

A new approach for Analyzing I/O in parallel scientific applications^{*}

Sandra Méndez, Javier Panadero, Alvaro Wong, Dolores Rexachs and Emilio Luque

Computer Architecture and Operating System Department (CAOS)
Universitat Autònoma de Barcelona, Barcelona, Spain

sandra.mendez@uab.es, {javier.panadero, alvaro.wong}@caos.uab.es, {dolores.rexachs, emilio.luque}@uab.es

Abstract. The efficient use of high performance computing is usually focused on the use of computational resources. However, scientific applications currently produce a large volume of information. Therefore, the Input/Output (I/O) subsystem also should be used efficiently. In order to do so, it is necessary to know the application I/O patterns and establish a relationship between these patterns and the I/O subsystem configuration. To analyze the I/O behavior of applications, we propose use a library of the PAS2P (Application Signature for Performance Prediction) tool. Parallel applications typically have repetitive behavior, and the I/O patterns of parallel applications also have that behavior. We propose to identify the portions (I/O phases) where the application does I/O. From these I/O phases, we extract an application model that can be used to evaluate it in different I/O subsystems considering the I/O phases and compute-communication phases. In this paper, we present the concepts used in the PAS2P methodology, which have been adapted for MPI-IO applications. We have extracted the I/O model of applications. This approach was used to estimate the I/O time of an application in different subsystems. The results show a relative error of estimation lower than 10%.

1 Introduction

Due to the historical “gap” between the computing and Input/Output (I/O) performance, in many cases, the I/O system becomes the bottleneck of parallel systems. In order to hide this “gap”, the I/O factors with the biggest influence on performance must be identified. Furthermore, the increased computational power of processing units and the complexity of scientific applications, which use high performance computing, require more efficient Input/Output Systems.

^{*} 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).

The configuration of the I/O subsystem affects the application performance. It is important to understand the I/O subsystem structure: filesystem type, I/O devices, interconnection networks and I/O libraries. The I/O subsystem in computer clusters can have several I/O configurations and the user should select the configuration depending on the I/O requirements of his application. However, usually the user does not know the I/O subsystem and the I/O requirement of application.

We propose a methodology to extract the I/O requirements of the application expressed by a I/O abstract model that can be used in different I/O subsystem. Also, we propose a method to select the I/O configuration from a set existing configurations, depending on the I/O abstract model of application.

Since the I/O operations are affected by time between operations, we need to analyze the parallel application, taking into account the computing and communication. For this reason, we selected the PAS2P library to trace the I/O operations of standard MPI-2. In previous work [1] we have presented a methodology for performance evaluation of the I/O system which is focused on I/O path. In this paper, we explain the I/O analysis for the application's access pattern extraction, the I/O phases identification and the I/O abstract model of application.

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 conclusions and future work.

2 Related Work

There are other tools which are closely related to the I/O analysis of message-passing applications. Darshan [2] is a parallel I/O characterization tool designed to characterize the MPI-IO file access of HPC applications in a non-intrusive way. It characterizes the application by using statistics and cumulative timing information. The information obtained can be used to analyze the I/O behavior of a MPI-program. It is implemented as a set of user space libraries. These libraries require no source code modification and can be added in a transparent way. This approach differs from PAS2P-I/O because Darshan provides statistical averages of I/O instead of information by I/O operation.

LANL-Trace [3] is a tracing framework that wraps the standard Unix library and system call tracing utility `ltrace`. LANL-Trace generates three types of outputs which are useful for the I/O analysis. One advantage of LANL-Trace is that it is simple to understand and use. Because of its simple nature, it is also easy to modify. However, LANL-Trace's simplicity is a trade-off because causes higher overhead. This work differs of our proposal because PAS2P-I/O intercepts the MPI functions imposing minimal overhead.

There are other general-purpose instrumentation tools, that profile and trace general MPI and CPU activity. These tools allows the analysis of the I/O parallel applications. The most common tools used by the scientific community are: Jumpshot [4], TAU [5] and STAT [6]. The main difference with our approach

lies in the fact that these tools are focused primarily on the general analysis of the application, without providing specific details and information about the MPI-I/O operations.

