Multiple-Level MPI File Write-Back and Prefetching for Blue Gene Systems

Javier García Blas,[1] Florin Isailă[1], J. Carretero[1]
Robert Latham[2], and Robert Ross[2]

[1] University Carlos III, Spain
{fjblas,florin,jcarrete}@arcos.inf.uc3m.es
[2] Argonne National Laboratory
{robl,rross}@mcs.anl.gov

Abstract. This paper presents the design and implementation of an asynchronous data-staging strategy for file accesses based on ROMIO, the most popular MPI-IO distribution, and ZeptoOS, an open source operating system solution for Blue Gene systems. We describe and evaluate a two-level file write-back implementation and a one-level prefetching solution. The experimental results demonstrate that both solutions achieve high performance through a high degree of overlap between computation, communication, and file I/O.

Key words: MPI-IO, Parallel I/O, Parallel File Systems, Supercomputers.

1 Introduction

The past few years have shown a continuous increase in the performance of supercomputers and clusters, as demonstrated by the evolution of Top 500 [1]. IBM's Blue Gene supercomputers have a significant share in the Top 500 lists and bring additionally the advantage of a highly energy-efficient solution. Blue Gene systems scale up to hundreds of thousands of processors. In order to allow data-intensive applications to make efficient use of the system, compute scale needs to be matched by a corresponding performance of file I/O.

In earlier work [2] we presented the design and implementation of an MPI-based hierarchical I/O cache architecture for Blue Gene/L systems based on open source software. Our solution was based on an asynchronous data-staging strategy that hides the latency of file system access of collective file operations from compute nodes. Blue Gene/L presents two limitations, which affected the efficiency of our solution. First, the BG/L compute nodes do not support multithreading. Therefore, computation can not be overlapped with the communication with the I/O node. Second, because L1 caches of the CPUs are not coherent, even though multithreading is supported on the I/O nodes, the threads could use only one processor. Consequently, the communication with the compute node could be not overlapped with file system activity. The limitations discussed above have been removed from the Blue Gene/P architecture: multithreading is supported on the compute nodes, and the L1 caches of the cores are cache coherent.

In this paper we discuss the evolution of out hierarchical I/O cache architecture and its deployment on Blue Gene/P systems. This paper makes the following contributions:

– We discuss the extensions of the initial Blue Gene/L solution necessary for the efficient porting to Blue Gene/P systems.
– The novel solution includes a compute-node write-back policy, which asynchronously transfers data **from the compute nodes to the I/O nodes**. This policy complements the initial data-staging strategy, which performed asynchronous transfers **between I/O nodes and the final storage system**.
– We describe and evaluate an I/O node prefetching strategy that asynchronously transfers file data from the storage system to the I/O node.
– We present and analyze a novel evaluation on a Blue Gene/P system.

The remainder of the paper is structured as follows. Section 2 reviews related work. The hardware and operating system architectures of Blue Gene/P are presented in Section 3. We discuss our novel solution for Blue Gene/P systems in Section 4. The experimental results are presented in Section 5. We summarize and discuss future work in Section 6.

2 Related Work

Several researchers have contributed techniques for hiding the latency of file system accesses. Active buffering is an optimization for MPI collective write operations [3] based on an I/O thread performing write-back in background. Write-behind strategies [4] accumulate multiple small writes into large, contiguous I/O requests in order to better utilize the network bandwidth. based on application disclosed access patterns [5],signatures derived from access pattern classifications [6], and speculative execution [7, 8].

A limited number of recent studies have proposed and evaluated parallel I/O solutions for supercomputers. Yu et al. [9] present a GPFS-based three-tiered architecture for Blue Gene/L. The tiers are represented by I/O nodes (GPFS clients), network-shared disks, and a storage area network. Our solution extends this hierarchy to include the memory of the compute nodes and proposes an asynchronous data-staging strategy that hides the latency of file accesses from the compute nodes. An implementation of MPI-IO for Cray architecture and the Lustre file system is described in [10]. In [11] the authors propose a collective I/O technique, in which processes are grouped together for collective I/O according to the Cray XT architecture.

3 Blue Gene/P

This section presents the hardware and operating system architectures of Blue Gene/P.

