Technical Articles

File Systems for HPC Clusters, Part Two: Parallel File Systems

Jeffrey Layton, Ph.D.
Posted: Sept 15, 2007

Introduction

Part 1 of this article covered Distributed File Systems focusing primarily on those using NFS as the protocol. There was one exception however - **pNFS**. pNFS could easily become the future parallel file system for clusters. However, it will take some time for it to be approved and for vendors to write the appropriate drivers. In the meantime, there are parallel file systems available for clusters now. Part 2 of this article will touch on some of the major parallel file systems that use more traditional approaches to file systems and are available now. I can't cover all of them due to space considerations. I apologize if I don't cover one that you are interested in or one that you use.

Parallel File Systems

Parallel File Systems - what do they look like?. OK, this isn't **Real Genius** but parallel file systems are a hotly discussed technology for clusters. They can provide lots of IO for clusters if that's what you truly need (I'm constantly surprised at how many people don't know how IO influences the performance of their codes). They can also provide a centralized file system for your clusters and in many cases you can link parallel file systems from geographically distinct sites so you can access data from other sites as though it was local. Centralized file systems can also ease a management burden and also improve the scalability of your cluster storage. You can also use them for diskless systems by providing a reasonable high performance file system for storage and scratch space.

Parallel File Systems are distinguished from Distributed File Systems because the clients contact multiple storage devices instead of a single device or a gateway. An easy explanation of the difference is that with Clustered NAS devices, the client contacts a single gateway. This gateway contacts the various storage devices and assembles the data to be sent back to the client. A Parallel File System allows the client to directly contact the storage devices to get the data. This can greatly improve the performance by allowing parallel access to the data, using more spindles, and possible moving the metadata manager out of the middle of every file transaction.

There are lots of ways a parallel file system can be implemented. I'm going to categorize the parallel file systems into two groups. The first group uses more traditional methods such as file locking as part of the file system. They also use more traditional approaches to file systems such as block based or even file based schemes. The second group are object based file systems and will be covered in Part 3 of this article. Let's start the discussion of parallel file systems by covering the file systems that are more traditional in their architecture. The first one I want to cover is perhaps the oldest and probably the most mature parallel file system, GPFS.

GPFS

GPFS (General Purpose File System) is perhaps the oldest and most mature parallel file system. For many years it was available only on AIX systems, but several years ago IBM ported it to Linux systems. Initially it was available only on IBM hardware, but in about 2005 IBM made GPFS available for OEMs so that it's now available for non-IBM hardware. The only OEM I'm aware of is **Linux Networx**.

GPFS is a high-speed, parallel, distributed file system. In typical HPC deployments it is configure similarly to other parallel file systems There are nodes within the cluster that are designated as IO

nodes. These IO nodes have some type of storage attached to them whether it's direct attached storage (DAS) or some type of SAN (Storage Area Network) storage. (Note There are many variations on this **theme** where you can combine various types of storage.).

GPFS achieves high-performance by striping data across multiple disks on multiple storage devices. As data is written to GPFS, it stripes successive blocks of each file across successive disks. But it makes no assumptions about the striping pattern. There are 3 striping patterns it uses:

- **Round Robin**: A block is written to each **LUN** in the file system before the first one is written to again. The initial order of LUNS is random, but that same order is used for subsequent blocks.
- **Random**: A block is written to a random LUN using a uniform random distribution with the exception that no two replicas can be in the same failure group.
- **Balanced Random**: A block is written to a random LUN using a weighted random distribution. The weighting comes from the size of the LUN.

To further improve performance, GPFS uses client-side caching and deep prefetching such as read-ahead and write-behind. It recognizes standard access patterns such as sequential, reverse sequential, and random and will optimize IO accesses for these particular patterns. Furthermore GPFS can read or write large blocks of data in a single IO operation.

The amount of time GPFS has been in production has allowed it to develop several more features that can be used to help improve performance. When the file system is created you can chose the blocksize. Currently, 16K, 64K, 512K, 1M, and 2M block sizes are supported with 256K being the most common. Using large block sizes helps improve performance when large data accesses are common. Small block sizes are used when small data accesses are common. GPFS subdivides the blocks into 32 sub-blocks. A block is the largest chunk of contiguous data that can be accessed. A sub-block is the smallest contiguous data that can be accessed. Sub-blocks are useful for files that are smaller than a block and are stored using the sub-blocks. This can help the performance of applications that use lots of small data files (i.e. Life Sciences applications).

