   # newfs /dev/ad0s1a
   # tunefs -n enable /dev/ad0s1a
   # mount /rootbad
   # cd /rootbad
   # dump 0f - / | restore rf -
   # rm restoresymtable
   ```

4.1.3. Exiting Degraded Mode

1. Enter single-user mode.

   ```
   # shutdown now
   ```

2. Put `/etc/fstab` back to normal and reboot.

   ```
   # cd /rootbad/etc
   # rm fstab
   # mv fstab.bak fstab
   # reboot
   ```

3. Reboot and hit **F1** to boot from `/dev/ad0` when prompted by BootMgr.

4.1.4. Simulation

This kind of failure can be simulated by shutting down to single-user mode and then booting as shown above in Section 4.1.1.

4.2. Drive ad2 Fails

This section deals with the total failure of `/dev/ad2`.

4.2.1. Configure Server for Degraded Mode

1. After the kernel is loaded, hit any key but **Enter** to interrupt the boot sequence. Boot into single-user mode.

   ```
   Hit [Enter] to boot immediately, or any other key for command prompt.
   ```

```
Booting [kernel] in 8 seconds...

Type '?' for a list of commands, 'help' for more detailed help.
ok boot -s
```

2. Change `/etc/fstab` to avoid the bad drive. If you used the `bootvinum` Perl script from Appendix A below, then these commands should configure your server for degraded mode.

   ```
   # fsck -p /
   # mount /
   # cd /etc
   # mv fstab fstab.bak
   # cp fstab_only_have_ad0s1 fstab
   # cd /
   # mount -o ro /
   # vinum start
   # fsck -p
   # ^D
   ```

   If you do not have modified versions of `/etc/fstab` that are ready for use, then you can use `ed` to make one. Alternatively, you can `fsck` and `mount /usr` and then use your favorite editor.

4.2.2. Recovery

We assume here that your server is up and running multi-user in degraded mode on just `/dev/ad0` and that you have a new spindle now on `/dev/ad2` ready to go.

You will need a new spindle with enough room to hold root and swap partitions plus a **Vinum** partition large enough to hold `/home` and `/usr`.

1. Create a BIOS partition (slice) on the new spindle.

   ```
   # /stand/sysinstall
   ```

   a. Select `Custom`.
   b. Select `Partition`.
   c. Select `ad2`.
   d. Create a FreeBSD (type 165) slice large enough to hold everything mentioned above.
   e. Write changes.
   f. Yes, you are absolutely sure.
   g. Select BootMgr.
   h. Quit Partitioning.
   i. Exit `/stand/sysinstall`.

2. Create disk label partitioning based on current `/dev/ad0` partitioning.

   ```
   # disklabel ad0 > /tmp/ad0
   # disklabel -e ad2
   ```

   This will drop you into your favorite editor.

   a. Copy the lines for the `a` and `b` partitions from `/tmp/ad0` to the `ad2` disklabel.
   b. Add the `size` of the `a` and `b` partitions to find the proper `offset` for the `h` partition.
   c. Subtract this `offset` from the `size` of the `c` partition to find the proper `size` for the `h` partition.
   d. Define an `h` partition with the `size` and `offset` calculated above.
   e. Set the `fstype` column to `vinum`.
   f. Save the file and quit your editor.

3. Tell **Vinum** about the new drive.

   a. Ask **Vinum** to start an editor with a copy of the current configuration.

      ```
      # vinum create
      ```

   b. Uncomment the drive line referring to drive `UpWindow` and set `device` to `/dev/ad2s1h`.
   c. Save the file and quit your editor.

4. Now that **Vinum** has two spindles again, revive the mirrors.

   ```
   # vinum start -w usr.p1.s0
   # vinum start -w home.p1.s0
   ```

5. Now we need to restore `/rootback` to a current copy of the root filesystem. These commands will accomplish this.

   ```
   # newfs /dev/ad2s1a
   # tunefs -n enable /dev/ad2s1a
   # mount /dev/ad2s1a /mnt
   # cd /mnt
   # dump 0f - / | restore rf -
   # rm restoresymtable
   # cd /
   # umount /mnt
   ```

4.2.3. Exiting Degraded Mode

1. Enter single-user mode.

   ```
   # shutdown now
   ```

2. Return `/etc/fstab` to its normal state and reboot.

   ```
   # cd /etc
   # rm fstab
   # mv fstab.bak fstab
   # reboot
   ```

4.2.4. Simulation

You can simulate this kind of failure by unplugging `/dev/ad2`, write-protecting it, or by this procedure:

1. Shutdown to single-user mode.

2. Unmount all non-root filesystems.
3. Clobber any existing **Vinum** configuration and partitioning on `/dev/ad2`.

   ```
   # vinum stop
   # dd if=/dev/zero of=/dev/ad2s1h count=512
   # dd if=/dev/zero of=/dev/ad2 count=512
   ```

4.3. Drive ad0 Fails

Some BIOSes can boot from drive 1 or drive 2 (often called `C:` or `D:`), while others can boot only from drive 1. If your BIOS can boot from either, the fastest road to recovery might be to boot directly from `/dev/ad2` in single-user mode and install `/etc/fstab_only_have_ad2s1` as `/etc/fstab`. You would then have to adapt the `/dev/ad2` failure recovery instructions from Section 4.2.2 above.

If your BIOS can only boot from drive one, then you will have to unplug drive `YouCrazy` from the controller for `/dev/ad2` and plug it into the controller for `/dev/ad0`. Then continue with the instructions for `/dev/ad2` failure recovery in Section 4.2.2 above.

A. bootvinum Perl Script

The `bootvinum` Perl script below reads `/etc/fstab` and current drive partitioning. It then writes several files in the current directory and several variants of `/etc/fstab` in `/etc`. These files significantly simplify the installation of **Vinum** and recovery from spindle failures.

