PETER BAER GALVIN

Pete's all things Sun: open source and free deduplication

Peter Baer Galvin is the chief technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."

pbg@cptech.com

IN THE PREVIOUS ISSUE OF *;LOGIN:*
I concluded the column with a quick introduction to the new deduplication feature of OpenSolaris. In this issue of "Pete's All Things Sun" I dive deeper into the details of how to gain access to that feature, how it works, and how to use it.

Overview

There is certainly a lot of industry-wide interest in deduplication. Companies like Data Domain (now purchased by EMC) were founded on the premise that companies are willing to add complexity (e.g., appliances) in exchange for reducing the number of blocks used to store their data. For instance, deduplication seems to be a perfect addition to a backup facility. Consider the power of a device that can be a backup target: as it receives blocks of a backup stream, it throws out blocks it has previously stored, replacing that block with a pointer to the duplicate block.

A quick logic exercise of analyzing the types of data that are being backed up should convince you that there is quite a lot of duplication in general (operating system images, binaries, and repeated backups of the same user and application data) and that there is quite a huge potential for savings of disk space via deduplication. Virtual machine images are very deduplicatable, for example, while user and application files are less so. But even when data is not intrinsically duplicated, from the time it is created through its life-cycle there may end up being many, many copies of it. Consider that deduplication can be used as part of a business continuance (disaster recovery) scenario, in which the deduplicated on-disk backup is replicated to a second site. Only sending a given block once can be quite a savings in network bandwidth, as well as the obvious savings of only needing enough storage at the second site to hold one copy of each block.

It's an established pattern in IT that a new feature implemented first by a startup as part of a separate product goes on to become a standard component of other companies' products. That pattern certainly seems true of Sun's implementation of deduplication as part of ZFS, included in an open source and free OpenSolaris distribution. The announcement of the integration of deduplication into ZFS and details of the implementation are available in a blog post by Jeff Bonwick, Sun's lead engineer on the project [1]. I would expect to see deduplication,

just like snapshots, thin provisioning, compression, replication, and myriad other features, becoming a component of many storage devices. Thus, even if you are not interested in ZFS deduplication, you may be interested in how deduplication works and what problems it can solve.

How It Works

Deduplication works by "thumb-printing," in which an entity (either a file or a block, typically) is checksummed, resulting in a hash value. Hashing is very effective, providing in almost all cases a unique value for a unique entity. If the values match, the entities are probably the same, and the new entity is not stored; rather, a pointer is stored pointing to the already stored matching entity.

The checksumming occurs at one of two possible times, depending on the implementation. The checksum analysis is overhead, taking up CPU cycles and I/O cycles as an inbound block is checksummed, and that result is checked against the checksums of all other blocks currently stored. For that reason and others, some implementations perform deduplication in post-processing. That is, they store all entities on write request, and then later compare the entities and remove duplicates. That is how NetApp deduplicates on their filers.

Alternately, the deduplication can occur at the time of writing, which is how Data Domain and ZFS deduplication works. This case takes a performance penalty at write time, but does not use up as much space as the post-processing method.

ZFS deduplication, as with other features of ZFS such as compression, only works on data written after the specific feature is enabled. If a lot of data already exists in a ZFS pool, there is no native way to have that deduplicated. Any new data will be deduplicated rather than written, but for the existing data to be deduplicated, that data would need to be copied to another pool (for example) or replicated to a ZFS file system with enabled deduplication.

In ZFS, once deduplication is enabled, the ZFS variable dedupratio shows how much effect deduplication is having on data in a ZFS pool. ZFS has file system checksumming enabled by default. Deduplication uses checksumming too, and enables a "stronger" checksum for the file system when enabled. (,"Stronger" means less likely to have a hash collision. See Bonwick's blog for more details.) By default it uses sha256. As mentioned above, hashing almost always results in matches only when the hashed entities exactly match. But there is a long way between "almost" and "always." Hashes can have collisions in which hashes of two non-matching entities have the same values. In those cases, there could be corruption as one entity is thrown out and replaced by a pointer to the other entity, even though the entities are not the same. See the discussion below about the ZFS deduplication "verify" option for details on how to solve this problem within ZFS.

