Softraid: OpenBSD's virtual HBA, with benefits

Marco Peereboom
OpenBSD

Abstract

Prior to softraid(4)[1] OpenBSD[2] had two RAID implementations. RAIDframe[3] is a full blown RAID implementation written by Carnegie Mellon University and ccd(4)[4] is a minimal RAID-like I/O transformation engine developed at the University of Utah.

RAIDframe has not been synchronized with the upstream codebase for years and has never been enabled by default. In the OpenBSD project that translates to unsupported code. The only work done on RAIDframe was an occasional change to ensure that it still compiles. The RAIDframe code base is large, complex and intended for research purposes.

The ccd(4) stack is enabled by default but has also not seen any significant change or update in years. The implementation is bare-bones and its use is complex and error prone.

Both of these implementations interact at the block I/O layer. This results in the need for implementation specific tools rather than using system defaults such as bioctl(8)[5], the tool normally used for RAID management.

Due to the above mentioned reasons a new generic implementation of I/O transformation was devised. It had to fit the OpenBSD philosophy of being simple and powerful, which led to the development of a virtual Host Bus Adapter (HBA) with benefits. These benefits are the capability to transform any standard SCSI I/O operation into one or more complex operations, enabling functionality such as RAID, iSCSI and encryption.

1 Introduction

The softraid framework interacts in the I/O stack as a standard SCSI HBA and therefore has to provide exactly the same entry points as a hardware driver would. It has a top-half and bottom-half, an IOCTL path and the required MINPHYS function to determine the maximum size I/O it can handle. Additionally it adds sensors just like a hardware HBA would in order to indicate the health of a virtual disk.

An I/O transformation driver is known as a discipline. A discipline is essentially a driver that plugs into the softraid framework and it is responsible for all I/O transformations. The transformations are arbitrarily complex and span a wide range of uses. For example, a discipline could mirror an I/O, stripe an I/O or encrypt an I/O.

All I/Os at some point end up stored on chunks. These chunks are disk partitions with a RAID type. The RAID partition type is reused from RAIDframe, however the format used is incompatible. All chunks contain metadata at the start of the partition to indicate that it is a softraid chunk. This metadata also contains all of the information that a discipline needs to completely describe a virtual disk. The act of bringing up a virtual disk is known as "assembling a volume". An exception to this are target devices such as the AOE target. Such disciplines do not appear as a device within the operating system, however they do use all of the metadata functionality provided by softraid.

Virtual disks, or volumes, consist of any number of chunks depending on the discipline. For example, a RAID 0 discipline requires at least 2 chunks. A virtual disk behaves just like any other SCSI disk attached to the system. All of the standard utilities and tools can operate on it as if it were a physical device.

When a SCSI I/O arrives at the softraid framework a corresponding Work Unit (WU) is created and sent to the appropriate discipline. A WU can consist of any number of operations that need to complete before a WU can complete the original SCSI I/O. An operation is contained in a Command Control Block (CCB) and is usually, but not limited to, a physical I/O.

In order to prevent data corruption when I/Os physically overlap the same LBA range softraid implements a feature known as "colliders". When an I/O physically overlaps an LBA range of any outstanding I/O it will be deferred until the underlying I/O completes. At completion time, in the interrupt handler, the collider I/O will be resubmitted. Each I/O can have at most one collider however, the colliding I/O can also have a collider forming a chain of ordered I/Os. This feature is also used during RAIL volume recovery scenarios where I/Os need to complete in order to form an "atomic" I/O.

Since softraid pretends to be a HBA it has some limitations and requirements. A HBA can be called in both process and interrupt context. Inherently softraid cannot perform any operation that requires process context, such as tsleep(9)[6]. In order to properly detect a failed chunk softraid relies on the underlying hardware and layers to correctly propagate errors up the stack.

2 Disciplines

A discipline contains all variables and function pointers for the softraid framework to handle said discipline. The entry pointers are divided into two logical areas namely those that are SCSI and discipline specific. Not all function pointers are required and some have generic implementations. In some specific cases the discipline reuses other discipline functions (e.g. the crypto discipline reuses the RAID 1 sd_set_vol_state).

All sd_ functions deal with the discipline specific parts of the generic functions. This is a theme that is repeated throughout all the softraid code.

