Universitat Autònoma de Barcelona

Analyzing the Parallel Applications' I/O Behavior Impact on HPC Systems

Pilar Gómez Sánchez

Universitat Autònoma de Barcelona

Escola d'Enginyeria

Departament d'Arquitectura de Computadors i Sistemes Operatius

Analyzing the Parallel Applications' I/O Behavior Impact on HPC Systems

Thesis submitted by **Pilar Gómez Sánchez** for the degree of philosophae Doctor by the Universitat Autònoma de Barcelona, under the supervision of **Dr. Dolores Rexachs del Rosario**, developed at the Computer Architectures and Operating Systems department, PhD in Computer Science.

Barcelona, March 2018

Analyzing the Parallel Applications' I/O Behavior Impact on HPC Systems.

Thesis submitted by **Pilar Gómez Sánchez** for the degree of Philosophiae Doctor by the Universitat Autònoma de Barcelona, under the supervision of Dr. Dolores Rexachs del Rosario, at the Computer Architecture and Operating Systems Department, Ph.D in Computer Science.

Supervisors Dr. Dolores Rexachs del Rosario

Barcelona, March 2018

Dedicado

A una mujer luchadora...
a mi ángel,...mi yaya.

A mi madre...
porque nunca permites que me rinda.

A mis padres y mis hermanas...
por su infinito apoyo, por sus valores y por su amor.
Porque siempre me decís... adelante, tú puedes.
¡Os amo con locura!

A las chicas de E/S...
sin vosotras no lo hubiera conseguido.
Un lujazo y honor trabajar a vuestro lado,
al lado de dos grandes investigadoras.

A los que apoyan y apuestan por la investigación...
de cualquier ámbito.
¡Bendita y necesaria investigación!

"Para abrir nuevos caminos, hay que inventar, experimentar, crecer, correr riesgos, romper las reglas, equivocarse... y divertirse."
- Mary Lou Cook

"Uno nunca ve lo que se ha hecho, sino que ve lo que queda por hacer"
- Marie Curie

"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad"
- Albert Einstein

Agradecimientos

En primer lugar, mi agradecimiento es para tí, Lola... mi profesora, mi directora... pero sobretodo, mi AMIGA. Gracias por tus consejos, por permitir que pertenezca al club de las chicas de E/S, porque no hay dia que no aprenda algo contigo, por aguntarme, por preocuparte por mi, por tu afán de que me divierta investigando, por tus conocimientos e ideas...pero sobretodo por nuestras charlas, tu amistad y tu humanidad.

A las dos personas artífices de que yo decidiera hacer la tesis. Fue la mejor decisión que pude tomar. Y eso os lo debo a vosotros: Emilio y Lola. Gracias por apostar y confiar en mi. ¡Muchas gracias! Vuestra pasión por la investigación ha hecho que estos años hayan sido apasionantes. Gracias por esa forma de entender la investigación, por vuestra dedicación. Por todo lo que he compartido estos años con vosotros y los que espero seguir compartiendo.

A Sandra, esa argentina luchadora que me ha tendido la mano, transmitido su conocimiento y acompañado estos años de investigación. ¡No cambies! Eternamente agradecida...por tus desvelos. Gracias por la estancia en Munich, en el LRZ.

A Remo, gracias por tu ayuda y consejos.

A Marce, Jose (Josefina) y Alejandro Chacón por ser más que compañeros de investigación, por nuestras conversaciones y risas. Reir,...para mi la mejor medicina. Gracias chicos por esos buenos momentos. Marce, me hiciste muy fácil y ameno el tiempo que compartimos escritorio, por nuestro intento de hacer teatro... Jose, gracias por aguantar mis “cosas", por facilitarme que consiguiera alojamiento en Munich...tarea imposible sin tu ayuda; en fin, por tenderme siempre la mano. Che, afortunada de tenerte como amiga.

A Toni, Sonsoles, Maria del Mar...gracias por vuestro apoyo incondicional desde hace muchos años, vuestra forma de ver la vida, vuestros consejos, vuestra sinceridad. Siempre sumáis.

A Susana Angueira porque tu fortaleza y apoyo me ayudaron en su día. Gracias por tu amistad.

A las Yolis, Anna-FIAC, a la mesa 9, Andrea y Hayden, Genaro, Claudia, Christianne, Aprigio, Eli y Sergio, Carmen y Juan, Bet y Christian, Hai de Munich...gracias por vuestros ánimos, por cuidarme, por crear momentos para que desconectara, por las conversaciones y las risas.

A las personas que me han ayudado con el inglés, en especial a Sandra, Jose, Andrea y Hayden, y a Nick. Gracias por vuestra ayuda, vuestro tiempo y vuestras correcciones.

A mis compañeros de las reuniones de investigación...Alvaro, Carlos, Laura, Jorge, Joe y Javier Panadero, por discutir, por preguntar, por cuestionar las ideas o, simplemente por escuchar mis explicaciones. ¡Gracias!

Como me olvidaré de alguien y no quiero...quiero agradecer al departamento CAOS (tanto a los becarios y compañeros que he conocido allí como al staff) los momentos vividos hasta el momento. Gracias por dejaros enredar para los cumples, barbacaos...gracias por vuestra participación en todo. Para mí, es como mi segunda casa porque llevo ya algunos añitos.

A Gemma Roque y Daniel Ruiz, dos personas que hacen fácil todo lo administrativo y técnico. Gracias por vuestra ayuda y por vuestra labor. Gemma, gracias por nuestras charlas.

Y por supuesto, mi más sincero agradecimiento a la “Fundacion Escuelas Universitarias Gimbernat"por financiar mis años de investigación. ¡Gracias!

Abstract

The volume of data generated by scientific applications grows and the pressure on the I/O system of HPC systems also increases. For this reason, an I/O behavior model is proposed for scientific MPI (*Message Passing Interface*) parallel applications. The goal is to analyze the applications' impact on the I/O system. Analyzing the MPI parallel applications at POSIX-IO level allows observing how the application's data are treated at that level. In this research work, the following is presented: the I/O behavior model definition at POSIX-IO level (PIOM-PX model definition), the methodology applied to extract this model and the PIOM-PX-Trace-Tool. As PIOM-PX is based on the I/O phase concept, it can identify the more significant phases. Phases that have more influence than others in the I/O system and they could provoke a bottleneck or a poor performance. Analysis based on I/O phases allows identifying, delimiting, and trying to reduce each phase's impact on the I/O system. PIOM-PX is part of proposed model PIOM. PIOM integrates the I/O behavior model at POSIX-IO level (PIOM-PX) and the I/O behavior model at MPI-IO level (PIOM-MP, formerly known as PAS2P-IO). The model provides the information necessary to replicate an application's behavior in different systems using synthetic programmables programs. PIOM-PX-Trace-Tool allows interception of POSIX-IO instructions used during the application execution. The experiments carried out are executed in several standar HPC systems and the Cloud platform, where it is able to test the utility of the proposed model PIOM.

Keywords: HPC Systems; Parallel I/O; I/O model; I/O phase; POSIX-IO.

Resumen

Dado que el volumen de datos generado por las aplicaciones científicas crece y la presión sobre el sistema de E/S de los sistemas HPC también aumenta, se propone un modelo de comportamiento de E/S para las aplicaciones científicas paralelas de paso de mensajes (MPI -*Message Passing Interface*-) con el objetivo de analizar el impacto de las aplicaciones en el sistema de E/S. Analizar las aplicaciones paralelas MPI a nivel POSIX-IO permite observar cómo se tratan los datos de la aplicación a ese nivel. En este trabajo de investigación se presenta: la definición del modelo PIOM-PX, la metodología aplicada para extraer dicho modelo y la herramienta PIOM-PX-Trace-Tool. Dado que PIOM-PX está basado en el concepto de fase de E/S, se pueden identificar las fases más significativas. Fases que tienen más influencia que otras en el sistema de E/S, que podrían provocar un cuello de botella o un rendimiento pobre. El análisis en base a las fases de E/S permite identificar, acotar e intentar reducir el impacto de esas fases sobre el sistema de E/S. PIOM-PX forma parte del modelo propuesto PIOM que integra el modelo de comportamiento de E/S a nivel de POSIX-IO (PIOM-PX) y el modelo de comportamiento de E/S a nivel de MPI-IO (PIOM-MP, antiguo PAS2P-IO). El modelo proporciona la información necesaria, para que utilizando programas sintéticos programables se pueda replicar el comportamiento de la aplicación en diferentes sistemas. PIOM-PX-Trace-Tool permite interceptar instrucciones de POSIX-IO utilizadas durante la ejecución de la aplicación. Los experimentos realizados se han ejecutado en varios sistemas HPC estándar y en la plataforma Cloud, donde se ha podido comprobar la utilidad del modelo propuesto, PIOM.

Palabras clave: Sistemas HPC; E/S paralela; Modelo de E/S; Fase de E/S; POSIX-IO.

Resum

Donat que el volum de dades generat per les aplicacions científiques creix i la pressió sobre el sistema d'E/S dels sistemes HPC també augmenta, es proposa un model de comportament d'E/S per les aplicacions cientifiques paral.leles de pas de missatges MPI *(Message Passing Interface)* amb l'objectiu d'analitzar l'impacte de les aplicacions en el sistema d'E/S. Analitzar les aplicacions les aplicacions paral.leles MPI a nivell POSIX-IO permet observar com es tracten les dades de l'aplicació en aquest nivell. En aquest treball de recerca es presenta: la definició del model PIOM-PX. la metodologia aplicada per extraure el model i l'eina PIOM-PX-Trace-Tool. Donat que PIOM-PX està basat en el concepte de E/S, es poden identificar les fases més significatives. Fases que tenen més influència que altres en el sistema d'E/S, provocant un coll d'ampolla o un rendiment pobre. L'anàlisis en base a les fases d'E/S permeten identificar, acotar i intentar reduir l'impacte d'aquestes fases sobre el sistema d'E/S. PIOM-PX forma part del model proposat PIOM que integra el model de comportament d'E/S a nivell de POSIX-IO (PIOM-PX) i el model de comportament d'E/S a nivell de MPI-IO (PIOM-MP, antic PAS2P-IO). El model proporciona la informació necessaria, per a que utilitzant programes sintètics programables es pugui replicar el comportament de l'aplicació en diferents sistemes. PIOM-PX-Trace-Tool permet interceptar instruccions de POSIX-IO utilitzades durant l'execució de l'aplicació. Els experiments realitzats s'han executat en varis sistemes HPC estandard i en la plataforma Cloud, on s'ha pogut comprovar la utilitat del model proposat, PIOM.

Paraules clau: Sistemes HPC; E/S paral.lela; Model d'E/S, Fase de E/S; POSIX-IO.

Índice general

Índice de figuras

Índice de tablas

0

International Doctor Mention (Thesis Overview, Introduction and Conclusions)

1. Introduction

Nowadays, scientific applications can be executed in HPC (High Performance Computing) systems without big problems. Scientific applications generate a lot of data which must then be analyzed and visualized. The scientific community thinks that the data volume generated by the scientific applications will increase in the next years. However, when the computational system requests or saves the generated information in the corresponding storage system, significate wait times may occur, owing to the time needed by the I/O system to attend requests.

There are different factors that can generate wait times:

- Storage velocity: when the I/O operations represent a significant percentage on the execution time of applications.
- Applications executed in a specific HPC system: the features of an application can differ and patterns of I/O access can influence wait times.
- The I/O system: is a complex system with a hardware and software components. These components have different interrelated levels. Besides, these components can be configured with different features such as performance for the I/O system.

To solve these factors, the scientific community has presented different proposals. For the first factor, there are different works proposing how to design I/O architecture and manage the I/O software stack [1, 2]. For the second factor, one proposed solution is MPI-IO collectives. In other words, one of the design objectives of I/O system is to reduce these wait times.

Taking into account these factors, we asked:

- Why the same application executed in a specific HPC system has no a problem, but this application executed in another HPC system generates bottlenecks and poor performance?
- Why an application has wait times and others applications have no wait times in the same HPC system?
- Why are the bottlenecks produced?

In fact, a system is needed that can analyze the interaction between the application and the HPC system in order to establish strategies to avoid these problems. There are several factors that influence the generation of bottlenecks such as the interaction of the application with the system (I/O network system and I/O system) [3]. For this reason, sometimes, the combination of characteristics of the I/O system together with the features of the applications cause bottlenecks and poor application performance. But, what relation/dependency exists between the application and the I/O system configuration of HPC system?

In traditional HPC systems, the applications executed are different and their features can also be different. In addition, this type of HPC systems have to take into account that applications share the I/O subsystem and the modifications carried out to improve a specific application can negatively affect others applications' performance. Traditional HPC systems only allow changing some parameters at the user level to attempt obtain an I/O configuration more favorable for a specific application. However, when the workload of the HPC system is unknown, the task of tuning that HPC system to the set of applications is complicated. For this reason, it is important to analyze what occurs when an application is executed in a specific HPC system.

To carry out the evaluation or to experiment with these systems, it is important to have private dedicated systems, via simulation or using the Cloud, to install different components such as the filesystems. Thus, with these systems, a user can evaluate and analyze the application with different I/O configurations. But, to look for the I/O configuration more suitable for an application, would it be necessary to execute the application in all possible I/O configurations? Would it be possible to delimit the number of possible configurations to evaluate? If the application needs hours to execute, would it be necessary to execute the real application to detect if a specific configuration is advisable for that application?

1.1. Hypothesis and key challenges

Our interest is focused on the I/O system of HPC systems. In particular, to analyze the I/O behavior of scientific parallel applications. This analysis is carried out to understand the relationships and dependencies that exist when these applications are executed in an HPC system with a specific I/O configuration. The goal is to obtain an analysis tool that doesn't require the modification of source code or the recompilation of I/O software stack components.

However, there is a dependency between application and the I/O system but the application's functional behavior is always the same and it is independent of HPC system where the application is executed. But it has been observed that the application and the system influence one another mutually. For this reason, it is important to know how the application accesses the file, in what order the I/O operations are carried out when they access the storage system and how the I/O system processes the requests. The access pattern that the application uses is a factor that affects to the performance obtained for the application in a specific HPC system. Other factors that affect the performance are: the network components and the I/O configuration system [3].

The importance of knowing an application's I/O requirements a priori become more evident with the apparition of Cloud platform [4] due to the cost that the user must take into account per used resources, the used time and data volume transferred. To use the resources efficiently, it is important to know the application requirements because the user builds, configures, and manages a Virtual Cluster on Cloud (VCC).

Before beginning the analysis, we raised a number of challenges we have faced during the investigation:

- at instrumentation level: where do you make the instrumentation inside of the I/O software stack? What information do you have to extract? What do you have to measure? When do you make the instrumentation? How much overhead does the instrumentation introduce in the application's execution? What overhead value is acceptable and when?
- at analysis level: How do you establish the relation between obtained data and analyzed data? Can you extract more information that you can process? When do you discover the correspondence between data? How do you detect the principal cause that generates the performance problems?

The purpose of resolving these challenges is to know the possibility to develop better practices and adjust the applications, improve the system software and, to design and acquire better systems.

1.2. Objectives

The principal goal of this research has been

The definition a model that can describe the intrinsic behavior of I/O operations of parallel scientific applications for HPC systems. This model has to be independent of the system where is executed (portable) and must be able to to represent and replicate the application behavior both on physical and virtual clusters.

The research aim is to propose a model to represent the application I/O behavior at POSIX level and a methodology to obtain relevant information about the behavior of parallel scientific applications, in particular message passing (MPI) applications, that is able to reduce the impact on the I/O system. It is proposed to address the I/O characterization subject of a parallel application in order to:

- Know the application behavior, to obtain a behavior model that allows replicating this behavior and, to make the analysis and the evaluation of systems with the goal to improve the application performance and to mitigate the bottlenecks generate for the I/O.
- Select the resources taking into account efficiency criteria on the cloud platform.

To achieve this, the following specific goals have been established:

- To select the parameters that will form part of the model and that describe the application I/O behavior. These parameters can't depend on the I/O system where the application is executed.

- To identify the significant I/O phases that allows the analysis of I/O behavior in different systems.
- To use the information that describes the I/O phases to replicate the application's I/O behavior in different systems.
- To establish a methodology that can use this model to select the Cloud configuration parameters.

1.3. Contribution

The analysis of message passing (MPI) application I/O behavior can be used to understand the relation/interaction with the I/O system. This knowledge will allow the improvement of some application aspects such as performance, efficiency and mitigate the generation of bottlenecks from I/O system.

This work provides the scientific community with a new model to analyze and understand application I/O behavior, selecting the significant phases of an application when it accesses to the storage system. This model allows extracting knowledge about the application behavior at MPI-IO level and it is completed with the behavior information at POSIX-IO level. Having information in two levels facilitates observing what happens in these levels and allows establishing relationships regarding what happens at each level.

In addition, this research proposes a methodology to use the Cloud as a testing platform to make tests (testing different I/O configurations, including the filesystem).

1.4. Structure of Thesis

This thesis is organized as follows: in Section 2 the I/O concepts are described and the related work is presented. Section 3 presents the features of the I/O model defined at POSIX-IO level, the PIOM-PX model, to represent the I/O behavior. Section 4 describes the PIOM-PX-Trace tool and PIOM-PX-Analysis tool. Section 5 explains the methodology applied to extract the PIOM-PX model. Section 6 presents a use case of the PIOM-PX model in Cloud. Finally, in Section 7, the conclusions and future work are explained.

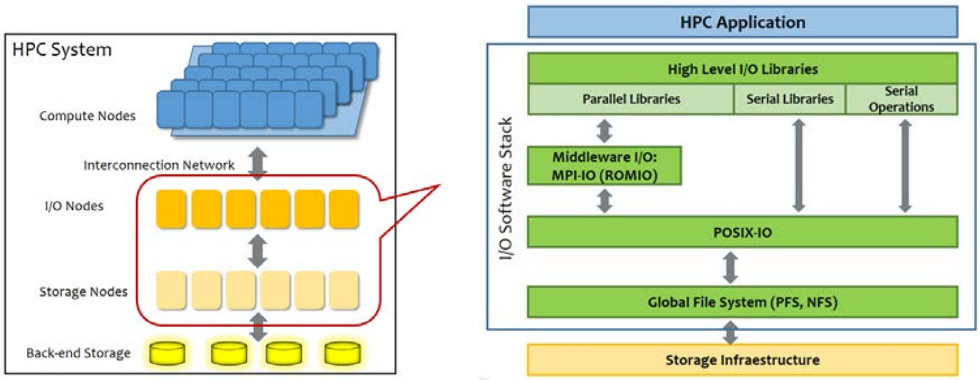
2. I/O System for the High Performance Computing

In this chapter, the I/O subsystem of HPC systems and the application features executed in these systems are described. Additionally, the related works that focus on the analysis, evaluation, and improvement of the I/O system are presented.

2.1. I/O System.

The performance of most parallel scientific applications (known as HPC applications) not only depend on computation, but also depend on memory, network and the I/O system (see Figure 1). For this reason, improper use, management, or configuration of the I/O system can negatively affect to the HPC system performance.

The I/O system is a complex system with two components: a hardware component and a software component (I/O Software Stack). These components have different layers and the interrelations between these layers, are also complex. Figure 1 shows the different layers of the hardware and software component.



(a) Components of I/O System architecture (b) modules of I/O Software Stack

Figura 1: I/O System

The hardware component (see Figure 1(a)) is composed by the I/O network (between compute nodes and I/O nodes), I/O nodes, the network between I/O nodes and storage physical devices, and the storage physical devices. In recent years, a new hardware solution has appeared named Burst Buffer [5].

The software component (see Figure 1(b)), or known as I/O software Stack, has the following layers:

- High-Level I/O libraries: optional layer that abstracts the I/O system features from the user. These libraries allow the development of portable programs. Some of these libraries are HDF5 [6, 7], PnetCDF [8], ADIOS [9], PASSION [10], PIDX [11] y SIONLib [12],
- I/O middleware: layer that covers features of parallel I/O such as collective I/O operations. These libraries are MPI-IO [13] and ROMIO [14, 15] (ROMIO is a high-performance, portable implementation of MPI-IO),
- Low-Level I/O libraries: POSIX-IO [16] and,

- Global Filesystem: PVFS2 [17], Lustre [18], GPFS [17],

When HPC systems were developed the HPC scientific community bet on POSIX-IO (*Portable Operating System Interface*) as the standard model to access to the storage system in a global way. POSIX-IO has been updated but has one problem, it does not allow applications to scale. For this reason, some authors developed new layers in the I/O Software Stack.

The layers closest to the Global Filesystem are more dependent to the I/O system.

2.2. Scientific parallel applications.

Scientific applications are another important element in HPC system performance. Scientific applications generate a large volume of data and they need to save the data to do posterior analysis and visualization, or to perform checkpointing.

Scientific applications can be classified depending on if they are compute, data or I/O intensive. Mendez et al. [19] define the I/O severity concept to identify the I/O impact degree on an HPC system.

Scientific parallel applications' features can affect the HPC systems at compute, network and I/O level. Understanding an application's I/O behavior and how it uses the I/O system is not an easy task due to the existence of complex interactions between software layers [1]. For example, in the case of increasing the number of processes executing the parallel application. For this reason, a parallel application classified as compute intensive can become an I/O intensive parallel application.

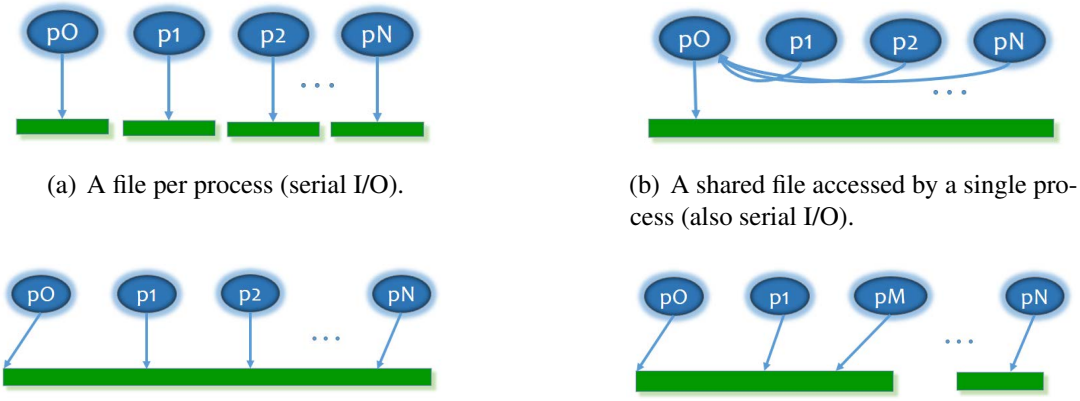
To discover the application I/O features, the I/O strategies used to distribute access to file data must be identified. Depending on the file access type, the I/O is classified as serial or parallel. As our research is focused on the MPI scientific parallel applications executed in HPC systems, the parallel I/O would have to provided a better performance that serial I/O.

Inside of parallel I/O there are different cases (see Figure 2): a file per process (serial I/O), a shared file which can only be accessed by a single process at a time (also will be serial I/O), a shared file which can be accessed by all processes executing the application, and and shared file which can be accessed by some processes (where this number is less than the total number of processes executing the application).

Furthermore, the access pattern used by the parallel application to access data in a specific file must be taken into account. The behavior is represented by access patterns in the different files of the application. Four access patterns can be differentiated (see Figure 3).

Identifying the access type to the files and detecting what I/O operations are carried out in every access is important. But, another important factor is the order that the I/O operations are carried out in to maintain information consistency in parallel filesystems that allow concurrency.

Depending on the volume of data transferred in each access, sometimes techniques must be applied to optimize for this access in order to try to optimize the parallel I/O. One of these techniques is the



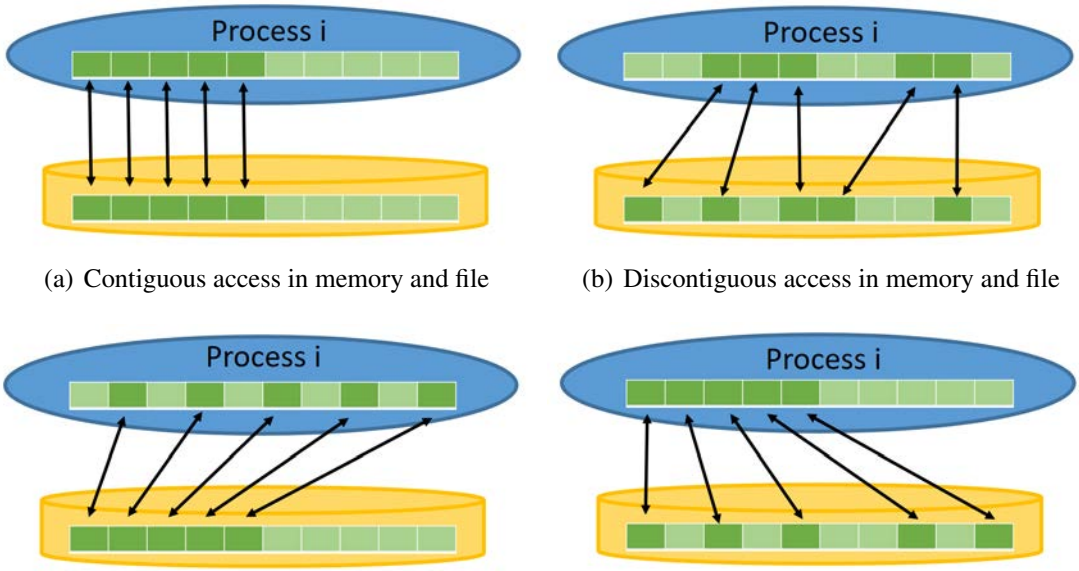
(a) A file per process (serial I/O).

(b) A shared file accessed by a single process (also serial I/O).

(c) A shared file accessed by all processes executing the application.

(d) A shared file accessed by some processes.

Figura 2: I/O strategies



(a) Contiguous access in memory and file

(b) Discontiguous access in memory and file

(c) Discontiguous access in memory

(d) Discontiguous access in file

Figura 3: Access pattern types.

collective I/O operations [20] which consists of improving performance by grouping I/O requests from various processes.

2.3. Tools to analyze the I/O.

There exist a number of tools to analyze I/O activity and application I/O behavior, such as the following.

Wright at al. in [2] propose RIOT I/O, a tool that characterizes the application from MPI level to the filesystem. Besides, they introduce a post-process tool is able to generate a statistical reports and graphical representation of I/O activity. Kim et al. in [21] propose IOPin, a binary tool that detects only one I/O critical path, a bottleneck. It captures functions called at the MPI-IO level and associate these functions with calls to the filesystem of PVFS. Luu et al. in [1, 22] propose Recorder, a tool designed to work with parallel filesystems that allows identification of the bottlenecks in the current implementation of reading metadata of HDF5. Pillet et al. in [23, 24] present Paraver, an analysis tool to detect performance problems and to understand an application's behavior. Vijayakumar et al. in [25] propose ScalaIOTrace, a tool that intercepts functions in different levels of the I/O Software Stack. Shende et al. in [26] present TAU, a tool to characterize an application at different levels of the I/O Software Stack. Kunkel et al in [27] present SIOX, a modular and scalable architecture to monitor, analyze, and optimize the I/O Software Stack. SIOX intercepts some functions at POSIX-IO, MPI-IO and the high-level I/O libraries.

The most of these tools require neither modification of the source code nor recompilations of I/O Software Stack components and provide statistical information.

There exist other approaches to analyze the application I/O behavior such as that proposed by Mendez et al. in [28]. These authors propose a methodology to characterize the parallel I/O of scientific applications at MPI-IO level. This methodology looks to obtain an application I/O behavior model based on the access pattern which is obtained from PAS2P-IO [29] (the result of adding to PAS2P [30], a module to analyze the application I/O and to identify the significant I/O phases). Or the approach to give the support to the configuration of I/O system proposed by B.Behzad et al. in [31]. This author proposes a tool that can generate a better I/O configuration of the I/O system because this configuration will allow better performance of I/O system.

Darshan and Vampir.

Two tools that have been used during this research are:

Darshan [32] is an I/O profiling tool that provides statistical information on I/O behavior. Darshan provides information such as the number of MPI-IO and POSIX-IO operations, shows I/O access sizes in ranges and the mode access type. Darshan instruments the HDF5, Parallel netCDF, MPI-IO and POSIX layers for MPI applications.

Vampir [33] is a tool that provides accurate information about the application behavior to allow a performance analysis and to visualize the events performed. Besides, Vampir provides statistical information.

In contrast of other tools, our tool allows us to analyze the application's I/O behavior based on I/O phase. Every I/O phase identifies a different behavior pattern, the operation type carried out, the phase weight (I/O data volume), and the number of repetitions are analyzed. This analysis allows us to

understand the behavior of the application at two layers (MPI-IO an POSIX-IO) of the I/O system and to to have an accurate vision of the access patterns.

2.4. Conclusions.

The I/O subsystems of HPC systems are complex and prone to bottlenecks. Nevertheless, analyzing only the I/O system is not sufficient to discover why these bottlenecks are generated. Analyzing and identify the I/O strategy that and application uses, the access patterns that the processes use to access the file information, the data volume transferred in every access and the I/O operations carried out by the application provide information to understand why the bottlenecks are generated and/or poor performance is obtained in a HPC system with a specific I/O configuration.

3. PIOM-PX: an I/O behavior model for parallel applications at POSIX-IO level

In this chapter, a model is presented to describe parallel applications' I/O behavior at the POSIX-IO level and its integration with application I/O behavior at the MPI-IO level. Because, to minimize the I/O impact in the parallel systems (architecture and application) it is necessary to know the scientific parallel applications' I/O behavior.

3.1. Introduction.

Scientific applications resolve problems of different knowledge areas such as physics, chemistry, materials and biomedicine. These applications are developed by scientifics domain expertise, but the majority of them are unaware of the characteristics of HPC systems where the applications are executed. When these applications are executed in the HPC systems, in the I/O system can generate bottlenecks or poor performance.

To identify the reason why these problems originate in the I/O system, the application, the interaction between the application and the I/O system, and the I/O system must be analyzed. In our case, the research focuses on the analysis of MPI parallel applications to try understand and manage the impact in the I/O system. The analysis of parallel applications focuses on extracting the I/O behavior of these applications.

3.2. Modelling the I/O behavior.

Firstly, in this thesis, when the **I/O behavior** of an application is mentioned, it refers to the way in which the application's I/O accesses the open file(s) are made during the execution of that application.

In other words, the I/O behavior describes when the application accesses the file or files, how the application accesses the file or files, how much data are transferred by the application to the file in every I/O access, what I/O operations are carried out and what order they are carried out in.

To extract the parallel applications' I/O behavior, it is necessary analyze the I/O bursts generates by the application, to identify the I/O patterns. The applications can have different I/O patterns, and these patterns affect the application performance. An **I/O pattern** is a set of recurrent I/O events, that repeat during the application execution.

To represent the I/O patterns a model is needed. In our case, an I/O behavior model for the I/O intensive parallel MPI applications at POSIX-IO level is defined. In order to identify which applications are considered I/O intensive, a way to identify these types of applications was proposed in [34] taking into account the I/O requirements and the I/O time required by the application. To define a model, the basic elements must first be identified, then the relationships between them (phases) need to be considered, and then finally a way to represent the phases must be proposed.

The model proposed to describe the parallel application I/O behavior, named PIOM-PX, is carried out to the file level (file logical view level). At file level the spatial and temporal pattern is identified. The **spatial pattern** provides information about: what file is used by the application, what process accessed a file and where (file positions) it was accessed by the process during the application execution. The **temporal pattern** informs the order that the access events are carried out to the file.

Figure 4 shows an example of a parallel application I/O behavior representation proposed by PIOM-PX. PIOM-PX can represent the following information

- Number of processes involved in the application execution,
- I/O events (read and write operations) carried out by the application,
- Event order,
- Offset used by the I/O events to displacement inside of file, and
- Number of phases that the application has.

In the case that the application uses more than one file, PIOM-PX also allows to represent the files used during the application execution. The output model also shows information regarding the data volume that is accessed or transferred.

Analyzing Figure 4, it can be observed with PIOM-PX that the application was executed with 16 processes, the application uses a share file and the 16 processes access the same file. PIOM-PX detects 40 write phases (each phase of the 16 processes has write operations and it is represented with a circle) and 1 read phase (represented with a square). This representation shows the order of phases (that contains event order). PIOM-PX detects 40 write phases following each other and, finally, one read phase.

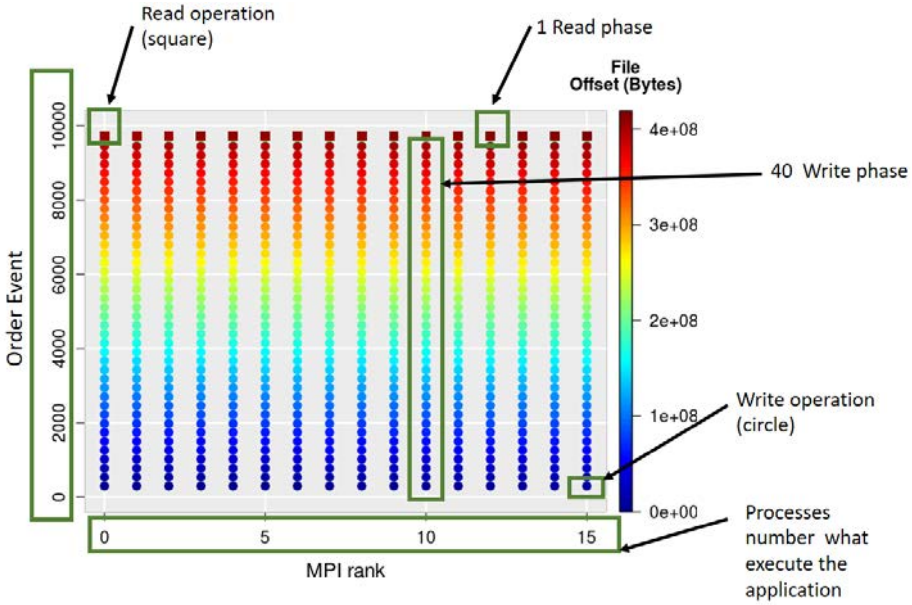


Figura 4: Representation of parallel application I/O behavior with PIOM-PX.

Since applications that use the MPI interface could be analyze with PIOM-MP (named PAS2P-IO in [29]). To build the model at POSIX-IO level, the methodology proposed by Mendez et al. [29] is followed.. They propose an I/O behavior model for the parallel applications at MPI-IO level (previously called PAS2P-IO PAS2P-IO and now named PIOM-MP). As a last phase, a tool designed to integration PIOM-PX and PIOM-MP is proposed. This integration gives rise to PIOM.

3.2.1. Description of I/O phase.

In most scientific parallel applications, homogeneous portions of code that are repeated during the execution can be detected. These similar portions receive the name phases [35]. This repetitive feature also can be applied to the I/O operations in the files. For this reason, an I/O behavior model for scientific parallel application that takes into account the phase concept has been defined.

To detect an I/O phase the file I/O operations are analyzed. An **I/O phase** is a consecutive sequence of I/O operations in a file. In a file, one or several I/O phases can be detected and and the number of times a phase appears determines the repetitions of the I/O phase in the file. To detect how many I/O phases contain a file, the I/O operations that appear in each phase are analyzed, including the order that they appear in and the data volume transferred in each phase. As PIOM-PX is based on the phase concept, it provides this information. In the example shown in the Figure 4, PIOM-PX detects that in each write phase only has one write operation while the read phase has 40 read operations.

The analysis of I/O operations allows identifying spatial and temporal patterns. This identification allows describing the application I/O behavior. The principal features taken into account in the operation

analysis are:

- Operation type (*IOP_type*): write (*w*), read (*r*) and write/read (w/r),
- Request size or I/O operation size (*rs*),
- Position in the file logical view (*offset*),
- File displacement (*disp*): difference between *offset* of two I/O operations consecutive.

The spatial pattern is extracted from the features of I/O operations. This pattern indicates what I/O operations are carried out and how the processes access the file. Considering always a logical view of file.

To extract the temporal pattern the order of occurrence of I/O events must be identified. In our case, one goal is to define a portable I/O model to represent the application I/O behavior independently of the system where that it is extracted from and that can be replicated in other systems. In a parallel system, it is always possible to order the events of the same node (if the monotonicity of the clock is respected), but causal relationships between events of different nodes can be distorted. To obtain the temporal pattern the order of the events is needed, for this **tick** and **subtick** are defined, two logical counters to control the order of occurrence. The tick controls the logical order that the MPI-IO events occur, while the subtick controls the logical order of POSIX-IO events.

Each process generates an events sequence and these events can be different such as send a message, receive a message, MPI I/O operations and POSIX I/O operations.

So, another I/O operations important feature is:

- Distance between two I/O operations (*dist*): difference between *tick.subtick* of two consecutive I/O operations.

Summarizing, the I/O behavior model of a file is described by the I/O phases set, that describe the file access and the repetitions number of every phase. Thus, the I/O behavior model for a file is represented by the set of I/O phases.

3.2.2. Definition of parallel application I/O behavior model.

To define an I/O behavior model of an MPI parallel application, at POSIX-IO level, PIOM-PX is proposed. Because, there are MPI applications that use only the POSIX-IO interface.

As we have seen, our first goal is to identify what information needs to be extracted to have an application representative model. As to the POSIX-IO layer is closer to the system, the information obtained can be system dependent. For this reason, our second goal, is to obtain information independent of system to have a portable model.

To achieve the two initial goals, the application is analyzed from two points of view: file and I/O operation. In this way, the application features at file and I/O phase level can be identified.

3.2.3. I/O features of parallel application.

PIOM-PX extracts the application I/O behavior analyzing the files generated by the application.

The parameters defined for the model PIOM-PX, to represent the parallel application I/O behavior, are shown in table 1. The model parameters are defined at a file level, I/O phase level, and application level. These parameters can define most application features. The last column of table indicates where the field is obtained (more details in Chapter 4).

File level parameters.

The parameters that are of interest are those that determine what file is accessed, how the file is accessed and how many processes access the file. The parameters are obtained per analyzed file. Some parameters are extracted directly from application traces while others are calculated. These parameters are:

- *file_id*: file identifier. Allows the the identification of the I/O operations associated with the analyzed file.
- *file_name*: file name to analyze. When a specific file is opened, it is assigned a *file_id* to control the analyzed file.
- *file_size*: size of the analyzed file.
- *file_np*: counter that registers the number of MPI processes that open the file $file\_id$.
- *file_accessmode*: mode how access to the file: sequential, strided or random.
- *file_fileaccesstype*: determines the type how the file is opened: read-only (R), write-only (W) or write and read (W/R).
- *file_accesstype*: determines if the file is a file per process or a share file. This information is obtained by the process number ($file\_np$) that accesses a specific file.
- *file_nphase*: counter that registers the number of phases that contain a file.

I/O phase level parameters.

The I/O behavior model allows representing the application I/O access patterns when it accesses the file. Thus, an I/O phase can be defined as a repetitive sequence of the same access pattern in a file.

