A New Workload Model for Parallel I/O Systems

Daniel Versick
University of Rostock, Institute of Computer Science
Rostock, Germany
e-mail: daniel.versick@uni-rostock.de

Djamshid Tavangarian
University of Rostock, Institute of Computer Science
Rostock, Germany
e-mail: djamshid.tavangarian@uni-rostock.de

ABSTRACT
The analysis of computational systems for high performance computing (HPC) applications has been examined in many scientific surveys in the past. However, the input/output (I/O) of such applications has been marginally considered. There are some I/O benchmarks for MPI-IO, a standard interface for I/O access to secondary storage in HPC environments, which often use a static workload that cannot be changed by the user. Therefore, resulting measurements are often not representing user-relevant applications. This paper describes a novel I/O measurement approach for MPI-IO based applications enabling the user to employ any I/O workload he wants to use. Thus, it is necessary to define a workload model specifying any I/O workload in a simple, but exact way. This paper introduces such an I/O workload model for MPI-IO and an I/O benchmark using this workload model. Practical tests describe how such a workload can be generated and present measurements to prove the introduced approach.

Keywords
I/O benchmarking, high performance I/O, workload model

1. INTRODUCTION
Performance evaluation of parallel computer systems usually examines the performance of CPUs and network connections. This is an acceptable way when analyzing applications that only load or store small amounts of data. However, there is an increasing number of data-intensive applications that have to read large amounts of information from the file system. For example, NWChem [1] and QCRD (Quantum Chemical Reaction Dynamics) [2] are codes that have to read large matrices from secondary storage that are not fitting into main memory. Their input/output (I/O) behavior has a big impact on the whole application performance. As the performance of data transfer from I/O systems to CPUs does not increase as fast as the processor speed, future CPUs will not be able to exploit full computational capabilities. An accurate measurement and optimization of I/O systems is necessary to minimize their negative impact on performance [3]. It also supports the development of future highly efficient I/O systems.

There are three ways of analyzing application performance: analytical modeling, simulations, and measurements. Whereas all methods have their use cases, measurements are the only possibility of performance analysis comprising the whole complexity of an I/O system in the field. In case of measuring computer systems performance, benchmarks are widely-used measurement tools [4]. Benchmarks are applications that generate a specific workload and measure its performance when processing this workload. I/O benchmarks as a special form of benchmarks use an I/O workload and measure I/O performance. Most existing I/O benchmarks employ static workloads that can only be adapted in a few or even no parameters [5]. Thus, these benchmarks deploy one specific workload that is characteristic for only one application. Other user-defined workloads that are necessary for user-relevant results are not applicable. This paper presents a novel I/O benchmark enabling the user to define any I/O workload and thus to analyze the I/O performance of any possible application. The benchmark system focuses on the definition of workloads using the MPI-IO interface that is part of the MPI-2 (Message Passing Interface) standard for HPC applications [6] and has been specified for parallel data access of high performance applications to secondary storage systems.

At the beginning, this paper introduces an I/O workload model as the basis for a novel benchmark approach that allows to define any I/O workload for MPI-IO. Afterwards, we demonstrate a benchmark application implementing this novel approach before showing measurement results to prove the concept.

2. I/O BENCHMARKING
Existing I/O benchmarks often employ static I/O workloads defined by the programmer of the benchmark software. Sometimes this workload characterizes a common application behavior of an existing application (e. g. by using the application itself or by reproducing the workload as close as possible). Such application benchmarks determine the I/O performance of a computer system executing the application that is reproduced by the benchmark. Other applications that could be much more interesting for the user are not included by this approach. Their performance can vary very strongly from performance values determined by application benchmarks. Besides the application benchmarks, there are synthetic benchmarks that use workloads representing no currently available application. They perform operations defined by the programmer attempting to represent many applications as best as possible. Indeed, these benchmarks measure the performance of an ,,average application" and not of the specific one that is interesting for the user [7]. Only I/O benchmarks allowing to handle any I/O workload can be used to measure exact I/O performance of any user-defined application. Such software tools will be called application-based I/O benchmarks. Currently there are only a few application-based I/O benchmarks especially for the POSIX-I/O interface. POSIX-I/O is mainly used on UNIX (and UNIX-like) systems to access files in a file system. The well-known IOzone handles I/O workloads defined

in two files, one with write and one with read telemetry information. Every line in each file represents one I/O call and consists of a byte offset, size of transfer and compute delay in milliseconds. Thus, user-defined I/O workloads can be realized, whereas some parameters of the I/O behavior like the synchronism or the mix of read and write calls cannot be specified. In [8] a benchmark is introduced that realizes any POSIX-I/O behavior. It defines five parameters (called dimensions) for POSIX-I/O workloads. The whole I/O workload is summarized to these five parameters resulting in a very abstract definition of an I/O workload.

