Parallel Platters: File Systems for HPC Clusters Part Three

In the last installment of our Parallel Platters series, Jeff Layton looks at the next generation of parallel file systems: Object Based File Systems.

Jeffrey B. Layton
Thursday, November 15th, 2007

As clusters require faster and faster access to data, while maintaining reliability, new filesystems are being developed to try to break the bottlenecks and provide scalable solutions. The next generation of filesystems, dubbed *object based* file systems, help break performance barriers by allowing data devices to manage where data is stored.

The first part of this series covered *distributed file systems*, with a focus on those using *NFS* as the underlying protocol. In the second part, we examined three parallel file systems based on more traditional architectures: *GPFS* from IBM, *Ibrix*, and *MPFSi* from EMC.

In this third and final installment, I will cover what might be described as the next generation of parallel file systems, object based file systems. As with the previous installments, I won't be able to cover all of the file systems which fit the description due to space considerations, but will try to focus on those that are most relevant.

A Brief Introduction to Object Based Storage

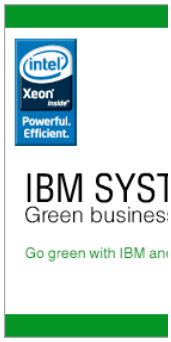
Object storage sounds simple, and conceptually it is, but just like all of the other parallel file systems many gotchas need to be addressed to make it reliable and useful as a real file system. The key concept of an object based file system needs to be emphasized since it is a powerful concept for parallel file systems.

In a more classic file system, such as a block based file system, the metadata manager is contacted and a set of inodes or blocks is allocated for a file (assuming a write operation). The client passes the data to the metadata manager, which then sends the data to the file system and ultimately to the disk. The metadata manager is responsible not only for the metadata itself but also where the data is located on the storage. If you notice, the metadata manager is a key part of the process. It also creates the potential for a performance problem.

On the other hand, object storage takes a different approach allowing the storage devices themselves to manage where the data is stored. In an object storage system, the metadata manager and the storage are usually separate devices. The storage devices are not just dumb hard drives, but have some processing capability of their own. With an object storage system the metadata manager is contacted by a client about a file operation, such as a write.

The metadata manager contacts the storage devices and determines where the data can be placed based on some rules in the file system (e.g. striping for performance, load or capacity balancing across storage devices, RAID layout for resiliency). Then the metadata manager passes back a map of which storage devices the client can use for the file operation. Once this is complete the metadata manager gets out of the way of the actual file operation and allows the client to directly contact the assigned storage devices.

The metadata manager constantly monitors the operations so that it knows the basics of where the data is located but it stays out of the middle of the data operations. If there is a change in the file operation, such as another client wanting to read or write data to the file, then the metadata manager has to get involved to arbitrate the file operations. But, in general, the data operations happen without the metadata manager being in the middle.



The Advantage to Object Based Storage

Object storage's key benefit is that if the data on the storage devices needs to be moved, it can happen easily without the intervention of the metadata manager. The movement of the data can be for many reasons, most of them important for performance or resiliency of the file system. The storage devices just move the data and inform the metadata manager where it is now located.

Jan Jitze Krol of Panasas has a great analogy for object storage. Let's assume you are going to a mall to go shopping. You can drive your car into the parking garage and park on the third level, zone 5, seventh spot from the end and then go in and shop. When you finish you come out and you have to remember where you parked and then you go directly to your car. This is an analogy to the way a typical file system works. For the object storage analogy, let's assume you arrive at the parking garage and the valet parks your car for you. You go in and shop and when you finish you just present the valet with your parking ticket and the valet determines where your car is located and gets it for you.

The advantage of the valet is that they can move the car around as they need to (perhaps they want to work on part of the garage). They can even tear down and rebuild the garage and you won't even know it happened. The parking analogy demonstrates the flexibility of object based storage. An object based system can optimize data layout (where your car is located), add more storage (add to the parking garage), check the data to make sure it's still correct (check your car to make sure it's not damaged and if it is, fix it for you or inform you), and a whole host of other tasks. Notice that these tasks can be done with minimal involvement of the metadata manager (in this analogy, you).

