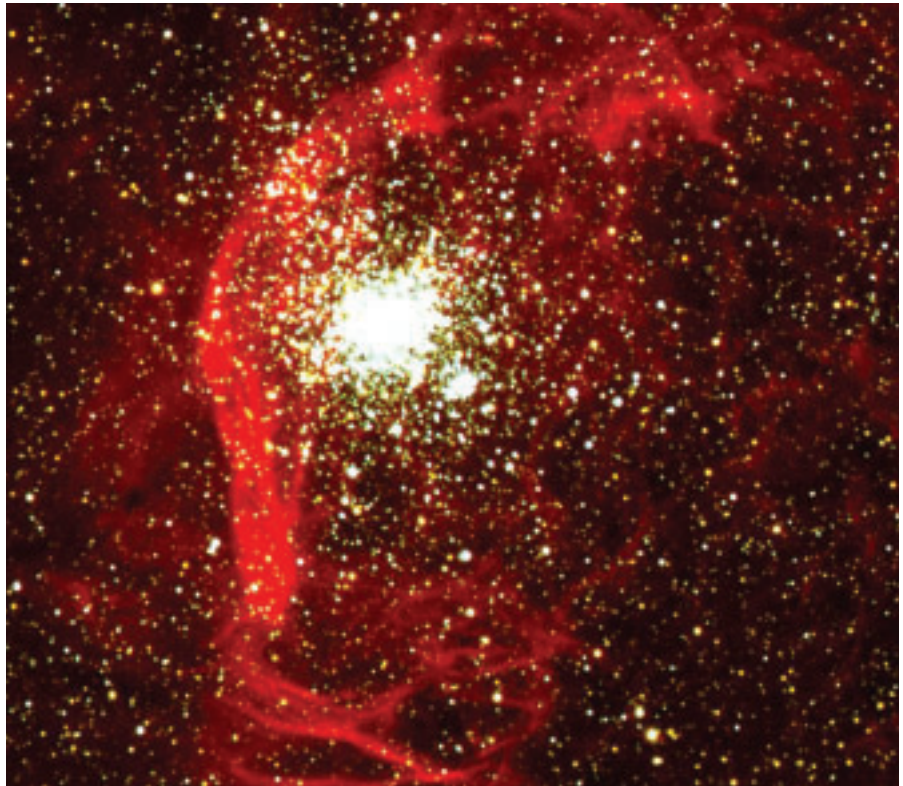


Cluster Filesystems

Grand Designs

Building a cluster with Linux is the first idea when some older PIII nodes are replaced on the desktop. The ease of installation and the readily available software packages to build a cluster computer has changed from an art performed only in the largest computation centers, into the common practice of everyday system installation and administration.

BY JOS VAN WEZEL



Most of the configuration, when building a cluster, is similar to what is done on a single machine. Setting the IP address and changing some startup environment are basically the same on all Linux machines. No specialized knowledge is required. Besides the standard Linux programs, some cluster specific software is needed. One such program is responsible for handing out jobs and controlling the workload. Another special software package helps to install hundreds of machines at the same time. Management software to control or adapt a single node or the whole cluster with a few simple commands is another necessary tool in a cluster.

This article is about the specialized system software that is used in clusters to write and read data. The software glue between the cluster nodes is the cluster filesystem. Highly optimized for the cluster or large installation a cluster filesystem is not found in ordinary stan-

dard Linux distributions. The choice for a certain cluster filesystem depends on the file access pattern and the costs.

A filesystem is necessary

Clusters need background storage that is available to every node. Programs in a cluster usually work on partitioned work areas and need to share data through a common resource like a filesystem. For programs running in a cluster it is most convenient to use the filesystem interface offered by the underlying operating system. It is very difficult to build the storage sharing into the application. After all, the application has to run on different installations, different Linux versions or even different operating systems. Besides, it would severely limit performance.

A filesystem to store data is normally part of each operating system. Linux computers in a cluster have ext, JFS, ReiserFS or equivalent installed, but this storage is confined to the single node.

