Technical Disclosure Commons

Defensive Publications Series

January 10, 2019

Reducing Memory Footprint for In-Memory Root Filesystem (TMPFS)

Erik Jacobson
Hewlett Packard Enterprise

Paul Schliep
Hewlett Packard Enterprise

Christopher Holmes
Hewlett Packard Enterprise

Reducing Memory Footprint for In-Memory Root Filesystem (TMPFS)

Abstract:

This disclosure relates to the field of high-performance computing (HPC) clusters and techniques cluster managers or system administrators can use to avoid expensive and complicated storage for compute resources. By making in-memory root filesystems more desirable to use, it is easier to reduce hardware complexity and cost in the system.

Description:

This disclosure describes a means to reduce the compute node memory usage when it has its entire operating system in system memory. In large compute clusters, one means to reduce cost and complexity in the system is to avoid the installation of hard drives or solid-state disks (SSDs). In these systems, the compute resources in the cluster need to have their operating system located in memory instead of on a physical storage device. The main source of resistance to in-memory root filesystems is the reduction of available memory for running jobs. This disclosure includes ideas to reduce that burden.

Cluster managers offer various methods to compute nodes for their root filesystems. Methods include simply installing to their system disks, using a network filesystem (NFS) for the root filesystem, or using an in-memory filesystem where compute node is installed similarly to a node with a system disk, but to memory instead. The latter method is often called a TMPFS root filesystem. This is because in Linux, the implementation of in-memory filesystems available in the kernel is referred to as a temporary in-memory filesystem and is more commonly described by its acronym 'TMPS'.

Some reasons customers prefer TMPFS root filesystems include:

- There are no restrictions to writing as there can be in some network filesystem (NFS) solutions
- There is no need for the cost and maintenance of disk drives or solid-state storage
- The root filesystem, being in-memory, is extremely fast

Some reasons customers prefer not to use TMPFS root filesystems include:

- The root filesystem uses up critical system memory needed to run their jobs and programs on the compute nodes
- There is no retained state, so if a file is saved in the root filesystem, it is lost at client node reboot
    - Various methods exist to mitigate this, which we won't go in to here, but it's still a core problem

The ideas presented here address the first bullet – system memory usage.

For the cluster manager, managing NFS roots is complicated. The complications multiply if the customer wants the head node providing services to the client nodes to be highly available (HA). There is a lot of management infrastructure in place to properly manage NFS roots for customers that is prone to bugs not just by the cluster manager itself, but by the kernel and Linux distro.

Anything that encourages customers to choose a different diskless option both benefits the customer and allows the system vendor to simplify the solutions offered.

General Content of an OS Root Lends Itself to Compression

Root filesystems are generally made up of text files, executables, and libraries. These things are easily compressed. One way to view this is by looking at a typical RPM-based Linux Distribution. If you add up the space used by all the RPMs (software packages), which are compressed, and compare it to the space used on the system root after installation, you can see a 2 or 3 times expansion in space used by the system. In experiments run while preparing this idea, it was found that the commonly used BTRFS filesystem with compression enabled ended up using 50% less space using the default compression mode than no compression at all for a typical root filesystem tree.

Implementation of a Compressed Root on TMPFS

Some cluster management software supports provisioning TMPFS compute nodes by using a multicast stream (one-to-many provisioning). The compute node first brings up an installer environment. The compute node then requests the image, joins a transfer group, and then the head node sends out the root image as a stream using multicast to all compute nodes in the transfer group at the same time. This stream is often a tar file (archive) stream or encrypted tar stream. Other methods include using BitTorrent for the root filesystem tar file so that all compute nodes help their peers install as they go. The important part is that they use a method to avoid network saturation during installation. While scalable methods are discussed here, this idea can be used with any transport method of the root over the network including the rsync protocol.

In a typical cluster environment with TMPFS root support, the install environment on the compute node handles creating the TMPFS mount point for the TMPFS filesystem and the tar stream representing the root filesystem is transferred on to that TMPFS mount. Later, the install environment of the node switches to the just-installed root filesystem to complete start up. This step is often referred to as ‘switchroot’, the process of switching from the initial boot environment root to the "real root" to complete Linux start up.

The new idea layers some additional components on top of the TMPFS Mount point (See Figure 1). This includes an image file that is used to house a root filesystem (for example, BTRFS).

