On Making GPFS Truly General

DEAN HILDEBRAND AND FRANK SCHMUCK

Dean Hildebrand manages the Cloud Storage Software team at the IBM Almaden Research Center and is a recognized expert in the field of distributed and parallel file systems. He pioneered pNFS, demonstrating the feasibility of providing standard and scalable access to any file system. He received a BSc degree in computer science from the University of British Columbia in 1998 and a PhD in computer science from the University of Michigan in 2007. dhildeb@us.ibm.com

Frank Schmuck joined IBM Research in 1988 after receiving a PhD in computer science from Cornell University. He is a Distinguished Research Staff Member at IBM's Almaden Research Center, where he serves as a Technical Leader of the Parallel File Systems group and Principal Architect of IBM's General Parallel File System (GPFS). His research interests include storage systems, distributed systems, and fault tolerance. fschmuck@us.ibm.com

GPFS (also called IBM Spectrum Scale) began as a research project that quickly found its groove supporting high performance computing (HPC) applications [1, 2]. Over the last 15 years, GPFS branched out to embrace general file-serving workloads while maintaining its original distributed design. This article gives a brief overview of the origins of numerous features that we and many others at IBM have implemented to make GPFS a truly general file system.

Early Days

Following its origins as a project focused on high-performance lossless streaming of multimedia video files, GPFS was soon enhanced to support high performance computing (HPC) applications, to become the "General Parallel File System." One of its first large deployments was on ASCI White in 2002—at the time, the fastest supercomputer in the world. This HPC-focused architecture is described in more detail in a 2002 FAST paper [3], but from the outset an important design goal was to support general workloads through a standard POSIX interface—and live up to the "General" term in the name.

An early research prototype was based on an architecture similar to today's HDFS and pNFS, with a single server storing directories and handling metadata operations, redirecting reads and writes to a separate set of data servers. While scaling well for sequential I/O to large files, performance of metadata operations and small file workloads was typically worse—or at least no better—than for a traditional file server. In other words, this early prototype did not do the G in GPFS justice.

The improved design, which largely remains to this day, eliminates the single server bottleneck by managing both data and metadata in a fully distributed fashion across the whole cluster. Both data and metadata are stored on shared storage devices that are equally accessible from all cluster nodes. A distributed lock manager coordinates access to the file system, implements a cache consistency protocol, and provides necessary synchronization for proper POSIX semantics of individual file system operations. This allows each node to safely modify metadata directly on disk instead of going through a separate metadata server. Over the years, this original design has proven flexible enough to support numerous other application domains, such as cloud computing, network attached storage (NAS), and analytics, as shown in Figure 1.

The Basics

Since its beginnings, GPFS has been deployed on a wide range of cluster types and sizes, with the larger clusters serving the scientific computing needs of national research laboratories, small-to-medium-0sized clusters serving commercial HPC applications (e.g., oil exploration and engineering design), and clusters as small as two nodes, where GPFS may be used primarily for its fault-tolerance rather than scaling abilities (e.g., highly available database server). The original design was targeted at storage area networks (SANs). Support for network shared disk (NSD) access over TCP/IP and eventually InfiniBand via dedicated I/O server nodes was added for increased scalability and flexibility. This then also enabled support for

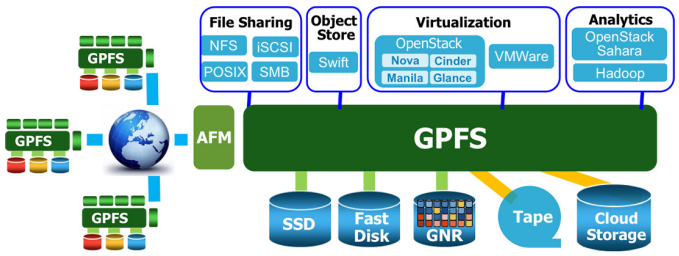

Figure 1: IBM Research has prototyped the use of GPFS with numerous APIs and storage devices within a single namespace, including the use of Active File Management (AFM) to share data across WANs.

commodity clusters consisting of server nodes with internal disks and SSDs.

The distributed locking architecture is a good match for scalable, general file-serving applications, especially for workloads consisting of a large collection of independent working sets (e.g., different users accessing different sets of files). Once an application has collected lock tokens that cover its working set from the distributed lock manager, it can read, cache, and update all data and metadata it needs independently, without any further interaction with other cluster nodes. In this manner, file access in GPFS can be just as efficient as a local file system, but with the ability to scale out by adding nodes and storage devices to meet growing bandwidth and capacity demands.