Parallel applications have special requirements when accessing secondary storage. Often the applications are processing big matrices. Their input and output results in I/O access with non-contiguous character which means, that processes write alternative bytes of a larger data type. Especially for such applications parallel I/O interfaces are specified. MPI-IO as part of the MPI-2 standard specification is such an I/O interface for parallel I/O. There are some I/O benchmarks for MPI-IO but most of them use one or more static workloads (like b_eff_io [9], FLASH-I/O, LANL MPIO-IO Test, mpi-tile-io, NPBIO [10], Parallel Input / Output Test Suite, and PIO-Bench). The workload of IOR [11] in its newest version and the HPIO (formerly known as NCIO) [12] can be configured more flexible. IOR supports several parameters, but it is not possible to specify asynchronous I/O and any combination of read and write access. HPIO has the goal to measure methods of accessing secondary storage via so-called non-contiguous I/O. The access pattern of non-contiguous I/O can be varied by using three parameters. Therefore, HPIO allows to change the non-contiguous I/O in an arbitrary manner without allowing to change the I/O access methods itself (e. g. asynchronous I/O is not possible).

Existing I/O benchmarks only apply limited workload descriptions that are caused by a sketchy or even missing analysis of important parameters for describing I/O workloads. Therefore, existing I/O workload models for parallel I/O are not sufficient to specify any relevant I/O workload of user applications. This paper presents a novel workload model for parallel I/O that is developed by analyzing the MPI-IO interface. Hence, it includes the most important parameters for characterizing I/O workloads of parallel applications. It also introduces an I/O benchmark using this workload model and shows measurements that prove the validity of the presented approach.

3. I/O WORKLOAD MODEL FOR MPI-IO

MPI-IO is a complex I/O interface designed for the specific requirements of parallel application I/O access. It supports an optimized access of multiple processes to one large file assuming the file system supports a high performance parallel access to secondary storage. In the following we introduce an I/O workload model for MPI-IO that covers the important performance-related options of MPI-IO. Therefor, parameters are defined that can be used to quantify specific attributes of I/O calls of parallel applications. Moreover, this paper presents, how an I/O workload can be generated using these I/O workload parameters.

3.1 I/O Workload Parameters

Table 1 shows all data access functions of MPI-IO as specified in [6]. Any of these data access methods has at least the following I/O-related parameters:

- file handle that is returned when opening the file
- initial address of data in main memory (begin of buffer)
- number of data elements in buffer
- data type of each buffer element

These function parameters are the basis for first parameters of the MPI-IO workload model. The file handle is a unique handle defining a specific file in the file system and is returned when opening the file by the MPI_File_open() call. MPI_File_open() itself needs a file name, mode bits for defining access rights to the file as well as an MPI communicator, that is a data structure for defining the processes that open the file concurrently. From the mentioned parameters the number of processes is performance-related because the more processes are involved in I/O the more processes have to divide the bandwidth to access the file. Access rights define which process is allowed to access the file by reading or writing methods but are not performance-related. The file name defines which file has to be accessed. Every file has a special file structure in the file system which could have a performance impact. Files with a random order of blocks are accessed slower than sequentially ordered blocks when using a sequential access method. Since the I/O workload model should abstract from specific data to allow the execution of the developed benchmark on every computer system, the specific file layout is ignored in the following. The initial address of data in main memory has a performance influence in case that the computer system is a so-called non-uniform-memory-architecture (NUMA) where memory regions belonging to the current node are faster accessible than those who correspond to other computer nodes. On widely-used uniform-memory-architectures (UMA) the initial address of data in main memory has no performance influence. Hence, this parameter is not included in the introduced workload model. The number of data elements and the data type (respectively the size) of each buffer element are fundamental workload parameters influencing the I/O performance when using the workload.

