IBM Systems MAGAZINE

Issue Date: Open Systems edition
December 2006

Breaking the Bottleneck

Jaqui Lynch
jaqui.lynch@mainline.com

An overview of GPFS components and terminology

As more high I/O workloads move onto midrange systems, the capability to reliably provide high I/O bandwidth is becoming more critical. High-performance computing (HPC) has had this requirement for some time and it was addressed on the scalable parallel systems in 1998 with the initial releases of the General Parallel File-System (GPFS). GPFS has now moved out of the HPC arena and can be used for commercial multinode workloads to provide data access.

GPFS is a high-performance, shared-disk file-system that supports data access from multiple nodes to multiple terabytes of disk in a single file-system with support for files larger than 2 GB. It allows applications to simultaneously access data from any node that has GPFS mounted, while still providing performance, reliability and consistency.

Data Access

Normal UNIX* file systems provide access to files but lock at the file level. This means that one or more applications could be reading from the file, but only one can be writing. On high-write workloads, this seriously affects performance and throughput. GPFS addresses this by using byte-range locking to allow multiple tasks to read from, and/or write to, a file without serialization, while still maintaining consistency.

The data itself is stored in the GPFS, which is a striping file system that distributes disk blocks across all the disks. The striping is implemented in the file system, not the logical volume manager (LVM). Large files are divided into equal sized blocks and the consecutive blocks are placed on different disks in a round-robin fashion. To further improve throughput, GPFS makes extensive use of read-ahead and write-ahead caching. To provide data consistency, a token-management system is used to track which nodes own which locks. Data is prefetched into the GPFS buffer pool, where it's made available to the applications.

GPFS also provides an additional mode of access to the normal modes (read, write and execute). This new mode, called "c" or "control," specifies who manages the access control list (ACL) for a file apart from the defaults of owner and root.

Components and Terminology

The following is a list of useful GPFS components and terminology:

- Cluster - This consists of a number of nodes and network shared disks (NSDs) for management purposes.
- Storage Pool - This groups a file system's storage and allows a user to partition storage based on characteristics such as performance, locality and reliability.
- Node - This is an individual OS instance within a cluster.
- Nodeset - This is a group of nodes that are all running the same level of GPFS and working on the same file systems. A cluster can contain more than one nodeset, but a node can only belong to one nodeset. A nodeset can contain up to 32 file systems.
- Configuration manager - This has overall responsibility for correct operation of all the nodes and the cluster as a whole. It selects the file-system manager node for each file-system, determines succession if a file-system manager node fails and also monitors quorum. By default, quorum is set to 50% +1.
- File-system manager - Also referred to as the "stripe group manager," there can be only one at a time. This node maintains the availability information for the disks in the file system. In a large cluster, this may need to be a dedicated node that's separate from the disk servers.
- Stripe group - This is basically a collection of disks a file system gets mounted on.
- Token manager - This handles tokens for the file handles and synchronizes concurrent access to files, ensuring consistency among caches. The token-manager server also synchronizes certain pieces of GPFS metadata and some of the internal data structures. The token-manager server usually resides on the file-system manager and may need significant CPU power.
- Metanode - A node that handles metadata, also referred to as "directory block updates."
- Application node - This mounts a GPFS file system and runs a user application that accesses the file system.
- Network shared disk (NSD) - This component is used for global device naming and data access in a cluster. NSDs are created on local disk drives. The NSD component performs automatic discovery at each node to see if any physical disks are attached. If there are no disks, an NSD must be defined with a primary server. Best practices dictate that a secondary server should also be defined. I/O is then performed using the network connection to get to the NSD server that performs the I/O on behalf of the requesting node. NSDs can also be used to provide backup in case a physical connection to disk fails.

Planning for GPFS

All nodes in a GPFS cluster should be running the same release of the GPFS code if they're mounting the same file systems. It's recommended that they also run the same AIX* version and release level. Additionally, IBM has registered port 1191 with IANA for GPFS-specific traffic. GPFS also provides the capability to use openssh and openssl to secure communications. This means any firewalls need to allow access to these ports as required. The GPFS daemon is called mmfsd and needs to run on the nodes.

Before creating a cluster, several planning steps must be taken. Some of these can have significant performance impacts. Specifically, four parameters - pagepool, maxFilesToCache, maxStatCache and maxMBps - must be determined and attention needs to be paid to the correct selection of the blocksize and raid stripe-size. Where NSDs are being used, it's critical that all network tuneables (no - a) be reviewed to ensure there are sufficient network buffers and that ipqmaxlen is set to an optimal value. Since this is a high I/O file system, settings for memory and I/O (vmo and ioo) as well as pbufs should be carefully monitored and regularly reviewed.

One of the first steps is to define the GPFS root directory for definitions. If we use /usr/local/gpfs, then we would expect to see the configuration file (mmfs.cfg), nodelist and the disk description file (disk.desc) in that directory. Examples of the contents of these files are in Appendix A, and a sample planning spreadsheet can be viewed online at www.ibmsystemsmag.com/os/dec06/appendixb.