As part of the resiliency of the file system, GPFS uses distributed metadata so that there is no single point of failure nor a performance bottleneck. To get HA (High Availability)capabilities, GPFS can be configured using logging and replication. GPFS will log (journal) the metadata of the file system so in the event of a disk failure, it can be replayed so that the file system can be brought to a consistent state. Replication can be done on both the data and the metadata to provide even more redundancy at the expensive of less usable capacity. GPFS can also be configured for fail-over both at a disk level and at a server level. This means that if you lose a node GPFS will not lose access to data nor degrade performance unacceptably. As part of this it can transparently fail-over lock servers and other core GPFS functions, so that the system stays up and performance is at an acceptable level.

As mentioned earlier GPFS has been in use probably longer than any other parallel file system. In the Linux world, there are GPFS clusters with over 2400 nodes (clients). One aspect of GPFS that should be mentioned in this context is that the GPFS is priced by the node for both IO nodes and clients.

In the current version (3.x) GPFS only uses TCP as the transport protocol. In version 4.x it will also have the ability to use native IB protocols (presumably SDP). In addition, the IO nodes of GPFS can act as NFS servers if NFS is required. Typically this is done for people who want to mount the GPFS file system on their desktop. More over, GPFS can also use CIFS for Windows machines. Of course NFS and CIFS are a lot slower than the native GPFS, but it does give people access to the file system using these protocols.

While GPFS has more features and capabilities than I have space to write about, I want to mention two other somewhat unique features. GPFS has a feature called multi-cluster. This allows two different GPFS file systems to be connected over a network. As long as gateways on both file systems can "see" each other on a network, you can access data from both file systems as though they are local. This is a great feature for groups that may be located around the world to easily share data.

The last feature I want to mention is called the GPFS Open Source Portability Layer (GPL). GPFS comes with certain GPFS kernel modules built for a basic distribution. The portability layer allows

these GPFS kernel modules to communicate with the Linux kernel. While some people may think this is a way to create a bridge from the GPL kernel to a non-GPL set of kernel modules, it actually serves a very useful purpose. Suppose there is a kernel security notice and you have to build a new kernel. If you are dependent upon a vendor for a new kernel, you may have to wait a while for them to produce a new kernel for you. During this time you are vulnerable to a security flaw. With the portability layer you can quickly rebuild the new modules for the kernel and reboot the system yourself.

IBRIX

IBRIX offers a distributed file system that presents itself as a global name space to all the clients. IBRIX' *Fusion* product is a software only product takes whatever data space you designate on what ever machines you choose and creates a global, parallel file system on them. This file system, or "name space," can be mounted by clients who can share the same data with all of the other clients. In essence, each client sees that exact same data, hence the phrase, "single" or "global" name space. The key to Fusion is that the bottlenecks of other parallel global file system approaches have been removed. According to IBRIX, this allows the file systems to scale almost linearly with the number of data servers (also called IO servers). It can scale up to 16 Petabytes (a Petabyte is about 1,000 Terabytes or about 1,000,000 Gigabytes). It can also achieve IO (Input/Output) speeds of up to 1 Terabyte/s (TB/s).

The IBRIX file system is composed of "segments." A segment is a repository for files and directories. A segment does not have to be a specific part of a directory tree and the segments don't have to be the same size. This allows the segments to be organized however the file system needs them or according to policies set by the administrator. The file system can place files and directories in the segments irrespective of their locations within in the space. For example, a directory could be on one segment while the files within the directory could be spread across several segments. In fact, files can span multiple segments. This improves throughput because multiple segments can be accessed in parallel. The specifics of where and how the files and directories are placed occurs dynamically when the files or directories are created based on an allocation policy. This allocation policy is set by the administrator based on what they think the access patterns will be and any other criteria that will influence it's function (e.g. file size, performance, ease of management). IBRIX Fusion also has a built-in Logical Volume Manager (LVM) and a segmented lock manager (for shared files).

