

THE PRIOMark

PARALLEL I/O-BENCHMARK

Michael Krietemeyer, Daniel Versick, Djamshid Tavangarian
[michael.krietemeyer | daniel.versick | djamshid.tavangarian]@informatik.uni-rostock.de
Chair of Computer Architecture, Department of Computer Science
University of Rostock
Albert-Einstein-Strasse 21, D-18059 Rostock, Germany

ABSTRACT

This paper introduces the novel I/O benchmark PRIOMark that measures file system and disk I/O performance of modern computer systems. It characterizes the file system performance of single and distributed systems by means of two different file system interfaces. Disk I/O performance is measured using the operating system specific raw-devices. PRIOMark features the possibility of a complex workload definition in order to specify workloads for many different application domains. The output format simplifies comparability of benchmark results of different architectures. Additionally, the paper exemplarily illustrates the use of PRIOMark by comparing the file system performance of three file systems (NFS, GFS and PVFS) on a cluster with 12 processors.

KEY WORDS

I/O Performance, MPI, Parallel I/O, NFS, GFS, PVFS

1. Introduction

As the performance gap rapidly increases between CPU and I/O devices in modern computer systems, the measurement and optimization of I/O performance of computers is of great interest. Modern computer systems use a hierarchical storage architecture called memory hierarchy. Each level of this hierarchy is larger and slower than higher levels. The memory hierarchy typically consists of CPU registers, Level 1 cache, Level 2 cache, main memory and disk storage. Since the bandwidth between main memory and disk storage is the smallest between the described hierarchy levels, benchmarking the performance of data transfers between these levels is of special interest.

In the context of the IPACS project that develops scalable, portable and realistic benchmarks for distributed systems [1], we create the PRIOMark: an I/O benchmark capable to characterize access performance to secondary storage of modern computer systems. PRIOMark does not only measure the I/O performance of single systems, it also characterizes the access performance to secondary storage of distributed systems. PRIOMark allows to define very complex workloads for imitating I/O behavior of many different applications, thus, the user is able to estimate and compare the performance of secondary storage in different hard- and software environments.

This paper introduces the PRIOMark benchmark by defining goals regarding the design of parallel I/O benchmarks and show how our software reaches them. Additionally, it presents some measurement results for an example architecture with the three different file systems NFS, GFS and PVFS, that are especially designed for the use in distributed environments.

2. I/O Benchmarks

Modern operating systems organize data storage in hierarchical structures called file systems. Per definition, files store data and directories store organizational information to collect files in logical groups. This allows easy and efficient access to data by users.

Every access to a file is done by an operating system call. This enables the operating system to cache accesses to secondary storage in the main memory. File system benchmarks measure the performance of accesses to files in file systems. Additionally, often other system calls like opening a file or closing it are taken into account.

For accessing files the operating system provides an interface e.g. the POSIX I/O interface common in most modern operating systems. It was designed to support file systems on a local storage device. Today's available I/O benchmarks use the POSIX I/O interface for performance measurements of file systems. However, it is not developed to allow parallel accesses to files from many nodes of a distributed computer system. To overcome this, the message passing library specification MPI-2 contains a file system interface, called MPI-IO. This interface is optimized for concurrent accesses to one file. As many processes can access one file concurrently, MPI-IO provides various methods for synchronizing on these accesses. *Individual file pointer* accesses allow every process to work on its own part of the file, thus having its own file pointer. *Shared file pointer* access methods allow all processes to work with a common file pointer. Concurrent accesses act as if they are serialized, but the real disk I/O can be done in parallel. When using accesses with *explicit offsets* a process specifies the position of any operation within the file.

Each described access method can either be used in blocking or non-blocking as well as in collective or non-collec-

tive mode. Blocking I/O operations suspend the calling process until an operation completes, while non-blocking operations return immediately indicating a successful execution. Collective I/O-calls are done by a group of processes. As all processes of the group access the file, there are many more possibilities to optimize the accesses by the system than by using the corresponding versions of these functions [2]. Thus, there is a multitude of possibilities for accessing files in a file system when using the MPI-IO interface. A distributed I/O benchmark has to take all these possibilities into consideration.

File system benchmarks examine the performance of the complete file system calls and thus, take caches of other memory hierarchy levels into account. To avoid disturbing effects of such caches there are benchmarks that bypass the disk cache of the operating system, called disk I/O benchmarks. These benchmarks access disks directly or use a special device driver provided by the operating system. Thus, disk I/O benchmarks are often hardware or operating system dependent.

3. Related Work

In the following the possibilities and drawbacks of important existing I/O benchmarks for single and distributed systems are described and the necessity for implementing a new I/O benchmark is clarified.

IOZone is a very common, highly portable file system benchmark for the POSIX I/O interface. It supports a variety of access types, including sequential and random read and write accesses. *IOZone* supports the definition of a complex workload by providing files with the exact position, size, and delay for every read and write I/O system call. Combinations of read and write accesses in workloads can only be specified by using this method or by setting the percentage of read accesses for the random read/write test. Since *IOZone* provides a lot of result information without a user-friendly summary of the results, it is very difficult to compare different systems [3].

Bonnie and *Bonnie++* are well-known and widely used file system benchmarks. It supports a sequential read and write test and a random read/write test. Four processes seek randomly in a new created file, read one block and rewrite it in 10 percent of all read accesses. So the workload characteristics are very specific and cannot be parameterized for user defined applications [4].

IOBENCH is a file system benchmark that uses the POSIX I/O interface for accessing the secondary storage. It allows a complex workload specification but also has a very complex output, making manual post-processing necessary [5]. As *IOZone* and *Bonnie*, *IOBENCH* only uses the POSIX I/O interface and cannot be used to measure secondary storage performance of distributed systems.

The *NAS BTIO* benchmark is an I/O benchmark that is derived from the compute benchmark BT that employs a fairly complex domain decomposition called multi-partitioning. The solution matrix of this calculation process is written to a file after every five time-steps and ordered by certain criteria at the end of the operation. *BTIO* does not use the POSIX I/O interface but collective write operations of MPI-IO for writing the matrix. Thus, *BTIO* only examines one workload and a very limited number of possibilities of MPI-IO. It cannot be used to specify parallel I/O performance for many applications [6].

b_eff_io is also a pure MPI-IO benchmark. It offers many more possibilities for measuring parallel I/O performance than *BTIO*, e.g. examining 36 different access patterns that cover a wide range of possible access patterns used in parallel applications. The major drawback is that it only takes blocking MPI-IO functions into consideration. Non-blocking I/O is an important access method in parallel applications, because it allows to produce new data while writing calculated data to disk. Additionally, measuring operations with explicit offsets is impossible. Altogether, this benchmark covers a variety of possibilities of the MPI-IO interface but it still leaves some open questions [7].

ior_mpiio is developed by the scalable I/O project at Lawrence Livermore National Laboratory. It measures combinations of read and write accesses of concurrent processes via the MPI-IO interface. The defined workload is very specific: A file is created, written and the written data is read back. Every process works on its part of the file [8].

By comprising all these benchmarks, none is able to work with more than one I/O interface. Most of them do not allow the possibility of defining complex workloads for a user-specific application. Especially the MPI-IO benchmarks only use a subset of the MPI-IO interface, e.g. no described MPI-IO benchmark measures the performance of asynchronous I/O.

PRIOmark overcomes these problems. It implements a variety of sub-benchmarks for measuring different access methods and I/O interfaces. Furthermore, it is extensible and easy to use.

The *PRIOmark* enhances available disk performance measurement programs by the following features:

- support of POSIX I/O and MPI-IO file system interfaces,
- a complex workload definition for all implemented sub-benchmarks,
- even POSIX I/O interface benchmarks are available in distributed environments,
- an asynchronous MPI-IO and POSIX I/O benchmark,
- a disk I/O benchmark (raw disk I/O),
- a single value as benchmark result for an easy comparison,
- and a plug-in architecture to integrate extended sub-benchmarks.

4. PRIOMark Benchmark

In the following part, the architecture of the novel I/O benchmark is described. This includes an overview about implemented sub-benchmarks, the possibilities of defining complex workloads and the usage in non-parallel environments.

4.1 Structure

The PRIOMark environment is a small, easy portable and extensible plug-in-based architecture. It primarily offers functions for time measurement and result output of the benchmark plug-ins. As shown in Figure 1 the sub-benchmarks *common_file*, *strided*, *async*, *single_file*, and *raw* are included. The last plug-in *total* calculates one single value out of the results of the previously mentioned sub-benchmarks. Those are described in the following paragraphs.

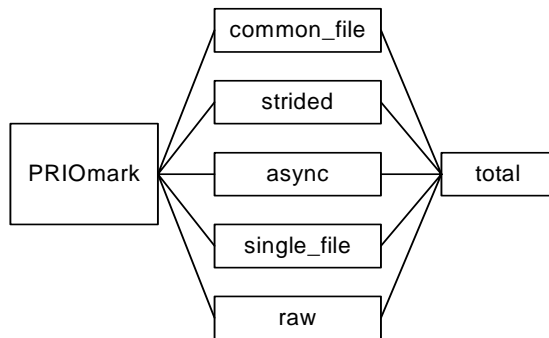


Figure 1: PRIOMark Benchmarks

common_file: This sub-benchmark acquires bandwidths for block-by-block accesses of a huge amount of processes to a single file. Each process is represented by one consecutive data block within the file (see Figure 2.a). The file size and the size for an I/O request varies. The following access patterns are tested:

- *Write*: creating and filling a new file with the specified file size,
- *Read/Write*: reading and re-writing of an existing file, and
- *Random Read/Write*: reading and re-writing of randomly chosen blocks in an existing file.

For read/write and random read/write accesses the ratio between read and write operations can be defined freely

(see Section 4.2). All measurements can use the collective and non-collective blocking MPI-IO access methods *individual file pointer* and *explicit offset*. As the MPI-IO interface is not available in all MPI implementations, measurements use POSIX I/O functions, too.

strided: This benchmark uses MPI file views to measure the I/O performance of accesses from many processes to a single file. Contrary to the *common_file* test, every process has a defined count of non-consecutive data blocks (see Figure 2.b). It tests the access patterns write and read/write (with the same semantics as for *common_file*). The number of continuous blocks of one stride as well as the file and block sizes can be defined freely. For read/write accesses the read-write-ratio can also be specified by the user. This benchmark only uses the MPI-IO *individual file pointers* for data access.

async: Asynchronous I/O is very important for high performance computing applications. It allows reading or writing data and still being able to calculate new information. Concurrent data transfers and computing increases the load of the bus system. Due to this fact the performance of the concurrently executed operations is lower as in the case both were serialized. Therefore, the *async* benchmark measures the asynchronous calculation loss (ACL) and the asynchronous bandwidth loss (ABL) of the parallel operations in relation to the serialized operations. The test uses the read/write access pattern with MPI-IO *individual file pointer*, *explicit offset* and POSIX I/O access methods.

single_file: This test is designed to measure the bandwidth of block-by-block accesses to the local disk. It only uses the POSIX I/O interface. Contrary to the POSIX I/O benchmarks introduced in Section 3, processes using this benchmark can also be distributed. In that case every process has its own test file. Figure 2.c presents the available storage access modes. In addition to the access patterns write, read/write and random read/write, this benchmark supports a backwards read test. This means, that the test file is read block-by-block in backwards order. File size and block size can be defined freely, as well as the ratio between read and write accesses for read/write and random read/write access patterns.

raw: Unlike the other tests this sub-benchmark is not a file system but a disk I/O benchmark. It measures the perfor-

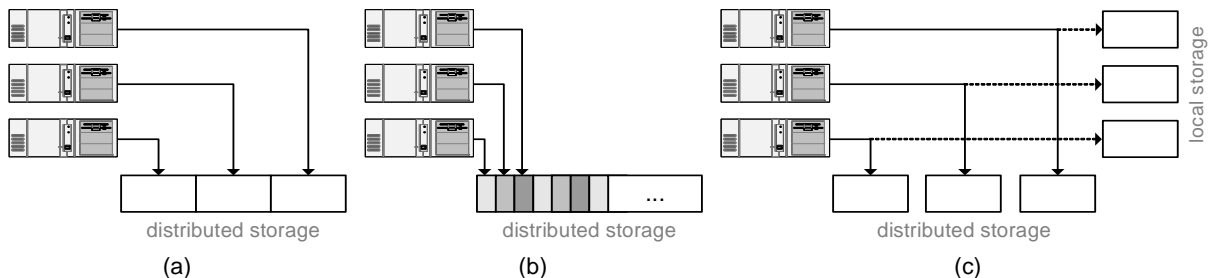


Figure 2: benchmark access modes

mance of raw-disk accesses to the local secondary storage. This allows to measure the true speed of the storage because raw-devices do not use operating system buffer caches. The bandwidth is measured by the access patterns read/write and random read/write. As for all other tests the read-write-ratio can be specified together with the data size and the block size (which must be multiples of the raw disk block size).

total: Because of the detailed output of the different sub-benchmarks *total* can be used to collect all their values and calculates an average. This value guarantees an easy comparison of the I/O performance of many different systems.

When running the PRIOMark program, the user can specify which of the tests *common_file*, *strided*, *async*, *single_file* and *raw* should be executed. This allows to run dedicated benchmarks for local storage access (*raw*, *single_file*), distributed file access (*common_file*, *strided*, *async*) or mixed accesses.

4.2 The Workload Definition

The PRIOMark benchmark supports processing of a complex application-specific workload definition. For all sub-benchmarks with exception of the *async* test the user may specify a range of file/data sizes and a range of block sizes for a single I/O request. For the access patterns read/write and random read/write the user may select the ratio between read and write requests. If a sub-benchmark supports different access methods (POSIX, MPI-IO individual file pointer, ...) the methods to be chosen are a matter of configuration, too. For the *async* sub-benchmark the user may choose the expected data size only, because this test has to read or write the whole data in a single request.

The workload definition allows to simulate the access behavior of a practical problem or program.

The whole configuration is stored in a single file. Thus, it becomes easy to repeat the benchmark using the same settings. It is possible to store workloads for different application domains in separate files.

4.3 Parallel or Non-Parallel

At build time of the benchmark the user may choose whether he wants to build a parallel version of PRIOMark able to run on distributed systems or a non-parallel version for single workstations. However, the sub-benchmarks *common_file* and *strided* are not available in the non-parallel version as they depend on MPI. Additionally, it is possible to build a parallel benchmark without using the MPI-IO interface, because it is not available in every MPI implementation. That version uses the MPI communication interface for synchronizing the single processes and collecting the measured values. Thus, it is possible to acquire the I/O performance of many machines when applying the local storage test concurrently using a single benchmark run, only.

5. Measurements

This section presents some benchmark results produced by our PRIOMark benchmark using an example architecture.

5.1 Test System

During our tests, three different file systems are examined for the measurements, designed especially for distributed environments. NFS (Network File System) is the most common file system for sharing data over networks using a central NFS server to store the data. Hence, the server represents a bottleneck and a single-point-of-failure. GFS (Global File System) by Sistina and RedHat is developed as a scalable alternative to NFS. It allows configuration to avoid single-point-of-failures (any component can be instantiated more than once) and is laid out to scale much better than NFS because data can be striped or mirrored to more than one server [9]. PVFS (Parallel Virtual File System) is designed to optimize concurrent accesses of many processes. It distributes a file to a given amount of I/O nodes. Moreover, a meta-data server provides the information about which node contains what part of a file. PVFS scales very well but again leads to one single-point-of-failure the meta-data server [10].

Our test system for the measurements is a Linux cluster with the Rocks Linux 3.2.0 operating system. It consists of one frontend and five compute nodes. The frontend node contains a dual processor 2 GHz Intel Xeon processor with 2 GB RAM and a three disk SCSI RAID 5 storage array. The compute nodes are 1.4 GHz Pentium III dual processor systems with 1 GB RAM and a single IDE hard disk. All nodes are connected by Gigabit Ethernet.

The frontend system exports a directory via NFS and a file system via GFS to all compute nodes. Furthermore, it runs the meta-data server for the PVFS file system.

GFS is configured as follows: The frontend exports one disk partition via GNBD (Global Network Block Device), containing the file system, to all nodes. Moreover, the frontend acts as lock server. For more information on GFS and GNBD and their configuration see [9] and [11]. In PVFS all compute nodes export a chunk of their disk space (I/O server). The frontend acts as required meta-data server, while the nodes act as PVFS clients and mount the file system. For detailed information about PVFS see [10].

Our system uses MPICH version 1.2.5 with ROMIO version 1.2.4 as MPI implementation. It is compiled with support for NFS and PVFS and general support for all Unix like file systems (UFS) [12][13].

5.2 Parallel Benchmarks

Figure 3 shows the accumulated I/O bandwidth of all processes depending on the concurrently running processes for NFS, PVFS and GFS.

The presented results are calculated averages of the sub-benchmarks *common_file*, *strided* and *async*. The data size

for *common_file* and *strided* sub-benchmarks are set to 32 MB per process and the block size is set to 16 KB. The data size for the *async* sub-benchmark is set to 8 MB. As the amount of data is too small to exceed the size of the disk cache, the shown results represent operating system cache performance using different file systems. In the (random) read/write tests the ratio between read and write accesses evaluates to 1/1. For the PVFS file system the *async* test only uses the POSIX I/O access methods, because the native PVFS interface for MPICH does not support asynchronous I/O calls.

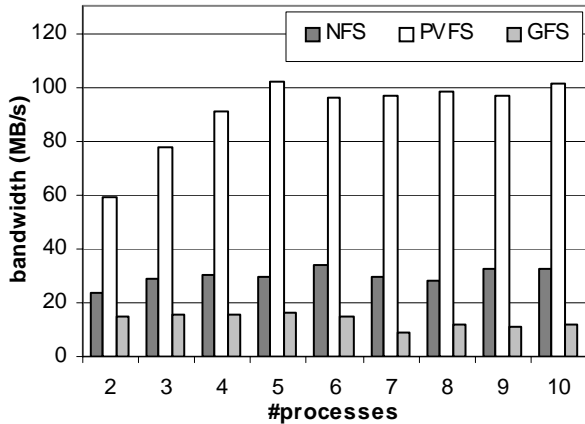


Figure 3: file system performance

As NFS and GFS have a nearly constant bandwidth in the shown measurement range, the PVFS bandwidth increases to a maximum. PVFS distributes files to all I/O nodes, thus, concurrent accesses to one file can be distributed over parallel network connections. Parts of these files are always stored on the local disk of a node. This explains the good performance and scalability of PVFS. Additionally, the measured GFS performance is not as good as the NFS performance. This happens because our GFS configuration does not take account of all possibilities of this file system. Using more than one GFS server will increase the I/O performance of the file system and the availability of data.

The PRIOMark provides the possibility of defining different workloads. Figure 4 shows the bandwidth of two workloads as specified in [14] depending on the number of concurrent running processes using MPI-IO for file system accesses. PVFS is used for these measurements. A web server workload applying mostly read accesses with small request sizes between 1 kB and 8 kB and a typical workload used in OLTP (Online Transaction Processing) applications are shown. OLTP applications are databases that support real-time processing of database operations often used in e-commerce and other time-critical applications. They typically use combinations of read and write accesses in the ratio of 1:2 and request sizes of 8 kB. Both workloads use 1 GB of data per process leading to usage of the systems cache and secondary storage. This reflects

the bad results compared to the bandwidths shown in Figure 3.

Figure 4 shows that the performance of the web server workload is very poor. The bandwidth of this workload is three times smaller than the bandwidth of the OLTP workload. Mainly, this is caused by the small request size used by web servers.

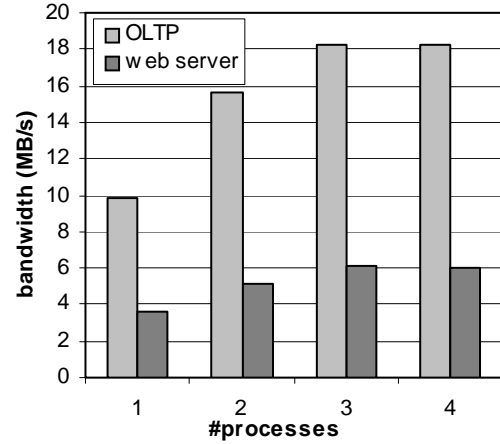


Figure 4: workload comparison

5.3 Local Benchmarks

The average measurement results of the *single_file* sub-benchmark in dependency to the number of concurrently running processes on one compute node is shown in Figure 5. The benchmark is configured with the following parameters: file size 256 MB, block size 32 KB and read/write ratio 1/1 for test patterns read/write and random read/write.

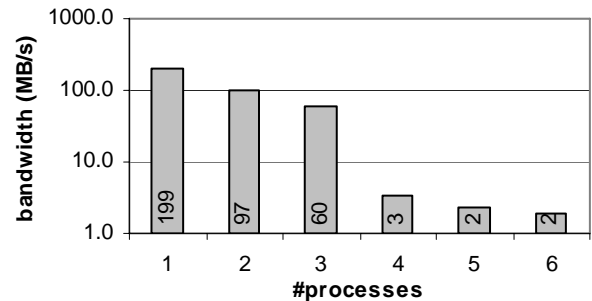


Figure 5: single_file sub-benchmark

The chart shows that if the accumulated data size is smaller than the system's RAM the operating systems buffer cache guarantees good performance. There is a bandwidth break-in in case the data size of all running processes exceeds the 1 GB RAM of the system.

Figure 6 shows the average measured bandwidth from the *raw* sub-benchmark in dependence to data and block size on one compute node. The ratio between read and write accesses for the read/write and random read/write tests is set to 1/0.

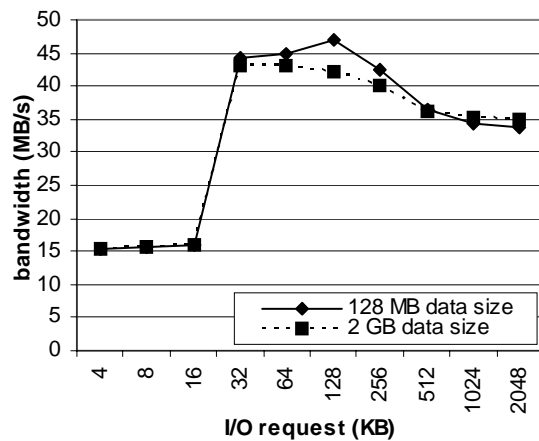


Figure 6: raw sub-benchmark

As one can see, the raw performance of the used disks is about 15 MB/s for small and 35 MB/s for large I/O requests. There is a maximum available bandwidth of more than 40 MB/s between an I/O request size of 64 KB and 256 KB. The examined data sizes show that the available transfer speed is independent of the system's RAM.

6. Conclusion and Further Work

The introduced I/O benchmark PRIOMark is a combined file system and disk benchmark. It works with the POSIX I/O as well as the MPI-IO interface. The benchmark allows a complex definition of workloads for sub-benchmarks and an I/O performance analysis for single workstations as well as distributed systems like cluster computers.

The measured values show the I/O performance of different distributed file systems on a Linux cluster and the performance of accesses to a local disk of a single system. The measurement results allow an easy comparison of the different systems.

Our future work will include an enhancement of the workload definition with a file that contains the order, file offset and length for any I/O request. With this description it is possible to reproduce the I/O behavior of an application precisely. It is even possible to reproduce I/O behavior of commercial applications by using a *libc*-wrapper that writes all I/O interface calls to a file while running the application. This is feasible for non-parallel as well as for distributed applications. In a distributed environment one more step is necessary to collect the data-files from the single hosts. Such a data-file is valid only for one system configuration (like: 4 hosts, 2 processes per host).

Additionally, PRIOMark will be ported and tested on more architectures.

7. Acknowledgement

This research is sponsored by the German Federal Ministry of Education and Research grant 01IRB03E in the context of the IPACS project [1]. We thank the sponsors and the project partners Fraunhofer Institute for Techno- and Business Mathematics, T-Systems, National Energy Research Scientific Computing Center and the University of Mannheim for their support.

References:

- [1] IPACS, <http://www.ipacs-benchmark.org>
- [2] Gropp, Huss-Ledermann, Lumsdaine, Lusk, Nitzberg, Saphir, Snir, *MPI - the complete reference vol. 2* (Cambridge: MIT Press, 1998)
- [3] IOzone, <http://www.iozone.org>
- [4] Bonnie Benchmark, <http://www.textuality.com/bonnie>
- [5] B. L. Wolman, T. M. Olson, IOBENCH: A System Independent IO Benchmark, *Computer Architecture News*, 1989
- [6] P. Wong, R. F. Van der Wijngaart, NAS Parallel Benchmarks I/O Version 2.4, *Technical Report NAS-03-002*, 2003
- [7] R. Rabenseifner and A.E. Koniges, The Effective I/O Bandwidth Benchmark (*b_eff_io*), *Proceedings of the Message Passing Interface Developer's Conference*, 2000
- [8] Lawrence Livermore National Laboratory Homepage, <http://www.llnl.gov>
- [9] Red Hat GFS 6.0 Administrator's Guide, <http://www.redhat.com/docs/manuals/csgfs/admin-guide/>
- [10] P. H. Carns, W. B. Ligon III, R. B. Ross, R. Thakur, PVFS: A Parallel File System For Linux Clusters, *Proceedings of the 4th Annual Linux Showcase and Conference*, Oct. 2000
- [11] Red Hat Cluster Project Page, <http://sources.redhat.com/cluster/>
- [12] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A High-performance, Portable Implementation of The MPI Message Passing Interface Standard, *Parallel Computing*, 22(6), 1996, 789-828
- [13] R. Thakur, E. Lusk, W. Gropp, Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation, *Technical Memorandum ANL/MCS-TM-234*, May 2004.
- [14] R. Bryant, D. Raddatz, R. Sunshine, PenguinoMeter: A New File-I/O Benchmark for Linux, *Proceedings of the 5th Annual Linux Showcase and Conference*, Nov. 2001