Getting to the Right Bits

Deduplication was integrated into OpenSolaris build 128. That takes a little explanation. Solaris is Sun's current commercial operating system. OpenSolaris has two flavors—the semiannual supportable release and the frequently updated developer release. The current supportable release is called 2009.06 and is available for download [2]. Also at that location is the SXCE latest build. That distribution is more like Solaris 10—a big ol' DVD including all the bits of all the packages. OpenSolaris is the acknowledged future of

Solaris, including a new package manager (more like Linux) and a live-CD image that can be booted for exploration and installed as the core release. To that core more packages can be added via the package manager.

For this example I started by downloading the 2009.06 OpenSolaris distribution. I then clicked on the desktop install icon to install OpenSolaris to my hard drive (in this case inside VMware Fusion on Mac OS X, but it can be installed anywhere good OSes can live). My system is now rebooted into 2009.06. The good news is that 2009.06 is a valid release to run for production use. You can pay for support on it, and important security fixes and patches are made available to those with a support contract. The bad news is that deduplication is not available in that release. Rather, we need to point my installation of OpenSolaris at a package repository that contains the latest OpenSolaris developer release. Note that the developer release is not supported, and performing these next steps on OpenSolaris 2009.06 makes your system unsupported by Sun. But until an official OpenSolaris distribution ships that includes the deduplication code, this is the only way to get ZFS deduplication.

```
host1$ pfexec pkg set-publisher -O http://pkg.opensolaris.org/dev
opensolaris.org
Refreshing catalog
Refreshing catalog 1/1 opensolaris.org
Caching catalogs ...
```

Now we tell OpenSolaris to update itself, creating a new boot environment in which the current packages are replaced by any newer packages:

```
host1$ pfexec pkg image-update
Refreshing catalog
Refreshing catalog 1/1 opensolaris.org
Creating Plan . . .
DOWNLOAD        PKGS       FILES      XFER (MB)
entire          0/690      0/21250    0.0/449.4
SUNW1394        1/690      1/21250    0.0/449.4
. . .

A clone of opensolaris-1 exists and has been updated and activated. On the
next boot the Boot Environment opensolaris-2 will be mounted on /. Reboot
when ready to switch to this updated BE. You should review the release notes
posted at [3] before rebooting.
```

A few hundred megabytes of downloads later, OpenSolaris adds a new grub (on x86) boot entry as the default boot environment, pointing at the updated version. A reboot to that new environment brings up the latest OpenSolaris developer distribution, in this case build 129:

```
host1$ cat /etc/release
                    OpenSolaris Development snv_129 X86
        Copyright 2009 Sun Microsystems, Inc. All Rights Reserved.
                     Use is subject to license terms.
                       Assembled 04 December 2009
```

At this point, ZFS deduplication is available in this system.

```
host1$ zfs get dedup rpool
NAME      PROPERTY      VALUE     SOURCE
rpool     dedup         off       default
```

Testing Deduplication

Now that we have the deduplication bits of OpenSolaris, let's try using them:

```
host1$ pfexec zfs set dedup=on rpool
cannot set property for 'rpool':
pool and or dataset must be upgraded to set this property or value
```

Hmm, the on-disk ZFS format is from the 2009.06 release. We need to upgrade it to gain access to the deduplication feature.

```
host1$ zpool upgrade
This system is currently running ZFS pool version 22.

The following pools are out of date and can be upgraded. After being upgraded, these pools will no longer be accessible by older software versions.
VER  POOL
---  ------------
14   rpool

Use zpool upgrade -v for a list of available versions and their associated features.

host1$ zpool upgrade -v
This system is currently running ZFS pool version 22.

The following versions are supported:
VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
 4   zpool history
 5   Compression using the gzip algorithm
 6   bootfs pool property
 7   Separate intent log devices
 8   Delegated administration
 9   refquota and refreservation properties
10   Cache devices
11   Improved scrub performance
12   Snapshot properties
13   snapused property
14   passthrough-x aclinherit
15   user/group space accounting
16   stmf property support
17   Triple-parity RAID-Z
18   Snapshot user holds
19   Log device removal
20   Compression using zle (zero-length encoding)
21   Deduplication
22   Received properties

For more information on a particular version, including supported releases, see http://www.opensolaris.org/os/community/zfs/version/N, where N is the version number.

host1$ pfexec zpool upgrade -a
This system is currently running ZFS pool version 22.

Successfully upgraded 'rpool'
```

Now we are ready to start using deduplication.

```
host1$ zfs get dedup rpool
NAME   PROPERTY   VALUE      SOURCE
rpool  dedup      off        default
host1$ pfexec zfs set dedup=on rpool
host1$ zfs get dedup rpool
NAME   PROPERTY   VALUE      SOURCE
rpool  dedup      on         local
host1$ zpool list rpool
NAME   SIZE   ALLOC   FREE   CAP  DEDUP  HEALTH  ALTROOT
rpool  19.9G  10.7G   9.19G  53%  1.00x   ONLINE  -
```