3.1 Blue Gene/P Architecture

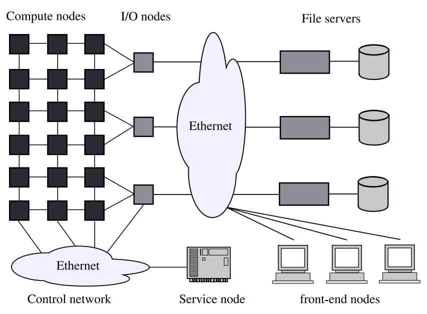

Fig. 1. Blue Gene/P architecture overview.

Figure 1 shows a high-level view of a Blue Gene/P system. Applications run on compute nodes. Compute nodes are grouped into processing sets, or “psets.” Applications run in exclusivity on partitions, consisting of multiples of psets. Each pset has an associated I/O node, which performs I/O operations on behalf of the compute nodes from the pset. The compute and I/O nodes are controlled by service nodes. The file system components run on dedicated file servers connected to storage nodes though a 10Gbit Ethernet switch.

Compute and I/O nodes use the same ASIC with four PowerPC 450 cores, with core-private hardware-coherent L1 caches, core-private stream prefetching L2 caches, and a 8 MB shared DRAM L3 cache. There are 4 GBytes of main memory per node.

Blue Gene/P nodes are interconnected by five networks: 3D torus, collective, global barrier, 10 Gbit Ethernet, and control. The 3D torus (5.1 GBytes/s) is typically used for point-to-point communication between compute nodes. The collective network (1700 MBytes/s) has a tree topology and serves collective communication operations and I/O traffic. The global barrier network offers an efficient barrier implementation. The 10 Gbit Ethernet network interconnects I/O nodes and file servers. The service nodes control the whole machine through the control network.

3.2 Operating System Architecture

In the IBM solution [12], the compute node kernel (CNK) is a light-weight operating system offering basic services: creation of process address spaces, simple

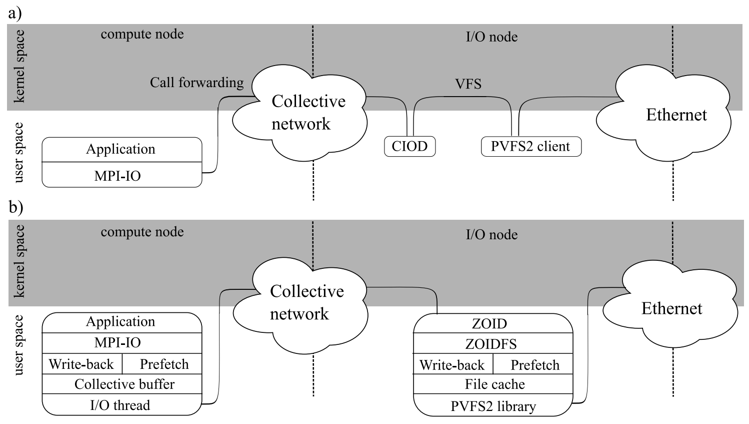

Fig. 2. I/O forwarding for IBM and ZeptoOS solutions.

system calls such as setting an alarm, and forwarding I/O-related system calls to the I/O nodes. As shown in Figure 2(a), the I/O system calls are forwarded through the tree collective network to the I/O node. The I/O nodes run a simplified Linux OS kernel (IOK) with a small memory footprint, an in-memory root file system, TCP/IP and file system support, and no swapping and lacking the majority of classical daemons. The forwarded calls are served on the I/O node by the control and I/O daemon (CIOD). CIOD executes the requested system calls on locally mounted file systems and returns the results to the compute nodes.

An open-source alternative to the IBM's solution is developed in the ZeptoOS project [13]. Under ZeptoOS, Blue Gene compute nodes may run Linux, while the I/O forwarding is implemented in a component called ZOID. The I/O forwarding process shown in Figure 2(b) is similar to the one based on CIOD, in the sense that I/O related calls are forwarded to the I/O nodes, where a multithreaded daemon serves them. However, there are two notable differences in design and implementation between CIOD-based and ZOID-based solutions. First, ZOID comes with its own network protocol, which can be conveniently extended with the help of a plug-in tool, which automatically generates the communication code for new forwarded calls. Second, the file system calls are forward through ZOIDFS [14], an abstract interface for forwarding file system calls. ZOIDFS abstracts away the details of a file system API under a stateless interface consisting of generic functions for file create, open, write, read, close, and so forth.

4 Write-Back and Prefetching Policies

In this section we describe how we extended our initial parallel I/O design for Blue Gene systems. The solution is based on the ROMIO implementation of the MPI-IO standard [15] and ZOIDFS and is shown in Figure 2(b).

