**ZFS, copies, and data protection**

**Friday May 04, 2007**

OpenSolaris build 61 (or later) is now available for download. ZFS has added a new feature that will improve data protection redundant copies for data (aka ditto blocks for data). Previously, ZFS stored redundant copies of metadata. Now this feature is available for data, too.

This represents a new feature which is unique to ZFS: you can set the data protection policy on a per-file system basis, beyond that offered by the underlying device or volume. For single-device systems, like my laptop with its single disk drive, this is very powerful. I can have a different data protection policy for the files that I really care about (my personal files) than the files that I really don't care about or that can be easily reloaded from the OS installation DVD. For systems with multiple disks assembled in a RAID configuration, the data protection is not quite so obvious. Let's explore thisfeature, look under the hood, and then analyze some possible configurations.

## Using Copies

To change the numbers of data copies, set the *copies* property. For example, suppose I have a zpool named "zwimming." The default number of data copies is 1. But you can change that to 2 quite easily.

```
# zfs set copies=2 zwimming
```

The copies property works for all new writes, so I recommend that you set that policy when you create the file system or immediately after you create a zpool.

You can verify the copies setting by looking at the properties.

```
# zfs get copies zwimming
NAME      PROPERTY  VALUE     SOURCE
zwimming  copies    2         local
```

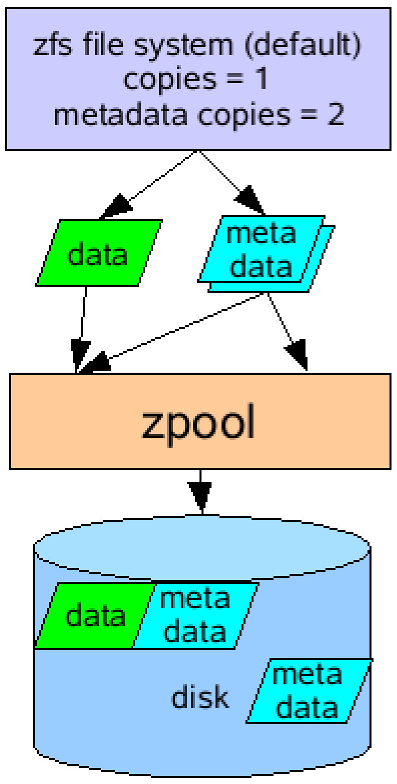
ZFS will account for the space used. For example, suppose I create three new file systems and copy some data to them. You can then see that the space used reflects the number of copies. If you use quotas, then the copies will be charged against the quotas, too.

```
# zfs create -o copies=1 zwimming/single
# zfs create -o copies=2 zwimming/dual
# zfs create -o copies=3 zwimming/triple
# cp -rp /usr/share/man1 /zwimming/single
# cp -rp /usr/share/man1 /zwimming/dual
# cp -rp /usr/share/man1 /zwimming/triple
# zfs list -r zwimming
NAME              USED  AVAIL  REFER  MOUNTPOINT
zwimming         48.2M   310M  33.5K  /zwimming
zwimming/dual    16.0M   310M  16.0M  /zwimming/dual
zwimming/single  8.09M   310M  8.09M  /zwimming/single
zwimming/triple  23.8M   310M  23.8M  /zwimming/triple
```

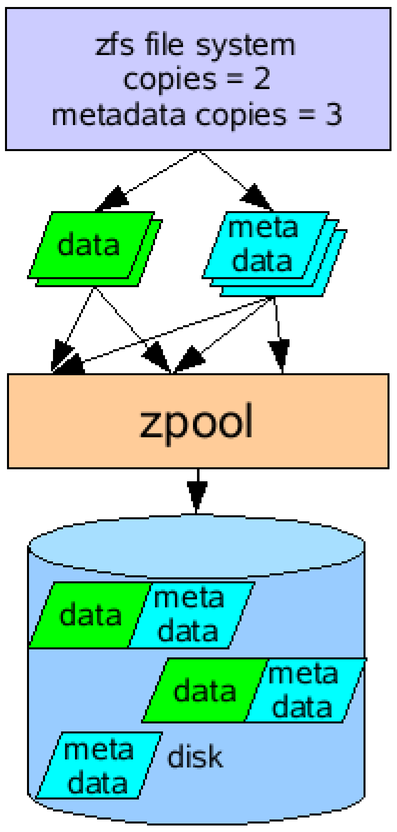
This makes sense. Each file system has one, two, or three copies of the data and will use correspondingly one, two, or three times as much space to store the data.

