# A Methodology to characterize the parallel I/O of the message-passing scientific applications

**Sandra Méndez, Dolores Rexachs and Emilio Luque**
Computer Architecture and Operating Systems Department (CAOS)
Universitat Autònoma de Barcelona, Barcelona, Spain

**Abstract**— *The increase in computational power of processing units and the complexity of scientific applications which use high performance computing require more efficient Input/Output (I/O) systems. To use the I/O subsystems efficiently it is necessary to know its performance capacity to determine if it fulfills applications I/O requirements. Evaluating the performance capacity of the I/O subsystem is difficult due to the diversity of architectures and the complexity of its software stack.*

*Furthermore, parallel scientific applications have different behavior depending on their access patterns. Then, it is necessary to have some method to evaluate the I/O subsystem capacity taking into account the applications access patterns that can be used in different I/O subsystems.*

*We propose a methodology to characterize the parallel I/O of scientific applications, including the I/O subsystem at library and devices levels. We represent the message-passing applications through an I/O model. The model allows us to evaluate the I/O subsystem taking into account the I/O phases of the application.*

**Keywords:** Parallel I/O System, Access Pattern, I/O Configuration, I/O Modeling, I/O phases

## 1. Introduction

Due to the historical “gap“ between the computing and Input/Output (I/O) performance, the I/O system is, in many cases, the bottleneck in parallel systems. Increasing computational power of processing units and the complexity of scientific applications that use high performance computing require more efficient Input/Output systems. In order to hide the "gap" and to efficiently use the I/O, it is necessary to identify the I/O factors with the biggest effect on performance. The I/O factors depend on the I/O architecture and I/O software stack, however the application performance will depend on its access pattern.

Computer clusters are built to provide parallel computing to several applications. These applications have different I/O requirements and the same I/O system is not always appropriate for all applications. Programmers can modify their programs to efficiently manage I/O operations, but they need to know the I/O system, especially the I/O software stack.

Users need information to answer questions like: Is the I/O subsystem a problem for the access patterns of the application? How much I/O subsystem capacity is being used by the application? How the application access patterns are done in a target subsystem?

We use an I/O model of the scientific applications to support the evaluation of I/O performance of computer clusters. We have implemented a tracing tool (PAS2P library extension [1]) for extracting "I/O patterns" in message-passing applications, and based on these patterns, we have defined the concept of "I/O phase" of parallel scientific applications. These I/O phases are the key elements to define an I/O behavior model of the parallel scientific applications. This model allows us to evaluate an I/O subsystem taking into account the parallel application.

In a previous paper [2], we have proposed a methodology for performance evaluation of the I/O system which was focused in I/O path. In the present paper, we extend our methodology focused in the I/O characterization of the application. We have refined the stages of characterization and analysis of the I/O subsystem. We explain the process to performance evaluate of the I/O subsystem focused in the I/O model.

We have applied our methodology in three computer clusters where we have used parallel filesystem OrangeFS, Lustre and network filesystem NFS. We have evaluated the performance on three I/O systems taking into account the I/O model of the application. The methodology is applied to MadBench2 [3] and Flash-IO [4]. The characteristics of the I/O subsystem are evaluated, as well as their usage by the different I/O phases of the I/O model.

The rest of this article is organized as follows: in Section II we review the related work, Section III introduces our proposed methodology. In Section IV we review the experimental validation. Finally, we present our conclusions and future work.

## 2. Related Work

There are several papers [5] [6] [7] [8] that present the characterization of I/O of parallel applications in specific computers clusters or supercomputers. Due to the diversity of I/O architectures and the complexity of stack software, each researcher try to evaluate the I/O components with the

biggest impact in their subsystems. Usually the evaluation is done for I/O benchmarking and the results are used to evaluate the performance of an specific I/O system.

Other important point in the evaluation of I/O subsystem is the tracing tool to identify the I/O access pattern.

Byna et. al. [9] presented a classification of I/O patterns for parallel applications, I/O signatures at local process level and an applying of signatures to prefetching technicals. We use their proposed to identify access patterns. However, we have identified the global access pattern because we need the I/O for the parallel application. From local access patterns and by similarity, we have defined the global access pattern, then global access patterns are divided in the I/O phases.

H. Shan and J. Shalf [10] have used IOR to mimic the I/O pattern of parallel scientific applications. Also, they used this mimic to predict the performance for the application. We have used IOR to represent the I/O model of the application. The I/O model is represented by an I/O phases sequence and IOR is applied to each I/O phases. In this way, we only focus in time where the application does I/O operations.