So you can see that object storage has a great deal of flexibility and possibilities for improving file system performance and scalability for clusters. Now that the basics are out of the way, let's examine the three major object file systems for clusters, Lustre, Panasas, and PVFS.

Lustre

We'll start with *Lustre*, one of the few open source parallel file systems. Lustre is an object based file system that has the potential of scaling to ten of thousands of nodes and Petabytes of data. Lustre stores data as objects called containers that are very similar to files, but are not part of a directory tree.

The advantage to an object based file system is that allocation management of data is distributed over many nodes, avoiding a central bottleneck. Only the file system needs to know where the objects are located, not the applications, so the data can be put anywhere on the file system.

As with other file systems, Lustre has a metadata component, a data component, and a client part that accesses the data. However, Lustre allows these components to be put on different machines, or a single machine (usually only for home clusters or for testing). The metadata can be distributed across machines called MetaData Servers (MDS), to ensure that the failure of one machine will not cause the file system to crash. The metadata servers support failover as well. In Lustre 1.x, you can use up to two MDS machines (one in active mode and one in standby mode) while in Lustre 2.x, the goal is to have tens or even hundreds of MDS machines.

The file system data itself is stored as objects on the Object Storage Servers (OSS) machines. The data can be spread across the OSS machines in a round-robin fashion (striped in a RAID-0 sense) to allow parallel data access across many nodes resulting in higher throughput. This distribution also helps to ensure that a failure of an OSS machine will not cause the lose of data. But, the data is not written in any kind of RAID format to help preserve or recover data. The client machines mount the Lustre file system in the same way other file systems are mounted. Lustre mount points can be put into /etc/fstab or in an automounter.

The MDS machines, OSS machines, and clients can all reside on one machine (most likely for testing or learning) or can be split among multiple machines. For example, you can designate a set of nodes within a cluster as OSS machines (sometimes called I/O nodes), and two nodes for MDS. Then the compute nodes within the cluster are clients and mount the file system.

Lustre uses an open network protocol to allow the components of the file system to communicate. It uses an open protocol called Portals, originally developed at Sandia National Laboratories. This allows the networking portion of Lustre to be abstracted so new networks can be easily added. Lustre supports TCP networks (Fast Ethernet, Gigabit Ethernet, 10GigE), Quadrics Elan, Myrinet GM, Scali SDP, and Infiniband. Lustre also uses Remote Direct Memory Access (RDMA) and OS-bypass capabilities to improve I/O performance.

Notice that Lustre is purely a file system. You have to select the storage hardware you want to use for the OSS and MDS systems. If you examine some of the Lustre systems that are functioning around the world you can gain some insight into how you can architect such a system. For example, it's advisable to connect a set of storage between at least two OSS nodes so that if one OSS fails, the other (s) can continue to access the storage. It's also advisable to use some sort of RAID protection on the storage itself so that you can recover from a failed disk. One other thing to note is that it is a good idea to configure the MDS node using HA (High Availability) in the event that the MDS node goes down.

One of the previous complaints about Lustre has been that you have to patch the kernel for the clients to use Lustre. Very few people could take the patches and apply them to a Kernel.org kernel and successfully build and run Lustre. People had to rely on ClusterFS for kernels. This was a burden on the users and on ClusterFS. With the advent of Lustre 1.6.x, ClusterFS now has a patchless kernel for the client if you use a 2.6.15-16 kernel or greater. According to ClusterFS, there may be a performance reduction when using the patchless client, but for many people this is a useful thing since they can quickly rebuild a Lustre client kernel if there is a security problem.

Lustre follows a slightly unusual open source model so that development for the project can be paid for. In this model, the newest version of Lustre is only available from Cluster File Systems, the company developing Lustre, while the previous version is available freely from www.lustre.org.