The segments are managed by segment servers with each segment only having one server. However segment servers can manage more than one segment. The clients mount the resulting file system through one of 3 ways: (1) using the IBRIX driver, (2) NFS, (3) or CIFS. Naturally the IBRIX driver knows about the segmented architecture and will take advantage of it to improve performance by routing data requests directly to the segment server(s). If you use the NFS or CIFS protocol the client mounts the file system from one of the segment servers. You will most likely have more than segment server so this allows you to load balance NFS or CIFS.

The segmented file system approach has some unique properties. For example the ownership of a segment can be migrated from one server to another while the file system is being actively used. You can also add segment servers while the file system is functioning, allowing segments to be spread out to improve performance (if you are using the IBRIX driver). In addition to adding segment servers to improve performance, you can add capacity by adding segments to current segment servers.

The segmented approach as implemented in IBRIX Fusion has some resiliency features. For example, parts of the name space can stay active even though one or more segments might fail. To gain additional reliency you can configure the segment servers with fail-over (High Availability - HA). IBRIX has a product called IBRIX Fusion HA, that allows you can also configure the the segment servers to fail over to a stand-by server. These servers can be configured in an active-active configuration so that both servers are being used rather than have one server in a standby mode where it's not being used.

EMC MPFS

Not to be outdone by other companies, EMC has developed a parallel file system that is appropriate for clusters. Originally, the file system was called HighRoad (also called MPFS) and used Fibre Channel (FC) SAN protocols to deliver data to the clients. This limited it's appeal to customers because every client had to have an FC adapter to connect to HighRoad. EMC then developed Highroad into **MPFSi** (Multi-Path File System) by adding **iSCSI** as the protocol for delivering data to and from the storage devices. This increased the appeal of MPFSi since you only needed a TCP network to get data to and from the clients.

The data transfer process of MPFS/MPFSi is similar to both Lustre and Panasas (more on those two file systems in Part 3 of this article). There is a **EMC Celerra** server that is the sole metadata manager for the file system. The data is stored on either the Celerra itself (for up to 30 clients) or on a SAN using **EMC Symmetrix** or **CLARiiON** units. The clients communicate with the metadata server using NFS and the storage using either iSCSI (TCP) or an FC SAN network. Typically people will use iSCSI to communicate to an FC switch that translates the iSCSI protocol to the FC protocol and sends the request to the Symmetrix or CLARiiON.

Each client runs two pieces of software, an MPFS agent and a SAN initiator (iSCSI or Fibre Channel). When a client makes a data request, the MPFS agent contacts the metadata server using NFS as the protocol and gets back a map of where the data is located (or to be located in the case of a write function). Then the client uses iSCSI to contact the storage in parallel using the map. So the file system behaves in much the same way as Lustre, Panasas, and pNFS. But from what I can tell, MPFSi is not an object based file system, but a more traditional file system that uses SAN based (block based) storage and uses a lock manager.

When MPFSi writes the data it stripes the data across multiple LUNs where each LUN is part of a RAID group (RAID-3 or RAID-5). This increases the parallelism for better throughput. EMC claims that MPFSi is suitable for applications that access data in 32KB chunks or greater. However, with a single metadata server, even with the performance of a Celerra, there might be a bottleneck in the metadata response for very large number of small blocks. But in general the Celerra is a very high performing system, so there should not be an issue scaling to hundreds of clients performing simultaneous file access.

To maintain compatibility with NFS and CIFS, the Celerra acts as a NFS and CIFS gateway for the clients. This allows the clients to access the file system without using the EMC supplied software.

Summary

The three file systems I covered in this part of the article are what you might think of as more traditional file system architectures in that they are block oriented file systems. However, this does not mean they are difficult to manage, difficult to configure, difficult to scale, or have bad performance. On the contrary, these file systems perform and scale very well and with some planning, are very easy to configure and manage.

In Part 3 of this article, I take a look at a somewhat new approach to parallel file systems - an object-based approach. Look for this part of the article to appear soon. I think you will be pleasantly surprised by what object-based storage can offer.

Dr. Jeff Layton has all of his degrees in Aeronautical and Astronautical Engineering with a focus on topics that require HPC. He has been working with HPC for over 20 years and has been working with clusters for over 10 years. He has written on a variety of topics including clusters, MPI, file systems, and performance tuning. He can be reached at laytonjb@gmail.com.