As it is depicted in Table 1 every data access function is existing as a reading version and a writing counterpart. There can be defined a value that specifies all read data of an application as a fraction of the completely transfered data. These considerations lead to the following parameters of I/O requests that are named as in [8]:

- **processNum**– the number of processes involved in an I/O operation
- **sizeMean**– average size of one buffer element in bytes
- **uniqueBytes**– whole amount of transfered data in bytes (the number of buffer elements multiplied with its size)
- **readFrac**– fraction of read bytes to all transfered bytes

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_File_read_at MPI_File_write_at	MPI_File_read_at_all MPI_File_write_at_all
	nonblocking & split collective	MPI_File_iread_at MPI_File_iwrite_at	MPI_File_read_at_all_begin MPI_File_read_at_all_end MPI_File_write_at_all_begin MPI_File_write_at_all_end
individual file pointers	*blocking*	MPI_File_read MPI_File_write	MPI_File_read_all MPI_File_write_all
	nonblocking & split collective	MPI_File_iread MPI_File_iwrite	MPI_File_read_all_begin MPI_File_read_all_end MPI_File_write_all_begin MPI_File_write_all_end
shared file pointer	*blocking*	MPI_File_read_shared MPI_File_write_shared	MPI_File_read_ordered MPI_File_write_ordered
	nonblocking & split collective	MPI_File_iread_shared MPI_File_iwrite_shared	MPI_File_read_ordered_begin MPI_File_read_ordered_end MPI_File_write_ordered_begin MPI_File_write_ordered_end

Table 1. Overview of MPI-IO data access functions

Table 1 also presents further parameters. It shows that every I/O function is existing in a non-collective and a collective

version as well as a blocking and a non-blocking counterpart. Collective I/O are I/O calls done by a group of processes. It is giving the possibility of collecting, reordering and optimizing the I/O calls of the involved processes. Non-blocking I/O that doesn't block processes during sending or receiving, gets more and more important in the high performance computing sector. It allows to calculate new values even while sending or receiving data concurrently. Thus, the computational power is used in a more efficient way. These two attributes result in two more parameters defined in our novel workload model:

- **collFrac**– fraction of collective I/O calls as part of all I/O calls
- **syncFrac**– fraction of blocking I/O calls as part of all I/O calls

Furthermore, Table 1 shows three kinds of setting the start position of data within a read or written file: explicit offsets, individual file pointers and shared file pointers. I/O calls with explicit offsets are writing at a position within a file that is specified as a function parameter of the call and usually result in a random access on disk memory. Access using individual file pointers use a file pointer defined per process that is automatically increased by every I/O call. Hence, individual file accesses result in sequential access behavior for every process. Shared file pointer accesses use one file pointer for all processes of an application. The access of the processes is sequentialized which is usually used for writing log files or similar. Anyhow, access to non-sequential addresses using shared file pointers can also be realized by using a seek-instruction before the data access call. Seeks change the file pointer position, and result in a non-sequential behavior. Hence, there can be defined two different parameters which represent the three described kinds of setting the start position:

- **seqFrac**– fraction of I/O calls accessing sequentially after the previous I/O call onto the disk memory
- **sharedFrac**– fraction of I/O calls with shared file pointer

Every I/O call employs the parameter data type of each buffer element. As shown before, this parameter is mapped to *sizeMean*, the average size of the transfered data elements. However, MPI-IO supports very complex data types composed of simple data types like characters. It is also possible to define ,,holes" within the data type that are not written by the specifying process. Different MPI processes can use different self constructed data types in the same I/O call. This results in the possibility of strided access where one process of an application for example writes the first and third byte of a data type and another process the second and fourth byte consecutively. This, so-called non-contiguous I/O cannot be covered by the parameter *sizeMean* only. Therefore, it is necessary to define one more parameter to specify the number of bytes that is accessed by every process. Actually, it would be necessary to specify such a parameter *contMean* for every process of an application separately. Since most high performance applications are doing similar computations and I/O operations in every process, in most cases *contMean* would be equal for all processes of the application. For this reason our introduced workload model uses one value of *contMean* for all attending processes describing the number of contiguously accessed bytes. This simplified model of the two parameters *contMean* and *sizeMean* is employed to specify MPI data types.

In summary, we defined nine parameters for specifying an I/O workload behavior. The five parameters *uniqueBytes*, *sizeMean*, *processNum*, *readFrac*, and *seqFrac* have also been specified by Chen and Patterson in [8]. They defined a workload model for classical non-parallel I/O. The new parameters *collFrac*, *syncFrac*, *sharedFrac*, and *contMean* have been introduced with special focus on parallel I/O. All these parameters can be employed to specify workloads of parallel I/O by abstracting the attributes of many I/O calls. In this way the whole application I/O behavior can be abstracted to nine parameter values.

