# GPFS : A Shared-Disk File System for Large Computing Clusters

Frank Schmuck and Roger Haskin, IBM Almaden Research Center, San Jose, CA

**Pramod Kundapur Nayak**
UB# 37646792

## SUMMARY

With the thirst for higher computing power in demand, cluster computing has become a trend. Fault-tolerance, boundless computing power, unrestrained storage capacity being prime requirements of a reliable system, cluster computing has been of keen interest among researchers. This paper focuses on the storage aspect of cluster computing by introducing GPFS (General Parallel File System), a file system package from IBM, which provides functionalities similar to standard POSIX file system.

To summarize:

- GPFS appears to work like traditional POSIX file system but provides **parallel access to files.**
- Enhanced performance achieved through **data striping at block level** across all disks in file system.
- Supports upto 4096 disks of upto 1TB each, providing a total of 4 petabytes per file system.
- Both file data and metadata of any disk is accessible from any node through disk I/O calls. Further, GPFS facilitates parallel flow of both data and metadata from node to disk.
- Highly reliable with fault-tolerance and replication mechanism.

This paper highlights GPFS's answers to performance, scalability, concurrency and fault-tolerance issues of large file system and provides bird's eye view of GPFS.

## GPFS ARCHITECTURE – OVERVIEW

There are three main types of nodes: file system, storage and manager. Any node can perform one of the above functions

**File system node**: co-ordinates administrative task

**Manager nodes** exist one per file system. Different manager nodes are global lock manager, local lock manager, allocation manager etc

**Storage node** implement shared access to files, co-ordinate with manager node during recovery and facilitate both file data and metadata to be striped across multiple storage nodes.

Some other auxiliary nodes include:

**Metanode**: A node is dynamically elected as metanode for centralized management of file metadata. Election of metanode is facilitated by *token server*.

**Token Server**: A token server tracks all tokens granted to all nodes in the cluster. A token granting algorithm is used to reduce cost of token management.

## GPFS AND LARGE FILE SYSTEM ISSUES

The highlight of this paper shows how GPFS addresses the large file system issues.

- ✓ **High throughput**
  GPFS achieves high throughput by striping data across multiple disks and nodes.
  GPFS uses large block size of 256 bytes than small block sizes to increase transfer speed.
  A trade-off has to be chosen by the administrator to choose between space utilization or throughput.
  Implements extensible hashing and hence provides efficient lookup for large directories by requiring only single directory block access, no matter how huge the directory file is.

- ✓ **High Availability**
  Recovery is implemented by maintaining logs on each node for every file system which is readable by any node thus facilitating recovery.

- ✓ **Parallelism**
  Synchronization achieved through
  *Distributed locking* : inquire with other nodes before acquiring locks, also guarantees POSIX semantics.

*Centralized management* of locks , where every node contacts a designated node to perform conflicting read / update.

Lock management performed by assigning *local lock managers* in every node and assigning a single node as a *global lock manager* to co-ordinate locks between local lock managers.

*Byte-range locking* to facilitate concurrent write to different parts of the same file.

*Data shipping* mode, primarily used by MPI/IO library, is implemented to facilitate fine-grain sharing for applications that do not require POSIX semantics by allocating data blocks to nodes.

Parallel access to files by multiple nodes results in concurrent updates to inode and hence a *metanode* is assigned, where only the *metanode* updates the inode on the disk and the other nodes trying to modify the file send updates to this *metanode*.

Operations such as disk space allocation also needs to be synchronized between nodes, which is done by allocating *allocation manager* which maintains and initializes free space statistics with the aid of allocation maps.

- ✓ **Fault-tolerance**

  *Disk failures*:

  *File system manager* is informed about the disk failure by the node that detects it and is marked.

  GPFS facilitates *replication* and hence enables recovery on disk failures.

  *Node failures*:

  Restore metadata that was being updated by the failed node and release any tokens held by it, finally appoint replacement and log recovery can be performed by any other live node.

  *Communication failures*:

  Communication failures are handled by GPFS by fencing nodes that are no longer members of the group from accessing shared disks.

- ✓ **Scalability**

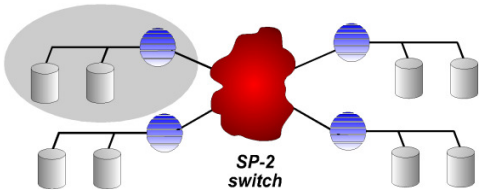
  GPFS can be scaled by adding and removing disks from the existing file system.

  File system manager co-ordinates these activities and updates metadata as required.