Carns [11] presented the Darshan tracing tool for the I/O workloads characterization of the petascale. Darshan is designed to capture an accurate picture of the application I/O behavior, including properties such as patterns of access within files, with minimum overhead. It is a tools available to download and it is free.

Most of these researches are aimed at supercomputers, while our strategy is focused on computer clusters. However, the main difference is that our methodology is focused to evaluate the performance capacity of I/O subsystem from an I/O model. We use the model to describe the I/O requirements of the application and to compare qualitatively the I/O subsystems. We have expressed the access patterns of the application in an I/O model and this can be used on different I/O subsystems.

# 3. Proposed Methodology

The proposed methodology is composed of three stages: Characterization, I/O Analysis and Evaluation. Next, we explain each stage.

## 3.1 Characterization

The characterization is applied to the I/O subsystem and parallel scientific application. This stage has two objectives: i) Extracting the I/O model of the application; and ii) Identifying and obtaining of performance basic characteristics of the different configurations in the I/O subsystem. These activities are independent.

### 3.1.1 Scientific Application

The I/O model of an application is represented by three major characteristics: i) meta-data, ii) the temporal global I/O pattern; and iii) the spatial global I/O pattern.

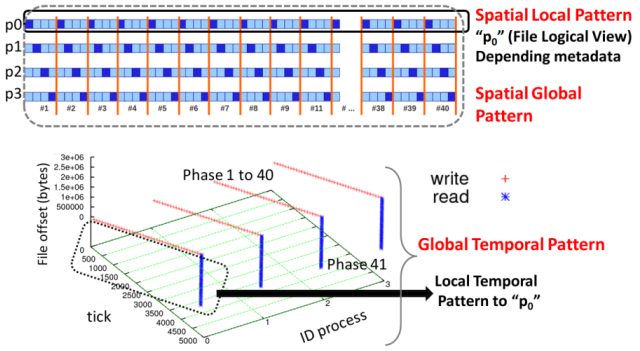



Fig. 1: I/O model example for 4 processes with request size 2MB, 40 I/O phases of a writing operation and weight 40MB, 1 phase of 40 reading operations with weight 1600MB

We characterize the application off-line and once at I/O library level because this allows us to obtain a model of the application's I/O independent from the execution environment, i.e. the computer cluster.

The I/O model of application is expressed by I/O phases, where an I/O phase is a repetitive sequence of same pattern on a file for a number of processes of the parallel application. The process to extract the I/O model is described in [12].

In order to consider the order of occurrence of the events of message-passing parallel applications we use the concept of tick. A tick is defined as a logical unit time, and it is incremented by each communication event and I/O event. An I/O event is a segment of application where the I/O operation is called. The I/O event is composed of an ID file, mpi-io operation, offset, displacement, size of etype, size of filetype, name of file, number of event, size of request, logical time, duration, count of datatype, and size of datatype.

The algorithm to identify the I/O patterns is based on type of operation, request size, displacement, and distance. Where, distance is the number of tick between two I/O operations and the displacement is the difference between the offset of two consecutive I/O operations. The similarity of pattern is determined by the relation between the pattern and the new value. If the relation is $> 0.8$ and $< 1.2$ then we consider the new value how a new occurrence of the pattern analyzed. This criteria is used to the displacement and the distance. The new value must be equal to the type of operation and the request size of the pattern analyzed to be considered a new occurrence.

The weight of a phase depends on the number of processes, request size and repetitions of each access pattern that is part of a phase. The weight is expressed in Megabytes and it is used to determine transferred data in each I/O phase. The I/O model depends on I/O phases and weight, allowing us to know "when" and "how" the I/O subsystem will be used.

Figure 1 shows an example of I/O model, where the global access pattern is shown through its spatial local pattern,

spatial global pattern, temporal local pattern, and temporal global pattern. Also, we show the global access pattern in three dimensional space, where for each operation the *file Offset* indicates the position where the *process* "p" is accessing in the *tick* "t".

