FreeBSD® JOURNAL

January/February 2023

ZFS's Atomic I/O and PostgreSQL

Virtual Lab—BSD Programming Workshop

An Introduction to ZFS

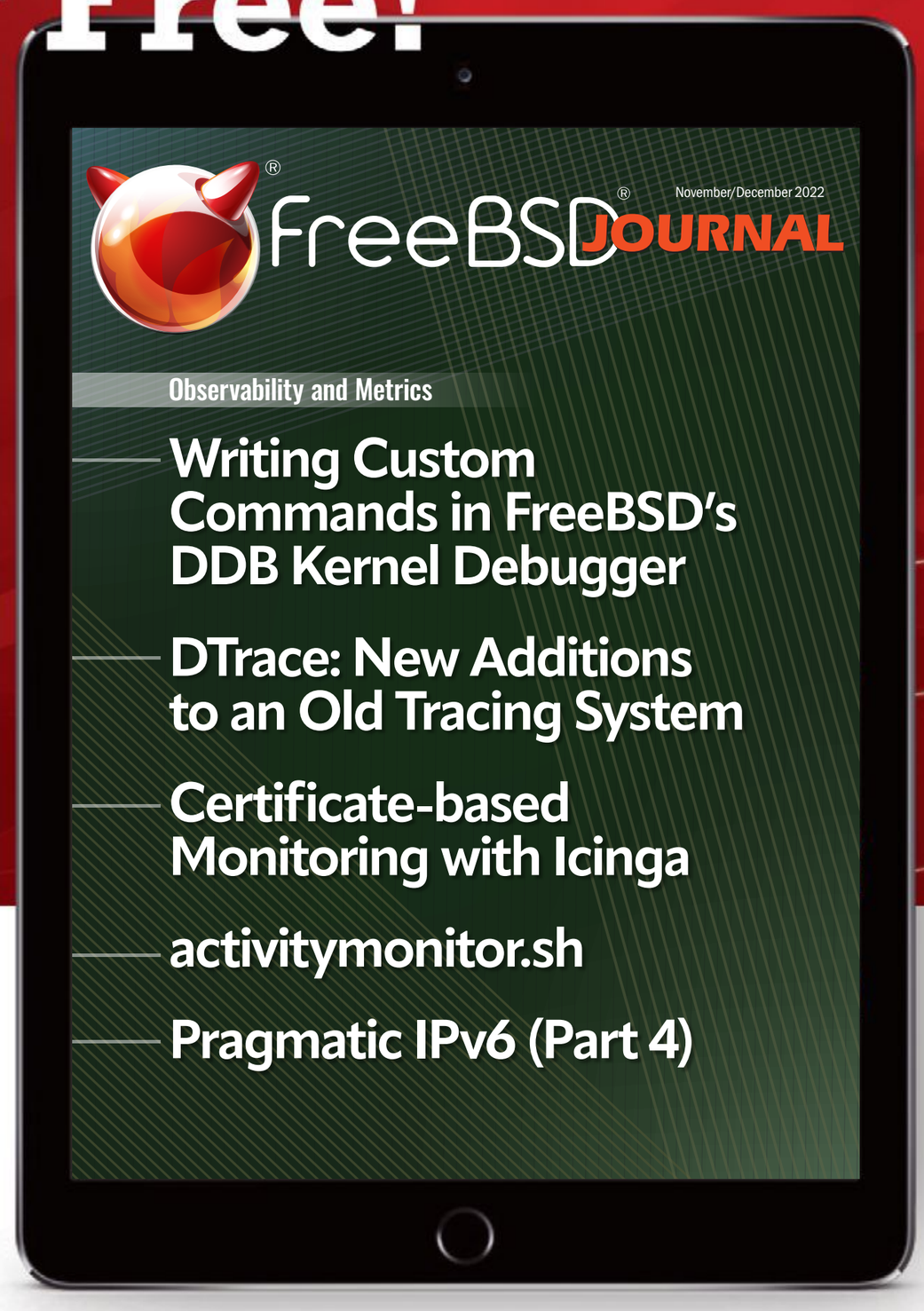
FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



2023 Editorial Calendar

- Building a FreeBSD Web Server (January-February)
- Embedded (March-April)
- FreeBSD at 30 (May-June)
- Containers and Cloud (Virtualization) (July-August)
- FreeBSD 14 (September-October)
- To be decided (November-December)

Find out more at: freebsdfoundation/journal

FreeBSD® JOURNAL

S&W PUBLISHING LLC
PO BOX 3757 CHAPEL HILL, NC 27515-3757

Editor-at-Large • James Maurer
maurer.jim@gmail.com

Design & Production • Reuter & Associates

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Foundation

Welcome to the January/February issue. This edition features articles on several topics to aid in using FreeBSD to deploy web applications. Drew Gurkowski provides an introduction to using ZFS to manage storage. Roller Angel describes the creation of a virtual test lab built on top of jails. Finally, Thomas Munro dives into interactions between PostgreSQL and ZFS.

On a different note, Anne Dickison sat down with Ed Maste from the FreeBSD Foundation to talk about support for desktop use in FreeBSD.

In a previous welcome letter, I was eagerly anticipating a return to an in-person conference—EuroBSDCon 2022, that was held in Vienna, Austria. As a trip report by Kyle Evans in the November/December 2022 issue indicated, the conference was a tremendous success. In the new year, many of our beloved conferences are returning to in-person formats. FOSDEM was recently held in person, and AsiaBSDCon, BSDCan, and EuroBSDCon are all returning this year. At BSDCan in Ottawa, we will celebrate FreeBSD's 30th birthday! Editorial board members and *FreeBSD Journal* authors will be at these conferences and they're eager to chat with readers or answer questions.

As always, we love to hear from readers whether in person or via email. If you have feedback or suggestions for topics for a future article, or are interested in writing an article, please email us at maurer.jim@gmail.com.

