

How Expensive is GPFS? *

By

Nancy Tran

National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana Champaign

* This work was supported in part by the National Science Foundation under grants NSF ACI 01-22296, NARA NSF 02-02 GPG, and the State of Illinois.

1 Goal

This study is the first phase of our investigation on the performance of the IBM General Parallel File System (GPFS). We use application total I/O time (for all open, close, read, write operations) as a preliminary performance yardstick because it represents the aggregate end-to-end I/O delay. To assess how expensive GPFS is, we focus on estimating 3 metrics and correlating them with application I/O characteristics:

- delay caused by GPFS processing on a single local node
- delay caused by the network and processing of network protocol
- benefits/overhead when striping over the SAN (Storage Area Network) disk arrays

2 Methodology

In this 1st phase of our investigation, we focus on applications executing in a single host machine. We compare the I/O times of applications for five different configurations (illustrated in Figures 1-5). By altering one factor from one configuration to the next, we can determine the effects of each factor on application I/O time. Subsequently, we use results from successive configurations to derive estimates for our target performance metrics.

2.1 Configurations

1. Linux file system (LinuxFS) using one local SCSI disk
2. Linux file system using a directly attached Storage Area Network (SAN) disk, i.e., a single logical unit (LUN) was allocated on the SAN disk arrays to represent one disk.
3. IBM's General Parallel File System (GPFS) using 3 SAN disk configurations listed below. We varied the *number* of LUNs allocated from the SAN disk arrays and the *physical attachment* (direct or remote) of the SAN to the application node – the PC on which the application executes.
 - (a) SAN disk (1 LUN allocation) directly attached to the application node
 - (b) SAN disk (1 LUN allocation) remotely attached to the application node
 - (c) SAN disk (3 LUNs allocation) remotely attached to the application node

3 Experimental Environment

3.1 Hardware Setup

Our test cluster is composed of 12 Intel P3 547 MHz dual-processor PCs. Each PC is equipped with 1 GB of main memory, a 9 GB SCSI disk, a 33 MHz PCI bus at 132 MB/sec peak throughput. One PC node has been chosen as the front end node for program development and job submission.

Three other PC nodes are attached to a set of Brocade SAN switches connecting to a pool of disk arrays. They serve as GPFS disk server nodes. Each node, installed with an adapter card that