3 Proposed Methodology

We propose a methodology to analyze the I/O requirements of parallel scientific applications, which is composed by three steps: Modeling Input/Output of Application, using the Application Input/Output Model, and Validation of Application I/O Model Applicability.

3.1 Modeling Input/Output of Application

The I/O model of applications is defined by three characteristics: metadata, spatial global pattern and temporal global pattern. To obtain the application I/O model, we have implemented a library extension named `libpas2p-io.so`. This dynamic library will be used to instrument the application.

PAS2P tool implements the PAS2P methodology [7], which is based on the high repetitive behavior of the applications. The PAS2P methodology is composed by two-step. The first step is to analyze the application to build an application model and extract its phases and weights. Finally, it uses that information to build an executable signature. The second step is to execute the application signature in a target system in order to predict the total execution time of the application.

To extract the three characteristics for the I/O model of parallel application, we have focused on the first step of PAS2P methodology: data collection, pattern identification, parallel application model, and extraction of phases and weights.

Data collection The aim of this step is to generate a trace log with the behavior of computation and communication of a parallel application. In order to intercept and collect communication events (action of sending or receiving a message) interposition functions are used. To collect the computational time, PAS2P has extended the concept of Basic Block (BB)[8] for parallel applications. We have defined a new concept named Extended Basic Block (EBB) as a segment of a process, whose beginning and end are defined by occurrences of MPI events, either sent or received. Also it may say that it is a "computational time" segment bounded by communication events, as shown in Figure 1(a) that also illustrates the event information recollected, which will be used to generate the trace log.

An event contains information of each application process, the MPI event, the source-destination the process is involved with, the count events, the communication volume, the wall clock time, the computational time of the event occurred as well as its type (send: 1, recv: 0, collective events: -1). This log contains the whole application trace. It can be used in order to analyze the application behavior and the trade-off between compute-communication.

In order to consider the I/O data of the application, we use the concept of the I/O event, which is a segment of process where the I/O operation is called. The

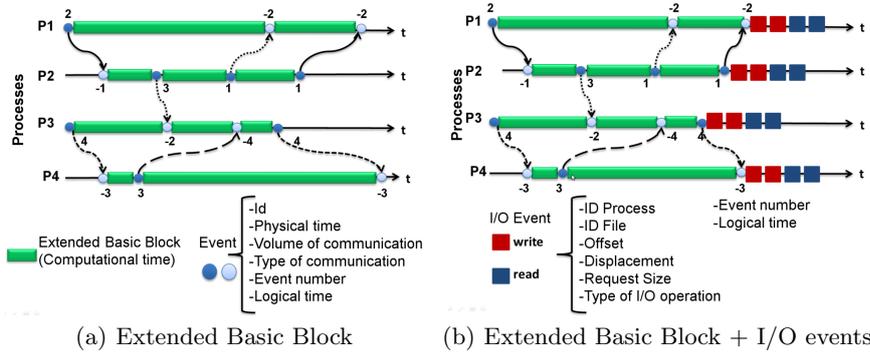


Fig. 1. Data Collection

I/O event is composed by ID process, ID file, offset, displacement, request size, type of I/O operation, event number, and logical time. We have incorporated the I/O events to PAS2P library. This allows to identify the relationship between the events of communication, computation and I/O. Figure 1(b) shows an example of physical traces with EBB and the I/O events.

Parallel application model Synchronization between processes is necessary in parallel applications. PAS2P has developed a Logical ordering algorithm [7] inspired by Lamport's [9]. Through this algorithm, PAS2P defines a new logical ordering, in which, if one process sends a message in a Logical Time (LT), its receive will be modeled to arrive at $LT + 1$.

