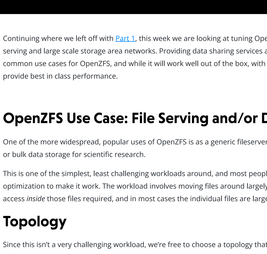
OpenZFS Storage Best Practices and Use Cases – Part 2: File Serving and SANs

NOVEMBER 28, 2023

In our continuing series of ZFS best practices, we examine several of the most common use cases around file serving, and provide configuration tips and best practices to get the most out of your storage.



Continuing where we left off with Part 1, this week we are looking at tuning OpenZFS for specific use cases involving file serving and large scale storage area networks. Providing data sharing services as a NAS or SAN is one of the most common use cases for OpenZFS, and while it will work well out of the box, with the right tuning and optimizations, it can provide best in class performance.

OpenZFS Use Case: File Serving and/or Data Warehousing

One of the more widespread, popular uses of OpenZFS is as a generic fileserver—which might mean an office fileserver, or bulk data storage for scientific research.

This is one of the simplest, least challenging workloads around, and most people will not need any sort of performance optimization to make it work. The workload involves moving files around largely intact, with little or no random access *inside* those files required, and in most cases the individual files are large as well.

Topology

Since this isn't a very challenging workload, we're free to choose a topology that maximizes fault tolerance and storage efficiency, rather than one which maximizes bare metal performance. If you've got a lot of disks, this likely means RAIDz2—ideally, in vdevs four, six, or ten disks wide.

Once you've picked your topology, you'll want to set a relatively large recordsize on the datasets used for file service—even if the individual files average considerably smaller. This is because OpenZFS can and does automatically store small files in small blocks—a 3KiB file will be stored in a single 4KiB sector regardless of recordsize, but larger files are split into chunks of the recordsize, and fewer chunks will provide more efficient storage.

Tunables

Keeping our recordsize large when random access inside files is rarely required does several things for us—it minimizes fragmentation, reduces the IOPS necessary to fulfill requests, and maximizes compression ratios. This is also ideal for large media files and other similar use cases.

If you're specifically running a Linux fileserver, and you have lots of files—especially small files—you may also want to consider setting **xattr=sa**, which makes OpenZFS store a file's metadata inside the first block of the file itself, rather than in a separate object.

Support vdevs

If your fileserver primarily interacts with other machines via NFS, you may want to consider a **LOG** vdev—most NFS exports are synchronous, which means acceleration of sync writes will drastically increase performance. But if you *aren't* using NFS—or if you're specifically using asynchronous NFS and are okay with that fact—you don't need a **LOG** vdev and having one won't make your system go any faster.

You might *consider* a **CACHE** vdev—also known as L2ARC—for your fileserver. But **CACHE** vdevs tend to provide far less benefit than most users assume they will. It's almost always better to spend that money on more RAM, rather than on an SSD to use as a **CACHE** vdev. Consider consulting Klara for design review or system audit to see if an L2ARC is right for your workload.

A **SPECIAL** vdev can be useful in systems with very wide, slow vdevs (think ten-wide RAIDz2 on rust disks). The **SPECIAL** stores metadata blocks itself rather than on the pool's main storage, which can significantly accelerate metadata-heavy operations such as listing or calculating the size of millions of files.

With that said, a **SPECIAL** should not be implemented casually. Losing the **SPECIAL** means losing the entire pool along with it—so it must be just as redundant as the pool's storage vdevs and should use devices with high reliability and write endurance. Sizing a **SPECIAL** vdev is extremely tricky, but the Klara team has the experience to measure your current workload and estimate future capacity requirements and help you make the right choice.

OpenZFS Use Case: Massive Storage Servers / SANs

If you're building a truly massive storage system—anywhere from 60 drives to *hundreds* of drives—the sheer size of your pool can become more important than the individual workloads running beneath it.

Most "massive" servers are used primarily for simple file service, possibly with a few light database workloads thrown in for fun. It's not usually a good idea to run performance-critical databases on massive servers, since latency is the most critical metric for databases, and it's difficult to ensure predictable latency from massive systems with high user counts and variable workloads.

Topology

With sixty plus disks on hand, the performance bottleneck tends to shift away from either individual drives or even entire vdevs. Instead, massive systems tend to bottleneck on their storage controllers—and on the ways those controllers connect to individual drives.

With the per-drive and even per-vdev performance focus decreased due to sheer drive count, we recommend paying more attention to storage efficiency and fault tolerance. For the absolute maximum performance and not a care in the world for storage efficiency, this means three-way mirrors.

Most admins of massive servers will tend to prefer collections of wider RAIDz vdevs—this can mean six-wide or ten-wide RAIDz2 vdevs, or seven-wide or eleven-wide RAIDz3 vdevs. Remember, narrower vdevs means higher performance, but less usable storage. The wider you go, the lower (and less predictable) your performance will get. Finding the right tradeoffs take experience and knowledge of ZFS internals.