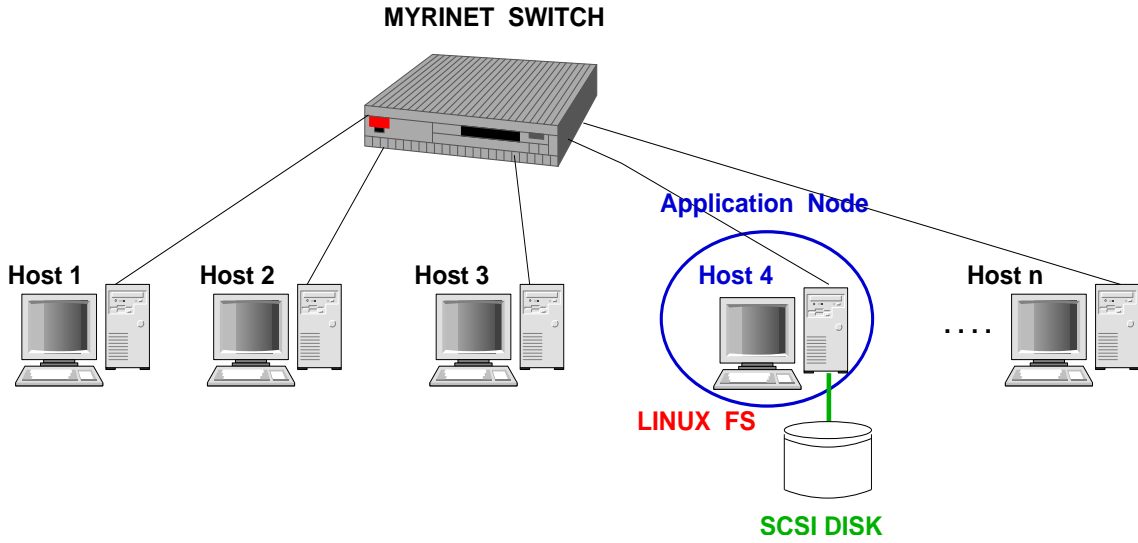


Figure 1: Config 1: Application on Host4 Using LinuxFS and 1 SCSI Disk

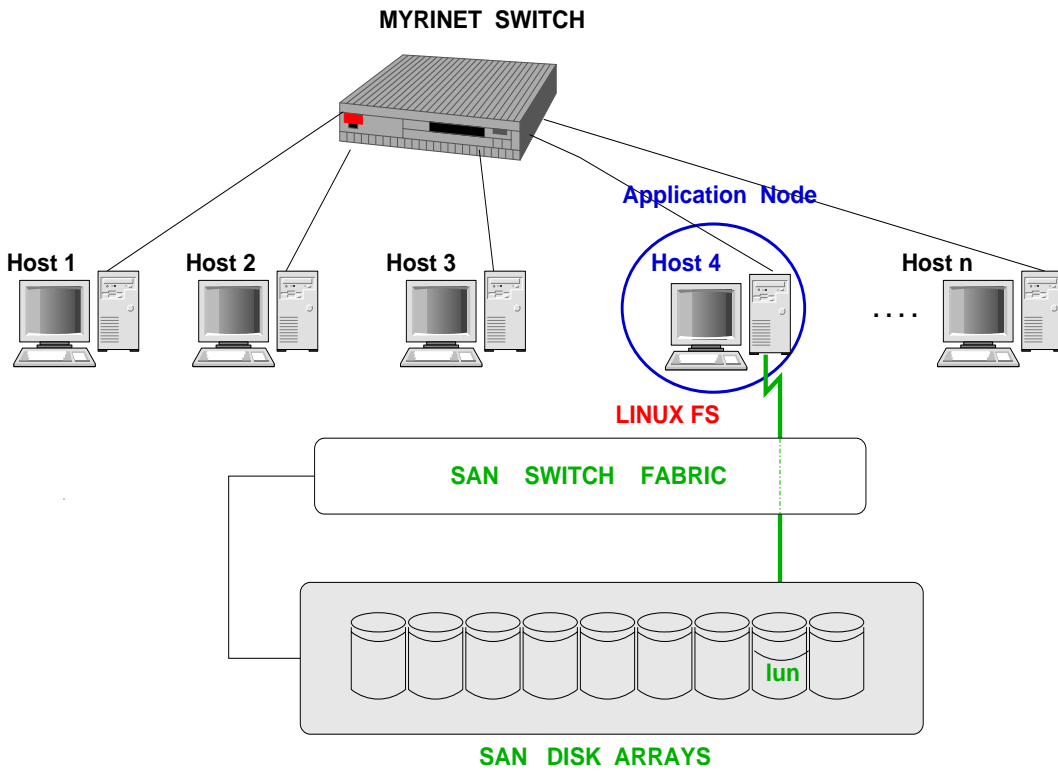


Figure 2: Config 2: Application on Host4 Using LinuxFS and a Directly Attached SAN Disk