PanFS

Panasas is a storage company focused on the HPC market. Its ActiveStor Storage Cluster is a high-speed, scalable, global, storage system that uses an object based file system called *PanFS*. Panasas couples its file system with a commodity based storage system termed Panasas Storage Blades. The basic storage unit consists of a 4U chassis, called a shelf, that can accommodate up to 11 blades. Each blade is a complete system with a small motherboard, a CPU, memory, NICs, one or two disks, and two internal GigE connections. There are two types of blades— a Director Blade (DB) that fills the role of a metadata manager, and a Storage Blade (SB) that stores data.

The most common configuration is to have one Director Blade (DB) and 10 Storage Blades (SB) in one shelf. But you can have up to three DB's or 11 SB's per shelf. As of this writing Panasas has 500GB, 1TB, and 1.5TB storage blades giving a nominal storage of 5TB, 10TB, and 15TB per shelf using off the shelf SATA drives. In addition, the Storage Blade can have either 512MB of memory or 2GB of memory, the extra memory being used for cache. In addition, each shelf has redundant power supplies as well as battery backup.

Each ActiveStor shelf connects to the network using up to four GigE links, which are typically grouped together to improve performance. The ActiveStor product line only supports TCP for data communication, so the storage hardware and the clients use TCP for communication. Some customers have used what are termed I/O-routers to connect clients over high-speed links (such as Infiniband) to improve the throughput for a single client. These I/O routers are connected to the storage using GigE and then push the data onto an IB network, typically using IPoIB (IP over IB).

With ActiveStor, you can use a very large number of shelves for a single file system. You just install a shelf into the rack, plug it into the network and power up the shelf. Then you can log into the GUI for the file system and either add the new shelf to an existing name space or use it to create a new name space. In either case as soon as you define where the storage goes, it is available to the file system. Also, notice that shelves usually includes multiple Director Blades. So as you add shelves, you get better more metadata capability.

The object-based file system, PanFS, is the unique feature of Panasas' storage system. It works in pretty much the same fashion as other object based file systems. The client contacts a metadata manager when it wants to perform a file operation. The metadata manager sends a map of where the data is located (or to be located) to the client. Then the client communicates with the storage blades directly and in parallel.

As data is written to the Storage Blades, it is broken into 64KB chunks. File attributes, an object ID, and other information are added to each chunk to complete an object. The objects are then written to the Storage Blades. The objects are typically written using either RAID-1 for 64KB or smaller files or RAID-5 for file larger than 64KB. So the objects and the resulting parity are written to the Storage Blades in what is called a stripe. A stripe is just a number of blades where data is written across. So a stripe width is the number of storage blades used for writing the objects and the parity.

Panasas also has a unique notion of a stripe depth. After a stripe is written the set of objects in the next stripe are written to the same set of Storage Blades. This is done until a certain number of stripes, called the stripe depth, are written. Then a new set of Storage Blades is chosen for the next set of stripes. This allows the file system to be capacity balanced which also helps performance.

When the client accesses the Storage Blades directly and in parallel, it's using a Panasas developed data access protocol, DirectFlow. This protocol allows the clients to send data access request in parallel, improving the performance greatly. Using DirectFlow, Panasas has measured the performance of a single shelf (four GigE links) with a single DirectorBlade at 350MB/s for reading data.

Panasas has announced that they will be releasing big parts of the Direct Flow client as open source to help speed the development and deployment of pNFS. The Direct Flow Client is very similar to the NFSv4.1 client so this should definitely help the development of pNFS.

In addition to DirectFlow, clients can also access the file system using NFS and CIFS. The Director Blades act as NFS and CIFS servers. So if you have more than one DB, you can load balance NFS and CIFS across them, resulting in a clustered NAS solution. This makes the file system very useful for desktops or for systems that do not run Linux.

One aspect of object storage that Panasas has taken advantage of is the ability to quickly reconstruct the data if a blade fails. With classic RAID controllers when a drive fails, the controller reconstructs the missing data from the remaining data and the parity and writes it to a new drive.