```
#!/usr/bin/perl -w
use strict;
use FileHandle;

my $config_tag1 = '$Id: VinumBootstrap.sgml,v 1.28 2001/10/14 14:08:39 bob Exp bob $';
# Copyright (C) 2001 Robert A. Van Valzah
#
# Bootstrap Vinum
#
# Read /etc/fstab and current partitioning for all spindles mentioned there.
# Generate files needed to mirror all filesystems on root spindle.
#  A new partition table for each spindle
#  Input for the vinum create command to create Vinum objects on each spindle
#  A copy of fstab mounting Vinum volumes instead of BSD partitions
#  Copies of fstab altered for server's degraded modes of operation
# See handbook for instructions on how to use the files generated.
# N.B. This bootstrapping method shrinks size of swap partition by the size
# of Vinum's on-disk configuration (265 sectors).  It embeds existing file
# systems on the root spindle in Vinum objects without having to copy them.
# Thanks to Greg Lehey for suggesting this bootstrapping method.
# Expectations:
#  The root spindle must contain at least root, swap, and /usr partitions
#  The rootback spindle must have matching /rootback and swap partitions
```

```
#  Other spindles should only have a /NOFUTURE* filesystem and maybe swap
#  File systems named /NOFUTURE* will be replaced with Vinum drives

# Change configuration variables below to suit your taste
my $vip = 'h';     # VInum Partition
my @drv = ('YouCrazy', 'UpWindow', 'ThruBank', # Vinum DRiVe names
  'OutSnakes', 'MeWild', 'InMovie', 'HomeJames', 'DownPrices', 'WhileBlind');
# No configuration variables beyond this point

my %vols;  # One entry per Vinum volume to be created
my @spndl;  # One entry per SPiNDLe
my $rsp;  # Root SPindle (as in /dev/$rsp)
my $rbsp;  # RootBack SPindle (as in /dev/$rbsp)
my $cfgsiz = 265; # Size of Vinum on-disk configuration info in sectors
my $nxtpas = 2;  # Next fsck pass number for non-root filesystems

# Parse fstab, generating the version we'll need for Vinum and noting
# spindles in use.
my $fsin = "/etc/fstab";
#my $fsin = "simu/fstab";
open(FSIN, "$fsin") || die("Couldn't open $fsin: $!\n");

my $fsout = "/etc/fstab.vinum";
open(FSOUT, ">$fsout") || die("Couldn't open $fsout for writing: $!\n");