#### 3.1.2 I/O System

The I/O subsystem characterization has as objective to development of adequate yardsticks for measuring and comparing such configurations in a appropriate manner taking into account the application access patterns. To do this, we apply the following steps:

i) Identifying I/O configurations: In this step, we identify the I/O subsystem configurations. An I/O configuration depends on number and type of filesystem (local, distributed and parallel), number and type of network (dedicated use and shared with the computing), number of I/O devices, I/O devices organization (RAID level, JBOD), and number and placement of I/O node.

ii) Setting input parameters for the Benchmarks: IOR [13] is applied at I/O library level and Global Filesystem level. IOzone [14] is applied at I/O devices level on local filesystem. Parameters values are selected according to the characteristics of the configurations identified. We consider that minimum size of file to test must be $= 2 * RAMsize$, where the RAM size is of the node where the benchmark will be executed. This is necessary to guarantee the access to disk. The access mode to a file can be sequential, strided, and random. The access type can be shared or unique, where shared is one file for all processes and unique is one file per process. Also, it is necessary to select the type of operations (write, read) and the request size (KBytes, MBytes, Gbytes) of the operations.

We have executed the benchmarks in each I/O subsystem with different I/O patterns and we have generated a data base by I/O subsystem configuration with performance measures (bandwidth, latency, iops).

### 3.2 Input/Output Analysis

To compare the I/O pattern of the application with benchmarks, we define the similar data structure for the access pattern of each I/O phase.

We analyze the I/O phases of the application and its weight to select the candidate I/O system. We search the I/O patterns of phases on performance databases. Then, we calculate the I/O time for the I/O phases and we select the I/O systems with less I/O time.

When there is not a characterization of I/O subsystem for similar patterns to the most significant phases of I/O model we use the I/O model to mimic the I/O model of the application. The I/O model is used to set up the input parameters of the benchmark IOR [13]. We only execute the benchmark for the phases with higher weight of the I/O model.

The following setting of input parameters are applied on IOR for each I/O phase:

- Strided Access: $s = Iter$; $b = RS_{(IdPh)}$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$; $-F$ if there is 1 file per process; $-c$ if there is collective I/O.
- Sequential Access: $s = 1$; $b = weight(IdPh)$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$; $-F$ if there is 1 file per process; $-c$ if there is collective I/O.

Then IOR is run in the subsystem target. I/O time and transfer rate obtained from IOR running are named $Time_{io(CH)}$ and $BW_{(CH)}$. The estimated I/O time is calculated by expression (1).

$$Time_{io} = \sum_{i=1}^{n} Time_{io}(phase[i]) \quad (1)$$

Where the $Time_{io}(phase[i])$ is I/O time for each I/O phase that is calculated by expression (2).

$$Time_{io}(phase[i]) = \frac{weight_{(phase[i])}}{BW_{(CH)}(phase[i])} \quad (2)$$

$BW_{(CH)}(phase[i])$ is the characterized transfer rate at I/O library level for a similar access pattern.

### 3.3 Evaluation

We evaluate the utilization of I/O subsystem by the relation between the bandwidth characterized $BW_{(PK)}$ at I/O devices level and measured $BW_{(MD)}$ when the application is executed, expressed in equation 3.

$$SystemUsage(phase[i]) = \frac{BW_{(MD)}(phase[i])}{BW_{PK(IOP(phase[i]))}} * 100 \quad (3)$$

The I/O model is used to determine what system can provide the best I/O performance for the I/O phase with more impact in the I/O of the application. To evaluate the estimation's accuracy of the I/O time estimated we evaluate the relative error produced by the I/O time estimation. Relative error is calculated by expression (4); where $Time_{io(MD)}$ and $BW_{(MD)}$ are the I/O time and transfer rate obtained from running of application.

$$error_{rel} = 100 * (\frac{error_{abs}}{Time_{io(MD)}(phase[i])}) \quad (4)$$

Where absolute error is calculated by the expression (5).

$$error_{abs} = |Time_{io(CH)}(phase[i]) - Time_{io(MD)}(phase[i])| \quad (5)$$

## 4. Experimentation

We present the experiments in two part: 1) We extract the I/O model of MadBench2 [3] and we used it to evaluate the utilization of the two I/O subsystems. This approach is adequate when there is an exhaustive characterization of the I/O subsystem. 2) We extract the I/O model of Flash-IO

