Creating a Consistent Image-Based Backup On Linux Using I/O Tracing and Short-Term Filesystem Freezing

Nick Garvey
ngarvey@dattobackup.com

December 17, 2013

1 Problem

When performing an image-based backup, it isn't always possible to ensure that the data on disk is in a consistent state. While a system is running, information is changing on disk, which makes getting a consistent point-in-time backup challenging. There are two current solutions to this.

The first solution is taking a copy-on-write snapshot of the device being backed up. This allows the snapshot to be the target of the backup , which is guaranteed to be point-in-time by the snapshot software. However, taking a snapshot requires hardware support or software support through LVM. Snapshots can currently only be taken on a Linux system which was setup to do so during installation; snapshots can't be taken with a typical configuration.

An alternative solution is freezing the device for the duration of the backup process. There is no risk of data changing during the backup if the device is frozen. Freezing the entire device during a backup has an obvious problem: data that is changing needs to be stored somewhere. This is often known as redirect-on-write. Sending it over the network or to another disk would solve this, but support for this isn't standard and again must be configured before installation.

2 Technique

2.1 Overview

The core idea of the new method is to keep track of which blocks are out-of-sync with the backup destination and make passes over the device to synchronize those blocks. An interval set is the perfect data structure for this. If blocks 7-10 are out-of-sync, then we will store the endpoints $[7, 10]$ instead of individually tracking $\{7, 8, 9, 10\}$.

When performing the original full backup, we need to determine what blocks are in-use on disk. By parsing the filesystem metadata structures, it is possible to determine what parts of the disk are in-use and initialize the out-of-sync interval set with the intervals corresponding to the in-use blocks.

When performing the actual backup, the live system will likely perform modifications of the disk. In order to keep track of these modifications, block tracking is used to determine which blocks are modified. These modifications are added to the out-of-sync interval set during the backup, so synchronization logic will pick up on the changes.

The method also makes incremental backups very straightforward. By tracking the changes that occur after a backup, it is possible to know exactly what needs to be synchronized during an incremental.

In order to make sure that no writes occur at the end of the backup process, the device can be frozen for about a second. This will ensure that any cached data is written to the disk and

is detected by the block tracking software. The freeze process also ensures the data will be in a consistent state on disk. This allows for use of the image without needing to correct any data that was half-written when the backup completed.

2.2 Block Change Tracking

As disk I/O tracking is built into the kernel of modern versions of Linux (2.6.17+), it is possible to track block changes without special hardware or pre-installation configuration.

In Linux, this disk I/O tracking facility is called blktrace. Whenever any type of I/O is sent to a device, the I/O is recorded and is available to userspace applications.

2.3 I/O Throttling

The iterative copy process will never complete if the amount of out-of-sync data increases each iteration. As such, the rate at which the data is copied must, on average, be greater than the rate at which data is changing. In general this will be the case. Typical systems tend to write in bursts and then stop, giving the backup process a chance to catch up.

However, in the non-typical case we need to guarantee that writes will slow down in a reasonable time frame.

This can be enforced by making sure the copy I/O is faster than the change I/O. If the copy speed can't be increased, then the write speed must be reduced. This can be accomplished by software or hardware throttling during the backup. In Linux, this can be accomplished using cgroups blkio throttling.

While this need for throttling might seem like a limitation of the method, it is an inherent problem with any sort of backup procedure. Inability to copy data faster than it is changing makes a point-in-time image impossible, regardless of method.

2.4 Short-Term Device Freezing

By using the FIFREEZE ioctl in Linux, the filesystem on a device can have all writes frozen until the FITHAW ioctl is given. Freezing the filesystem does three things: it suspends all writes, sends any data in the write cache to disk, and marks the filesystem as clean. This is perfect for our purposes, as this means the filesystem has a reduced chance of being corrupted. Additionally, as the iterative backup process completes, freezing the filesystem removes any concern of a race condition that might cause a write to be missed at the end of the procedure.

Freezing only at the end of the backup has minimal impact on the running system. The only noticeable effect will be all disk I/O blocking for about a second.

3 Procedure

1. Start tracing to add disk writes to the out-of-sync structure
2. Read the filesystem metadata to determine which blocks are in-use
3. Add the in-use blocks to the out-of-sync data structure
4. Begin copying blocks that are marked in the out-of-sync structure
5. Check the number of out-of-sync blocks after a period, if it is increasing between iterations then throttle the device

6. Repeat the above two steps until a small number of blocks are out-of-sync
7. Freeze the device
8. Copy the last changed blocks
9. Unfreeze the device
10. Unthrottle the device if it is throttled