On the *compute node side*, the file system calls performed through MPI-IO are mapped to ADIO. ADIO [16] is an abstract device interface, which consists of general-purpose components and a file-system specific implementation.In this case ADIO maps the calls to the ZOIDFS file system. The general-purpose components are the collective cache, the write-back module, and the prefetching module. In this paper we describe the design, implementation, and evaluation of the write-back module. The compute node prefetching module is part of our current efforts.

On the *I/O node side*, ZOIDFS maps the forwarded calls to specific file system calls (in our case PVFS). Between ZOIDFS and the file system there is a file cache accessed by the I/O node write-back and prefetching modules. The write-back module was implemented in our previous work and its performance evaluated for BG/L. The novelties presented in this paper are the prefetching module and an evaluation of the whole solution on BG/P systems. The write-back mechanism is implemented at each I/O node in one I/O thread. The I/O thread asynchronously writes file blocks to the file system following a high-low watermark policy, where the watermark is the number of dirty pages. When the high watermark is reached, the I/O thread is activated. The I/O thread flushes to the file system the last modified pages until the low watermark is reached.

In our solution, file consistency is ensured by assigning a file block exclusively to one aggregator and caching the respective file block only on the I/O node responsible for the pset of the aggregator. Therefore, all the processes send their accesses to the aggregators over the torus network, and in its turn each aggregator exclusively accesses a nonreplicated cache block at I/O node over the tree network.

4.1 Write-Back Collective I/O

Our initial Blue Gene/L collective I/O solution was based on view-based collective I/O [17]. In view-based I/O each file block is uniquely mapped to one process called aggregator, which is responsible to perform the file access on behalf of all processes of the application. A view is an MPI mechanism that allows application to see noncontiguous regions of a file as contiguous. View-based I/O leverages this mechanism for implementing an efficient collective I/O strategy. When defined, the views are sent to aggregators, where they can be used by all the accesses. View-based I/O writes and reads can take advantage of collective buffers managed by aggregators, which are responsible for performing the file access on behalf of all processes of the application. At access time, contiguous view data can be transferred between compute nodes and aggregators: using the view parameters, the aggregator can perform locally the scatter/gather operations between view data and file blocks.

In our initial Blue Gene/L solution, the modified collective buffers were transfered to the I/O node either when the memory was full or when the file was closed. This approach was taken because the Blue Gene/L did not offer support for threads on the compute nodes.

For Blue Gene/P, however, the collective buffer cache is asynchronously written back to the file system by an I/O thread. The write-back allows for overlapping the computation and I/O, by gradually transferring the data from the collective buffer cache to the I/O nodes. This approach distributes the cost of the file access over the computation phase and is especially efficient for scientific applications, which alternate computation and I/O phases.

The new implementation of our hierarchical I/O cache includes a two-level asynchronous file write strategy. The compute nodes asynchronously write back data to the I/O nodes, while the I/O nodes write asynchronously back data to the storage system.

4.2 Asynchronous I/O Node Prefetching

Another contribution of this paper is the design and implementation of a prefetching module at I/O node. The objective of I/O node prefetching is to hide the latency of read operations by predicting the future accesses of compute nodes.

The prefetching mechanism is leveraged by each I/O thread running at the I/O node. The prediction is based on the round-robin collective buffer distribution over the aggregators. Given that all collective buffers of an aggregator are mapped on the same I/O node, we implemented a simple strided prefetching policy. When an on-demand or a prefetch read for a collective block corresponding to an aggregator returns, a new prefetch is issued for the next file block of the same aggregator. The prefetching is performed by the I/O thread, while on-demand reads are issued by the thread serving the requesting compute node. A cache block, for which a read request has been issued, is blocked in memory, and all threads requesting data from the same page block at a condition variable. A further relaxation of this approach by allowing reads from page hits to bypass reads from page misses is possible, but it has not been implemented in the current prototype.

In the current implementation, only the I/O nodes perform prefetching into their local caches. However, we are developing an additional layer of prefetching between compute nodes and I/O nodes and its integration with the I/O prefetching module at I/O node presented in this paper.

5 Experimental Results

The experiments presented in this paper have been performed on the Blue Gene/P system from Argonne National Laboratory. The system has 1024 quad-core 850 MHz PowerPC 450 processors with 2 GB of RAM. All the experiments were run in Symmetric Multiprocessor mode (SMP), in which a compute node executes one MPI process per node with up to four threads per process. The PVFS file system at Argonne consists of four servers. The PVFS files were striped over all four servers with a stripe size of 4 MB.