The first major feature added to GPFS for general, non-HPC workloads were read-only file system snapshots in 2003. This is particularly useful for serving a large user data set, since it allows an individual user to retrieve accidentally deleted files without requiring administrator assistance. Initially limited to 32 file system snapshots, the feature was later expanded to larger numbers and finer-grained snapshots, including writable snapshots of individual files.

At the same time, GPFS expanded its reach by extending its OS and hardware support from AIX on IBM Power servers to Linux on x86 and Power and later to Windows and Linux on mainframes. While the initial Linux release in 2001 did not allow mixing AIX and Linux in a single cluster, full heterogeneous cluster support was added a couple years later.

Larger and more diverse clusters also required continuing improvements in cluster management infrastructure. In 2004, the external cluster manager used early on was replaced with a built-in cluster manager using a more-scalable, hierarchical architecture. A subset of designated "quorum nodes" is responsible for ensuring system integrity by electing a unique cluster leader, monitoring the status of all other nodes in the cluster, and driving recovery in response to node failures. In 2007, support was added for rolling upgrades, allowing GPFS software to be upgraded one node at a time without ever shutting down the whole cluster, a critical feature for both HPC and general computing alike. Other features added in subsequent years, or actively being developed, include extended file attributes, a scalable backup solution, encryption, and compression.

Protecting Data, the Crown Jewels

The need for advanced data protection first became apparent in the HPC context, but growing data volumes means that reliability issues previously only seen in very large clusters now affect general-purpose storage systems as well. Experiences with traditional RAID controllers in the GPFS deployment for the ASC Purple supercomputer at Lawrence Livermore National Laboratory in 2005 had shown that when aggregating ~10,000 disks into a single system, very rare failure events become frequent enough that in these systems partial data loss became a real possibility. These included double disk failures, loss of a RAID stripe due to checksum errors during rebuild, off-track writes, and dropped writes. Furthermore, since disk failures were constantly occurring and their rebuild times were taking longer due to increased disk capacity, the whole system was being slowed down.

To eliminate the drawbacks of hardware storage controllers, in 2011 GPFS introduced an advanced, declustered software RAID algorithm integrated into its I/O servers, called GPFS Native RAID (GNR) [4]. Apart from simple replication, GNR offers a choice of Reed-Solomon erasure codes that tolerate up to three concurrent failures. Data, parity, and spare space are distributed across large numbers of disks, speeding up rebuild times with minimal impact on the foreground workload. Write version numbers and end-to-end checksums allow GNR to detect and recover from lost or misdirected writes, and care is taken to ensure related erasure code strips are placed in separate hardware failure domains, e.g., disk drawers, to improve availability. A background scrubbing process verifies checksum and parity values to detect and fix silent disk corruption or latent sector errors before additional errors might render them uncorrectable. The current implementation relies on a conventional, dual-ported disk enclosure filled with disks in a JBOD ("just a bunch of disks") configuration to provide redundant paths to disk in order to handle a failure of one of its primary I/O servers by a designated backup server. A current research project is exploring the use of internal disks by spreading data and parity across disks in different server nodes (network RAID).

Now that GPFS no longer relies on storage controller hardware, support was added for other "typical" storage controller features, including the ability for data to be replicated across different geographical locations for disaster recovery purposes. For shorter distances, synchronous replication is performed via standard GPFS data replication by creating a cluster that stretches across nodes at multiple sites. For larger distances, GPFS Active File Management (AFM), which was originally designed for file caching across wide area networks, can be

configured to asynchronously replicate files between two file systems at separate sites [5].

Pooling Your Data Without Getting Wet

In 2003, IBM introduced its Total Storage SAN File System (SAN-FS) as a "file virtualization" solution for storage area networks. From the outset, it was aimed at general commercial applications, but soon also branched out into data-intensive applications. By 2005, it became apparent that SAN-FS and GPFS catered to increasingly overlapping market segments, and IBM started an effort to merge the two product lines. This lead to GPFS adopting some of the unique features of SAN-FS, including native Windows support and, most notably, Information Lifecycle Management (ILM) through the concepts of storage pools and filesets [6].