John Baldwin
Chair of the *FreeBSD Journal* Editorial Board

FreeBSD® JOURNAL
January/February 2023

5 ZFS's Atomic I/O and PostgreSQL
By Thomas Munro

9 Virtual Lab— BSD Programming Workshop
By Roller Angel

18 An Introduction to ZFS
By Drew Gurkowski

3 Foundation Letter
By John Baldwin

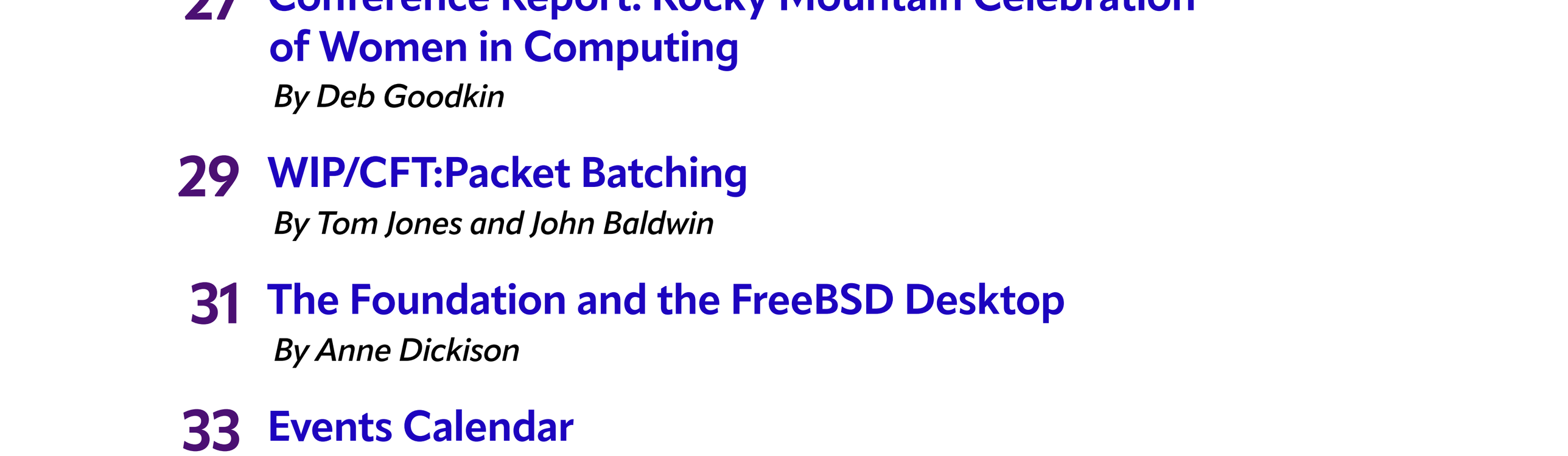
24 We Get Letters
By Michael W Lucas

27 Conference Report: Rocky Mountain Celebration of Women in Computing
By Deb Goodkin

29 WIP/CFT:Packet Batching
By Tom Jones and John Baldwin

31 The Foundation and the FreeBSD Desktop
By Anne Dickison

33 Events Calendar
By Anne Dickison



ZFS's Atomic I/O and PostgreSQL

BY THOMAS MUNRO

PostgreSQL is a relational database management system implementing the SQL standard, with a BSD-like license. Its pre-SQL ancestor POSTGRES began at Berkeley University in the mid 1980s. It's popular on FreeBSD, where it is usually deployed on ZFS storage.

Many articles about PostgreSQL on ZFS recommend changing ZFS's `recordsize` setting and PostgreSQL's `full_page_writes` setting. The real impact of the latter setting on performance and crash-safety is not often explained, perhaps because it's not generally safe to adjust it on most popular file systems. In this article I summarize the logic and trade-offs behind this mysterious mechanism—after a brief detour to talk about block sizes.

Blocks

Nearly all of PostgreSQL's disk I/O is aligned on 8KB blocks, or pages. It is possible to recompile it to use a different size, but that is rarely done. This size may originally have been chosen to match UFS's historical default block size (though note that FreeBSD's UFS now defaults to 32KB). ZFS uses the term record size, and defaults to 128KB. Unlike other file systems, ZFS allows the record size to be changed easily at any time, and to be configured separately for each dataset.

POSTGRES began at Berkeley University in the mid 1980s.

If the data will be accessed randomly, then in theory the size should ideally match PostgreSQL's 8KB blocks. Otherwise, random I/O could suffer from two effects:

- I/O amplification, because every read or write of an 8KB block also transfers extra neighboring data
- read-before-write when storage blocks are not currently in the OS's cache and an 8KB block must be written, so the neighboring data must be read first

If the data will be accessed mostly sequentially, or rarely, and especially if the benefits of ZFS compression using larger records outweigh concerns about I/O bandwidth and latency, then it can be a good idea.

Some sources make a blanket recommendation of 16KB, 32KB or 128KB record size, as a sweet spot for better compression without too much write amplification or latency. My aim

here isn't to make such recommendations–I doubt there is one answer–but rather to explain what's going on.

Some applications have a mix of requirements for different kinds of data. Tablespaces can be used to store different tables in different ZFS datasets with different record size, compression or physical media. It's also possible for a table to be partitioned, for example with older data in one tablespace and current active data in another.

```
CREATE TABLESPACE compressed_tablespace
LOCATION '/tank/pgdata/compressed_tablespace';

ALTER TABLE t
SET TABLESPACE compressed_tablespace;
```

One problem reported with small ZFS record sizes is fragmentation. A table that receives frequent random updates might finish up with blocks scattered all over the place, and we'd prefer them to be physically clustered for good sequential read performance. A simple way to ask PostgreSQL to rewrite the files that hold a table and its indexes in order to defragment them at the ZFS level would be to issue **`VACUUM FULL table_name`** or **`CLUSTER table_name`**, if you are prepared to lock queries out of the table for the duration of the rewrite. Rewriting a table also allows a new record size to take effect, if it has been changed at the dataset level.

Torn Writes

The PostgreSQL setting **`full_page_writes`** defaults to on, and ZFS users often turn it off. The performance of write-intensive workloads then becomes faster and more consistent. For example, in a simple pgbench test on a low end cloud VM I measured a 32% increase in transactions per second by turning it off.

So what does it really do? That requires a surprising amount of background explanation.

The short version is that PostgreSQL uses *physiological logging* for crash safety, and that means that writes to individual database pages must be *atomic on power failure*, or it may not be able to recover after a crash. Unless you promise that your storage stack has that property, then PostgreSQL has to do some extra work to protect your data.

Atomicity on power failure is the property that if a physical write was in progress when power was lost, later we can expect to read back either the old version or the new version of a block, for some given block size, but not a partially modified or torn version. This is not to be confused with atomicity of concurrent reads and writes (see below). Physiological logging, short for *physical-to-the-page*, *logical-within-the-page*, is a term from textbook classifications of logging strategies, and it means that log records identify a block to be changed by file and block number, but then describe the change to make within that page in a notation that requires us to read in the existing page to understand how to modify it "logically", rather than just updating bits at a physical address.

After a crash, the recovery algorithm can cope with the "old" page contents or the "new" page contents, applying any logged changes required to bring it up to date. If it encounters a non-atomic mash-up of old and new data, then logical changes to the page cannot be replayed, and recovery fails! A superficial problem is that if **`data_checksums`** is enabled, then PostgreSQL's page-level checksum check will fail even to read the page in. If checksums are disabled, we'll get further, but a logical change such as "insert tuple (42,Fred) in slot 3" can't

be replayed reliably. In order to apply the change in this example we need to understand a table of slots using pre-existing meta-data on the page, but it's potentially corrupted.

Physiological logging is a very widely used technique in the database industry, and different RDBMSs have found different solutions to the problem of torn pages. Since open source systems have been developed and used on a wide variety of low end systems often without various forms of hardware protection against power loss, failures were common and software solutions had to be developed.

PostgreSQL's current solution is to switch to page-level physical-only logging or *full page writes*, where the whole data page is dumped into the log, for the first modification to each data page after each checkpoint. Checkpointing is a periodic background activity, and in an ideal world would have minimal effects on foreground transaction performance. However, due to the first-touch rule, once a checkpoint starts, write-heavy workloads might suddenly start generating a lot more log data, as small updates suddenly require many 8KB pages to be logged. This effect typically decays gradually because subsequent modifications to each page go back to being physiological, until the next checkpoint, sometimes resulting in a sawtooth pattern in I/O bandwidth and transaction latency.

Another popular open source database has a different solution that also involves writing all data out twice with a synchronization barrier between, since both copies can't be torn.

ZFS needs none of that! It has record-level atomicity by virtue of its own copy-on-write design. It's not possible to see a mixture of the old and new contents of a ZFS record, because it doesn't physically overwrite them, and its system of TXGs and the ZIL makes writes transactional. Therefore, it is safe to set **`full_page_writes=off`** as long as **`recordsize`** is at least 8KB.

It's not possible to see a mixture of the old and new contents of a ZFS record.

Note that ZFS itself also physically writes data twice in some scenarios. A common recommendation is to consider setting **`logbias=throughput`** for the dataset holding the main data files (but perhaps not the one holding PostgreSQL's log directory **`pg_wal`**—a topic not explored in this article). That option tries to write blocks directly into their final location instead of logging them first in the ZIL. If you use the ZFS default **`logbias=latency`** and the PostgreSQL default **`full_page_writes=on`**, data may in fact be written out four times in total as both PostgreSQL and ZFS perform extra work to create record-level atomicity, while both of those changes bring it down to one copy.