At operation level, the interesting information for the model is:

- *IOP_type*: data access operation type: write, read or write/read.

Tabla 1: PIOM-PX Model Characteristics

Identifier	**File**	**Origin**
file_id	File Identifier.	Trace File
file_name	File Name.	Trace File
file_size	File Size.	Post-process
file_np	Count of MPI processes that open the file $file\_id$.	Post-process
file_accessmode	This can be sequential, strided or random.	Post-process
file_fileaccesstype	Read-only(R), write-only (W) or write and read (W/R).	Trace File
file_accesstype	$file\_np$ processes can access to shared Files or 1 File per Process.	Post-process
file_nphase	Count of phases of the file.	Post-process
	I/O Phase (PhIO)	
Ph_id	Identifier of an I/O phase.	Post-process
Ph_processid	Identifier of process implied in the phase.	Post-process
Ph_np	Number of processes implied in the phase.	Post-process
Ph_weight	Transferred data volume during the phase. Expressed in bytes.	Post-process
Ph_nrep	Number of repetitions per phase.	Post-process
Ph_niop	Number of I/O operations.	Post-process
IOP_type	Data access operation type, which can be write, read, or write/read.	Trace File
rs	Request size or size of an I/O operation.	Post-process
offset	Operation offset, which is a position in the file's logical view.	Trace File
disp	Displacement into file, which is the difference between the offset of two consecutive I/O operations.	Post-process
dist	Distance between two I/O operations, which is the difference between $tick.subtick$ of two consecutive I/O operations.	Post-process
	Application	
app_np	Number of processes that the application needs to be executed.	Post-process
app_nfiles	Number of files used by the application.	Post-process
app_st	Storage capacity required by the application for the input files, temporary files and input/output files.	Post-process

- *rs*: request size of an I/O operation size.
- *offset*: file logial view position where the process has to write or read the data.
- *disp*: displacement in the file. Determined by the difference between the *offset* of two consecutive I/O operations.

- *dist*: distance between two I/O operations, that it is the difference between $tick.subtick$ of two consecutive I/O operations.

Every I/O phase has one or several consecutive operations. The general parameters that define and identify a phase in a file are:

- *Ph_id*: phase identifier.
- *Ph_processid*: identifies the process involved in the phase. It can be one or various processes identifiers.
- *Ph_np*: number of processes involved in the phase.
- *Ph_nrep*: number of times that a specific phase is repeated in a file.
- *Ph_niop*: I/O operations number that contains a phase.

Another important factor is the weight of an I/O phase (*Ph_weight*) that corresponds to the data volume transferred during the phase. It is expressed in bytes and, it is calculated with the following formula:

$$Ph\_weight_(Ph\_id) = Ph\_np \times rs \times Ph\_niop \times rep \quad (1)$$

Determining the data volume transferred in a phase is a criteria that allows identifying the significant phases. The significant phases are the phases that can have more influence on the application performance.

Application level parameters.

The analysis at file and operation level allows obtaining information at the application level. At the application level, the parameters considered important are:

- *app_np*: number of processes that the application needs to be executed.
- *app_nfiles*: number of files opened by the application during execution. Taking into account both the files that the application needs to execute (input files) and the generated files as a result of execution (output files).
- *app_st*: the input and output files size allows the storage capacity that the application requires to be calculated. If applicable, it must also take into account the temporary files generated during the application execution. This value is calculated as follows:

$$app\_st = \sum_{i=1}^{nf\_IOF} file\_size\_IO_i + \sum_{i=1}^{nf\_INF} file\_size\_IN_i \quad (2)$$

where *nf_IOF* is the number of output files, $file\_size\_IO_i$ is the size of each output file, *nf_INF* is the number of input files necessary to execute the application, and $file\_size\_IN_i$ is the size of each input file.

Finally, PIOM-PX has been integrated into PIOM-MP. As a result, the result is PIOM. Thus, PIOM can represent the application I/O behavior at two levels: MPI-IO and POSIX-IO. Figure 5 shows the postion of PIOM-PX and PIOM-MP models in the I/O Stack Software.

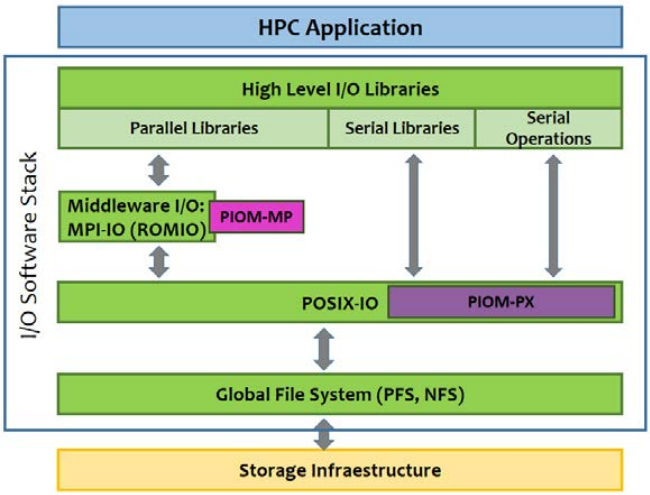


Figura 5: Location of PIOM-MP and PIOM-PX models in the I/O Stack Software. These models allow intercepting and analyzing the operations at MPI-IO level and POSIX-IO respectively.

3.3. Cases analysis

In this point, the experiments carried out have the goal of validating the functionality of PIOM-PX and the integration with PIOM-MP. These experiments allow checking how the PIOM model represents different I/O access patterns that show different I/O behaviors.

3.3.1. Applications.

Initially, the experiments have been carried out taking into account the IOR benchmark [36] as the application and a synthetic program written in Fortran. IOR allows generating different I/O access patterns via customizable parameters This benchmark is written in C. The parameters that are configured due to their relevance are: -a, -b, -c, -F, -N, -s and -t.

IOR has been executed with Intel and GNU compilers to analyze its influence on the detected operations. The MPI distribution used was Intel MPI 2017.

3.3.2. HPC system.

The experiments are carried out in two HPC system: Finesterrae2 (CESGA) [37] and SuperMUC (LRZ) [38]. Table 2 shows the HPC systems features. These systems use different filesystems, which allows an appreciation of the impact that the filesystems have at the POSIX-IO level.

Tabla 2: Description of HPC Systems

Components	**Finisterrae2 (CESGA)**	**SuperMUC (LRZ)**
Compute Nodes	306	9216
CPU cores (per node)	24	16
RAM Memory	128GB	32GB
Local Filesystem	Linux ext4	Linux ext3
Global Filesystem (GFS)	NFS	NFS
Capacity of GFS	1.1TB	10x564x10TB
Global Filesystem (PFS)	Lustre	GPFS
Capacity of PFS	695TB	12PB
MPI-Library	mpich2 (1.5)	IBM PE
Number of data servers	4 OSS and 12 OSTs	80 NSD
Number of Metadata Server	1	
Stripe Size	1MiB	8MiB
Interconnection	Infiniband FDR@56Gbps	Infiniband FDR10

3.3.3. Experiments

The aim of these experiments is to extract the I/O behavior from the proposed and performed experiments. The analysis presented below was published in [34]. The experiments with IOR were defined to check two strategies: `1 file per process` and `1 shared file`. The *strided* pattern was achieved using the collective buffering technique in `enable` and `automatic` mode. We executed the experiments for 16 MPI processes per compute node. Each experiment is described as follows:

(a) 1 File per Process using POSIX interface:

- Objective: Detect the POSIX-IO operations for an application that only uses POSIX as I/O library.
- Command Line: `IOR -a POSIX -s 1 -b 8m -t 1m -F`

(b) 1 File per Process using MPI-IO interface:

- Objective: Detect the POSIX-IO operations generated by an application that uses independent MPI-IO operations.
- Command Line: `IOR -a MPIIO -s 1 -b 8m -t 1m -F`

(c) A single shared file using collective buffering technique in `automatic` mode for a strided pattern:

- Objective: Detect the POSIX-IO operations generated by an application that uses collective MPI-IO operations.
- Command Line: `IOR -c -a MPIIO -s 16 -b 512k -t 512k`

(d) A single shared file using collective buffering technique in `enable` mode for a strided pattern:

- Objective: Detect the POSIX-IO operations generated by an application that uses collective operations with the collective buffering technique enabled.
- Command Line:

```
romio_cb_read = enable
romio_cb_write = enable
IOR -c -a MPIIO -s 16 -b 512k -t 512k
```

Table 3 presents PIOM-PX model parameters for the four IOR experiments at application and file level.

Tabla 3: PIOM-PX Parameters for the IOR Benchmark

Identifier	**(a)**	**(b)**	**(c)**	**(d)**
app_np	16	16	16	16
app_nfiles	16	16	1	1
app_st	128 MiB	128 MiB	128 MiB	128 MiB
File				
file_name	testFile<IdProcess>	testFile<IdProcess>	testFile	testFile
file_size	8 MiB	8 MiB	128 MiB	128 MiB
file_accessmode	Seq	Seq	Strided	Strided
file_fileaccesstype	W/R	W/R	W/R	W/R
file_accesstype	1Fx1Proc	1Fx1Proc	Shared	Shared
file_nphase	2	2	2	2
file_np	1	1	16	16

To explain the I/O phases detected, we present snippets of trace files for the first experiment, where lines with "##"specify the names of the displayed fields of the following trace lines. Furthermore, we detected two I/O phases for the four experiments because this depends on the IOR logic. The four

experiments are presented in [34]. For this reason, we only show a detailed figure of the I/O behavior for experiment (a). The first experiment is explained as follows:

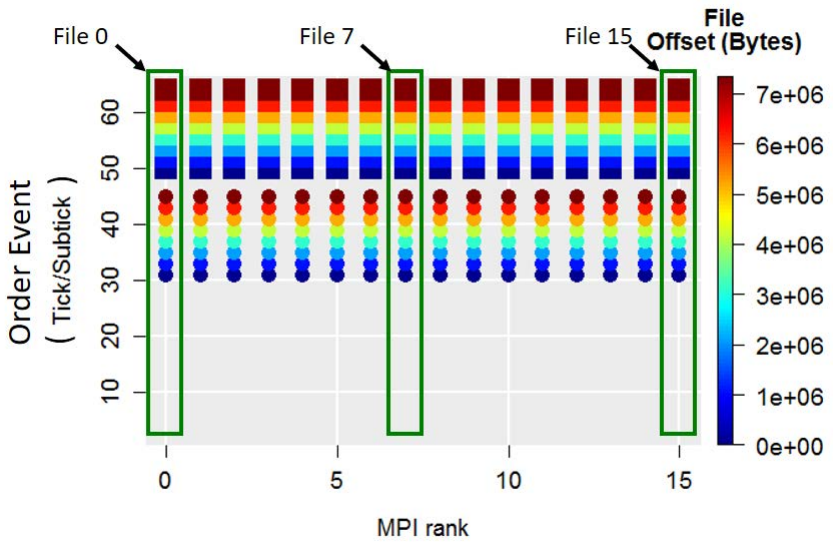
(a) 1 File per Process using POSIX interface: IOR is configured to write 8 MiB per MPI process by using the POSIX interface for the I/O strategy 1 File per process. Each MPI process writes and reads in request size of 1 MiB ($rs = 1MiB$). Figure 6 shows the I/O behavior for experiment (a). Two I/O phases are shown per file. The first phase (phase 1) contains 8 write operations and the second (phase 2) 8 read operations. Thus, the Ph_niop parameter of our model will be 8 for each of the phases. 16 files are accessed in parallel. When executing IOR there is a burst of communication between process 0 and the rest (see Figure 6). This burst generates a sequence of 30 communication events. Event 30 (tick =30) is when the writing phase begins, since it is when the first writing operation takes place that will go from tick.subtick 30.1 to 30.8. The reading phase is generated in the tick.subtick equal to 50.0 through 50.7.

Snippet 1 presents part of the trace file of *IdProcess* 2, which shows part of the operations of Phase 1. We can observe that a `lseek64` operation is called before each `write` and this also occurs for `read` operations.

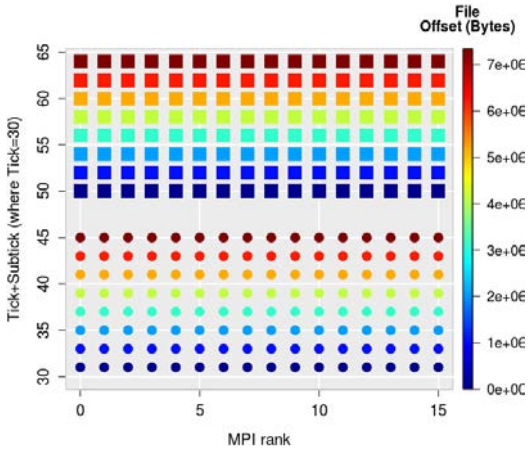
Snippet 1: Trace file of *IdProcess* 2

```
## IdProcess file_id file_name NameOperation  tick subtick
2 6 testFile.00000002 open64 30 0
## IdProcess file_id NameOperation offset tick subtick
2 6 lseek64 0 30 1
## IdProcess file_id NameOperation offset rs tick subtick
2 6 write 0 1048576 30 2
2 6 lseek64 1048576  30 3
2 6 write 0 1048576 30 4
...
```

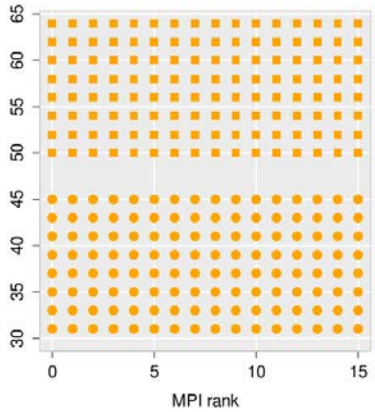
Finally, it can be conclude that PIOM-PX detects the spatial and temporal pattern for the POSIX-IO level. These patterns are represented by the I/O phases. The results showed for the spatial pattern depend on I/O operation and the I/O method. It has selected the same quantity of data, the MPI processes number and a similar strategy, but the behavior is influenced by the techniques and the I/O interface. It has been detected that the collective technique does not work in automatic mode because an I/O aggregator by a computer node was expected and it has checked that all MPI processes made I/O. Experiments (c) and (d) have shown that the MPI-IO operations are the same, but at POSIX level the operations depend on hint values of the ROMIO library and the hints are explicitly defined in the application. PIOM-PX considers this property to provide information to the user and help her to understand what the I/O library is doing with the defined operations in the application logic.



(a) File offset representation



(b) File offset representation (Zoom)



(c) Phase Weight representation (Zoom)

Figura 6: IOR for POSIX interface configured for 16 MPI processes, 1 File per Process for a sequential pattern. Bullet (smaller circle) corresponds to write operations and the filled squares to read operations. Two I/O phases can be observed for each file. The Phase 1 is composed of 8 write operations and Phase 2 of 8 read operations. The color scale in Figure 6(c) shows the weight, which is 1 MiB $\times$ $file\_np$ per subtick. The weight for both Phase 1 and Phase 2 is 8 MiB for each file per process.

3.4. Conclusions

PIOM-PX is an I/O behavior model that allows extracting the application information at POSIX-IO level. The vision at this level allows understanding what data volume is transferred in every access to the file, or how many accesses are carried out to transfer the data volume that user has defined at

the application level. However, it is important to establish the correlation between the data that the user thinks they send in every I/O bursts and the data sent in every access to the file. Establishing this correlation is not obvious. Because in every level of the I/O software stack, application data are treated in a different way, due to the I/O system configuration such as environment variables and filesystem.

To analyze an application at POSIX-IO level, it must be taken into account that POSIX-IO is the closest level to the filesystem. So, the dependency with the system increases. But, as one goal is to achieve a model which is independent of the system where the analysis was carried out (a portable model), the application analysis must detect the data dependent on the system and establish a relationship between one or more application features to disassociate the system data.

Finally, the integration of the model for the parallel applications that use serial I/O library (PIOM-PX) to the model for the parallel applications that use parallel I/O library (PIOM-MP) allows achieving our principal goal. This goal is to have a holistic portable model, PIOM, for the I/O of parallel applications. This holistic model allows covering the analysis of the most of the calls made to the global file systems.

The analysis of application I/O behavior with PIOM model is made over the files (the logical view).PIOM is focused on the I/O phase concept. Understanding the I/O phase as a sequence of consecutive I/O operations in a file. Identifying the I/O phases that a file has, allows the analysis that every I/O phase has on the HPC system.

4. PIOM at POSIX-IO level

This chapter describes the tool for tracing the application at POSIX-IO level and the algorithms used to identify the I/O phases with the goal of extracting the I/O behavior model at the POSIX-IO level. This work was presented in [34].

4.1. Introduction

The scientific parallel applications analyzed are MPI applications. The analysis is focused on applications written in C and Fortran. Thus, the intercepted events are between the instructions MPI_Init() and MPI_Finalize().

To intercept the POSIX-IO instructions the PIOM-PX-Trace tool has been developed and integrated with PIOM-MP-Trace to form the PIOM-Trace tool. This last tool allows intercepting the I/O functions at MPI and POSIX-IO level. PIOM-Trace has three trace options:

- only MPI events,
- only POSIX-IO events,

- MPI and POSIX-IO events.

The characterization of the application is done dynamically. To intercept the I/O operations of an application, load the tool, PIOM-trace, with the following command

export LD_PRELOAD=path_library/libpiom.so

This allows the tool to intercept the MPI and/or POSIX-IO events generated by the application without interfering with its execution.

4.2. PIOM-PX-Trace Tool

In this section, the implementation of the PIOM-PX-Trace tool necessary to intercept I/O events at POSIX-IO level will be described. This tool is used in the *Application Trace stage* of the proposed methodology. In relation to the POSIX-IO operations that are intercepted, the following list describes the operations that are taken into account for the characterization of the application. The choice of intercepting these operations is due to the fact that they are the most commonly used operations in applications. These operations are:

```
open, open64, fopen64, close, fclose, creat, creat64, read,
write, pread, pwrite, pread64, pwrite64, readv, writev,
fread, fwrite, lseek, lseek64, fseek, __xstat, __xstat64,
mmap, mmpa64, __lxstat, __lxstat64, __fxstat, __fxstat64,
fdatasync, aio_read, aio_write, aio_read64, aio_write64,
aio_return, aio_return64.
```

Each time an MPI communication or POSIX-IO event is intercepted, a trace line (TL) is recorded with the structure of the Table 4 in the trace file, with the physical times and the updated logical counters. In capturing these events/operations, the values of the function call parameters are extracted. At this point, the *tick* and *subtick* counters are updated. The *tick* is increased each time an MPI event is intercepted and this causes the *subtick* to be reset to zero each time the *tick* is increased. The *subtick* is increased whenever consecutive POSIX-IO operations are intercepted. However, post-processing is required to logically sort the ticks of events [35].

Finally, the traces files are obtained. One trace file for each MPI process.

4.3. Extracting I/O operations per file

All trace files are analyzed to identify the used files and the I/O operations that belong to every file defined by the tuple (`file_id`, `file_name`) used by the application. For every file used by the

Tabla 4: Structure of the trace line (PIOM-PX TL)

Identifier	**Description**
IdProcess	Identifier of Process.
file_id	Identifier of File.
TypeOperation	'MPI' or 'POSIX'.
NameOperation	Name of POSIX-IO operation.
offset	Operation offset, which is a position in the file's logical view.
rs	Request size of for data access operations.
Metadata-Line	
file_name	File Name.
FileAccessType	Open mode.
Added Fields	
Time	Logical time of the occurrence of an MPI or POSIX-IO event.
Final_compute	Duration of the call of an MPI or POSIX-IO event.
tick	Order of occurrence of the MPI events.
subtick	Order of occurrence of the POSIX-IO events.

application an *I/O file* is created. From information contained in the I/O file, the spatial and temporal patterns are extracted. In these files, the number of processes that accessed the used file together with the events that the processes used to access the file and the order of events can be observed. This analysis and the subsequent analysis are carried out off-line.

4.4. Updating I/O Operation fields

Each line registered in each I/O File that has missing values in some fields is analyzed.

This analysis refers to I/O operations that require the evaluation of other operations to obtain offset and rs values. For example, write and read operation depend on the evaluation of lseek operation to obtain the $offset$ value.

To determine the correct offset,the value of the *whence* parameter of `lseek` must be considered (see Figure 7): *SEEK_SET* (the file offset is set to offset bytes), *SEEK_CUR* (the file offset is set to its current location plus offset bytes) and *SEEK_END* (the file offset is set to the size of the file plus offset bytes) [39, 40] (see Figure 7).

For example,

- write(): *ssize_t write(int fd, void *buffer, size_t count)*;
- pwrite(): *ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset)*.

The *pwrite()* instruction has the explicit offset in the function call while the *write()* instruction

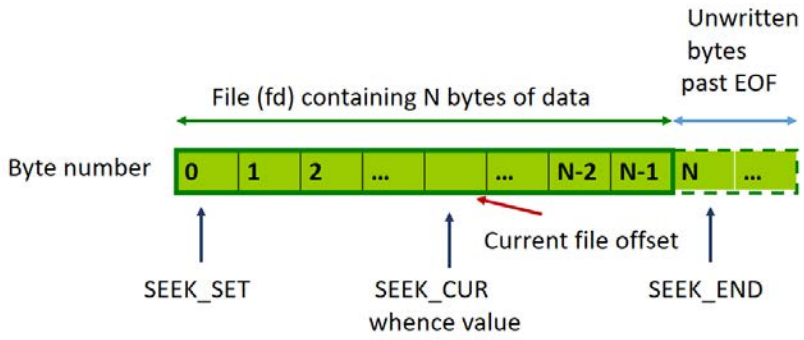


Figura 7: Representation of parameter *WHENCE* of *lseek* operation.

doesn't. To identify the request size (*rs*) and how the displacement is moved, a new field (*disp*) is added to the trace line.

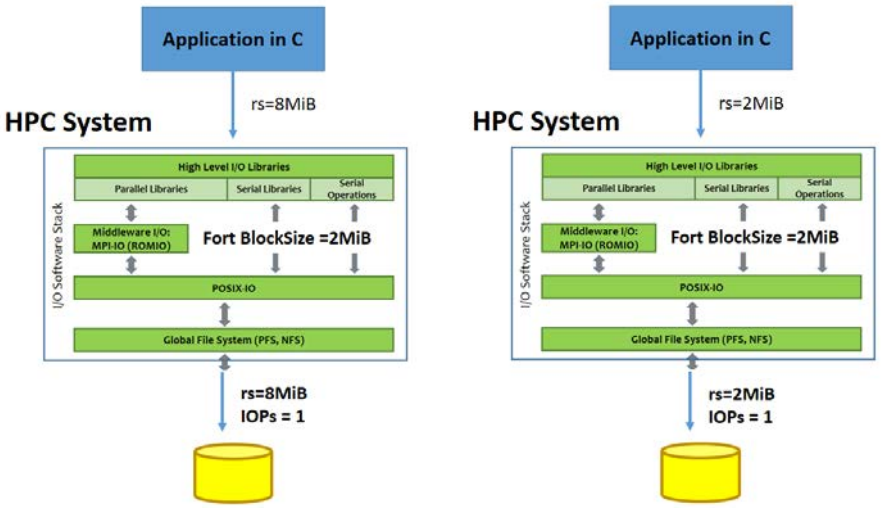
For the programs written in Fortran the FORT_BLOCKSIZE environment variable is evaluated. Because the information obtained at POSIX-IO level differs to the information that the user wants to send at the application level (see Figure 8). When the application is written in C the value of request size that can be observed is the same at application level and the POSIX-IO level. While when the application is written in Fortran the value differs.

4.5. Spatial pattern description

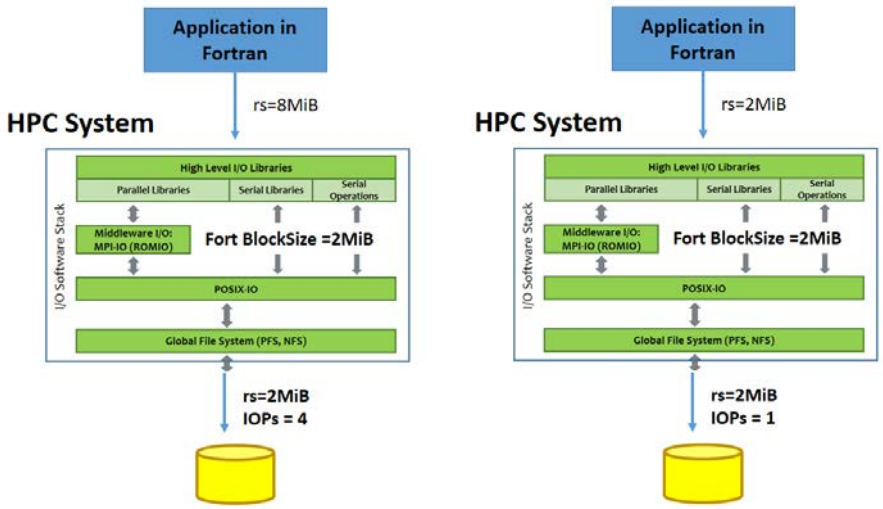
To extract the spatial pattern, every *I/O file* is analyzed. For the read and write operations the following fields are saved: `NameOperation, file_id, file_name, offset, rs, tick` and `subtick`.

For all operations that have the same $file\_id$, $file\_name$ and *rs*, the offset is calculated between two consecutive read or write operations and the displacement (*disp*) is calculated between two them.

To identify an I/O phase, the sequence of detected I/O operations must be analyzed. A sequence will have an operation or a set of consecutive operations with a similar $offset$ and *rs* similar. This sequence will be considered an I/O phase. The beginning and the end of an I/O phase are determined by the apparition of tick or a distance in apparition an I/O event. If a sequence is detected with the same operations, with similar $offset$ and *rs* and in the same order of appearance as in a previous sequence, the *rep* counter is increased. This counter indicates the number of repetitions in which a certain sequence appears in a file. If this sequence does not correspond to any of the detected sequences, a new sequence is detected. At the end of the analysis, each different sequence detected is an I/O phase. Figure 9 reflects how an I/O phase would be detected.



(a) The request size observed at application and POSIX-IO level is the same when the applications are written in C.

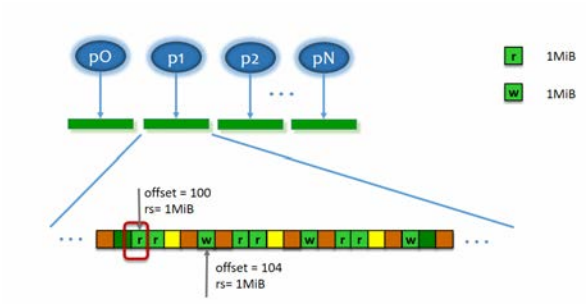


(b) The request size observed at application and POSIX-IO level is the same if FORT_BLOCKSIZE <= *rs* when the applications are written in Fortran.

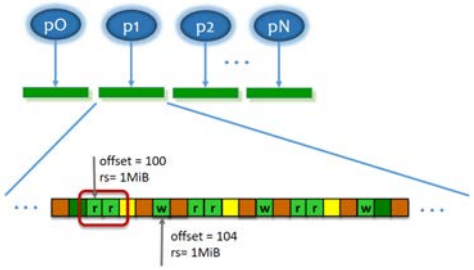
Figura 8: Influence of the environment variable value FORT_BLOCKSIZE on C and Fortran applications.

4.6. Extracting of temporal pattern

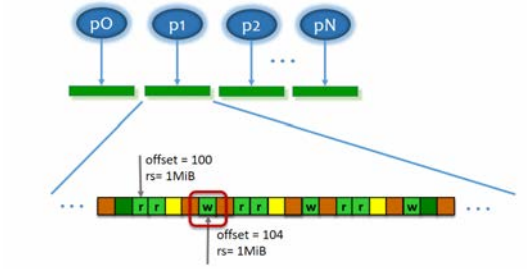
To extract the temporal pattern, the tick (for MPI events) and subtick (for POSIX-IO events) must be observed. The tick determines when a I/O phase at POSIX level begins and finishes. When a new



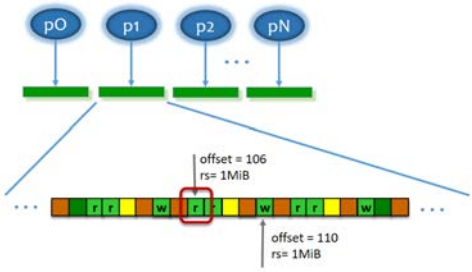
(a) First POSIX-IO operation detected.



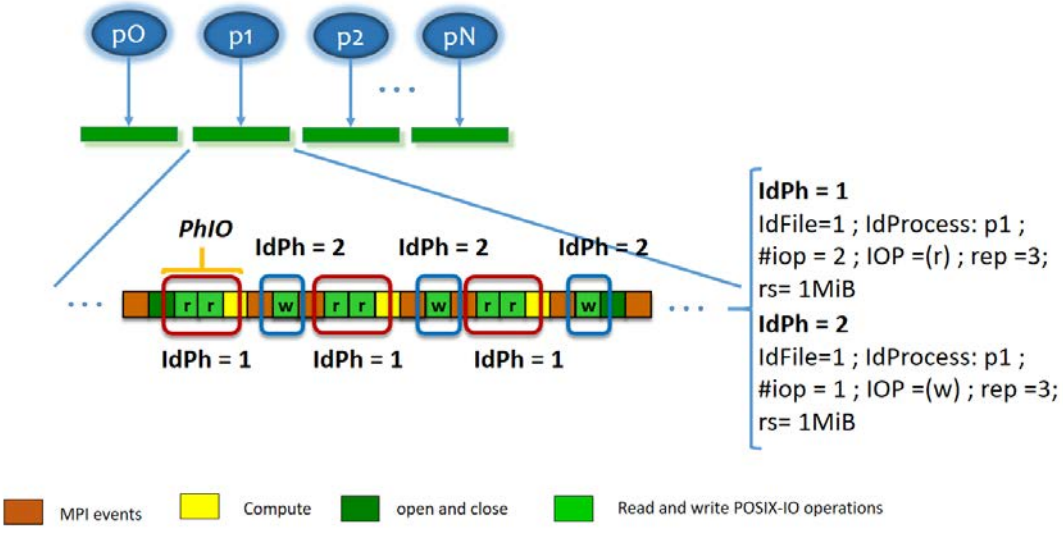
(b) Next POSIX-IO operation detected.



(c) New POSIX-IO operation detected.



(d) Detection of a POSIX-IO operation similar to 9(a). Values of offset and rs are analyzed.



(e) Detected Ph_id phases for a file_id file.

Figura 9: Detection of all I/O phases.

MPI event or MPI-IO operation is detected, the subtick is reset to zero and it is always incremented when there are consecutive POSIX-IO operations.

If the processes exchange `MPI_Sends` messages, the tick and subtick of the receptor processes must be reviewed.

4.7. Identification of significant I/O phases

Once the I/O phases are identified, it is possible to detect the significant I/O phases, those that have the most influence in the application.

In our case, the criteria for choosing the significant phases are:

- Percentage that represents the I/O time in an execution.
- File selected by the user to be analyzed.
- I/O phases with a data volume transferred higher than other I/O phases. This depends on both the number of operations and the operation size.

Finally, after a criteria is applied, a synthetic program is customized with the I/O model values to replicate the I/O behavior representative of the analyzed application. This synthetic program can be executed in another HPC system or with another I/O configuration without the need to execute the real application. For example, to evaluate the application performance obtained in another HPC system or with a new I/O configuration.

4.8. Experimental validation

In this section, the results obtained by applying the PIOM-Trace tool are shown.

Different benchmarks, such as MADbench2, have been selected to have I/O behavior models extracted and to show that the information obtained is system independent.

4.8.1. MADbench2

MADbench2 [41] 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 can be run in IO mode, in which all calculations/communications are replaced by busy-work. Running MADbench2 requires n^2 number of processors. A detailed description of the different parameters is presented in [41].

The HPC systems features used are shown in Table 2.

For this purpose, we work with the default values of the HPC systems. For this reason, the HPC system, Finisterrae 2, will have 1 Data Server (DF), while in the previous chapter, the Data Server number was 4 OSS and 12 OST.

For MADbench2 the selected I/O strategy is one file per process and the analysis is focused on POSIX mode. The MADbench2 parameter values used were IOMETHOD=POSIX, BWEX=1.0, RMOD=WMOD=1, FILETYPE=UNIQUE, NO_BIN=8 and KPIX=25; and FBLOCKSIZE is adjusted to 8 MiB for GPFS in LRZ and to 1 MiB for Lustre in Finisterrae2.

Analyzing the trace files, 5 I/O phases were identified per every file of the parallel application. In Table 5 the phase level I/O parameters are presented.

Tabla 5: MADbench2 phases using 16 MPI processes, 25KPIX and 8 NO_BIN.

Concepts	Phases for $\forall\ File_i\ with\ i \in (0..app\_np-1)$				
Ph_id	Ph1	Ph2	Ph3	Ph4	Ph5
file_id	$File_i$	$File_i$	$File_i$	$File_i$	$File_i$
Ph_processid	i	i	i	i	i
$Ph\_np(Ph\_id)$	1	1	1	1	1
rep	1	1	1	1	1
Ph_niop	8	2	12	2	8
IOP_type	w	r	(w,r)	w	r
rs	312500000				
Ph_weight	2.32GiB	596MiB	3.5GiB	596MiB	2.32GiB
offset	312500000				
disp	312500000				

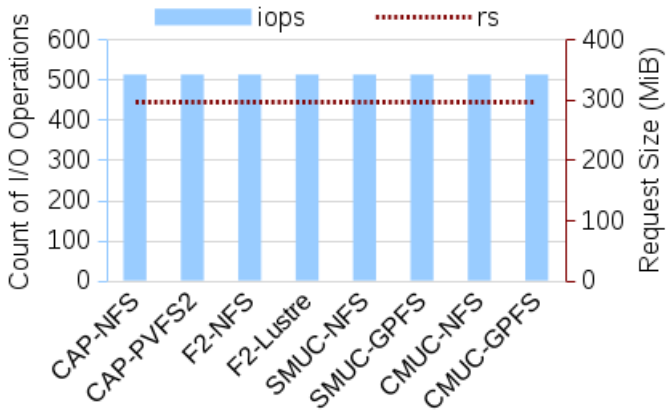


Figura 10: Madbench2 using 128 MPI processes. The total counter (*Ph_niops*) of I/O operations and the request size (*rs*) on different HPC systems and filesystems are shown.

Figure 10 presents a total of 512 POSIX operations that correspond to the `fwrite` and `fread` operations, and it is shown that the request size is 298 MiB for every operations. It can be observed that values are independents of underlying I/O system.

In Table 6 are detailed the parameters at application and file level and the values obtained using different configurations. In the four HPC systems the same values are obtained. The values shown from app_np = 25 are obtained in the SuperMUC and CoolMUC2 systems. The results correspond a different number of processes for workloads 25KPIX and 90KPIX.

Tabla 6: MADbench2 - Application features - AccessType = 1 File x Process - AccessMode = sequential - FileAccessType = W/R - file_np =1. The values from 64 proceses not are obtained in the cluster CAPITA.

	Application values			File Concepts		Phase Concepts
	app_np	app_nfiles	app_st (GiB)	$file\_size$ (Bytes)	$file\_nphase$	rs (Bytes)
$KPIX = 25$						
	16	16	37.25	2500000000	5	2500000000
	25	25	37.25	1600000000	5	1600000000
	64	64	37.25	625000000	5	625000000
	100	100	37.25	400000000	5	400000000
	400	400	37.25	100000000	5	100000000
$KPIX = 90$						
	256	256	965.60	4050000000	5	4050000000
	400	400	965.60	2592000000	5	2592000000
	576	576	965.60	1800000000	5	1800000000

The I/O phases of MADbench2 are similar using different numbers of MPI processes and different workloads. However, to define generic phases the following equations for *Ph_niop* have to be considered:

- Phase 1: *Ph_niop* = NO_BIN
- Phase 2: *Ph_niop* = 2
- Phase 3: *Ph_niop* = NO_BIN - 4
- Phase 4: *Ph_niop* = 2
- Phase 5: *Ph_niop* = NO_BIN

Even so, *rs* depends on NO_PIX and app_np. To obtain other I/O parameters at phase level, it is necessary to consider the values presented in Table 6. Values of *rs* are presented using different NO_PIX and *app_np*.

It can be observed that the I/O behavior model of MADbench2 is similar for different workloads and MPI processes.

4.9. Conclusions

Analyzing scientific applications and extracting the application I/O behavior helps to understand how an application is influenced by the I/O system and why the I/O system offers a poor performance when an application is executed. Obtaining knowledge of applications at POSIX-IO level allows identifying what impact the application has on the I/O system. Because this level is related directly with the filesystem.

To define the application I/O behavior at POSIX-IO level allows the analysis of the parallel applications that use serial I/O library and also allows an in-depth analysis of parallel applications that use parallel I/O library. It was necessary to determine how the I/O phases at POSIX level are identified in the PIOM-PX definition, where it was decided that MPI operations delimit the I/O phases at POSIX-IO level. In case that only POSIX operations exist, the identification is determined by the distance between the I/O events. This distance is the compute events generated between the POSIX events.

PIOM-PX introduces the *subtick* concept which allows the logical order of POSIX events to be established. To integrate PIOM-PX into the PIOM-MP, the *subtick* allows the dependency/relationship of POSIX events with MPI events generated by the application to be estabished.

The PIOM-PX definition and the integration into the PIOM-MP have contributed to the PIOM model. This model allows analyzing the application in two levels: POSIX and MPI. But, it is possible to select the analysis only the POSIX, MPI-IO or POSIX+MPI levels. In the experiments carried out, it is observed the utility of being able to obtain knowledge in two levels of the I/O software stack, one of them being, the POSIX interface. Because all applications cross the POSIX interface to go to filesystem.

5. Functional description

In this chapter, it is explained the design carried out to extract the PIOM model, it is presented the model extraction at POSIX level and its integration with PIOM-MP to give a global vision of both layers.