while (<FSIN>) {
  my ($dev, $mnt, $fstyp, $opt, $dump, $pass) = split;
  next if $dev =~ /^#/;
  if ($mnt eq '/' || $mnt eq '/rootback' || $mnt =~ /^\/NOFUTURE/) {
    my $dn = substr($dev, 5, length($dev)-6); # Device Name without /dev/
    push(@spndl, $dn) unless grep($_ eq $dn, @spndl);
    $rsp = $dn if $mnt eq '/';
    next if $mnt =~ /^\/NOFUTURE/;
  }
  # Move /rootback from partition e to a
  if ($mnt =~ /^\/rootback/) {
    $dev =~ s/e$/a/;
    $pass = 1;
    $rbsp = substr($dev, 5, length($dev)-6);
    print FSOUT "$dev\t\t$mnt\t$fstyp\t$opt\t\t$dump\t$pass\n";
    next;
  }
  # Move non-root filesystems on smallest spindle into Vinum
  if (defined($rsp) && $dev =~ /^\/dev\/$rsp/ && $dev =~ /[d-h]$/) {
    $pass = $nxtpas++;
    print FSOUT "/dev/vinum$mnt\t\t$mnt\t\t$fstyp\t$opt\t\t$dump\t$pass\n";
    $vols{$dev}->{mnt} = substr($mnt, 1);
    next;
  }
  print FSOUT $_;
}
close(FSOUT);
die("Found more spindles than we have abstract names\n") if $#spndl > $#drv;
die("Didn't find a root partition!\n") if !defined($rsp);
die("Didn't find a /rootback partition!\n") if !defined($rbsp);

# Table of server's Degraded Modes
# One row per mode with hash keys
#   fn FileName
#   xpr eXPRession needed to convert fstab lines for this mode
#   cm1 CoMment 1 describing this mode
#   cm2 CoMment 2 describing this mode
#   FH  FileHandle (dynamically initialized below)
my @DM = (
  { cm1 => "When we only have $rsp, comment out lines using $rbsp",
    fn  => "/etc/fstab_only_have_$rsp",
    xpr => "s:^/dev/$rbsp:#\$&:",
  },
  { cm1 => "When we only have $rbsp, comment out lines using $rsp and",
    cm2 => "rootback becomes root",
    fn  => "/etc/fstab_only_have_$rbsp",
    xpr => "s:^/dev/$rsp:#\$&: || s:/rootback:/\t:",
  },
  { cm1 => "When only $rsp root is bad, /rootback becomes root and",
    cm2 => "root becomes /rootbad",
    fn  => "/etc/fstab_${rsp}_root_bad",
    xpr => "s:\t/\t:\t/rootbad: || s:/rootback:/\t:",
  },
);

# Initialize output FileHandles and write comments
foreach my $dm (@DM) {
  my $fh = new FileHandle;
  $fh->open(">$dm->{fn}") || die("Can't write $dm->{fn}: $!\n");
  print $fh "# $dm->{cm1}\n" if $dm->{cm1};
  print $fh "# $dm->{cm2}\n" if $dm->{cm2};
  $dm->{FH} = $fh;
}

# Parse the Vinum version of fstab written above and write versions needed
# for server's degraded modes.
open(FSOUT, "$fsout") || die("Couldn't open $fsout: $!\n");
while (<FSOUT>) {
  my $line = $_;
  foreach my $dm (@DM) {
    $_ = $line;
    eval $dm->{xpr};
    print {$dm->{FH}} $_;
  }
}

# Parse partition table for each spindle and write versions needed for Vinum
my $rootsiz; # ROOT partition SIZe
my $swapsiz; # SWAP partition SIZe
my $rspminoff; # Root SPindle MINimum OFFset of non-root, non-swap, non-c parts
my $rspsiz; # Root SPindle SIZe
my $rbspsiz; # RootBack SPindle SIZe
foreach my $i (0..$#spndl) {
  my $dlin = "disklabel $spndl[$i] |";
#  my $dlin = "simu/disklabel.$spndl[$i]";
  open(DLIN, "$dlin") || die("Couldn’t open $dlin: $!\n");

  my $dlout = "disklabel.$spndl[$i]";
  open(DLOUT, ">$dlout") || die("Couldn’t open $dlout for writing: $!\n");

  my $dlb4 = "$dlout.b4vinum";
  open(DLB4, ">$dlb4") || die("Couldn’t open $dlb4 for writing: $!\n");