However, the problem is that with block based storage, the RAID controller reconstructs all of the blocks on the failed drive even if the blocks contained no data. So it is rebuilding non-existent data. Moreover if you hit an Unrecoverable Read Error (URE) during the reconstruction, then the reconstruction process stops and you have to restore all of the data from a backup. The probability of hitting a URE is fairly high considering that the URE rate has not been improving and yet the size of the drives has increased dramatically. In fact, with 1TB drives (or larger) and RAID-5 (seven drives), you are virtually guaranteed to hit a URE during reconstruction.

Many RAID systems are using RAID-6 now, to help reduce the probability of hitting a URE that would cause the reconstruction to stop. However, RAID-6 requires an extra drive, more computational work, and also has a slower write rate than RAID-5. Even with RAID-6's advantages, it may not be able to help you to overcome the UREs during reconstruction.

In PanFS, the RAID is done on a per file basis, not a drive basis. So if a blade fails, the reconstruction is done from the remaining objects and parity for only the affected files. It does not reconstruct non-existent data. Moreover, if a URE is encountered during the reconstruction, then PanFS just moves on to the next file and continues to reconstruct. Then you just have to restore the missing file, as opposed to all of the data.

In addition, rather than have just one Director Blade (DB) reconstruct the data, all of the DB's can help in the reconstruction. According to Panasas after about three DB's the reconstruction happens at a faster rate than any RAID controller. The reconstruction rate also increases fairly linearly as you add DB's. Given the size of today's drives and the probability of losing a drive and encountering a URE, the approach that Panasas has taken greatly improves the recovery of the file system in the event of a failure.

Panasas also has some other capabilities up their sleeve. Its blades run a daemon that checks all of the sectors on the drives. This allows bad sectors to be found before they are used. In addition, another daemon checks the data that is already on the blades by comparing the existing parity with a computed parity. This allows data that has encountered a bad sector to either be reconstructed or restored if necessary. The ActiveScale operating environment also has a feature called blade drain. If a blade is found to" bad" then the administrator can mark the blade as down so the the file system won't write any more data to it and the data that is on the blade can be moved to other blades. Once all of the data has been moved off of the blade then it can be replaced. I'm not aware of other file systems with this capability.

PVFS2

The *Parallel Virtual File System* (PVFS) is an open source project with a number of contributors. PVFS is not designed to

be a persistent file system for something like user's home directories but rather as a high-speed scratch file system for the storage of data during the run of a job.

The original PVFS system, now called PVFS1, was originally developed at Clemson University but with the advent of PVFS2 it is not in use much any more. PVFS2 is a complete rewrite of PVFS1 focusing on improving the scalability of PVFS, the flexibility of PVFS, and the performance of PVFS. Currently PVFS2 natively supports TCP, Infiniband, and Myrinet networks. In addition, PVFS2 accommodates heterogeneous clusters allowing x86, PowerPC, and Itanium machines to all be part of the same PVFS file system. PVFS2 also adds some management interfaces and tools to allow easier administration.

PVFS2 divides the functions of the file system into two pieces, metadata servers and data servers. In general, PVFS2 has only one type of server- pvfs2-server. What function a particular server fulfills depends upon how it is started on the node. PVFS2 also has the ability to accommodate multiple metadata servers (PVFS1 can only accommodate one). The clients mount PVFS as if it were a normal file system after the PVFS software has been installed.

The" V" in PVFS stands for Virtual. This means that it doesn't write directly to the storage devices, but instead the data resides in another file systems that does the actual I/O operations to the storage devices. When data is written to PVFS, it is sends to the underlying file system in chunks. The underlying file system then writes the data to the storage. For example, you could use Ext3, ResierFS, JFS, XFS, Ext2, etc. as the underlying file system for PVFS. If you happened to look at a PVFS directory on the underlying file system, you will see lots of files with strange names. This is PVFS data. Consequently PVFS looks a lot like an object based file system and is, in fact, an object based file system.