5.1. Methodology proposed

The methodology proposed to carried out the analysis, in general terms, consist of four steps (see Figure 11)

The methodology goal is to identify the characteristics that provides relevant information over the application. From this information, the significant behaviour of the application can be replicated. The values of this characteristics depend on the HPC system used to execute the parallel application. For

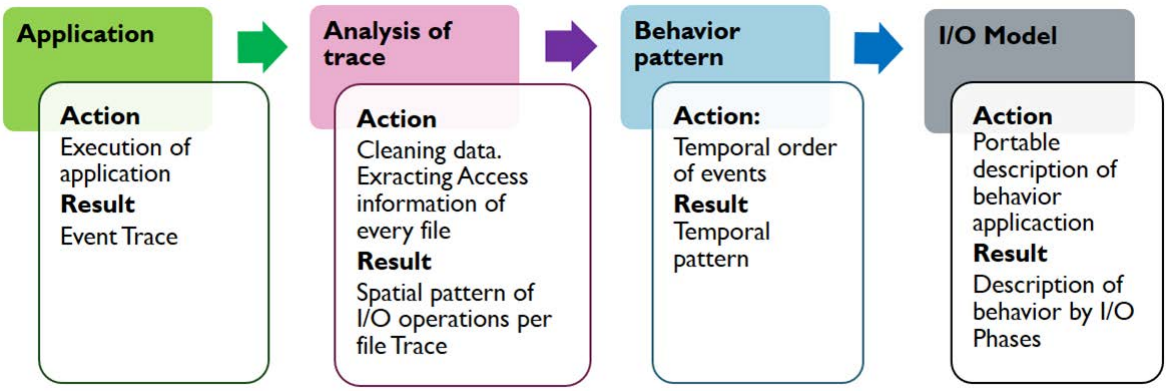


Figura 11: Methodology to provide the I/O behavior model of an parallel application.

this reason, with this methodology the relation between the values obtained in the different layer is established with the aim of obtaining relative information about the application, not relative to the system. Obtaining this information facilitates to understand why an application has o can have an I/O problem when it is executed in a specific HPC system.

In this methodology, two stages can be differentiated: the execution of the application in a specific HPC system and the post-processing offline. Figure 12 shows a detailed schema of methodology:

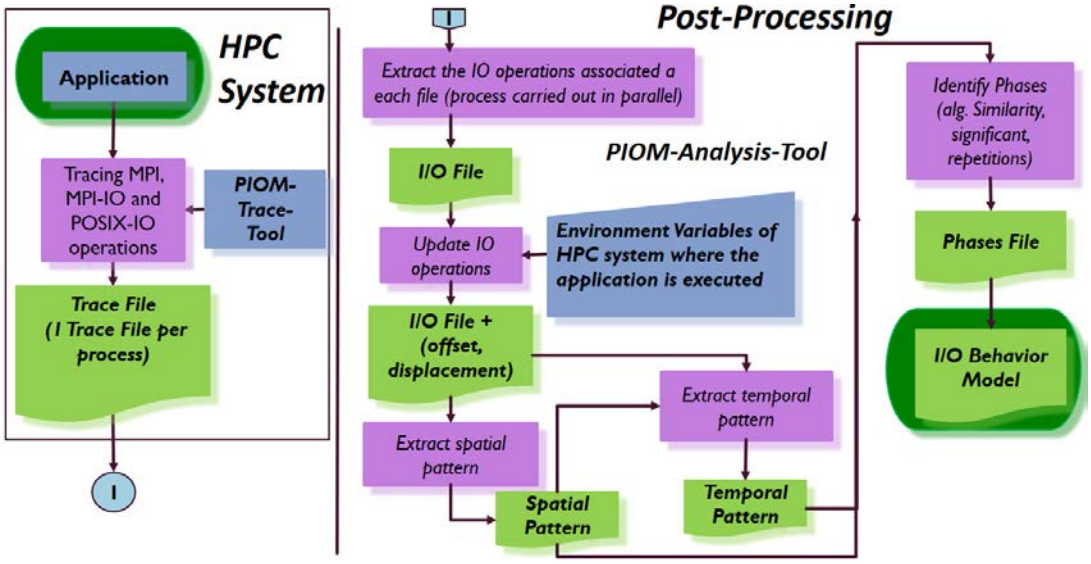


Figura 12: Module of Pre-processing (PIOM-Trace-Tool) and Post-processing (PIOM-Analysis-Tool)

Trace Application

The analysis consists of applying the methodology that allows us to identify where, when, how and why an I/O problem occurs. For this reason, the first step is to trace the application to intercept the I/O operations and to know the access pattern or the accesses patterns that the application uses. The trace offers a temporal event series which can be to process off-line. The operation types intercepted are:

- at MPI level: I/O operations of MPI-IO and communication operations MPI,
- at POSIX-IO level: I/O operations.

At I/O operation level, it is interesting to know what process (*Ph_processid*) has accessed to the file, what file (*file_id*, *file_name*) is accessed and how (*file_fileaccesstype*), what operation type (*IOP_type*) the process has made when accessing to the file, where (file position) (*offset*) has made the operation and what data volume (*rs*) is transferred when it has been carried out the operation. When the corresponding instruction is intercepted, the *IdProcess* (equivalent to *Ph_processid*), *file_id* is recorded in the trace file, *file_name*, *FileAccessType* (corresponding to the *file_fileaccesstype*), *NameOperation* (corresponding to the *IOP_type*), *offset* and *rs*. To control the logical order in which operations have been carried out, the fields *Time*, *final_compute*, *tick*, *subtick* are added. The field *TypeOperation* is added to identify the operation type, "MPI" or "POSIX-IO". The structure of the line recorded, in the trace file, is described in chapter 4, Table 4.

Two trace files are obtained per MPI process, because the information has been divided depending on the type of operations intercepted: MPI *(*. tmp_piom-mpi)* or POSIX *(*. tmp_piom-posix)*. In one, MPI communication events and MPI-IO I/O operations are stored, and in the other, POSIX-IO intercepted I/O operations are recorded.

The method to intercept the instructions, proposed in this research, does not add an *overhead* considerable because it does not interfere in the application execution. However, it must pay attention with the trace files generated because these files must be saved and they need storage resources.

Extracting I/O operations

From here on, it begins the post-processing offline, named *PIOM-Analysis*. The Figure 12 shows the modules implemented to carried out the post-processing.

The trace files are analyzed to detect the open files by the application. Afterwards, it is created a new file (*I/O File*) by every open file during the execution of application. In this new file , all I/O operations for a specific open file are registered. This new file contains information of the access patterns to the file carried out by one, several or all the processes.

Updating I/O operations

In this step, every *I/O File* is reviewed to identify the value of the model parameters. There may be trace lines that do not have value in the model parameters. Therefore, when it is carried out the trace analysis, to update the model parameters may need to analyze several I/O operations or clean up the information contained that is dependent on the system in which it was plotted. The purpose of " cleaning" the parameters is to make the application's behavior reproducible and to have a portable model.

Spatial pattern description

For extracting the application spatial pattern. the application I/O accesses are analyzed, the I/O operations performed by the application in the file and where in the file these operations are accessing are identified. With this aim, an analysis of each *I/O File* is performed.

At this point, a similarity algorithm is applied to identify when operations are considered similar to include them or not in the same I/O phase and the number of repetitions of each phase is updated to replicate the movement of the same amount of information.

Extracting temporal pattern

To detect the temporal pattern, the *tick* and *subtick* are defined as unities of logical time. The *tick* identifies the order of MPI communication events and the MPI-IO operations, while the *subtick* identifies the order of POSIX-IO operations. If all processes write in one or more share files, it must be detected the relation between all operations carried out by all processes to determine the current logical order. This order helps to detect dependencies between all processes.

From the spatial and temporal pattern, the consecutive operations are analyzed to detect the I/O phases. An I/O phase is a sequence with the same consecutive operations, with $offset$ and rs similar and in the same apparition order. Also, it is count how many times the same I/O phase appears in the file (rep).

5.2. Experimental Results.

In this point, the methodology is applied for different benchmarks such as BT-IO benchmark [42] and S3D-IO [43]. Following, the experiment carried out with BT-IO benchmark to discover the application I/O behavior is shown.

BT-IO is a benchmark generated from the block-Tridiagonal (BT) problem, which is part of a parallel benchmark that belongs to the NPB-MPI class developed by NASA Advanced Supercomputing Division. BT in MPI presents a decomposition called diagonal multipartition in a three-dimensional

matrix through a square number of processes. BT-IO is an extension of BT in which data is stored on disk.

It has been selected BT-IO to show the temporal pattern considering the tick and subtick concepts. Besides, as BT-IO has computing and communication events, the identified I/O phases from these events are shown. BT-IO is implemented in Fortran and allows the influence of Fortran I/O library in the request size at the POSIX-IO level to be evaluated. It is selected the FULL subtype because it implements the I/O with collectives operations, derived data type, `MPI_File_view` and `MPI_Info` to enabled the collective technique in the logical of application. BT-IO has been executed in the supercomputers Finisterrae2 and SuperMUC. The supercomputers features are described in the Chapter 3, Table 2.

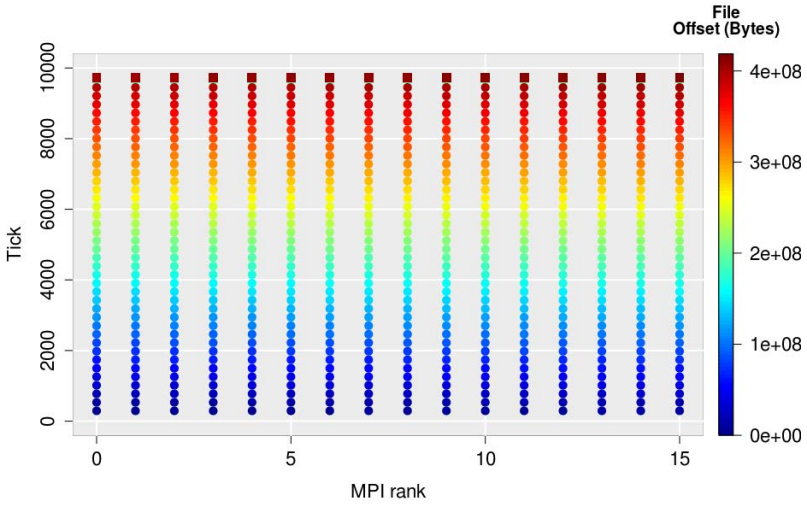
Tabla 7: PIOM-PX parameters for the BT-IO benchmark subtype FULL

Identifier	**Class A**	**Class B**	**Class C**
app_np	16	16	36
app_nfiles	1	1	1
app_st	400 MiB	1.6 GiB	6.4 GiB
File			
file_name	btio.full.out	btio.full.out	btio.full.out
file_size	400 MiB	1.6 GiB	6.4 GiB
file_accessmode	Strided	Strided	Strided
file_fileaccesstype	W/R	W/R	W/R
file_accesstype	Shared	Shared	Shared
file_nphase	41	41	41
file_np at MPI-IO level	16	16	36
file_np at POSIX-IO level	1	1	3

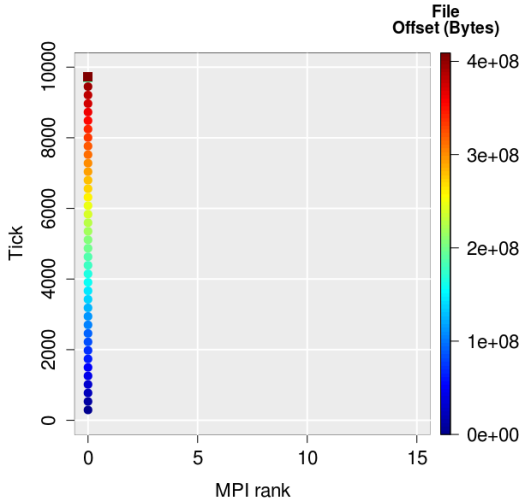
Figure 13 shows the I/O phases detected at MPI-IO (Fig.13(a)) and POSIX-IO(Fig.13(b)) level. PIOM-PX parameters are described in Table 7 for Classes A, B and C at application and file level. The Table 7 shows the values obtained to do the analysis and it has been observed that 41 I/O phases have been identified in a single shared file for a strided access mode.

In Figure 13(a) each bullet line (y-axis) represents an I/O phase composed of $file\_np$ write operations. The red square represents Phase 41, which is composed of $40 \times file\_np$ read operations. The operation size is similar for read and write operations. At MPI-IO level, the number of MPI processes per I/O phase correspond to the $file\_np$.

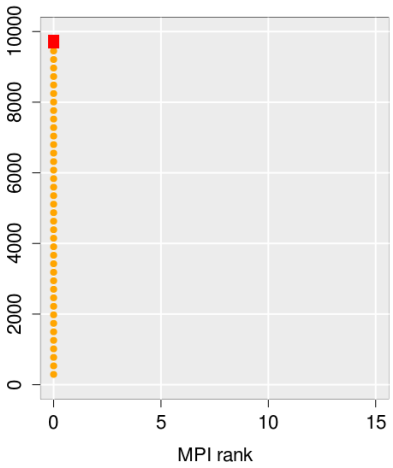
In Figure 13(b), the effect of the collective buffering techniques at POSIX level can be observed, where only process 0 performs I/O operations. In this layer, the number of processes per I/O phase is equal to the number of compute nodes utilized for running the application.



(a) File offset at MPI level by using PIOM-MP



(b) File offset at POSIX-IO level



(c) Phase Weight at POSIX-IO level

Figura 13: BT-IO subtype FULL, Class A using 16 MPI processes for a strided pattern. The bullets (smaller circles) correspond to write operations and the shaded squares to read operations. Each first forty I/O phases (circles) are composed of 1 MPI write operation and Phase 41 of 40 read operations. At MPI-IO level, the weight of the Phase 1 to Phase 40 is 655360 Bytes $\times$ $file\_np$ for each one and for Phase 41 it is 10 MiB $\times$ np $\times$ 40. At POSIX-IO level, the colored scale in Fig.13(c) shows the weight for Phase 1 to Phase 40, which is 10 MiB and the Phase 41 weight is 10 MiB $\times$40.

5.3. Conclusions

Adding PIOM-PX model to PIOM allows analyzing the application I/O behavior at POSIX-IO level. Also, this integration facilitates to observe the different treatments that the application data suffer when the data goes through different layers of I/O software stack. Observing this behavior helps to

understand what information the system is receiving in the different layers.

6. Use Case: PIOM in HPC Cluster in Cloud platform

The appearance of Cloud platform has caused the scientific community interest, focused on the HPC systems, to analyze and evaluate if these platforms can be an alternative to the standard cluster. That interest consists of the benefits that offer Cloud because the users can acquire and release resources on order and, can configure and customize their own virtual cluster [44].

In a lot of research areas exist the necessity of having a test environment to evaluate the concept tests, validate ideas, check implementations without affecting the production environment. In many cases, the scientific community has simple, basic and enclosed test environments for carryng out the first estimation, validation and/or testing.

For the researchers focused on the scientific parallel applications that use parallel I/O, this platform offers a lot of opportunities because the Cloud environments can be used as test environment (*Test-Bed*). In contrast to the traditional cluster, in Cloud, the user can create, configure the I/O system and install different filesystems without affecting the work of other users or the performance of other applications. This represents an advantage over the I/O system of HPC systems.

These environment types offer a lot of resources that the user must select to create a virtual cluster in Cloud (VCC). However, there are also some disadvantages, one of which would be that the user has to pay attention to the selected resources and used time. Because the user has to pay by the select resources number, type, used time and the data volume transferred. Thus, an important challenge is the efficient use of resources. Another challenge is the complexity to configure the system due to the number of parameters to consider. Another factor that the user must take into account is the variability because this factor affects the I/O bandwith obtained for an application.

In this point, it is proposed a methodology to guide the user in the evaluation of virtual clusters that will be configured taking into account the application I/O requirements. Besides, this methodology, in general, reduce the evaluation cost because it does not necessarily the real application is executed. Both cost and relation cost-performance for the virtual cluster can help the user to make the decision over the select the resources in the Cloud platform. Although the user, in that decision, has also to take into account the performance variability obtained.

6.1. Related Work

Different research groups have focused on the evaluation of *Cloud* platform to check your suitability for the HPC [45]. The interest in the I/O system has increased because this system must configure and use efficiently to control the cost per resource used. Next, it is described some research works carried out in the Cloud environment:

R.Expósito et al. [45] evaluate the performance of storage system I/O in the cluster of Amazon EC2 and the new I/O instances provided to discover if this platform is appropriate for the applications with I/O intensive. They evaluate the ephemeral disk (local and temporal storage) and EBS (Elastic Block Storage) in different levels.

Juve et al. [46] evaluate the performance and cost of three real scientific applications in Amazon EC2, using different storage: ephemeral (local) and S3. Besides, they use different filesystems: NFS and PVFS2. Liu et al. [44] evaluate two applications with different I/O configurations using NFS and PVFS and the ephemeral disk as storage. All use the instances of Cluster computing (CC).

The Cloud platform introduces another abstract level, the virtualization, that increases the system complexity. For this reason, Noorshams et al. [47] present a I/O performance analysis focused on the virtual storage systems based in queue theory.

Sivathanu et al. [48] present a study to measure the I/O performance. They are focused on the influence of the supply of disk, the interference of positioning the workload in the I/O performance and the implications of virtualization with different workload types.

6.2. Methodology

The methodology described following (see Figure 14) allows carrying out to evaluate of VCC since the application I/O behavior model. This evaluation and comparison allows rejecting the instances which can't cover minimally the application I/O requirements. Following, it is explained with detail every step.

6.2.1. Characterization of virtual cluster

The first step is select an instance and to characterize it. Amazon recommends measuring the instance performance. The IOzone benchmark [49] allows knowing the I/O bandwith that provides a node. IOzone is used as a tool to evaluate the filesystems and provides the node dynamic features. This benchmark generates and evaluates several file access operations.

However, PIOM provides knowledge about the application. In particular, it is obtained the *rs* size that uses the application. This information allows customizing IOzone to evaluate the instance with the specific *rs* values of application and to minimize the evaluation time of instances.

IOzone analyzes the write/rewrite and read/reread operations for different access modes of a file. On this way, it is obtained the average bandwidth for the *rs* sizes of the application. Also, it can establish a *rs* rank to evaluate better the instance. In this step, the instances can be discarded if they provide a low bandwidth.

Besides, IOzone can calculate the maximum value for a global filesystem of the VCC. In this case, the maximum value is the addition of obtained values in every I/O node of the global filesystem.

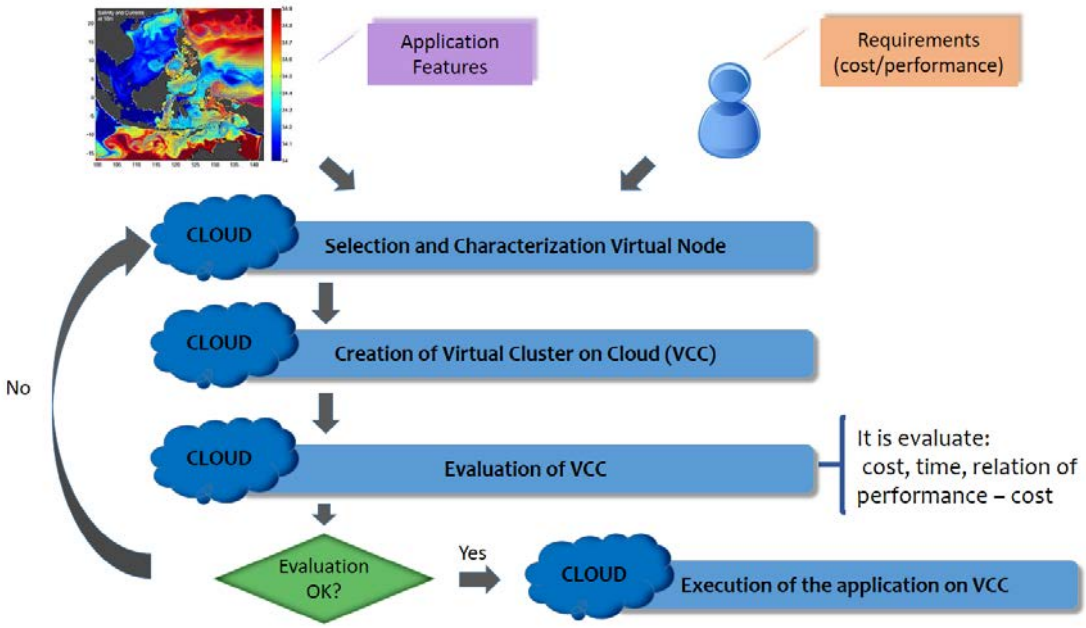


Figura 14: Methodology for the performance evaluation and cost of the I/O in VCC.

6.2.2. Creation and configuration of Virtual Cluster

Once selected the instance, to create a Virtual Cluster in Cloud (VCC) must be taken into account the components shown in Table 8.

Tabla 8: Components of Virtual Cluster (VC)

Parameters	**Description**
Instance type (*)	Number of cores, processor capacity, RAM memory size.
Number of instances(*)	Number of nodes (include compute nodes and I/O nodes)
Number of I/O nodes (-)	Data servers and metadata server
Storage type(+)	Temporal and/or persistent
Device type temporal(+)	HDD or SSD
Device type persistent(+)	HDD or SSD
Capacity of temporal storage(+)	As minimum the storage capacity required (app_st)
Capacity of persistent storage(-)	
Network performance (+)	Low, Moderate, High, Unknown
I/O library (-)	MPI, NetCDF, pnetcdf, HDF5.
Local filesystem (+)	Filesystem Linux ext3, ext4, xfs, etc.
Global filesystem (-)	Parallel, Distributed or Network Filesystems
Stripe size (-)	Related by the parallel file system

These components are the components provided by the *Amazon Web Services (AWS)* platform [50].

Moreover, for the StarCluster tool [51] is created a plugin what allows unfold the PVFS2 filesystem in a fast and easy way. Also, it is possible to select where to ubicate the I/O nodes. It has been considered different architectures:

- Shared nodes for the compute and I/O nodes,
- Independent nodes for the compute and I/O nodes.

The components presented in Table 6.1 are classified in three categories: the components can be selected by the user (*), must configure by user manually (-) and the components can not be changed by the user because are default values depending on the instance type selected (+).

To select the components of I/O system of a VCC and accomplish the user requirements, it has to take into account the next considerations:

- *Instance type*: it depends on cost that user can pay.
- *Instances*: the instances number is a parameter determinate by the processes number required by the application I/O behavior model and compute.
- *Storage type*: can be temporary (without cost) and/or persistent (cost per GB/month). Usually, the scientific parallel applications use the temporary storage to save the input files and the files generate during the execution. Only, the data that need to postprocessing are saved in the persistent storage.
- *Storage capacity*: it must guarantee as to the minimum the storage capacity required by the application (see expression 3.2).
- *Networking performance*: can be higher, lower, moderate and unknown. It is associated with the instance type. The network workload depends on the processes number if the application is strongly or weakly coupled, the request size and the I/O operations number.
- *I/O library*: it depends on the I/O library used by the application. In our case, it is only used MPI-IO and POSIX-IO because it is extracted the I/O behavior of MPI applications.
- *Filesystem type*: it depends if the access type is shared or unique (a file per process). If the access type is shared so the filesystem must be a global file system such as NFS, PVFS2 or Lustre. In contrast, when the access type is unique, it is possible to use the local file system to take advantage of the local disks connected a compute nodes.

The basic software of VCC for every compute node depends the machine selected image. Although, as well as to the standard cluster, the operative system most used is Linux, specially to the I/O Software Stack. The software to the parallel processing both MPI and the global file system must be installed and configured by the user. The cost-performance will be a restriction to consider of a VCC. The components cited before will allow making a decision considering the application requirements.

6.2.3. Evaluation of performance in the virtual cluster for the application I/O behavior model

Having the application I/O behavior model and identified the phases more significant, it is customized the IOR benchmark to carried out the evaluation of VC with every phases. With IOR is achieved to replicate the phases less favorable in the VC to observe the I/O configuration suitable to execute the application. This observation is based in the obtained performance. The Table 9 shows the input parameters for IOR based on the phases of application I/O behavior model. The output of this process is the transfer rate named $BW_{CH}(phase[i])$ and expressed in MB/s, and the I/O time for the application I/O behavior model.

Tabla 9: Input parameters for IOR based on the application I/O model.

Access Mode	Access Type(AT)	Param. for AT	Number of processes	Number of segment	Block size	Transfer size
Strided	SHARED		np=np(IdPh)	-s=rep	-b=rs(IdPh)	-t=rs(IdPh)
Sequential	SHARED		np=np(IdPh)	-s=1	-b=weight(IdPh)	-t=rs(IdPh)

6.2.4. Evaluation the cost of Virtual Clusters

To predict the using cost of I/O in a VCC is used the performance obtained of I/O behavior model with IOR. The total cost for a Virtual Cluster depends of instance type, instances number and used time of theses instances. It is composed by a variable cost ($cost\_var$) and a fix cost ($cost\_fix$). The variable cost would correspond to the instances number used and the use time. While, the fixed cost would correspond to the instance type selected to create the VCC.

To calculate the cost, the expression used is 3.

$$cost\_tot = cost\_var + cost\_fix \tag{3}$$

As the proposed model is based on the I/O phase concept, the cost is also calculated by phases, allowing carry out a fast evaluation of the I/O cost.

The selected metric for the IOR is the transfer rate (BW_{CH}) and it is expressed in MB/s.

The variable cost estimated for the application I/O behavior model is calculated with expression 4. This variable is proportional to the used time of I/O and the quantity of transfer data. In Cloud, it is billed by used hours.

$$cost\_var = \sum_{i=1}^{phases} cost(phase[i]) \times num\_inst \tag{4}$$

Where *num_inst* is the instances number used during the execution and $cost(phase[i])$ is calculated by the expression 5.

$$cost(phase[i]) = cost(phase\_i) + [EBS] \tag{5}$$

$$cost(phase\_i) = \frac{weight(phase[i])}{BW_{(CH)}(phase[i])} / 3600 \times cost_{inst} \tag{6}$$

$cost_{inst}$ is the cost per hour for every instance type *inst* and EBS represents the permanent storage by mes and, it is a fixed value. EBS is an optional value depending if it is used or not as storage. If this storage type is used, it must be considerated fixed cost. Using EBS is optional, but if the application can not execute in the temporal storage, it can add permanent storage to increment the storage capacity or simply to have permanent storage.

6.2.5. Comparison of performance-cost relation for a Virtual Cluster

Both performance and cost of Virtual cluster is calculated to help the user the decision making. To establish the relation between the performance and the cost, it is used the expression 7.

$$perf\_cost_{ci} = \frac{performance_{ci}}{cost_{ci}} \tag{7}$$

where *ci* represents a Virtual Cluster and $i \in \{1..TotalVirtualClusters\}$. Where $TotalVirtualClusters$ is the total number of VCCs configurations to analyze.

Also, it can be calculate the relation time-cost of different virtual cluster with the expression 8.

$$perf\_cost_{ci} = \frac{Time_{ci}}{cost_{ci}} \tag{8}$$

where $Time$ is expressed in sec., *ci* represents a VCC, and $i \in \{1..TotalVirtualClusters\}$.

To compare the cluster *ck* and the cluster cj,it can be said that *ck* has a performance-cost relation more efficient than cj, if $perf\_cost_{ck}$ is highest than $perf\_cost_{cj}$.

6.3. Experimental Results

It has been carried out several experiments to check that Cloud platform can be used to execute the HPC applications; in particular, the MPI scientific parallel applications. Following, it is shown the case analysis for an application.

6.3.1. Analysis of ABySS application using Cloud like Test-Bed

Taking into account the ABySS application, it was tested if a global filesystem would be able to impact positively in the application performance. Because a user would was interested in to incorporate a global file system in its HPC cluster. It was necessary to resort to the Cloud platform to can modify the filesystem.

Taking into account the ABySS application, it was tested if a global filesystem would be able to impact positively in the application performance. Because a user would was interested in to incorporate a global file system in its HPC cluster. It was necessary to resort to the Cloud platform to can modify the filesystem. With intent to recreate, as much as possible, the user HPC cluster was create two VCC with the features shown in the Table 10.

Tabla 10: Descriptive Features of Virtual Cluster (VCC) in Amazon EC2.

I/O components	**VCC 1**	**VCC 2**
Instance	c3.2xlarge	c3.2xlarge
Number of Instances	6	6
Temporal Storage	Ephemeral	Ephemeral
Persistent Storage	EBS	EBS
Temporal Device	SSD	SSD
Persistent Device	SSD(GP 192/3000)	SSD(GP 300/3000)
Capacity of Persistent Storage	100GB	64GB
File system Local	ext4	ext4
File system Global	NFS	PVFS2 (2.8.8)
Parallel Storage Capacity	-	400GB
Number of data servers	-	5
Number of Metadata Server	-	1
Stripe Size	-	64KB
MPI library	mpich-3.2	mpich-3.2

It is used our plugin for the StarCluster tool, which allows unfolding the PVFS2 filesystem in a fast and automatic way. Due to the cost and the time necessary, to explore all alternatives that offer the Cloud platform is a prohibitive option. Thus, the Cloud' resources are only selected to evaluate the I/O behavior model of ABySS-P.

Figure 6.5 presents the bandwidth obtained for the IOR customized with the I/O model for ABySS-P in NFS and PVFS2. With NFS is obtained a bandwidth of 117 MiB/sec, while with PVFS2 is obtained 48 MiB/sec as a maximum for the analyzed configurations.

As a result of these experiments, it can conclude that a filesystem likes PVFS2 is not favorable for the I/O pattern of ABySS-P. The poor performance obtained with PVFS2 is related to the little request

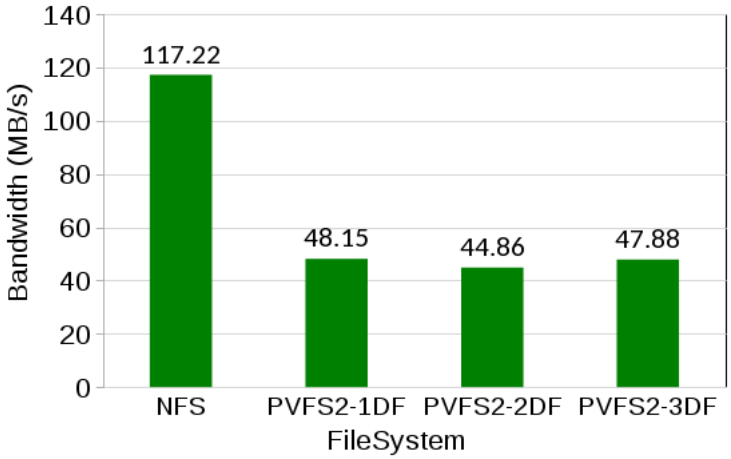


Figura 15: Bandwith obtained on VCC1 and VCC2 using the bechmark IOR customized for the ABYSS-P's I/O model.

size of read operations. In ABySS-P only exists two processes that their function is to read. In fact, there are two MPI processes that read two input files and, then send the data to other processes. This is an obstacle to the scalability.

7. Conclusions and Future Work

In this research work has been proposed a model to represent the I/O behavior of parallel MPI applications. This I/O model is organized in phases which allow to replicate the I/O behavior more significant in other HPC systems. As some parallel applications use serial I/O library, the analysis is focused on the POSIX level of I/O Software Stack. It has been described the principal parameters to define the I/O behavior model and these parameters are system independent because one goal is to obtain a portable model.

This work complements an I/O behavior model previous defined by our research group, renamed as PIOM-MP. How some parallel applications use serial I/O library, the analysis is focused on the POSIX layer of I/O Software Stack. It has been described the principal parameters to define the I/O behavior model, named PIOM-PX. These parameters are system independent because the I/O model must be independent to replicate the I/O behavior in other HPC systems.

The contribution of this work is to included the subtick concept, a logical counter which allows controlling the logical order of POSIX-IO events and establish the relation/dependency with the MPI events. Another contribution is that POSIX-PX allows to analyze the parallel applications that use serial I/O library. Besides, the integration the PIOM-PX model to the PIOM-MP model allows to deepen in the knowledge to the I/O behavior for the parallel applications that use the parallel I/O. Because, it was observed that the information provides in the MPI-IO level differs to the information

obtained at the POSIX-IO level. Observing a parallel application since two points of view of I/O software stack allows to check that the data treatment in the stack layers are different, and it facilitates understand why it can be generate a bottleneck or it can have a poor performance.

7.1. Future Work

A future line of work, it will be to analyze the scientific parallel applications written in other programming languages such as Python to extract their I/O behavior. Selecting this applications type, it wants to check if the environment variables have to take into account how the case of applications written in Fortran.

Another future line of work would be to analyze the I/O behavior at filesystem level, to observe what data volume is transferred to the physical drives and how is transferred. This analysis would allow to identify what module of the file system is more relevant and how the filesystem manages the transferred data by other interface.

At Cloud level, a future line of work would be to evaluate the scientific parallel applications with different filesystem such as Lustre and GPFS, to examine the file system more appropriate for a specific application. Or to evaluate what changes the user has to make in the filesystem of standard HPC system to achieve an appropriate performance.

1

Introducción

"A supercomputer is a device for converting a CPU-bound problem into an I/O bound problem"

- Ken Batcher. 2011

Actualmente, las aplicaciones científicas paralelas se procesan en sistemas de Computadores de Altas Prestaciones (HPC - *High Performance Computing* -) sin grandes problemas. Estas aplicaciones generan grandes volúmenes de datos para su posterior análisis y visualización, y se espera que este volumen de datos vaya en aumento. Cuando las aplicaciones solicitan datos al sistema de Entrada/Salida (E/S) o necesitan guardar datos en el sistema de almacenamiento se generan tiempos de espera en el sistema de cómputo. Estos tiempos de espera son variables y dependen de diferentes factores, uno de los objetivos del diseño de los sistemas de E/S es minimizar estos tiempos de espera. Esto se debe a que las peticiones recibidas en el sistema de E/S por parte del sistema de procesamiento están influenciadas por el tiempo en el que son atendidas. El sistema de E/S, sistema en el que se centra esta investigación, de un sistema HPC es un sistema complejo compuesto por un componente hardware (con diferentes elementos) y un componente software (con diferentes capas) interrelacionados entre sí. Estos componentes a su vez se pueden configurar permitiendo que los sistemas HPC tengan diferentes configuraciones del sistema de E/S.

Un primer aspecto a considerar sería la velocidad de almacenamiento, ya que ésta es menor que la velocidad de procesamiento. La diferencia entre estas dos velocidades se conoce como *"gap"*. Para solucionar este problema de los dispositivos de almacenamiento lentos, se han realizado diferentes propuestas de cómo debería ser la arquitectura de E/S y cómo se debería organizar la pila de software de la E/S (*I/O Stack Software*).

Otro aspecto a considerar serían las aplicaciones que se ejecutan en un determinado sistema HPC. Las aplicaciones pueden tener unos requerimientos y comportamientos o patrones de acceso diferentes. Es decir, una aplicación puede tener características diferentes del resto de aplicaciones debido a que factores como la cantidad de datos que transfiere en cada acceso de E/S, cuándo se produce ese acceso y cómo se produce, pueden provocar también tiempos de espera. Dicho de otro modo, el patrón de acceso que utiliza la aplicación o el comportamiento que tiene la aplicación también influye en que hayan tiempos de espera. Así, el sistema de E/S se ha convertido en uno de los factores que limitan la velocidad o el tiempo de ejecución de algunas aplicaciones científicas paralelas [1], cuando las operaciones de E/S representan un porcentaje significativo del tiempo de ejecución de este tipo de aplicaciones [2]. Para mejorar este aspecto se han propuesto nuevas técnicas como las colectivas en MPI-IO.

Teniendo en cuenta estas consideraciones, nos preguntamos, ¿por qué una aplicación cuando se ejecuta en un determinado sistema HPC se generan cuellos de botella y/o el rendimiento de la aplicación es menor del esperado? Y ¿porqué cuando esa misma aplicación se ejecuta en otro sistema HPC no se generan cuellos de botella y/o el rendimiento es similar al esperado?

Los sistemas HPC son diferentes, sus componentes son diferentes. En otras palabras, los sistemas de E/S o las configuraciones del sistema de E/S son diferentes y, en concreto, los dispositivos físicos de E/S. Pero, si los dispositivos físicos son iguales, ¿a qué se deben los cuellos de botella?

¿Por qué en el mismo sistema HPC, donde una aplicación cuando se ejecuta tiene grandes esperas,

se obtiene una respuesta lenta del sistema de E/S y otras aplicaciones se ejecutan sin que se generen estos tiempos de espera?

En realidad, existen varios factores que influyen en la generación de los cuellos de botella tales como el sistema de interconexión, las aplicaciones que se ejecutan y la configuración del sistema de E/S en el que se esté ejecutando [3]. Por esta razón, y teniendo en cuenta las preguntas planteadas anteriormente se podría decir que la combinación de la configuración (características) del sistema de E/S junto con las características de la aplicación, algunas veces, es lo que provoca que se generen cuellos de botella y/o el rendimiento de la aplicación sea pobre.

Pero, ¿cómo puede afectar una aplicación *(software)* al sistema de E/S *(hardware)*?¿Qué relación/-dependencia existe entre la aplicación y la configuración del sistema de E/S de un sistema HPC? Al ejecutar las aplicaciones paralelas en sistemas HPC se espera que no se generen cuellos de botella y/o el rendimiento obtenido sea similar al que ofrece el sistema HPC en vacio. ¿Cómo se podría averiguar que configuración de E/S de un sistema HPC es conveniente para una determinada aplicación? Se podría pensar que la solución sería personalizar el sistema HPC a una determinada aplicación. Sin embargo, esta solución no es viable porque el sistema de E/S de un sistema HPC es compartido por diferentes aplicaciones, y modificaciones realizadas para mejorar una determinada aplicación pueden afectar negativamente al rendimiento de otra aplicación o a la carga de trabajo del sistema. Otra solución posible sería tener un sistema HPC dedicado para cada aplicación. Aunque esta solución no es factible cuando se habla de sistemas HPC tradicionales (no virtualizados).

En los sistemas HPC tradicionales se ejecutan diferentes aplicaciones científicas cuyas características pueden ser diferentes. Cuando la carga de trabajo de un sistema HPC no es conocida, la tarea de sintonizar ese sistema HPC al conjunto de aplicaciones es complicado. Además, el inconveniente de los sistemas HPC tradicionales es que sólo permiten modificar algunos parámetros a nivel de usuario para intentar obtener una configuración de E/S lo más favorable a una determinada aplicación. Por esta razón, es importante analizar que ocurre cuando una aplicación se ejecuta en un determinado sistema HPC.

Para realizar la evaluación o experimentar con estos sistemas es importante disponer de sistemas privados y dedicados, via simulación o creándolo en *Cloud*, para instalar diferentes componentes como los sistemas de ficheros. Así, un usuario puede evaluar y analizar la aplicación con diferentes configuraciones de E/S. Pero, para encontrar la configuración de E/S más adecuada a una aplicación ¿se ha de ejecutar la aplicación en todas las configuraciones de E/S habidas y por haber?¿No hay ninguna manera de acotar el número de posibles configuraciones a evaluar? Y si, la aplicación dura horas ¿se ha de ejecutar toda la aplicación para detectar si una determinada configuración es conveniente para esa aplicación?

1.1. Hipótesis y retos claves

Nuestro interés se centra en el sistema de E/S de los sistemas HPC. Principalmente, en analizar el comportamiento de E/S de las aplicaciones paralelas científicas para entender la relación/dependencia que se establece cuando esas aplicaciones se ejecutan en un sistema HPC con una configuración de E/S determinada. La realización de nuestro análisis no requiere la modificación del código fuente ni la recompilación de los componentes de la pila de E/S.