The required discipline functions are:

- sd_create
  - This function is the discipline specific part of a creation call. This function validates that the the discipline meets the minimal requirements and sets several internal variables such as the discipline name, maximum number of CCBs per WUs etc.
- sd_assemble
  - This function is the discipline specific part of an assembly call. For most disciplines it comes down to only filling out the sd_max_wu variable however disciplines such as crypto perform additional work to properly assemble the discipline.
- sd_alloc_resources
  - This function allocates all resources the discipline needs to operate. It generally is advised to not allocate resources in the I/O path due to latency.
- sd_free_resources
  - This function frees all resources previously allocated by sd_alloc_resources. This function is called prior to shutdown of the discipline.
- sd_ioctl_handler
  - The IOCTL handler is used by user-space tools such as bioctl to display the status of softraid, its volumes and chunks.
- sd_start_discipline
  - This function is only used for target devices that are not virtual disks. Disciplines such as the AOE target use this function to bring up the discipline and make it ready to serve requests.
- sd_set_chunk_state
  - This function is used to modify the state of a chunk into a valid new state. Chunk state is what drives the volume state and this function changes the state of a single chunk and subsequently calls sd_set_vol_state in order to alter the volume state.
- sd_set_vol_state
  - This functions calculates the new volume state based on the current state of all chunks. This function should only be called by sd_set_chunk_state.

Optional:

- sd_openings
  - This function is optional and is called only if set. It returns a sensible value for openings that are used to initialize the SCSI-midlayer when the volume is brought online. The disciplines that use this function are ones that tax the system unfairly due to internal colliding I/Os and therefore cannot use the sd_max_wu heuristic. Disciplines that make use of this facility are RAID P (level 4 & 5) and RAID 6.

The SCSI functions are mostly generic however the read and write functions are always discipline dependent. The functions correspond exactly to the SCSI specification as define by T10[7].

The SCSI functions that must be implemented are:

- sd_scsi_inquiry. This function returns the INQUIRY(12) data and is generic providing sd_create filled in all required fields.
- sd_scsi_read_cap. This function returns the READ_CAPACITY(25) and READ_CAPACITY16(9E) data and are generic providing sd_create filled in all required fields.
- sd_scsi_tur. This generic function returns success if the volume has been brought up successfully and it will return failure if the volume subsequently fails.
- sd_scsi_req_sense. This function is generic and simply returns the last sense data that was generated, before zeroing the sense buffer.
- sd_scsi_start_stop. This function is generic and will bring a volume online or offline depending on the parameter provided.
- sd_scsi_sync. This function is generic and drains all outstanding I/O as per the SYNCHRONIZE_CACHE(35) command.
- sd_scsi_rw. This function is discipline specific and performs the read or write for the following SCSI opcodes:

- READ6(08)
- READ(28)
- READ16(88)
- WRITE6(0a)
- WRITE(2a)
- WRITE16(8a)

The following sections will briefly detail all currently implemented disciplines, however these sections will not question or debate RAID as a technology, nor will they discuss its relative merit. RAID concepts are generally well understood and are outside the scope of this paper.

2.1 RAID 0

The RAID 0 discipline provides striping across disks without providing any form of redundancy. The strip size is fixed at 64KB which is optimal for performance with the OpenBSD block I/O layer.

At less than 500 lines of code, including the license and comments, this discipline makes for an easy read. It is considered to be the reference implementation due to its small size and the lack of complexity.

2.2 RAID 1

The RAID 1 discipline provides mirroring across N chunks, whereas most traditional RAID 1 implementations only support two chunks in a mirror. The softraid implementation reads single I/Os on a round-robin basis from all active chunks and writes I/Os to all active chunks. This presents the user with some interesting choices at volume creation time. A mostly read-only system can favor a larger number of chunks at the cost of disk space, resulting in increased read performance.

Also unlike traditional RAID 1 implementations the softraid version does not mirror the chunks upon creation. The reason for this is that all blocks are always written to prior to being read. If this is not the case then there is a bug in the layer above softraid, in which case softraid will not behave any differently to a traditional disk.

The RAID 1 discipline can automatically recover from a failed chunk if a hot-spare chunk of bigger or equal size is available at the time of the failure. This process can also be manually initiated by the user. A rebuild will result in every block being read from the active chunks and being written to the rebuild chunk. If a rebuild is aborted due to a reboot or crash it will resume upon the subsequent boot.