Unfortunately there are two special scenarios where **`full_page_writes=on`** is still needed for correct behavior: while running **`pg_basebackup`** and **`pg_rewind`**. Those tools are used for backups, or to create or re-synchronize streaming replicas from another server; in the case of **`pg_basebackup`**, full page writes will be silently enabled while running the command, while in the case of **`pg_rewind`**, the command will refuse to run if it is not manually enabled (an annoying inconsistency in current releases). These tools make raw file system-level copies of data files, along with the logs required for crash recovery to deal with consistency problems caused by concurrent changes. Here we run into a different meaning of I/O atomicity: reading from a file that might be concurrently written to. The first problem is that file systems on Linux and Windows (but not ZFS, or any file system on FreeBSD, due to the use of range locks) can show readers a random selection of before and after

bits when there is an overlapping concurrent write. Furthermore, the I/O is currently done in a way that isn't suitably aligned, so even on ZFS, torn pages could be copied. To defend against that, `full_page_writes` behavior is needed. This problem should eventually be fixed in PostgreSQL, by copying the raw data files with appropriate alignment and interlocking. Note that ZFS snapshots can be used instead of `pg_basebackup` if certain precautions are taken (primarily that the snapshot must atomically capture the logs and all data files), thus reducing the impact when cloning or backing up a very busy system.

Recovery

We've seen how `full_page_writes=off` improves the performance of write transactions, and ZFS makes that safe. Unfortunately there can also be negative performance implications for replication and crash recovery. These activities both perform *recovery*, meaning that they replay the log. Although full page images are a pessimization when they're written, they act as an optimization when they're replayed at recovery time. Instead of having to perform a random synchronous read that might block recovery's serial processing loop, we have the contents of the page to be modified already in our nice sequential log, and after that it is cached.

PostgreSQL 15 includes a partial solution to this problem: it looks ahead in the log to find pages that will soon be read, and issues `POSIX_FADV_WILLNEED` advice, to generate a configurable degree of I/O concurrency (a sort of poor man's asynchronous I/O). At the time of writing, FreeBSD ignores the advice, but a future version of OpenZFS will hopefully connect it up to FreeBSD's VFS (OpenZFS pull request #13958). Eventually, this should be replaced by a true asynchronous I/O subsystem that is currently being developed and proposed for a future version of PostgreSQL.

The effect of `full_page_writes=off` on recovery I/O stalls was studied by a group using PostgreSQL on ZFS on the illumos operating system at scale. They developed a tool called `pg_prefaulter` as a workaround. They had found that their streaming replicas couldn't keep up with their primary servers due to predictable I/O stalls. They may have been uniquely placed to see this effect since most large scale users of PostgreSQL don't even have the option of setting `full_page_writes=off`. `pg_prefaulter` may be a solution if you run into this problem, until built-in prefetching is available.

Looking Ahead

Block size alignment is likely to become a bigger topic in future PostgreSQL releases that will hopefully include proposed direct I/O support, which for now exists only in prototype form. This coincides happily with the development of direct I/O support for OpenZFS (pull request #10018), which will probably require block size agreement to work effectively (the current prototype reverts to the ARC otherwise; some other file systems simply refuse non-aligned direct I/O). Another OpenZFS feature in the works that is likely to be very useful for databases is block cloning (pull request #13392), along with new systems interfaces for FreeBSD, which PostgreSQL should hopefully be able to use for fast cloning of databases and database objects with finer granularity than whole datasets.

THOMAS MUNRO is an open source database hacker working for Microsoft Azure, who is usually logged into a FreeBSD box.

Virtual Lab – BSD Programming Workshop

BY ROLLER ANGEL

Our virtual lab will consist of a FreeBSD Host system that uses the FreeBSD Jails technology to provide each system we want to install within the lab its own separate environment to run services and perform its duties. These duties could be any number of things like serving up web pages, storing and retrieving database records, querying and answering DNS requests, caching system update files, etc. The idea is to build a solid foundation that will enable future growth for our virtual lab. Given that the nature of FreeBSD Jails is to provide a lightweight system for containing our services, we can rest assured knowing that any number of rabbit holes we may find ourselves in won't limit our creativity or exploration due to reaching our resource limitations. We can have a separate environment for each idea and not have to worry about the expense of provisioning another operating system to support the services necessary for that idea to flourish. I've experimented with many different methods of hosting operating system installs for my work and I'm pleasantly surprised by the peace of mind I get when provisioning a new jail. It's so economical! I'm no longer burdened with a financial concern and a question of how long this expense will be ongoing, rather it's just another environment in my ever growing lab and doesn't inherently come with a minimum monthly fee to use it. Not to get too technical and go down a financial rabbit hole regarding the cost of electricity, internet connectivity, host hardware; yes, I agree those things have a cost, but once they are in place, the addition of new hosts isn't anywhere near the considerations and expenses that go into the initial lab setup. I recommend repurposing an existing machine as the host machine, maybe even a laptop, as it comes with a built in battery backup, giving you time to gracefully shutdown your systems in the event of a sustained power outage. The issue of needing multiple physical network interfaces to connect physical Ethernet switches to Ethernet cables and access points that your physical hosts will use are not a material concern in our virtual lab. These interfaces can be created with words in a text file and virtual Ethernet cables can be created to connect the pieces of our virtual network.

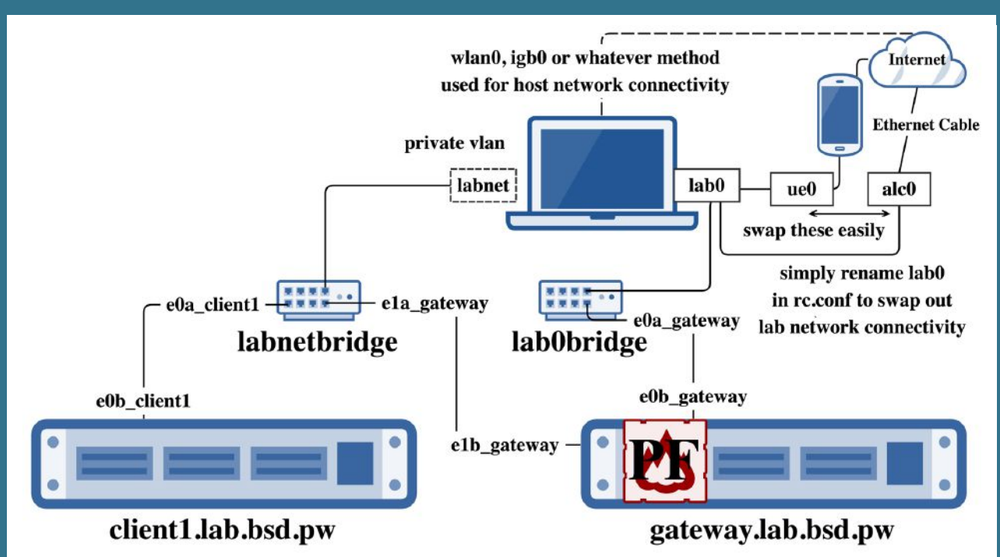
FreeBSD Host

This host machine will need a few network interfaces. We want the host to have its own way out to the internet. This can be a good ole DHCP assigned address provided by a local router, or your host could be performing the role of router and have a direct connection to the outside world through a modem. However your host gets its internet connection, we'll want to have an additional network interface that we can use solely for our lab network. This interface may have a name like `em0`, `igb0`, or similar depending on the network card driver and the number of installed

interfaces. Even if we don't have multiple physical network interfaces, we can always make a separate network interface to use with `cloned_interfaces`. In my case, I'm repurposing some laptops. One has a working WiFi card I use for connecting to the internet. This is an older Asus ROG laptop from the early 2010s. It has a built in Ethernet port as well. On my X1 Carbon 7th Gen ThinkPad, there aren't built in Ethernet ports. I just use the USB port and my Android USB Tethering. However, I'm most excited about the Dell Precision laptop. It's a great lab host as it comes with an Intel Xeon processor and can handle 128GB of RAM and multiple NVME hard drives. It has a built in ethernet port on the back of the laptop and the second one is a USB-C dongle that has an Intel `igb0` ethernet port on it that I use for the dedicated interface for the lab network. Finally, old towers are also great as a host, there are some fantastic PCI network cards you can install with mulitple Ethernet ports. The key here is to find what works for you, install FreeBSD and get going with our virtual lab. See the Getting Started With FreeBSD Workshop from the July/August 2022 *FreeBSD Journal* issue if you'd like to get some tips on installing and configuring FreeBSD.