3.2 Application I/O Workload

In [8] it is proposed to combine all I/O calls of an application to one I/O working point that would consist of the introduced nine workload parameters when using the described workload model. Because this approach produces a very abstract specification we propose to permit the user to decide how many I/O calls should be combined to one I/O working point. Hence, the user defines these nine parameter values only for a subset of all application I/O calls such in a way that the whole application I/O behavior is a sequential list of vectors each of them described by the nine parameters. Hence, one application I/O behavior can be specified using different descriptions that have varying granularities. High-granular descriptions would use one I/O working point (nine I/O workload parameters) for every I/O call, and low-granular descriptions utilize one working point for the whole I/O behavior of an application. Granularities in-between are also possible.

3.3 PRIOmark - Parallel I/O Benchmark

In the context of the IPACS (Integrated Performance Analysis of Computer Systems [7]) project we implemented an I/O benchmark using the described workload model. The PRIOmark - Parallel I/O benchmark [13] can be employed to emulate the I/O behavior of any application using the MPI-IO interface. I/O workloads have to be defined using an XML file that supports a concatenation of I/O working points with each of the introduced workload parameters. At the moment all but the *contMean* parameter are supported. *ContMean* will be part of the next software version. The generation of workloads can be done using two approaches: The user produces an XML workload definition with a graphical workload definition tool or he employs the so-called I/O profiler. This tool analyzes the I/O behavior of a running MPI-IO application and automatically produces an XML workload definition representing the analyzed application.

4. MEASUREMENTS

In the following we present measurements of the PRIOmark that prove the validity of the introduced workload model. They have been made on a small PC cluster consisting of four Linux-nodes and connected with Gigabit ethernet network interfaces. Three of the four nodes are used as I/O nodes (using IDE disks) that are combined to a big storage using PVFS (Parallel Virtual File System).

Figure 1 depicts the results of the first measurement. Basis of this measurement is an abstract workload representing an average I/O behavior of a workstation used for standard office work (*uniqueBytes*=1 GB, *sizeMean*=8 kB, *readFrac*=80 %, *seqFrac*=20 %, *processNum*=8, *syncFrac*=78 %, *collFrac*=0 %, *sharedFrac*=0 %). Using this workload eight measurements with changing values for every currently supported parameter have been arranged to see the impact of this parameter on the I/O performance. *ProcessNum* is eight for all measurements except for the last two (*sharedFrac*, *syncFrac*). There *processNum* had to be reduced to avoid crashes of the MPI-IO system that were caused by internal MPI errors. All measurements show the throughput while accessing the secondary storage depending on the parameter values. In most cases the results show an expected