Table 1: Description of the I/O subsystems of systems A and B

| I/O Element | System A | System B |
|---|---|---|
| I/O library | mpich2 | OpenMPI |
| Communication Network | 1 Gbps Ethernet | 1 Gbps Ethernet |
| Storage Network | 1 Gbps Ethernet | 1 Gbps Ethernet |
| Filesystem Global | OrangeFS | NFS Ver 3 |
| I/O nodes | 10 | 32 DAS and 1 NAS |
| Metadata Server | 1 | 1 |
| Filesystem Local | Linux ext3 | Linux ext4 |
| Level Redundancy | - | RAID 5 |
| Number of I/O Devices | 11 disks | 5 disks |
| Capacity of I/O Devices | 500 GB | 1.8 TB |
| Mounting Point | /mnt/orengafs | /home |

benchmark [4] and we used it to tune parameters of IOR benchmark in order to evaluate the I/O performance for the I/O model. This approach is adequate when there is not an exhaustive characterization of the I/O subsystem. Table 1 shows the I/O subsystems of the System A and B.

System A is composed of 14 computing nodes:

- 4 cores of AMD Phenom™ II (8MB cache) or Athlon™ II (2MB cache), 4 DIMM slots for up to 16GB DDR3

System B is composed of 32 IBM x3550 Nodes:

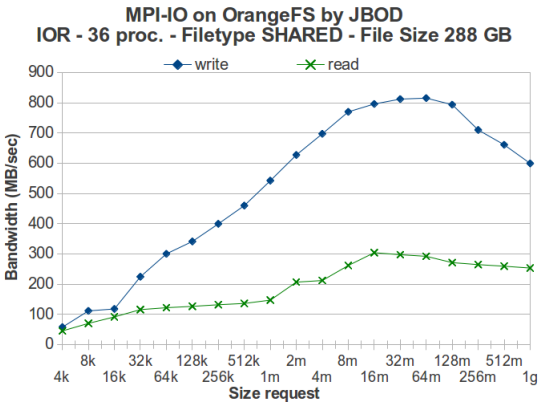
- 2 x Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2), 12 GB Fully Buffered DIMM 667 MHz

## 4.1 I/O subsystem Characterization

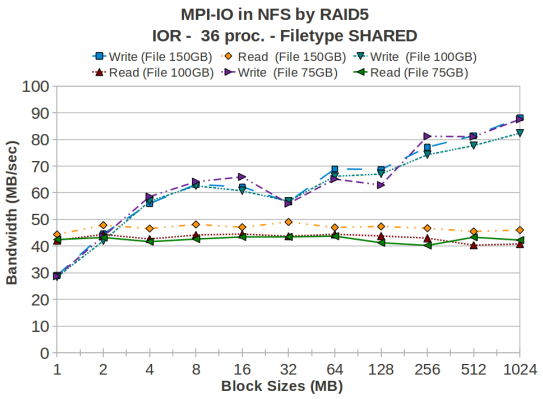
Figure 2 shows performance measured with IOR at I/O library level for the I/O system of A and B.

The I/O subsystem A (Figure 2(a)) shows a increasing I/O performance with the increment of request sizes for the writing operations and reading operations. For the request sizes upper to 16 MB we can observe a performance drop for reading operations. For the writing operations we can observe a drop in performance for the request sizes upper to 64MB. The I/O subsystem B (Figure 2(b)) shows a regular behavior for the reading operations regardless of the file size and request sizes. However, for the writing operations the request size is the I/O factor with the highest impact in performance. We can observe greater transfer rates for bigger request sizes ($> 256MB$), we also can observe a drop in performance for request size of 32 MB.

Peak values for the I/O subsystems are: Write=1120 MB/sec, Read=1260MB/sec for the system A and Write=165 MB/sec, Read=180MB/sec for the system B. These values allows us to limit the performance waited, because usually these are not possible to achieve due to the overhead of I/O software stack. However, the peak value allows us to know: How much performance capacity can provide I/O hardware?,




(a) I/O library on OrangeFS of the System A




(b) I/O library on NFS of the System B

Fig. 2: Performance characterized at I/O library on global filesystem

and How much I/O subsystem capacity are applications really using?

We can observe that the I/O subsystem performance of the system A is higher to I/O subsystem performance of the system B. Also, when we have evaluated the I/O performance for the I/O subsystem B we have observed that this I/O system is not adequate to I/O intensive applications with small request size. Also, we have observed in I/O characterization of the system A that is not adequate to application with request size upper to 1GB. The I/O subsystem A has been configured to applications that will use parallel HDF5 and parallel NetCDF on MPI-IO through a parallel filesystem. However, the I/O subsystem B is configured to parallel applications that can use MPI-IO without support of a parallel filesystem.