Applications that run on one node may run on another the next time and need to see the same data space. Secondly there are classes of highly parallel applications that need parallel access to the same data from many nodes at the same time. Therefore data needs to be available to every cluster node and access to the data has to follow the well known UNIX paradigm through `open()`, `read()`, `write()` and `close()` calls.

A cluster filesystem gives global access, that means 'visible at every node', as well as multiple simultaneous access to on-line storage.

Blocks, files and connections

Files are written by the operating system in chunks of a certain size called blocks. This is tunable to a maximum of 4 KB. The filesystem is created with a certain block size. Consequently small changes to a file are handled less efficiently on a filesystem with 4 KB blocks size. Such a layout is better in handling large sequen-

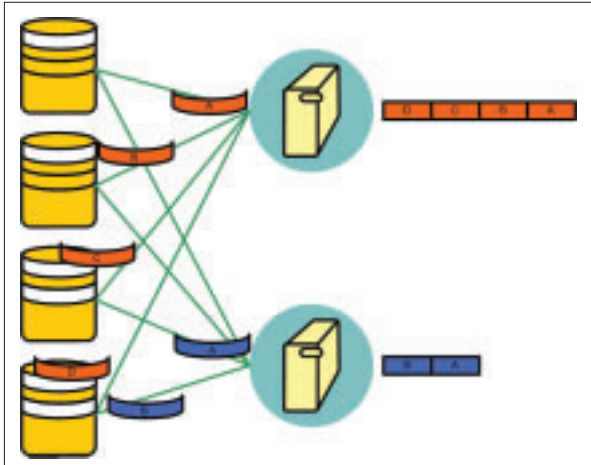


Figure 1: Striped filesystem.

tial IO. The kernel operates on blocks to improve speed and does so also for the IO subsystem. Disk drivers take care of the block transfers from host to disk. The filesystem takes care of the allocation and administration of disk blocks.

Filesystem data access can be block oriented or file oriented. There are solutions that use and extend the local storage and systems that implement the actual block storage themselves. The first method makes the implementation and portability a lot easier, but limits throughput. The other approach is to handle the block storage in the cluster filesystem. This allows optimization of data access and the addition of features for which normally a Logical Volume Manager is responsible. The system is also easier to manage because it has full data control for the complete path from disk to application.

To improve on throughput many solutions offer the possibility to read and write data in parallel to many disks. Files are striped over several disk platters or alternatively, individual files are written to different disks. See Figure 1. The maximum IO throughput is limited by the disk transfer speed capacity and can only increase when more disks are accessed in parallel.

To access the local disks on other cluster nodes, an individual node can use the network connection. An IP/Ethernet link is already available to connect to the outside world. Sometimes a secondary Ethernet connection is used solely for the filesystem originated data transfers. For Linux the Enhanced Network Block Device (<http://www.it.uc3m.es/~ptb/>

nbd) allows to attach a disk over the network. Ethernet is not optimal for block oriented IO and many filesystems support the use of additional inter node connecting hardware to demonstrate their superior throughput. The difference in speed is also reflected in the costs for interconnect hardware. Examples of also called memory attached hardware are Myrinet, Infiniband or Fibre Channel. It is cheaper to run piggy back on Ethernet.

Metadata

A Linux filesystem stores data in files and directories and keeps record about these in i-nodes. The i-nodes contain, among other things, information about the stored files. Examples are the size, creation time, type or the owner of a file. This information is called meta-data. Manipulations to the meta-data can be handled separately from the actual data input and output. This makes it possible to offload the meta-data handling to a dedicated meta-data server. In cluster filesystems the separate meta-data servers are usually also responsible for write locks on a file.

Separation is used to improve throughput. Read operations do not need the meta-data after the location and access rights of the actual data are established.

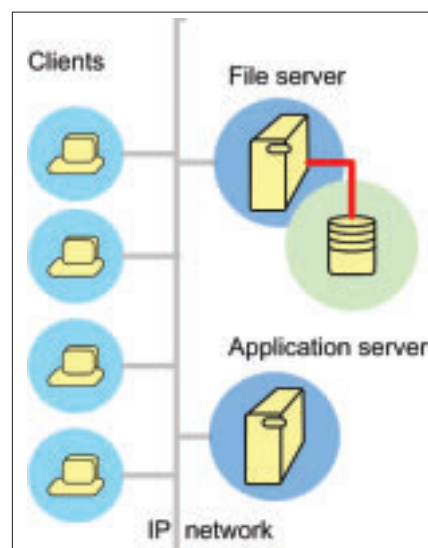


Figure 2: Network attached storage.

