# Parallel, Multi-Axis Regression and Performance Testing with FreeBSD, OpenZFS, and bhyve

Michael Dexter
editor@callfortesting.org

AsiaBSDCon 2019, Tokyo, Japan

Contemporary Unix, defined as the sum of open source BSD Unix projects, Illumos distributions and GNU/Linux distributions, plus the OpenZFS cross-platform file system, can attribute their success to the collaborative work of like-minded academic, commercial, and volunteer developers around the world. Governed by a mix of licenses, best practices, community norms, and personal passion, open source projects like modern Unix operating systems and OpenZFS largely lack centralized Quality Engineering institutions, deferring Quality Engineering and Quality Control responsibilities to participating developers and the end user. This arrangement promises the widest-possible array of regression and performance testing tools, loads, and procedures, at the expense of providing any guarantees, true to the disclaimers of the licenses under which these projects are distributed. This paper will examine how "parallel, multi-axis" testing, defined as testing multiple software versions, operating systems, "options", compilers, and architectures, or *axes*, *in parallel*, will improve the identification and isolation of reliability and performance regressions.

### Identifying a Computing Axis

Borrowing from the mathematical definition of an Axis, *a fixed reference line for the measurement of coordinates*, the quintessential *computing axis* is any given software versioning: it increments, in the case of SVN revisions, from zero to an architectural limit, where the highest number is always the latest revision and any point in the history is easily located and visited. Less-linear, yet equally traversable axes include multiple operating systems, *their* sequential versions, their userland and kernel build "options", their supported computing architectures, and their supported hypervisors. Each of these axes is of equally-unique and identifiable value, enabling for their linear traversal and most importantly, testing in *parallel*. Parallel testing is facilitated by multiple identical hardware machines, unless of dissimilar hardware architecture, and multiple virtual machines executed in parallel. While identical hardware machines will provide the greatest consistency for performance testing, virtual machines are adequate for providing meaningful reliability testing of many computing resources.

### The Version Axis

Of the testing Axes within the scope of this paper, the version axis is the most familiar. Incrementing software versioning is provided manually by the developer, or automatically by a version control system. The testing host operating system for this paper, FreeBSD, provides two distinct version identifiers: Named Releases and pre-releases, i.e. 12.0-RELEASE and 12.0-RC1, and incrementing SVN Revisions, i.e. 341707. These named and numeric identifiers allow for unambiguous revision identification, in contrast to the hash-based

"numbering" semantics used in some version control systems. Two distinct challenges exists however, to obtaining open source releases by named binary Release: incomplete historical release preservation and the rising popularity of distributing open source operating systems via content delivery networks (CDNs). The first of these challenges can largely be attributed to the unavailability of terabyte and larger-capacity storage devices until late in each operating system project's history, allowing for centralized and distributed preservation of project history. The second of these challenges is simply the fact that content *distribution* networks are designed for the rapid *distribution* of the latest software releases, and not the *preservation* of historic releases. The result is a rapid expiration of available releases distributed via CDN than with traditional mirrors. Part of this paper's work is a rebuilding of the FreeBSD release history in coordination with users around the World.

**The Operating System Axis**

The set of available operating systems is extensive, and multiplied by their individual version axes, overwhelming. The scope of this testing will be limited to operating systems that support the OpenZFS file system with a limited number of versions and a goal of all supported architectures. The result is a focus on FreeBSD, NetBSD, Illumos derivatives, primary (not derived) GNU/Linux distributions, macOS, and Microsoft Windows.

**The "Options" Axis**

Of the target operating systems within the scope of this work, FreeBSD is rich with userland and kernel "source build options" that determine what features are included or excluded from the compiled operating system. Similar to the situation with FreeBSD historic releases however, these options are often under-documented or non-functional, resulting in the first test of this paper: ongoing Build Option Survey (/usr/src/tools/tools/build_option_survey/) runs, and the development of a "STUDENT" kernel configuration file that progressively introduces the options needed to build and eventually boot the FreeBSD kernel under the bhyve hypervisor.

**The Compilers Axis**

While FreeBSD employs the Clang compiler as its default, in-base compiler, the buildability of the operating system with the GCC and other compilers provides an important validation vector. FreeBSD 8.0 could be built with the Portable C Compiler (pcc) and this testing will facilitate the institutional compilation of FreeBSD with alternative compilers and across the versions axis. By extension, FreeBSD's promise, but not guarantee that each previous and future release of FreeBSD should be buildable under any given release, a forward/backward version axis traversal test should be trivial to conduct.

**The Architecture Axis**

FreeBSD offers the most OpenZFS-supported architectures of any operating system. The relative low-cost of embedded and used non-Intel machines allow for this testing to include non-Intel architectures including ARM, ARM64, PowerPC, and Sparc64. GNU/Linux distributions are candidates for inclusion when their non-Intel support expands.