Virtual Network Design

Just like in the physical server world where we have multiple physical Ethernet ports on our router/firewall systems, we can provision multiple interfaces for our lab router/firewall system which I will refer to as the gateway from now on. These interfaces will be connected to each other via bridges. Think of an Ethernet switch from a physical network, a bridge provides similar functionality. And for our Ethernet cables, we will use epair(4) interfaces. These consist of two sides, side A and side B. We'll hook up side A to the bridge and side B to our jail. This way all the jails can communicate with each other over the virtual bridge much like physical hosts in a network connected via Ethernet cable to a switch would be able to communicate with one another. The gateway jail will have an additional virtual cable to connect to a separate bridge that allows connectivity outside the lab and onto the internet. All jails will set their default route to point to this gateway. The way we give that separate bridge the ability to connect to the outside world is by assigning a physical interface as a member of the bridge. Again, this is akin to plugging the Ethernet cable into the switch, but in this case the Ethernet cable is plugged into the FreeBSD Host system and is then virtually plugged into the bridge. By doing this, the other virtual Ethernet cables that are also members of the same bridge will be able to communicate with it and get their packets flowing in and out of the lab environment.



Let's break it down, each jail gets its own network using a technology called vnet. We attach our physical interface on the FreeBSD Host to one virtual bridge and we create a second virtual bridge for our lab network that has only virtual interfaces from lab jails connected to it. We then use routing and firewall rules to push packets around as we see fit.

Virtual Lab – BSD Programming Workshop

Firewall

We'll be using PF for the firewall. The way FreeBSD Jails works is that each jail is sharing the host kernel, so if there's a kernel module that you want to have access to inside the jail, you just need to allow it and then configure access in the **`jail.conf`** settings for the particular jail. Take a look at the **`/etc/defaults/devfs.rules`** file. For our gateway to use the PF firewall, we'll need to set the configuration of the jail to use the pf ruleset listed in this file. We'll setup our own custom devfs rules later on and include the configuration from the **`devfsrules_jail_vnet`** rule.

Configuration

Now that we've covered what we're going to do, let's get to doing it. We start with a FreeBSD Host running 13.1-RELEASE. As described above, this host should have an active internet connection. We'll use that connection to download some files for use in creating our jails. The host has NTPD running, so it gets accurate time. Check for any services listening on the host with **`sockstat -46`** and turn them off if unused. Remember that the host should be limited in what it does—we'll have plenty of fun things to do in jails inside our lab, so do your best to limit the services on the host. I plan on doing any management of my host in person by logging in with the attached keyboard and screen so I've not enabled SSH on the host.

Now we're ready to enable jails. A simple **`sysrc jail_enable=YES`** will do the trick. No need to install any package, jail management is built into FreeBSD. Take a look at the README file in **`/usr/share/examples/jails`** for some examples of how you might configure your jails. As you will see, there are many ways to go about jail configuration. I've done my research and picked the configuration method you'll see here. You're welcome to give one of the other approaches a try and see what fits. If you do go about this task another way, please consider writing about it so others can see what you've found useful and give it a try themselves. At this point, we've verified our host machine is ready for hosting jails and have enabled the jail service so we can do a quick **`reboot`** double check that minimal services are listening on the host and move on to creating the base configuration for all our jails to use. When editing configuration files we'll be using **`vim`**, for basic editing tasks you really only need to know a handful of things, spend a few minutes going through the interactive exercises as part of the command **`vimtutor`** to get your bearings and you'll be a vim novice in no-time at all.

Note: We're running all the following commands as the root user. Type **`sudo -i`** to become root.

edit jail.conf

```
vim /etc/jail.conf
```

put the following into /etc/jail.conf

```
$labdir="/lab";
$domain="lab.bsd.pw";
path="$labdir/$name";
host.hostname="$name.$domain";
exec.clean;
```

Virtual Lab – BSD Programming Workshop

```
exec.start="sh /etc/rc";
exec.stop="sh /etc/rc.shutdown";
exec.timeout=90;
stop.timeout=30;
mount.devfs;
exec.consolelog="/var/tmp/${host.hostname}";
```

base.txz

```
mkdir -p /lab/media/13.1-RELEASE
cd /lab/media/13.1-RELEASE
fetch http://ftp.freebsd.org/pub/FreeBSD/releases/amd64/13.1-RELEASE/base.txz
```

Gateway Jail

```
mkdir /lab/gateway
tar -xpf /lab/media/13.1-RELEASE/base.txz -C /lab/gateway
```

edit jail.conf

```
vim /etc/jail.conf
```

add to the bottom of the file

```
gateway {
  ip4=inherit;
}
```

feel free to add a user account to the jail with the following optional command, for this article we're just going to be using the user root

```
chroot /lab/gateway adduser
```

set the root password for the jail

```
chroot /lab/gateway passwd root
```

setup DNS resolution using OpenDNS servers

```
vim /lab/gateway/etc/resolv.conf
```

add the following lines to resolv.conf

```
nameserver 208.67.222.222
nameserver 208.67.220.220
```

copy the hosts time zone setting

```
cp /etc/localtime /lab/gateway/etc/
```

create an empty file system table

```
touch /lab/gateway/etc/fstab
```

Virtual Lab – BSD Programming Workshop

start jail

```
jail -vc gateway
```

login to jail

```
jexec -l gateway login -f root
```

logout of the jail

```
logout
```

list jails

```
jls
```

stop the jail

```
jail -vr gateway
```

create devfs.rules

```
vim /etc/devfs.rules
```

add the following lines to devfs.rules

```
[devfsrules_jail_gateway=666]
add include $devfsrules_jail_vnet
add path 'bpf*' unhide
```

restart devfs

```
service devfs restart
```

verify devfs rules

```
devfs rule showsets
```

assign our new ruleset to the gateway jail

```
vim /etc/jail.conf
```

add the following line to the gateway { } config block

```
devfs_ruleset=666;
```

restart the gateway jail

```
service jail restart gateway
```

verify ruleset was applied to gateway jail

```
jls -j gateway devfs_ruleset
```

we expect to see 666 as the output of the above command

load the PF kernel module on host

```
sysrc -f /boot/loader.conf pf_load=YES
kldload pf
```

Virtual Lab – BSD Programming Workshop

enable PF on gateway jail

```
sysrc -j gateway pf_enable=YES
```

edit pf.conf on gateway jail

```
vim /lab/gateway/etc/pf.conf
```

add the following config

```
ext_if = "e0b_gateway"
int_if = "e1b_gateway"
table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }

#Allow anything on loopback
set skip on lo0

#Scrub all incoming traffic
scrub in
no nat on $ext_if from $int_if:network to <rfc1918>

#NAT outgoing traffic
nat on $ext_if inet from $int_if:network to any -> ($ext_if:0)

#Reject anything with spoofed addresses
antispoof quick for { $int_if, lo0 } inet

#Default to blocking incoming traffic, but allowing outgoing traffic
block all
pass out all

#Allow LAN to access the rest of the world
pass in on $int_if from any to any
block in on $int_if from any to self

#Allow LAN to ping us
pass in on $int_if inet proto icmp to self icmp-type echoreq
```