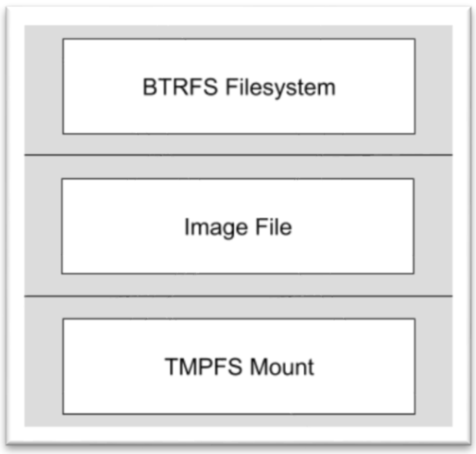

Figure 1

The new process is as follows (See Figure 2):

- The install environment creates a TMPFS mount point like before on the compute node.
- The install environment requests (or gets from the first bytes of the file) the estimated size of the root image being transferred
- The install environment then creates an empty file large enough to hold the root image
- The install environment puts a filesystem on that file – a filesystem that supports transparent compression. Examples in Linux include:
    - BTRFS
    - Zfs
- The filesystem is mounted with compression enabled, and the image is stored there (via multicast or BitTorrent transfer or other means)
- With the root filesystem now represented in the filesystem (BTRFS or otherwise) on an image file on top of the TMPFS mount, the used space is computed
- Using the filesystem's "shrink" feature, the filesystem space is reduced to just what is needed for the root environment plus some extra space for normal operations
    - In BTRFS, this would be 'btrfs filesystem resize'
- The file is then truncated to that size, this frees up the system memory for the in-memory filesystem to match. This means memory is freed up for jobs on the compute node.
    - The bytes in use can be used to determine a safe place to truncate the image file that won't lose data
- Next, the TMPFS mount point (representing the image above) is left mounted. Switchroot runs (changes "/" to be the filesystem we just made) and the system starts normally running the Linux init start up
- An optional parameter can be provided by the customer to make the image larger than just fitting the image if the customer wants additional space
- An optional daemon can be provided that, if enabled, can automatically grow the image file as needed and if desired.

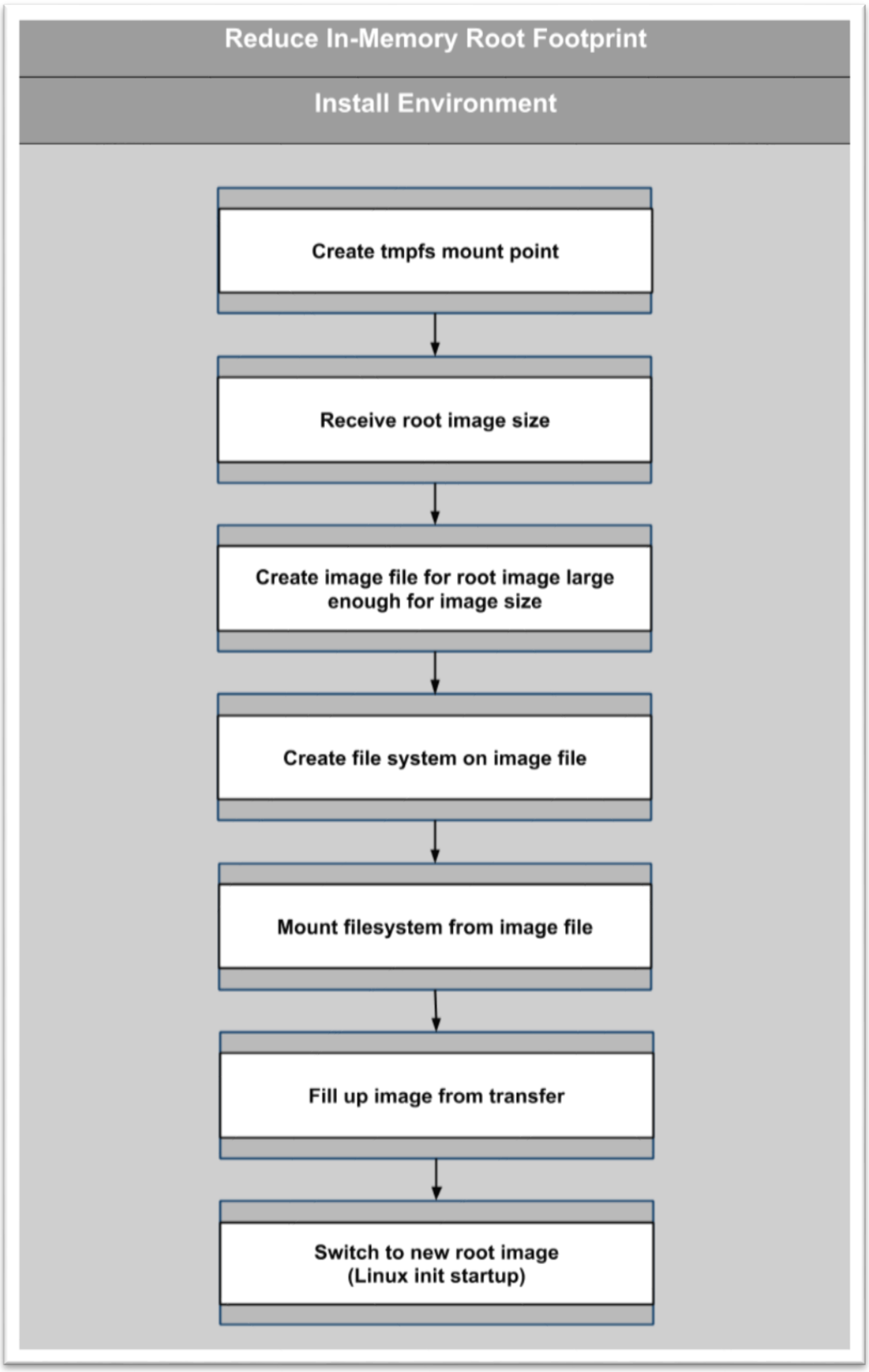
Reduce In-Memory Root Footprint
Install Environment
Create tmpfs mount point
Receive root image size
Create image file for root image large enough for image size
Create file system on image file
Mount filesystem from image file
Fill up image from transfer
Switch to new root image (Linux init startup)