The system then directs the IO transfers to dedicated IO server. An IO server with high speed connections can deliver the actual data to the application more efficiently and the meta-data server can continue with other tasks. Because meta-data is much smaller than the actual data it can be completely cached. As the working set in a cluster can become very large, file server caches are never large enough to store all active IO data.

Network Attached Storage

Well known types of cluster filesystems are network filesystems such as NFS or SMB. NFS is used in many installations to connect to Network Attached Storage (NAS) servers. See Figure 2.

High throughput rates are achieved on specialized NAS servers like those by NetApp Exanet, Panasas or Zambeel. These are proprietary, but highly optimized NFS solutions, with their own operating system. With Linux one can take a dual CPU, a RAID controller on a PCI card, 1 GB Ethernet interface, if not on-board already, put in some EIDE disks, install Linux as NFS server and your NAS box is ready.

NAS as a cluster filesystem has its drawbacks. Throughput for small files and non sequential access is slow because of the high latency since all the data has to transfer through the Server/TCP/IP stacks. Linux also has limited storage capacity because of the 32 bit size block pointers although with the 2.6 kernel this scales to 16 TB. The scalability of the self constructed net-

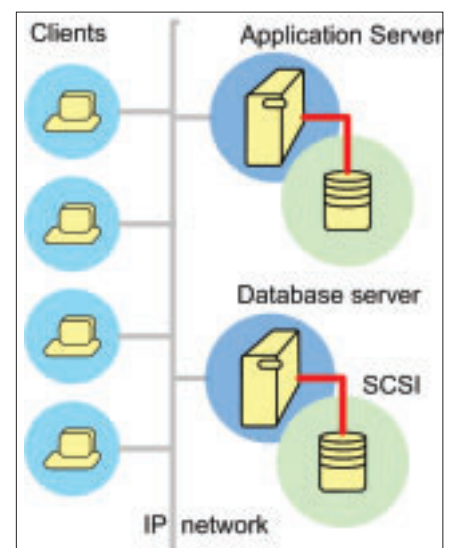


Figure 3: Direct attached storage.

work attached filesystem ends at the single network connection. Even a modern Gigabit link does not suffice to deliver IO for more than a few cluster nodes. A possible solution is Ethernet bonding, which combines two or more devices to increase bandwidth.

Depending on the requirements regarding throughput, costs and to a lesser extend security, a network attached storage system can function as cluster filesystem.

Direct attached storage

Usually these disks are located next to the computer in the same housing or in near vicinity. Therefore this is called direct attached storage as opposed to network attached. See Figure 3. The distance is limited by the specifications of the copper based connections. Fibre Channel has opened the possibility to connect disks at larger distance. FC can connect hosts to storage devices directly or via a FC switch. Switches can connect to switches to build a storage network or SAN. See Figure 4.

Where SCSI or ATA is limited to connections between host and storage, Fibre Channel is used to build a storage area network or SAN that connects hundreds of hosts and storage devices. The Fibre Channel protocol is optimized for storage devices. Features are the low latency and protocol offloading which reduces the interrupt and processing load on the host.

Network attached direct storage

Storage and networking are increasingly integrating on the hardware side. iSCSI is a standard that defines the SCSI protocol over IP. Conversely there is FC-IP, which defines IP over Fibre Channel. Infiniband offers both IP and FC, on the same connection hardware. Modern high reliability makes a Local Area Network a good IO candidate for block transfers which are usually handled by direct attached storage.

Well known cluster filesystems

We will limit ourselves to some of the most interesting cluster filesystems-

tem solutions for Linux. Not all solutions are designed for use in a cluster, but they are very capable to do so.