Configuring the Virtual Network

Setup an interface. Here we're using the dedicated network card named **alc0** and assigning it a name of lab0. All further configuration will use the interface name lab0 and the actual physical device it's assigned to can be changed by editing one line of configuration. This way it's easy to switch our interface if we move our lab to another host or add a new network interface down the road with a different name such as **igb0**, **re0**, or **em0**.

```
sysrc ifconfig_alc0_name=lab0
sysrc ifconfig_lab0=up
service netif restart
```

Virtual Lab – BSD Programming Workshop

copy the Jail Interface Bridge automation script into our lab scripts directory and make it executable

```
mkdir /lab/scripts
cp /usr/share/examples/jails/jib /lab/scripts/
chmod +x /lab/scripts/jib
```

edit jail.conf

```
vim /etc/jail.conf
```

Gateway jail.conf

At this point, we're ready to move from inheriting the ip4 network from the host and use vnet, remove the gateway {} configuration block from /etc/jail.conf and replace it with the following

```
gateway {
  vnet;
  vnet.interface=e0b_$name, e1b_$name;
  exec.prestart+="/lab/scripts/jib addm $name lab0 labnet";
  exec.poststop+="/lab/scripts/jib destroy $name";
  devfs_ruleset=666;
}
```

create the internal LAN network for the jails in the lab

```
sysrc cloned_interfaces=vlan2
sysrc ifconfig_vlan2_name=labnet
sysrc ifconfig_labnet=up
service netif restart
```

destroy and recreate gateway

```
jail -vr gateway
jail -vc gateway
```

configure networking for gateway jail

```
sysrc -j gateway gateway_enable=YES
sysrc -j gateway ifconfig_e0b_gateway=SYNCDHCP
sysrc -j gateway ifconfig_e1b_gateway="inet 10.66.6.1/24"
service jail restart gateway
jexec -l gateway login -f root
```

test connectivity

```
host bsd.pw
ping -c 3 bsd.pw
```

exit the jail

```
logout
```

Virtual Lab – BSD Programming Workshop

create another jail that only has one interface that's attached to the labnet LAN network

```
vim /etc/jail.conf
```

add the following to the bottom of the file

```
client1 {
  vnet;
  vnet.interface="e0b_$name";
  exec.prestart+="/lab/scripts/jib addm $name labnet";
  exec.poststop+="/lab/scripts/jib destroy $name";
  devfs_ruleset=4;
  depend="gateway";
}
```

make the directory structure for the new jail

```
mkdir /lab/client1
tar -xpf /lab/media/13.1-RELEASE/base.txz -C /lab/client1
```

set the root password

```
chroot /lab/client1 passwd root
```

setup DNS resolution using OpenDNS servers

```
vim /lab/client1/etc/resolv.conf
```

add the following lines to resolv.conf

```
nameserver 208.67.222.222
nameserver 208.67.220.220
```

copy the hosts time zone setting

```
cp /etc/localtime /lab/client1/etc/
```

create an empty file system table

```
touch /lab/client1/etc/fstab
```

start jail

```
jail -vc client1
```

configure networking for client1 jail

```
sysrc -j client1 ifconfig_e0b_client1="inet 10.66.6.2/24"
sysrc -j client1 defaultrouter="10.66.6.1"
service jail restart client1
```

login to jail

```
jexec -l client1 login -f root
```

Virtual Lab – BSD Programming Workshop

test connectivity

```
host bsd.pw
ping -c 3 bsd.pw
ping -c 3 10.66.6.1
```

grab a sample tcsh profile

```
fetch -o .tcshrc http://bsd.pw/config/tcshrc
chsh -s tcsh
```

exit the jail

```
logout
```

The next time you login, you'll have a green prompt due to the tcshrc settings, enjoy! Now you have a virtual lab with it's own virtual network that has a physical interface reserved for the outbound connection to the internet. Since we named the interface lab0, we can easily update it. Go ahead and give that a try. For instance, you can plug in an Android phone that has an internet connection, WiFi or Cellular is fine, as either will do. Go to the network settings on the phone after you plug it into the host and enable USB Tethering. A ue0 interface will now be available for use. Update the line in **`/etc/rc.conf`** that says **`ifconfig_alc0_name="lab0"`** to be **`ifconfig_ue0_name="lab0"`**. Reboot. Login to either jail and test connectivity. Your network has been swapped out. Your lab is now mobile!

I hope you had fun following along with this article. I'm super passionate about FreeBSD and sharing what I learn with others brings me joy. I hope you do amazing things with FreeBSD in your lab and I look forward to chatting with many of you at one of the fantastic BSD conferences.

ROLLER ANGEL spends most of his time helping people learn how to accomplish their goals using technology. He's an avid FreeBSD Systems Administrator and Pythonista who enjoys learning amazing things that can be done with Open Source technology — especially FreeBSD and Python — to solve issues. He's a firm believer that people can learn anything they wish to set their minds to. Roller is always seeking creative solutions to problems and enjoys a good challenge. He's driven and motivated to learn, explore new ideas, and to keep his skills sharp. He enjoys participating in the research community and sharing his ideas.

FreeBSD® guides

An Introduction to ZFS

BY DREW GURKOWSKI

ZFS combines the roles of volume manager and independent file system into one, giving multiple advantages over a stand-alone file system. It is renowned for speed, flexibility, and, most importantly, taking great care to prevent data loss. While many traditional file systems had to exist on a single disk at a time, ZFS is aware of the underlying structure of the disks and creates a pool of available storage, even on multiple disks. The existing file system will grow automatically when extra disks are added to the pool, immediately becoming available to the file system.

Getting Started

FreeBSD can mount ZFS pools and datasets during system initialization. To enable it, add this line to /etc/rc.conf:

```
zfs_enable="YES"
```

Then start the service:

```
# service zfs start
```

Identify Hardware

Before setting up ZFS, identify the device names of the disk associated with the system. A quick way of doing this is with:

```
# egrep 'da[0-9]|cd[0-9]' /var/run/dmesg.boot
```

The output should identify the device names, examples throughout the rest of this guide will use the default SCSI names: da0, da1, and da2. If the hardware differs, make sure to use the correct device names instead.

Creating a Single Disk Pool

To create a simple, non-redundant pool using a single disk device:

```
# zpool create example /dev/da0
```

To create files for users to browse within the pool:

```
# cd /example
# ls
# touch testfile
```

The new file can be viewed using ls:

```
# ls -al
```

We can already start using more advanced ZFS features and properties. To create a dataset within the pool with compression enabled:

```
# zfs create example/compressed
# zfs set compression=on example/compressed
```

The example/compressed dataset is now a ZFS compressed file system.
Disable compression with:

```
# zfs set compression=off example/compressed
```

To unmount a file system, use zfs umount and then verify with df:

```
# zfs umount example/compressed
# df
```

Verify that example/compressed is not included as a mounted file under the output.
The file system can be re-mounted with zfs:

```
# zfs mount example/compressed
# df
```

With the file system mounted, the output should include a line similar to the one below:

```
example/compressed 17547008 0 17547008 0% /example/compressed
```

ZFS datasets are created just like any other file system, the following example creates a new file system called data:

```
# zfs create example/data
```

Use df to see the data and space usage (some of the output has been removed for clarity).

```
# df
                                    . . .
example/compressed   17547008        0 17547008     0%    /example/compressed
example/data         17547008        0 17547008     0%    /example/data
```

Because these file systems are built on ZFS, they draw from the same pool for storage. This eliminates the need for volumes and partitions that other file systems rely on.