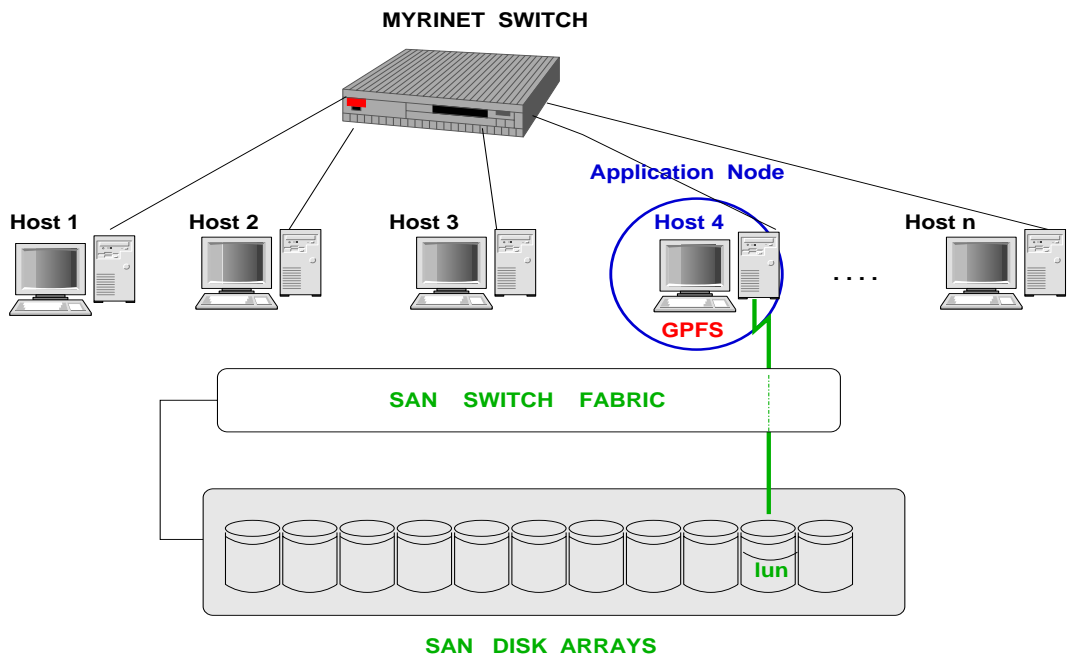


Figure 3: Config 3a – Application on Host4 Using GPFS and a Directly Attached SAN Disk

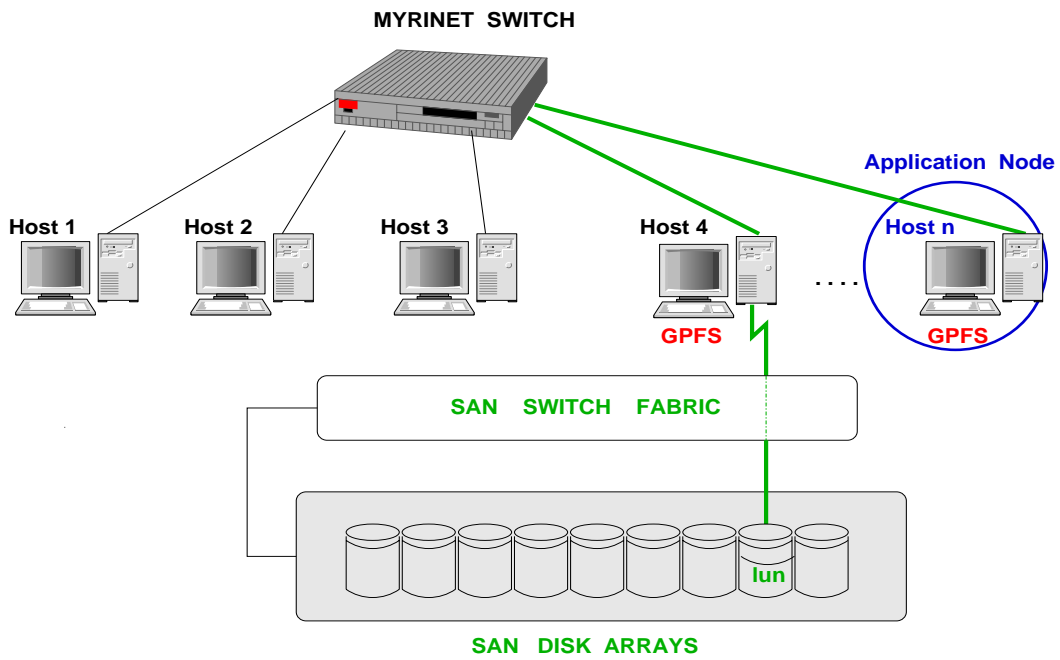


Figure 4: Config 3b – Application on Hostn Using GPFS and a Remotely Attached SAN Disk