## 4.2 I/O subsystem utilization

To evaluate the system usage we analyze the I/O phases to MADBench2 in the I/O subsystems of system A and B. MADbench2 is a tool for testing the overall integrated performance of the I/O, communication and calculation subsystems of massively parallel architectures under the stress of a real scientific application. MADbench2 is based

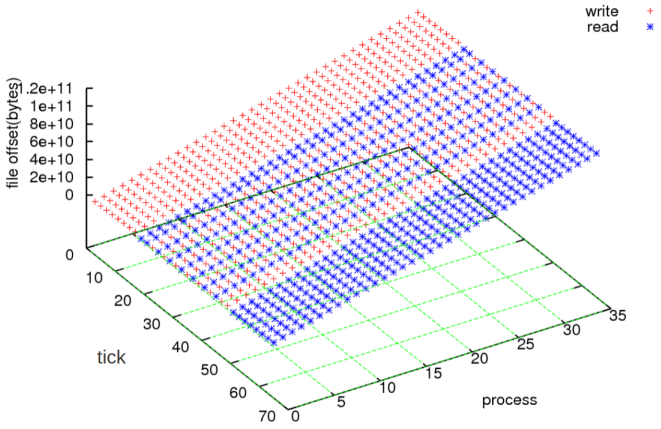



Fig. 3: I/O model of MADBench2 for 36 processes, 40KPIX, and file type SHARED

Table 2: I/O phases description of MADBench2 for $np = 36$ processes with request size $rs = 352$ in MB

| **Phase** | #Oper. | $rep$ | $weight$ |
|---|---|---|---|
| 1 | $(np * rep)$ write | 8 | 102GB |
| 2 | $(np * rep)$ read | 2 | 25GB |
| 3 | $(np * rep)$ write<br>$(np * rep)$ read | 6<br>6 | 75GB<br>75GB |
| 4 | $(np * rep)$ write | 2 | 25GB |
| 5 | $(np * rep)$ read | 8 | 102GB |

on the MADspec code, which calculates the maximum likelihood angular power spectrum of the Cosmic Microwave Background radiation from a noisy pixelated map of the sky and its pixel-pixel noise correlation matrix.

MADbench2 can be run on IO mode, in which all calculations/communications are replaced with busy-work.

MADbench2 reports the mean, minimum and maximum time spent by each function during calculation/communication, busy-work, reading and writing in each function. Running MADbench2 requires a $n^2$ number of processors.

We have obtained the I/O model of MADBench2 for 36 processes. Figure 3 shows I/O model of MADBench2 for 36 processes, 40KPIX, and file type SHARED.

Table 2 shows the five phases identified.

By tracing MADBench2 with our tool we have obtained its metadata: Individual file pointers, Non-collective I/O operations, Blocking I/O operations; sequential access mode, Shared access type; and a file shared by all processes.

The I/O subsystem utilization is analyzed for 36 processes in the I/O subsystems of the systems A and B.

Table 4 shows the utilization of the I/O subsystem B. We also show the amount of data transferred in each I/O phase ($weight$), the number and type of I/O operation (W=write, R=read, W-R=write-read), $BW_{(MD)}$ and $BW_{(PK)}$ in MB/second.

Table 3 shows the utilization of the I/O subsystem A

Table 3: I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System A

| **Phase** | #Oper. | weight | $BW_{(PK)}$ | $BW_{(MD)}$ | System Usage(%) |
|---|---|---|---|---|---|
| 1 | 288 W | 102GB | 1120 | 802 | 72 |
| 2 | 72 R | 25GB | 1260 | 254 | 20 |
| 3 | 432 W+R | 150GB | 1190 | 363 | 31 |
| 4 | 72 W | 25GB | 1120 | 636 | 57 |
| 5 | 288 R | 102GB | 1260 | 296 | 24 |

Table 4: I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System B

| **Phase** | #Oper. | weight | $BW_{(PK)}$ | $BW_{(MD)}$ | System Usage (%) |
|---|---|---|---|---|---|
| 1 | 288 W | 102GB | 204 | 76 | 37 |
| 2 | 72 R | 25GB | 300 | 44 | 15 |
| 3 | 432 W+R | 150GB | 252 | 41 | 16 |
| 4 | 72 W | 25GB | 204 | 60 | 30 |
| 5 | 288 R | 102GB | 300 | 41 | 14 |

for the MADBench2 to 40KPIX and 36 processes. We can observe that the I/O phases 1 and 4 (with writing operations) have utilized greater performance capacity than phases 2, 3 and 5 (phases with reading operations or composed). The third phase has used at about 31% of the performance capacity, a percentage similar to phases with reading operations that have used at about 20%.