Once all events have been assigned a LT, the logical trace is created from a physical trace, where Logical Times will be given by LT for the Send events (LTSend) and LT for the reception events (LTRRecv). Finally, once each event has been located in the respective Logical Time, the logical trace is divided into more logical times, that is, there can only be one event for each process in a Logical Time. Once we know this, we are able to introduce two new concepts. The first new concept is a *tick*, tick is defined as a logical unit time, and it is incremented by each communication event. Another new concept is *Parallel Basic Block (PBB)*, which is defined as a set of Extended Basic Blocks delimited by two ticks. The first tick defined as entry point has at least one event, and the second tick defined as exit point also has at least one event.

Synchronization of the I/O operations depend on the data consistency managed by the filesystem. Therefore, for the I/O operations the logical trace is the same as the physical trace and a *tick* is equal to a logical time unit. A PBB in I/O is equal to an I/O event where the entry/exit point is the call to I/O operation. In order to obtain the I/O abstract model of the application, we have to analyze the application I/O. Therefore, the response time of I/O operations is not needed, because it depends on performance of the target I/O subsystem.

Once this step has been done, the I/O model of application is obtained. This model is made up by three events: computing, communication and I/O events.

Pattern identification In order to find a representative behavior of a message-passing application, once we have obtained the logical trace, as shown in figure 2(a), we search the application phases.

With the objective of finding these phases, PAS2P has an algorithm [7], which is based on three criteria (type of communication, volume of communication and computational time by process). Finally, phases are created as sub-chains of grouped PBB's that repeat along the execution, as shown in figure 2(b).

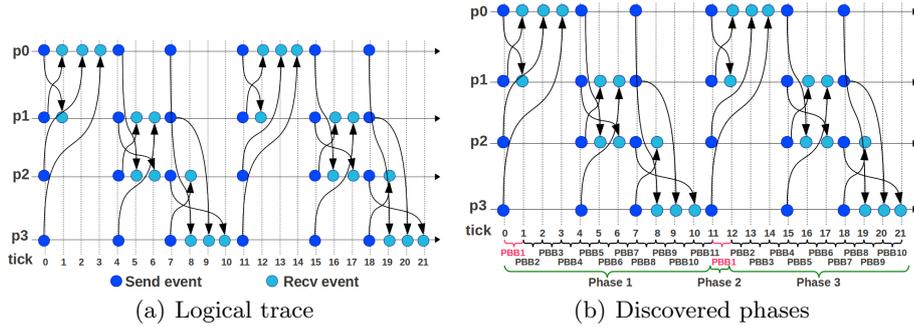


Fig. 2. Extracting phases from the Logical trace

The I/O phases depend on the local and global pattern, as well as the spatial and temporal occurrence of these patterns. To identify the I/O phases, we analyze the similarity of I/O events. I/O events are named Access Pattern (AP). An I/O phase is composed by AP with similar order and without events communication between the I/O events. From the logical trace (Figure 3(a)) the I/O phases of the application are obtained (Figure 3(b)). We show the access patterns denoted by AP and I/O phases identified in the logical trace.

Extract relevant phases and weights Once we have identified the phases of a parallel application, we define the weights and the relevant phases. The weight will be given by the frequency in which each phase repeats. A relevant phase is when the weight multiplied by the phase runtime is representative of the total application runtime. We have considered that this "representativeness" will be given if the phase is 1% or more of the total execution time of the whole application. Each phase contains also information about its patterns, such as the communication pattern, the volume of communication and the computational time by process.

PAS2P definitions of phase and weight predict the performance of an application in target systems with a minimum error. However, for the I/O analysis, we need to define concepts of the phase and the weight to create the I/O abstract model in which all I/O phases are considered. The weight depends on the number of processes, request size and repetitions of each AP that is part of a phase. The weight is expressed in Megabytes and it is necessary to determine transferred data in each I/O phase. The I/O abstract model depends on I/O