2.3 RAID P

The RAID P (Parity) discipline provides RAID level 4 and 5. The only difference being that RAID 4 uses a fixed parity chunk instead of distributing the parity across all chunks.

The RAID P discipline is currently considered experimental because it misses scrub and rebuild functionality. Unlike with RAID 1 discipline RAID P require that all parity is zeroed before use.

2.4 RAID 6

Unlike RAID 5, the RAID 6 discipline provides the capability to have two chunks fail while maintaining integrity. Additionally, in the case of silent data corruption it raises the chances of successful recovery since it potentially can recreate the data from multiple sources.

The RAID 6 discipline is currently considered experimental because it misses scrub and rebuild functionality. Unlike with RAID 1 discipline, RAID 6 requires that all parity is initialized before use.

2.5 Crypto

This discipline is the most complex. It requires a complex bring-up procedure that may result in multiple calls between userland and the kernel. The reason for this is that there are pieces of code that do not belong in the kernel and in the case of the crypto discipline this is the encryption of the keys. This procedure does facilitate additional functionality such as changing the password to unmask the keys.

The crypto discipline defaults to the AES XTS 256 encryption algorithm. This algorithm is a modified AES implementation that has a built-in tweak that was especially developed for disk encryption. The OpenBSD implementation uses one key per 0.5TB instead of the recommended 1TB. Currently the implementation has 32 keys which nets a maximum 16TB volume. These numbers are design decisions and can be easily modified in the future.

The encryption keys are automatically generated using a strong random number generator (arc4random_buf(9)[8]) and are encrypted based on a user provided password. If the user opts to use a key disk instead then the keys are written either unencrypted to the specified chunk. This enables auto-assembly of a crypto volume during boot. The use-case is writing the keys on an easily removable device, such as a USB flash disk, so that an encrypted volume can only be brought up with this removable disk inserted.

There are still a few things missing in the crypto discipline. Currently there is only one password that can decrypt the keys, whereas it is desirable to have multiple passwords. Another thing that is missing is some sort of utility that can facilitate reading and writing the decrypted keys so that they can be used in the future for a variety of reasons such as disaster recovery etc. Currently there is no way to select the number of bits the encryption algorithm uses even though the code supports two of them.

2.6 Missing or incomplete features

Softraid is still under development and there are some major features missing. On the discipline front there are still some ideas floating around for the following types:

- Concat. Simply concatenate chunks together to make a larger disk.
- Stacking. This is a complex problem that needs to be thought through. The intent is to be able to "stack" RAID volumes on top of each other to obtain hybrid types such as RAID 10 or RAID 1CRYPTO.
- AOE has not been actively maintained and needs to be revisited and brought into the enabled state. The target code might require additional framework features.
- iSCSI. There is an active iSCSI development using a different kernel interface known as vscsi(4)[9] however this code will run mostly in user-space and does not seem likely to ever support a target.
- FCoE[10]. This is currently the buzz of the industry and having target and initiator code would enable OpenBSD to play with some really large iron.
- Multipath. This one might not be written using softraid due to recent addition of the mpath(4)[11] driver.

Aside from new disciplines, the framework itself still requires some additional features. Work has begun on booting and rooting from softraid disciplines. A working prototype exists for the OpenBSD/sparc64 platform. Each architecture has its own challenges and this is still under investigation. The intent is that the softraid metadata has the boot loader code as a payload and the user-space utilities that are used to create bootable disks talk to softraid and provide the hints required to perform all boot loader related functions.

A particular problem that arises from using softraid is the potential for physical disks to move around. To work around this problem the code to mount filesystems using a disklabel ID has been written. This code will be integrated into the tree after the current release (4.7) has been tagged.

The metadata code has hooks to be able to read foreign metadata from different vendors, however there is currently no additional support for metadata formats other than softraid.

3 Metadata

Softraid uses on disk metadata to record state and persistent information. All chunks carry near identical metadata payloads, however each chunk designates which piece they represent.

The metadata is written at the beginning of the disk at a constant offset from the beginning of the partition. It has a fixed block count for size, which limits the amount of information it can carry. This limitation is currently in the order of 200 chunks.