Table 4 shows the utilization of the I/O subsystem for the MADBench2 to 40KPIX and 36 processes. We can observe that the I/O phases 1 and 4 (with writing operations) have utilized higher I/O performance capacity than phases 2, 3 and 5 (phases with reading operations or composed). The third phase has used at about 16% of the performance capacity, a percentage similar to phases with reading operations that have used at about 15%.

We can observe that the first phase has more impact in the I/O subsystem because need used more capacity of the I/O subsystem. The other phases with more impact are the fourth and the fifth. The second and the third phase have low impact in the I/O subsystem because the I/O operations are not consecutive, in fact, the I/O operations are done at interval of time sufficient to finish the I/O operations before that data are used.

## 4.3 FLASH-IO Benchmark

The aim of this experimentation is to extract the I/O model and select I/O phases with more weight to evaluate in other I/O subsystem. We have extracted the I/O model in I/O subsystem A and we have applied the I/O model in Finisterrae [15].

Finisterrae is composed of 143 computing nodes:

- 142 HP Integrity rx7640 nodes with 16 Itanium Montvale cores and 128 GB of memory each.
- 1 HP Integrity Superdome node with 128 Itanium Montvale cores and 1,024 GB of memory.

The I/O subsystem of Finisterrae used in this experiment is composed by: mpich2 and HDF5, 1 interconnection network Infinibad 20 Gbps, 1 storage Network Infinibad 20 Gbps, Filesystem Global Lustre (HP SFS), 18 OSS, 2 Metadata Server with 72 cabins SFS20, Filesystem Local Linux ext3, Level Redundancy RAID 6, 866 disks with a capacity of I/O Devices 866*250GB and a mounting point $HOMESFS.

FLASH-IO Benchmark simulates the I/O employed by FLASH for the purposes of benchmarking the code. FLASH is a block-structured adaptive mesh hydrodynamics code. The computational domain is divided into blocks which are distributed across the processors. Typically a block contains 8 zones in each coordinate direction (x,y,z) and a perimeter of guard-cells (presently 4 zones deep) to hold information from the neighbors. The code will produce a checkpoint file (containing all variables in 8-byte precision) and two plot files (4 variables, 4-byte precision, one containing corner data, the other containing cell-centered data). The checkpoint and plot file routines are identical to those used in the FLASH Code.

### 4.3.1 Extracting the I/O model

We have obtained the following meta-data of FLASH-IO in the parallel HDF5 version with our tool:

- Explicit offset, Blocking I/O operations, Collective operations and Non-collective.
- Strided access mode, Shared access type for three files.
- MPI-IO routine MPI_Set_view with etype of with different etype and filetype for collective and non-collective operations.

FLASH-IO performs only I/O operations and there are no communication events. I/O operations of parallel HDF5 are converted into MPI primitives. Table 5 shows the description of the I/O phases. Where IdPh. is the Identifier of the Phase, Oper. is the Type of operation, $RS$ is the Size of request, Iter. is the number of iterations, Dist. is the Distance, OI is the initial offset, Disp. is the Displacement, and 1ºTick is the first tick (1ºTick) of the phase. Distance is the number of events of communication or events of I/O between two phases. The first tick represents the tic's number from the first pattern of one phase. Displacement defines the location where a view begins, this is the file displacement.

We can observe five phases for the first file. In this case the are several MPI_File_set_view to achieve strided access. There are two types of writing operations: MPI_File_write_at and MPI_File_write_at_all. We show the phases for the collective operations because they represent the 90% of I/O. The I/O model of flash is shown in Figure 4.