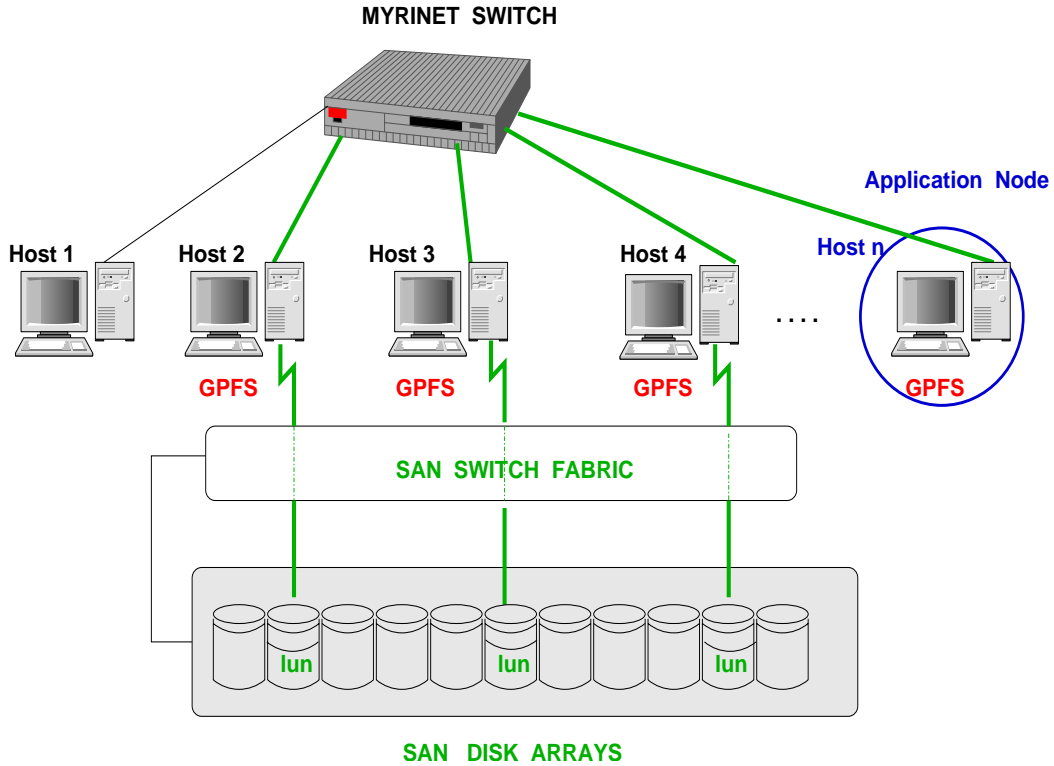


Figure 5: Config 3c – Application on Host_n Using GPFS and 3 Remotely Attached SAN Disks

runs the SCSI over fiber channel protocol, is allocated with a single SAN logical disk unit (LUN) of 64 GB. The link between a PC node and the closest SAN switch has a peak bandwidth of 1 gbps.

The cluster is connected via a 16-port Myrinet-2000 switch, equipped with duplex links at a speed of 1.28 gbps (160 MB/sec). We note that this maximum is limited by the PCI bus of 132 MB/sec, i.e., 82% of the peak.

3.2 Software Setup

The cluster was installed with Redhat Linux 7.3 operating system and the IBM General Parallel File System (GPFS) Release 1.2 for Linux. Applications executing from a node in the cluster can access a shared file allocated on the SAN disks via GPFS as shown in Figure 5. GPFS stripes data over a total storage of 192 GB distributed over 3 disks (LUNs). For a given node in the cluster, GPFS uses TCP/IP over Myrinet to exchange data among nodes.

4 Experiments

We measured the execution times and I/O times of 2 applications of the Cactus Code, Wave and ADM. Wave is a simple code to simulate a 3-D scalar field. ADM is a code that evolves the standard 3+1 form (ADM form) of the Einstein equations, decomposing spacetime into layers of 3-D space-like hypersurfaces. The major difference between ADM and Wave is that ADM incurs a larger computation time per iteration.