To destroy the file systems and then the pool that is no longer needed:

```
# zfs destroy example/compressed
# zfs destroy example/data
# zpool destroy example
```

RAID-Z

RAID-Z pools require three or more disks but offer protection from data loss if a disk were to fail. Because the ZFS pools can use multiple disks, support for RAID is inherent in the design of the file system

To create a RAID-Z pool, specifying the disks to add to the pool:

```
# zpool create storage raidz da0 da1 da2
```

With the zpool created, a new file system can be made in that pool:

```
# zfs create storage/home
```

Enable compression and store an extra copy of directories and files:

```
# zfs set copies=2 storage/home
# zfs set compression=gzip storage/home
```

A RAID-Z pool is a great place to store crucial system files, such as the home directory for users.

```
# cp -rp /home/* /storage/home
# rm -rf /home /usr/home
# ln -s /storage/home /home
# ln -s /storage/home /usr/home
```

File system snapshots can be created to roll back to later, the snapshot name is marked in yellow and can be whatever you want:

```
# zfs snapshot storage/home@11-01-22
```

ZFS creates snapshots of a dataset, allowing users to back up a file system for roll backs or data recovery in the future.

```
# zfs rollback storage/home@11-01-22
```

To list all available snapshots, zfs list can be used:

```
# zfs list -t snapshot storage/home
```

Recovering RAID-Z

Every software RAID has a method of monitoring its state. View the status of RAID-Z devices using:

```
# zpool status -x
```

If all pools are Online and everything is normal, the message shows:

```
all pools are healthy
```

If there is a problem, perhaps a disk being in the Offline state, the pool state will look like this:

```
  pool: storage
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using ‘zpool online’ or replace the device with
        'zpool replace'.
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        storage     DEGRADED     0     0     0
          raidz1    DEGRADED     0     0     0
            da0     ONLINE       0     0     0
            da1     OFFLINE      0     0     0
            da2     ONLINE       0     0     0

errors: No known data errors
```

"OFFLINE" shows the administrator took da1 offline using:

```
# zpool offline storage da1
```

Power down the computer now and replace da1. Power up the computer and return da1 to the pool:

```
# zpool replace storage da1
```

Next, check the status again, this time without -x to display all pools:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: resilver completed with 0 errors on Fri Nov 4 11:12:03 2022
config:

  NAME        STATE     READ WRITE CKSUM
  storage     ONLINE       0     0     0
    raidz1    ONLINE       0     0     0
      da0     ONLINE       0     0     0
      da1     ONLINE       0     0     0
      da2     ONLINE       0     0     0

errors: No known data errors
```

Data Verification

ZFS uses checksums to verify the integrity of stored data, these data checksums can be verified (which is called scrubbing) to ensure integrity of the storage pool:

```
# zpool scrub storage
```

Only one scrub can be run at a time due to the heavy input/output requirements. The length of the scrub depends on how much data is stored in the pool. After scrubbing completes, view the status with zpool status:

```
# zpool status storage
  pool: storage
 state: ONLINE
  scrub: scrub completed with 0 errors on Fri Nov 4 11:19:52 2022
config:


   NAME         STATE     READ WRITE CKSUM
   storage      ONLINE       0     0     0
     raidz1     ONLINE       0     0     0
       da0      ONLINE       0     0     0
       da1      ONLINE       0     0     0
       da2      ONLINE       0     0     0


