LA-UR-15-23758
Approved for public release; distribution is unlimited.

Title: Lustre on diskless servers using ZFS pools as targets

Author(s): Croonenberg, Ronald Johannes

Intended for: white paper describing issues with running Lustre diskless

Issued: 2015-05-19

Lustre on diskless servers using ZFS pools as targets

Ron Croonenberg
Los Alamos National Laboratory
Los Alamos, NM 87544
Email: rocr@lanl.gov

Introduction

High Performance Computing systems create an increasing demand for file systems that can sustain a high throughput, are robust, can be expanded without administration and maintenance becoming too complex.

Basically a Lustre file system is a cluster of servers with storage targets attached to them. Each and every server has a Linux operating system, network and software configurations that need patches, updates, additional software installs and anything else a regular Linux server would require.

In High Performance Computing (HPC), with clusters that have hundreds or thousands of compute nodes, it is common to boot these compute nodes via a network which makes administering, updating, patching and repairing a cluster a lot more manageable while nodes within a cluster are configured and installed uniformly.

This approach can also be used for Lustre file systems. A Lustre file system, can be configured so that it is no different in structure than a compute cluster, it can boot from a network, load its Operating System, ram disk and mount file systems it needs. This eliminates the need for local OS drives which eliminates local administration, updates, patching and increases security and reliability similar to a compute cluster with diskless nodes being better manageable.

Background

Parallel file systems are getting larger. Building, administering and expanding file systems and upgrading its servers is a challenge that is also getting bigger. It is a lot more efficient, secure and productive to manage a parallel file system as a compute cluster with data storage targets attached to its servers but without operating system drives.

Managing only two or three net boot images, one for an OSS, one for an MDS and possibly one for an LNET, is a lot more efficient than managing each and every node in the file system cluster individually. This can be done because the servers in the file system cluster are very similar and differ little in their configurations. Some software allows multiple hosts to be configured in a single configuration file for example ethcfg a LANL developed utility for network device configurations. The Utility ethcfg configures both ether net and IB interfaces, for clusters. The utility lists node name, device and ip address and a few other optional parameters, in it's configuration file, for each device that needs an ip address in the file system cluster. There is one ethcfg configuration file for all the nodes, so it can simply be put on the OSS and MDS boot images. Other software can use just one configuration file by simply making sure cabling is connected identical while configuration file of other software can easily be generated at boot time of a particular server depending on it's host name or IP address for example.

When using ZFS as the Lustre back end file system we can take advantage of utilizing options like data compression, meta-data check sums or even snap shots that ZFS offers over ldiskfs. The impact on the compute performance, using compression with ZFS, is nihil and might actually increase I/O performance because likely, depending on the uncompressed data, less data will be written to the hardware than without compression.

Both the MGT and MDT on the MDS use ldiskfs as their Lustre back end file system type because there is no substantial gain in using ZFS and we actually experienced ZFS slowing down meta-data I/O performance significantly regardless of using compression or not.

The OSS and MDS nodes boot from the network, using net boot provisioning software, for example Perceus, and the node's boot images are created using configuration management software, for example cfengine. Network booting file system cluster nodes is a lot more efficient than booting from local drives, for example an OS upgrade or software update only requires a restart of the system with a new boot image.

Net booting also reduces down time in case, for example, a server needs to be replaced. Provisioning software typically needs some configuration adjustment to bring that new server up again without a complete server build and configuration. Also, the absence of hard drives in the servers means a lower probability of system failure.

Problems

The MGT and MDT are both located on a single raid array. The MGT is very small, a 500MB partition, while the MDT partition occupies the rest of the device. Configuring Lustre to work with partitions on a device presented with IBSRP seems to causes mkfs.lustre to fail. We observed the same behavior with a regular drive being partitioned. This particular problem was resolved, but not further investigated, by creating logical volumes of these partitions. For these Logical Volumes however, since the Lustre Meta Data Servers (MDS) are diskless, the Logical Volume Group information might need to be stored elsewhere.

The majority of the problem is that system configurations can not be stored locally because of the lack of drives, storing the information on nfs mounts could work but with a large number of file system servers that would be impractical.

Most of the differences between the nodes are network device configurations, target/storage configurations and of course software configurations. It is advantageous to make sure that all port connections on the nodes are uniform so that devices listed on one node have the same function/connection as on another but functionally similar node.

Assigning OSTs to OSSs can be done in different ways, either by listing them, for example in heartbeat, or by finding an algorithm where each node can figure out which targets are assigned to it, for example if the Lustre “index of an OST” modulo “the number of OSSs” is the same as the node number/index then that OST belongs to that node.

For practical as well as security reasons we only want one boot image for the OSSs and also just one for the MDSs. However the configurations differ for every node and storing configurations for each and every node is undesirable.

Our system uses fine grain routing (FGR) for routes to the different groups of LNETs. That means different configuration files are needed in 'modprobe' for different for OSSs without creating a different boot image for every OSS.