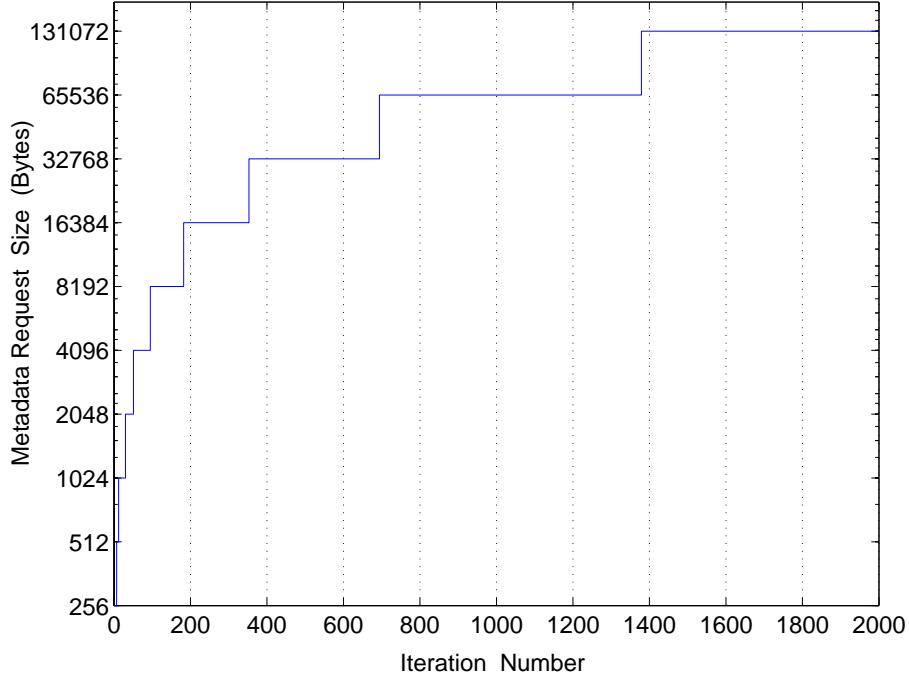


Figure 6: Wave – Dominating Read Request Behavior of HDF5 Metadata

We tested 2 different I/O methods, HDF5 and FlexIO. The Cactus code allows one to vary I/O request sizes and I/O intensity by simply changing the grid size and the number of iterations.

With *HDF5*, the Cactus code opens and closes files for each iteration. At the start of each iteration, a group of reads are initiated for each file to retrieve the metadata, followed by a write request of 1 MB for the results, and a series of writes to update the metadata. The group of metadata reads is dominated by a particular request that starts with 256 bytes at iteration 1. As the number of iterations increases, the size of this request doubles at *increasing intervals in a step-wise manner* as shown in Figure 6, which is plotted using a logarithmic scale on the Y axis. For example, in iteration 250, the request size grows to 16 KB (16384 bytes). Similar behavior is observed with the metadata writes.