Aunque existe una dependencia entre aplicación y el sistema de E/S, el comportamiento funcional de una aplicación paralela siempre es el mismo y este es independiente del sistema HPC donde se ejecute. Sin embargo, cuando una aplicación paralela es ejecutada en un sistema HPC, a nivel de prestaciones, se ha observado que la aplicación influye en el sistema y, el sistema también influye en la aplicación.

Por eso, es importante conocer como la aplicación accede al sistema de E/S, en qué orden se realizan las operaciones de E/S cuando se accede al sistema de almacenamiento y cómo es tratada la aplicación en el sistema de E/S. Ya que uno de los factores que hace que se obtenga un rendimiento pobre de una aplicación HPC es el patrón de acceso que utiliza la aplicación, a parte de los componentes de la red de interconexión y la configuración del sistema de E/S [3].

La importancia de conocer a priori los requerimientos a nivel de E/S que tiene una aplicación se hace más evidente con la aparición de la plataforma Cloud [4]. Debido al coste que se ha de tener en cuenta por el número de recursos utilizados, el tiempo de utilización y el volumen de datos transferidos. Para realizar un uso de los recursos de forma eficiente, es importante conocer los requerimientos de la aplicación, porque el usuario crea, configura y gestiona un Clúster Virtual (VCC – *Virtual Cluster on Cloud* -).

Antes de realizar el análisis, nos planteamos una serie de cuestiones que han sido los retos a los que nos hemos enfrentado durante la investigación:

- a nivel de instrumentación: ¿A qué nivel de la pila de Software de E/S se debe realizar la instrumentación?¿Qué información se debe extraer de la aplicación?¿Qué se debe medir? ¿Cuándo se debe realizar la instrumentación? ¿Cuánto overhead introducirá esta instrumentación en la ejecución de la aplicación?

- a nivel de análisis:¿Cómo establecer la relación entre los datos obtenidos de los diferentes niveles analizados? ¿Se puede extraer más información procesable a partir de encontrar la correspondencía entre los datos?¿Cómo detectar la causa principal que genera los problemas de rendimiento?

El propósito de resolver estos retos es intentar desarrollar mejores prácticas y sintonizar aplicaciones, mejorar el software del sistema y, diseñar y adquirir mejores sistemas.

1.2. Objetivos

El objetivo de este trabajo de investigación es proponer una metodología para obtener información relevante sobre el comportamiento de aplicaciones científicas paralelas basadas en paso de mensajes (MPI) para reducir su impacto sobre el sistema de Entrada/Salida (E/S). Para ello se propone abordar el tema de la caracterización de la E/S de una aplicación paralela con el objetivo:

- conocer el comportamiento de la aplicación, obtener un modelo de comportamiento que permita replicar el comportamiento. El nivel de descripción debe permitir realizar el análisis y la evaluación de sistemas con el objetivo de mejorar el rendimiento de la aplicación y mitigar los cuellos de botella generados por la E/S,
- seleccionar los recursos considerando criterios de eficiencia en la plataforma Cloud.

Por lo tanto, nuestro principal objetivo, al realizar esta investigación, ha sido

definir un modelo que permita describir el comportamiento intrínseco de las operaciones de E/S de las aplicaciones científicas paralelas para sistemas HPC. Dicho modelo debe ser independiente del sistema en el que se ejecuta (portable), y debe permitir representar y replicar el comportamiento de la aplicación tanto en clúster físico como en un clúster virtual.

Para conseguir este objetivo se establecieron los siguientes objetivos específicos:

- seleccionar los parámetros que formarán parte del modelo y que describen el comportamiento de la E/S de una aplicación, sin que estos dependan de la configuración del sistema de E/S en el que ha sido ejecutado,
- identificar las fases de E/S significativas de la aplicación que permiten analizar el comportamiento de la E/S en diferentes sistemas,
- replicar el comportamiento de la E/S de una aplicación en diferentes sistemas,
- establecer una metodología que permita utilizar este modelo para seleccionar los parámetros de configuración de un Cloud.

1.3. Contribución

El análisis del comportamiento de E/S de una aplicación paralela MPI puede ser usado para entender la relación/interacción con el sistema de E/S. Este conocimiento permitirá mejorar algunos

aspectos de la aplicación como el rendimiento, la eficiencia; y mitigar la generación de cuellos de botella por parte del sistema de E/S.

Este trabajo aporta a la comunidad científica un nuevo modelo para analizar y entender el comportamiento de E/S de una aplicación, seleccionando los accesos o fases significativas de la aplicación cuando accede al sistema de almacenamiento. Este modelo permite extraer conocimiento sobre la aplicación completando la información del comportamiento a nivel MPI-IO con información a nivel de POSIX-IO, facilitando la observación de lo que está ocurriendo en dos niveles y permitiendo establecer relaciones sobre lo que está pasando en cada nivel.

Además, en este trabajo de investigación se ha comprobado que la plataforma Cloud puede ser utilizado como plataforma de evaluación de diferentes configuraciones de E/S, incluyendo los sistemas de ficheros. Para Cloud se aporta una metodología que pretende facilitar a los usuarios la creación de un sistema HPC en Cloud.

1.4. Estructura de la memoria

En esta sección se describe como está organizada el resto de la memoria.

En el capítulo 2, *Sistema de E/S para Computadores de Altas Prestaciones*, se explican conceptos de E/S y se presentan los trabajos relacionados.

En el capítulo 3, *PIOM-PX: Un modelo del comportamiento de la E/S de aplicaciones paralelas a nivel de POSIX-IO*, se describen las características del modelo definido para representar el comportamiento de la E/S de una aplicación paralela a nivel de POSIX y su integración con el modelo de comportamiento a nivel de MPI. Se muestran los experimentos realizados para comprobar que con el modelo definido se obtienen las características principales de la aplicación.

En el capítulo 4, *PIOM a nivel de POSIX-IO*, se describe el framework y la herramienta para interceptar las operaciones de E/S a nivel de POSIX-IO. Para facilitar la compresión de la tesis, en este capítulo se han añadido los experimentos realizados para la validación experimental del modelo de comportamiento y, los resultados experimentales obtenidos al aplicar nuestro framework a una aplicación.

En el capítulo 5, *Descripción funcional*, se describe la metodología definida para extraer el comportamiento de la E/S de una aplicación paralela a nivel de POSIX y su integración con PIOM-MP para dar una visión global de ambos niveles.

En el capítulo 6, *Caso de Uso: PIOM en Clúster HPC en plataformas Cloud*, se muestra el análisis realizado en la plataforma Cloud para determinar si es una alternativa a los sistemas HPC tradicionales. Dado que el número de recursos que proporciona la plataforma Cloud y los parámetros a configurar es considerable, se ha definido una metodología para ayudar al usuario en la toma de decisiones al crear un sistema HPC en Cloud. En este capítulo también se añade una sección de resultados experimentales

para mostrar como a partir de los requerimientos de E/S tanto de la aplicación como del usuario se selecciona y se crea un cluster virtual en Cloud.

Por último, en el capítulo 7, *Conclusiones*, se presentan las conclusiones que nos ha aportado esta investigación y las líneas abiertas generadas para trabajos futuros.

2

Sistema de E/S para Computadores de Altas Prestaciones

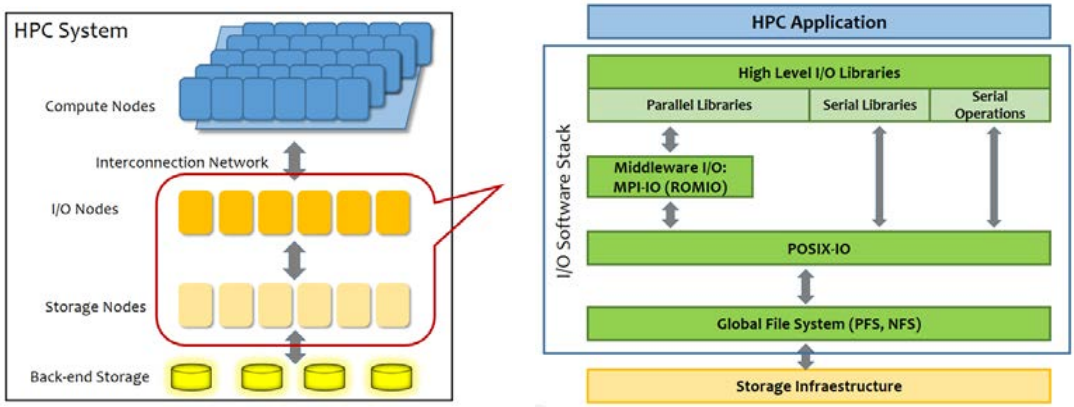
En este capítulo se describe el subsistema de E/S de los Computadores de Altas Prestaciones (HPC - *High Performance Computing* -). Asimismo, se describen los trabajos relacionados que están centrados en el análisis, evaluación y mejora del subsistema de E/S.

Este trabajo de investigación es una extensión del trabajo realizado en el proyecto de investigación 'Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones"[52], realizado en nuestro grupo de investigación. En ese trabajo se proponía una metodología para caracterizar la E/S paralela de las aplicaciones que utilizan librerías de E/S paralela y, se definía un modelo a nivel de MPI-IO.

2.1. Sistema de E/S Paralelo.

Las prestaciones de la mayoría de las aplicaciones científicas paralelas o también conocidas como aplicaciones HPC no sólo dependen del cómputo, también dependen de la memoria, de la red y del subsistema de E/S (Ver Figura 2.1). Por lo tanto, un mal uso, gestión, configuración del sistema de E/S puede afectar negativamente a las prestaciones de los sistemas HPC.

El Sistema de E/S de los sistemas HPC es un sistema complejo que está compuesto por un componente *hardware* y un componente *software*. En el caso del componente *hardware* con diferentes elementos y, en el caso del componente *software* con diferentes capas. Tanto la interrelación entre estos dos componentes como la interrelación entre los elementos y las capas de cada componente también añaden complejidad en el momento de analizar el sistema de E/S. En la Figura 2.1 se muestran los diferentes elementos de los que consta el componente *hardware* y las diferentes capas de las que consta el componente *software* del sistema de E/S.



(a) Componentes de la arquitectura del sistema de E/S

(b) Módulos de la pila software de E/S

Figura 2.1: Sistema de E/S

Dentro del componente *hardware* (ver Figura 2.1(a)) se pueden distinguir diferentes elementos: la red de interconexión de E/S (entre los nodos de cómputo y los nodos de E/S), los nodos de E/S, la red de interconexión entre los nodos de E/S y los dispositivos físicos de almacenamiento propiamente dichos, y los dispositivos de almacenamiento. En los últimos años, una solución hardware ha prosperado para mejorar la transferencia de datos entre el sistema de computación y el sistema de E/S. Esta solución se conoce como *I/O Forwarding* y consiste en añadir un servidor remoto al que se envían peticiones de E/S para su procesamiento. Solución que implica añadir una nueva capa al sistema de E/S [53, 54]. La técnica *Burst-Buffer* [5] se podría considerar una mejora de esta capa *I/O Forwarding*. Esta nueva técnica combina componentes de memoria (*DRAM*) y discos de estado sólido (*SSD*). Para aplicar esta técnica se recomienda que la ubicación de los SSD estén entre los nodos de cómputo y el sistema de ficheros, para funcionar como almacenamiento local de un nodo de cómputo.

El componente *software* (ver Figura 2.1(b)), o también llamado pila de software de E/S *(I/O Software Stack)*, está compuesto por diferentes capas donde se ubican:

- librerías de E/S de alto nivel: Las librerías de E/S de alto nivel es una capa opcional que permite al usuario no conocer en detalle características del sistema de E/S. Estas librerías permiten que los programas implementados con ellas sean portables.
- *middleware* de E/S: MPI-IO [13] y ROMIO [14, 15],
- librerias de bajo nivel (POSIX-IO [16, 20]), y
- el sistema de ficheros global.

A medida que las capas están más cerca del sistema de ficheros, más dependiente del sistema físico son esas capas. Con lo cual, la información que se puede extraer es menos portable.

Dada la complejidad del sistema de E/S y su influencia en el rendimiento, existen diferentes propuestas de la literatura que se centran en investigar el sistema de E/S. Existen grupos de investigación que están proponiendo mejoras en las diferentes capas que componen el sistema de E/S. Como resultado de estos trabajos de investigación, a nivel de librería de alto nivel están entre otros, HDF5 [6, 7], PnetCDF [8], ADIOS [9], PASSION [10], PIDX [11] y SIONLib [12].

A nivel de middleware se puede encontrar normalmente el estándar MPI-IO con diferentes implementaciones (destacamos, ROMIO MPI-IO). MPI-IO se definió porque la interfaz POSIX-IO no proporciona la portabilidad y optimización que necesita la E/S paralela. MPI-IO es una interfaz definida por el MPI Forum, e incluida en el MPI-2 estándar, para cubrir características de la E/S paralela tales como accesos no-contiguos en memoria y en fichero, operaciones colectivas de E/S y E/S asíncrona. ROMIO es una implementación optimizada de MPI-IO, en la que se incluyen y están implementadas dos optimizaciones: las operaciones colectivas de E/S y *data sieving* (una técnica para mejorar los patrones de acceso no-contiguos).

A nivel de librería de bajo nivel está el estándar POSIX-IO (*Portable Operating System Interface*). La comunidad científica de HPC, cuando se desarrollaron los sistemas HPC, apostó por el uso de POSIX-IO como el modelo estándar para acceder de forma global al sistema de almacenamiento. POSIX-IO se ha actualizado, a lo largo de los años, para corregir posibles errores y para añadir algunas de las nuevas necesidades que la comunidad HPC requería. Sin embargo, independientemente de las actualizaciones que ha sufrido POSIX desde sus inicios, uno de los problemas que ofrece POSIX es que no permite la escalabilidad de las aplicaciones. Por esta razón, muchos grupos de investigación desarrollaron nuevas capas en la pila de software de E/S y se siguen desarrollando nuevas soluciones para organizar los accesos y optimizar la E/S antes de interaccionar con el nivel de POSIX.

A nivel de sistema de ficheros, se puede tener un sistema de ficheros paralelo, distribuido o de red. De todos los sistemas de ficheros destacamos:

- NFS *(Network File System)* [20, 55, 56]: sistema de ficheros distribuido que permite que procesos remotos ubicados en nodos cliente accedan a un fichero o conjunto de ficheros en un nodo servidor, como si esos procesos estuvieran trabajando con un fichero en su sistema de ficheros local.

- PVFS2 *(Parallel Virtual File System)* [17] : sistema de ficheros paralelo que permite acceder y guardar los datos y los metadatos utilizando servidores basados en objetos. Es decir, los ficheros se representan como objetos y se almacenan en los servidores. Los componentes de este sistema de ficheros son servidores de metadatos (MS) y de datos(DS). Los metadatos son distribuidos en un conjunto de servidores usando una variable aleatoria para seleccionar el servidor de metadatos que le corresponde a un fichero creado.

- Lustre [18] : sistema de ficheros distribuido que permite implementar diferentes configuraciones, independientemente del número de nodos cliente, almacenamiento en disco y ancho de banda. Los componentes de este sistema de ficheros son: servidores de metadatos (MDS) que gestionan los metadatos, objetos para almacenar los metadatos (MDT), servidores para almacenamiento (OSS) que gestionan el almacenamiento, objetos para almecenar los ficheros de datos del usuario (OST) y los clientes de Lustre. Además, tiene como particularidad la habilidad de fragmentar los datos (*stripe data*) de un fichero y asignar estos fragmentos a múltiples OST (objetos donde se guardan los ficheros de datos del usuario) utilizando la técnica *Round-robin*.

- GPFS *(General Parallel File System)* [17]: sistema de ficheros paralelo, en el cual, todos los datos, metadatos y ficheros son accesibles por cualquier nodo del clúster, ya que se basa en un modelo de almacenamiento compartido. Igual que Lustre, GPFS tiene la habilidad de fragmentar los ficheros grandes en bloques y asignar esos fragmentos a diferentes discos utilizando la técnica *Round-robin*.

2.2. Aplicaciones Científicas Paralelas.

Otro elemento muy importante, en HPC, son las aplicaciones científicas que se ejecutan en estos sistemas. Las aplicaciones científicas generan una gran cantidad de datos que se almacenan para su posterior análisis y visualización, o para restaurar el punto de ejecución en un determinado momento.

Estas aplicaciones científicas se pueden clasificar dependiendo de si son intensivas en cómputo, en datos o en E/S. Méndez y otros [19] definen el concepto de severidad de E/S para identificar el grado de impacto de la E/S de una aplicación sobre un sistema HPC, y de esta manera catalogar si una aplicación es intensiva en E/S. El concepto de severidad de E/S se basa en las características de E/S de una aplicación. Dependiendo de los requerimientos de E/S, tales como el porcentaje de la capacidad de almacenamiento requerida por la aplicación, el peso de las operaciones de E/S y la E/S paralela en un nodo de cómputo, que necesita una aplicación para ejecutarse en un sistema HPC, definen 5 grados de severidad. En nuestro caso, el trabajo de investigación se ha centrado en las aplicaciones científicas paralelas de paso de mensajes (MPI) intensivas en E/S.

Las características de las aplicaciones científicas paralelas pueden afectar a las prestaciones de los sistemas HPC a nivel de computación, comunicación y de E/S. A nivel de E/S, entender el comportamiento de una aplicación científica paralela y como utiliza el sistema de E/S no es tarea fácil por la existencia de iteraciones complejas entre los múltiples niveles software [1]. Por lo tanto, es interesante descubrir los requisitos de la aplicación y su patrón de comportamiento o si se pueden producir cuellos de botella para el sistema de E/S. Identificar si se están produciendo esos cuellos de botella es importante. Por ejemplo, en el caso de incrementar el número de procesos para ejecutar la aplicación paralela, dado que una aplicación paralela intensiva en cómputo se puede convertir en una aplicación paralela intensiva en E/S, porque se disminuye el tiempo de cómputo pero se puede mantener (no se cambia la E/S) o aumentar (se fragmentan los datos que se envían al sistema de E/S) el tiempo de E/S.

En resumen, la combinación de las características de E/S de la aplicación junto con las características de la configuración de E/S del sistema, algunas veces, es lo que provoca que se genere un cuello de botella o que el rendimiento obtenido sea pobre.

2.3. Estrategias de E/S: serie y paralela.

Además de las características del sistema de E/S y de las características de las aplicaciones HPC, también se ha de tener en cuenta las estrategias de E/S utilizadas por la aplicación para distribuir y acceder a los datos en ficheros. Estas estrategias están relacionadas con como declara la aplicación el tipo de fichero (compartido o independiente) y como con los metadatos se accede al fichero o ficheros. Dependiendo del tipo de acceso a los ficheros se puede clasificar la E/S como serie o paralela. Dado que nuestra investigación se centra en las aplicaciones paralelas científicas MPI ejecutadas en sistemas

HPC, la E/S paralela tendría que ser una buena opción para proporcionar un mejor rendimiento, a nivel de MiB/s o IOPS, en este tipo de computadores; a diferencia de la E/S serie,.

Dentro de la E/S paralela, que puede realizar una aplicación, se pueden diferenciar varios casos:

- un fichero por proceso, accediendo a múltiples ficheros en paralelo; con lo cual, la E/S es individual (no colectiva) como se muestra en la Figura 2.2. Por esta razón, se considera que la E/S es serie,

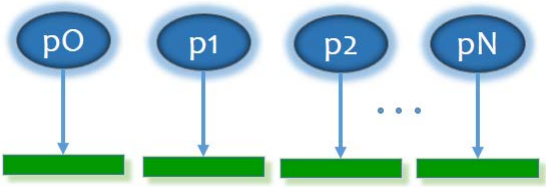


Figura 2.2: Cada proceso accede a un fichero.

- un fichero compartido donde sólo un único proceso accede al fichero. Es decir, el fichero se genera o se trata en paralelo, pero el acceso al sistema de almacenamiento lo realiza un único proceso (ver Figura 2.3). La E/S también se considera serie,

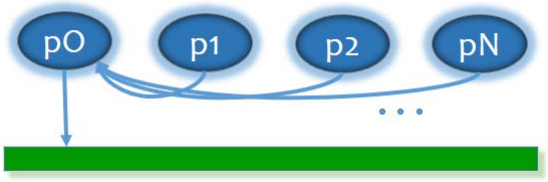


Figura 2.3: Un proceso accede al fichero compartido.

- un único fichero compartido por todos los procesos definidos para ejecutar la aplicación como se muestra en la Figura 2.4,

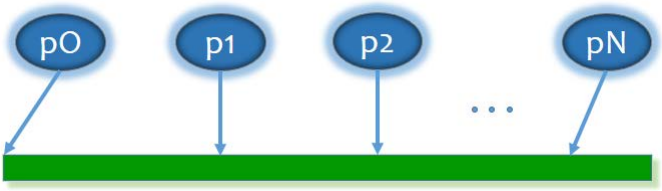


Figura 2.4: Todos los procesos acceden al mismo fichero compartido.

- un único fichero compartido por un número determinado de procesos (M, donde M<N) y este será menor al número de procesos total definidos para ejecutar la aplicación (ver Figura 2.5).

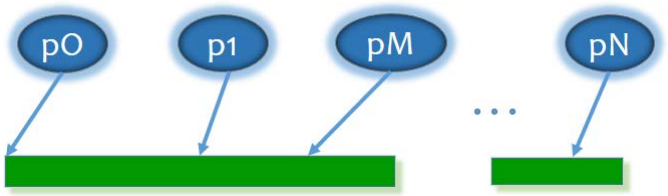
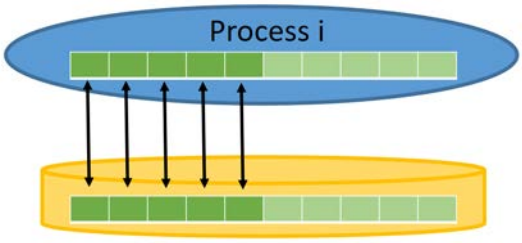


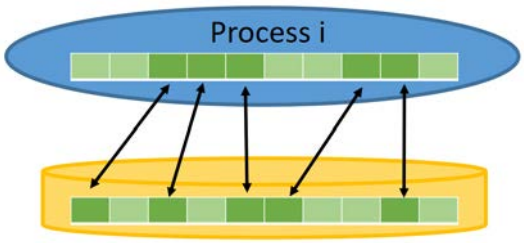
Figura 2.5: Un conjunto de procesos acceden a un determinado fichero mientras otro conjunto de procesos acceden a otro fichero.

2.4. Patrones de acceso.

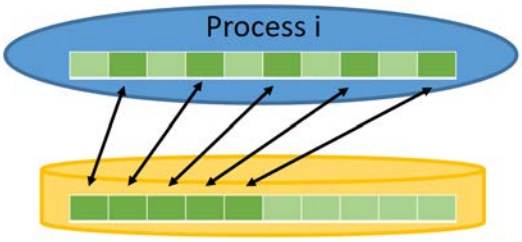
Anteriormente, se describió como los procesos, implicados en la ejecución de una aplicación, acceden a los ficheros abiertos por la aplicación. A continuación, se describirán los patrones de acceso más comunes que utilizan las aplicaciones para acceder a los datos contenidos en un fichero u obtener los datos de un fichero. Como se puede apreciar en la Figura 2.6, los patrones de acceso indican cómo se accede a los datos de un fichero determinado. Se pueden distinguir cuatro patrones de acceso: acceso contiguo en memoria y en el fichero (ver Figura 2.6(a)), acceso discontiguo en memoria y en el fichero (ver Figura 2.6(b)), acceso discontiguo en memoria (ver Figura 2.6(c)) y, finalmente, acceso discontiguo en el fichero (ver Figura 2.6(d)).



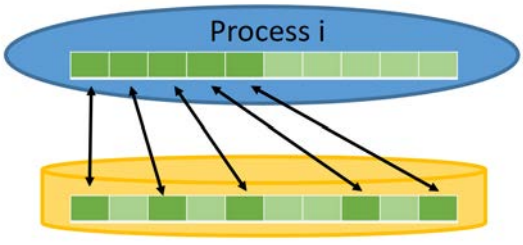
(a) Acceso contiguo en memoria y en el fichero



(b) Acceso discontiguo en memoria y en el fichero



(c) Acceso discontiguo en memoria



(d) Acceso discontiguo en el fichero

Figura 2.6: Tipos de acceso a los ficheros.

Identificar el tipo de acceso a los ficheros es importante, así como detectar que operaciones de E/S se han realizado en cada acceso. Pero, otro factor importante a detectar es el orden en el que

se producen las operaciones de E/S para mantener la consistencia de la información, dado que los sistemas de ficheros paralelo permiten la concurrencia. Es decir, los procesos pueden acceder al mismo fichero.

2.5. Operaciones colectivas de E/S.

Como se ha comentado en el capítulo 2.3, la E/S paralela tendría que proporcionar un buen rendimiento en los sistemas HPC cuando se ejecutan aplicaciones paralelas; sin embargo, esto no siempre es así. Dependiendo del volumen de datos que se transfiere en cada acceso, a veces se deben aplicar técnicas para optimizar esos accesos con el objetivo de intentar optimizar la E/S paralela. Una de estas técnicas se conoce con el nombre de *colectivas de E/S* [20]. Esta técnica consiste en mejorar el rendimiento agrupando peticiones de E/S de varios procesos. Para realizar una escritura colectiva se juntan los datos a escribir de varios procesos para crear un dato más grande y continguo, y este dato es el que se guardará en disco. Para realizar una lectura colectiva se recuperan datos de tamaño grande del disco y este dato se distribuirá a los procesos que han realizado peticiones de lectura. Con esta técnica se permite reducir el número de accesos a disco haciendo que estos accesos sean más eficientes. Esta técnica está implementada a nivel de MPI-IO.

2.6. Herramientas para analizar la E/S.

Para analizar la actividad de E/S, es decir, el comportamiento de la E/S de las aplicaciones científicas en los diferentes niveles, existen varias herramientas. A continuación, se detallan algunas de estas herramientas:

Wright y otros en [2] proponen RIOT I/O, una herramienta que caracteriza la aplicación desde el nivel de MPI hasta el sistema de ficheros paralelo. Además, introducen una herramienta de post-procesamiento capaz de generar informes estadísticos y representación gráfica de la actividad de E/S.

Kim y otros en [21] proponen IOPin, una herramienta binaria que detecta sólo un camino crítico de E/S, un cuello de botella. Detecta el proceso MPI que tiene la máxima latencia de E/S y traza e intercepta sólo ese proceso. Captura llamadas a nivel de MPI-IO y las asocia con las subllamadas al sistema de ficheros de PVFS. Proporciona información estadística de rendimiento, como la latencia, el rendimiento del disco, el número de solicitudes de E/S desde el cliente al servidor y la cantidad de accesos al disco para la solicitud de E/S. Al igual que la herramienta anterior no requiere modificación del código fuente ni la recompilación de los componentes de la pila de E/S.

Luu y otros en [1, 22] proponen Recorder, una herramienta diseñada para trabajar con sistemas de ficheros paralelos que permite identificar los cuellos de botella en la implementación actual de los

metadatos de lectura de HDF5. Esta herramienta captura las llamadas de las funciones de E/S en los diferentes niveles de la pila de E/S, incluido HDF5, MPI-IO y POSIX-IO. Con esta herramienta los usuarios pueden controlar que niveles se quieren trazar. E igual que las anteriores no requiere ninguna modificación de la aplicación o del código fuente de la librería de E/S.

Pillet y otros [23, 24] presentan Paraver, una herramienta de análisis para detectar problemas de rendimiento y entender el comportamiento de las aplicaciones. Permite analizar diferentes aplicaciones tales como aplicaciones MPI y OpenMP. Paraver se basa en trazas. A partir de los ficheros de traza, realizan un análisis de esas trazas y visualizan el comportamiento de las aplicaciones. Paraver puede mostrar la ejecución concurrente de diferentes aplicaciones. Proporciona información estadística.

Vijayakumar y otros [25] proponen ScalaIOTrace, una herramienta que intercepta funciones en diferentes niveles de la pila software de E/S y proporciona información estadística de las funciones de E/S. Además, proporciona una herramienta de análisis para detectar cuellos de botella o detectar el orden de precedencia de eventos.

Shende y otros en [26] presentan TAU, una herramienta para caracterizar la aplicación a diferentes niveles de la pila de software, aunque por ahora se intercepta el nivel MPI-IO y POSIX-IO. Esta herramienta permite analizar el rendimiento de la aplicación y proporciona información estadística sobre el tiempo total que pierde cada subrutina.

Kunkel y otros [27] presentan SIOX, una arquitectura modular para monitorizar, analizar y optimizar la pila software de E/S. SIOX permite analizar la E/S paralela en diferentes niveles, ya que intercepta algunas funciones a diferentes niveles: MPI-IO, POSIX-IO, NetCDF y HDF5. Sin embargo, no requiere MPI y puede ser aplicado a aplicaciones POSIX. SIOX porporciona información sobre las actividades de E/S en todos los niveles, así como información hardware relevante y métricas sobre la utilización del nodo.

Existen otros enfoques para analizar el comportamiento de la aplicación tales como el propuesto por Méndez y otros en [28]. Estos autores proponen una metodología para caracterizar la E/S paralela de las aplicaciones científicas a nivel de MPI-IO. Esta metodología busca obtener un modelo de comportamiento de la aplicación basado en el patrón de acceso que tiene y que es obtenido a partir de PAS2P-IO [29] (resultado de añadir a PAS2P [30], un módulo que permite analizar la E/S de la aplicación e identificar las fases significativas). Esta metodología consta de 3 fases: caracterización, análisis y evaluación que permiten describir el uso que hace la aplicación del sistema de E/S en cada una de las fases significativas y reproducir el comportamiento de la aplicación en diferentes sistemas de E/S. Esta metodología identifica el patrón espacial y el patrón temporal de acceso a cada uno de los ficheros ya que tiene como objetivo dar un modelo del comportamiento de E/S de la aplicación. Cada fase se describe proporcionando la información necesaria que permite replicar el comportamiento utilizando programas sintéticos programables. Se ha diseñado e implementado una herramienta que integra diferentes módulos para automatizar el método propuesto y obtener el

modelo de comportamiento de la aplicación describiendo el comportamiento de cada una de las fases significativas de acceso a cada uno de los ficheros. Existen varias utilidades, pero una utilidad que se puede destacar es dar soporte a la toma de decisiones cuando se tienen que seleccionar los recursos, dando información sobre el comportamiento de las aplicaciones: su patrón de acceso y el volumen de acceso.

Otro tema en el que se está trabajando es en dar soporte a la configuración del sistema de E/S. B.Behzad y otros en [31] proponen una herramienta que permite generar la mejor configuración del sistema de E/S, es decir, aquella que permite el mejor rendimiento del sistema de E/S. Esta herramienta tiene tres fases y en ella se utiliza un algoritmo genético para hallar la mejor configuración.

2.6.1. Darshan y Vampir.

Anteriormente, se han citado algunas de las herramientas de análisis existentes. Y de todas las existentes, a continuación, se comentan dos de las herramientas utilizadas durante esta investigación.

Darshan.

Darshan [32] es una herramienta que caracteriza la aplicación proporcionando estadísticas e información de tiempo acumulado (cantidad de bytes leídos o escritos, tiempo gastado en las operaciones, etc). Está implementada como un conjunto de librerías de usuario y captura las llamadas de las operaciones de E/S a nivel de MPI-IO y POSIX. Además, es transparente a la aplicación, es decir, no necesita que se modifique el código de la aplicación.

Vampir.

Vampir [33] es una herramienta para analizar el rendimiento de las aplicaciones paralelas. Esta herramienta proporciona información detallada del comportamiento de la E/S de las aplicaciones porque captura llamadas de MPI, funciones OpenMP, funciones de E/S y llamadas de asignación de memoria. Esta herramienta, también, proporciona estadísticas.

2.7. Conclusiones

En un sistema HPC, uno de los sistemas donde se pueden generar cuellos de botella es el sistema de E/S. El sistema de E/S de un sistema HPC es un sistema complejo por que está compuesto por diferentes elementos *hardware* y por la pila software de E/S. Dada esta complejidad se podría pensar que analizando sólo el sistema de E/S se pueden descubrir las causas por las que se generan los cuellos de botella. Sin embargo, esto no es cierto.

Analizar e identificar la estrategia de E/S que utiliza una aplicación, el patrón de acceso que utilizan los procesos al acceder a la información de un fichero, la cantidad de volumen que se transfiere, las operaciones de E/S que realiza la aplicación en esos ficheros y el orden en el que se realizan esas operaciones proporcionan información clave para entender porque se producen cuellos de botella y/o se obtiene un rendimiento pobre en un sistema HPC con una determinada configuración de E/S.

3

PIOM-PX: Un modelo del comportamiento de la E/S de aplicaciones paralelas a nivel de POSIX-IO

En este capítulo, se presenta un modelo para describir el comportamiento de la Entrada/Salida (E/S) de las aplicaciones paralelas en la capa de POSIX y su integración con el modelo de comportamiento a nivel de MPI. Ya que, para analizar, configurar y minimizar el impacto de la E/S en los sistemas paralelos (arquitectura y aplicación), es necesario conocer el comportamiento de la E/S de las aplicaciones científicas paralelas.

3.1. Introducción

Las aplicaciones científicas resuelven problemas de diferentes áreas de conocimiento tales como física, química, materiales y biomedicina. Estas aplicaciones son implementadas por científicos con un gran conocimiento en su área, pero la gran mayoría de esos expertos desconoce las características de los sistemas HPC dónde se ejecutan. En algunas ocasiones cuando una aplicación científica es ejecutada en un determinado sistema HPC, en el sistema de E/S se produce un cuello de botella o el rendimiento obtenido, de esa aplicación, es pobre.

Para identificar el motivo por el que se originan estos problemas en el sistema de E/S se ha de analizar la aplicación, su interacción con el sistema de E/S y el propio sistema de E/S. En nuestro caso, el trabajo de investigación propuesto se centra en el análisis de las aplicaciones paralelas MPI intensivas en E/S para obtener información que permita analizar el impacto en el sistema de E/S de los sistemas HPC.

En líneas generales, cuando una aplicación se ejecuta en un computador o sistema HPC, genera un fichero o ficheros donde se guardan los resultados obtenidos de procesar los datos de la aplicación. Para reducir el impacto de las aplicaciones paralelas en el sistema de E/S, es necesario conocer como se comportan estas aplicaciones paralelas a nivel de E/S. Es decir, es necesario conocer el comportamiento de E/S de la aplicación.

3.2. Modelizar el comportamiento de la E/S.

En primer lugar, en este trabajo, cuando se menciona el ***comportamiento de E/S*** de una aplicación se hace referencia a la forma en que se realizan los accesos de E/S de la aplicación al fichero o ficheros abiertos durante la ejecución de esa aplicación. Es decir, el comportamiento de E/S describe cuándo accede la aplicación al fichero o ficheros, cómo la aplicación accede al fichero o ficheros, cuánto volumen de datos transfiere la aplicación al fichero o ficheros cada vez que hay un acceso de E/S, qué operaciones de E/S se realizan y en qué orden se realizan esas operaciones.

Para extraer el comportamiento de E/S de una aplicación paralela se necesita analizar las ráfagas de E/S generadas por la aplicación, para identificar los patrones de E/S. Dada la diversidad de las aplicaciones científicas pueden existir patrones de E/S diferentes tales como accesos a un mismo

fichero, accesos a ficheros de una misma aplicación, accesos a aplicaciones que se ejecutan en un mismo sistema. De ahí la necesidad de identificar el comportamiento de E/S de una aplicación, ya que es uno de los factores que influye en el rendimiento de la aplicación. Pero, para entender el comportamiento de E/S se necesita un modelo que permita representar los patrones de E/S de una aplicación. Un ***patrón de E/S*** es un conjunto de eventos de E/S recurrentes, que se repiten, durante la ejecución de la aplicación.

Existen diferentes tipos de modelos para representar un sistema tales como los modelos matemáticos, algorítmicos, lógicos y funcionales. Aunque para seleccionar el modelo apropiado se ha de tener en cuenta las características del sistema a representar. Por ejemplo, si en el sistema existen variables aleatorias se debería definir un modelo determinístico o estocástico. Mientras que si existen parámetros se tendría que utilizar un modelo paramétrico o no paramétrico.

En nuestro caso, se quiere definir un modelo de comportamiento de E/S de las aplicaciones paralelas MPI intensivas en E/S a nivel de POSIX-IO. Para identificar que aplicaciones se consideran intensivas en E/S se propuso, en [34], una forma de identificar este tipo de aplicaciones teniendo en cuenta los requerimientos de E/S y el tiempo de E/S que necesita la aplicación. El primer paso para modelizar el comportamiento es identificar los elementos básicos, el segundo paso identificar las relaciones entre ellos (fases) y como último paso se propone la forma de representación en fases.

El modelo propuesto, PIOM-PX, para describir el comportamiento de E/S de una aplicación paralela se realiza a nivel de fichero (a nivel de vista lógica del fichero). A este nivel se identifica el patrón espacial y el patrón temporal. El **patrón espacial** informa qué ficheros son abiertos por la aplicación, qué proceso accede a un fichero y a qué posiciones del fichero accede el proceso durante la ejecución de la aplicación. El **patrón temporal** informa en qué orden se realizan los eventos de acceso al fichero.

A continuación, en la Figura 3.1, se muestra un ejemplo de la representación del comportamiento de E/S de una aplicación paralela propuesta por PIOM-PX. Analizando la gráfica de esa figura se puede observar que PIOM-PX muestra:

- el número de procesos implicados en la ejecución de la aplicación,
- los eventos de E/S (operaciones de lectura y escritura) que realiza la aplicación,
- el orden en el que se han producido los eventos,
- el *offset* utilizado por los eventos de E/S para desplazarse dentro de un fichero,
- y, el número de fases que contiene la aplicación.

En el caso que la aplicación usara más de un fichero, PIOM-PX también permite representar el número de ficheros utilizados durante la ejecución de la aplicación. La salida del modelo también muestra el volumen de información que se está accediendo o transfiriendo.

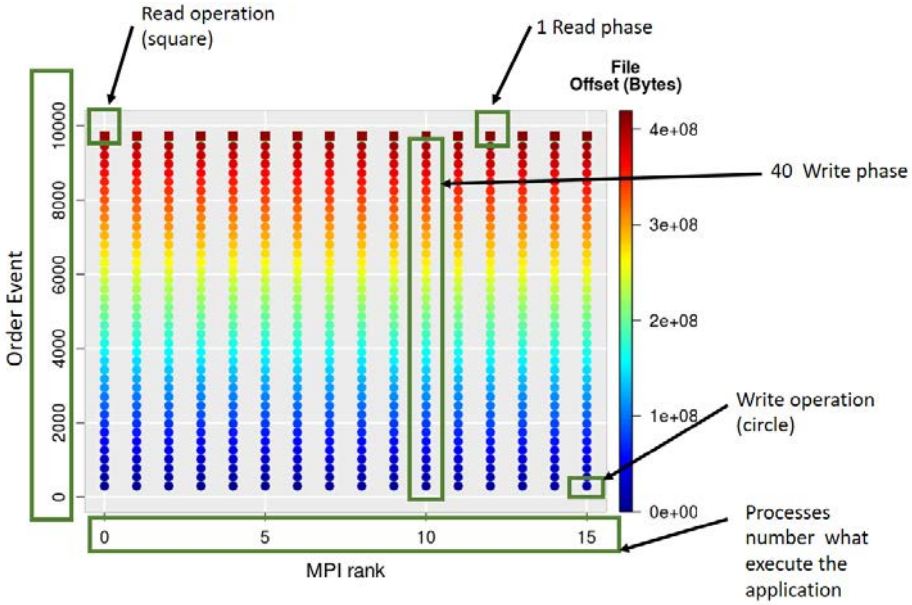


Figura 3.1: Representación del comportamiento de E/S para una aplicación paralela con PIOM-PX. Se muestra el comportamiento de la aplicación NASBT-IO clase A, ejecutada con 16 procesos, los cuales acceden a un fichero compartido y utilizan el desplazamiento discontiguo.