GPFS Configuration Settings

The following list highlights common GPFS configuration settings:

- Pagepool - The pagepool is used for I/O buffers to cache user data and indirect blocks. It's always pinned, and the default is fairly small. It's used to implement read/write requests asynchronously using the read-ahead and write-behind mechanisms. Increasing the pagepool increases the amount of data available in the cache for applications to use. This parameter is critical where applications perform significant amounts of random I/O.
- maxFilesToCache - This is the total number of different files that can be cached at any one time. This needs to be set to a large enough value to handle the number of concurrently open files and allow for caching those files.
- maxStatCache - This is additional pageable memory that's used to cache file attributes that aren't in the regular file cache. It defaults to 4 * maxFilesToCache.
- maxMBps (definition from the provided default mmfs.cfg) - maxMBpS is an estimate of how many MBps of data can be transferred in or out of a single node. The value is used in calculating the amount of I/O that can be performed to effectively pre-fetch data for readers and/or or write-behind data from writers. The maximum number of I/Os in progress concurrently will be 2 * min(nDisks, maxMBpS*avgIOtime/ blockSize), where nDisks is the number disks that make a filesystem; avgIOtime is a measured average of the last 16 full block I/O times; and blockSize is the block size for a full block in the file-system (e.g., 256K). By lowering this value, you can artificially limit how much I/O one node can put on all the virtual shared disk (VSD) servers, if there are lots of nodes that can overrun a few VSD servers. Setting this too high will usually not hurt because of other limiting factors such as the size of the pagepool, or the number of prefetchThreads or worker1Threads.
- Blocksize - The blocksize determines the largest file system size and should be set to the application buffer size or the stripe size on the raid set. If this is done incorrectly, performance will suffer significantly. Once the blocksize is set, the minimum space required for a file will be 1/32 of the blocksize, so this setting requires an understanding of file sizes as well.
- preFetchThreads - These are the maximum number of threads that can be dedicated to prefetching data for files that are read sequentially.
- Worker1Threads - The maximum number of threads that can be used for controlling sequential write-behind.
- Worker2Threads - The maximum number of threads that can be used for controlling other operations.

Each cluster needs a unique name and ID number, as well as a GPFS UID domain. Primary and secondary cluster configurations servers must be defined and the nodes within the cluster need to be determined, including which ones will be the token-manager server, file-system manager, application nodes, NSD nodes, etc. You need to assign each node a node number, name, IP address and full name. The IP address can be on a private network that's only used for GPFS communications and it's recommended that this be the case.

Disks then need to be attached, and a plan for what file systems will be striped across which disks should be determined. At this point, stripe sizes and block sizes need to be matched up to ensure optimum performance.

For each file system, you'll need to map out the file-system name, its manager node and the disks that it will be using. Additionally, you'll need to determine blocksize.

After the planning is done, the installation process is fairly straightforward and GPFS provides great failover and redundancy options. GPFS also provides a snapshot capability that allows for the creation of point-in-time copies that can be used to perform backups "while open" while maintaining consistency of the data.

Flexible and Scalable

The technology behind GPFS provides great flexibility and scalability, allowing for the addition and removal of disks while a file system is mounted and spreading the reads and writes to a single file system across multiple disks attached to multiple nodes. While GPFS planning may seem complex, it's in fact not that much more difficult than properly planning out a disk subsystem to be attached to a server. However, it provides the additional benefit of providing throughput far beyond what can

be provided by regular file systems and it provides that performance while also providing reliability and flexibility. With some basic planning, GPFS can significantly enhance your I/O bandwidth for high I/O applications and is well worth considering as part of any overall plan for I/O in the datacenter.

Appendix A

```
/usr/local/gpfs/nodelist

jaqui02:quorum
jaqui03:quorum
jaqui04:quorum

/usr/local/gpfs/disks.desc

jgpfslv1::::1:gpfsnsd1
jgpfslv2::::1:gpfsnsd2
jgpfslv3::::1:gpfsnsd3
jgpfslv4::::1:gpfsnsd4
jgpfslv5::::1:gpfsnsd5
jgpfslv6::::1:gpfsnsd6
jgpfslv7::::1:gpfsnsd7
jgpfslv8::::1:gpfsnsd8
jgpfslv9::::1:gpfsnsd9
jgpfslv10::::1:gpfsnsd10

/usr/local/gpfs/mmfs.cfg

maxFilesToCache 1000
maxMBpS 800
```

Jaqui Lynch, *a senior systems engineer focusing on System p* and Linux* at Mainline Information Systems, has worked in the IS industry for more than 28 years. She's been responsible for a wide variety of projects and OSs across multiple vendor platforms, including mainframes, UNIX systems, midrange systems and personal workstations. Jaqui can be reached at jaqui.lynch@mainline.com.*