To test out the space savings of deduplication, let's start with a fresh zpool. I added another virtual disk to my OpenSolaris virtual machine. Now let's make a pool, turn on deduplication, copy the same file there multiple times, and observe the result:

```
host1$ pfexec zpool list
NAME    SIZE   ALLOC  FREE   CAP  DEDUP  HEALTH  ALTROOT
rpool   19.9G  10.8G  9.08G  54%  1.05x  ONLINE  -
host1$ pfexec zpool create  test c7d1
host1$ zpool list
NAME    SIZE   ALLOC  FREE   CAP  DEDUP  HEALTH  ALTROOT
rpool   19.9G  10.8G  9.08G  54%  1.05x  ONLINE  -
test    19.9G  95.5K  19.9G  0%   1.00x  ONLINE  -
host1$ zfs get dedup test
NAME   PROPERTY  VALUE  SOURCE
test   dedup     off    default
host1$ pfexec zfs set dedup=on test
host1$ zfs get dedup test
NAME   PROPERTY  VALUE  SOURCE
test   dedup     on     local
host1$ df -kh /test
Filesystem  Size  Used  Avail  Use%  Mounted on
test        20G   21K   20G    1%    /test
host1$ ls -l /kernel/genunix
-rwxr-xr-x 1 root sys 3358912 2009-12-18 14:37 /kernel/genunix
host1$ pfexec cp /kernel/genunix /test/file1
host1$ pfexec cp /kernel/genunix /test/file2
host1$ pfexec cp /kernel/genunix /test/file3
host1$ pfexec cp /kernel/genunix /test/file4
host1$ pfexec cp /kernel/genunix /test/file5
host1$ df -kh /test
Filesystem  Size  Used  Avail  Use%  Mounted on
test        20G   14M   20G    1%    /test
host1$ zpool list test
NAME   SIZE   ALLOC   FREE   CAP  DEDUP  HEALTH  ALTROOT
test   19.9G  3.43M   19.9G  0%   4.00x  ONLINE  -
```

So, a file approximately 3MB in size and copied five times to a deduplicated ZFS pool seemingly takes up 14MB but in reality only uses 3.43MB (this space use must include the file, but also deduplication data structures and other metadata).

Also, according to PSARC (architecture plan) 557, deduplication also applies to replication, so in essence a deduplicated stream is used when replicating data [4]. Let's take a look. Fortunately, I have another (virtual) OpenSolaris system to use as a target of the replication (which we will call host2):

```
host2$ pfexec zpool create test c7d1
host2$ pfexec zfs set dedup=on test
host2$ zfs list test
NAME   USED   AVAIL   REFER   MOUNTPOINT
test   73.5K  19.6G   21K     /test
```

Now I take a snapshot on host1 (as that is the entity that can be replicated) and send it to host2:

```
host1$ pfexec zfs snapshot test@dedup1
host1$ pfexec zfs send -D test@dedup1 | ssh host2 pfexec /usr/sbin/zfs
receive -v test/backup@dedup1
Password:
receiving full stream of test@dedup1 into test/backup@dedup1
received 3.30MB stream in 1 seconds (3.30MB/sec)
```

On the receiving end, we find:

```
host2$ zfs list test
NAME   USED   AVAIL   REFER   MOUNTPOINT
test   16.4M  19.6G   21K     /test
host2$ zpool list test
NAME   SIZE   ALLOC   FREE   CAP   DEDUP   HEALTH   ALTROOT
test   19.9G  3.48M   19.9G  0%    5.00x   ONLINE   -
```

Sure enough, ~3MB were sent as part of the replication, and although the receiving system thinks it has ~16MB of date, it only has ~3.4MB.

Unfortunately, the current zfs send -D functionality is only a subset of what is really needed. With -D, within that send, a given block is only sent once (and thus deduplicated). However, if additional duplicate blocks are written, executing the same zfs send -D again would send the same set of blocks again. There is no knowledge by ZFS of whether a block already exists at the destination of the send. If there was such knowledge, then zfs send would only transmit a given block once to a given target. In that case ZFS could become an even better replacement for backup tape: a ZFS system in production replicating to a ZFS system at a DR site, only sending blocks that the DR site has not seen before. Hopefully, such functionality is in the ZFS development pipeline.