Para el ejemplo mostrado en la Figura 3.1, se observa que la aplicación se ha ejecutado con 16 procesos, la aplicación abre un único fichero y los 16 procesos acceden al mismo fichero. En esa misma figura, también se refleja el número de fases identificadas en la aplicación: 40 fases de escritura(cada fase, de los 16 procesos, tiene escrituras) y una fase de lectura. La fase de escritura está representada por un círculo y la fase de lectura por un cuadrado. También, se observa el orden (*order event*) en el que se han producido las fases, 40 fases de escritura seguidas y finalmente una fase de lectura. Asimismo, también se observa el desplazamiento de los procesos.

Dado que las aplicaciones que utilizan la capa/interfaz de MPI se podían analizar con PIOM-MP (llamada PAS2P-IO en [29]), se necesitaba analizar las aplicaciones que sólo utilizan la interfaz POSIX-IO para interceptar su comportamiento de E/S. Para construir el modelo a nivel de POSIX, PIOM-PX, se sigue la metodología propuesta por Méndez y otros en [29], donde tal y como se ha descrito en el capítulo 2 se propone un modelo de comportamiento de la E/S de una aplicación a nivel de MPI-IO.

Cómo última fase se propone el diseño de una herramienta con la integración de PIOM-PX y POSIX-MP, dado que los dos modelos son independientes. Esta integración da lugar a PIOM, un modelo de comportamiento de E/S de las aplicaciones paralelas MPI a nivel de MPI y a nivel de POSIX-IO. La Figura 3.2 muestra donde estarian ubicados los dos modelos que componen PIOM: PIOM-PX y PIOM-MP.

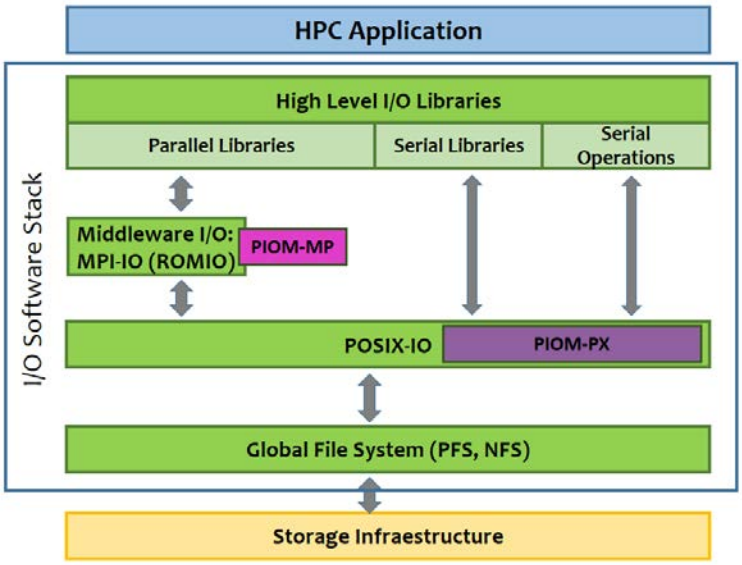


Figura 3.2: Ubicación de los modelos PIOM-PX y PIOM-MP en la pila software de E/S. Estos modelos permiten interceptar y analizar las operaciones a nivel de POSIX-IO y MPI-IO, respectivamente.

3.2.1. Descripción de la fase de E/S.

La mayoría de las aplicaciones científicas paralelas son repetitivas. Este comportamiento repetitivo se puede observar durante la ejecución de dichas aplicaciones. En realidad, durante el tiempo de ejecución se pueden detectar porciones con un comportamiento, normalmente, homogéneo. Estas porciones similares, comúnmente, son llamadas fases [35]. Esta propiedad de repetitividad también se puede aplicar a las operaciones de E/S en los ficheros. Por esta razón, se ha definido el modelo de comportamiento de E/S de las aplicaciones paralelas teniendo en cuenta el concepto de fase.

Para detectar una fase de E/S se analizan las operaciones de E/S de un fichero. El modelo de comportamiento de E/S de una aplicación paralela está definido por el modelo de comportamiento de cada uno de los ficheros de la aplicación.

Una ***fase de E/S*** es una secuencia de operaciones consecutivas de E/S a un fichero (se considera la vista lógica de un fichero para obtener el patrón espacial de los accesos). En un fichero se puede tener una o varias fases y el número de apariciones de una determinada fase de E/S en un fichero determina el número de veces que se repite (número de repeticiones) la fase de E/S en el fichero. Para detectar cuantas fases contiene un fichero se analizan las operaciones de E/S que aparecen en cada fase, el orden en el que se realizan y el volumen de datos que se transfiere en una fase. Por lo tanto, como PIOM-PX está basado en el concepto de fase, también, informa de las operaciones de E/S que contiene cada fase, el volumen de datos transferido en cada fase y el número de repeticiones de cada fase. En el ejemplo mostrado en la Figura 3.1, a través de PIOM-PX se obtiene que cada fase de escritura tiene sólo una operación de escritura mientras que, la fase de lectura tiene 40 operaciones de lectura.

El análisis de las operaciones de E/S permite identificar el patrón espacial y temporal. Esta identificación posibilita describir el comportamiento de la E/S de una aplicación. Para analizar las operaciones, las características principales que se tienen en cuenta en el análisis de una operación son:

- tipo de operación (IOP_type): escritura (w), lectura (r) o escritura/lectura (w/r),
- tamaño de la petición o tamaño de una operación de E/S (rs),
- posición en la vista lógica del fichero ($offset$),
- desplazamiento en el fichero ($disp$): diferencia entre los offsets de dos operaciones de E/S consecutivas.

A partir de estas características de las operaciones de E/S se extrae el patrón espacial. Para obtener el patrón espacial de los accesos se considera la vista lógica del fichero e indica que operaciones y cómo se accede al fichero.

Para extraer el patrón temporal se debe identificar el orden en el que se realizan los eventos de E/S. En nuestro caso, uno de los objetivos es definir un modelo portable. Es decir, un modelo que represente el comportamiento de la aplicación independientemente del sistema en el que se extrae y que pueda ser replicado en otros sistemas. En un sistema paralelo, siempre es posible ordenar los eventos de un mismo nodo (si se respeta la monotonicidad del reloj), en cambio, relaciones de causalidad entre eventos de nodos diferentes pueden verse distorsionadas. Para obtener el patrón temporal sólo se requiere ordenar los eventos (y no tanto conocer el instante exacto en que ocurrieron), para ello se definieron el ***tick*** y ***subtick***, dos contadores lógicos para establecer el orden de los eventos. El ***tick*** corresponde al contador que permite controlar el orden lógico en el que se producen los eventos MPI. Mientras que el ***subtick*** corresponde al contador que permite controlar el orden lógico en el que se producen los eventos POSIX.

Cada proceso genera una secuencia de eventos y estos eventos pueden ser de diferentes tipos:

- envío de un mensaje (al ejecutar enviar),
- recepción de un mensaje (al ejecutar recibir),
- Operaciones MPI de E/S (MPI_IO):
  - apertura y cierre de un fichero
  - lectura y escritura en un fichero,
- Eventos locales/internos (eventos, sin comunicación):
  - Operaciones POSIX de apertura y cierre de un fichero

- Operaciones POSIX de lectura y escritura en un fichero.

Otra característica importante de las operaciones de E/S es:

- distancia entre dos operaciones de E/S (*dist*): diferencia entre *tick.subtick* de dos operaciones de E/S consecutivas.

En resumen, el modelo de comportamiento de E/S de un fichero está descrito por el conjunto de fases de E/S, que describen el acceso al fichero, y el número de repeticiones de cada fase. Por tanto, el modelo de comportamiento de E/S de un fichero, está representado por un conjunto de fases de E/S.

3.2.2. Definición del modelo de comportamiento de E/S de una aplicación paralela

Para describir el modelo de comportamiento de E/S de una aplicación paralela MPI, se define PIOM. El propósito de definir PIOM es representar el comportamiento de E/S de la aplicación a dos niveles: MPI-IO y POSIX-IO. Puede ocurrir que una aplicación paralela MPI:

- sólo utilice la librería POSIX-IO para hacer la E/S (se mostrará el comportamiento a nivel POSIX-IO),
- sólo utilice la líbreria MPI-IO (de esa aplicación se describirá el comportamiento a nivel MPI-IO y a nivel POSIX-IO),
- utilice ambas librerías.

Nuestro primer objetivo es identificar que información se necesita extraer de la aplicación para que el modelo sea representativo de ésta. Dado que la capa POSIX-IO es la más próxima al sistema de ficheros, la información obtenida puede ser dependiente del sistema. Por esta razón, nuestro segundo objetivo, es obtener información que no sea dependiente del sistema para que el comportamiento sea portable.

Para conseguir estos objetivos iniciales se analiza la aplicación desde puntos de vista diferente: fichero y operación de E/S. Permitiendo identificar el comportamiento de la aplicación a nivel de fichero y de fase de E/S. Considerando la información de todos los ficheros, se obtiene el comportamiento de E/S de la aplicación.

3.2.3. Características de E/S de una aplicación paralela.

Como se mencionó anteriormente, el modelo de comportamiento del sistema de E/S se realiza a nivel de fichero. Por lo tanto, el análisis a nivel de POSIX-IO (PIOM-PX), también, se realiza a nivel de fichero/ficheros generados por la aplicación.

En la tabla 3.1 se describen los parámetros seleccionados para cada nivel. Este conjunto de parámetros son definidos para obtener el comportamiento de la E/S de una aplicación paralela, ya que con ellos se pueden definir la mayoría de las características de E/S de una aplicación. La última columna de la tabla indica de donde se obtiene cada campo, explicado con más detalle en el Capítulo 4.

Tabla 3.1: PIOM-PX Model Characteristics

Identifier	**File**	**Origin**
file_id	File Identifier.	Trace File
file_name	File Name.	Trace File
file_size	File Size.	Post-process
file_np	Count of MPI processes that open the file $file\_id$.	Post-process
file_accessmode	This can be sequential, strided or random.	Post-process
file_fileaccesstype	Read only(R), write only (W) or write and read (W/R).	Trace File
file_accesstype	$file\_np$ processes can access to shared Files or 1 File per Process.	Post-process
file_nphase	Count of phases of the file.	Post-process
	I/O Phase (PhIO)	
Ph_id	Identifier of an I/O Phase.	Post-process
Ph_processid	Identifier of Process implied in the phase.	Post-process
Ph_np	Number of processes implied in the phase.	Post-process
Ph_weight	Transferred data volume during the phase. It is expressed in bytes.	Post-process
Ph_nrep	Number of repetitions per phase.	Post-process
Ph_niop	Number of I/O operations.	Post-process
IOP_type	Data access operation type, which can be write, read, or write/read.	Trace File
rs	Request size or size of an I/O operation.	Post-process
offset	Operation offset, which is a position in the file's logical view.	Trace File
disp	Displacement into file, which is the difference between the offset of two consecutive I/O operations.	Post-process
dist	Distance between two I/O operations, which is the difference between $tick.subtick$ of two consecutive I/O operations.	Post-process
	Application	
app_np	Number of processes that the application needs to be executed.	Post-process
app_nfiles	Number of files used by the application.	Post-process
app_st	Storage capacity required by the application for the input files, temporal files and input/output files.	Post-process

A continuación, se comentan las características/parámetros del modelo PIOM-PX, aunque es

extrapolable al modelo PIOM.

Parámetros a nivel de fichero.

A nivel de fichero, los parámetros que se consideran de interés son aquellos que determinan a qué fichero se accede, cómo se accede al fichero y cuántos procesos acceden a él. Los parámetros se obtienen por fichero analizado y, algunos de ellos se extraen directamente a partir del fichero de traza de la aplicación mientras que otros son calculados. Estos parámetros son:

- *file_id*: identificador del fichero. Permitirá identificar las operaciones de E/S que están asociadas al fichero a analizar.
- *file_name*: nombre del fichero a analizar. Al abrir un determinado fichero se le asignará un *file_id* que permitirá controlar el archivo que se está analizando.
- *file_size*: tamaño del fichero analizado.
- *file_np*: contador que registra la cantidad de procesos MPI que abren el fichero $file\_id$.
- *file_accessmode*: indica cómo se accede al fichero: secuencial, strided o aleatorio.
- *file_fileaccesstype*: determina el tipo de apertura del fichero: sólo lectura (R), sólo escritura (W) o escritura y lectura (W/R).
- *file_accesstype*: se determina si se está trabajando con un fichero por proceso o un fichero compartido. Esta información se obtiene por el número de procesos ($file\_np$) que accede a un determinado fichero.
- *file_nphase*: contador que registra el número de fases que contiene un fichero.

Parámetros a nivel de fase de E/S

El modelo de comportamiento de E/S permite representar los patrones de acceso de E/S de una aplicación cuando accede a un fichero. Así, una fase de E/S se puede definir como una secuencia repetitiva del mismo patrón de acceso en un fichero para un número de procesos de la aplicación paralela.

Para el modelo, la información considerada de interés, a nivel de operación, es:

- *IOP_type*: tipo de operación para acceder a los datos: escritura, lectura o escritura/lectura.
- *rs*: tamaño de petición o tamaño de una operación de E/S.

- *offset*: es la posición de la vista lógica del fichero, a partir de la cual se han de escribir o leer los datos.
- *disp*: desplazamiento en el fichero. Viene determinado por la diferencia entre el *offset* de dos operaciones de E/S consecutivas.
- *dist*: distancia entre dos operaciones de E/S, el cuál es la diferencia entre $tick.subtick$ de dos operaciones de E/S consecutivas.

Cada fase de E/S contendrá una o varias operaciones consecutivas. Los parámetros generales que definen e identifican una fase en un fichero son:

- *Ph_id*: identificador de la fase.
- *Ph_processid*: identificador del proceso/procesos implicados en la fase.
- *Ph_np*: número de procesos implicados en la fase.
- *Ph_nrep*: número de veces que se repite una determinada fase dentro de un fichero.
- *Ph_niop*: número de operaciones de E/S que contiene una fase.

Otro parámero importante es el peso de la fase de E/S (*Ph_weight*) que corresponde al volumen de datos transferidos durante la fase. Se expresa en bytes y, se calcula con la fórmula mostrada en la expresión 3.1:

$$Ph\_weight_(Ph\_id) = \sum_{i=0}^{Ph\_niop} Ph\_np \times rs \times rep \tag{3.1}$$

para tener en cuenta si existen operaciones con diferente *rs*.

Determinar el volumen de datos transferidos en una fase, es uno de los criterios que permite identificar que fases serán más significativas. Es decir, que fases pueden influenciar más en el rendimiento de la aplicación.

Parámetros a nivel de aplicación.

Al realizar el análisis a nivel de fichero y a nivel de operación de E/S, implica obtener información respecto a la aplicación. A nivel de aplicación, los parámetros que se consideran importantes son:

- *app_np*: número de procesos que la aplicación necesita para ser ejecutada.
- *app_nfiles*: número de ficheros abiertos por la aplicación durante el tiempo de ejecución. Se tendrán en cuenta tanto los ficheros que la aplicación necesita para su ejecución (ficheros de entrada) como los ficheros generados como resultado de su ejecución (ficheros de salida).

- *app_st*: el tamaño de estos ficheros permite conocer la capacidad de almacenamiento que requiere la aplicación. Para calcular este parámetro se han de tener en cuenta, si es el caso, los ficheros de entrada que necesita la aplicación para ser ejecutada, los ficheros temporales que genera la aplicación cuando está en ejecución y los ficheros de salida que genera la aplicación al finalizar su ejecución (ver expresión 3.2).

$$app\_st = \sum_{i=1}^{nf\_IOF} file\_size\_IO_i + \sum_{i=1}^{nf\_INF} file\_size\_IN_i \tag{3.2}$$

donde *nf_IOF* es el número de ficheros de salida, *file_size_IO* es el tamaño de cada uno de los ficheros de salida, *nf_INF* es el número de ficheros de entrada que la aplicación necesita para su ejecución y *file_size_IN* es el tamaño de cada uno de los ficheros de entrada.

3.3. Análisis de Casos.

En este punto, los experimentos realizados se definieron con el objetivo de validar la funcionalidad de PIOM-PX y su integración con PIOM-MP. Estos experimentos permitirán comprobar como se representan con el modelo PIOM diferentes patrones de acceso que muestran distintos comportamientos de E/S. Para ello se utilizarán programas sintéticos programables que generen patrones conocidos, se analizarán y se mostrará la información que ofrece PIOM para describir el comportamiento de la aplicación.

3.3.1. Aplicaciones sintéticas o benchmarks

Los experimentos se han realizado con el benchmark IOR [36] como aplicación y un programa sintético escrito en Fortran.

IOR permite generar diferentes patrones de acceso de E/S para distintas interfaz de E/S: HDF5, MPI-IO, POSIX-IO. Este benchmark está escrito en C y para el proceso de sincronización usa MPI. IOR está diseñado para medir el rendimiento de la E/S de los sistemas de ficheros paralelos. Este programa paralelo realiza escrituras y lecturas hacia y desde los ficheros, bajo unas ciertas condiciones, y muestra las tasas de rendimeinto resultantes. IOR proporciona diferentes parámetros para su personalización. En nuestro caso, los parámetros que se configuran dada su relevancia son:

- $-a$: permite seleccionar la interfaz: MPI-IO, POSIX-IO,
- $-b$: hace referencia al tamaño de bloque, cantidad de bytes consecutivos que se quieren escribir por proceso. En nuestro caso, se utiliza para determinar el tamaño de la petición (*rs*) que se quiere transferir,

- $-c$: se habilitan las operaciones colectivas,
- $-F$: habilita que se genere un fichero por proceso,
- $-N$: número de procesos que ejecutan el benchmark,
- $-s$: número de segmentos. Para nuestro modelo, este parámetro permite indicar el número de repeticiones de las fases,
- $-t$: tamaño de transferencia en bytes. En nuestro caso, se utiliza para indicar el tamaño del fichero que se quiere generar.

Se ha ejecutado IOR usando los compiladores Intel y GNU para analizar su influencia en las operaciones detectadas. La distribución MPI utilizada ha sido Intel MPI 2017.

El programa sintético escrito en Fortran, que se ha utilizado, realiza 100 escrituras en un fichero compartido. A continuación, se muestra su código:

Algorithm 1 Fortran Program

```
integer ierr, J, I, myrank, BUFSIZE, thefile
parameter (BUFSIZE=1024)
integer buf(BUFSIZE)
integer rank
integer intbuf, myposition
character(10) :: a
character(len=1024) :: filename
character(len=1024) :: format_string
double precision starttime, endtime, timerank

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, num_procs, ierr)
  inquire(iolength=intbuf) buf
  starttime = MPI_Wtime()
  OPEN(7,FILE="DASDlarge.txt",ACCESS='DIRECT',RECL=intbuf)
  do J = 1, 100
        myposition = (myrank*100)+J
        WRITE(7,REC=myposition) buf
  enddo
  CLOSE(7)
  endtime   = MPI_Wtime()
  timerank = endtime-starttime
  print *, 'rank: ', myrank, 'Time:', timerank

 call MPI_FINALIZE(ierr)
```

3.3.2. Sistemas HPC

Las características del entorno experimental en el que se realizaron los experimentos se detallan en la tabla 3.2. Como se puede apreciar se utilizaron dos sistemas HPC: SuperMUC (LRZ) [38] y Finisterrae2 (CESGA) [37]. Estos dos supercomputadores utilizan diferentes sistema de ficheros: SuperMUC utiliza GPFS y Finisterrae2 utiliza Lustre, lo que ha permitido apreciar el impacto del sistema de ficheros a nivel de POSIX.

Tabla 3.2: Descripción de los Sistemas HPC

Components	**Finisterrae2 (CESGA)**	**SuperMUC (LRZ)**
Compute Nodes	306	9216
CPU cores (per node)	24	16
RAM Memory	128GB	32GB
Local Filesystem	Linux ext4	Linux ext3
Global Filesystem (GFS)	NFS	NFS
Capacity of GFS	1.1TB	10x564x10TB
Global Filesystem (PFS)	Lustre	GPFS
Capacity of PFS	695TB	12PB
MPI-Library	mpich2 (1.5)	IBM PE
Number of data servers	4 OSS and 12 OSTs	80 NSD
Number of Metadata Server	1	
Stripe Size	1MiB	8MiB
Interconnection	Infiniband FDR@56Gbps	Infiniband FDR10

3.3.3. Experimentos

El objetivo de estos experimentos es extraer el comportamiento de E/S de los experimentos propuestos y realizados. El análisis presentado a continuación fue publicado en [34]. Los experimentos con IOR fueron definidos para evaluar dos estrategias diferentes de E/S: `1 Fichero por Proceso` y `1 Fichero Compartido`. El patrón *strided* se consiguió utilizando la técnica de las colectivas utilizando el modo habilitado y automático. Cada experimento se ejecutó con 16 procesos MPI por nodo de cómputo. A continuación, se describen los experimentos definidos:

(a) 1 fichero por proceso usando la interfaz POSIX:

- Objetivo: Detectar las operaciones POSIX-IO para una aplicación que sólo usa POSIX como librería de E/S. Estas serían las operaciones POSIX consideradas puras de escritura y lectura. Cada proceso MPI escribe y lee su propio fichero.

- Línea de comando:
```
IOR -a POSIX -s 1 -b 8m -t 1m -F
```

(b) 1 fichero por proceso usando la interfaz MPI:

- Objetivo: Detectar las operaciones POSIX-IO generadas por una aplicación que usa operaciones MPI independientes.
- Línea de comando:
```
IOR -a MPIIO -s 1 -b 8m -t 1m -F
```

(c) Un único fichero compartido usando la técnica de las colectivas en modo *automático* para un patrón strided:

- Objetivo: Detectar las operaciones POSIX-IO generadas por una aplicación que usa operaciones colectivas MPI-IO.
- Línea de comando:
```
IOR -c -a MPIIO -s 16 -b 512k -t 512k
```

(d) Un único fichero compartido usando la técnica de las colectivas en modo *habilitado* para un patrón strided:

- Objetivo: Detectar las operaciones POSIX-IO generadas por una aplicación que usa operaciones colectivas con la técnica de las colectivas habilitada.
- Línea de comando:
```
romio_cb_read = enable
romio_cb_write = enable
IOR -c -a MPIIO -s 16 -b 512k -t 512k
```

La tabla 3.3 presenta los valores de los parámetros para el modelo PIOM a nivel de POSIX (PIOM-PX) obtenidos a partir de los experimentos definidos para IOR. Los parámetros que se muestran son los pertenecientes a nivel de aplicación y a nivel de fichero.

Al ejecutar estos experimentos se detectaron dos fases de E/S en todos ellos, debido a la lógica del benchmark IOR. Seguidamente, para entender las fases detectadas se explican con más detalle los experimentos ejecutados:

(a) 1 fichero por proceso usando la interfaz POSIX: IOR está configurado para escribir 8 MiB por proceso MPI usando la interfaz POSIX para la estrategia de entrada 1 fichero por proceso. Cuando cada proceso MPI escribe y lee, el tamaño de la petición es 1MiB ($rs = 1MiB$).

La Figura 3.3 muestra el comportamiento de la E/S para el experimento (a). Se muestran dos fases de E/S por fichero. La primera fase (fase 1) contiene 8 operaciones de escritura y la segunda (fase 2) 8 operaciones de lectura. Así, el parámetro *Ph_niop* de nuestro modelo será 8 para cada una de las fases. Se acceden a 16 ficheros en paralelo. Cuando se ejecuta IOR existe una ráfaga de

Tabla 3.3: Características de PIOM-PX para el benchmark IOR

Identifier	**(a)**	**(b)**	**(c)**	**(d)**
app_np	16	16	16	16
app_nfiles	16	16	1	1
app_st	128 MiB	128 MiB	128 MiB	128 MiB
File				
file_name	testFile<IdProcess>	testFile<IdProcess>	testFile	testFile
file_size	8 MiB	8 MiB	128 MiB	128 MiB
file_accessmode	Seq	Seq	Strided	Strided
file_fileaccesstype	W/R	W/R	W/R	W/R
file_accesstype	1Fx1Proc	1Fx1Proc	Shared	Shared
file_nphase	2	2	2	2
file_np	1	1	16	16

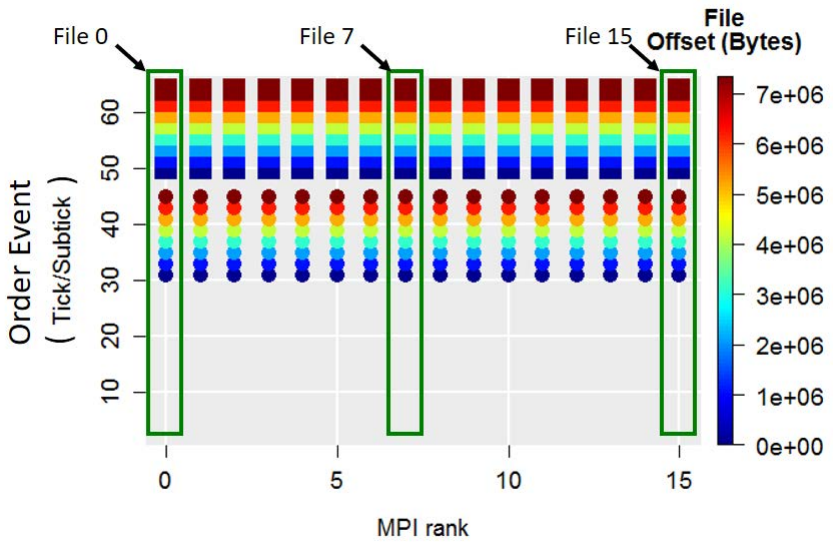
comunicación entre el proceso 0 y el resto (ver Figura 3.3(a)). Esta ráfaga genera una secuencia de 30 eventos de comunicación. En el evento 30 (tick =30) es cuando empieza la fase de escritura ya que es cuando se produce la primera operación de escritura que irá desde el tick.subtick 30.1 al 30.8. La fase de lectura es generada en el tick.subtick igual a 50.0 hasta 50.7.

El *Snippet*1 presenta parte del fichero de traza del *IdProcess* 2, el cual muestra parte de las operaciones de la fase 1. Se puede observar que antes de que ocurra la operación `write` se genera una operación `lseek64`. Cuando se analiza la fase 2, también se aprecia que antes de que ocurra una operación `read` se genera una operación `lseek64`.

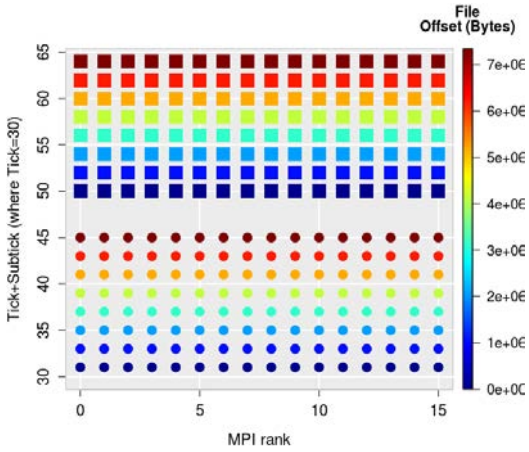
Snippet 1: Trace file of *IdProcess* **2**

```
## IdProcess file_id file_name NameOperation  tick subtick
2 6 testFile.00000002 open64 30 0
## IdProcess file_id NameOperation offset tick subtick
2 6 lseek64 0 30 1
## IdProcess file_id NameOperation offset rs tick subtick
2 6 write 0 1048576 30 2
2 6 lseek64 1048576  30 3
2 6 write 0 1048576 30 4
...
```

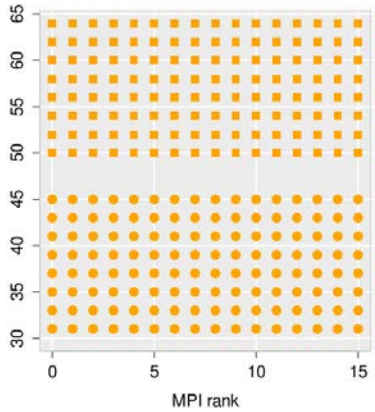
(b) 1 fichero por proceso usando la interfaz MPI-IO: las fases de E/S son similares a las detectadas en el experimento (a). En el *Snippet*2, se muestra una parte del fichero de traza correspondiente al proceso 2 (*IdProcess*= 2). En este caso, el número de operaciones de E/S que ocurren a nivel de POSIX-IO difiere del experimento (a) ya que por cada operación `MPI_File_write_at` generada



(a) Representación del $offset$ del fichero



(b) Representación del $offset$ del fichero (Zoom)



(c) Representación del peso de la fase (Zoom)

Figura 3.3: IOR para la interfaz POSIX configurado para 16 procesos MPI, 1 fichero por proceso para un patrón secuencial. Los círculos corresponden a las operaciones de escritura y los cuadrados corresponden a las operaciones de lectura. Para cada fichero se observan 2 fases de E/S: la fase 1 tiene 8 operaciones de escritura y la fase 2 tiene 8 operaciones de lectura. La escala de color en la Figura 3.3(c) muestra el peso, el cual es 1 MiB $\times$ $file\_np$ por subtick. El peso de la fase 1 y 2 son 8 MiB para cada fichero por proceso.

en el nivel MPI-IO, se genera una operación `write` pero no la operación `lseek`. Ya que el $offset$ está incluido en la operación `MPI_File_write_at` y por lo tanto a nivel de POSIX no se genera la operación lseek64. En el caso de la lectura, por cada `MPI_File_read_at` se genera una operación `read`. De nuevo, no se genera ninguna operación lseek64 porque la función `MPI_File_read_at`

ya contiene el valor del offset. Cuando es una operación MPI sólo se muestra el tick, mientras que cuando se intercepta la operación POSIX se muestra el tick y el subtick.

Snippet 2: Trace file of *IdProcess* 2

```
## IdProcess file_id NameOperation file_name tick
2 0x6e38b8 MPI_File_open testFile.00000002 31
## IdProcess file_id file_name NameOperation tick subtick
2 22 testFile.00000002 open64 31 0
## IdProcess file_id NameOperation offset rs tick
2 0x6e38b8 MPI_File_write_at 0 1048576 32
## IdProcess file_id NameOperation offset rs tick subtick
2 22 write 0 1048576 32 0
2 0x6e38b8 MPI_File_write_at 1048576 1048576 33
2 22 write 0 1048576 33 0
 ...
```

(c) Un único fichero compartido usando la técnica de las colectivas en modo `automático` para un patrón strided:

Snippet 3: Trace file of *IdProcess* 0

```
## IdProcess file_id NameOperation file_name tick
0 0x2124fc8 MPI_File_open testFile3 31
## IdProcess file_id file_name NameOperation tick subtick
0 6 testFile3 open64 0 66 31 0
0 0x2124fc8 MPI_File_get_info 32
## IdProcess file_id NameOperation offset rs tick
0 0x2124fc8 MPI_File_write_at_all 0 524288 33
## IdProcess file_id NameOperation offset rs tick subtick
0 6 write 0 524288 33 0
0 0x2124fc8 MPI_File_write_at_all 8388608 524288 34
## IdProcess file_id NameOperation offset tick subtick
0 6 lseek64 8388608 34 0
0 6 write 0 524288 34 1
...
```

Se ha definido el patrón strided modificando los valores de los parámetros tamaño de bloque (-b) y el tamaño de transferencia (-t) con el mismo valor. Además, para obtener la misma E/S en el experimento a y b, al parámetro número de segmentos (-s) se le ha asignado el valor 16 que corresponde al número de repeticiones. En total, cada proceso escribe y lee 8MiB usando como tamaño de petición 512 kiB.

En el *Snippet*3, se puede observar una operación `lseek64` y `write` para cada `MPI_File_write_at_all` exceptuando para la primera operación de escritura colectiva. El desplazamiento es igual a $file\_np \times rs = 8388608$ Bytes, donde $file\_np = 16$ y $rs = t = 524288$ Bytes. En este caso, cada proceso MPI produce un fichero de traza similar, con la excepción del offset, donde inicialmente el offset es

$$offset(i) = Ph\_processid \times rs \tag{3.3}$$

y

$$offset(i+1) = offset(i) + rs \times file\_np \times (i-1) + Ph\_processid \times rs \tag{3.4}$$

donde $i \in \{1..s\}$ y `s` es el número de segmentos establecidos para el benchmark IOR.

También se puede apreciar que al habilitar las colectivas de E/S se detecta un nueva operación MPI: $MPI\_File\_get\_info$. Operación para obtener los valores clave de las variables (*hints*) asociados a un archivo que está siendo utilizado por MPI. Los *hints* afectan a los patrones de acceso y a los datos (diseño/tratamiento que se le dan). Existen varios *hints*, tales como $collective\_buffering$ (que especifica si la aplicación puede beneficiarse del almacenamiento en el buffer para las colectivas) y (*cb_block_size*) (que especifica el tamaño del bloque que se utiliza para el acceso colectivo al archivo).

(d) Un único fichero compartido usando la técnica de las colectivas habilitada para el patrón strided:

Snippet 4: Trace file of IdProcess 0

```
## IdProcess file_id NameOperation file_name tick
0 0xac80f0 MPI_File_open testFile3 31
## IdProcess file_id file_name NameOperation  tick subtick
0 6 testFile3 open64 31 0
0 0xab9338 MPI_File_get_info 32
## IdProcess file_id NameOperation offset rs tick
0 0xab9338 MPI_File_write_at_all 0 524288 33
### IdProcess file_id NameOperation offset rs tick subtick
0 6 write 0 8388608 33
0 0xab9338 MPI_File_write_at_all 8388608 524288 34
## IdProcess file_id NameOperation offset tick subtick
0 6 lseek64 8388608 34 0
0 6 write 0 8388608 34 1
...
```

Para trazar las operaciones de E/S para las aplicaciones que tienen la técnica de la colectivas habilitada, se han modificado algunas variables (*hints*) de ROMIO. El patrón strided explicado en el experimento (c) es el mismo que el empleado en este experimento. En el *Snippet*4, se puede observar operaciones de E/S similares para el experimento (c), pero el tamaño de la petición a nivel de POSIX es diferente. En este caso, corresponde a $file\_np \times rs(MPI) = 1 \times 8388608$ Bytes, donde $rs(MPI)$ es el tamaño de petición de las operaciones MPI. Este comportamiento es el esperado porque se han seleccionado los parámetros de IOR para observar el comportamiento de las colectivas a nivel de POSIX-IO.

Finalmente, se puede concluir que PIOM-PX detecta el patrón espacial y temporal para el nivel de POSIX-IO y estos patrones se representan mediante fases de E/S. Los resultados mostrados del patrón espacial dependen de la operación de E/S y del método de E/S. Se ha seleccionado la misma cantidad de datos, el número de procesos MPI y una estrategia de E/S similar, pero se ha comprobado que el comportamiento está influenciado por las técnicas y la interfaz de E/S. Se ha detectado que la técnica de las colectivas no funcionaba en modo automático porque se esperaba que sólo un proceso MPI realizara E/S por nodo de cómputo y se ha comprobado que todos los procesos de MPI estaban realizando E/S. En los experimentos (c) y (d), las operaciones MPI-IO son las mismas, pero a nivel de POSIX-IO ellas dependen del valor de los *hints* de la libreria ROMIO y de los *hints* explicitamente definidos en la aplicación. PIOM-PX considera esta propiedad para proporcionar información al usuario y ayudarlo a entender que está haciendo la libreria de E/S con las operaciones definidas en la lógica de la aplicación.

3.4. Conclusiones

PIOM-PX es un modelo de comportamiento de E/S que permite extraer información de la aplicación a nivel de la interfaz POSIX-IO. El análisis a este nivel permite comprender qué cantidad de datos son transferidos en cada acceso al fichero, o cuantos accesos son realizados para transferir la cantidad de datos que el usuario ha definido a nivel de aplicación. Sin embargo, es importante establecer una correlación entre los datos que quiere enviar el usuario en cada ráfaga de E/S y los datos que son enviados en cada acceso al fichero. Establecer esta correlación no es obvia, ya que en cada nivel de la pila de sofware los datos de la aplicación son tratados de forma diferente, debido a la configuración del sistema de E/S: variables de entorno, sistema de ficheros, etc.

Al analizar una aplicación a nivel de POSIX-IO, hay que tener en cuenta que este nivel es el más cercano al sistema de ficheros; con lo cual, la dependencia con el sistema se incrementa. Como uno de los objetivos marcados es conseguir un modelo de la aplicación, independiente del sistema donde se realiza el análisis (un modelo portable), se deben detectar los datos que son dependientes del sistema y establecer una relación entre una o más características de la aplicación para desvincular los datos del sistema.

Finalmente, la integración del modelo para las aplicaciones paralelas que usan librerías de E/S serie (PIOM-PX) al modelo para las aplicaciones paralelas que usan librerías de E/S paralelas (PIOM-MP), permite conseguir nuestro principal objetivo que consiste en tener un modelo holístico portable (PIOM) para la E/S de las aplicaciones paralelas MPI. Este modelo holístico permite tener cubierta el análisis de la mayoría de las llamadas de E/S que se realizan contra el sistema de ficheros global.

El análisis del modelo de comportamiento del sistema de E/S, PIOM, se realiza a nivel de fichero (a nivel de vista lógica del fichero). PIOM se basa en el concepto de fase de E/S. Entendiendo como fase de E/S una secuencia de operaciones consecutivas de E/S a un fichero. Identificar las fases de E/S, que contiene un fichero, permite analizar el impacto que tiene cada fase de E/S sobre el sistema HPC.

4

PIOM a nivel de POSIX-IO

Una vez descrito el modelo, en este capítulo se describe la herramienta para trazar la aplicación a nivel de POSIX-IO y los algoritmos para identificar las fases de E/S, con el objetivo de extraer el modelo de comportamiento de E/S a nivel de POSIX-IO (PIOM-PX). Este trabajo se presentó en [34].

4.1. Introducción

Las aplicaciones paralelas analizadas son aplicaciones científicas MPI, escritas en C y Fortran. Por lo tanto, los eventos que se interceptan están comprendidos entre las instrucciones `MPI_Init()` y `MPI_Finalize()`.