## Under the Covers

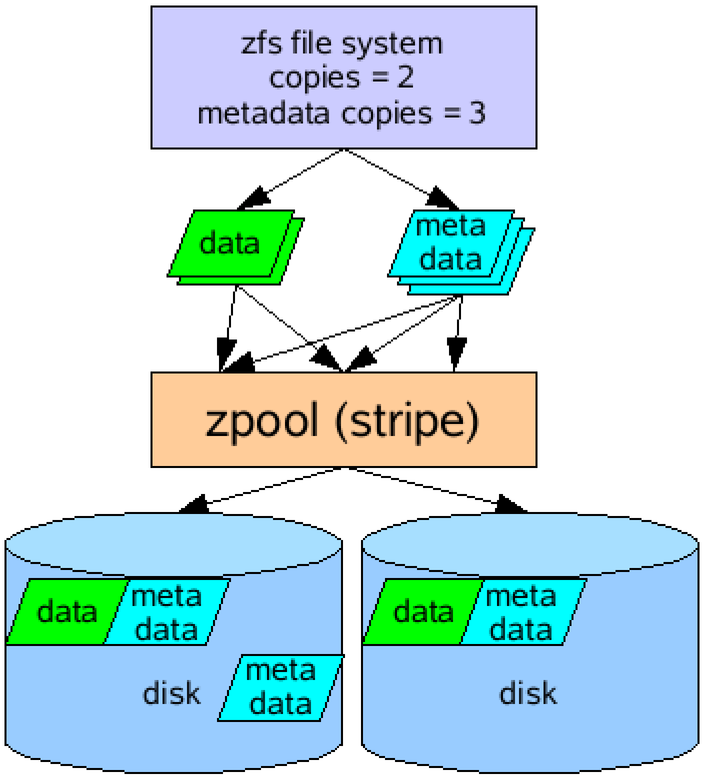
ZFS will spread the ditto blocks across the vdev or vdevs to provide spatial diversity. Bill Moore has previously blogged about this, or you can see it in the code for yourself. From a RAS perspective, this is a good thing. We want to reduce the possibility that a single failure, such as a drive head impact with media, could disturb both copies of our data. If we have multiple disks, ZFS will try to spread the copies across multiple disks. This is different than mirroring, in subtle ways. The actual placement is ultimately based upon available space. Let's look at some simplified examples. First, for the default file system configuration settings on a single disk.



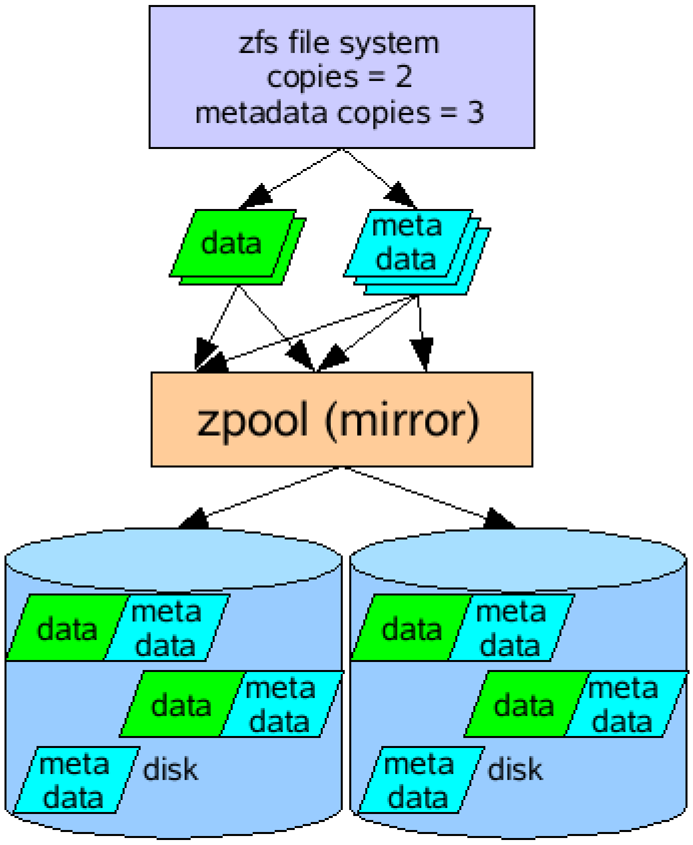
Note that there are two copies of the metadata, by default. If we have two or more copies of the data, the number of metadata copies is three.