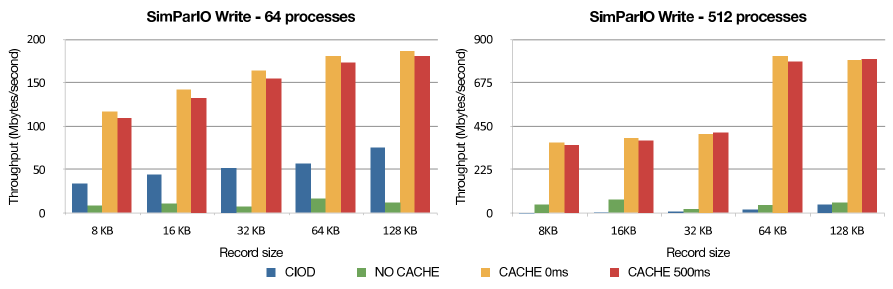

Fig. 3. SimParIO write performance for different record sizes for 64 and 512 processors.

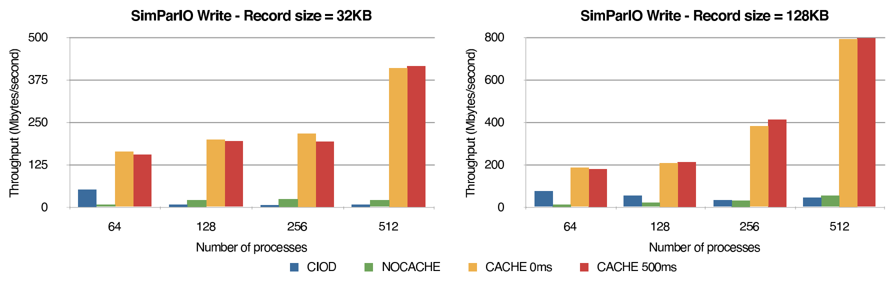

Fig. 4. SimParIO write performance for a fixed record size of 32KB and 128KB.

5.1 SimParIO

We have implemented an MPI benchmark called SimParIO that simulates the behavior of data-intensive parallel applications. The benchmark consists of alternating compute and I/O phases. The compute phases are simulated by idle spinning. In the I/O phase all the MPI processes write nonoverlapping records to a file. The number of alternating phases was 20. The maximum file size produced by 512 processes and record size of 128 KB was 1.25 GB. The compute nodes do not use any caching.

We compare four cases: IBM's CIOD-based solution, ZOID without cache, ZOID with cache and zero-time compute phase, and ZOID with cache with a 0.5 seconds compute phase. In one setup we have fixed the record size (we report two cases: 128 KB and 1 MB) and varied the number of processors from 64 (one pset) to 512 (8 psets). In the second setup we use a fixed number of processors (we report two cases: 64 and 512 processors) and vary the record size from 1 KB to 128 KB. Figures 3 and 4 show the aggregate file write throughput results. Figures 5 and 6 plot the aggregate file read throughput.

As expected, the file writes based on the write-back cache significantly outperform the CIOD-based and the no-cached solution in all cases. The duration of the compute phase does not seem to influence the performance of the write-back strategy. This indicates a good overlap of communication from the compute nodes to I/O nodes and the write-back from compute nodes to the file system.

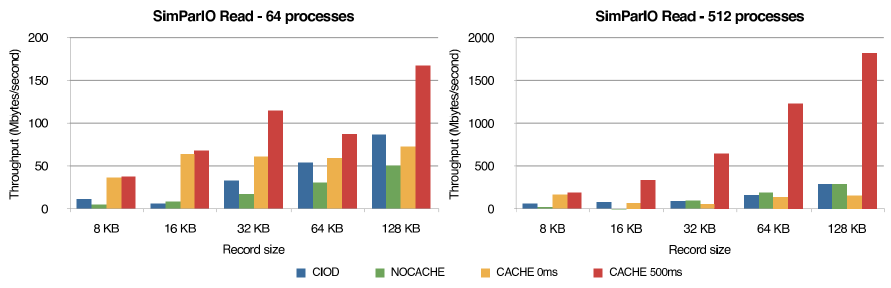

Fig. 5. SimParIO read performance for different record sizes for 64 and 512 processors.