Para trazar las aplicaciones a nivel POSIX-IO se ha implementado un trazador, PIOM-PX-Trace. Este trazador junto a la herramienta que se desarrolló para trazar las aplicaciones a nivel de MPI, conformarán la herramienta $PIOM-Trace$. Herramienta neceseria para extraer el modelo PIOM de las aplicaciones paralelas, tanto las que usan librerías de E/S paralela (MPI-IO) y las que usan librerías de E/S serie (POSIX-IO); permitiendo, también, analizar el comportamiento a nivel POSIX de las aplicaciones que utilizan MPI-IO.

La herramienta $PIOM-Trace$ es un trazador modular con el objetivo de facilitar su modificación y permitir añadir nuevos módulos, si fuera necesario. $PIOM-Trace$ permite interceptar las operaciones de E/S a dos niveles de la pila software de E/S: MPI y POSIX-IO, proporcionando información útil para entender como está siendo tratada la aplicación en estos dos niveles de la pila. $PIOM-trace$ tiene tres opciones de ejecución, se pueden trazar:

- sólo eventos MPI
- sólo eventos POSIX-IO
- eventos MPI y eventos POSIX-IO

La caracterización de la aplicación se hace de forma dinámica. Para interceptar las operaciones de E/S de una aplicación se ha de cargar la herramienta, $PIOM-trace$, con el siguiente comando

export LD_PRELOAD=ruta_libreria/libpiom.so

Esto permite que mientras la aplicación se está ejecutando, la herramienta intercepta los eventos MPI y/o POSIX-IO que genera la aplicación sin interferir en su ejecución.

4.2. Herramienta PIOM-PX-Trace

En este apartado, se describirá la implementación de la herramienta PIOM-PX-Trace necesaria para interceptar los eventos de E/S a nivel de POSIX-IO. En relación a las operaciones POSIX-IO

que se interceptan, en la siguiente lista se describen las operaciones que se tienen en cuenta para la caracterización de la aplicación. La elección de interceptar estas operaciones se debe a que son las operaciones más comunmente utilizadas en las aplicaciones. Estas operaciones son:

```
open, open64, fopen64, close, fclose, creat, creat64, read,
write, pread, pwrite, pread64, pwrite64, readv, writev,
fread, fwrite, lseek, lseek64, fseek, __xstat, __xstat64,
mmap, mmpa64, __lxstat, __lxstat64, __fxstat, __fxstat64,
fdatasync, aio_read, aio_write, aio_read64, aio_write64,
aio_return, aio_return64.
```

Cada vez que se intercepta un evento MPI, de comunicación o POSIX-IO se registra una línea de traza (TL) con la estructura de la Tabla 4.1 en el fichero de traza, los tiempos físicos y los contadores lógicos actualizados. En la captura de estos eventos/operaciones, se extraen los valores de los parámetros de la llamada a la función. En este punto, se actualizan los contadores *tick* y *subtick*. El *tick* se incrementa cada vez que se intercepta un evento MPI y esto provoca que el *subtick* se inicialice cada vez que se incrementa el *tick*. El *subtick* se incrementa siempre que se intercepten operaciones POSIX-IO consecutivas. Sin embargo, se requiere un post-procesamiento para ordenar lógicamente los ticks de los eventos [35].

Tabla 4.1: Estructura de una línea de traza de PIOM-PX (PIOM-PX TL)

Identifier	**Description**
IdProcess	Identifier of Process.
file_id	Identifier of File.
TypeOperation	'MPI' or 'POSIX'.
NameOperation	Name of POSIX-IO operation.
offset	Operation offset, which is a position in the file's logical view.
rs	Request size of for data access operations.
Metadata-Line	
file_name	File Name.
FileAccessType	Open mode.
Added Fields	
Time	Logical time of the occurrence of a MPI or POSIX-IO event.
Final_compute	Duration of the call of an MPI or POSIX-IO event.
tick	Order of occurrence of the MPI events.
subtick	Order of occurrence of the POSIX-IO events.

Al finalizar este paso se obtendrán los ficheros de traza. Un fichero de traza por cada proceso MPI.

4.3. Extracción de operaciones de E/S por fichero

Todos los ficheros de traza son analizados para identificar los ficheros utilizados por la aplicación y sus operaciones de E/S. Es decir, en los ficheros de traza se identifican todas las operaciones de E/S que pertenecen a cada uno de los ficheros utilizados por la aplicación. Estos últimos ficheros son identificados con la tupla `(file_id, file_name)` y se crea un *I/O File* por cada fichero utilizado por la aplicación. Este análisis y los posteriores se hacen fuera de línea (*offline*).

La información de los ficheros *I/O file* es lo que realmente se analiza para extraer el patrón espacial y temporal. En estos ficheros, se puede observar el número de procesos que acceden al fichero utilizado, los eventos que los procesos han utilizado para acceder al fichero y el orden de los eventos.

4.4. Actualización campos de las operaciones de E/S

En este punto se analiza cada línea registrada en cada *I/O File* que no tengan informado los valores de todos los campos.

El análisis de varias operaciones se refiere a operaciones de E/S en la que el $offset$ y el tamaño de petición (*rs*) requieren la evaluación de otras operaciones para obtener esos valores. Por ejemplo, en el caso de las operaciones `write` y `read` dependen de la evaluación de la operación `lseek` para obtener el valor de su $offset$. Para modificar el $offset$ se ha de tener en cuenta el parámetro *whence* de la operación `lseek`

- *off_t lseek(int fd, off_t offset, int whence);*;

donde *whence* toma los valores: $SEEK\_SET$ (corresponde a la posición donde se ha de ubicar la operación en bytes), $SEEK\_CUR$ (la posición en que se ubica la operación se calcula a partir de la posición actual más el offset en bytes) y $SEEK\_END$ (en este caso, la posición se calcula considerando el tamaño del fichero más el offset en bytes) [39, 40] (ver Figura 4.1).

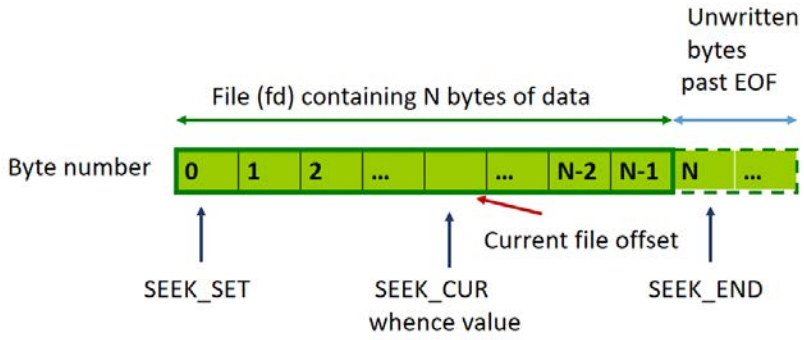


Figura 4.1: Representación del parámetro $WHENCE$ de la instrucción *lseek*

Se puede tomar, como ejemplo, el caso de la instrucción `write`:

- write(): *ssize_t write(int fd, void *buffer, size_t count)*;
- pwrite(): *ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset)*.

Como se puede observar la instrucción `pwrite()` tiene explicito el $offset$ en la llamada de la función mientras que la instrucción `write()`, no. Para la instrucción `write()` se necesita evaluar la operación `lseek`.

Para identificar el tamaño de la petición (*rs*) y como el desplazamiento se mueve, se agrega un nuevo campo desplazamiento (*disp*) a la estructura de la línea de traza (TL) que está registrada en I/O File.

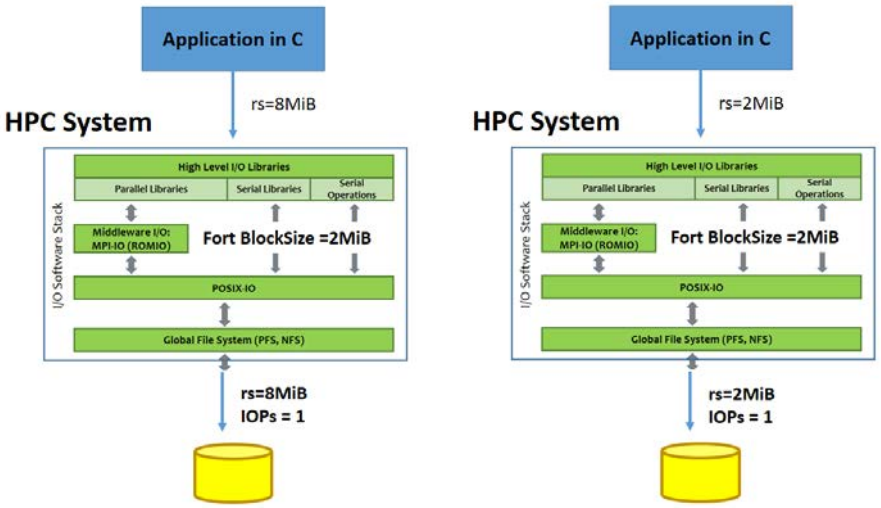
En los programas escritos en Fortran se evalúa la varible de entorno `FORT_BLOCKSIZE` para determinar el tamaño de la petición de los eventos POSIX con la que el usuario quiere trabajar, en realidad. En la Figura 4.2, se muestra que al analizar las aplicaciones paralelas se ha de tener en cuenta la variable de entorno $FORT\_BLOCKSIZE$ cuando la aplicación está escrita en Fortran. Ya que a nivel de POSIX, la información que se obtiene difiere de los datos transferidos por la aplicación. Sin embargo, en C, independientemente del tamaño de la petición y del valor de la variable $FORT\_BLOCKSIZE$, el valor del tamaño de petición que se puede observar a nivel de aplicación es el mismo que se observa a nivel POSIX-IO.

4.5. Descripción del patrón espacial

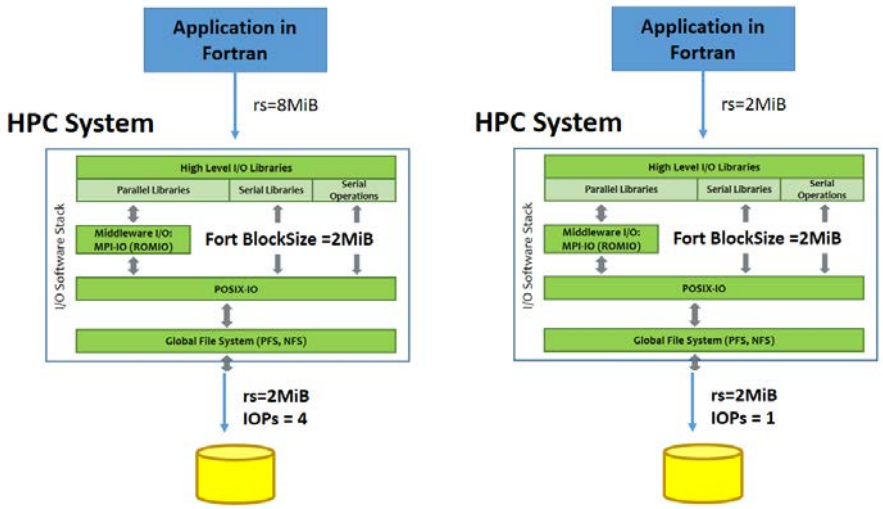
Para extraer el patrón espacial de la aplicación, de nuevo, se ha de realizar un análisis de cada *I/O File*. Al iniciar este análisis, tanto para la escritura como para la lectura, se guardan los campos: `NameOperation, file_id, file_name, offset, rs, tick` and `subtick`.

Se calcula la diferencia de $offset$ para todas las operaciones que tienen el mismo $file\_id$, $file\_name$ y *rs*. La diferencia del offset es calculada entre dos operaciones consecutivas de lectura o escritura y el desplazamiento (*disp*) es calculado entre dos operaciones de escritura y lectura.

La operaciones de E/S detectadas, que se consideran consecutivas y pertenecen a la misma fase, se guardan hasta detectar un *tick* diferente, ya que se quiere identificar si existen secuencias repetidas. El conjunto de operaciones guardadas corresponderá a la primera secuencia identificada. El detectar un *tick* diferente indica que se ha producido un evento MPI y también indica el comienzo de una nueva secuencia o la detección de una repetición de la secuencia detectada anteriormente. Para saber si es un caso u otro, de cada nueva operación detectada se compara el $offset$ y el *rs* para ver si son similares a las operaciones de la primera o anteriores secuencias detectadas. Si se detecta una secuencia con las mismas operaciones, con offset y rs similar y en el mismo orden de aparición, que en una secuencia anterior, se incrementa un contandor rep. Este contador nos indica el número de repeticiones en la que



(a) El tamaño de la petición observado a nivel de aplicación y a nivel de POSIX-IO es el mismo cuando las aplicaciones están escritas en C

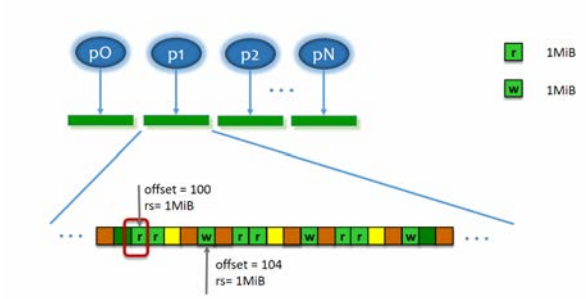


(b) El tamaño de la petición observado a nivel de aplicación y a nivel de POSIX-IO no es el mismo si FORT_BLOCKSIZE <*rs* cuando las aplicaciones están escritas en Fortran.

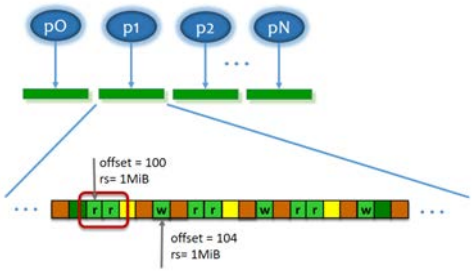
Figura 4.2: Influencia del valor de la variable entorno FORT_BLOCKSIZE sobre las aplicaciones escritas en C y Fortran.

aparece una determinada secuencia en un fichero. Si esa secuencia no corresponde a ninguna de las secuencias detectadas, se detecta una nueva secuencia. Al finalizar el análisis, cada secuencia diferente detectada es una fase de E/S. En la Figura 4.3 se refleja gráficamente como se detectaría una fase de

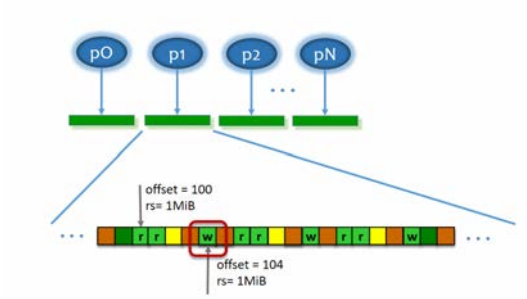
E/S, en líneas generales.



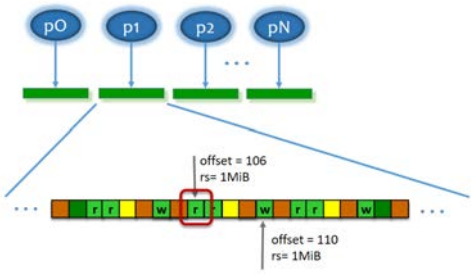
(a) Detección de la primera operación POSIX-IO.



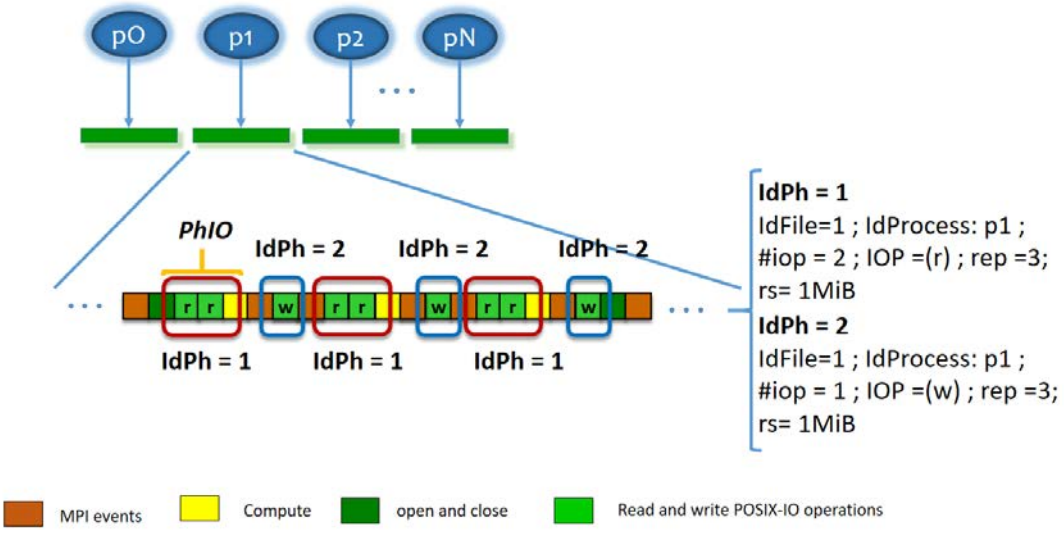
(b) Detección de la siguiente operación POSIX-IO.



(c) Detección de una nueva operación POSIX-IO.



(d) Detección de una operación POSIX-IO similar a 4.3(a). Se analiza el offset y el rs.



(e) Fases detectadas Ph_id para un fichero file_id.

Figura 4.3: Detección de todas las fases de E/S de un 1 fichero por proceso.

4.6. Extracción del patrón temporal

Se identifica el patrón temporal haciendo uso de las unidades del contador lógico del *tick* (para eventos MPI) y del *subtick* (para eventos POSIX-IO). Cuando se detecta un nuevo evento MPI u operación MPI-IO, el subtick se inicializa y se incrementa siempre que existan operaciones POSIX-IO consecutivas.

Si los procesos se intercambian mensajes `MPI_Sends`, en los procesos receptores (los que reciben `MPI_Receives`) se debe revisar el *tick*, pero no el *subtick*, y actualizarlo. Porque, dependerá del valor *tick* del `MPI_Send`.

La Figura 4.3 es un ejemplo de la descripción de un patrón temporal de una fase.

4.7. Identificación de las fases más significativas

Como el modelo está definido en base a las fases de E/S se puede analizar cada fase por separado y determinar que fases pueden tener más impacto en la aplicación. Es decir, se pueden identificar las fases más significativas.

Existen varios criterios para determinar que fases son más significativas. Estos son:

- Porcentaje que representa el tiempo de I/O en una ejecución.
- Fichero que el usuario quiere analizar.
- Fases con mayor movimiento de datos. Esto depende tanto del número de operaciones como del tamaño de las operaciones.

Una vez obtenido el modelo e identificado las fases más significativas, se puede personalizar un programa sintético con los parámetros del modelo para replicar el comportamiento de E/S que se quiere analizar de la aplicación. Este programa sintético se puede ejecutar en otro sistema HPC o con otra configuración de E/S, sin necesidad de ejecutar la aplicación real. Por ejemplo, para poder evaluar el rendimiento que se obtiene de la aplicación en otro sistema HPC o con una nueva configuración de E/S.

4.8. Validación Experimental

Seguidamente, se muestran los resultados obtenidos al aplicar la herramienta *PIOM-Trace*. Se han seleccionado diferentes benchmarks tales como MADBench2 para extraer el modelo de comportamiento de E/S, y se ha utilizado la herramienta para obtener el modelo en diferentes sistemas, con el objetivo de mostrar que la información obtenida es independiente del sistema.

Sistemas HPC

Además de los sistemas HPC descritos en la Tabla 3.2, se añadieron dos sistemas HPC más para realizar los experimentos mostrados en la Tabla 4.2.

Tabla 4.2: Sistemas HPC

Components	**Capita (HPC4EAS)**	**CoolMUC2 (LRZ)**
Compute Nodes	13	384
CPU cores (per node)	4	28
RAM Memory	16GB	64GB
Local Filesystem	Linux ext3	Linux ext3
Global Filesystem (GFS)	NFS	NFS
Capacity of GFS	46GB	1000TB
Global Filesystem (PFS)	PVFS2 (2.8.7)	GPFS
Capacity of PFS	175GB	1.3PB
MPI-Library	mpich2 (1.5)	Intel MPI
Number of data servers	4	4 NSD
Number of Metadata Server	1	
Stripe Size	64KiB	8MiB
Interconnection	1 Gbps Ethernet	Infiniband FDR

Para esta ocasión, se trabaja con los valores por defecto de los sistemas HPC. Por esta razón, el sistema HPC, Finisterrae 2, tendrá como 1 Data Server(DS), mientras que en el capítulo anterior, el número de Data Server eran 4 OSS y 12 OST.

4.8.1. Resultados para MADbench2

En esta sección se muestra la extracción del comportamiento de la E/S para el benchmark MADbench2 [57]. MADbench2 es una herramienta para probar el rendimiento global integrado de los sistemas de E/S, los sistemas de comunicación y de cómputo de arquitecturas altamente paralelas bajo el estres de aplicaciones científicas reales. MADbench2 está escrito en C y MPI; y, puede ser ejecutado en modo de E/S, en el cual todas los cálculos/comunicaciones son reemplazados por *busy-work*. Ejecutar MADbench2 requiere un número n^2 de procesadores. Una descripción detallada de los diferentes parámetros se presenta en [41].

La estrategia de la E/S seleccionada en esta ocasión es de 1 fichero por proceso. Y el análisis se ha centrado en el modo POSIX. Se ha extraído el modelo de comportamiento de la E/S de MADbench2, el cual tenía los siguientes valores: IOMETHOD=POSIX, BWEX=1.0, RMOD=WMOD=1, FILETYPE=UNIQUE, NO_BIN=8 y KPIX=25; y FBLOCKSIZE es ajustado al valor por defecto del

StripeSize que tenga el sistema. En el caso del sistema HPC del LRZ se ajusta a 8 MiB para GPFS y a 1 MiB para Lustre en Finisterrae2.

Al analizar los ficheros de traza se identificaron 5 fases de E/S por cada fichero de la aplicación paralela. En la tabla 4.3 se presentan los parámetros de E/S a nivel de fase.

Tabla 4.3: Fases de MADbench2 usando 16 MPI procesos, 25KPIX y 8 NO_BIN.

Concepts	Phases for $\forall\ File_i\ with\ i \in (0..app\_np-1)$				
Ph_id	Ph1	Ph2	Ph3	Ph4	Ph5
file_id	$File_i$	$File_i$	$File_i$	$File_i$	$File_i$
Ph_processid	i	i	i	i	i
Ph_np	1	1	1	1	1
rep	1	1	1	1	1
Ph_niop	8	2	12	2	8
IOP_type	w	r	(w,r)	w	r
rs	312500000				
Ph_weight	2.32GiB	596MiB	3.5GiB	596MiB	2.32GiB
offset	312500000				
disp	312500000				

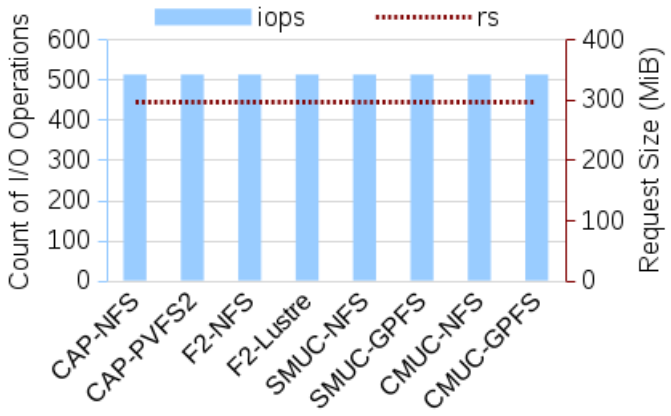


Figura 4.4: MadBench2 usando 128 MPI procesos. Se muestra el contador total (*Ph_niop*) de las operaciones de E/S y el tamaño de la petición (rs) en diferentes sistemas HPC y sistemas de ficheros.

En la figura 4.4, se presenta un total de 512 operaciones de POSIX-IO que corresponden a las operaciones `fwrite` y `fread`, y se muestra que el tamaño de la petición es 298 MiB para todas las operaciones. Se puede observar que estos valores son independientes del sistema de E/S subyacente.

En la tabla 4.4 se detallan los parámetros a nivel de aplicación y fichero y, los valores obtenidos usando diferentes configuraciones. En los cuatro sistemas HPC se obtienen los mismos valores. Los valores mostrados a partir de $app\_np = 25$ son obtenidos en los sistemas SuperMUC y CoolMUC2.

Los resultados corresponden a diferentes números de procesos para las cargas de trabajo 25KPIX y 90KPIX. Los valores a partir de 64 procesos no son obtenidos en el cluster CAPITA.

Tabla 4.4: MADbench2 - Características de la aplicación - AccessType = 1 File x Process - AccessMode = sequential - FileAccessType = W/R - $file\_np$ =1.

	Application Concepts			File Concepts		Phase Concepts
	app_np	app_nfiles	app_st (GiB)	$file\_size$ (Bytes)	file_nphase	rs (Bytes)
$KPIX = 25$						
	16	16	37.25	2500000000	5	2500000000
	25	25	37.25	1600000000	5	1600000000
	64	64	37.25	625000000	5	625000000
	100	100	37.25	400000000	5	400000000
	400	400	37.25	100000000	5	100000000
$KPIX = 90$						
	256	256	965.60	4050000000	5	4050000000
	400	400	965.60	2592000000	5	2592000000
	576	576	965.60	1800000000	5	1800000000

Las fases de E/S de MADBench2 son similares usando diferente número de procesos MPI y carga de trabajo. Sin embargo, para hacer una descripción de la fase se han considerado las siguientes ecuaciones para *Ph_niop*:

- Phase 1: *Ph_niop* = NO_BIN
- Phase 2: *Ph_niop* = 2
- Phase 3: *Ph_niop* = NO_BIN - 4
- Phase 4: *Ph_niop* = 2
- Phase 5: *Ph_niop* = NO_BIN

No obstante, *rs* depende de NO_PIX y app_np. Para obtener otros parámetros de E/S a nivel de fase, se necesita considerar los valores presentados en Table 4.4. Esto presenta diferente *rs* usando diferente NO_PIX y app_np. Se puede observar que el modelo de comportamiento de E/S de MADBench2 es similar para diferentes cargas de trabajo y procesos MPI.

4.8.2. Resultados para ABySS

ABySS [58] es un ensamblador de secuencias paralelas que se especializa en ensamblar grandes genomas de secuencias cortas, utilizando el método de pares finales (adelante y atrás). El algoritmo

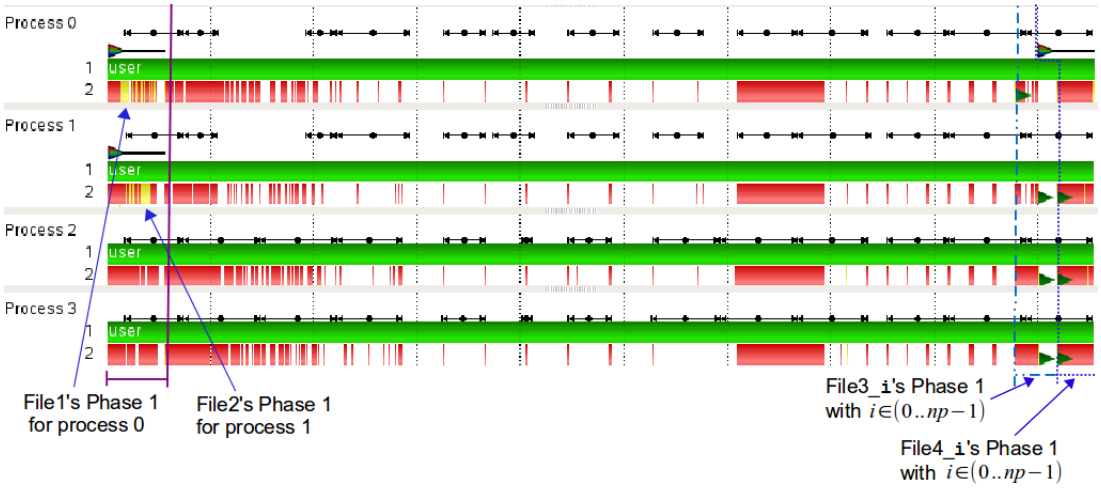


Figura 4.5: Traza de Vampir para ABYSS-P usando 4 MPI procesos. Los triángulos verde oscuro y rojo corresponden a múltiples eventos de E/S. El color amarillo corresponde al tiempo de las operaciones de E/S. Las áreas con color verde corresponden al tiempo de las operaciones de usuario y los de color rojo al tiempo de los eventos de MPI. Se pueden observar dos ráfagas de E/S: al principio, cuando los procesos 0 y 1 están leyendo desde el File1 y File2, respectivamente; al final, cada proceso MPI escribe dos ficheros temporales (File3_i and File4_i) por proceso. ABYSS-P accede a un total de $2 + np \times 2$ files.

ABySS se desarrolla en dos etapas. ABYSS-P es una versión MPI del ensamblador de gráficos Bruijn, que corresponde a la primera fase.

Se ha extraído el modelo de comportamiento de la E/S de ABYSS-P en los sistemas CoolMUC2, SuperMUC y Finisterrae2.

En la figura 4.5 se muestra un pequeño test hecho para ABySS utilizando 4 procesos MPI al ser trazado con VampirTrace. Para ensamblar pares se lee de dos ficheros llamados `reads1.fastq` y `reads2.fastq` y uno llamado `test-bubbles.fa`; k=25. Este pequeño caso permite identificar el orden temporal de las fases de E/S. Al principio de la ejecución se puede observar una fase de E/S para dos ficheros de entrada y otra fase de E/S para dos ficheros temporales por cada proceso MPI al final de la ejecución.

En este caso, para los parámetros de la aplicación se han utilizado dos ficheros de entrada y $app\_np * 2$ ficheros de salida. Por ejemplo, si la aplicación se ejecuta usando 128 procesos se obtienen 256 ficheros de salida.

Pero, independientemente del número de procesos, sólo los procesos 1 y 0 leen el fichero1 y fichero2 respectivamente. El tamaño del fichero para cada uno es 17GiB, el tipo de acceso es secuencial y el tipo de acceso al fichero es de sólo lectura. El tamaño de los ficheros temporales es menor que 1MiB en total. Por lo tanto con respecto a los parámetros del modelo, sólo se tienen en cuenta los ficheros de entrada porque los ficheros temporales son muy pequeños y tienen poco impacto en el sistema de E/S. En los ficheros de entrada se han identificado, a nivel POSIX, operaciones de *read* y

seek.

En la tabla 4.5, se observó que la aplicación ejecuta 2135744 operaciones de E/S (*Ph_niops*) en cada fichero de entrada y 8191 Bytes es el tamaño de petición (*rs*).

Independientemente del sistema HPC y el sistema de ficheros, cuando ABYSS-P ha sido ejecutada, se tiene el mismo valor para el contador de las operaciones de E/S (*Ph_niops*) y el tamaño de la petición (rs) (Ver Figure 4.6). Tabla 4.5 muestra los parámetros a nivel de fase para los dos ficheros de entrada. ABYSS-P tiene las mismas fases de E/S para cada fichero de entrada y los valores obtenidos son iguales en los tres sistemas HPC utilizados.

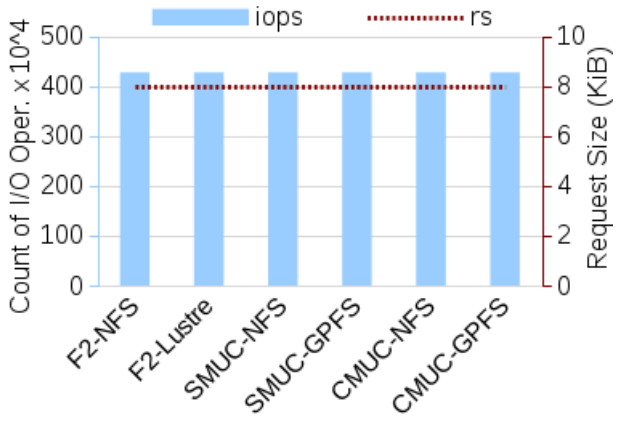


Figura 4.6: ABySS usando 128 MPI procesos.Se proporciona el contador total de operaciones de E/S (*Ph_niops*) y el tamaño de la petición (rs) en diferentes sistemas HPC y sistemas de ficheros

Tabla 4.5: Fases de ABySS para los ficheros de entrada.

Concepts	Phases	
Ph_id	Ph1	Ph1
file_id	File1	File2
Ph_processid	0	1
Ph_np	1	1
Ph_niop	2135745	2135745
IOP_type	r	r
rs	8191	8191
rep	1	1
Ph_weight	17GiB	17GiB
offset	8191	8191
disp	8191	8191

4.9. Conclusiones

Analizar las aplicaciones científicas y extraer el comportamiento de E/S de una aplicación es de gran utilidad para entender de que manera una aplicación influye sobre el sistema de E/S y, analizar cuando el sistema de E/S ofrece un rendimiento pobre cuando se está ejecutando una determinada aplicación. Obtener conocimiento de la aplicación a nivel POSIX-IO permite identificar que impacto tiene ésta sobre el sistema de E/S, ya que este nivel es el que está relacionado directamente con el sistema de ficheros.

Definir el modelo de comportamiento de E/S de una aplicación a nivel POSIX-IO permite poder analizar las aplicaciones paralelas que utilizan librerias de E/S serie y, además, profundizar en el análisis de las aplicaciones paralelas que utilizan librerias de E/S paralela. En la definición de PIOM-PX se ha tenido que determinar como se identificaban las fases a este nivel, si existen operaciones MPI serán las que acotarán las fases de E/S a nivel de POSIX. En el caso, que existan largas ráfagas de cómputo entre operaciones POSIX, la identificación de las fases de E/S viene determinado por el número de eventos de cómputo que se generan entre los eventos POSIX.

PIOM-PX introduce el concepto *subtick*, el cual permite establecer el orden lógico de los eventos POSIX. Al integrar PIOM-PX con PIOM-MP, el subtick también permite establecer la dependencia/relación de los eventos POSIX con los eventos MPI generados por la aplicación.

La definición de PIOM-PX y su integración con PIOM-MP ha dado lugar al modelo PIOM, el cual, permite analizar las aplicaciones a dos niveles: a nivel POSIX y/o a nivel MPI. Pudiendo seleccionar si sólo se quiere analizar a nivel POSIX, MPI-IO o a nivel POSIX+MPI-IO. En los experimentos realizados se ha visto la utilidad de obtener conocimiento en dos niveles de la pila software de E/S, siendo el nivel POSIX, por el que pasan los datos de todas las aplicaciones al sistema de ficheros.

5

Descripción funcional

En este capítulo se explica el diseño realizado para extraer el modelo PIOM, se presenta la extracción del modelo a nivel POSIX y su integración con PIOM-MP para dar una visión global de ambas capas.

5.1. Introducción

Para obtener el modelo PIOM se define una metodología para adquirir los valores de las características de E/S (ver Tabla 3.1) de una aplicación. Es decir, mediante la aplicación de esta metodología se obtendrá el modelo de comportamiento de E/S de cada una de las aplicaciones analizadas. Permitiendo descubrir la utilidad de PIOM para entender como se tratan los datos de la aplicación, realmente, a nivel MPI y a nivel POSIX-IO.

5.2. Metodología

La metodología propuesta para realizar el análisis, en términos generales, consta de cuatro pasos (ver Figura 5.1).

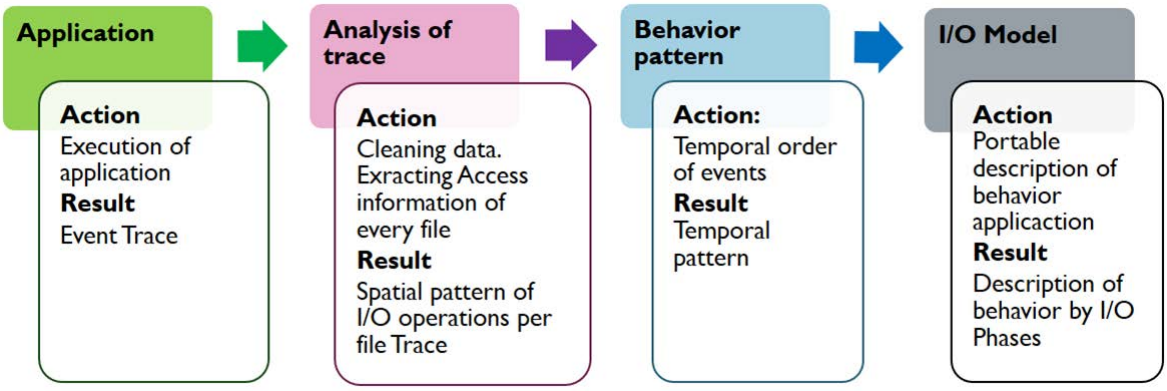


Figura 5.1: Metodología de análisis para obtener el modelo de comportamiento de la E/S de una aplicación paralela.

El objetivo de esta metodología es identificar los parámetros que aporten información relevante de la aplicación, y a partir de esta información, se pueda replicar el comportamiento más significativo de la aplicación. Los valores de los parámetros dependerán del sistema HPC donde se haya ejecutado la aplicación paralela. Con esta metodología, también, se quiere establecer la relación entre los valores obtenidos en las diferentes capas, de la pila de software, para obtener información relativa al comportamiento de la aplicación en los sistemas. Obtener esta información también facilita entender porque una aplicación tiene o puede tener un problema de E/S cuando se ejecuta en un determinado sistema HPC.

En la metodología propuesta se pueden diferenciar dos etapas: la ejecución de la aplicación en un sistema HPC específico (que corresponde al paso *Application*) y, un post-procesamiento realizado fuera de línea (*off-line*) (correspondiente a *Analysis of trace*, *Behavior pattern* y *I/O model*). La figura 5.2 muestra un esquema de la metodología con más detalle:

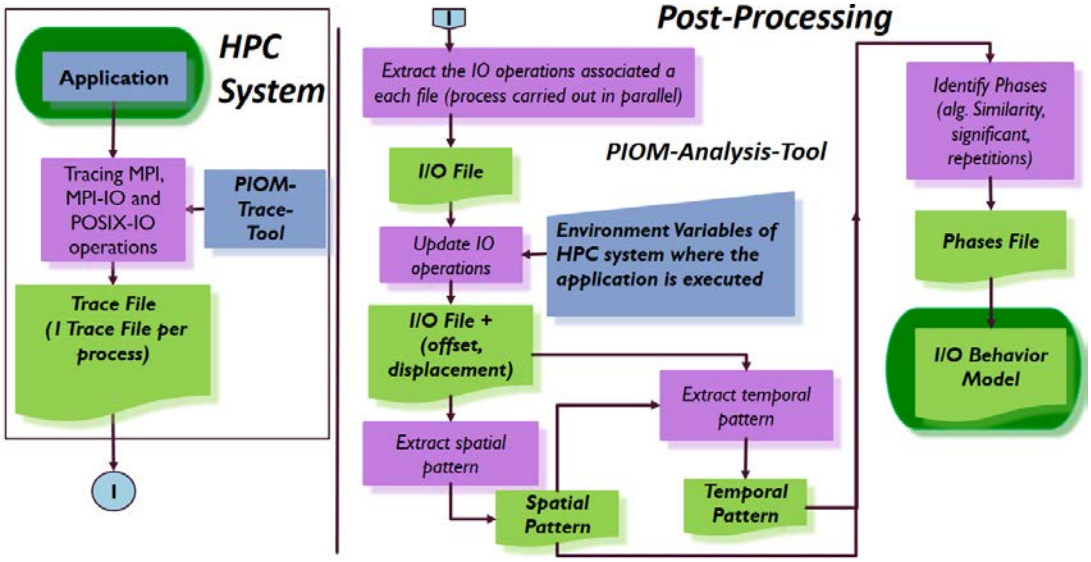


Figura 5.2: Módulos del Pre-procesamiento (PIOM-Trace-Tool) y del Post-Procesamiento (PIOM-Analysis-Tool)

5.2.1. Trazar Aplicación

El análisis consiste en aplicar una metodología que nos permita identificar dónde, cuándo, cómo y el porqué se produce un problema de E/S. Por lo tanto, el primer paso es trazar la aplicación para interceptar las operaciones de E/S y conocer el patrón de acceso o patrones de acceso que está utilizando la aplicación. Las trazas ofrecen una serie temporal de eventos que se pueden procesar fuera de línea.