All chunks have at least 2 pieces of metadata.

1. sr_metadata. The sr_metadata chunk is divided into 2 areas; a variant and an invariant area. The invariant piece contains persistent information that will not change, such as the volume ID, metadata version, number of chunks, number of optional metadata areas, discipline type, etc. The variant piece contains the invariant checksum, on disk metadata version, etc.
2. sr_meta_chunk. All chunks are written in sorted order following the sr_metadata piece. The sr_meta_chunk also has a variant and an invariant area. The invariant area contains persistent information such as the volume ID, chunk ID, volume UUID, etc. The variant piece contains the invariant checksum and current chunk status.

The area directly after the last chunk metadata is where the optional metadata is stored. The number of optional metadata members are stored in the sr_metadata area. These optional areas contain things such as crypto keys or boot loader code, etc.

Each chunk maintains an "on-disk" version number. This number is used to resolve bring-up and failure conflicts. The metadata code will only use the latest on-disk version when bringing up a volume. Every time a change is made to any metadata component a new version is written to all chunks. At a minimum, a power-up to power-down of an auto assembled volume will have increased the on-disk version of the firmware by two.

The metadata code is written with foreign data formats in mind. It provides hooks in strategic locations for so called translation functions. For example, when foreign metadata is read from a disk it is translated into a softraid legible format and conversely when written.

4 Related Work

There are quite a few distinct RAID implementations out in the open source arena, however it is believed that softraid is the first one to implement it at the SCSI level by presenting itself as an HBA. The name softraid is actually a poorly chosen name, however it stuck.

Some other implementations are device-mapper[12] on Linux and GEOM[13] on FreeBSD.

5 Acknowledgments

I would like to acknowledge several people in the OpenBSD community who have worked on parts of softraid:

- Joel Sing. Joel has worked tirelessly on softraid and he implemented several major features. From code re-factoring to bug hunting to brand new features, he has done it all. He must have touched virtually all code in the softraid stack.
- Ted Unangst. Ted, is as usual, responsible for getting other people to do the work after he shows a working prototype. He wrote the crypto and AOE prototypes.
- Hans-Joerg Hoexer. Hans-Joerg wrote the entire implementation of the crypto discipline during a hackathon.
- Damien Miller. Before Damien had his second child he found some time during a hackathon to help me out with the metadata format. He also helped Hans-Joerg with the crypto discipline.
- Jordan Hargrave. One day out of nowhere Jordan handed me a more or less working versions of the RAID P and RAID 6 disciplines. Since then he has made many improvements and continues to do so. He told me “It is wondrous what one can do with math” after his latest performance improvements.

Theo de Raadt for his supportive yelling and seemingly unending energy and devotion to keep OpenBSD running, allowing me to play in the most intellectually stimulating open source community in the world. His insight into operating system design is simply unparalleled and he has lent me some of it to make several key architectural decisions in the softraid stack.

I’d like to acknowledge Clarissa, my lovely wife, who puts up with me and my open source addiction.

Finally, my lovely daughter Holland who always manages to brighten my day and JT who watches her while I am coding.

References

[1] softraid http://www.openbsd.org/cgi-bin/man.cgi?query=softraid&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[2] OpenBSD http://www.openbsd.org/

[3] RAIDframe http://www.pdl.cmu.edu/RAIDframe/

[4] ccd http://www.openbsd.org/cgi-bin/man.cgi?query=ccd&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[5] bioctl http://www.openbsd.org/cgi-bin/man.cgi?query=bioctl&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[6] tsleep http://www.openbsd.org/cgi-bin/man.cgi?query=tsleep&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[7] T10 http://www.t10.org/

[8] arc4random_buf http://www.openbsd.org/cgi-bin/man.cgi?query=arc4random_buf&apropos=0&sektion=9&manpath=OpenBSD+Current&arch=i386&format=html

[9] http://www.openbsd.org/cgi-bin/man.cgi?query=vscsi&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[10]FCoE http://en.wikipedia.org/wiki/Fibre_Channel_over_Ethernet

[11]mpath(4) http://www.openbsd.org/cgi-bin/man.cgi?query=mpath&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

[12]Device-mapper http://sources.redhat.com/dm/

[13]GEOM http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/geom.html