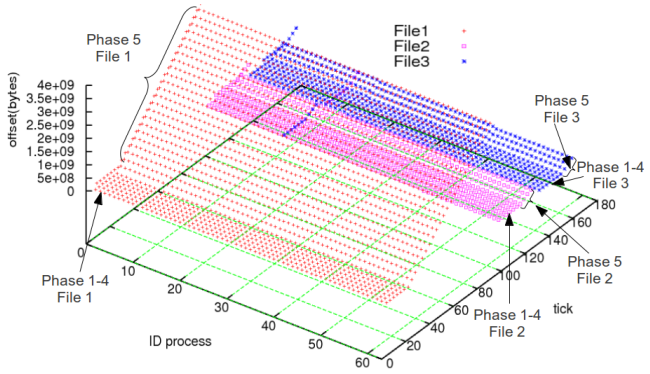



Fig. 4: I/O model of FLASH-IO for 64 processes

Table 5: I/O model of FLASH-IO Write_at_all

| IdPh. | RS | Iter. | Dist. | OI | Disp. | 1ºTick |
|---|---|---|---|---|---|---|
| 1º F. | | | | | | |
| 1 | 320 | 2 | 3 | 6288 | 20732 | 9 |
| 2 | 4800 | 1 | 3 | 47752 | 310980 | 15 |
| 3 | 1920 | 2 | 3 | 358732 | 124392 | 18 |
| 4 | 3840 | 1 | 3 | 609564 | 439012 | 24 |
| 5 | 2621440 | 24 | 3 | 1048576 | 169869312 | 27 |
| 2º F. | | | | | | |
| 1 | 320 | 2 | 3 | 6288 | 20732 | 109 |
| 2 | 4800 | 1 | 3 | 47752 | 310980 | 115 |
| 3 | 960 | 2 | 3 | 358732 | 62196 | 118 |
| 4 | 1920 | 1 | 3 | 485172 | 301260 | 124 |
| 5 | 1310720 | 4 | 3 | 786432 | 84934656 | 127 |
| 3º F. | | | | | | |
| 1 | 320 | 2 | 3 | 6288 | 20732 | 149 |
| 2 | 4800 | 1 | 3 | 47752 | 310980 | 155 |
| 3 | 960 | 2 | 3 | 358732 | 62196 | 158 |
| 4 | 1920 | 1 | 3 | 485172 | 301260 | 164 |
| 5 | 1572160 | 4 | 3 | 786432 | 101974016 | 167 |

The I/O model shows three file used during the run of the application. Files are enumerated taking into account the order in which they were opened. Table 5 shows the description of phases for the first file (1º F.), second file (2º F.), and the third file (3º F.). We can observe small request size for the phases 1 to 4 for the three files and 2MB for first file and 1MB for the second and third file in the fifth phase.

### 4.3.2 Applying the I/O model

We use the I/O model of FLASH-IO to set the parameters of IOR. We select phase 5 to mimic with IOR because it is the most weighted phase. The parameters are set as follows and we run IOR in the same order:

- File 1: -np 64 -a MPIIO -c -s 24 -b 2621440 -t 2621440
- File 2: -np 64 -a MPIIO -c -s 4 -b 1310720 -t 1310720
- File 3: -np 64 -a MPIIO -c -s 4 -b 1572160 -t 1572160

We have evaluated the I/O time of IOR and I/O time of flash in the cluster Finisterrae. Table 6 shows the I/O time obtained with IOR $Time_{io(CH)}$, I/O time for FLASH-IO $Time_{io(MD)}$, and relative error $error_{rel}$. We observe higher

Table 6: Error of I/O time estimation on Finisterrae for 64 and 128 processes for phase 5 of FLASH-IO

| **Phase** | $Time_{io(CH)}$ | $Time_{io(MD)}$ | $error_{rel}$ |
|---|---|---|---|
| **64p** | | | |
| File 1 | 47.73 | 51.02 | 6% |
| File 2 | 3.07 | 4.62 | 33% |
| File 3 | 3.80 | 4.83 | 21% |
| **128p** | | | |
| File 1 | 98.88 | 102.25 | 3% |
| File 2 | 8.74 | 8.44 | 3% |
| File 3 | 10.02 | 11.14 | 10% |

errors for 64 processes in the File 2 and File 3, this is due to the size of Files ( 400MB). However, we can observe that the error is decreased in 128 processes for the File 1. This is the file with the highest size.

Flash increases its I/O requirements when the number of processes is increased. For example, the weight of the fifth phase of the File 1 is $2621440*24*512$ to 512 processes and $2621440 * 24 * 256$ to 256 processes. We have analyzed the I/O model for 256 and 512 processes and we have observed that the I/O phases have the same request size, therefore, the I/O requirements are increased in function of the number of processes for each I/O phases. For this application the I/O system have more impact when the number of processes is increased.