Los tipos de instrucciones que se interceptan son:

- a nivel de MPI: operaciones de E/S de MPI-IO y operaciones de comunicación MPI,
- a nivel de POSIX-IO: operaciones de E/S.

A nivel de operación de E/S interesa conocer que proceso (*Ph_processid*) ha accedido al fichero, a que fichero (*file_id*, *file_name*) se ha accedido y cómo (*file_fileaccesstype*), qué tipo de operación (*IOP_type*) ha realizado el proceso cuando ha accedido al fichero, en qué punto (*offset*) del fichero ha realizado esa operación y que volumen de datos (*rs*) se ha transferido cuando se ha hecho la

operación. Cuando se intercepta la instrucción correspondiente se registra, en el fichero de traza, el *IdProcess* (equivalente con *Ph_processid*), *file_id*, *file_name*, *FileAccessType* (correspondiente al *file_fileaccesstype*), *NameOperation* (correspondiente a *IOP_type*), *offset* y *rs*. Para controlar el orden lógico en el que se han realizado las operaciones, se añaden los campos *Time*, *final_compute*, *tick*, *subtick*. El campo *TypeOperation* se añade para identicar de que tipo "MPI" o "POSIX-IO" es la operación. La estructura de la linea que se registra, en el fichero de traza, se describe en el capítulo **??**, Tabla 4.1.

Se obtienen dos ficheros de traza por proceso MPI, porque se ha dividido la información dependiendo del tipo de operaciones interceptadas: MPI *(*.tmp_piom-mpi)* o POSIX *(*.tmp_piom-posix)*. En uno se guardan los eventos MPI de comunicación y las operaciones de E/S de MPI-IO, y en el otro se registran las operaciones de E/S interceptadas de POSIX-IO.

En este punto, al interceptar las instrucciones se podría considerar que se introduce *overhead* en la ejecución de la aplicación. Sin embargo, el método para interceptar las intrucciones, que se ha planteado en este trabajo de investigación, no añade un *overhead* considerable ya que no se interfiere en la ejecución de la aplicación. En contra, se ha de tener en cuenta si los recursos de almacenamiento que se tienen son suficientes porque, cuando se interceptan las instrucciones, se generan ficheros de traza y éstos necesitan ser almacenados para su posterior análisis. El número de ficheros de traza puede ser un número considerable debido a que depende del número de procesos que se utilizan en la ejecución de la aplicación.

5.2.2. Extracción de operaciones de E/S por fichero

Después de trazar la aplicación se hace un post-procesamiento de la información registrada en los ficheros de traza. Este post-procesamiento se realiza fuera de linea (*off-line*) y, se le asigna el nombre *PIOM-Analysis-Tool*. La figura 5.2 muestra los módulos implementados para realizar el post-procesamiento.

Los ficheros de traza son analizados para detectar los ficheros abiertos por la aplicación. A partir de esa detección se crea un nuevo fichero (*I/O File*) por cada fichero utilizado/abierto durante la ejecución de la aplicación; en el cual, se registran las operaciones de E/S de cada fichero de traza que acceden al fichero abierto. Ese nuevo fichero contiene infomación de los patrones de acceso al fichero realizados por un proceso, por un conjunto de procesos o por todos los procesos.

5.2.3. Actualización de operaciones de E/S

En esta etapa, se revisan los ficheros *I/O File* para identificar el valor de los parámetros del modelo. Pueden existir líneas de traza que no tienen informados algunos valores de los parámetros del modelo. Por lo tanto, a la hora de realizar el análisis de la traza, los parámetros del modelo pueden necesitar el

análisis de varias operaciones de E/S o limpiar la información que llevan dependiente del sistema en el que ha sido trazado. El objetivo de la 'limpieza" de los parámetros es para que el comportamiento de la aplicación sea reproducible y para poder tener un modelo portable.

5.2.4. Extracción del patrón espacial

Para extraer el patrón espacial de la aplicación se analizan los accesos de E/S que realiza la aplicación, se identifican qué operaciones está realizando la aplicación contra el fichero y a qué posiciones del fichero acceden esas operaciones. Para ello, se realiza un análisis de cada *I/O File*.

En este punto, se aplica un algoritmo de similitud que permita identificar cuando las operaciones se consideran similares para incluirlas o no en la misma fase de E/S y se actualiza el número de repeticiones de cada fase, para que se pueda replicar el movimiento de la misma cantidad de información.

5.2.5. Extracción del patrón temporal

Para detectar el patrón temporal, se definen como unidades de tiempo lógico el $tick$ y el $subtick$. El tick identifica el orden de los eventos de comunicación MPI y las operaciones MPI-IO, mientras que el subtick identifica el orden de las operaciones POSIX-IO.

Si todos los procesos escriben en uno o más ficheros compartidos, se debe detectar la relación entre todas las operaciones realizadas por todos los procesos para determinar el orden lógico actual. Este orden ayuda a detectar dependencias entre todos los procesos.

A partir del patrón espacial y temporal se analizan las operaciones consecutivas para detectar las fases de E/S. Una fase de E/S es una secuencia con las mismas operaciones consecutivas, con $offset$ y rs similar y en el mismo orden de aparición. A su vez, se contabiliza el número de apariciones (repeticiones) de esa fase de E/S en el fichero.

5.3. Resultados Experimentales

Para mostrar la parte experimental se han seleccionado los benchmark BT-IO y S3D-IO, y se ha aplicado la metodología descrita para analizar su comportamiento de E/S.

Resultados para BT-IO

.

BT-IO [42] es un benchmark generado a partir del problema *Block-Tridiagonal* (BT), el cual es parte de un benchmark paralelo que pertenece a la clase NPB-MPI desarrollada por NASA Advanced

Supercomputing Division. BT en MPI presenta una descomposición llamada multipartición diagonal en una matriz tridimensional a través de un número cuadrado de procesos. BT-IO es una extensión de BT en el que los datos son guardados en disco.

Se ha seleccionado BT-IO para mostrar el patrón temporal considerando los conceptos de tick y subtick. Como BT-IO tiene eventos de comunicación y de cómputo se muestra como se identifican las fases de E/S a partir de esos eventos. BT-IO está implementado en Fortran y permite evaluar la influencia de la libreria de E/S de Fortran en el tamaño de petición a nivel de POSIX-IO. Se ha seleccionado el subtipo FULL ya que para la E/S se usa MPI-IO con operaciones colectivas, tipos de datos derivado, `MPI_File_view` y `MPI_Info` para habilitar la técnica de las colectivas en la lógica de la aplicación. Se ha ejecutado BT-IO en los supercomputadores SuperMUC y Finisterrae2. Las características de estos supercomputadores están descritas en el capítulo 3 en la Tabla 3.2.

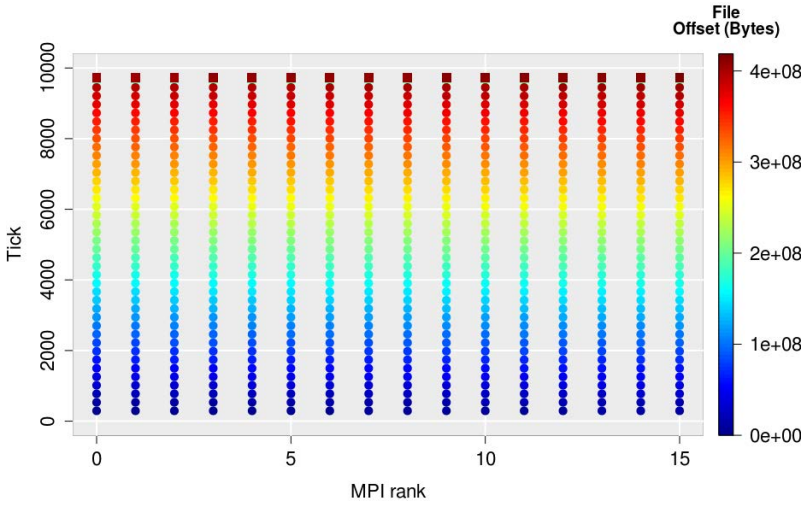
En la Figura 5.3 se puede apreciar las fases de E/S detectadas a nivel de MPI-IO (Fig.5.3(a)) y a nivel de POSIX-IO (ver Figura 5.3(b)) para la clase A de BT-IO. Los parámetros de PIOM-PX son descritos en la Tabla 5.1 para la clase A, B y C a nivel de aplicación y de fichero. La Tabla 5.1 refleja los valores obtenidos al hacer el análisis y se puede observar que se han identificado 41 fase de E/S en un fichero compartido con modo de acceso strided.

Tabla 5.1: Parámetros para el benchmark BT-IO subtipo FULL

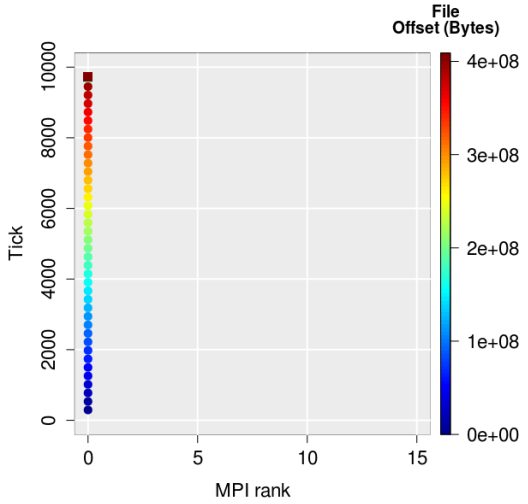
Identifier	**Class A**	**Class B**	**Class C**
app_np	16	16	36
app_nfiles	1	1	1
app_st	400 MiB	1.6 GiB	6.4 GiB
File			
file_name	btio.full.out	btio.full.out	btio.full.out
file_size	400 MiB	1.6 GiB	6.4 GiB
file_accessmode	Strided	Strided	Strided
file_fileaccesstype	W/R	W/R	W/R
file_accesstype	Shared	Shared	Shared
file_nphase	41	41	41
file_np at MPI-IO level	16	16	36
file_np at POSIX-IO level	1	1	3

En la Figura 5.3(b) cada círculo (y-axis) representa una fase de E/S compuesta de $file\_np$ operaciones de escritura. El cuadrado rojo representa la fase 41 (*Ph_Id*), el cual está compuesto de $40 \times file\_np$ operaciones de lectura. El tamaño de la operación es similar para los dos tipos de operaciones: lectura y escritura. A nivel de MPI-IO, Figura 5.3(a), el número de procesos MPI por fase de E/S corresponde a $file\_np$.

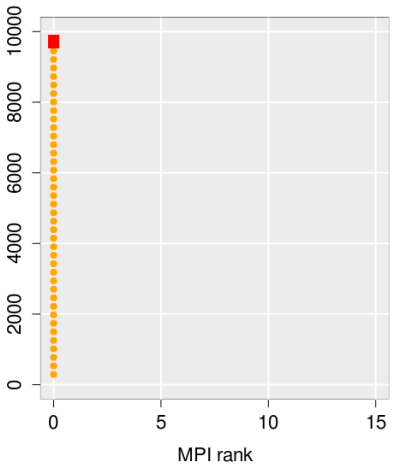
En la Figura 5.3(b), se puede observar el efecto de la técnica de las colectivas a nivel de POSIX. El



(a) Representación del $offset$ en el fichero a nivel MPI usando PIOM-MP



(b) Representación del $offset$ en el fichero a nivel de POSIX-IO



(c) Representación del peso de la fase a nivel POSIX-IO

Figura 5.3: BT-IO subtipo FULL, clase A, patrón strided y utilizando 16 procesos. Los círculos corresponden a operaciones de escritura y los cuadrados a operaciones de lectura. Las 40 primeras fases de E/S contienen, cada una, una operación MPI de escritura y la fase 41 contiene 40 operaciones de lectura. A nivel de MPI, el peso de la fase 1 hasta la fase 40 es 655360 Bytes $\times$ $file\_np$ para cada una y para la fase 41 es 10 MiB $\times$ np $\times$ 40. A nivel de POSIX, la escala coloreada en Figura 5.3(c) muestra el peso para la fase 1 hasta la fase 40, el cual es 10MiB y el peso de la fase 41 es 10 MiB $\times$40.

proceso 0 es el que realiza las operaciones de E/S. En esta capa, el número de procesos por fase de E/S es igual al número de nodos de cómputo utilizados para ejecutar la aplicación.

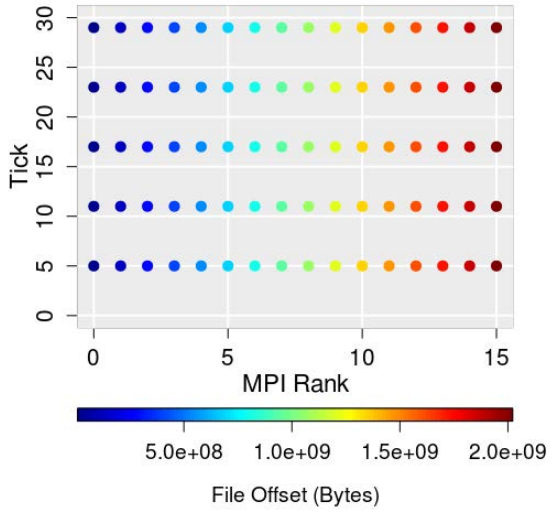
Resultados para S3D-IO

S3D-IO [43] es un benchmark para resolver en paralelo el problema de la combustión turbulenta que utiliza operaciones colectivas de E/S y usa NetCDF paralelo para *checkpointing*. El kernel S3D-IO genera un punto de control (*checkpointing*) a intervalos regulares y sus datos consisten en vectores tridimensionales de 8 Byte. Se genera un fichero por *checkpointing*, independientemente del número de procesos MPI utilizados. S3D-IO está escrito en Fortran y utiliza NetCDF paralelo para los *checkpointing*. Todos los procesos MPI comparten un fichero.

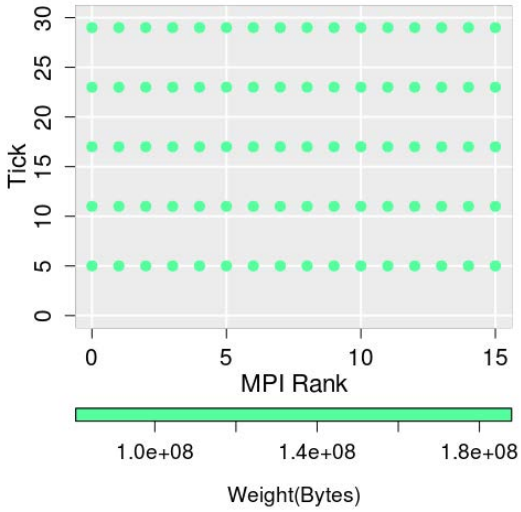
La selección de esta aplicación fue mostrar como se observa el comportamiento de E/S para una aplicación Fortran y como se observa un comportamiento diferente en MPI y en POSIX en función de las colectivas. En nuestro caso, S3D-IO fue ejecutado con 16 procesos y generó cinco ficheros de *checkpointing*. La Figura 5.5 muestra el comportamiento de E/S obtenido, de S3D-IO, con PIOM. A nivel de MPI-IO, se observa que todos los procesos acceden a los ficheros de *checkpointing* (ver Figura 5.4(a)) y en la Figura 5.4(b)) se observa la cantidad de volumen transferido en cada fase. Sin embargo, cuando se observa a nivel de POSIX se puede apreciar que sólo el proceso 0 accede a los ficheros de *checkpointing* (ver Figura 5.4(c)) y en la Figura 5.4(d) se muestra el volumen de datos transferido en cada fase de E/S. Analizando, los ficheros de traza se detecta que el resto de procesos sólo abren y cierran los ficheros de *checkpointing*.

5.4. Conclusiones

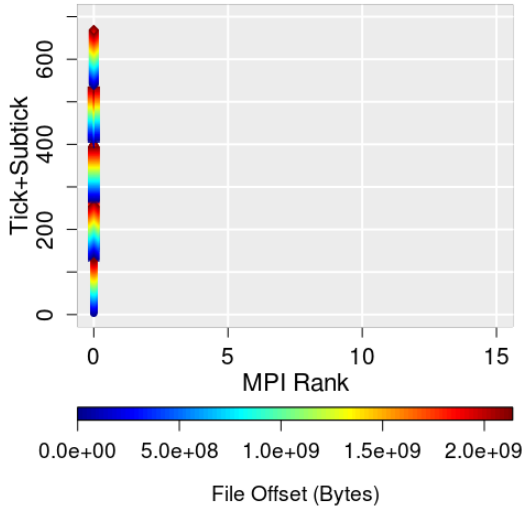
Añadir el modelo PIOM-PX a PIOM permite que se pueda analizar el comportamiento de E/S de una aplicación a nivel de POSIX-IO, facilitando observar los diferentes tratamientos que sufren los datos de una aplicación cuando pasan por diferentes niveles de la pila de software de E/S. Así mismo, observar este comportamiento en diferentes niveles, ayuda a entender qué información está recibiendo el sistema en las diferentes capas.



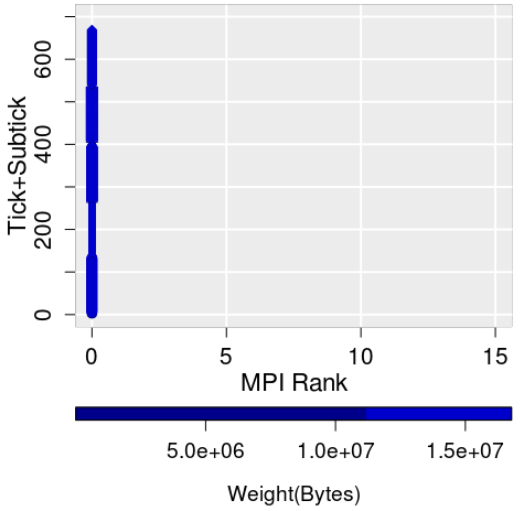
(a) Fases de E/S de S3DIO a nivel de MPI-IO



(b) Peso de la fase de S3DIO a nivel de MPI-IO



(c) Fases de E/S de S3DIO a nivel de POSIX-IO



(d) Peso de la fase de S3DIO a nivel de MPI-IO

Figura 5.4: Comportamiento de E/S de S3D-IO- 2D - usando 16 MPI procesos.

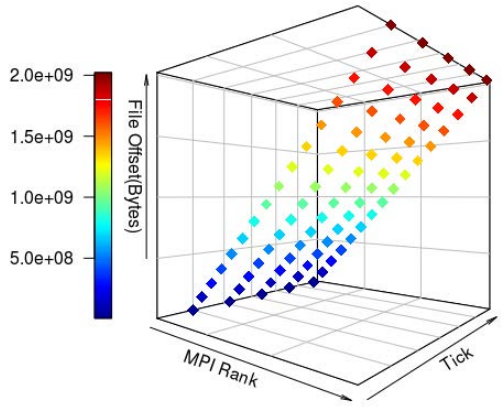


Figura 5.5: Comportamiento de E/S de S3D-IO- 3D - usando 16 procesos

6

Caso de Uso: PIOM en Clúster HPC en plataformas Cloud

La aparición de entornos Cloud ha provocado el interés de la comunidad científíca, enfocada en los sistemas HPC tradicionales, por analizar y evaluar si estos entornos se pueden considerar como alternativa a los clúster tradicionales. Ese interés radica en los beneficios que ofrece Cloud ya que los usuarios pueden adquirir y liberar recursos bajo demanda y pueden configurar y personalizar su propio clúster virtual [44].

En muchas áreas de investigación existe la necesidad de tener un entorno de pruebas para evaluar pruebas de concepto, validar ideas, testear implementaciones sin afectar a los entornos de producción. En muchos casos, se tienen sencillos, básicos y/o acotados entornos de prueba para hacer una primera estimación, validación y/o comprobación.

Para los investigadores centrados en las aplicaciones científicas paralalelas que usan E/S paralela esta plataforma ofrece muchas oportunidades ya que los entornos de Cloud se pueden utilizar como entornos de prueba (*Test-Bed*). A diferencia de los clúster tradicionales, en Cloud el usuario puede crear, configurar el sistema de E/S e instalar diferentes sistemas de ficheros sin afectar el trabajo de otros usuarios o el rendimiento de otras aplicaciones. Esto representa una ventaja sobre los sistemas de E/S de los sistemas HPC tradicionales.

Sin embargo, en estos entornos el usuario ha de tener en cuenta el tiempo de ejecución y también el coste de los recursos seleccionados, ya que un reto importante es el uso eficiente de los recursos. Otro reto de los entornos Cloud es la complejidad para configurar el sistema debido al número de parámetros que se han de considerar para crear un clúster virtual. También se ha de tener en cuenta que las prestaciones (ancho de banda del sistema de E/S) tiene una gran variabilidad. Esta variabilidad puede depender de factores tales como la hora y el número de usuarios; y, determina el ancho de banda que se puede esperar para una aplicación específica. La selección de una instancia para los nodos de un clúster no es trivial porque una instancia comprende una combinación de CPU, memoria, almacenamiento y red de interconexión. Además, se puede observar que los usuarios necesitan seleccionar componentes relacionados con la configuración del clúster tales como número de instancias, capacidad del almacenamiento global, tipo de sistema de ficheros, etc. En este capítulo, se muestra el uso del modelo PIOM en la plataforma Cloud.

6.1. Introducción

El modelo propuesto es portable (independiente de la máquina) y proporciona un modelo del comportamiento de la E/S de la aplicación replicable. Puede ser usado por el usuario para hacer una comparación rápida de las prestaciones que ofrecen diferentes clúster virtuales en Cloud (VCCs) teniendo en cuenta el comportamiento de E/S de la aplicación sin necesidad de ejecutar la aplicación y permitiendo probar el impacto de diferentes configuraciones y diferentes sistemas de ficheros. Uno de los principales beneficios de utilizar un programa sintético programable personalizado con la

información extraída del framework propuesto es la no necesidad de ejecutar la aplicación real, no ejecutando el cómputo y sin necesidad de cargar los ficheros de entrada. .

En el entorno Cloud, es el propio usuario el que crea, gestiona y configura un clúster virtual (VC). Este tipo de entornos ofrecen muchos recursos que el usuario debe seleccionar para crear su clúster virtual en Cloud (VCC). Sin embargo, también existen desventajas, una de ellas sería que el usuario tiene que prestar atención sobre los recursos seleccionados y el tiempo de utilización. Porque el usuario debe pagar por número de recursos seleccionados, tipo, tiempo de utilización y por la cantidad de datos transferidos.

En este punto, se propone una metodología para guiar al usuario en la evaluación de los clúster virtuales que serán configurados teniendo en cuenta los requerimientos de E/S de la aplicación. Además, esta metodología, en general, reduce el coste de la evaluación porque no es necesario ejecutar la aplicación real. Tanto el coste como la relación coste-rendimiento para los clúster virtuales puede ayudar al usuario a tomar una decisión sobre los recursos a seleccionar en la plataforma Cloud y cómo configurar la E/S. En esa decisión, el usuario, también tiene que tener en cuenta la variabilidad del rendimiento que se obtiene.

6.2. Estado del Arte

Diferentes grupos de investigación se han centrado en evaluar el entorno *Cloud* para comprobar su idoneidad para la computación de altas prestaciones [45]. El interés por el sistema de E/S ha crecido ya que se ha de configurar y utilizar eficientemente para controlar el coste por recurso utilizado. A continuación, se describen algunos de los trabajos de investigación realizados sobre el entorno Cloud:

R.Expósito y otros en [45] evalúan el rendimiento del subsistema de almacenamiento de E/S en el clúster de Amazon EC2 y las nuevas instancias de E/S que proporciona para averiguar si es apropiado para aplicaciones con E/S intensiva. Esta evaluación la hacen sobre el disco *ephemeral* (almacenamiento local y temporal) y el volumen EBS (*Elastic Block Storage*) en diferentes niveles, usando el sistema de ficheros local y un sistema de ficheros distribuido (NFS). Trabajan con las instancias de cómputo para clúster (CC) (CC1, CC2) y la instancia de altas prestaciones para la E/S (HS1). No evalúan sistemas de ficheros tal como PVFS.

Juve y otros en [46] evalúan el rendimiento y coste de tres aplicaciones científicas reales en Amazon EC, usando diferentes sistemas de almacenamiento: local y S3 (almacenamiento para Internet). Pero cada una de estas aplicaciones tiene una característica especial: una de ellas hace un uso de la memoria intensivo, otra se puede englobar en las aplicaciones que hacen un uso de la CPU intensivamente y la tercera se puede clasificar dentro de las aplicaciones con E/S intensiva. En este trabajo utilizan el sistema de ficheros distribuido (NFS) y el sistema de ficheros paralelo (PVFS). Trabajan con varias

instancias: c1.xlarge (la característica de este tipo de instancias es tener *cores* potentes), cc1.4xlarge (esta instancia es recomendada para clúster(CC)).

Liu y otros en [44] evalúan dos aplicaciones paralelas (BT-IO, POP) con diferentes configuraciones de E/S usando el sistema de ficheros distribuido (NFS) y el sistema de ficheros paralelo (PVFS). Usan el disco local y trabajan con instancias de cómputo para clúster (CCI).
La plataforma Cloud, además, introduce un nuevo nivel de abstracción, la virtualización, que incrementa la complejidad del sistema. Debido a esto, hay varios estudios que se centran en la evaluación de las prestaciones, entre ellos el trabajo de Noorshams y otros en [47] que presentan un modelado de las prestaciones de la E/S enfocado para los sistemas de almacenamiento virtualizados basado en la teoría de colas.

Sivathanu y otros en [48] presentan un estudio para la medición de las prestaciones de E/S. Se centran en la influencia del aprovisionamiento del disco, la interferencia de la colocación de la carga de trabajo en las prestaciones de la E/S y las implicaciones de la virtualización con diferentes tipos de cargas de trabajo.

6.3. Metodología

La metodología que se describe a continuación (ver Figura 6.1) permite realizar la evaluación del VCC partiendo del modelo de comportamiento de la aplicación obtenido previamente. Esta evaluación y comparación permite descartar aquellas instancias que no pueden cubrir mínimamente los requerimientos de E/S de la aplicación. A continuación, se explicará detalladamente cada uno de los pasos.

6.3.1. Caracterización de los nodos virtuales

El primer paso es la selección de una instancia y su caracterización. Amazon recomienda medir el rendimiento de las instancias. Es importante conocer el ancho de banda de E/S que proporciona un nodo antes de empezar a instalar capas software, para ello se utiliza el benchmark IOzone [49]. Este benchmark se ejecuta en un nodo, para obtener el promedio de los valores de la tasa de transferencia a nivel del sistema de ficheros local. IOzone es un benchmark que se utiliza como herramienta para evaluar el sistema de ficheros y proporciona las características dinámicas de un nodo. Este benchmark genera y evalúa varias operaciones de acceso a fichero.

IOzone se puede usar en modo automático, considerando que el tamaño del fichero es igual al doble del tamaño de la RAM, el minimo tamaño de la petición (*rs*) que se puede evaluar es 4KiB y el máximo es 1GiB o 2GiB dependiendo del rendimiento de la red. Esta evaluación puede durar un tiempo considerable. Sin embargo, con PIOM se obtiene conocimiento de la aplicación, en particular se obtienen los tamaños de *rs* que utiliza la aplicación. Esta información permite que IOzone se pueda

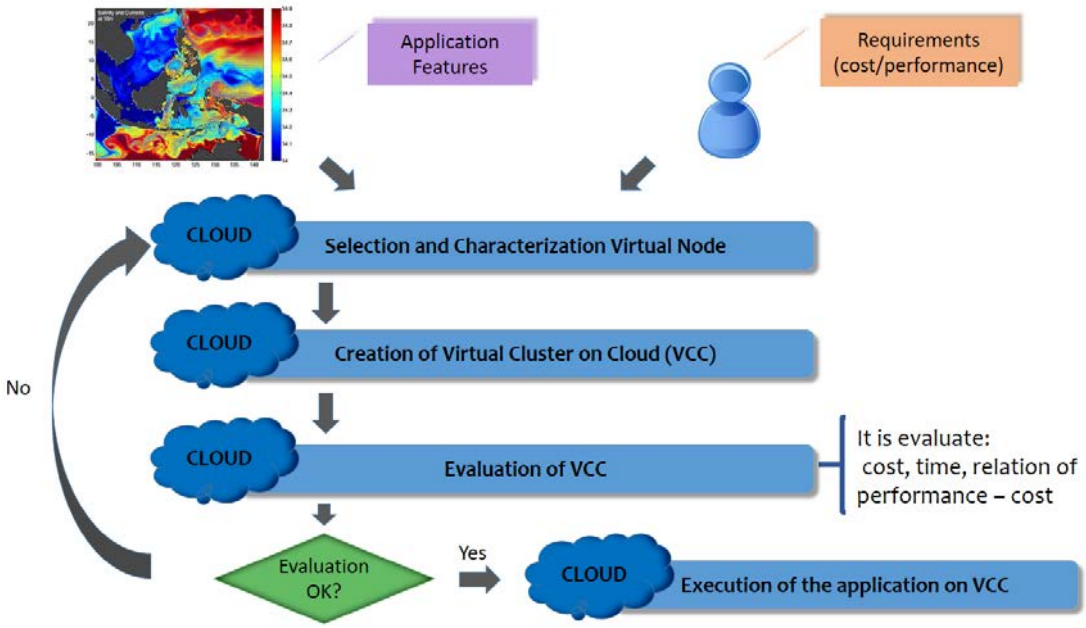


Figura 6.1: Metodología para la evaluación de rendimiento y coste de la E/S de un VCC

personalizar para evaluar esa instancia con los valores de *rs* específicos de una aplicación, permitiendo reducir el tiempo de evaluación de las instancias.

Con IOzone sólo se analizan las operaciones write/rewrite y read/reread. De esta forma se obtiene el ancho de banda promedio para los tamaños *rs* de la aplicación. Se puede establecer un rango de *rs* para evaluar mejor la instancia. En este paso, una instancia o instancias pueden ser descartadas si proporcionan un bajo ancho de banda.

Además, a partir de los datos obtenidos con IOzone se puede calcular el valor máximo para un sistema de ficheros global de los VCC. En este caso, el valor máximo es la suma de los valores obtenidos en cada nodo de E/S del sistema de fichero global.

6.3.2. Creación y Configuración de clúster virtuales

Una vez se ha seleccionado una instancia y un sistema de almacenamiento local o temporal (*ephemeral*) o persistente (el volumen EBS- *Elastic Block Storage*) ya que la instancia tiene una tasa de transferencia adecuada, los componentes que se han de tener en cuenta para crear un clúster virtual en Cloud (VCC) se muestran en la Tabla 6.1. Los componentes mostrados son los que proporciona la plataforma *Amazon Web Services (AWS)* [50].

Además se ha creado un plugin para la herramienta StarCluster [51], el cual permite desplegar el sistema de ficheros PVFS2 de una forma rápida y automática; y, también, seleccionar donde ubicar los nodos de E/S. Se han considerado diferentes arquitecturas:

Tabla 6.1: Componentes de un Clúster Virtual (VC)

Parameters	**Description**
Instance type (*)	Number of cores, processor capacity, RAM memory size.
Number of instances(*)	Number of nodes (include compute nodes and I/O nodes)
Number of I/O nodes (-)	Data servers and metadata server
Storage type(+)	Temporal and/or persistent
Device type temporal(+)	HDD or SSD
Device type persistent(+)	HDD or SSD
Capacity of temporal storage(+)	As minimum the storage capacity required (app_st)
Capacity of persistent storage(-)	
Network performance (+)	Low, Moderate, High, Unknown
I/O library (-)	MPI, NetCDF, pnetcdf, HDF5.
Local filesystem (+)	Filesystem Linux ext3, ext4, xfs, etc.
Global filesystem (-)	Parallel, Distributed or Network Filesystems
Stripe size (-)	Related by the parallel file system

- Nodos compartidos para nodos de cómputo y nodos de E/S,
- Nodos independientes para cómputo y E/S.

Se clasifican los parámetros presentados, en la Tabla 6.1, en tres categorías: los que puede seleccionar el usuario (*), los que el usuario debe configurar manualmente (-) y por último, los parámetros que el usuario no puede cambiar porque son valores por defecto dependiendo del tipo de instancia seleccionada (+).

Para seleccionar los componentes del sistema de E/S de un VCC y que cumplan los requerimientos del usuario se parte de las siguientes consideraciones:

- *Tipo de Instancia*: depende del coste que el usuario puede o quiere pagar.
- *Instancias*: el número de instancias es un parámetro determinado por el número de procesos requerido por el comportamiento de E/S de la aplicación y el cómputo.
- *Tipo de almacenamiento*: puede ser temporal (normalmente no supone un coste adicional al de la instancia) y/o persistente (coste por GB/month). Normalmente, las aplicaciones científicas paralelas usan el almacenamiento temporal para guardar los ficheros de entrada y los ficheros temporales generados durante la ejecución. Sólo los datos que se necesitan postprocesar son guardados en el almacenamiento persistente.
- *Capacidad de almacenamiento*: se debe garantizar como minimo la capacidad de almacenamiento requerida por la aplicación (ver expresión 3.2).

- *Rendimiento de la red de interconexión*: puede ser alta, baja, moderada y desconocida. Está asociado con el tipo de instancia. La carga de trabajo en la red de interconexión dependerá del número de procesos si la aplicación es fuertemente o débilmente acoplada, el tamaño de la petición, la arquitectura del sistema de E/S (nodos compartidos para nodos de cómputo y nodos de E/S o nodos independientes para cómputo y E/S). y el número de las operaciones de E/S.

- *Librerias de E/S*: depende de la librería de E/S utilizada por la aplicación. En nuestro caso, sólo se utiliza MPI-IO and POSIX-IO porque se extrae el modelo del comportamiento de E/S de aplicaciones MPI.

- *Tipo de sistema de ficheros*: depende de si el tipo de acceso es compartido o único (un fichero por proceso). Si el tipo de acceso es compartido entonces el sistema de ficheros debe ser un sistema de fichero global tales como NFS, PVFS2 o Lustre. En cambio, cuando el tipo de acceso es único es posible usar el sistema de fichero local para aprovechar los discos locales conectados a nodos de cómputo.

El software básico de un VCC para cada nodo de cómputo depende de la imagen de la máquina seleccionada. Aunque, al igual que en los clúster tradicionales, el sistema operativo Linux es el más usado, especialmente para la pila de software de E/S. El software para el procesamiento paralelo tanto como MPI y el sistema de ficheros global deben ser instalados y configurados por el usuario. El coste-rendimiento será una restricción a considerar en un VCC. Los componentes previamente citados permitirán tomar una decisión considerando los requirimientos de la aplicación.

6.3.3. Evaluación del rendimiento en los clúster virtuales para el modelo de comportamiento de la E/S de una aplicación.

Teniendo el modelo de comportamiento de E/S de una aplicación e identificadas las fases más significativas, se personaliza el benchmark IOR para realizar la evaluación de los VCC con cada una de esas fases. Con IOR se consigue replicar las fases en los VCC para observar que configuración de E/S es más idónea para ejecutar la aplicación. Esta observación se basa en el rendimiento obtenido.

La Tabla 6.2 muestra los parámetros de entrada para IOR basado en las fases del modelo de comportamiento de E/S de la aplicación. La salida de este proceso es la tasa de transferencia llamada $BW_{CH}(phase[i])$ y expresada en MB/s, y el tiempo de E/S para el modelo de comportamiento de E/S de la aplicación.

Tabla 6.2: Parámetros de entrada para IOR basado en el modelo de E/S de una aplicación

Access Mode	Access Type(AT)	Param. for AT	Number of processes	Number of segment	Block size	Transfer size
Strided	SHARED		np=np(IdPh)	-s=rep	-b=rs(IdPh)	-t=rs(IdPh)
Sequential	SHARED		np=np(IdPh)	-s=1	-b=weight(IdPh)	-t=rs(IdPh)

6.3.4. Evaluación del coste de los clúster virtuales

Para predecir el coste de utilización de la E/S de un VCC se utiliza el rendimiento obtenido al configurar IOR con los parámetros obtenidos con PIOM y que reflejan el comportamiento de cada una de las fases de E/S de la aplicación. El coste total para un Clúster Virtual depende del tipo de instancia, número de instancias y del tiempo de uso de esas instancias. Está compuesto por un coste variable ($cost\_var$) y un coste fijo ($cost\_fix$). El coste variable correspondería al número de instancias utilizadas y el tiempo de uso, mientras que el coste fijo correspondería al tipo de instancia seleccionada para crear el VCC.

Para calcular el coste se utiliza la expresión 6.1.

$$cost\_tot = cost\_var + cost\_fix \tag{6.1}$$

Como el modelo propuesto está basado en el concepto de fases de E/S, a la hora de calcular el coste también se hace por fases, permitiendo realizar una evaluación rápida del coste de la E/S.

La métrica seleccionada para el IOR es la tasa de transferencia (BW_{CH}) y se expresa en MB/s. El coste variable estimado para el modelo de comportamiento de la E/S de la aplicación es calculado con la expresión 6.2. Esta variable es proporcional al tiempo de utilización de la E/S, y por tanto a la cantidad de datos transferidos. En Cloud, se factura por horas utilizadas.

$$cost\_var = \sum_{i=1}^{phases} cost(phase[i]) \times num\_inst \tag{6.2}$$

Donde num_inst es el número de instancias usadas durante la ejecución y el $cost(phase[i])$ es calculado por la expresión 6.3.

$$cost(phase[i]) = cost(phase\_i) + [EBS] \tag{6.3}$$

$$cost(phase\_i) = \frac{weight(phase[i])}{BW_{(CH)}(phase[i])}/3600 \times cost_{inst} \tag{6.4}$$

$cost_{inst}$ es el coste por hora para cada tipo de instancia $inst$ y EBS representa el almacenamiento permanente por mes y, es un valor fijo. EBS es un valor opcional dependiendo si se ha utilizado o

no como almacenamiento. Si se utiliza este tipo de almacenamiento se ha de considerar coste fijo. Como se ha comentado *EBS* es opcional, pero si no se puede ejecutar en el almacenamiento temporal se puede añadir almacenamiento permanente para incrementar la capacidad de almacenamiento o simplemente para tener almacenamiento permanente.