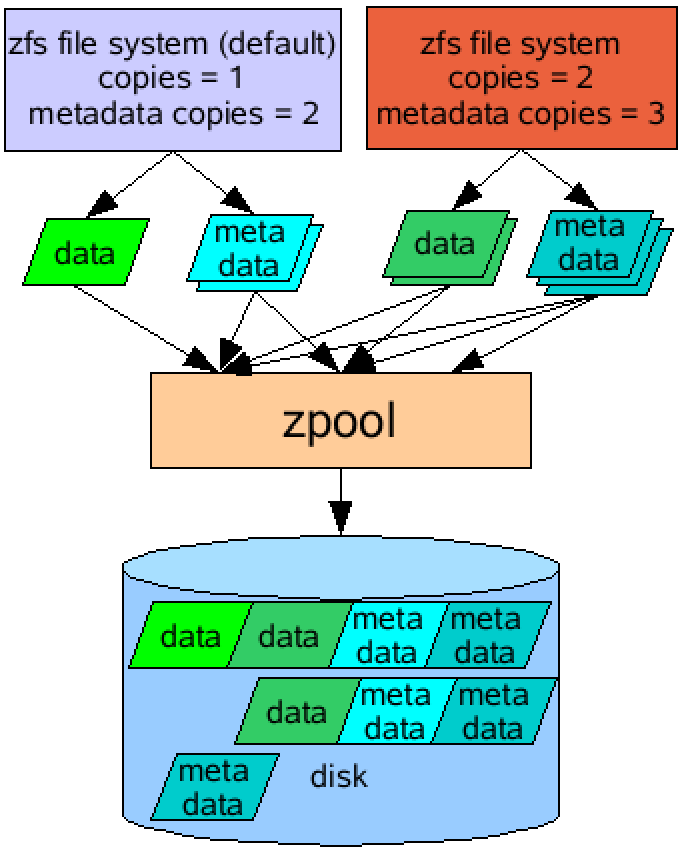
Suppose you have a 2-disk stripe. In that case, ZFS will try to spread the copies across the disks.



Since the copies are created above the zpool, a mirrored zpool will faithfully mirror the copies.



Since the copies policy is set at the file system level, not the zpool level, a single zpool may contain multiple file systems, each with different policies. In other words, you could have data which is not copied allocated along with data that is copied.



Using different policies for different file systems allows you to have different data protection policies, allows you to improve data protection, and offers many more permutations of configurations for you to weigh in your designs.

## RAS Modeling

It is obvious that increasing the number of data copies will effectively reduce the amount of available space accordingly. But how will this affect reliability? To answer that question we use the MTTDL[2] model I previously described, with the following changes:

First, we calculate the probability of unsuccessful reconstruction due to a UER for N disks of a given size (unit conversion omitted). The number of copies decreases this probability. This makes sense as we could use another copy of the data for reconstruction and to completely fail, we'd need to lose all copies:

$$P_{recon\_fail} = ((N-1) * size / UER)^{copies}$$

For single-disk failure protection:

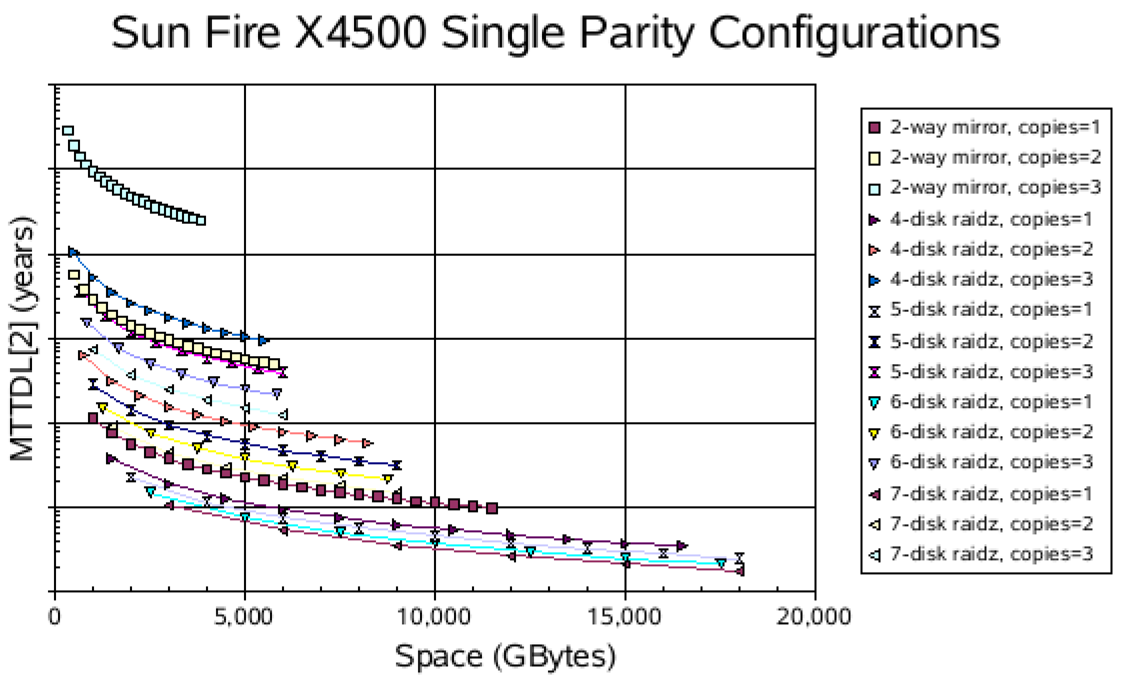
$$MTTDL[2] = MTBF / (N * P_{recon\_fail})$$

For double-disk failure protection:

$$MTTDL[2] = MTBF^2 / (N * (N-1) * MTTR * P_{recon\_fail})$$

Note that as the number of copies increases, $P_{recon\_fail}$ approaches zero quickly. This will increase the MTTDL. We want higher MTTDL, so this is a good thing.

OK, now that we can calculate available space and MTTDL, let's look at some configurations for 46 disks available on a Sun Fire X4500 (aka Thumper). We'll look at single parity schemes, to reduce the clutter, but double parity schemes will show the same, relative improvements.



You can see that we are trading off space for MTTDL. You can also see that for raidz zpools, having more disks in the sets reduces the MTTDL. It gets more interesting to see that the 2-way mirror with copies=2 is very similar in space and MTTDL to the 5-disk raidz with copies=3. Hmm. Also, the 2-way mirror with copies=1 is similar in MTTDL to the 7-disk raidz with copies=2, though the mirror configurations allow more space. This information may be useful as you make trade-offs. Since the copies parameter is set per file system, you can still set the data protection policy for important data separately from unimportant data. This might be a good idea for some situations where you might have permanent originals (eg. CDs, DVDs) and want to apply a different data protection policy.

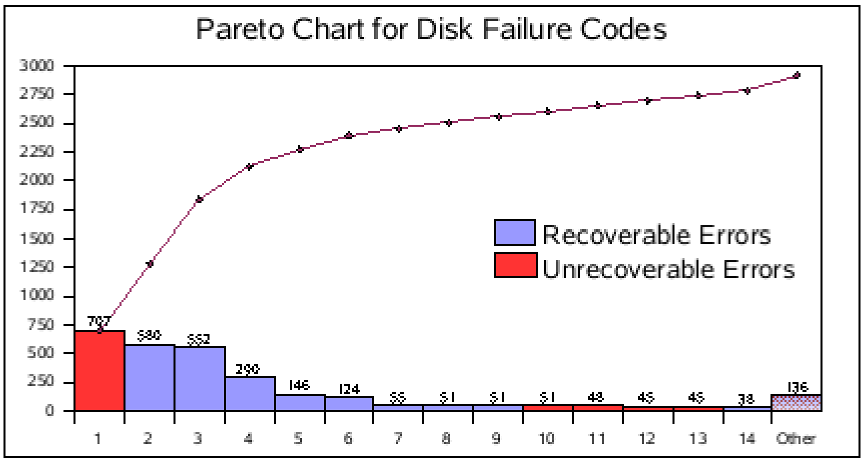
In the future, once we have a better feel for the real performance considerations, we'll be able to add a performance component into the analysis.