Storage pools are a means of partitioning the storage devices that make up a file system into groups with similar performance and reliability characteristics. User-defined "placement policy" rules allow assigning each file to a storage pool so as to match application requirements to the most appropriate and cost-effective type of storage. Periodically evaluated "management policy" rules allow migrating files between pools as application requirements change during the lifecycle of a file. Policy rules may also change file replication; delete files; invoke arbitrary, user-specified commands on a selected list of files; or migrate rarely accessed data to an "external pool" for archival storage. The policy language allows selecting files based on file attributes, such as its name, owner, file size, and timestamps, as well as extended attributes set by the user. Data migrated to external storage either via policy or a traditional external storage manager is recalled on demand using the standard Data Management API (DMAPI).

Filesets provide a way to partition the file system namespace into smaller administrative units. For example, the administrator may define user and group quotas separately for each fileset or place limits on the total amount of disk space occupied by files in each fileset. GPFS also allows creating snapshots of individual filesets instead of a whole file system. Filesets also provide a convenient way to refer to a collection of files in policy rules.

Three Amigos: NFS, SMB, and Object

A parallel file system provides a powerful basis for building higher-level scalable, fault-tolerant services by running a service instance on each cluster node. Since all nodes have equal access to all file system content, an application workload can be distributed across the cluster in very flexible ways, and if one node fails, the remaining nodes can take over. This is easiest to implement for services that do not need to maintain any state outside of the file system itself. The canonical example is an NFS file server, due to the stateless nature of the NFSv3 protocol: servers running on different nodes in the cluster can simply export the same file system without requiring any additional coordination among the different servers. For client-side data caching, the NFS protocol relies on file modification time (`mtime`) maintained by the file system, but since mtime is not critical for HPC applications, GPFS only provided an approximate mtime value with eventual consistency semantics. This was soon fixed since approximate mtime is not sufficient to guarantee NFS close-to-open consistency semantics: if a reader opens a file after a writer has closed it, the reader should see the new file content.

An example of one of the first systems exploiting GPFS capabilities to provide a scalable file server solution is the Global Storage Architecture (GSA) service deployed within IBM starting in 2002. This replaced the existing AFS and DCE-based infrastructure, and is still actively used within IBM worldwide today. To help customers implement similar solutions, we added a "clustered NFS" (CNFS) feature to the base product, which manages NFS server failover and failback, including IP address takeover and `lockd` recovery.

While NFSv3 was nominally stateless, support for richer, stateful protocols like NFSv4 and the Windows Server Message Block (SMB) make it harder to turn a single server into a scalable, clustered solution. The simplest approach is to partition the namespace across the cluster and let only one node at a time serve files under each directory subtree. This avoids complexity, but limits load balancing since a "hot" subtree may overload its assigned node. A better approach is to add a clustering layer for managing distributed, protocol-specific state above the file system. The Clustered Trivial Database (CTDB) is just such a layer, developed in collaboration with the open source community, which integrates Samba servers running on different nodes within a cluster into a single, scalable SMB server. A scalable NFS and SMB file-serving solution based on this technology was made available as an IBM service offering in 2007 and as a NAS appliance in 2011.

One downside with layering protocol-specific cluster managers on top of a parallel file system is a lack of coordination between different protocols. For example, a file lock granted to an SMB client will not be respected by a client accessing the same file over NFS or by an application running on one of the nodes in the cluster accessing the file directly. So a third approach to implementing richer services is to add functionality to the file system for maintaining protocol-specific state consistently across the cluster. By taking advantage of features originally added for the GPFS Windows client, such as a richer ACL model and extensions to the GPFS distributed lock manager to implement Windows share-modes, the NFS server can implement features such as delegations, open modes, and ACLs—without a separate clustering layer. An immediate advantage is better coordination

between NFS clients and file access via SMB, FTP, and HTTP protocols.

In 2014, GPFS provided support for OpenStack Swift [7], which provides a stateless clustered layer for REST-based data access through protocols such as Swift and S3. Object storage systems have a lot in common with HPC, as they tend to have a large capacity (several PBs and larger), have a high number of simultaneous users, and span multiple sites. Many GPFS features have a direct benefit in this new domain, including scalable file creation and lookup, data tiering and information lifecycle management, GNR software-based erasure coding, and snapshots. Support for Swift does much more than provide a simplified method for data access; it includes several new features such as secure and multi-tenant access to data, role-based authentication, REST-based storage management, and a simplified flat namespace. All objects are stored as files, which enables native file access (e.g., Hadoop, NFS, SMB, POSIX) to objects without a performance-limiting gateway daemon. This capability means that objects within GPFS aren't in their own island of storage, but are integrated into a user's entire workflow.