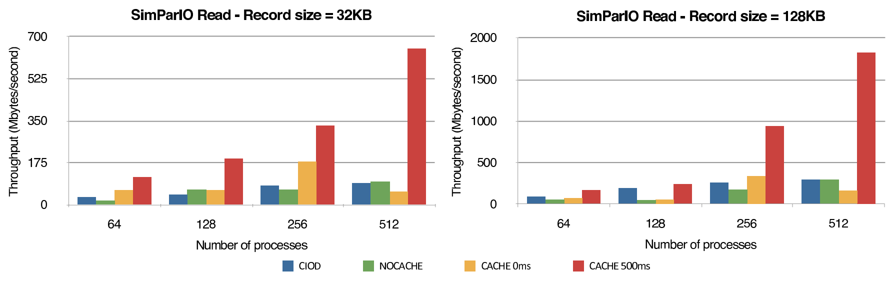

Fig. 6. SimParIO read performance for a fixed record size of 32KB and 128KB.

For file reads, the prefetching policy at I/O nodes works especially well for compute phases of 0.5 seconds. This performance benefit comes from the overlap between computation and file reads at I/O nodes. For 0 seconds compute phases, all 64 processes of a pnode request file reads on-demand roughly at the same time and reissue a new on-demand read request as soon as the previous one has been served. In this case the current prefetching implementation has little margin for improvement, and its costs may even outweigh the benefits.

5.2 BTIO benchmark

NASA's BTIO benchmark [18] solves the block-tridiagonal (BT) problem, which employs a complex domain decomposition across a square number of compute nodes. The execution alternates computation and I/O phases. Initially, all compute nodes collectively open a file and declare views on the relevant file regions. After each five computing steps the compute nodes write the solution to a file through a collective operation. At the end, the resulting file is collectively read and the solution verified for correctness. The aggregator cache on each compute node has 128 MB, while the I/O node buffer cache had 512 MB. All nodes acted as aggregators. We report the results for class B producing a file of 1.6 GB.

We have run the BTIO benchmark with four variants of collective I/O file writes: two-phase collective I/O from ROMIO, view-based I/O with no write-back (VB-original), view-based I/O with one-level compute-node write-back

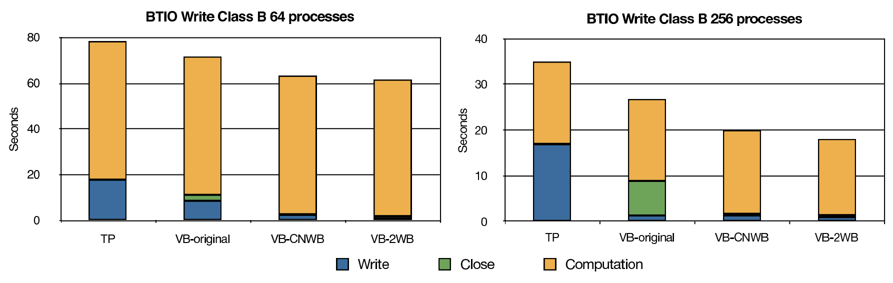

Fig. 7. BTIO class B times for 64 and 256 processors.

(VB-CNWB), and view-based I/O with two-level write back (VB-2WB). Figure 7 shows the breakdown of time into compute time, file write time, and close time. The close time is relevant because all the data is flushed to the file system when the file is closed. We notice that in all solutions the compute time is roughly the same. VB-original reduces the file write time without any asynchronous access. VB-CNWB reduces both the write time and close time, as data is asynchronously written from compute node to I/O node. For VB-2WB, the network and I/O activity are almost entirely overlapped with computation. We conclude that the performance of the file writes gradually improved with the increasing degree of asynchrony in the system.

6 Conclusions and Future Work

In this paper we describe the novel implementation of a multiple-level cache architecture for Blue Gene systems. The novel solution includes a compute-node write-back policy, which asynchronously transfers data from the compute nodes to the I/O nodes. This policy complements the initial data-staging strategy, which was performing asynchronous transfers between I/O nodes and the file system. Additionally, we implemented and evaluated an I/O node prefetching strategy, which asynchronously transfers file data from the file system to the I/O node. Both the data-staging and prefetching strategies provide a significant performance benefit, whose main source comes from the efficient utilization of the Blue Gene parallelism and asynchronous transfers across file cache hierarchy.

There are several directions we are currently working on. The solution presented in this paper is specific for Blue Gene architectures. However, the compute node part runs unmodified on any cluster, by leveraging the ADIO interface. In our cluster solution the I/O node cache level is managed by AHPIOS middleware [19]. Additionally, we are implementing and evaluating the prefetching module on the compute node. Finally, we plan to perform a cost-benefit analysis of various coordination policies among the several levels of caches.