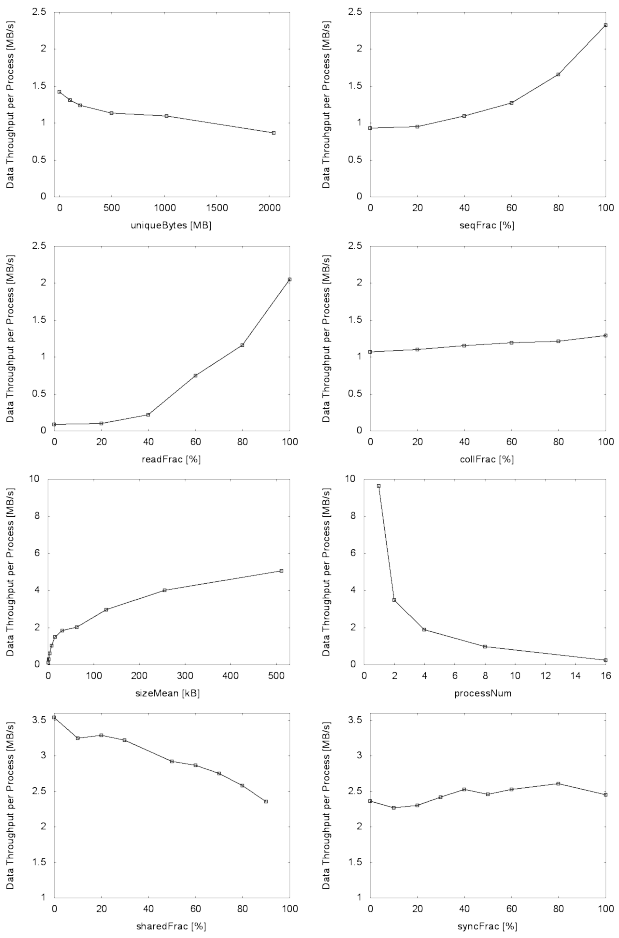

Figure 1. Variation of Workload Parameters

behavior. Rising values for uniqueBytes result in a reduced performance because caches cannot be used as efficiently. Sequential access behavior causes a better performance as well as a read-dominated workload. Collective I/O calls give a higher potential of optimization. Thus, more collective I/O calls result in a better I/O bandwidth as it is depicted in Figure 1. Many small I/O requests (small values for *sizeMean*) result in a poor I/O performance because every call produces an overhead degrading the performance. Shared file pointer access needs more synchronization between the processes resulting in a performance loss. The performance influence of non-blocking I/O cannot be seen very well in these measurements. But it can be seen that the performance rises up to a value of 80 per cent for *syncFrac*. After that there is a small performance loss. This behavior could be verified in other measurements, too. Probably the behavior emerged from the higher parallelism of asynchronous I/O and the related conflicts of concurrently used resources.

The measurements show the impact of eight of the introduced parameters on the I/O performance. The influence complies with the expected behavior which shows that the performance model can be used to represent many different workloads. Since the workload parameters map the important performance-related options of MPI-IO calls, it can be used to specify any relevant MPI-IO workload. In contrast to other I/O benchmarking systems the PRIOmark allows to define very complex I/O workloads and enables the user to acquire user-relevant results.

5. RESULTS

This paper introduced a novel approach of benchmarking parallel I/O systems. The approach enables users to define their own I/O workloads resulting in a very realistic I/O measurement that is relevant for the user. For benchmarking I/O systems based on a user-defined workload it is necessary to specify these workloads. Such a workload specification model had been introduced within this paper. The model specifies I/O workloads that consist of I/O working points each of them are defined by nine parameters. Finally, the paper presented measurements proving the concept.

6. ACKNOWLEDGEMENT

We thank the German Ministry of Education and Research for sponsoring this research as part of the Project *Integrated Performance Analysis of Computer Systems* with research grant 01IRB03E as well as our project partners Fraunhofer Institute for Techno- and Business Mathematics, T-Systems, National Energy Research Scientific Computing Center, and the University of Mannheim for supporting our work.

REFERENCES

[1] Molecular Sciences Software Group. NWChem User Documentation Release 5.1, 2007.

[2] Evgenia Smirni and Daniel Reed. Lessons from characterizing input/output behavior of parallel scientific applications. *Performance Evaluation*, 1245:169–180, November 1998.

[3] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.

[4] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., New York, 1991.

[5] Michael Krietemeyer, Daniel Versick, and Djamshid Tavangarian. The PRIOmark – Parallel I/O Benchmark. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 595 – 600, 2005.

[6] Message Passing Interface Forum. Mpi-2: Extensions to the message-passing interface. Technical report, University of Tennessee, Knoxville, nov 2003.

[7] Michael Krietemeyer, Henry Ristau, Daniel Versick, and Djamshid Tavangarian. Chapter 7. In Matthias Merz and Michael Krietemeyer, editors, *IPACS Benchmark - Integrated Performance Analysis of Computer Systems*, pages 121–156. Logos Verlag, 2006.

[8] Peter M. Chen and David A. Patterson. A new approach to I/O performance evaluation—self-scaling I/O benchmarks, predicted I/O performance. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 1–12, Santa Clara, CA, USA, 10–14 1993.

[9] Rolf Rabenseifner, Alice E. Koniges, Jean-Pierre Prost, and Richard Hedges. The Parallel Effective I/O Bandwidth Benchmark: b_eff_io. In *2e soumission a Calculateurs paralleles*, 11 2001.

[10] Parkson Wong and Rob F. Van der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical Report NAS-03-002, NASA Advanced Supercomputing (NAS) Division, 01 2003.

[11] W. Loewe, R. Hedges, McLarty T., and C. Morrone. Llnl's parallel i/o testing tools and techniques for asc parallel file systems. In *Proceedings of the 2004 IEEE Cluster Computing Conference*, San Diego, 2004.

[12] Avery Ching, Alok Choudhary, Wei Keng Liao, Lee Ward, and Neil Pundit. Evaluating I/O characteristics and methods for storing structured scientific data. In *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2006.

[13] Michael Krietemeyer, Heiko Kopp, Daniel Versick, and Djamshid Tavangarian. Environment for i/o performance measurement of distributed and local secondary storage. In *Proceedings of the International Conference on Parallel Processing Workshops*, pages 501–508. IEEE, 2005.