Solutions

The MDT and MGT are located on the same device, a raid array, LUN, on DDN hardware. In order for Lustre to see two different devices, GNU parted was used to divide the raid array (DDN MDT) into two partitions, a small one, 500MB, for the Lustre MGT and the rest of the space on the raid array for the Lustre MDT. We used parted because of size limitations with fdisk. The Linux Logical Volume Management (LVM) was used to configure the partitions into logical volumes so that Lustre recognizes them as different devices instead of different partitions of a single device. Initially a separate Logical Volume Group (LVG) was created for the two logical volumes but the group was not needed for correctly running Lustre and therefore not used in later configurations. The logical volumes however were needed to install the MGT and MDT and are part of the default group.

Other devices, like a dual port IB card, that is not used as an Ethernet device, can be configured for IBSRP/OpenSM using the same configuration file that contains an options for telling IBSRP/openSM to use specific ports on specific cards, in this case port 1 and port 2 both on card 0. This will, work instead of mentioning specific port UIDs, if all physical connections are uniform. In fact mentioning individual ports or using the setting SRP_DEVICES="auto" instead of SRP_DEVICES="mlx4_0:1 mlx4_0:2" in ibsrp.conf won't work.

The data targets for the OSSs physically are hardware raid arrays, 8 drives plus two parity drives. We chose hardware raids, for better performance with this particular hardware, instead of ZFS raids. The ZFS pools are configured to use one ZFS pool consisting of one device, a raid array. The raid arrays are presented by the DDN SFAs as LUNS, devices, to the OSSs using the IBSRP protocol.

The OST name depends on the SFA index (index of an SFA pair) and the index of the raid array in that pair. For example; the first target,SFA pair, first OST would be named OST00-00, the second SFA pair first OST would be OST01-00. The index, as used in Lustre depends on the SFA pair and OST number in that pair. Lustre.index= (100 * 'SFA index') + raid index (raid index within the SFA pair). For example the Lustre.index for the 3rd drive in the 2nd array would be 100*1 + 2, or 102.
Distribution of the OSTs among the OSSs was initially done with an algorithm. Dividing the OSTs up over the OSSs was done in a “round robin” like way. Assuming there are N OSSs (N > 0) OST00-XX would be mounted by OSS-(XX Mod N). With 8 OSSs OSS-03 would mount OST00-03, OST00-11, OST00-19 etc. When implementing heartbeat, which starts Lustre and assigns the OSSs resources the drives were assigned to an OSS using a heartbeat configuration file.

Because of a bug in Lustre, creating the file system on ZFS, additional parameters in the mkfs.lustre command need to be written to the ZFS pool separately.

An OST can be mounted after the ZFS pool it is located on, has been successfully imported. Because multiple paths to a target can exist, importing ZFS pools should be done so that ZFS only detects every

pool once. ZFS pools being mounted more than once or the same device being considered part of a pool multiple times very likely causes meta-data corruption on the pool. An import with default settings very likely leads to this meta data corruption, resulting in a corrupted OST and data is lost.
To prevent this, ZFS pools are imported from the same location as was used to create them, in our case /dev/mapper.

It seems counter intuitive to only use one device for a pool, but the device itself is a raid array and there is no need to use the ZFS raid capability. ZFS is used for it's compression options and also features like check sums on data etc. The overhead on the OSSs is nihil and the LZ4 data compression setting in ZFS, depending on the data, is expected to be 30%-50%.

Our system uses Fine Grain Routing (FGR), this complicates network booting the nodes of the Lustre file system with respect to routing. There are several FGR groups defined for OSS/LNET connections, and each of these groups requires a unique 'modprobe' configuration file. The distribution of the OSSs and the choice of fail over partners needs to be designed with the FGR configuration in mind. The fail over OSS needs to be in a different FGR group than the primary OSS. The different routes are inserted into the Lustre configuration files using scripts.

Conclusion

Diskless booting nodes in a Lustre file system works very well. Kernel upgrades, patches etc. can be applied without the file system being unavailable for longer than it takes to reboot all the nodes.

For example, we upgraded the Lustre cluster Operating System to a newer Linux version and later applied a vulnerability patch to that version. The upgrades and patch were applied to the boot images and downtime for the system only took a few minutes for restarting the file system.

There is no difference, operationally, running the system. Since all nodes boot from the network and the hard drives are all removed, a node can't fail because of a failed drive. Also, security of the system is better managed, since all the boot images and configurations are in a single location.
Another plus is that after a software update when the system inadvertently doesn't run optimally, the nodes in the system simply can be booted with the previous version of the boot images.

Acknowledgments:

Kyle Lamb	(kyle_lamb@lanl.gov), team lead file systems
Susan Coulter	(skc@lanl.gov), tech lead networking
Mike Mason	(mmason@lanl.gov), tech lead file systems
David Sherrill	(dsherril@lanl.gov), HPC3 file systems
David Bonnie	(dbonnie@lanl.gov), HPC3 file systems
Chris Hoffman	(cphoffma@lanl.gov), file systems
Dominic Manno	(dmanno@lanl.gov), file systems