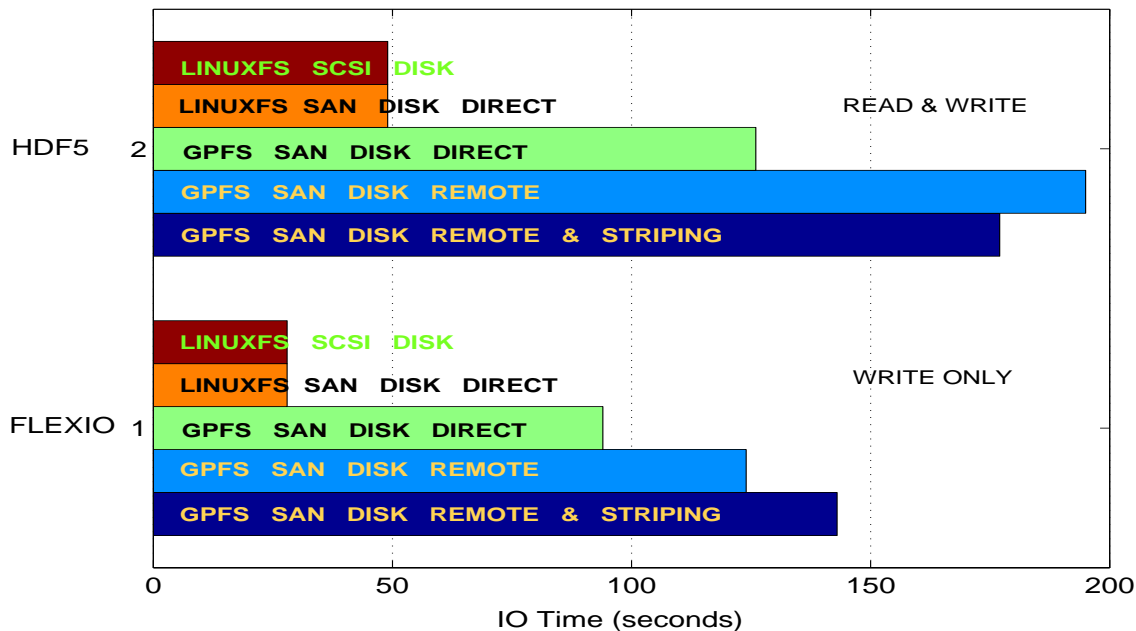
Unlike HDF5, the FlexIO library in the Cactus Code Release 4.12 issues only write requests, 2 per iteration – a request of 8 KB followed by a large request around 1 MB. In this release, file reads and opens/closes in each iteration are eliminated, resulting in a substantial reduction in I/O activity.

For all experiments in this study, we used a grid size of $50 \times 50 \times 50$ and 2000 iterations to evolve the Cactus applications, producing a single file of ≈ 2 GB/experiment. We computed the application I/O time from timestamps of Cactus I/O system calls recorded via the *strace* tool in Linux.

4.1 Baseline

To facilitate comparisons, all measurements for our experiments are *assessed relative to a baseline* chosen to be the application I/O time for configuration 2 (LinuxFS with a directly attached SAN disk). The reason for our choice is that configuration 2 should incur the smallest application I/O time.

Figure 7: Wave I/O Times under HDF5 and FLEXIO



4.2 Cactus Wave

Figure 7 shows the total I/O time of Cactus Wave under LinuxFS and GPFS with different disk configurations. Tables 1 and 2 show the corresponding *percentage I/O time improvement/degradation relative to the baseline*. This percentage is computed as:

$$\frac{\text{Application I/O Time for a target Config} - \text{Baseline}}{\text{Baseline}} \times 100$$

For all configurations, the total IO times with FLEXIO are smaller than those of HDF5 because FLEXIO in the latest Cactus release (4.12) has eliminated the read, open and close operations in each iteration. For 2000 iterations, the total number of I/O operations produced by Wave HDF5 is about 28,000, compared to 4002 operations produced by Wave FLEXIO.

CONFIGURATION (Baseline: Configuration 2)	Total Exec. Time (sec)	IO Time (sec)	Percentage IO Time Relative to Baseline			
			Improvement (+) or Degradation (-)	GPFS Overhead	Network Delay & Overhead	Striping Benefits
1. LinuxFS SCSI Disk	551	49	0%			
2. LinuxFS SAN Disk Direct 1 LUN	551	49	0%			
3a. GPFS SAN Disk Direct 1 LUN	647	126	-158%	158%		
3b. GPFS SAN Disk Remote 1 LUN	716	195	-298%		140%	
3c. GPFS SAN Disk Remote 3 LUNs	690	177	-261%			37%

Table 1: Wave via HDF5 - I/O Time Comparisons among Various Configurations

CONFIGURATION (Baseline: Configuration 2)	Total Exec. Time (sec)	IO Time (sec)	Percentage IO Time Relative to Baseline			
			Improvement (+) or Degradation (-)	GPFS Overhead	Network Delay & Overhead	Striping Benefits(+) Overhead(-)
1. LinuxFS SCSI Disk	533	28	0%			
2. LinuxFS SAN Disk Direct 1 LUN	532	28	0%			
3a. GPFS SAN Disk Direct 1 LUN	620	94	-236%	236%		
3b. GPFS SAN Disk Remote 1 LUN	650	124	-343%		107%	
3c. GPFS SAN Disk Remote 3 LUNs	665	143	-411%			-68%

Table 2: Wave via FLEXIO - I/O Time Comparison among Various Configurations

4.2.1 LinuxFS – SCSI Disk versus SAN Disk

With an I/O workload that includes several *reads and large sequential writes*, Wave’s I/O time and execution time under HDF5 using a directly attached SAN disk is almost the same as that using a SCSI disk (all numbers are rounded to seconds). The same behavior was observed with FLEXIO.

4.2.2 GPFS – Comparisons among Various SAN Disk configurations

- *GPFS overhead* was computed by comparing the application I/O times of GPFS (row 3a in tables 1 and 2) with those of LINUXFS (row 2 in tables 1 and 2). Relative to the baseline, GPFS overhead is 158% for HDF5, 236% for FLEXIO.

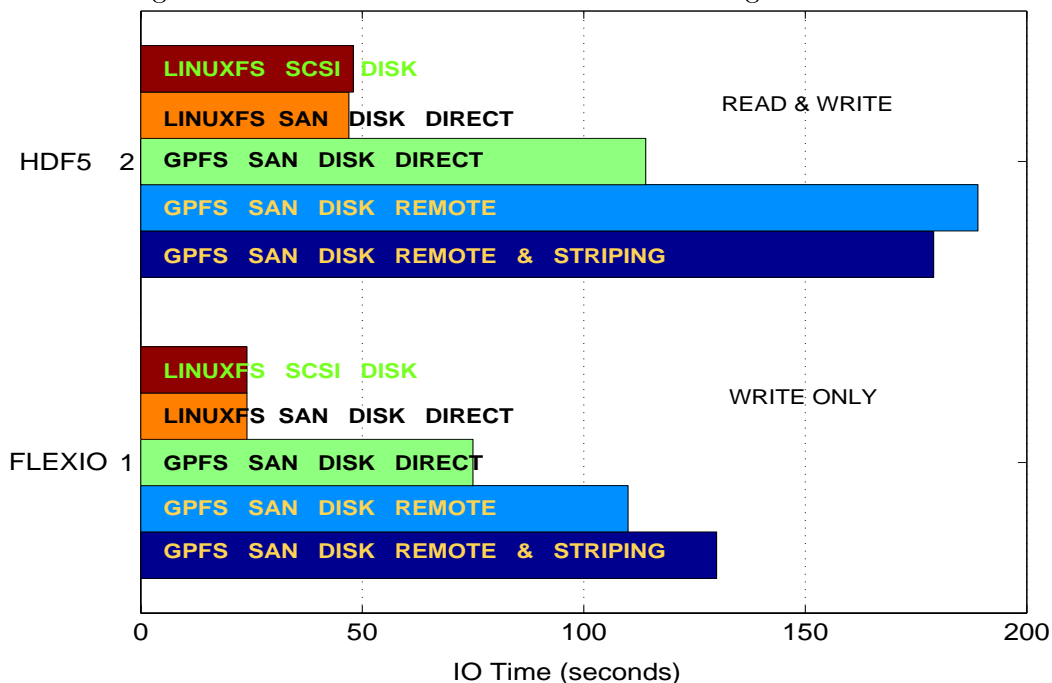
The smaller overhead with HDF5 can be attributed to GPFS caching at the application nodes – there were cache hits for read requests. However, unlike HDF5, FLEXIO issued only write requests. These requests were unnecessarily cached by GPFS as there were no reads, increasing the overhead.

- For *I/O delay due to network processing and latency*, HDF5 incurred a higher delay of 140%; FLEXIO incurred 107% delay. These delays were estimated by taking the differences between the percentages IO time improvement/degradation over the baseline for rows 3a and 3b in tables 1 and 2. Again, the higher delays are caused by the additional read, open, and close operations exchanged over the network.
- With HDF5, *GPFS striping* results in 37% improvement; This improvement is mainly due to parallel data transmission to 3 different nodes and parallel disk I/O to 3 different SAN disks (LUNs).

4.3 Cactus ADM

Except for higher computation times used to evolve the Einstein equations, ADM’s I/O time comparisons shown in Figure 8, tables 3 and 4 are similar to those of Cactus Wave. Our experiments with ADM allow us to verify results for Cactus Wave.

Figure 8: Execution Times of Cactus ADM Using HDF5 and FLEXIO



4.4 Conclusions

GPFS accumulates significant delay as the number of I/O operations increases. For example, each open request requires data structures to be allocated; each read/write operation must be accessed through the GPFS cache; each close operation flushes the dirty blocks in the GPFS cache to the disks – an expensive operation when close requests are issued several times.

Delay caused by GPFS can be mitigated when read requests can benefit from cache hits. In contrast, caching write-only requests in GPFS introduces unnecessary delays.

For configurations 3b and 3c, cache misses will result in *data exchanges* between the SAN disk server nodes to perform physical I/O and the application client node caching the data. GPFS will

CONFIGURATION (Baseline: Configuration 2)	Total Exec. Time (sec)	IO Time (sec)	Percentage IO Time Relative to Baseline			
			Improvement (+) or Degradation (-)	GPFS Overhead	Network Delay & Overhead	Striping Benefits
1. LinuxFS SCSI Disk	2847	48	-2.2%			
2. LinuxFS SAN Disk Direct 1 LUN	2846	47	0%			
3a. GPFS SAN Disk Direct 1 LUN	2915	114	-143%	143%		
3b. GPFS SAN Disk Remote 1 LUN	3009	189	-302%		159%	
3c. GPFS SAN Disk Remote 3 LUNs	2943	179	-281%			21%

Table 3: ADM via HDF5 - I/O Time Comparison among Various Configurations

CONFIGURATION (Baseline: Configuration 2)	Total Exec. Time (sec)	IO Time (sec)	Percentage IO Time Relative to Baseline			
			Improvement (+) or Degradation (-)	GPFS Overhead	Network Delay & Overhead	Striping Benefits (+) Overhead (-)
1. LinuxFS SCSI Disk	2865	24	0%			
2. LinuxFS SAN Disk Direct 1 LUN	2863	24	0%			
3a. GPFS SAN Disk Direct 1 LUN	2928	75	-212%	212%		
3b. GPFS SAN Disk Remote 1 LUN	2973	110	-358%		146%	
3c. GPFS SAN Disk Remote 3 LUNs	2953	130	-442%			-84%

Table 4: ADM via FLEXIO - I/O Time Comparison among Various Configurations

incur large network delays when the number of read requests is large, the request sizes are small (requiring large network processing overhead), and the request pattern is non-sequential (causing cache misses).

4.5 Future Work

This preliminary study provides a high-level evaluation of the overall end-to-end performance of applications running atop GPFS on a single PC node, producing rough estimates of GPFS overhead, network latency and processing overhead, and striping benefits/degradation. The next phase of our investigation is to experiment with applications running on multiple nodes via MPI.

In addition, better understanding on the time evolution of I/O exchanged over the network is needed to address the following issues:

- How does I/O behave along the round-trip path of I/O requests in a client-server model – from the application node (client), to the network, the storage servers, the disks, back to the network and finally to the client?
- Can we identify and characterize I/O patterns for each component? Knowledge of these patterns can provide insights to efficient distributions of I/O in a cluster environment.
- Where are the major bottlenecks?
- What effects does striping has on write-only versus read-write workloads?

To investigate these issues, we plan to extract data at the device driver level for each component, investigate I/O behaviors modulated by each component, and characterize the data via time series analysis.