NFS

NFS is a network based, file oriented filesystem. Because NFS, when run over UDP, is stateless, clients experience only a short stall if the network is unavailable or if a NFS server is rebooted. Clients may disappear without notice and the server does not have to do anything to recover. This is in contrast with all other mentioned systems below that do need to clean up after (contact to) a client or cluster member is lost.

Directory hierarchies local to the server are made available to others by exporting them. Clients mount the exported directories on any location in their own filesystem. A client cannot export a mounted NFS filesystem. As NFS uses a weak security model this makes it impossible to safely share over WAN.

Exported filesystems are usually maintained in a database. The database can be NIS or LDAP. The autofs system, uses this database to automatically mount the proper file hierarchy. The trigger for the automated mount is a program entering the directory which is defined as mount point. NFS mounts within NFS mounts are allowed.

The combination with autofs and the network wide database makes NFS a very good candidate for use in a cluster. Perhaps the only limitation is its speed, but this depends heavily on the access pattern.

OpenAFS

This is both a network and a distributed filesystem. It offers file sharing, even over a WAN, and a global name space. The filesystem is completely virtual and kept on (replicated) data servers and metadata servers. Clients build up a cache of recently used files which is regularly flushed to the data servers. Applications use the cached data and continue to work when the connection to the data server is broken but have to wait on close of the open file until the server is on line again.

AFS has been branched into the Distributed File System (DFS) which was maintained and marketed by an IBM subsidiary. DFS is no longer supported and development has stopped. It is an excellent shared filesystem for a campus or even world wide because it also has a well established security model. An instance or administrative domain is called a cell. Users authenticate themselves and are sent a security token with a limited lifetime. The token allows the cache manager on the local machine to talk to the AFS file server.

With OpenAFS an administrator can add disk space and replace disk space without service interruption. Because the cache manager sits between user and server, data migration to other servers can happen transparently. A special version of OpenAFS called MultiResident-AFS interfaces to tape and allows automatic data migration to offline storage.

Use of OpenAFS in a cluster is not recommended for high data throughput.

The benefit of a cache, which may enhance stability, is obviated by the slow file server access which runs in user space.

GPFS

The General Parallel File System is a commercial product of IBM. GPFS is a truly parallel filesystem. Data can be striped over many disks and any node can access the same file at the same time.

There are two possible access configurations, either via SAN or via direct attached storage. In the SAN configuration each node sees each block on all storage elements that are made available to the

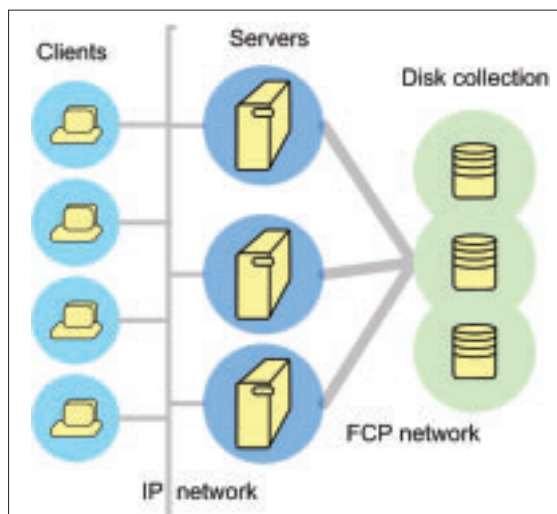


Figure 4: Storage area network storage.

GPFS. Files are assembled from the blocks distributed in the SAN and are directly available on the node. The direct attached configuration relies on a high speed inter-node network which can be Ethernet or Myrinet. Blocks from local disks are shipped to other nodes. Files are assembled by gathering blocks over the IP network from local disks on several nodes. It is similar to PVFS.

Management is very simple. Commands can be issued from any node. The system has the capability of adding and removing disks, rebalance data access, change the block size and the number of possible I-nodes. It overcomes the maximum filesystem size limitation because it allows a configurable block size as large as 1 MB.

There is one node per open file for handling the metadata. All nodes can access the same file, but changes to the meta-data are handled by the meta-data node. The locking is distributed over the nodes accessing the file. All data is written and read in parallel and throughput scales linearly with the number of disks and nodes. GPFS filesystems can be exported with AFS or NFS from dedicated servers. The compute nodes then mount the exported filesystems.