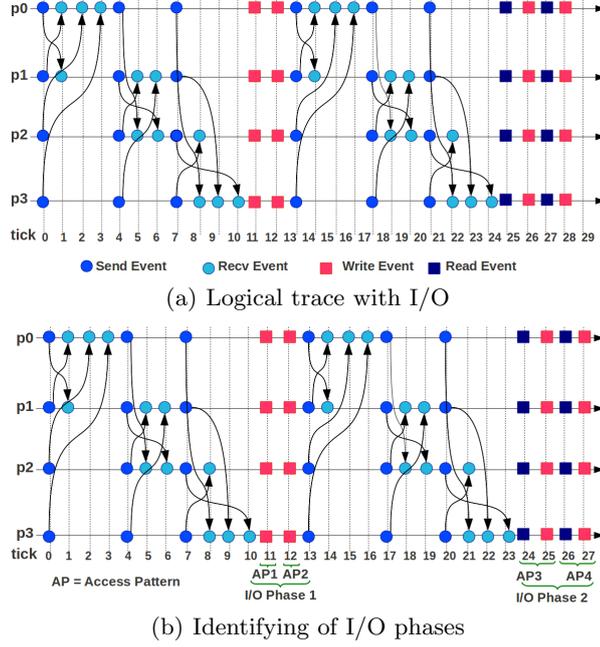


Fig. 3. Identifying the I/O phases from the Logical trace

phases and weight, allowing us to know “when“ and “how“ the I/O subsystem will be used. The I/O abstract model of application can be used to mimic the application I/O and determine the I/O subsystem where the application will be less penalized by its I/O behavior.

3.2 Using the Application Input/Output Model

In previous work [1], we have presented the I/O system performance evaluation through exhaustive characterization with the benchmark IOR [10] for I/O library (MPI-IO) and benchmark IOzone [11] for I/O devices. In the present paper, we use the I/O abstract model to estimate the application time in different subsystems in order to select the configuration with less time. In this way, we achieve a characterization of I/O subsystem for the different types of access patterns of the application. Furthermore, we can obtain a performance characterization of a specific application in less time. The characterization time represents the 10% to 20% of running time of full application.

We use the I/O abstract model to set up the input parameters of the benchmark IOR. We only execute the benchmark for the phases of the I/O model. The following setting of input parameters are applied on IOR for each I/O phase: s depends on access mode; $b = weight_{(ph)}$; $t = rs_{(ph)}$; $NP = np_{(ph)}$; $-F$ if there is 1 file per process; $-c$ if there is collective I/O.

The selected metric for IOR is the transfer rate (MB/sec), named $BW_{(CH)}$. The estimated I/O time for each I/O phase is calculated by expression (1).

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

Where the $Time_{io}(phase[i])$ is calculated by expression (2).

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

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

The I/O model is used to determine what system can provide the best performance for the application at I/O library level.

3.3 Validation of Application I/O Model Applicability

It is necessary to evaluate the estimation's accuracy for the selected configuration. In order to do that, we evaluate the relative error produced by the I/O time estimation. Relative error is calculated by expression (3);

$$error_{rel} = 100 * \left(\frac{error_{abs}}{BW_{MD}(phase[i])} \right) \quad (3)$$

Where absolute error is calculated by the expression (4).

$$error_{abs} = |BW_{(CH)}(phase[i]) - BW_{MD}(phase[i])| \quad (4)$$

Where $BW_{CH}(phase[i])$ is *transferRate* characterized at I/O library level. When a phase has two or more I/O operations, the $BW_{(CH)}$ is defined by the average of the $BW_{(CH)}$ of each I/O operation that composes the I/O phase.

4 Experimental validation

We have applied the proposed methodology to select I/O configuration of two I/O subsystem. Table 1 shows both the configuration A and the configuration of cluster Finisterrae [12].

The I/O phases identification is applied to Block Tridiagonal(BT) application of NAS Parallel Benchmark suite (NPB)[13]. The BTIO benchmark performs large collective MPI-IO writes and reads of a nested strided datatype, and it is an important test of the performance that a system can provide for non-contiguous workloads. After every five time steps the entire solution field, consisting of five double-precision words per mesh point, must be written to one or more files. After all time steps are finished, all data belonging to a single time step must be stored in the same file, and must be sorted by vector component, x-coordinate, y-coordinate, and z-coordinate, respectively.