Acknowledgments

This work was supported in part by Spanish Ministry of Science and Innovation under the project TIN 2007/6309, by the U.S. Dept. of Energy under Contracts DE-FC02-07ER25808, DE-FC02-01ER25485, and DE-AC02-06CH11357, and NSF HECURA CCF-0621443, NSF SDCIOCI-0724599, and NSF ST-HEC CCF-0444405.

References

1. http://www.top500.org: Top 500 list
2. F. Isaila, J. Garcia, J. Carretero, R. Latham, S. Lang, R. Ross: Latency hiding file I/O for Blue Gene systems. In: CCGRID '09. (2009)
3. Ma, X., Winslett, M., Lee, J., Yu, S.: Improving MPI-IO output performance with active buffering plus threads. In: In Proceedings of the International Parallel and Distributed Processing Symposium, Society Press (2003) 22–26
4. Liao, W., Coloma, K., Choudhary, A.N., Ward, L.: Cooperative write-behind data buffering for MPI I/O. In: PVM/MPI. (2005) 102–109
5. Patterson, R.H., Gibson, G.A., Ginting, E., Stodolsky, D., Zelenka, J.: Informed prefetching and caching. SIGOPS Oper. Syst. Rev. **29**(5) (1995) 79–95
6. Byna, S., Chen, Y., Sun, X.H., Thakur, R., Gropp, W.: Parallel i/o prefetching using mpi file caching and i/o signatures. In: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Piscataway, NJ, USA, IEEE Press (2008) 1–12
7. Chang, F., Gibson, G.: Automatic I/O hint generation through speculative execution. In: Proceedings of OSDI. (1999)
8. Chen, Y., Byna, S., Sun, X.H., Thakur, R., Gropp, W.: Hiding I/O latency with pre-execution prefetching for parallel applications. In: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press (2008) 1–10
9. Yu, H., Sahoo, R.K., Howson, C., Almasi, G., Castanos, J.G., Gupta, M., Moreira, J.E., Parker, J.J., Engelsiepen, T.E., Ross, R., Thakur, R., Latham, R., Gropp, W.D.: High performance file I/O for the Blue Gene/L supercomputer. The Twelfth International Symposium on High-Performance Computer Architecture
10. Yu, W., Vetter, J.S., Canon, R.S.: OPAL: An open-source MPI-IO Library over Cray XT. In: SNAPI '07: Proceedings of the Fourth International Workshop on Storage Network Architecture and Parallel I/Os, IEEE Computer Society (2007) 41–46
11. Yu, W., Vetter, J.: Parcoll: Partitioned collective I/O on the Cray XT. 37th International Conference on Parallel Processing **0** (2008) 562–569
12. Moreira, J., Brutman, M., José Casta n., Engelsiepen, T., Giampapa, M., Gooding, T., Haskin, R., Inglett, T., Lieber, D., McCarthy, P., Mundy, M., Parker, J., Wallenfelt, B.: Designing a highly-scalable operating system: The Blue Gene/L story. In: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, ACM (2006) 118
13. http://www unix.mcs.anl.gov/zeptoos/: ZeptoOs Project. (2008)
14. Iskra, K., Romein, J.W., Yoshii, K., Beckman, P.: ZOID: I/O-forwarding infrastructure for petascale architectures. In: PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM (2008) 153–162

15. Thakur, R., Gropp, W., Lusk, E.: On implementing MPI-IO portably and with high performance. In: Proc. of the Sixth Workshop on I/O in Parallel and Distributed Systems. (May 1999) 23–32
16. Thakur, R., Lusk, E.: An abstract-device interface for implementing portable parallel-I/O interfaces. In: Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society Press (1996) 180–187
17. Blas, J.G., Isaila, F., Singh, D.E., Carretero, J.: View-based collective I/O for MPI-IO. In: CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, IEEE Computer Society (2008) 409–416
18. Wong, P., der Wijngaart, R.: NAS Parallel Benchmarks I/O Version 2.4. Technical Report NAS-03-002, NASA Ames Research Center, Moffet Field, CA (January 2003)
19. Isaila, F., Blas, J.G., Carretero, J., Liao, W.K., Choudhary, A.: AHPIOS: An MPI-based ad-hoc parallel I/O system. In: Proceedings of IEEE ICPADS. (2008)