Figure 2

Automated Growth

Modern filesystems like Zfs and BTRFS support growing the filesystem as well as shrinking. This means, the space used by the filesystem on the device (in our case, an image file living on TMPFS) can be grown.

A daemon is presented that, when enabled, continually monitors the free space of the root file system. When space starts to get low, it can quickly extend the TMPFS-hosted file used for the filesystem image, and then grow the filesystem to the size of the file.

- Append data to the end of the image file that lives on TMPFS
- Then grow or resize the filesystem in the image
- For BTRFs, this could be done with 'btrfs filesystem resize max'

Using this daemon, the root filesystem is able to grow without running out of space, but in an efficient way as to keep system memory usage as low as possible. In the end, it provides a similar feel to normal TMPFS root implementation where the filesystem grows to consume memory up to the upper limit given to the Linux kernel – except that, due to the benefit of compression, growth is minimized.

Expanding (or Shrinking) the Root Filesystem after Boot

Some customers may wish to extend (or later reduce) the amount of system memory dedicated to their root filesystem after the node has booted. Perhaps they are running jobs that need to be output stored somewhere.

A tool is provided by this idea that:

- Takes the new filesystem size as input
- If the size is larger than current, then the tool:
    - Enlarges the filesystem image file on the original TMPFS filesystem by the necessary amount
        - Appends NULLs to the end of the file to match the new desired size
    - Uses the filesystem tools for the filesystem on the image (e.g. Zfs, BTRFS) to grow the filesystem residing on TMPFS
        - In BTRFS, this could be done, as mentioned above, with 'btrfs filesystem resize max'
    - The filesystem is now grown to the size of the file we extended
- If the size is smaller than current, then the tool:
    - Shrinks the filesystem size using the filesystem tool's shrink mechanism
        - In BTRFS, this would be 'btrfs filesystem resize'
    - Computes the new safe end to the file that contains the filesystem
    - Truncates the file, thus releasing this amount of system memory for normal use

End Result

- A TMPFS root using significantly less memory is supplied, leaving valuable memory for customer applications and jobs
- Because the filesystem is still sitting in system memory, even with the added overhead, it is extremely fast
- The customer has tools supplied by this implementation to grow the root filesystem after the fact without rebooting/re-installing.
- Prototyping has shown for a typical root filesystem using default BTRFS compression, space used by root is halved (default zlib compression)
    - Compression had poorer ratios with LZO compression
- Newer versions of BTRFS offer a means to change the compression level but the prototyping for this proposal was left at default settings. Larger compression ratios are likely available with newer Linux distributions and BTRFS versions.
- Prototyping of a compute node using this solution booted fast and was stable. It had a faster feel than existing NFS root solutions in prototyping.

A Second Approach with FuseCompress

In Linux, there is an infrastructure for implementing user space filesystems called 'FUSE'. Using standard interfaces with the kernel, modules using FUSE can implement interesting behaviours such as a union of two directories being used as a file system.

In researching what to do about a compressed TMPFS root option, the FuseCompress project was investigated. This would be a way to implement a compressed filesystem with less complication than we prototyped above.

However, this project hasn't had meaningful contributions in many years and appears to be unstable. It also isn't clear it would work well as a root filesystem. While not prototyped, here is how such a system could be used:

The FuseCompress project could be invested in so that it is stabilized and made reliable for a root filesystem. The appropriate modules and content could be supplied with the installer environment and the root images. This would allow the installer environment to:

- Create a FuseCompress data directory where compressed versions of the files are stored
- Mount the FuseCompress filesystem to a specific location
- Populate the FuseCompress filesystem at the mount point
- Provide switchroot operation to the target mount point to switch to Linux init startup
    - A modified switchroot that preserves the FuseCompress data directory may be necessary so that it doesn't purged during the switchroot operation

The benefit of this method over the others is that it is more natural and there is less maintenance.

FuseCompress project: https://github.com/tex/fusecompress