## The POSIX Testing Environment

Testing in parallel requires, by definition, a consistent testing environment in order to provide meaningful results. It is tempting to consider a cross-platform system orchestration solution such as Ansible or Puppet for the task of abstracting away platform-specific nuances, but these solutions provide high-overhead in exchange for limited domain-specific abilities, notably testing, rather than configuration. In consideration of the fact that the majority of the operating systems in the testing scope are near-POSIX compliant, establishing a common POSIX testing environment is the most reasonable strategy to minimize platform-specific nuances. In service of the goal of traversing the version axis on FreeBSD back to "historic" releases, a POSIX environment becomes a firm requirement for want of modern system orchestration tools on anything but the most recent operating system releases. In service of testing the Windows operating system, the Cygwin near-POSIX environment has proven the most flexible with the widest of array of third party open source packages for the Windows operating system.

With the operating system-specific ABI requirements of a POSIX environment satisfied, a base set of utilities will provide near-identical functionality on all platforms in the scope of the testing. These utilities include at a minimum `sh(1)`, `ssh(1)`, `time(1)`, `date(1)`, `touch(1)`, `dd(1)`, `truncate(1)`, `mkdir(1)`, `rmdir(1)`, `sha512(1)`, `zpool(1)`, `zfs(8)`, and `ztest(1)`. Supplementary utilities include `gdate(1)` for higher-resolution timing, and traditional benchmarking utilities like `bechmarks/fio`, `bechmarks/bonnie++`, and `bechmarks/sysbench`. Of these tools, disk partitioning utilities are the most platform-dependent, but fortunately, any discrepancy in the execution times of partitioning tools across operating systems are not relevant to to the runtime testing of a file system.

Finally, the bhyve Hypervisor and Jail containment system are essential to both preflighting tests prior to their deployment on dedicated hardware and the execution of some tests, such as those on historic versions of FreeBSD.

## The ptime(1) Utility

While the POSIX standard is well established, it makes makes no guarantees as to the machine parsability of utility output. This paper proposes the `ptime(1)` or *precision time* utility to provide enhanced, machine-parsable execution reporting to standard I/O shell interpreter pipelines:

```
NAME
            ptime - Precision execution time utility

SYNOPSIS
            ptime [options] [command]

DESCRIPTION

      ptime provides Unix epoch time and utility execution time in seconds,
milliseconds and nanoseconds. It can also provide the time difference
between two files based on their datestamps.

OPTIONS
      -h          Display usage
      -s          Display output in seconds (default without -s)
      -m          Display output in milliseconds
      -n          Display output in nanoseconds
```

```
       -f          First file (requires -l)
       -l          Last file (requires -f)
       -r          Override return value with output

EXAMPLES

       Output Unix Epoch time in seconds (-s implied)

       ptime
       1544000077              <equivalent to date +%s>

       Output Unix Epoch time in milliseconds

       ptime -m
       154849734068255         <equivalent to (( gdate +%s%N ))/1000000>

       Output Unix Epoch time in nanoseconds

       ptime -n
       1548497340682551000     <equivalent to gdate +%s%N>

       Output execution time of 'sha256 -t' time in nanoseconds

       ptime -n sha256 -t
       1548497340682551000

       Output the time difference between two files

       ptime -f /build/firstfile -l /build/lastfile
       123467

       Return Unix Epoch time in seconds

       ptime -r
       echo $?
       1544000077
```

Additional tools include the `bd(8)` block device, and `be(8)` boot environment utilities for the management of block device partitioning and formatting, and OpenZFS boot environments respectively (Dexter, AsiaBSDCon 2018).

**Regression and Performance Testing**

Equipped with a cross-platform, near-POSIX test environment and support utilities, a baseline of tests can be performed along each axis.

FreeBSD Version and Compiler Axis: Build forward and backward versions of FreeBSD on any given version with the built-in compiler and optional compilers.

FreeBSD Option Axis: Extend the Build Option Survey framework or a new framework to kernel configuration file build options, identifying their interdependencies.

bhyve Hypervisor vCPU Topology: Validate the January, 2019 bhyve vCPU topology improvements (reviews.freebsd.org/D18815 and related) that allow for up to 65 packages/sockets and 255 cores per package. Step through additional packages and cores one by one. This test is performed with a wrapper script that simply boots a virtual machine that is designed to shut down via `/etc/rc.local`. This test should eventually traverse the operating system axis, ensuring that a representative set of non-FreeBSD operating systems are validated with difference vCPU configurations.

OpenZFS Testing along the Operating System and Architecture Axes: Perform a myriad of tests *in parallel* across the operating system axis: repeated zpool creation and destruction, nested directory and file creation, high-count file `touch(1)`ing, cross-platform pool importation, identification of SMB and NFS performance cliffs based on the amount of data transferred, scripted `fio(1)` testing, and execution of the OpenZFS `ztest(1)` suite. With new OpenZFS platforms like Windows emerging, this testing has revealed that basic assumptions cannot be made, such as the success of the `touch(1)` utility.

**Conclusions**

The parallel, multi-axis testing approach for regressions and performance telemetry should provide new insights into reliability and performance issues that will be overlooked by domain-specific testing. This work is inspired by real-world OpenZFS on FreeBSD performance issues and combined with a version axis bisection strategy, should identify regressions at a faster pace than is possible with traditional testing methods. This testing also aims to accelerate the stability of new OpenZFS platforms including NetBSD and Windows. Finally, all of the tools used in this testing will be available on GitHub or equivalent.