errors: No known data errors
```

Displaying the completion date of the last scrubbing helps decide when to start another. Routine scrubs help protect data from silent corruption and ensure the integrity of the pool.

ZFS Administration

ZFS has two main utilities for administration: The zpool utility controls the operation of the pool and allows adding, removing, replacing, and managing disks. The zfs utility allows creating, destroying, and managing datasets, both file systems and volumes.

While this introductory guide won't cover ZFS administration, you can refer to zfs(8) and zpool(8) for other ZFS options.

Further Resources

While both the non-redundant and RAID-Z pools created using this guide will work in most use cases, more complex or specialized systems may require further ZFS management and setup. This guide barely scrapes the surface of what can be done using ZFS as it is an extremely powerful and customizable file system. The OpenZFS wiki has expansive documentation on installation, ZFS system administration, and manual pages. If tuning is required due to system architecture, ZFS tuning guides can be found on both the OpenZFS and FreeBSD wiki pages.

DREW GURKOWSKI, FreeBSD Foundation

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

WeGet letters

by Michael W Lucas

Dear FreeBSD Letters Columnist,

How are you? I am fine. Well, mostly fine. Sort of fine. Okay, no, I am not fine at all. I am a new sysadmin and utterly lost and am hoping you can offer guidance. I'm setting up a new web server, just like this issue is about, and I'm stopped in the installer. I hear all these things about optimizing filesystems for different applications, and there's lots of blog posts about ways to optimize, but I'm not sure which advice to take. Whatever I set up I must live with for years! Please help me make less bad choices.

—New Sysadmin

Dear NS,

Your letter is something of a relief, as it provides ample distraction from the horrors of administering the web server I set up in 2017, or my Sendmail configuration from 1992. It's like getting my mind off my abscessed tooth by busting a few of my ribs. Well done!

You might be a new sysadmin, but at least you understand that you must live with your bad decisions throughout the server's lifespan. Yes, you could get all DevOps and dynamically redeploy hosts with improved settings, but all you're doing is reducing the time you must live with one set of decisions before replacing them with a different set of equally bad ones. A change of poor choices is not as good as a rest.

How do you optimize a filesystem for a database at install time?

How do you optimize a filesystem for a database at install time? My answer is: don't. Premature optimization is the root of all evil, along with poor privilege management and nano. You have no idea how your database will interact with the filesystem until you run the application under real load. The only sensible choice is to arrange your new system so that you will have empty disks to move your database to. Yes, this is pretty much the same thing as devopsing to a new host, except it's not a new host and you don't need Ansible. If you're using one of those virtual host providers that offers block storage, that's dandy, except you'll be formatting those blocks with a filesystem. Where The Cloud is really "other people's computers," Block Storage is "other people's cast-off hard drives arranged in a Redundant Array of Inexpensive Crap." The main advantage is you're not the person who needs to trace the alarm beep to a drive tray.

If you insist on optimizing your filesystems, well, here's what you do.

First off, understand that storage devices are lying liars that lie. The newest solid state storage maintains a malformed compatibility with hard drives released in the previous century, which were built on standards designed for punch cards, which had their roots in 17th-century looms and the Luddites, so every time you plug in a storage device you're putting someone out of work but there's no ethical data storage under capitalism so go for it. The main lie that needs to concern you is the sector size. Today's drives overwhelmingly use 4K sectors, except for some NVMe devices that support multiple sector sizes but I don't have any of those so I'll pretend they don't exist. You need to make sure that the partitions—not the filesystems, the *partitions*—on your drives align with those 4K sector sizes. If the fancy boot loader you like requires a 98K GPT partition, it fills nineteen and a half disk sectors. Drives claim that they'll save you, but that's another lie. That next partition better begin at 100K, a nice multiple of 4, or all your filesystem blocks will be split between drive sectors and every interaction with the hardware will take twice as long and burn out the drive twice as quickly.

What about the filesystems? Filesystems need tuning! Balderdash, I say!

Once you have partitions, the filesystem blocks need to also be multiples of 4K. ZFS defaults to 128K stripes. UFS defaults to block sizes of 32K, which lets it use 4K fragments, so it should be good as-is but don't get clever and think that smaller blocks mean better performance because—no matter what a bunch of old blog posts say—they don't.

There you go. You can report to management that your filesystems are tuned for your hardware. Return to playing nethack.

But that's the partitions, I hear some of you whine. *What about the filesystems? Filesystems need tuning!* Balderdash, I say! You wouldn't tuna fish, why tune a filesystem? Filesystems are written for lazy people. Leave them alone, they have only caused you as much pain as their programmers insisted on, and that was only because marketing insisted on planned obsolescence to compel upgrades. BSD operating systems are not driven by profit, and user pain increases support requests, so the amount of filesystem agony has been methodically reduced until there's hardly any. Why, using filesystems barely qualifies as "torment" anymore. Any changes are likely to increase your pain.

You're still here? You still want advice?

Huh. You do know that therapists build careers out of helping people like you, right?

Fine.

Your filesystem should reflect your data. If you know that your data consists of, say, many 64KB files, you can set your blocks to that size. You know your own data, you should be able to figure this out. If your data is less predictable, don't optimize.

Databases can tempt even the most jaded sysadmin into optimizing their filesystem. Databases have predictable block sizes. MySQL uses a 16K block, so you could configure the underlying ZFS dataset to use a recordsize of 16K. MySQL can compress its data, as can ZFS. Attempting to compress already-compressed data wastes system resources. Study your application and pick a place to compress data.

Don't configure UFS to use 16K blocks, even when it's supporting MySQL. A UFS block has eight fragments, and thanks to the underlying disk each fragment has a minimum size

of 4K. Putting two UFS fragments on each disk drive sector would shatter performance. Tuning your MySQL install to use larger blocks on UFS might make sense, but again, leave the filesystem alone.

Postgres? 8K blocks everywhere by default. Again, tuning the database to match the disk might make sense, but 8K blocks on either ZFS or UFS ruin system performance. If your data demands 8K blocks, however, your best option is to use ZFS and set an 8K record size.

But in general, leave bad enough alone unless you want to make things worse. Which is a very common impulse among people who won't get the therapy they need. But rest assured, a few months of struggling to understand the interactions between applications and filesystems will soon make you an *experienced* system administrator.

You poor slob.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS has written over fifty books, and the UN's "Special Ambassador To Make Lucas Shut Up" was accidentally-on-purpose crushed beneath a stack of them. He wrote one book about UFS and two on ZFS. As the ZFS books were co-written with Allan Jude, they might even contain something helpful.

Rocky Mountain Celebration of Women in Computing

BY DEB GOODKIN

We were thrilled to be part of the Rocky Mountain Celebration of Women in Computing Conference, September 29-30, 2022, here in Boulder, Colorado! The Rocky Mountain Celebration of Women in Computing (RMCWiC) is a regional meeting modeled after the highly successful international Grace Hopper Celebration. The goal of RMCWiC is to encourage the research and career interests of local women in computing. RMCWiC has been held every two years since 2008 with a pause in 2020

RMCWiC offers an opportunity for students to present their research and to network with leaders from academia, government, and industry. In this way, RMCWiC provides a unique opportunity for technical women from Colorado and neighboring states to come

Just a few years ago, we were gaining momentum on showcasing FreeBSD at women in computing conferences and university groups. But that came to a standstill when Covid hit. We are now kickstarting that effort to attend more of these types of events, from meetups to celebration of women in computer conferences. So, I was thrilled when I saw the local Rocky Mountain Celebration of Women in Computing was taking place right here in Boulder, Colorado in September!

First, I love these events, because they are smaller and it's easier to talk to attendees about FreeBSD. This event brought in around 300 attendees from Colorado and surrounding states. I always love the energy of young folks as they meet others with similar interests in computing, while they learn from amazing role models in various technology fields.

> The goal of The Rocky Mountain Celebration of Women in Computing is to encourage the research and career interests of local women in computing.

I had the opportunity to give a talk on Open Source and why people should get involved. Of course, I used FreeBSD as an example of an open source project they should consider. After my talk, there was a career fair, where Justin Gibbs and I staffed a FreeBSD table, giving us the opportunity to talk with many of the attendees about FreeBSD. It was crazy loud, and everyone was wearing masks, so it was difficult, but we made it work. We had lots of attendees stop by our table to talk and ask us questions.

All in all, I'd say this was a great event for the Project, the students, and the Foundation. We always appreciate an opportunity to educate people about FreeBSD and encourage them to contribute to the project.

In 2023, we will be identifying a few women in computing conferences that we'd like to attend. Let us know if there is one you are familiar with that we should attend. Or maybe you'd like to present at one and staff a table in their career fair. We're here to support you if you choose that path!

DEB GOODKIN is the Executive Director of the FreeBSD Foundation. She's thrilled to be in her 15th year at the Foundation and is proud of her hardworking and dedicated team. She spent over 20 years in the data storage industry in engineering development, technical sales, and technical marketing. When not working, you'll find her on her road or mountain bike, running, hiking with her dogs, skiing the slopes of Colorado, or reading a good book.

WIP/CFT: Packet Batching

BY TOM JONES AND JOHN BALDWIN

In the last 30 years, the computers we use have grown unimaginably faster. The 1995 Alpha AXP paper talked about designing for a machine's continuing the trend of the previous 25 years, getting 1000 times faster.

We have certainly managed to meet that goal, the 386 machines that were the original target of FreeBSD are akin to the micro controllers we use in keyboards today.

Even with these changes, the core of computer performance has remained the same, execute fewer instructions per unit of work, and things will go faster. This fundamental truth underlying networking has led to several different approaches to improving performance. We have worked on mechanisms that moved work away from our CPU and, instead, into the network card with checksum offload. If the card runs the instructions to checksum outgoing packets, then our precious CPU time can be spent doing other things.