Branching out from HPC to NAS and object storage provided an impetus for numerous improvements in GPFS to handle small-file and metadata-intensive workloads more efficiently. Allowing data of small files to be stored in the file inode instead of a separate data block improves storage efficiency and reduces the number of I/Os to access small file data. Keeping metadata for a large number of files cached in memory proved vital for good NAS performance, but put a greater load on the token server for each file system. GPFS therefore introduced a new token protocol that uses consistent hashing to distribute tokens for all file systems across any number of nodes in the cluster. Since GPFS records metadata updates in a per-node recovery log, metadata commits can be sped up by placing recovery logs in a dedicated storage pool of fast storage devices. As wider use of fast storage devices eliminates the devices itself as the performance limiting factor, the efficiency of the file system software stack as a whole becomes increasingly important, with particular attention required to minimizing overhead for synchronizing access to in-memory data structures on modern NUMA architectures.

Cloudy with a Chance of High-Performance

In 2014, there was a shift towards delivering open systems and software-defined-storage to customers. This shift was primarily motivated by customers frustrated with vendor lock-in, lack of ability to customize a solution, and also the desire (primarily for cost reasons) to leverage commodity hardware.

The OpenStack project fits well with this new way of thinking, offering an open cloud management framework that allows vendors to plug into myriad APIs. Beyond supporting the Swift object storage layer discussed previously, we have integrated support for Nova (which provisions and manages large networks of VMs), Glance (the VM image repository), and Cinder (which provides block-based storage management for VMs). Most recently, we delivered a driver for Manila, which allows users to provision file-based data stores to a set of tenants, and we are currently investigating support for Sahara, the easy to use analytics provisioning project.

Initially, there was some concern that using a file system for all of these services was not a good fit, but we found the more we integrate GPFS with all of the OpenStack services, the more benefits arise from using a single data store. Workflows can now be implemented (and automated) where files and data stores are provisioned and utilized by applications and VMs with zero-data movement as the workflow shifts from one management interface to the next. In another example, AFM can be leveraged to build a hybrid cloud solution by migrating OpenStack data to the cloud and then back again as needed.

Virtualization, and its use in the cloud, has also introduced a relatively new I/O workload that is much different than both NAS workloads, which primarily perform metadata-intensive operations, and HPC workloads, which do large writes to large files. VMs write small, random, and synchronous requests to relatively large (8–80 GB) disk image files [8]. To support this workload, we implemented a Highly Available Write Cache (HAWC), which allows buffering of small VM writes in fast storage, allowing them to be gathered into larger chunks in memory before being written to the disk subsystem. Further, we increased the granularity at which GPFS tracks changes to a file to avoid unnecessary read-modify-write sequences that never occurred previously in HPC workloads.

Moving forward, as public and private clouds continue to emerge, and more and more applications make the transition (including HPC applications), new requirements are emerging above and beyond being able to deliver high performance data access. One area that has a much different model from HPC is security and management. The "trusted root" model common in HPC datacenters is rarely acceptable, replaced by a more fine-grained and scalable role-based management model that can support multiple tenants and allow them to manage their own data. For management, supporting a GUI and REST-API is no longer just nice to have, as is an audit log for retracing operations performed on the system. As well, scaling existing monitoring tools and delivering higher-level insights on system operation will be key features of any cloud storage system.

Another area of interest is data capacity, where HPC has traditionally led the way, but cloud computing is catching up and is possibly poised to overtake HPC in the near future. For example, some cloud datacenters are scaling by up to 100 PB per year. The challenge for GPFS is less about figuring out how to store all that data (the GPFS theoretical limit on the number of files in a single

file system is 2^{64}, after all), but more about providing a highly available system at scale that can be efficiently managed. For the issue of storing trillions of files, the classical hierarchical file-system directory structure, no matter how scalable, is not an intuitive method for organizing and finding data. GPFS support for object storage improves this by introducing both a simpler flat namespace as well as a database for faster indexing and searching. For the issue of high availability, the impact of catastrophic failure must be limited at any level of the software and hardware stack. To do this, failure domains must be created at every level of the software stack, including the file system, such that when a catastrophic failure occurs in one failure domain, the remaining failure domains remain available to users.