  my $minoff;  # MINimum OFFset of non-root, non-swap, non-c partitions
  my $totsiz = 0; # TOTal SIZe of all non-root, non-swap, non-c partitions
  my $swapspndl = 0; # True if SWAP partition on this SPiNDLe
  while (<DLIN>) {
    print DLB4 $_;
    my ($part, $siz, $off, $fstyp, $fsiz, $bsiz, $bps) = split;

    if ($part && $part eq ’a:’ && $spndl[$i] eq $rsp) {
 $rootsiz = $siz;
    }
    if ($part && $part eq ’e:’ && $spndl[$i] eq $rbsp) {
      if ($rootsiz != $siz) {
 die("Rootback size ($siz) != root size ($rootsiz)\n");
      }
    }
    if ($part && $part eq ’c:’) {
      $rspsiz  = $siz if $spndl[$i] eq $rsp;
      $rbspsiz = $siz if $spndl[$i] eq $rbsp;
    }
    # Make swap partition $cfgsiz sectors smaller
    if ($part && $part eq ’b:’) {
      if ($spndl[$i] eq $rsp) {
 $swapsiz = $siz;
      } else {
 if ($swapsiz != $siz) {
   die("Swap partition sizes unequal across spindles\n");
 }
      }
      printf DLOUT "%4s%9d%9d%10s\n", $part, $siz-$cfgsiz, $off, $fstyp;
      $swapspndl = 1;
      next;
    }
    # Move rootback spindle e partitions to a
    if ($part && $part eq ’e:’ && $spndl[$i] eq $rbsp) {
      printf DLOUT "%4s%9d%9d%10s%9d%6d%6d\n", ’a:’, $siz, $off, $fstyp,
 $fsiz, $bsiz, $bps;
      next;
    }
    # Delete non-root, non-swap, non-c partitions but note their minimum
    # offset and total size that’re needed below.
    if ($part && $part =~ /^[d-h]:$/) {
      $minoff = $off unless $minoff;
      $minoff = $off if $off < $minoff;
      $totsiz += $siz;
      if ($spndl[$i] eq $rsp) { # If doing spindle containing root
 my $dev = "/dev/$spndl[$i]" . substr($part, 0, 1);
 $vols{$dev}->{siz} = $siz;
 $vols{$dev}->{off} = $off;
 $rspminoff = $minoff;
      }
      next;
    }
    print DLOUT $_;
  }
  if ($swapspndl) { # If there was a swap partition on this spindle
    # Make a Vinum partition the size of all non-root, non-swap,
    # non-c partitions + the size of Vinum's on-disk configuration.
    # Set its offset so that the start of the first subdisk it contains
    # coincides with the first filesystem we're embedding in Vinum.
    printf DLOUT "%4s%9d%9d%10s\n", "$vip:", $totsiz+$cfgsiz, $minoff-$cfgsiz,
      'vinum';
  } else {
    # No need to mess with size size and offset if there was no swap
    printf DLOUT "%4s%9d%9d%10s\n", "$vip:", $totsiz, $minoff,
      'vinum';
  }
}
die("Swap partition not found\n") unless $swapsiz;
die("Swap partition not larger than $cfgsiz blocks\n") unless $swapsiz>$cfgsiz;
die("Rootback spindle size not >= root spindle size\n") unless $rbspsiz>=$rspsiz;