This approach gives PVFS a great deal of flexibility. For example, you could have several storage nodes running ext3 and another set of nodes running XFS and PVFS will work perfectly fine. When data is written to PVFS it is broken into chunks of a certain size and then sent to the data servers in some fashion, usually in a round-robin fashion. The size of the chunks, which storage nodes are used, and how the chunks are written, are all configurable, allowing PVFS to be tuned for maximum performance for a given application or class of applications.

PVFS2 uses a number of features to for improved performance. For example, it can utilize multiple network interfaces on the client node. You can also change the underlying file system for improved performance. The behavior of the local file system can have a big impact of the PVFS2 performance. You can also use extended attributes to set directory hints to improve performance.

Recall that PVFS is designed to be a high-speed scratch file system. Consequently it does not have much fault tolerance built into it. You can achieve some level of fault tolerance but you have to configure it. For example, PVFS2 does not naturally tolerate disk failures, but you can use disks that are mirrored or behind a RAID controller to give some fault tolerance. You can do the same for network interfaces by using more than one interface. You can even configure the servers to be fault tolerant by using HA between two servers. The PVFS Web site has an paper on how you can go about using HA.

However, PVFS is fault tolerant in one respect. If you lose a server the file system will not go down. For example let's assume you have several data servers and one of them goes down while it is being used by clients. If any client was actively writing to that server, then the data on the network or in the process of being written will be lost. But any client that was not using the server will then continue to run just fine. If a client was using the server but was not actively writing data to it, then it will continue to run but will not user the downed server during the next I/O phase. If the server is returned to service without any changes to the file system underneath PVFS, then the data that was on the server will be available once again.

But, if the disks in the server are wiped, then the data is lost. This is a design decision that the PVFS developers have made. PVFS is designed to be a high-speed storage space, not a place for home directories. The intent is to use PVFS for storage of data during an application run and then move the data off of PVFS onto a permanent storage space that is backed by tape or some other type of archiving media.

This decision has actually freed the developers so that they can focus on improving performance without being shackled by keeping compatibility with a file system interface. The developers have created several ways to access PVFS from the clients. First, there is a system interface that is used as a low-level interface for all of the other interfaces. The second interface is the management interface. It is intended for administrative tasks for PVFS such as fsck or for low-level file information. The there is a Linux kernel driver that is really a kernel module than can be loaded into an unmodified Linux kernel so the the Linux VFS (Virtual File System) can access PVFS2. This allows standard UNIX commands such as ls and cp to work correctly. And, finally, there is a ROMI/O PVFS2 device that is part of the standard ROMI/O distribution. ROMI/O supports MPI-I/O for various MPI libraries. Using this device ROMI/O can better utilize PVFS2 for I/O functions.

So Long and Thanks for All of the File Systems

The file systems that I have covered in the three parts of this article are just some of what is available. I've tried to cover the more popular file systems for clusters, but undoubtedly I have missed some. I invite you to use Google and investigate other file systems. I hope that the presentation of file systems in this series will provide you with the necessary information to look at other file systems with a more critical eye.

Jeff Layton is an Enterprise Technologist for HPC at Dell. He can be found lounging around at a nearby Frys enjoying the coffee and waiting for sales (but never during working hours).

Read More

1. Before You Commit: Four Key Questions To Ask About InfiniBand
2. The Personal Cluster: Coming To A Desk Near You
3. Fun and Games
4. Storage Convergence: Fibre Channel over Ethernet
5. HPC Hardware is Free

1 Comment on Parallel Platters: File Systems for HPC Clusters Part Three »

Comments via RSS

npate131 said: +0

This is a very good writing on file systems. Thanks Linux Magazine

July 16th, 2008 9:18 AM (permalink)

Reply to this comment

Comments via RSS

You are logged in as **goldjay**. Logout »

Have a Gravatar? Your Gravatar pic will appear next to your comments. [?]

Your Comment

You may use <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <code> <i> <strike> in your comment.

Add comment

LinuxMagazine.com v4.0