Checksum offload saw great results, and we started to move other things away from the CPU and into the network interface. TCP Segment Offload (TSO) was the next great mechanism that improved performance for a network sender. Rather than forming the IP packets for the TCP segments we will send, we can form one template and send that with a large block of data to the card. The network interface handles the segmenting as it places the packets onto the wire. TSO gives huge benefits to a TCP sender, providing us the ability to saturate 10-Gigabit network interfaces well before we run out of even a single core.

TSO lets us be more efficient with precious resources. We reduce the number of bus (memory and PCI) transactions required to send each packet by batching them together and creating the final chunks at the point of transmission. This is straightforward for TCP to do, most of the time if we are bulk sending a stream of data and the chunking of data is clear. To mirror these improvements on the TCP receiver, we have Large Receive Offload (LRO). LRO lets us again reduce the number of transactions required to maintain high-rate data transfers.

FreeBSD has excellent support for TSO and LRO, but is lacking mechanisms similar to GSO and GRO.

For UDP, Linux has generic mechanisms that attempt to replicate TSO-like mechanisms. This support comes with Generic Segment Offload (GSO) and Generic Receive Offload (GRO). GSO enables huge improvements on the order or 20% for a UDP sender, GRO is more difficult to measure, but the mechanism is there.

FreeBSD has excellent support for TSO and LRO, but is lacking mechanisms similar to GSO and GRO. At EuroBSDCon in Vienna last year I spoke to John Baldwin about a mechanism similar to GRO that he is working on, which he calls Packet Batching.

WIP/CFT: Packet Batching

TJ: What is the background to the packet batching work?
JB: The idea of packet batching on receive has been around for a while, at least in the form of a wish list item I've heard various people mention several times. We already have some forms of packet batching specific to TCP for both sending (TSO) and receiving (LRO). This packet batching aims to be more generic than LRO so that it can apply to other protocols (primarily UDP).
TJ: Why is the work needed?
JB: The goal of packet batching approaches such as TSO and LRO is to amortize per-packet costs (various checks in the network stack on header fields, etc.) by doing them once per batch rather than once per packet. The cost of per-packet overheads becomes an increasingly worse problem as network speeds increase faster than CPU speeds. It is true that one of the fixes for this problem, in general, which does help with per-packet overhead, is horizontal scaling by using RSS to distribute packets across separate queues bound to different CPUs. However, you can't distribute a single flow across multiple cores, and batching schemes are aimed at making the performance of a single queue more efficient.
TJ: What new features/enhancements does the work make possible?
JB: The goal is higher PPS and/or reduced CPU usage for network received workloads. I don't expect it to help with TCP when LRO is enabled, mostly to help with UDP.
TJ: How can people test the work? Normally we need to emphasize testing with more diverse workloads, does this apply here?
JB: Benchmarking would be welcome. My initial set of simple benchmarks using iperf3 were mixed and not a clear enough win to justify the changes. The changes do add complexity, so it needs to be a clear win in some workloads, I think, before it should be considered a commit candidate. I have not observed any regressions in my benchmarks to date, just meager to zero gains.
TJ: How would you like feedback?
JB: E-mail directly to me is probably the best way to send feedback for now. At some point in the future, I will start a public RFC thread on net@ and/or arch@ at which point that thread will be the best place to send feedback. Folks wishing to test the patches or review them can find them at https://github.com/freebsd/freebsd-src/compare/main...bsdjhb:freebsd:cxgbe_batching.

From John's responses here, it isn't yet clear where the benefits should be seen. iperf3 measurements can't simulate the workload of a very busy server. For Packet Batching to offer a benefit in FreeBSD it is likely that more workloads need to be tested and tuned for. By pulling down John's github branch and experimenting with your network traffic, you can help establish a new receiver optimization in FreeBSD.

TOM JONES wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting.

JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for 20 years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger and LLVM. John lives in Concord, California, with his wife, Kimberly, and three children: Janelle, Evan, and Bella.

The Foundation and the FreeBSD Desktop

BY ANNE DICKISON

The Desktop experience can be formative. I got my first PC in 1990 as an 8th Grade graduation gift. (Thanks Dad!) It helped instill my interest in computers and it got me through high school. I used it mostly for playing Zork, Jeopardy and, of course, writing papers on Word Perfect. The interface was rather clunky, but for the purposes of a small-town high school student in the 90s, it worked quite well. Once college came about, a new machine came my way and a GUI that made things work so much better. Using a computer became part of everyday life. In fact, one of the selling points of my university was that every dorm had its own desktop. Fast forward 20+ years, and the standards for a usable desktop are quite high. Intuitive, fast, pretty graphics, and speedy wi-fi are all expected. FreeBSD's desktop experience over the years has had its ups and downs. Twenty or so years ago, FreeBSD and Linux were mostly neck and neck in terms of desktop usability. Unfortunately, as time went on, FreeBSD did fall behind. The desktop experience became a lower priority. However, catch up eventually ensued and within the last 10 or so years, focusing on the desktop has increasingly become of greater importance for many members in the community. To help understand more about the Foundation's work on the desktop experience, we sat down with Ed Maste, Senior Director of Technology.

Unsurprisingly, one question the Foundation often gets is where the desktop experience falls in our list of priorities. The answer: Well, it varies. Because the Foundation's main goal is to support the Project in technical areas that aren't being fully addressed by the community, the desktop sponsored work ebbs and flows. When work stagnated about 10 years ago and the Project began to fall behind in terms of hardware support, the Foundation funded Kostik Belousov to work on Intel Graphics Drivers. More recently though, the Project has moved to using the Linux Kernel Interface (KPI) to help keep drivers up to date. The Foundation funded Bjorn Zeeb to work on the wireless side, and about 2 years ago, it funded Emmanual Vadot to work on graphics drivers.

These days, the FreeBSD community has continued the graphics work via the Linux KPI, while the Foundation is funding Bjorn to do the same on the wireless side. The net result is that generally, you can take a contemporary x86 laptop or desktop system, and the graphics and wireless will just work. The hope with this method is that as each, new generation

of hardware comes out, we'll be able to take the latest upstream drivers and just use them without any sort of significant rework to make them work on FreeBSD. Ed notes that while using the Linux KPI might not be the most popular solution, it does seem to be the most developer-efficient way to keep the drivers up to date.

"In an ideal world, with unlimited resources and an unlimited supply of qualified technical people, I would just have developers create bespoke FreeBSD drivers. While the current method may have its detractors, the result is that FreeBSD has a working driver that is performant and featureful, that should allow us to basically remain up to date."

Speaking of up to date, Ed was quick to mention one caveat when it comes to wireless drivers. While the wi-fi does work out of the box on many desktop systems, the speed is sometimes lacking in comparison to contemporary wi-fi standards. That doesn't mean you can't use FreeBSD as your daily desktop though. It's fast enough for video, conference calls, and web browsing. Ed mentioned how Bjorn's work has made the wi-fi on his Framework laptop stable and reliable. But when it comes to downloading large files, you will notice slower speeds. The Foundation has extended Bjorn's contract into 2023 and he is working on those standards now with the goal of having it available in FreeBSD 14.0, if not 13.2.

However, as mentioned above, FreeBSD can be used as your daily driver, an aspect that is very important to Ed and the Foundation. One of the reasons the Foundation has chosen to support the wi-fi efforts as of late is that there's a huge amount of value in being able to use the operating system that you're developing on as your desktop machine. In fact, Ed sees that as being connected to the Project's long-term viability and the ability to bring on new users.

"Let's take someone who is in university, I think it really is the case that FreeBSD is the best operating system for someone who is interested in learning about operating system internals. Someone who wants to become an operating system developer or wants to explore and learn about operating systems. FreeBSD is advanced enough that it can do what you need, but you can still find a niche and make your own impact. But, without a user-friendly desktop experience, it's hard to make the argument that someone should try FreeBSD if they're already familiar with Linux on their laptop."

Thanks to great work from members of the community along with Foundation supported efforts in key areas, the FreeBSD desktop experience is on a positive trajectory. As we head into 2023, Ed says the Foundation plans to continue to support Bjorn's wi-fi work and take another look at the installer to help make sure that you're able to get a usable graphical desktop environment--out of the box. Of course, that all may change as 2023 progresses, but ultimately, Ed and his team are dedicated to working with other community members to produce a modern and user-friendly desktop experience.

ANNE DICKISON joined the Foundation in 2015 and brings over 20 years experience in technology-focused marketing and communications. Specifically, her work as the Marketing Director and then Co-Executive Director of the USENIX Association helped instill her commitment to spreading the word about the importance of free and open source technologies.

2023 Events Calendar

BSD Events taking place through April 2023

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

SCALE 20X

March 9-12, 2023
Pasadena, CA
https://www.socallinuxexpo.org/blog/scale-20x

SCaLE is the largest community-run open-source and free software conference in North America. It is held annually in the greater Los Angeles area. Roller Angel will also be hosting a FreeBSD workshop during the conference.

Open Source 101

March 23, 2023
Charlotte, NC
https://opensource101.com/

Open Source 101 is a one-day educational conference covering the processes and tools foundational to open source, open tech, and the open web. All sessions will be delivered virtually/online and at an introductory or intermediate level to allow for the best onramp or refresh possible. Target audience includes developers, technologists, students and decision makers. The format includes TED-style keynote talks as well as track sessions from industry experts. The FreeBSD Foundation is pleased to be a Media Partner.

AsiaBSDCon 2023

March 30-April 2, 2023
Tokyo, Japan
https://2023.asiabsdcon.org/

AsiaBSDCon is for anyone developing, deploying and using systems based on FreeBSD, NetBSD, OpenBSD, DragonFlyBSD, Darwin and MacOS X. It is a technical conference and aims to collect the best technical papers and presentations available to ensure that the latest developments in our open source community are shared with the widest possible audience.

FreeBSD Office Hours

https://wiki.freebsd.org/OfficeHours

Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions.

Past episodes can be found at the FreeBSD YouTube Channel.
https://www.youtube.com/c/FreeBSDProject.