Analyze This (and That)

When analytics frameworks like Hadoop (and its file system HDFS) started, they focused on a specific class of problems that exploited locality to scale I/O bandwidth. So to support analytics, a Hadoop connector was implemented and a few key changes were made to GPFS to support storage rich servers. First, we increased the maximum replication from two to three, which was primarily a testing effort, and ensured one of those replicas was stored on the local server. Second, the traditional parallel file system method of striping small (1 MB) chunks across the entire cluster would overflow network switches, so block groups were introduced to allow striping in much larger chunks (128 MB). Third, failure groups were extended to understand network hierarchies, instead of just the flat networks common in HPC.

Recently, a shift has occurred that brings new life to running analytics on the original GPFS architecture. The combination of cheaper, fast networks with the emergence of new analytic workloads such as Hive and HBase mitigates the benefit of data locality in many cases. These workloads perform smaller and more random I/O, benefiting from the scalable metadata and optimized data path in GPFS. In addition, support for POSIX semantics (and therefore in-place updates) allows a wide range of such analytics workloads to be developed.

Conclusion

GPFS represents a very fruitful and successful collaboration between IBM Research and Product divisions, with customer experiences providing a rich source of interesting and challenging research problems, and research helping to rapidly bring advanced technology to the customer. Living up to the G in GPFS has thus been a fun if not always an easy or straightforward journey.

Looking ahead, GPFS will continue to evolve and strengthen its support for all types of enterprise workloads, enabling users to have a single common data plane (aka "data lake") for all of their application requirements. In HPC, GPFS has recently been chosen as the file system in two petaflop supercomputers set to go online in 2017 [9], whose "data-centric" design is a milestone in the path towards exascale computing. Simultaneously, GPFS's trip into the cloud is yielding exciting new features and functionality addressing new and evolving storage needs.

References

[1] IBM Spectrum Scale: www-03.ibm.com/systems/storage/spectrum/scale.

[2] D. Hildebrand, F. Schmuck, "GPFS," in Prabhat and Q. Koziol (eds), *High Performance Parallel I/O* (Chapman and Hall/CRC, 2014), pp. 107–118.

[3] F. Schmuck, R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02),* 2002: https://www.usenix.org/legacy/events/fast02/schmuck.html.

[4] IBM, GPFS Native RAID Version 4 Release 1.0.5, Administration, 2014: http://publib.boulder.ibm.com/epubs/pdf/c2766580.pdf.

[5] M. Eshel, R. Haskin, D. Hildebrand, M. Naik, F. Schmuck, R. Tewari, "Panache: A Parallel File System Cache for Global File Access," in *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '10),* 2010: https://www.usenix.org/legacy/events/fast10/tech/full_papers/eshel.pdf.

[6] IBM, General Parallel File System Version 4 Release 1.0.4, Advanced Administration Guide, (SC23-7032-01), 2014: http://publib.boulder.ibm.com/epubs/pdf/c2370321.pdf.

[7] IBM, "A Deployment Guide for IBM Spectrum Scale Object": http://www.redbooks.ibm.com/abstracts/redp5113.html?Open.

[8] Vasily Tarasov, Dean Hildebrand, Geoff Kuenning, Erez Zadok, "Virtual Machine Workloads: The Case for New Benchmarks for NAS," in *Proceedings of the Eleventh USENIX Conference on File and Storage Technologies (FAST '13),* 2013: https://www.usenix.org/conference/fast13/technical-sessions/presentation/tarasov.

[9] IBM and Nvidia to Build 100 Petaflop Supercomputers, November 2014: http://www.vrworld.com/2014/11/14/ibm-and-nvidia-to-build-100-petaflop-supercomputers.

Do you know about the USENIX Open Access Policy?

USENIX is the first computing association to offer free and open access to all of our conference proceedings and videos. We stand by our mission to foster excellence and innovation while supporting research with a practical bias. Your financial support plays a major role in making this endeavor successful.

Please help to us to sustain and grow our open access program. Donate to the USENIX Annual Fund, renew your membership, and ask your colleagues to join or renew today.

www.usenix.org/annual-fund