Let's try that final experiment. First I'll create more copies of the file, then create another snapshot and send it to host2:

```
host1$ pfexec cp /kernel/genunix /test/file6
host1$ pfexec cp /kernel/genunix /test/file7
host1$ pfexec cp /kernel/genunix /test/file8
host1$ pfexec cp /kernel/genunix /test/file9
host1$ df -kh /test
Filesystem   Size   Used   Avail   Use%   Mounted on
test         20G    30M    20G     1%     /test
host1$ zpool list test
NAME   SIZE   ALLOC   FREE   CAP   DEDUP   HEALTH   ALTROOT
test   19.9G  3.45M   19.9G  0%    9.00x   ONLINE   -
host1$ pfexec zfs snapshot test@dedup2
host1$ pfexec zfs send -D test@dedup2 | ssh host2 pfexec /usr/sbin/zfs
receive -v test/backup2@dedup2
Password:
receiving full stream of test@dedup2 into test/backup2@dedup2
received 3.34MB stream in 1 seconds (3.34MB/sec)
```

Note that, even though host2 already had all the blocks it needed, one copy of the file was sent again because the sending host has no knowledge of what the receiving host already has stored. On the receiving side:

```
host2$ df -kh /test/backup
Filesystem      Size    Used    Avail   Use%    Mounted on
test/backup     20G     17M     20G     1%      /test/backup
host2$ df -kh /test/backup2
Filesystem      Size    Used    Avail   Use%    Mounted on
test/backup2    20G     30M     20G     1%      /test/backup2
host2$ zpool list test
NAME    SIZE    ALLOC   FREE    CAP     DEDUP   HEALTH  ALTROOT
test    19.9G   3.46M   19.9G   0%      14.00x  ONLINE  -
```

Even though host2 was sent the extraneous copy of the file, it discarded it, leaving it to store only one copy of the file.

Additional Analysis

No hash algorithm is perfect, in that two blocks only have the same hash if they are exactly the same. There is a very small chance that two blocks could have matching hashes even if they are not identical. By default ZFS trusts the hash values and will declare a block to be a duplicate if the hash matches. To increase safety you can set ZFS to do a byte-by-byte comparison of two blocks if the hashes match, to ensure that the blocks are identical before declaring them to be duplicates.

```
$ pfexec zfs set dedup=verify rpool
```

Of course this will negatively affect performance, using more CPU time per duplicate block.

On another performance note, the jury is still out on the performance impact of deduplication in ZFS. Theoretically, the increased overhead of checking for an existing matching hash whenever a block is about to be written may be counterbalanced by the saved write I/Os when there is a duplicate block that need not be written. But, in fact, it is too early to tell what the net result will be.

Deduplication can cause a bit of confusion about exactly what is using how much space. For example, the results of `du` can be grossly wrong if the data in the directories has been well deduplicated. Only `zpool list` is dedupe-aware at this point. `df` and even other ZFS commands are not aware of deduplication and will not provide use information taking deduplication into account.

Conclusion

As it stands, ZFS deduplication is a powerful new feature. Once integrated into production-ready operating system releases and appliances, it could provide a breakthrough in low-cost data reduction and management. I plan to track that progress here, so stay tuned. For more details on the current state of ZFS deduplication, including bugs, features, and performance, please see the ZFS wiki [5].

Tidbits

As of this writing, Oracle has just acquired Sun Microsystems. Likely this will mean long-term changes with respect to which of Sun's products come

to market and how Sun customers continue on as Oracle/Sun customers. At first blush (and first announcement), however, there seem to be very few changes for Sun customers. There were no massive layoff announcements (as some analysts had predicted), and so far, very little change in product direction. SPARC and x86 servers, storage arrays, Java, and Solaris all appear to have bright futures, as Oracle not only continues those products but increases the R&D budgets for most of them. At least in the immediate shadow of the merger, all seems to be well in Sun's product portfolio and direction. For more details on Sun under Oracle, including replays of the Oracle presentations about the purchase, have a look at http://www.oracle.com/us/sun/.

REFERENCES

[1] http://blogs.sun.com/bonwick/entry/zfs_dedup.

[2] http://www.opensolaris.com/get/index.jsp.

[3] http://opensolaris.org/os/project/indiana/resources/relnotes/200906/x86/.

[4] http://arc.opensolaris.org/caselog/PSARC/2009/557/20091013_lori.alt.

[5] http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup.