Data Layout Optimization for Petascale File Systems

Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
sun@iit.edu

Yong Chen
Department of Computer Science
Illinois Institute of Technology
yong.chen@iit.edu

Yanlong Yin
Department of Computer Science
Illinois Institute of Technology
yyin2@iit.edu

ABSTRACT

In this study, the authors propose a simple performance model to promote a better integration between the parallel I/O middleware layer and parallel file systems. They show that application-specific data layout optimization can improve overall data access delay considerably for many applications. Implementation results under MPI-IO middleware and PVFS2 file system confirm the correctness and effectiveness of their approach, and demonstrate the potential of data layout optimization in petascale data storage.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communications**]: Interconnections (Subsystems) – *parallel I/O*. D.4.2 [**Operating Systems**]: Storage Management – *allocation/deallocation strategies, secondary storage*.

General Terms

Measurement, Performance, Experimentation.

Keywords

Data layout, parallel file systems, parallel I/O

1. INTRODUCTION

High-performance computing (HPC) has crossed the Petaflop mark and is moving forward to reach the Exaflop range [15]. However, while computing resources are making rapid progress, there is a significant gap between processing capacity and data-access performance. Due to this gap, although processing resources are available, they have to stay idle waiting for data to arrive, which leads to a severe overall performance degradation. Figure 1 shows the number of CPU cycles required to access cache memory (SRAM), main memory (DRAM), and disk storage [2]. It can be seen that the number of cycles for accessing disks is hundreds of thousands of times slower. This trend is predicted to continue in the near future. In the meantime, applications are becoming more and more data intensive. Due to the growing performance disparity and emerging data intensive applications, I/O and storage have become a critical performance bottleneck in HPC machines, especially when we are dealing with petascale data storage.

Data layout mechanism decides how data is distributed among multiple file servers. It is a crucial factor that decides the data access latency and the I/O subsystem performance for high-performance computing. The recent work in log-like reordering of data [1][7] has demonstrated the importance and performance improvement by arranging data in a proper manner. However, historically, parallel I/O middleware systems, such as ROMIO [14], and parallel file systems are developed separately with a simplified modular design in mind. Parallel I/O middleware systems often assume the underlying is a big file system, and, on the other hand, parallel file systems often rely on the I/O middleware for data access optimization and do little in data layout optimization. In this study, we argue that purely depending on I/O middleware for data retrieval optimization is costly and may not be effective in many situations. We argue that if we pass some of the application-specific I/O request information to file systems for data layout optimization, the results could be much better. Existing parallel file systems, such as PVFS2 [3], Lustre [5], and GPFS [11] provide high bandwidth for simple, well-formed, and generic I/O access characteristics, but their performance varies from application to application [4][8]. Tuning data layout according to specific I/O access patterns for a parallel I/O system is a necessity. This tuning requires understanding file system abstractions, gaining knowledge of disk storage, knowing the designs of high-level libraries, and making intelligent decisions. While PVFS2 and high-level parallel I/O libraries, such as MPI-IO [13] and HDF-5 [12] provide some functionality to customize data layout according to specific I/O workloads, few know how to use them effectively.

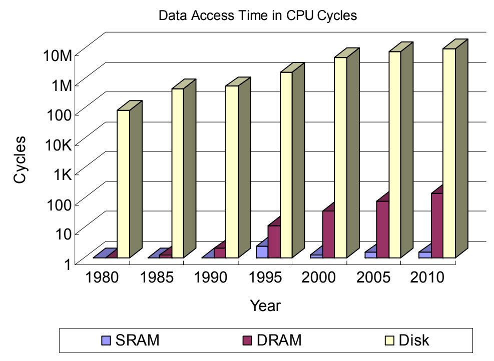

Figure 1. Comparison of data access latency.

In this research, we study data layout optimization of parallel file systems. We show that, with the consideration of application-specific I/O request, the data layout optimization can be totally different in a parallel file system. We present a *system-level application-specific data layout optimization strategy* for petascale data storage. By system-level, we mean that the proposed approach is integrated into the file system and is transparent to programmers and users. By application-specific, we mean that the proposed approach can adapt to specific data access patterns for a proper data layout. The contribution of this study is two folds. First, we show that the data layout optimization has a significant impact on petascale data storage performance. Second, we demonstrate with a simple performance model and current simple data layout functionalities provided by PVFS2 that we can achieve noticeable performance gain. While our results are preliminary, they demonstrate the potential of the data layout optimization approach.

2. APPLICATION-SPECIFIC DATA LAYOUT MODELING

Modeling and evaluating the performance of data layout strategy is essential in providing an application-specific data layout optimization. The conventional round-robin distribution (referred to as simple striping in some existing work) is in place in many of parallel file systems [3][5][11]. However, under parallel I/O systems, this simple distribution may not be the best data layout and can be improved.

We present a simple data layout performance model herein. In this model, we assume that the connection between compute (I/O) nodes and file servers is not a performance bottleneck and that the significant overhead is in accessing file servers. We further assume that each file server's performance can be measured as $\alpha+s\beta$, where α is the start up time (latency), s is the data size, and β is the transmission time of single unit data (the reciprocal of transmission rate).

In this model, we differentiate three data layout strategies, 1-D Horizontal Layout, 1-D Vertical Layout and 2-D Layout. The *1-D Horizontal Layout* (or 1-DH in short) refers to the strategy that data is distributed among all available file servers in a traditional round-robin fashion. This layout matches with the existing simple striping or round-robin strategy. The *1-D Vertical Layout* (or 1-DV in short) refers to the strategy that data to be accessed by each process is stored on one given file server. The *2-D Layout* (or 2-D in short) is the strategy in which data to be accessed by each process is stored on a subset of file servers. Figure 2 illustrates these three strategies with an example.

Assume that we have p computing (I/O) nodes, n file servers, where all computing nodes participate in an SPMD form of parallel computing, with a block-cyclic or some similar, even data partitioning. With 1-DH data layout, i.e., with simple striping round-robin layout where exactly s/n of the data are in any of the n file servers, the cost of accessing data of size s by one process and p processes are:

$$(\alpha+\frac{s}{n}\beta) \text{ and } p(\alpha+\frac{s}{n}\beta)= p\alpha+\frac{ps}{n}\beta \qquad (1)$$

respectively. With the 1-DH layout, each process accesses its required data concurrently, but multiple processes have to access data one by one sequentially; and the data of each process is distributed over different file servers evenly. This strategy makes accesses in a "sequential concurrent" way. The value of Equation (1) depends on the value of p, n, α and β. In any case, however, the 1-DH layout or the conventional round-robin layout may not be the best choice when $p \geq n$. If we take the 1-DV layout, i.e. taking a "concurrent sequential" approach, we can get a better performance, with $\left\lceil \frac{p}{n} \right\rceil (\alpha+s\beta)$. If $p < n$, then the data can be stored either on n servers using 1-DH layout or using 2-D layout, where each of the p processes gets n/p file servers for data storage. For the former layout, the cost is $\alpha+s\beta$ and for the latter case, the cost is $\alpha+\frac{s}{\left\lfloor \frac{n}{p} \right\rfloor}\beta$.

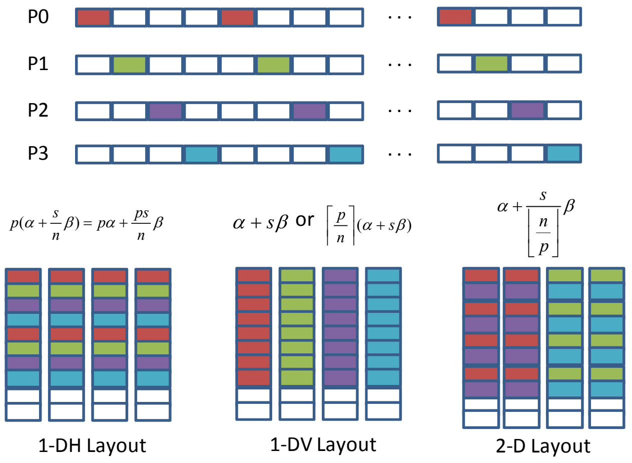
P0
P1
P2
P3
$p(\alpha+\frac{s}{n}\beta)=p\alpha+\frac{ps}{n}\beta$
$\alpha+s\beta$ or $\left\lceil \frac{p}{n} \right\rceil(\alpha+s\beta)$
$\alpha+\frac{s}{\left\lfloor \frac{n}{p} \right\rfloor}\beta$
1-DH Layout
1-DV Layout
2-D Layout

Figure 2. Data layout strategies.

3. APPLICATION-SPECIFIC DATA LAYOUT OPTIMIZATION

With application-specific data layout modeling, we are able to guide data layout towards a better way by considering data access characteristics. With the value of p, n and α, β, the proper data layout can be determined with the aforementioned data layout formulas for a given parallel I/O request. The data layout of a given application can then be determined based on the weighted summation of the costs of its I/O requests. The above model is deterministic and is ready to use under existing parallel file systems, such as PVFS2. In addition, in parallel I/O applications, it is common that an application accesses multiple files and each file in multiple occasions. We store each file in a different layout to improve performance. When an application accesses a file in multiple patterns, it is necessary to find a layout that is beneficial for all patterns. For example, a file is read in contiguous access pattern and written in a complex non-contiguous pattern. From many observations [8][13], accessing data in non-contiguous patterns performs worse than accessing contiguously. Storing data to facilitate non-contiguous accesses may deteriorate contiguous access performance. We have to find a balance between performance benefits when we decide on performance layouts. Based on pattern analysis, we can utilize a strategy by assigning each pattern a weight to represent its scope for I/O performance improvement.

Based on the modeling and observations, we define a set of data layout heuristics as shown in Table 1. When I/O access characteristics are unknown or completely random, we rely on 1-DH strategy or the default simple round-robin strategy. When the degree of I/O concurrency is high, it is beneficial to use 1-DV layout. 1-DH layout or 2-D layout can be configured for low degree of concurrency. In case of TCP Incast [10], it is better to stripe data among a certain set of file servers instead of all available file servers, which is 2-D layout. File systems such as PVFS2 provide features to extend and create new distributions [9]. We utilize these features in generating new application-specific distributions in our implementation.

Table 1. Heuristics for Choosing Layouts

Access Pattern Feature	**Data Layout Strategy**
Random	1-DH (round-robin) layout
High degree of I/O concurrency	1-DV data layout
Low degree of I/O concurrency	1-DH or 2-D data layout
Too many I/O servers on TCP/IP	2-D data layout

After making decisions on the layout, we store data on file servers using the new layout. The 1-DH layout strategy, or the simple round-robin layout, with different stripe size and striping factor can be set with MPI-IO hints, such as *striping_factor* and *striping_unit*. A more complex distribution, such as 1-DV or 2-D data layout, needs to be modified at the file system level to provide general support, but can be emulated with different *striping_factor* and *striping_unit* configurations. In addition, it is common for parallel file systems, such as PVFS2, to provide flexible and extendable data distributions [9]. PVFS2 includes a modular system for adding new data distributions to the system and using these for new files and optimized layouts. Since our current implementation focuses on prototyping the idea and verifying the potential performance gain, we employ a relatively quick prototyping strategy by using parallel file system configurations to provide support for various layout strategies. The current prototyping system has demonstrated a significant performance improvement over existing strategies as the following section shows. A general full-fledged data layout strategy support at parallel filesystem level is under development as well.

4. PRELIMINARY EXPERIMENTAL RESULTS

We have carried out a prototype implementation of application-specific data layout on PVFS2 parallel file system based on the previously discussed model and optimization strategy. We currently support three strategies, 1-DH, 1-DV and 2-D layouts. The following subsections present the initial experimental results of these application-specific strategies under different scenarios.

4.1 Experimental Setup

Our experiments were conducted on a 17-node Dell PowerEdge Linux-based cluster and a 65-node Sun Fire Linux-based cluster. The Dell cluster is composed of one Dell PowerEdge 2850 head node, with dual 2.8 GHz Xeon processors and 2 GB memory, and 16 Dell PowerEdge 1425 compute nodes with dual 3.4 GHz Xeon processors and 1 GB memory. The head node has two 73 GB U320 10K-RPM SCSI drives. Each compute node has a 40 GB 7.2K-RPM SATA hard drive. The Sun cluster is composed of one Sun Fire X4240 head node, with dual 2.7 GHz Opteron quad-core processors and 8GB memory, and 64 Sun Fire X2200 compute nodes with dual 2.3GHz Opteron quad-core processors and 8GB memory. The head node has 12 500GB 7.2K-RPM SATA-II drives configured as RAID-5 system. Each compute node has a 250GB 7.2K-RPM SATA hard drive. The experiments were tested on PVFS2 file system. For the Dell cluster, PVFS2 was configured with one metadata server node, the head node, and 8 I/O server nodes. All nodes are used as compute nodes. For the Sun Fire cluster, PVFS2 was configured with 32 I/O server nodes. The rest nodes are used as compute nodes.

4.2 Experimental Results and Analyses

4.2.1 Synthetic Benchmark

We have coded a synthetic benchmark which does sequential reads over the file stored with different layouts. We have performed a series of tests on the Dell cluster. The first set of experiments conducted is to compare the performance of different layout strategies with four compute processes. In this scenario, four processes retrieve data from 64MB, 160MB, 320MB, 800MB and 2000MB files respectively. These files are stored on eight file servers with three layouts, 1-DH, 1-DV and 2-D. We measured the performance of retrieving data in each case and the results are shown in Figure 3. The reported results are the average of three runs. We flushed the system buffer cache between each run.

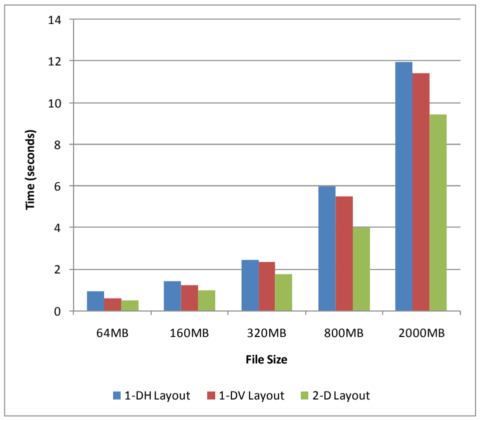

Figure 3. I/O performance with different layout strategies.

Figure 3 clearly shows that different layout strategies do have a considerable impact to the performance of parallel I/O system. Among three layouts, the 2-D layout achieved the best performance in all cases. This is consistent with our model and analysis that the 2-D layout is desired when the number of compute processes is less than that of I/O server nodes. In the meantime, the 1-DH layout, or the default round-robin layout, performed worse than both 1-DV and 2-D layouts, and the performance disparity was up to 48.8%.

We have also performed a detailed analysis to verify the proposed model. We compute the theoretical value with the model and the measured disk transfer time and startup time. The theoretical and experimental results are shown in Figure 4 (1-DH layout is omitted here due to the space limit). As can be seen from the results, there is a close match between the experimental results and theoretical results, which shows the model can estimate the performance of these layout strategies well.

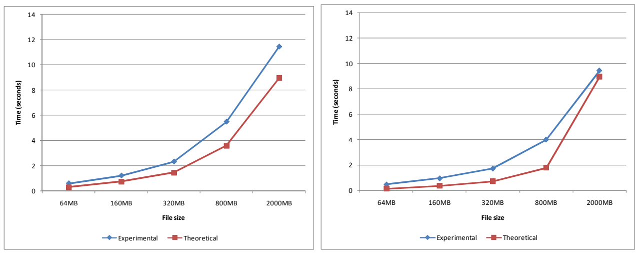

Figure 4. Experimental and theoretical results. (Left: 1-DV layout; Right: 2-D layout)

The other set of experiments we have conducted is to compare the impact of layout strategies with 16 compute processes. This set of tests is similar with the previous tests, but the file sizes are doubled in order to compare the performance with various file sizes. The results show that 1-DV layout outperformed the other two strategies in all cases, which is consistent with the model and analysis presented in Section 2. The results are shown in Figure 5.

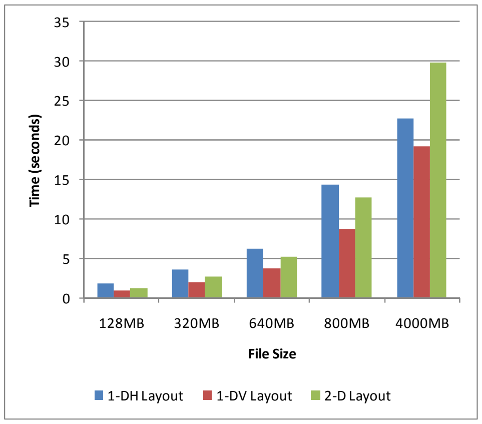

Figure 5. I/O performance with different layout strategies.

4.2.2 IOR Benchmark

In addition to the synthetic benchmark measurement, we have performed a series of testing on the Sun cluster with the IOR-2.10.2 benchmark from Lawrence Livermore National Laboratory [6]. In these experiments, we performed a larger scale of testing. We configured PVFS2 with 32 I/O server nodes and run testing with 64 processes on 32 client nodes (client nodes are separate from I/O server nodes). We performed both sequential reads/writes and random reads/writes tests, and varied the stripe size and the file size. Figure 6 and Figure 7 report the bandwidth results of accessing files with different layouts in a random or sequential manner, respectively, with 64KB stripe size for 1-DH and 2-D layouts. Figure 8 and Figure 9 report the results in a similar scenario, but with 1MB stripe size for 1-DH and 2-D layouts.

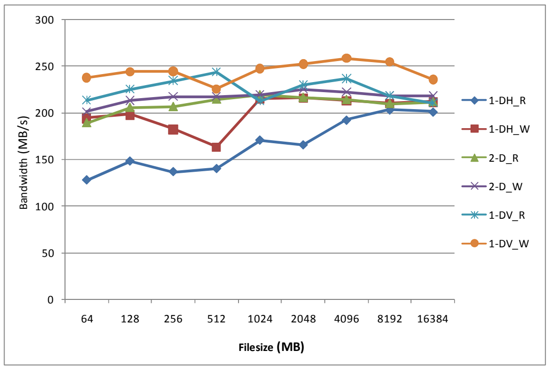

Figure 6. Random reads/writes with 64KB stripe size.

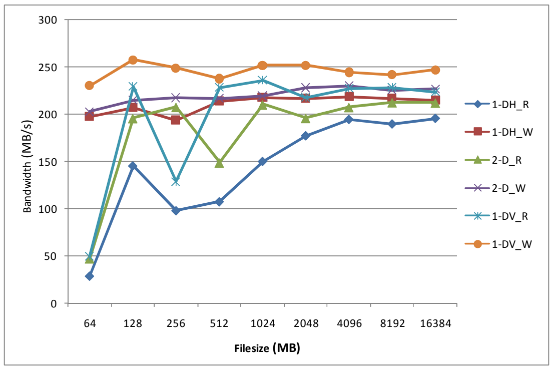

Figure 7. Sequential reads/writes with 64KB stripe size.

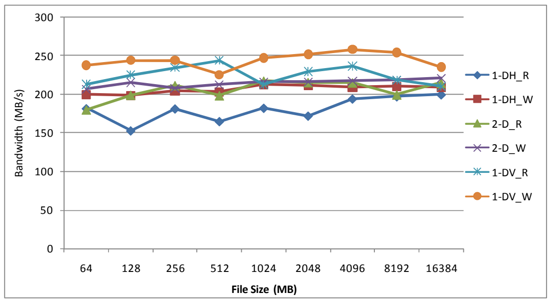

Figure 8. Random reads/writes with 1MB stripe size.

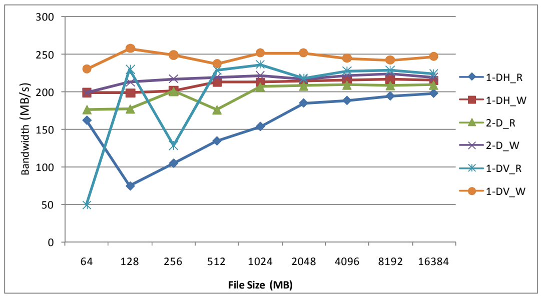

Figure 9. Sequential reads/writes with 1MB stripe size.

As can be seen from these results, different layout strategies can affect the IOR benchmark testing performance considerably. Among the three strategies we specifically analyze, the 1-DV strategy generally performs better than the other two, while the 2-D strategy performs better than the 1-DH strategy.