Table 1. Description of configuration A and Finisterrae

I/O Element	Configuration A	Finisterrae
I/O library	OpenMPI	mpich2, HDF5
Communication Network	1 Gbps Ethernet	1 Infinibad 20 Gbps
Storage Network	1 Gbps Ethernet	1 Infinibad 20 Gbps
Filesystem Global	NFS Ver 3	Lustre (HP SFS)
I/O nodes	8 DAS and 1 NAS	18 OSS
Metadata Server	1	2 with 72 cabins SFS20
Filesystem Local	Linux ext4	Linux ext3
Level Redundancy	RAID 5	RAID 5
Number of I/O Devices	5 disks	866 disks
Capacity of I/O Devices	1.8 TB hot-swap SAS	866*250GB
Mounting Point	/home	\$HOMESFS

Since NAS BTIO has an access mode strided and the IOR is not working in this mode, we have selected the sequential access mode to replicate the I/O phases. We have obtained the following meta-data of NAS BT-IO in the FULL subtype with our tool:

- Explicit offset, Blocking I/O operations, Collective operations.
- Strided access mode, Shared access type.
- MPI-IO routine MPI_Set_view with etype of 40.

We have obtained the I/O model of NAS BT-IO for 36, 64 and 121 processes for the Class D and this model has been applied in the configuration A and Finisterrae. Figure 4 shows the I/O abstract model for 36 processes.

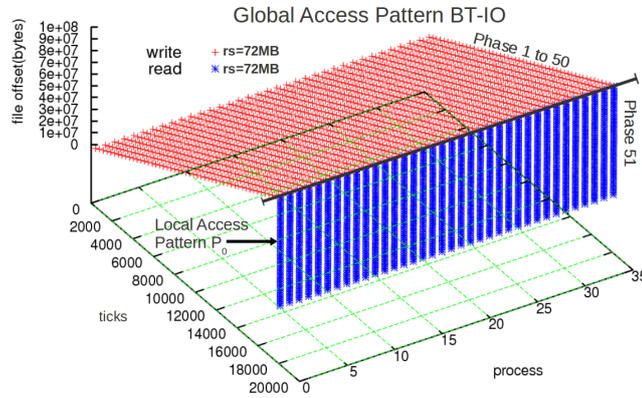


Fig. 4. I/O model to NAS BT-IO, class D, 36 processes, and subtype FULL on Configuration A and Finisterrae

Table 2 shows description of the I/O phases for NAS BT-IO where ph is the number of phase, rs is request size, np the number of processes of ph , idP is the rank of MPI process, $iter$ is the number of iteration into ph and rep is the number of repetitions of ph . Initial offset depends on rs and the number of phases. Also

the request size depends on number of processes of the application. NAS BT-IO have the following values for the class D: $rs=72\text{MB}$ for 36p, $rs=40\text{MB}$ for 64p and $rs=24\text{MB}$ for 121p.

Table 2. I/O phases description of NAS BT-IO subtype FULL, class D for np processes

Phase	#Operation	Initial Offset	rep	$weight$
1-50	np W in each phase	$rs * idP + (rs * (ph - 1) + (rs * (np - 1)) * (ph - 1))$	1 x phase	$np * rep * rs$
51	np R	$rs * idP + (rs * (iter - 1) + (rs * (np - 1)) * (iter - 1))$	50	$np * rep * rs$

Table 3. I/O time estimation ($Time_{io(CH)}$) on configuration A and Finisterrae for 64 processes

Phase	$Time_{io(CH)}$ on conf. A	$Time_{io(CH)}$ on Finisterrae
Phase 1-50	1167.40	932.36
Phase 51	2868.51	844.42

We have calculated the I/O time using our proposed methodology. Table 3 shows the $Time_{io(CH)}$, where we can observe that the configuration with less I/O time for NAS BT-IO with 64 processes is Finisterrae. Table 4 shows the $error_{rel}$ in the estimation for 64 processes on configuration A and Finisterrae.