GPFS depends on very expensive SAN infrastructure to achieve a high performance. The configuration where an IP network is used to assemble disk stripes can become an early bottleneck for many data access patterns.

LUSTRE

LUSTRE is a new and being actively developed. Although Lustre is marketed by HP, the project is committed to the open source license model. Lustre has excellent documentation. For configuration and logging Lustre relies on the open standards LDAP and XML. Lustre is a file (object) based system.

Everything stored in Lustre is considered an object. The objects of the filesystem are (special) files and directories. The attributes, meta-data, of these objects such as size, creation time, symbolic link pointers or backup flags are stored on metadata servers (MDS). The meta-data is kept separate from the actual content. The MDS takes care of file creation, attribute manipulation and is responsible for the namespace han-

dling: a file is found by asking the MDS. After opening, the MDS relays the actual IO to Object Storage Targets (OST) to take care of the data exchange. The MDS keeps track of the data exchange in a journal. Creating and writing a file involves the creation of an i-node on the MDS which then contacts an OST to allocate storage. The allocation can be striped across several OSTs to enhance performance.

Throughput achieved in some published tests is impressive. At the moment, Lustre still lacks important maintenance tools to use it in a production environment. There is no filesystem recovery utility and there is not yet an automatic failover for the single MDS. The original approach and the development from the ground up makes Lustre a solution that could develop into a very powerful and elegant system.

PVFS

The Parallel Virtual File system is block based and provides high performance for I/O intensive parallel or distributed applications. The usual application environment is a small (< 50 nodes) cluster but there are no inherent limits. Parts of the internal disk are made available to PVFS on IO nodes.

The file space of these disks is then distributed to the complete cluster and accessed via Ethernet a kernel module and the libpvfs lib installed on the clients. Clients can be IO nodes (IOD) themselves and one node or client has to be configured as the meta-data node (MDS).

Files are striped over the IO nodes. After the initial administrative data exchange with the MDS all data traffic with the IODs is handled by the clients individually via the libpvfs. The library orchestrates the assembly of the files from the stripes distributed over the IODs. PVFS supports Myrinet and Infiniband for intra-node communication.

PVFS currently contains neither means for data redundancy nor is it possible to recover from a failed node. There is a potential bottleneck at the manager level as the number of client nodes increases. PVFS cannot go beyond the restrictions introduced by TCP/IP on Linux, such as limits on the number of simultaneous open system sockets and TCP/IP proto-

col overhead. PVFS must be installed on the cluster nodes since it does not allow export via NFS or AFS.

PVFS2 is a code rewrite based on the experiences gathered with PVFS1. It has structural enhancements such as user controlled striping and distributed meta-data. This allows the installation of more than one meta-data controllers which relieves this bottleneck.

OpenGFS

OpenGFS or OGFS also implements a journaled block based filesystem that provides read and write access from multiple nodes. The dreaded 'pool' code was changed to allow OGFS to use any logical volume manager. ELVM is preferred. Most recently the memexp locking was replaced by the OpenDLM module. The old memexp was a single point of failure and very compute intensive. OGFS supports growing of filesystems and the addition of disks (through the separate LVM). Node failures are handled by log recovery and isolating the failed node.

Final remarks

None of the presented systems are ideal. None of the discussed variants of cluster capable filesystems is perfect for all purposes. The open source packages LUSTRE und OpenGFS are in their early development and not production ready. OpenAFS lacks throughput capacity. Clearly the Linux and open source community still have some time to go in developing stable, scalable cluster storage. The commercial software GPFS or Sistas GFS are reportedly more stable and scalable.

After you have made your choice it is not possible to run `rpm -i` and forget about it. The software has to be tuned for the specific environment. The default values are never the optimal values. Proper configuration forces you to think about data access patterns, optimal IO paths, possible bottlenecks, block sizes, strip sizes, cache usage etc. After installation of the cluster filesystem the fun just starts. ■

INFO

[1] OpenGFS: <http://www.sourceforge.net/projects/opengfs>

[2] GFS: <http://www.sistina.com>