Although the current experimental results are preliminary, they have demonstrated that data layout strategies have a considerable impact on parallel I/O systems. The proposed model and application-specific data layout optimization are desired to dynamically adapt the layout to achieve a better performance under different scenarios.

5. ONGOING WORK

We have reported some of initial results, while several studies are ongoing and are not ready to report at this time. For instance, we are working on a comprehensive data layout model to characterize the performance impact of layout strategy in general cases based on probability and queuing theory. The basic idea of the general model is that each I/O node can be modeled as an independent queue. I/O requests come into these queues and are serviced for either storing or retrieving data. When contention occurs, the request has to wait in the queue to be serviced. Multiple queues are independent from each other, and data layout optimization on parallel file servers are derived accordingly. This model characterizes concurrency (parallelism) and contention, two major roles that data layout strategy plays in affecting the system performance, to guide an optimal layout selection. We have developed a theoretical model and are working on the experimental part to verify the model. We are also moving the experimental testing to a much larger computer cluster than what we have used.

6. CONCLUSION

Parallel I/O middleware and parallel file systems are fundamental and critical components for petascale storages. While both of the technologies have made their success, little has been done to application-specific data layout. In most existing file systems, data is distributed among multiple servers primarily with a simple round-robin strategy. This simple data layout strategy does not always work well for parallel I/O system, where I/O requests are generated concurrently. In this study, we have proposed an application-specific data layout strategy to optimize the performance of accessing data according to distinct application features. This data layout strategy optimization is built upon a simple but effective data layout model, and has been prototyped with the configuration facility of the underlying PVFS2 parallel file system.

Parallel file systems have been designed as one-set-for-all and have been static. There is a great need for research into next-generation I/O architectures to support access awareness, intelligence, and application-specific adaptive data distribution and redistribution. Although our current results are very limited, our prototyping system has demonstrated the great potential in improving parallel I/O access performance via data layout optimization when access characteristics are taken into consideration. We believe that the application-specific data layout optimization approach needs a community attention. This approach appears to be a feasible solution to mitigating the I/O wall problem, especially for petascale data storages.

7. ACKNOWLEDGMENTS

The authors are thankful to Dr. Rajeev Thakur, Dr. Rob Ross and Sam Lang of Argonne National Laboratory for their constructive and thoughtful suggestions toward this study. This research was supported in part under NSF grant CCF-0621435 and CCF-0937877.

8. REFERENCES

[1] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, M. Wingate, "PLFS: A Checkpoint Filesystem for Parallel Applications," in Proc. of ACM/IEEE SuperComputing'09.

[2] R. E. Bryant and D. O'Hallaron, "Computer Systems: A Programmer's Perspective," Prentice-Hall, 2003.

[3] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters," in Proceedings of the 4th Annual Linux Showcase and Conference, 2000.

[4] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/Output Characteristics of Scalable Parallel Applications," in Proceedings of the ACM/IEEE Conference on Supercomputing, 1995.

[5] Cluster File Systems Inc., "Lustre: A Scalable, High Performance File System," Whitepaper, *http://www.lustre.org/docs/whitepaper.pdf.*

[6] Interleaved or Random (IOR) Benchmark, *http://sourceforge.net/projects/ior-sio/.*

[7] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin, "Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS)," in Proc. of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, 2008.

[8] J. May, "Parallel I/O for High Performance Computing," Morgan Kaufmann Publishing, 2001.

[9] PVFS2 Development Team, "PVFS Developer's Guide," *http://www.pvfs.org/cvs/pvfs-2-8-branch-docs/doc//pvfs2-guide.pdf.*

[10] A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-Based Storage Systems," in Proceedings of File and Storage Technologies (FAST), 2008.

[11] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in 1st USENIX Conference on File and Storage Technologies, USENIX, 2002.

[12] The HDF5 Project, HDF5 - A New Generation of HDF, NCSA, Univ. of Illinois at Urbana Champaign. Available at *http://hdf.ncsa.uiuc.edu/HDF5.*

[13] R. Thakur, W. Gropp and E. Lusk, "Optimizing Noncontiguous Accesses in MPI-IO," Parallel Computing, (28)1:83-105, 2002.

[14] R. Thakur, W. Gropp and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, 1999.

[15] Top 500 Supercomputing Website. *http://www.top500.org.*