# Generate input to vinum create command needed for each spindle.
foreach my $i (0..$#spndl) {
  my $cfn = "create.$drv[$i]"; # Create File Name
  open(CF, ">$cfn") || die("Can't open $cfn for writing: $!\n");
  print CF "drive $drv[$i] device /dev/$spndl[$i]$vip\n";
  next unless $spndl[$i] eq $rsp || $spndl[$i] eq $rbsp;
  foreach my $dev (keys(%vols)) {
    my $mnt = $vols{$dev}->{mnt};
    my $siz = $vols{$dev}->{siz};
    my $off = $vols{$dev}->{off}-$rspminoff+$cfgsiz;
    print CF "volume $mnt\n" if $spndl[$i] eq $rsp;
    print CF <<EOF;
  plex name $mnt.p$i org concat volume $mnt
    sd name $mnt.p$i.s0 drive $drv[$i] plex $mnt.p$i len ${siz}s driveoffset ${off}s
EOF
  }
}
```

B. Manual Vinum Bootstrapping

The `bootvinum` Perl script in Appendix A makes life easier, but it may be necessary to manually perform some or all of the steps that it automates. This appendix describes how you would manually mimic the script.

1. Make a copy of `/etc/fstab` to be customized.

   ```
   # cp /etc/fstab /etc/fstab.vinum
   ```

2. Edit `/etc/fstab.vinum`.

   a. Change the `device` column of non-root partitions on the root spindle to `/dev/vinum/mnt`.

   b. Change the `pass` column of non-root partitions on the root spindle to **`2`**, **`3`**, etc.

   c. Delete any lines with mountpoint matching `/NOFUTURE*`.

   d. Change the `device` column of `/rootback` from `e` to `a`.

   e. Change the `pass` column of `/rootback` to **`1`**.

3. Prepare disklabels for editing:

   ```
   # cd /bootvinum
   # disklabel ad0s1 > disklabel.ad0s1
   # cp disklabel.ad0s1 disklabel.ad0s1.b4vinum
   # disklabel ad2s1 > disklabel.ad2s1
   # cp disklabel.ad2s1 disklabel.ad2s1.b4vinum
   ```

4. Edit `/etc/disklabel.ad?s1`.

   a. On the root spindle:

      i. Decrease the `size` of the `b` partition by 265 blocks.

      ii. Note the `size` and `offset` of the `a` and `b` partitions.

      iii. Note the smallest `offset` for partitions `d`-`h`.

      iv. Note the `size` and `offset` for all non-root, non-swap partitions (`/home` was probably on `e` and `/usr` was probably on `f`).

      v. Delete partitions `d`-`h`.

      vi. Create a new `h` partition with `offset` 265 blocks less than the smallest `offset` for partitions `d`-`h` noted above. Set its `size` to the `size` of the `c` partition less the smallest `offset` for partitions `d`-`h` noted above + 265 blocks.

      > **Note: Vinum** can use any partition other than `c`. It is not strictly necessary to use `h` for all your **Vinum** partitions, but it is good practice to be consistent across all spindles.

      vii. Set the `fstype` of this new partition to **`vinum`**.

   b. On the rootback spindle:

      i. Move the `e` partition to `a`.

      ii. Verify that the `size` of the `a` and `b` partitions matches the root spindle.

iii. Note the smallest `offset` for partitions `d-h`.

iv. Delete partitions `d-h`.

v. Create a new `h` partition with `offset` 265 blocks less than the smallest `offset` noted above for partitions `d-h`. Set its `size` to the `size` of the `c` partition less the smallest `offset` for partitions `d-h` noted above + 265 blocks.

vi. Set the `fstype` of this new partition to **`vinum`**.

5. Create a file named `create.YouCrazy` that contains:

```
drive YouCrazy device /dev/ad0s1h
volume home
  plex name home.p0 org concat volume home
    sd name home.p0.s0 drive YouCrazy plex home.p0 len $hl driveoffset $ho
volume usr
  plex name usr.p0 org concat volume usr
    sd name usr.p0.s0 drive YouCrazy plex usr.p0 len $ul driveoffset $uo
```

   Where:

   - `$hl` is the length noted above for `/home`.
   - `$ho` is the offset noted above for `/home` less the smallest offset noted above + 265 blocks.
   - `$ul` is the length noted above for `/usr`.
   - `$uo` is the offset noted above for `/usr` less the smallest offset noted above + 265 blocks.

6. Create a file named `create.UpWindow` containing:

```
drive UpWindow device /dev/ad2s1h
  plex name home.p1 org concat volume home
    sd name home.p1.s0 drive UpWindow plex home.p1 len $hl driveoffset $ho
  plex name usr.p1 org concat volume usr
    sd name usr.p1.s0 drive UpWindow plex usr.p1 len $ul driveoffset $uo
```

   Where `$hl`, `$ho`, `$ul`, and `$uo` are set as above.

C. Acknowledgements

I would like to thank Greg Lehey for writing **Vinum** and for providing very helpful comments on early drafts. Several others made helpful suggestions after reviewing later drafts including Dag-Erling Smørgrav, Michael Splendoria, Chern Lee, Stefan Aeschbacher, Fleming Froekjaer, Bernd Walter, Aleksey Baranov, and Doug Swarin.

Notes

1. This assumes that you have not removed the line

```
options ATA_STATIC_ID
```

   from your kernel configuration.