## Comparison – GPFS Vs Local and Distributed file system

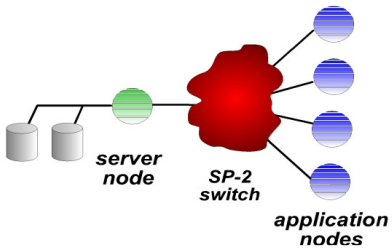
A brief comparison of different file systems with GPFS is shown below. [3]
(Please refer the cited presentation for more details. This is an excerpt from the cited presentation)

### Native AIX File System (JFS)



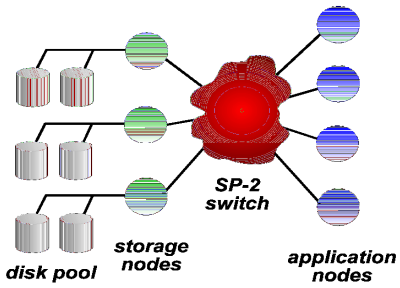

- Applications can only access files on its own node
- Applications must do their own data partitioning

### Distributed File System




- Application nodes share files on server node
- Fast LAN implemented using switches
- Server node is performance and capacity bottleneck

### GPFS Parallel File System




- GPFS file systems are striped across multiple disks on multiple storage nodes
- Independent GPFS instances run on each application node
- GPFS instances use storage nodes as "block servers" - all instances can access all disks

## Strengths

The strengths of GPFS lies in overcoming the issues of large file system and has been already discussed above.

## Weakness

Extending my study of paper, I came across a research work material titled "*On evaluating GPFS*" by Alejandro Calderon at HLRS [2]. Here are some of the excerpts from the presentation which highlights some drawbacks observed in GPFS.

The first evaluation is for ***metadata*** by using *fdtree*. *Fdtree* is software used to test metadata performance of a file system.

The test setup involved was to create several directories and files at several levels.

The test concluded with contention found at directory level. If *more than one process from a parallel application* write data, use each one on different subdirectories in GPFS workspace for efficiency.

Follow the reference link for more detailed explanation on the experiment [2].

The second experiment was ***bandwidth evaluation*** using *Bonnie*, *Iozone* etc

*Bonnie* is a file system benchmark program which tests the speed of file I/O using standard C library calls.

*Iozone* is a file system benchmarking tool.

The test setup involved reading and writing a 2 GB file, evaluating bandwidth performance by using several nodes with several file size and access size.

The conclusions from above experiment are as follows:

- The bandwidth decreases with *number of processes per node*
- Encouraging usage of MPI-IO API because this helps achieve users get more bandwidth.
- Metadata management does not cause main bottleneck with large files
- The performance may decrease if several processes try to write to the same file but on independent areas.
- A magic number 128KB was observed at which the algorithm changes to give high bandwidth.

Some of these interesting results with graphs and test cases can be read at the following link : www.arcos.inf.uc3m.es/~dcio/ALEX-gpfs.ppt

## Learning's

- ✓ The batch of work handed to each node must be sufficiently small and of equal size to balance work distributed to each node and exploit available bandwidth.
- ✓ GPFS makes efficient usage of computing resources by restricting management functions to a set of designated administrative nodes.
- ✓ Pre-fetching is implemented when multiple accesses to inodes in same directory are made thus speeding up directory scans.
- ✓ The paper ends with a crucial example signifying the importance of considering independent disk failures while designing file system. GPFS could rebalance data in the file system without loss in any form.

## Current application

- Six out of ten most powerful supercomputers in the world use GPFS
- Installed at several customer sites from few nodes with less than a TB to 512 nodes with 140TB of disk in 2 file systems.

## Class discussion

An excerpt of class discussion is pictured below.

**A query regarding Round-robin assignment of data blocks to each node where all read and write operations are forwarded to the node responsible for a particular data block would cause bottle neck in the system was raised.**

The presenter defended by explaining Data shipping, and token conflicts caused due to byte-range granularity. A experiment on comparing throughput with data shipping and byte range granularity explained in the paper was revisited to support the discussion.

**Another query said, multiple nodes can access the same file provided each node acquires a token for that particular byte range area, then how does token conflict arise even if individual write operations do not overlap.**

The class concluded that, GPFS uses byte-range tokens to synchronize data block allocation as well and hence token conflicts occur even if individual write operations do not overlap.

**Doubts regarding metadata update issue from multiple access to file via byte range tokens were raised.**

The presenter revisited the metanode feature in GPFS and supported with chalkboard examples.

## References

1. *GPFS: A Shared-Disk File System for Large Computing Clusters* , Frank Schmuck and Roger Haskin , IBM Almaden Research Center San Jose, CA
http://www.almaden.ibm.com/StorageSystems/projects/gpfs/Fast02.pdf

2. Research work that has been done at HLRS by Alejandro Calderon "*On evaluating GPFS*"
www.arcos.inf.uc3m.es/~dcio/ALEX-gpfs.ppt

3. *Architectural and Design Issues in the General Parallel File System* , Benny Mandler , IBM Research Lab in Haifa
http://www.cs.technion.ac.il/Courses/OperatingSystemsStructure/new_slides/GeneralParallelFileSystem13.ppt