You'll also likely want **SPARE** vdevs at this scale. A **SPARE** is a single disk which is added to the pool, but not immediately used for storage—instead, the pool can automatically resilver a **SPARE** into any vdev which experiences a drive failure.

Finally, you may at this scale consider the new DRAID topology. DRAID essentially functions like a collection of RAIDz vdevs and spares, all rolled into one vdev: for example, a **draid2:8d:2s** is functionally equivalent to a collection of 10-wide RAIDz2 vdevs, with two **SPARE** vdevs thrown in. If you are not familiar with it, here is an introduction to DRAID.

In a server which only has twelve bays, you'd be better off with the traditional ten-wide RAIDz2 plus two **SPARE**s. But in a server with 60 storage disks, you can put all 60 of them in that single **draid2:8d:2s**—making it the equivalent of six 10-wide RAIDz2 vdevs plus two spares.

The killer feature here revolves around the "spares" built into the DRAID vdev—they're not idle spare disks, they're spare *capacity* distributed among all sixty disks. This means that when a drive fails, the data on that disk doesn't need to resilver onto a single additional disk, it resilvers onto all fifty-nine of the remaining disks, at a massively faster rate.

There are downsides, though—for one, DRAID gives up on RAIDz's dynamic stripe size.

On a standard RAIDz2 vdev with **recordsize=1M** set, a 4KiB file will only occupy 12KiB on three drives—one sector of data, and two sectors of parity.

But if you've set **recordsize=1M** on a dataset on a **draid2:8d:2s** and you then save a 4KiB file to it, that file will occupy a full stripe distributed across ten drives (eight data and two parity). This makes the use of a **SPECIAL** vdev near-mandatory on DRAID systems.

You also don't entirely lose the need for conventional **SPARE**s with DRAID—the initial resilver onto spare capacity goes tremendously faster than a resilver onto a replacement drive, but you can't get that spare capacity *back* without replacing the actual failed drive.

Tunables

Massive storage systems are best suited to massively multi-user file service, which usually means tuning recordsize very large—1M, or potentially even larger.

It's usually a safe bet that massive storage servers will host massive numbers of files, so both **xattr=sa** and **atime=off** tend to be good ideas. You'll also want compression enabled, both to save space and increase performance.

Support vdevs

Sync writes can tie up massive numbers of drives, so a **LOG** vdev is almost always a good idea at this scale. If you have a lot of sync writes, be sure to spec your **LOG** vdev's physical device(s) accordingly: a fully-NFS-accessed massive storage server will need incredible write endurance to its **LOG.**

There's rarely a good case to be made for **CACHE** vdevs on massive servers. With enough spindles, even cheap, slow rust disks can more than saturate your storage controller—and once the storage controller is saturated, performance cannot go up any further.

The **SPECIAL** vdev is basically a must-have for any DRAID server. Remember, choosing DRAID meant giving up on dynamic stripe size—so each 4KiB metadata block ends up wasting 40+ KiB of space on a DRAID.

You may also want to tune your **SPECIAL** vdev to store small files—the optional **special_small_blocks** tunable can help here. This introduces a whole new wrinkling in the capacity planning for the **SPECIAL** vdev, as if it fills with small data blocks it can no longer offload metadata and performance will suffer greatly. If you are building a large array, or the workload is sensitive to latency, be sure to consult with Klara when deciding how many **SPECIAL** vdevs you need.

For example, say you've got an AI training workload which consists of lots of 16MiB+ files, each of which has a single 8KiB associated metadata file. If you set **special_small_blocks=64K**, then your metadata files will all land on the **SPECIAL**, which increases storage efficiency, reduces IOPS demands, and can massively increase the speed of any index-reading workloads (eg grepping through XML files looking for keywords).

As always, losing the **SPECIAL** means losing the entire pool along with it—so your **SPECIAL** must be at least as redundant as the pool, meaning three devices for each **SPECIAL** vdev supporting a pool of RAIDz2 vdevs (or a DRAID2 based pool).

In a massive server, there is *no* excuse for using cheap consumer SSDs as support vdevs, let alone trying to partition a single SSD to serve multiple roles. Enterprise-targeted SSDs with high write endurance, hardware QoS, and onboard nonvolatile cache should be considered *minimum* spec for support vdevs on massive servers!

Conclusions

There are many considerations when building critical infrastructure, especially storage, but you don't have to do it alone. Klara's ZFS Design and Implementation solution will guide you through every step of the process, from picking the right hardware and pool layout, to getting your new storage server integrated into your environment.

Klara has designed storage servers of every size, from small special purpose all NVMe NASes to enormous 10 PiB pools for mass data storage, and everything in between. Are you looking to run a database or your VM infrastructure on top of ZFS? Watch for Part 3 of our ZFS best practices series where we discuss how to tune ZFS for those workloads.

Topics / Tags

openzfs

Back to Articles