We evaluated these errors by executing several times NAS BT-IO and the error was similar in all the different tests. Furthermore, the I/O model has been obtained at a different time to discard the influence of the tracing tool. The same I/O model can be applied to estimate the I/O time in other systems, where $Time_{io(CH)}$ will be obtained by the expression (2), the BW_{CH} will be obtained by executing IOR with the input parameters explained in the section 3.2. As we can see, the estimation improves when increasing the number of processes. The error rate on both configurations is less than 10% and is reduced when workload and number of processes is increased.

Table 4. Error of I/O time estimation on configuration A and Finisterrae for 64 processes

Phase	$Time_{io(CH)}$	$Time_{io(MD)}$	$error_{rel}$
Configuration A			
Phase 1-50	1167.40	1153.05	1%
Phase 51	2868.51	2984.75	4%
Configuration Finisterrae			
Phase 1-50	932.36	924.85	1%
Phase 51	844.42	909.43	7%

5 Conclusions

A methodology to obtain an I/O abstract model of a parallel application has been proposed and tested. The application I/O model is defined by three characteristics: metadata, spatial and temporal global pattern. We instrument the application to obtain the access pattern and we analyze it to identify the I/O phases. This instrumentation is done at MPI-IO level which does not require the source code. We have obtained the I/O model of NAS BT-IO and we have evaluated two configurations taking into account the I/O phase behavior. We have used the I/O model to estimate the I/O time. Relative errors are acceptable and we have observed that the error rate decreases when the number of processes is increased. The error was about the 10%.

As future work, we are designing a benchmark to replicate the I/O when there are two or more operations in a phase to improve the characterization and reduce estimation error. We are extending the I/O phases identification to different applications that show different I/O behaviors. We are analyzing *upwelling* of ROMs framework, which is an application that opens different files in execution time. Our model is applicable for each file, but still, it is necessary to refine the methodology for the I/O phases with complex access patterns.

References

1. Mendez, S., Rexachs, D., Luque, E.: Methodology for performance evaluation of the input/output system on computer clusters. In: Workshop IASDS on Cluster Computing, 2011 IEEE International Conference on. (sept. 2011) 474–483
2. Carns, P.H., Latham, R., Ross, R.B., Iskra, K., Lang, S., Riley, K.: 24/7 characterization of petascale i/o workloads. In: CLUSTER, IEEE (2009) 1–10
3. Konwinski, A., Bent, J., Nunez, J., Quist, M.: Towards an i/o tracing framework taxonomy. In: Proceedings of the 2nd Int. Workshop on Petascale data storage: in SC'07. PDSW '07, New York, USA, ACM (2007) 56–62
4. Zaki, O., Lusk, E., Swider, D.: Toward scalable performance visualization with jumpshot. High Performance Computing Applications **13** (1999) 277–288
5. Shende, S.S., Malony, A.D.: The tau parallel performance system. Int. J. High Perform. Comput. Appl. **20**(2) (May 2006) 287–311
6. Arnold, D., Ahn, D., de Supinski, B., Lee, G., Miller, B., Schulz, M.: Stack trace analysis for large scale debugging. In: IPDPS 2007. IEEE International. (2007)
7. Wong, A., Rexachs, D., Luque, E.: Extraction of parallel application signatures for performance prediction. HPCC, 10th IEEE Int. Conference (2010) 223–230
8. Lau, J., Schoemackers, S., Calder, B.: Structures for phase classification is - sn -. Performance Analysis of Systems and Software, ISPASS (2004) 57–67
9. Lamport, L., Time, C.: The Ordering of Events in a Distributed System. Communications of the ACM **21**(7) (1978) 558–565
10. William Loewe, T.M., Morrone, C.: Ior benchmark (2012)
11. Norcott, W.D.: Iozone filesystem benchmark (2006)
12. Finisterrae, C.: Centre of supercomputing of galicia (cesga). Technical report, Science and Technology Infrastructures (in spanish ICTS) (2012)
13. Wong, P., Wijngaart, R.F.V.D.: Nas parallel benchmarks i/o 2.4. Technical report, Computer Sciences Corp., NASA Advanced Supercomputing Division (2003)