# 5. Conclusions

We have applied a methodology to characterize the parallel I/O of scientific applications. We have represented the message-passing applications through an I/O model. The model allows us to evaluate the I/O subsystems taking into account the I/O phases of the application. We have used an exhaustive performance characterization of I/O subsystem for different access patterns with the benchmark IOR at I/O library level and IOzone at I/O devices level.

Furthermore, we have used the I/O model to set the IOR benchmark input parameters for the access patterns of each I/O phase of Flash-IO. This approach is adequate when the I/O subsystem does not allows us an exhaustive I/O performance characterization. We have evaluated this approach to FLASH-IO. We have used the I/O model to estimate the I/O time. Relative errors have shown that the I/O time estimation is more accurate when the I/O is representative of the application.

The I/O model can be used to evaluate the performance of the application without executing it. This is very useful particularly for the real applications that usually need several libraries in order to be executed.

Currently, we are extending the I/O phases identification to different applications in order to show others I/O behaviors. Also, we are analyzing the relationship between the I/O model and number of I/O and stripe size. We plan to provide an useful configuration method to users and administrators. We expect that by using our tool they will be able to configure the number of I/O node and the stripe size, by considering only the most relevant phases of one application.

# Acknowledgment

This research has been supported by the MICINN Spain under contract TIN2007-64974, the MINECO (MICINN) Spain under contract TIN2011-24384, the European ITEA2 project H4H, No 09011 and the Avanza Competitividad I+D+I program under contract TSI-020400-2010-120.

Appreciation to The Centre of Supercomputing of Galicia (CESGA), Science and Technology Infrastructures (in spanish ICTS).

# References

[1] A. Wong, D. Rexachs, and E. Luque, "Extraction of parallel application signatures for performance prediction," in *HPCC, 2010 12th IEEE Int. Conf. on*, sept. 2010, pp. 223–230.

[2] S. Mendez, D. Rexachs, and E. Luque, "Methodology for performance evaluation of the input/output system on computer clusters," in *Workshop IASDS on Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, sept. 2011, pp. 474 –483.

[3] J. Carter, J. Borrill, and L. Oliker, "Performance characteristics of a cosmology package on leading hpc architectures," in *High Performance Computing - HiPC 2004*, ser. Lecture Notes in Computer Science, L. Bougé and V. Prasanna, Eds., vol. 3296. Springer Berlin / Heidelberg, 2005, pp. 21–34.

[4] A. Laboratory. (2013) Flash io benchmark. [Online]. Available: http://www.mcs.anl.gov/research/projects/pio-benchmark/

[5] J. M. Kunkel and T. Ludwig, "Performance evaluation of the pvfs2 architecture," in *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, feb. 2007, pp. 509 –516.

[6] J. H. Laros *et al.*, "Red storm io performance analysis," in *CLUSTER '07: Procs of the 2007 IEEE Int. Conf. on Cluster Computing*. USA: IEEE Computer Society, 2007, pp. 50–57.

[7] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O performance challenges at leadership scale," in *Proceedings of SC2009: High Performance Networking and Computing*, November 2009.

[8] P. Carns, K. Harms, W. Allcock, C. Bacon, R. Latham, S. Lang, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," in *27th IEEE Conference on Mass Storage Systems and Technologies (MSST 2011)*, 2011.

[9] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, "Parallel i/o prefetching using mpi file caching and i/o signatures," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, nov. 2008, pp. 1–12.

[10] H. Shan and J. Shalf, "Using IOR to analyze the I/O performance of HPC platforms," in *Cray Users Group Meeting (CUG) 2007*, Seattle, Washington, May 7-10, 2007.

[11] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 Characterization of Petascale I/O Workloads," in *Proceedings of 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, September 2009.

[12] S. Mendez, D. Rexachs, and E. Luque, "Modeling parallel scientific applications through their input/output phases," in *CLUSTER Workshops'12*, 2012, pp. 7–15.

[13] H. . S. J. Shan, "Using ior to analyze the i/o performance for hpc platforms," LBNL Paper LBNL-62647, Tech. Rep., 2007. [Online]. Available: www.osti.gov/bridge/servlets/purl/923356-15FxGK/

[14] W. D. Norcott, "Iozone filesystem benchmark," Tech. Rep., 2006. [Online]. Available: http://www.iozone.org/

[15] C. Finisterrae, "Centre of supercomputing of galicia (cesga)," Science and Technology Infrastructures (in spanish ICTS), Tech. Rep., 2012. [Online]. Available: https://www.cesga.es/