## Single Device Revisited

Now that we see how data protection is improved, let's revisit the single device case. I use the term device here because there is a significant change occurring in storage as we replace disk drives with solid state, non-volatile memory devices(eg. flash disks and future MRAM or PRAM devices) A large number of enterprise customers demand dual disk drives for mirroring root file systems in servers. However, there is also a growing demand for solid state boot devices, and we have some Sun servers with this option. Some believe that by 2009, the majority of laptops will also have solid state devices instead of disk drives. In the interim, there are also hybrid disk drives.

What affect will these devices have on data retention? We know that if the entire device completely fails, then the data is most likely unrecoverable. In real life, these devices can suffer many failures which result in data loss, but which are not complete device failures. For disks, we see the most common failure is an unrecoverable read where data is lost from one or more sector (bar 1 in the graph below). For flash memories, there is an endurance issue where repeated writes to a cell may reduce the probability of reading the data correctly. If you only have one copy of the data, then the data is lost, never to be read correctly again.

We captured disk error codes returned from a number of disk drives in the field. The Pareto chart below shows the relationship between the error codes. Bar 1 is the unrecoverable read which accounts for about 24% of the errors recorded. The violet bars show recoverable errors which did succeed. Examples of successfully recovered errors are: write error - recovered with block reallocation, read error - recovered by ECC using normal retries, etc. The recovered errors do not (immediately) indicate a data loss event, so they are largely transparent to applications. We worry more about the unrecoverable errors.



Approximately 1/3 of the errors were unrecoverable. If such an error occurs in ZFS metadata, then ZFS will try to read alternate metadata copy and repair the metadata. If the data has multiple copies, then it is likely that we will not lose any data. This is a more detailed view of the storage device because we are not treating all failures as a full device failure.

Both real and anecdotal evidence suggests that unrecoverable errors can occur while the device is still largely operational. ZFS has the ability to survive such errors without data loss. Very cool. Murphy's Law will ultimately catch up with you, though. In the case where ZFS cannot recover the data, ZFS will tell you which file is corrupted. You can then decide whether or not you should recover it from backups or source media.

## Another Single Device

Now that I've got you to think of the single device as a single device, I'd like to extend the thought to RAID arrays. There is much confusion amongst people about whether ZFS should or should not be used with RAID arrays. If you search, you'll find comments and recommendations both for and against using hardware RAID for ZFS. The main argument is centered around the ability of ZFS to correct errors. If you have a single device backed by a RAID array with some sort of data protection, then previous versions of ZFS could not recover data which was lost. Hold it right there, fella! Do I mean that RAID arrays and the channel from the array to main memory can have errors? Yes, of course! We have seen cases where errors were introduced somewhere along the path between disk media to main memory where data was lost or corrupted. Prior to ZFS, these were silent errors and blissfully ignored. With ZFS, the checksum now detects these errors and tries to recover. If you don't believe me, then watch the ZFS forum on opensolaris.org where we get reports like this about once a month or so. With ZFS copies, you can now recover from such errors without changing the RAID array configuration.

If ZFS can correct a data error, it will attempt to do so. You now have a the option to improve your data protection even when using a single RAID LUN. And this is the same mechanism we can use for a single disk or flash drive: data copies. You can implement the copies on a per-file system basis and thus have different data protection policies even though the data is physically stored on a RAID LUN in a hardware RAID array. I really hope we can put to rest the "ZFS prefers JBOD" argument and just concentrate our efforts on implementing the best data protection policies for the requirements.

ZFS with data copies is another tool in your toolbelt to improve your life, and the life of your data.