6.3.5. Comparación de la relación rendimiento-coste para un clúster virtual.

Tanto el rendimiento como el coste de los clúster virtuales se calcula para facilitar al usuario la toma de decisiones. Para establecer la relación entre el rendimiento y el coste se utiliza la expresión 6.5.

$$perf\_cost_{ci} = \frac{performance_{ci}}{cost_{ci}} \tag{6.5}$$

donde *ci* representa un clúster virtual e $i \in \{1..TotalVirtualClusters\}$. Donde $TotalVirtualClusters$ es el número total de las distintas configuraciones de VCCs seleccionadas para analizar.

También, se puede calcular la relación tiempo-coste de diferentes clusteres virtuales con la expresión 6.6.

$$perf\_cost_{ci} = \frac{Time_{ci}}{cost_{ci}} \tag{6.6}$$

donde *Time* es expresado en seg., *ci* representa un VCC, e $i \in \{1..TotalVirtualClusters\}$.

Para comparar el clúster o VCC *ck* y el VCC *cj*, se puede decir que *ck* tiene una relación rendimiento-coste más eficiente que *cj*, si $perf\_cost_{ck}$ es más alto que $perf\_cost_{cj}$.

6.4. Resultados Experimentales

En esta sección se muestran los experimentos realizados en la plataforma cloud y se utilizó nuevamente como aplicación el benchmark BT Class B y C. Se obtuvo el modelo de comportamiento de E/S y se ejecutó BT-IO usando 4, 9, 16, 25 y 36 procesos.

6.4.1. Entorno experimental

En esta ocasión se trabaja en la plataforma cloud de Amazon Web Services (AWS) EC2. Se seleccionaron tres instancias teniendo en cuenta los requerimientos de E/S de la aplicación y el coste de las instancias.

En la Tabla 6.3 se muestran las características de las instancias de Amazon consideradas en los experimentos realizados. Se ha considerado la instancia C3 porque es una de las instancias que Amazon

recomienda para crear clúster HPC. El uso de la instancia M1 fué para comprobar que implicaciones tenía usar las instancias catalogadas por Amazon como “uso general".

Tabla 6.3: Características de las instancias de Amazon seleccionadas

Instances	Processor	CPU	RAM (GB)	Storage(GB)	AWS Ireland ($ Per Hour)	AWS Virginia ($ Per Hour)
m1.small	Intel Xeon Family	1	1.7	1x160	0.047	0.044
m1.large	Intel Xeon Family	2	7.5	2x420 HHD	0.190	0.175
c3.xlarge	Intel Xeon E5-2680 v2 2.8 GHz	4	7.5	2x40 SSD	0.239	0.210

6.4.2. Aplicación de la metodología

Caracterización para los nodos virtuales

Una vez seleccionadas las instancias, se ejecutó IOzone para evaluar el disco ephemeral y obtener los valores pico de su tasa de transferencia. La evaluación se realiza en un sólo nodo. Sólo se evaluaron los discos ephemeral usados por los sistemas de fichero: NFS y PVFS2. Se evaluaron las operaciones de escritura y lectura para los tamaños de petición desde 4 KiB hasta 1GiB. En este caso, no se personalizó porque se quería saber que rendimiento ofrecían estas instancias. Tabla 6.4 presenta los valores pico máximo y mínimo de las instancias utilizadas en cada VCC, donde VCC1-VCC4 corresponde a los cuatro clúster creados. La evaluación con IOzone de las instancias permitió detectar la existencia de variabilidad en los resultados. Esta variabilidad depende de la hora en la que se está realizando la evaluación.

En la tabla 6.5 se describen los componentes de los clúster virtuales(VCC).

Extraer el modelo del comportamiento de la E/S de una aplicación

En los capítulos anteriores, se mostró como extraer el modelo de comportamiento de la E/S de una aplicación, en concreto se presentó como ejemplo el benchmark BT-IO.

La Tabla 6.6 presenta las fases de E/S detectadas en la ejecución del benchmark BT-IO para diferentes número de procesos. También se muestra la capacidad de almacenamiento requerida por BT-IO para diferentes clases.

Creación y Configuración de los clúster virtuales

Como BT-IO usa un fichero compartido se configuró un sistema de ficheros global: NFS y PVFS2. Teniendo en cuenta los cuatro clúster definidos anteriormente se describe, seguidamente, que clase de BT-IO se ha ejecutado en cada clúster. En VCC1 se ejecutan la clase B y C del BT-IO hasta 16

Tabla 6.4: Valores pico (máximo y mínimo) obtenidos con IOzone para las instancias que componen los VCC definidos.

Cluster	node	rs	Write (MB/s)	Read (MB/s)	Time (sec)
VCC1	m1.small				33.6
		32KB	108	114	
		256KB	106	115	
VCC2	m1.large				43.6
		32KB	85	84	
		1GB	78	91	
VCC3	c3.xlarge				24.1
		8MB	116	291	
		1GB	96	308	
VCC4	c3.xlarge				24.1
		8MB	930	2,328	
		1GB	771	2,462	

Tabla 6.5: Características descriptivas de los clúster virtuales configurados para los experimentos.

I/O components	**VCC1**	**VCC2**	**VCC3**	**VCC4**
Instance Type	m1.small	m1.large	c3.xlarge	c3.xlarge
Number of Instances	17	17	11	11
Storage Type Temporal	Ephemeral	Ephemeral	Ephemeral	Ephemeral
Storage Type Persistent	EBS	EBS	EBS	EBS
Device Type Temporal	HDD	HDD	SSD	SSD
Device Type Persistent	HDD	HDD	HDD	HDD
Capacity of Temporal Storage	160GB	420GB	40GB	300GB
Capacity of Persistent Storage	8GB	8GB	8GB	16GB
Networking Performance	Low	Moderate	High	High
Number of data servers	1	1	1	8
Number of Metadata Server	1	1	1	1
Filesystem Local	ext3	ext3	ext3	ext3
Filesystem Global	NFS	NFS	NFS	PVFS2
Stripe Size	—	—	—	64KB
I/O library	mpich2, pnetcdf	mpich2, pnetcdf	mpich2, pnetcdf	mpich2, pnetcdf
EBS Fixed Cost EU($ per GB-month)	0.55	0.55	0.55	0.55
EBS Fixed Cost US-East($ per GB-month)	0.50	0.50	0.50	0.50

procesos, en VCC2 se usan 17 nodos hasta 25 procesos, en VCC3 se usan 11 nodos porque es suficiente

Tabla 6.6: Fases de E/S para el modelo BT-IO usando 4, 9, 16, 25 y 36 procesos para las clases B y C.

IdPh	np	Operation	rs	rep	weight(IdPh) (rep*np*rs)	weight(app)	Storage Capacity required
Class B						3.2GB	
1 to 40	4	Write_at_all	10MB	1	$1*4*10MB$		1.6GB
41	4	Read_at_all	10MB	40	$40*4*10MB$		
1 to 40	9	Write_at_all	4.5MB	1	$1*9*4,5MB$		1.6GB
41	9	Read_at_all	4.5MB	40	$40*9*4,5MB$		
1 to 40	16	Write_at_all	2.5MB	1	$1*16*2,5MB$		1.6GB
41	16	Read_at_all	2.5MB	40	$40*16*2,5MB$		
1 to 40	25	Write_at_all	1.62MB	1	$1*25*1,62MB$		1.6GB
41	25	Read_at_all	1.62MB	40	$40*25*1,62MB$		
1 to 40	36	Write_at_all	1.12MB	1	$1*36*1,12MB$		1.6GB
41	36	Read_at_all	1.12MB	40	$40*36*1,12MB$		
Class C						12.9GB	
1 to 40	4	Write_at_all	40.55MB	1	$1*4*40,55MB$		6.4GB
41	4	Read_at_all	40.55MB	40	$40*4*40,55MB$		
1 to 40	9	Write_at_all	18MB	1	$1*9*18MB$		6.4GB
41	9	Read_at_all	18MB	40	$40*9*18MB$		
1 to 40	16	Write_at_all	10.14MB	1	$1*16*10,14MB$		6.4GB
41	16	Read_at_all	10.14MB	40	$40*16*10,14MB$		
1 to 40	25	Write_at_all	6.49MB	1	$1*25*6,49MB$		6.4GB
41	25	Read_at_all	6.49MB	40	$40*25*6,49MB$		
1 to 40	36	Write_at_all	4.50MB	1	$1*36*4,50MB$		6.4GB
41	36	Read_at_all	4.50MB	40	$40*36*4,50MB$		

para ejecutar la clase C de BT-IO con 36 procesos y en VCC4 se ejecutan la clase B y C.

Evaluación del rendimiento en los clúster virtuales para el modelo de E/S de una aplicación

Los parámetros de entrada que se obtuvieron para configurar IOR a partir del modelo de E/S se muestran en la Tabla 6.7. En esta misma tabla, se muestran las salidas obtenidas para el clúster VCC3.

En este punto, se obtiene la tasa de transferencia (BW_{CH}) y un tiempo de ejecución para el modelo de E/S del BT-IO. Estos valores son usados para calcular el coste para las diferentes clases y número de procesos en los 4 clúster virtuales. Se muestran los resultados obtenidos para el clúster VCC3.

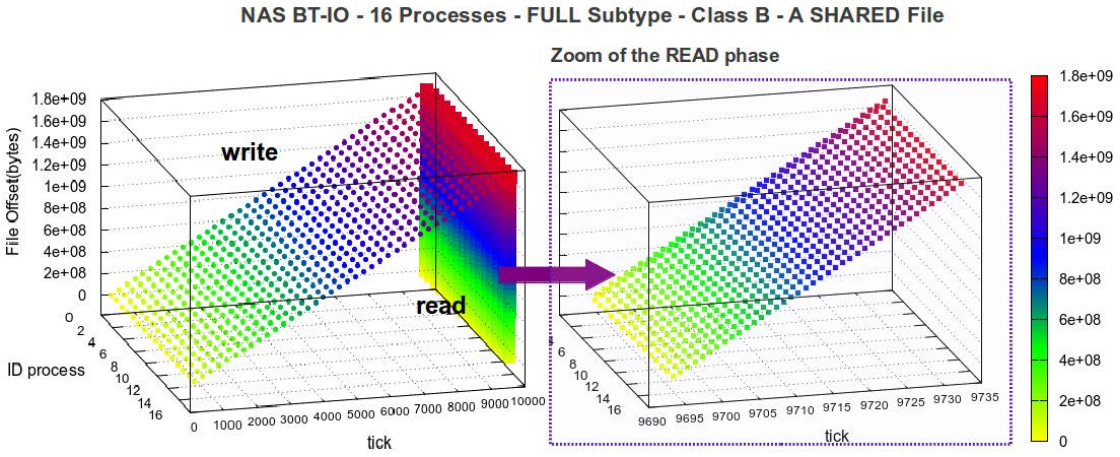


Figura 6.2: Modelo del comportamiento de la E/S para BT-IO

Tabla 6.7: Parámetros de entrada para IOR extraido de las fases del modelo de comportamiento de E/S de NAS BT-IO Subtipo FULL - Operaciones colectivas y $s = rep = 40$. Se muestran las salidas obtenidas para el clúster VCC3.

np(IdPh)	b=rs(IdPh) (MB)	t=rs(IdPh) (MB)	$BW_{(}CH)$ (MB/s)	Time (sec)
Class B				
4	10.0	10.0	104	15.5
9	4.5	4.5	91	17.7
16	2.5	2.5	96	16.9
25	1.6	1.6	73	22.6
36	1.1	1.1	74	22.2
Class C				
4	40.6	40.6	87	74.5
9	18.0	18.0	83	77.8
16	10.1	10.1	84	77.2
25	6.5	6.5	82	79.0
36	4.5	4.5	78	82.9

Evaluación del coste de los clúster virtuales

La Figura 6.3 presenta el coste de evaluación de los cuatro clúster VCC para el benchmark IOR configurado con los valores obtenidos del modelo de comportamiento de E/S para las clases B y C de BT-IO.

Comparando el coste entre los clúster se ha observado que el clúster VCC2 no es una buena opción para la clase B y C de BT-IO con NFS, porque el coste es más alto que los otros clúster. Pero, para la clase C cuando se trabaja con PVFS2 el coste decrece en el clúster VCC4. Con lo cual, ejecutar en el clúster VCC4 es una opción apropiada.

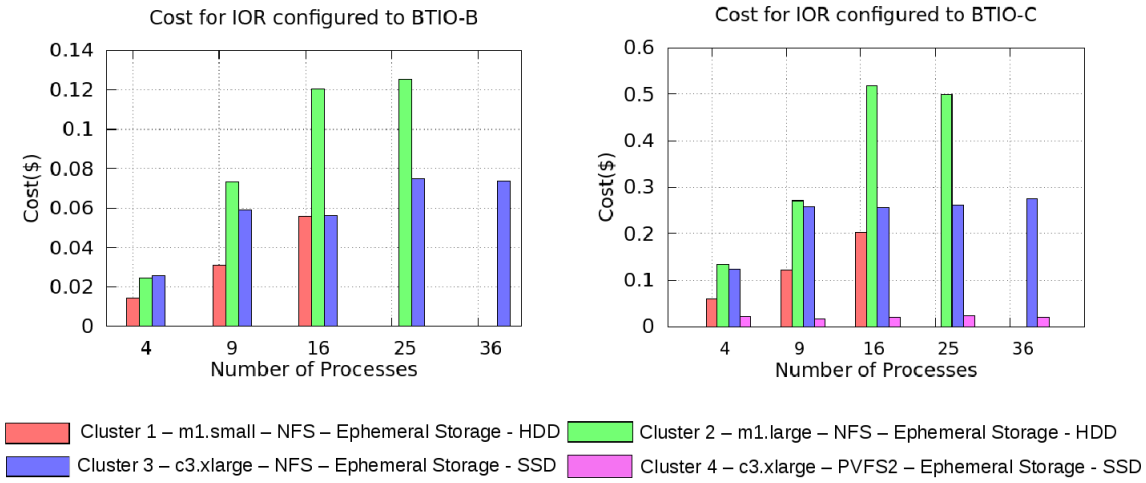


Figura 6.3: Evaluación del coste de los 4 VCC usando IOR configurado para BT-IO. El coste depende de los recursos seleccionados y del tiempo que se utilizan (medido en horas). La imagen de la izquierda corresponde a la clase B y la imagen de la derecha corresponde a la clase C. Los clúster con experimentos pero sin resultados están limitados por la capacidad de almacenamiento o por el grado de paralelismo requerido. La clase B no se ha ejecutado en el clúster 4 porque la carga de trabajo de E/S es pequeña para su sistema de E/S.

Comparación de la relación del coste-rendimiento para VCC

En la Figura 6.4 se muestra la relación coste-rendimiento para los 4 VCC. Esta relación es obtenida por la expresión 6.5. En el caso de la clase B el clúster VCC3 es el que presenta la mejor relación coste-prestaciones para ejecutar esta clase, mientras que para la clase C el clúster VCC4 es el más apropiado.

6.5. Análisis de casos

6.5.1. Análisis de la aplicación ABySS utilizando Cloud como Test-Bed

Teniendo en cuenta la aplicación ABySS se quiso comprobar si un sistema de ficheros global podría impactar positivamente en el rendimiento de esta aplicación. Para ello, se utilizó la plataforma cloud para poder modificar el sistema de ficheros. Con la intención de recrear, lo mejor posible, el clúster HPC donde se va a ejecutar la aplicación. Se crearon dos VCC con las características mostradas en la la Tabla 6.8.

Se ha usado nuestro propio plugin para la herramienta StarCluster, el cual permite desplegar el sistema de ficheros PVFS2 de una forma rápida y automática. Debido al coste y al tiempo necesario, explorar todas las alternativas que ofrece el entorno Cloud es una opción prohibitiva. Por lo tanto, sólo se selecciona los recursos de Cloud para evaluar el modelo de comportamiento de E/S de ABySS-P.

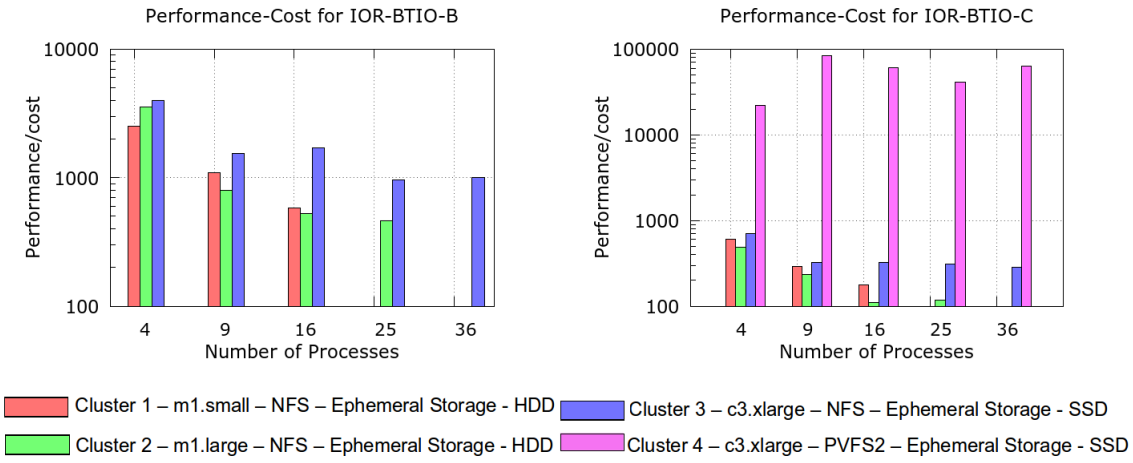


Figura 6.4: Relación del rendimiento-coste para los 4 VCC usando IOR configurado para el BT-IO. La imagen de la izquierda corresponde a la Clase B mientras que la imagen de la derecha corresponde a la Clase C. Los resultados se muestran con escala logarítmica. Los clúster con experimentos sin resultados están limitados por la capacidad de almacenamiento o el grado de paralelismo requerido. La clase B no se ha ejecutado en el clúster 4 porque la carga de trabajo de E/S es pequeña para su sistema de E/S.

Tabla 6.8: Características descriptivas de los clúster virtuales (VCC) en Amazon EC2.

I/O components	**VCC 1**	**VCC 2**
Instance	c3.2xlarge	c3.2xlarge
Number of Instances	6	6
Temporal Storage	Ephemeral	Ephemeral
Persistent Storage	EBS	EBS
Temporal Device	SSD	SSD
Persistent Device	SSD(GP 192/3000)	SSD(GP 300/3000)
Capacity of Persistent Storage	100GB	64GB
File system Local	ext4	ext4
File system Global	NFS	PVFS2 (2.8.8)
Parallel Storage Capacity	-	400GB
Number of data servers	-	5
Number of Metadata Server	-	1
Stripe Size	-	64KB
MPI library	mpich-3.2	mpich-3.2

La Figura 6.5 presenta el ancho de banda obtenido para IOR personalizado con el modelo de E/S para ABYSS-P en NFS y PVFS2. Como se puedo observar con NFS se obtiene un ancho de banda de 117 MiB/sec, mientras que con PVFS2 se obtiene como máximo 48 MiB/sec para las configuraciones evaluadas.

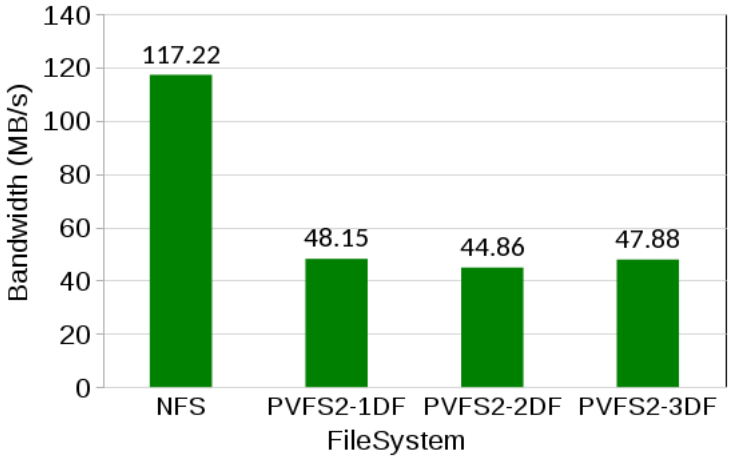


Figura 6.5: Ancho de banda obtenido en VCC1 y VCC2 usando el bechmark IOR personalizado para el modelo de E/S de ABYSS-P.

A raíz de estos experimentos podemos concluir que un sistema de ficheros paralelo tal como PVFS2 no es favorable para el patrón de E/S de ABySS-P. El bajo rendimiento obtenido con PVFS2 está relacionado con el tamaño de la petición de las operaciones de lectura, ya que éste es pequeño. En esta aplicación sólo tenemos dos procesos que leen. En realidad, hay dos procesos MPI que leen dos ficheros de entrada y luego envian los datos al resto de procesos y, esto es un obstáculo para la escalabilidad.

6.5.2. S3DIO

En este apartado presentamos los resultados al evaluar la aplicación S3DIO [43] con las configuraciones 200x3 y 400x3, usando 8, 16 y 32 procesos. En los VCC utilizados se instaló el sistema de ficheros PVFS2.

Caracterización de la aplicación paralela

La Tabla 6.10 presenta las fases de E/S para el modelo de E/S usando 16 y 36 procesos para las cargas de trabajo 200x3 y 400x3. Además, se muestra la capacidad de almacenamiento requerido por la aplicación y la memoria requerida para cada proceso.

La Figura 6.6 representa el modelo de comportamiento de la E/S para la aplicación S3D-IO, usando 8 y 16 procesos para una carga de trabajo 200x200x200 (200x3). Este comportamiento se observa para las cargas de trabajo 200x3 y 400x3. Además, se muestra la capacidad de almacenamiento requerida por la aplicación.

Tabla 6.9: Fases de E/S para el modelo S3D-IO usando 16 y 32 processes para las cargas de trabajo 200x3 y 400x3.

Workload	IdPh	np	Operation	rs (MB)	rep	weight(IdPh)MB (rep × np × rs)	weight(app)	STapp(GB)	MRxP (MB)	Files
200x3	1 to 5	16	write_all	61	1	976	4.8GB	4.8GB	305	1 per IdPh
	1 to 5	32	write_all	30.5	1	976	4.8GB	4.8GB	153	1 per IdPh
400x3	1 to 5	16	write_all	488	1	7808	39GB	39GB	2,442	1 per IdPh
	1 to 5	32	write_all	244	1	7808	39GB	39GB	1,221	1 per IdPh

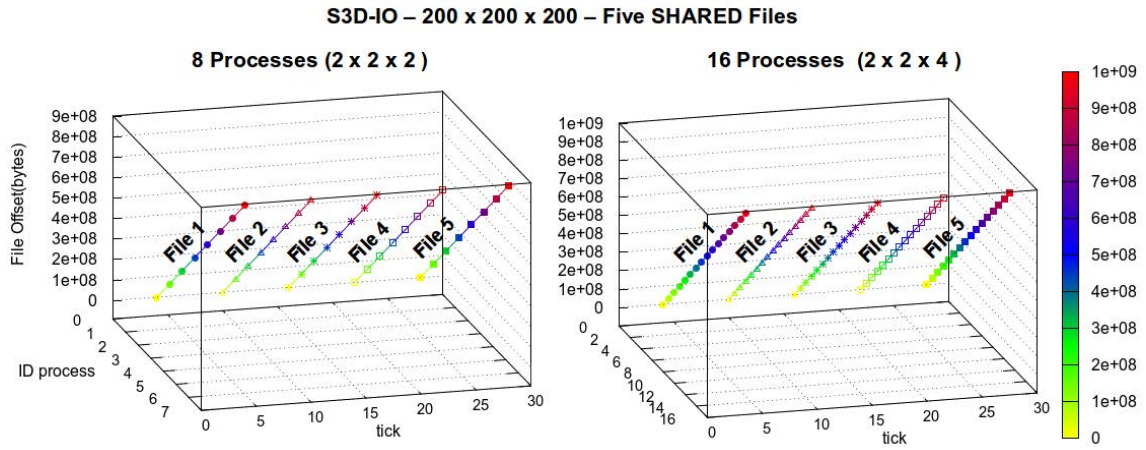


Figura 6.6: La imagen de la izquierda muestra el modelo de comportamiento de E/S para 8 procesos con una carga de trabajo 200x200x200. La aplicación usa 5 ficheros compartidos. Todos los procesos MPI escriben una vez en el fichero 1, después todos los procesos escriben en el fichero 2, y así sucesivamente. El patrón de acceso está representando el proceso de *checkpointing* para la aplicación S3D. El mismo comportamiento es observado en la imagen de la derecha para 16 procesos.

Caracterización de los clúster virtuales

Se ejecuta IOzone para evaluar el ancho de banda de la tasa de transferencia proporcionada por el disco *ephemeral* (disco local) para un nodo. Como en esta aplicación todos los procesos MPI utilizan un fichero compartido se instalará el sistema de ficheros PVFS2. Sólo se evalúan los *ephemeral* usados por el sistema de ficheros paralelo PVFS2. El PVFS2 se configuró con un servidor de metadatos (MDS), 8 servidores de datos (DF) y el tamaño de *stripe* fue 64KiB. El número de nodos de cómputo utilizado fue 8, teniendo en cuenta que los nodos del 1 al 8 funcionan como nodo de cómputo y nodo de E/S. Se usan en total 10 instancias, donde el nodo master es usado como un servidor MDS.

Se evalúa la tasa de transferencia para las operaciones de lectura y escritura para los tamaños de petición de 64KiB a 1GiB. En la tabla 6.11 se presenta el ancho de banda del almacenamiento, el promedio de la operación y la desviación estándar.

Tabla 6.10: Fases de E/S para el modelo de S3D-IO usando 8, 16 y 32 procesos para las cargas de trabajo 200x3 y 400x3, tipo de operación write_all y $rep = 1$ (Paso 3)

IdPh	**np**	**rs** (MB)	**weight(IdPh)** (MB)	**weight** (app)	**Storage** Capacity
200x3			(rep*np*rs)	4.8GB	4.8GB
1 to 5	8	122	976		
1 to 5	16	61	976		
1 to 5	32	30.5	976		
400x3			(rep*np*rs)	39GB	39GB
1 to 5	8	977	7816		
1 to 5	16	488	7808		
1 to 5	32	244	7808		

Tabla 6.11: Ancho de banda para el almacenamiento calculado con IOzone para los Clústeres Virtuales en Cloud (VCC) con diferentes tamaños.

Cluster Virtual	**Avg.Write (MB/s)**	**Std Write (MB/s)**	**Avg.Read (MB/s)**	**Std Read (MB/s)**
VCC 1	774	6	2,253	12
VCC 2	945	14	2,167	20

Evaluación del rendimiento de S3D-IO

En la tabla 6.12 se muestra los requerimientos de E/S para S3D-IO en VCC 1 y VCC 2. En este caso, los requerimientos de E/S están centrados en la memoria requerida porque el almacenamiento requerido, en los dos VCC, es menor que el 20 %.

Para analizar el impacto de la memoria requerida, se evalúa el modelo de E/S de S3DIO usando IOR. Figura 6.7 muestra la tasa de transferencia en VCC 1 (c3.xlarge) y VCC 2 (m3.xlarge).

Los requerimientos de memoria para S3DIO para la carga de trabajo 400x3 (cantidad de E/S = 39GiB) representa el 64 % de la RAM para un nodo de cómputo en VCC 1 mientras que es un 32 % para VCC2. En VCC 1, el valor es mayor que el 50 %, lo que podría evitar aprovechar la capacidad de rendimiento del sistema de E/S. En la figura 6.7, se puede observar para la cantidad de E/S de 39GiB (workload 400x3) que VCC 2 tiene 20 % más rendimeinto que VCC 1. Esto muestra que la aplicación obtiene más rendimiento en VCC2, donde la memoria requerida es 32 %, el cual es 50 % menos que VCC 1.

Finalmente, se compara el tiempo de ejecución de S3D-IO y su modelo de E/S. En la figura 6.8, se muestra que el tiempo de ejecución del modelo de E/S es menor que el tiempo de ejecución de la aplicación real. Como se paga por instancia usada y por hora utilizada, el coste será menor. En este

Tabla 6.12: Requerimientos de E/S para S3DIO en VCC 1 (c3.xlarge) y VCC 2 (m3.xlarge). 8 nodos realizan el cómputo y la E/S. Servidores de metadatos (MDS) y número de Ficheros de Datos(DF)

Class	Number of Processes	Compute Nodes (CN)	Processes x CN	Mem.Req. x CN (MB)	%Mem.Req. x CN (VCC 1)	%Mem.Req. x CN (VCC 2)
200x3	16	8	2	610	8 %	4 %
400x3	16	8	2	4883	64 %	32 %
200x3	32	8	4	610	8 %	4 %
400x3	32	8	4	4883	64 %	32 %

Class	Number of Processes	Compute Nodes (CN)	Processes x CN	MDS+DF	Filesystem Capacity(GB)	%Storage Used
200x3	16	8	2	1+8	320	1 %
400x3	16	8	2	1+8	320	12 %
200x3	32	8	4	1+8	320	1 %
400x3	32	8	4	1+8	320	12 %

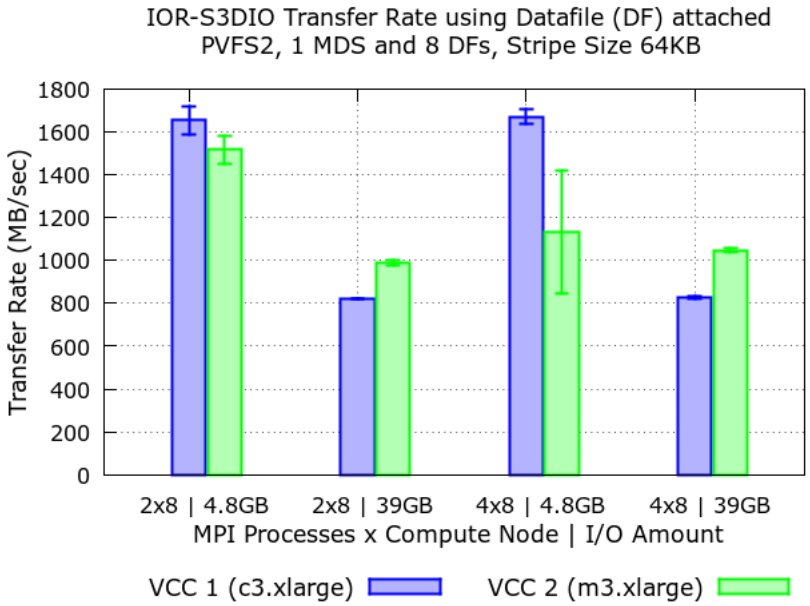


Figura 6.7: Velocidad de transferencia obtenida con IOR ajustada para los parámetros del modelo de E/S del S3DIO en VCC1 (c3.xlarge) y VCC2 (m3.xlarge).

sentido, proporcionamos un método para evaluar el rendimiento para el patrón de E/S de la aplicación en menos tiempo y con menos coste.

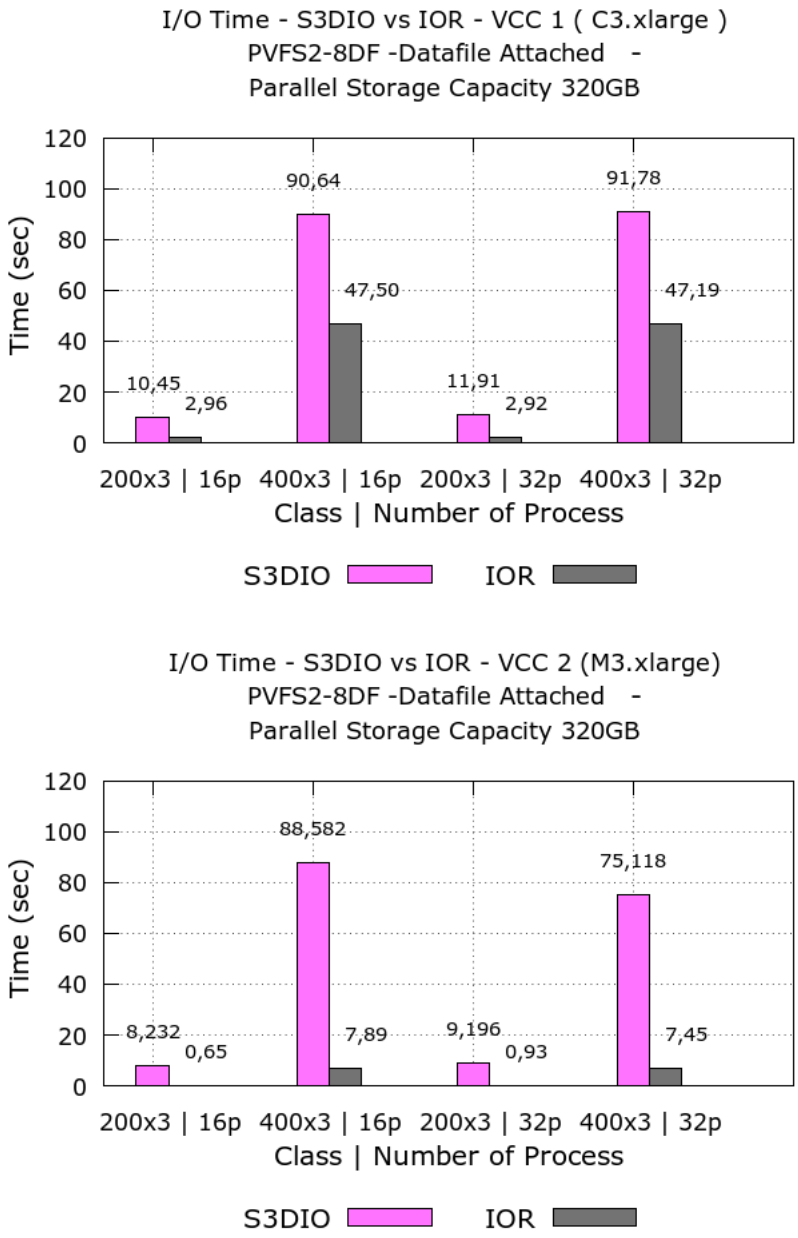


Figura 6.8: Comparación del tiempo entre S3D-IO y su versión correspondiente con IOR en VCC1 y 2

6.6. Conclusiones

La plataforma cloud se está analizando como alternativa a los sistemas HPC tradicionales para ejecutar aplicaciones científicas. Una de las características de esta plataforma es el número de parámetros que el usuario tiene que configurar es mucho mayor que en sistemas HPC tradicionales, dado que el

usuario debe gestionar y configurar su propio clúster. Aunque, en un principio se puede considerar que tener tantos parámetros para configurar puede ser una desventaja, con el tiempo se comprueba que es una ventaja. La razón, el usuario puede utilizar la plataforma cloud como un banco de pruebas (*Test Bed*) en el que puede cambiar factores como el número de nodos y la configuración del sistema E/S (líbrerias de E/S, sistemas de ficheros, etc.). Modificar la configuración del sistema de E/S permite al usuario la posibilidad de evaluar como impacta el comportamiento de E/S de una aplicación en diferentes configuraciones del sistema de E/S sin tener que preocuparse de la carga de trabajo que existe en un sistema HPC tradicional.

7

Conclusiones y Líneas Abiertas

7.1. Conclusiones

En este trabajo se ha propuesto un modelo para representar el comportamiento de la E/S de las aplicaciones paralelas MPI. Este modelo de comportamiento de la E/S está organizado en fases, las cuales nos permiten replicar el comportamiento de la aplicación en otros sistemas HPC. Como algunas aplicaciones usan librerías de E/S serie, el análisis se ha centrado en la capa POSIX de la pila software de la E/S. Se han descrito los principales parámetros para definir el modelo de comportamiento de la E/S y estos parámetros son independientes del sistema porque se quiere obtener un modelo portable.

Analizar la aplicación a nivel de POSIX-IO proporciona más conocimiento de cómo son tratados los datos de un aplicación en la pila software de E/S. Sin embargo, analizar a este nivel tiene el inconveniente de que al ser el nivel más cercano al sistema de ficheros, la información tiene más dependencia del sistema en el que se está ejecutando. Por lo tanto, para obtener el modelo es necesario conocer el sistema y limpiar los datos, para que nos de una información del comportamiento de la aplicación replicable en otros sistemas. Para poder integrar la información en el nivel de POSIX con la información del comportamiento de la aplicación a nivel MPI se ha creado un modelo que describe el patrón temporal de cada una de las fases de acceso a un fichero, para obtener el orden temporal de eventos se ha definido el concepto de tick.subtick. El subtick es un contador que permite controlar el orden lógico de los eventos POSIX-IO y establecer la relación que existe con los eventos MPI a través del tick.

En este trabajo se ha definido un modelo, PIOM-PX, que permite su integración con el modelo PIOM-MP desarrollado en nuestro grupo de investigación. PIOM-PX es un modelo de comportamiento de E/S a nivel de POSIX-IO para las aplicaciones paralelas MPI. Este modelo se ha definido para analizar las aplicaciones paralelas MPI que usan librerías de E/S serie. La integración de PIOM-PX a PIOM-MP, dando lugar al modelo PIOM, ha permitido observar la utilidad de analizar una aplicación paralela MPI con estos dos modelos integrados. Ya que, a parte de poder analizar las aplicaciones paralelas que usan librerias paralelas de E/S (MPI-IO) y las que ultizan librerias de E/S serie (POSIX-IO), también permite observar el comportamiento de E/S de las aplicaciones paralelas MPI que usan las librerias de E/S paralelas en dos capas diferentes de la pila software de E/S. Observar una aplicación paralela desde dos niveles de la pila de software de E/S permite evaluar el impacto de las colectivas, y facilita entender porque se puede generar un cuello de botella u obtener un rendimiento pobre.

También, se ha diseñado una herramienta, PIOM-PX-Trace-Tool, para interceptar las operaciones de E/S a nivel de POSIX-IO; y, un *framework* para obtener los valores de los parámetros correspondientes a nuestro modelo.

Este trabajo aporta a la comunidad científica un análisis de las aplicaciones paralelas MPI a nivel POSIX-IO de la pila de software de E/S, pero también un análisis más detallado de las aplicaciones paralelas que utilizan librerias de E/S paralelas.

PIOM-PX y PIOM sigue basando el análisis del comportamiento de E/S en el concepto de fase de

E/S. Identificar las fases de E/S de una aplicación permite detectar los accesos de E/S que tienen más impacto en el sistema de E/S posibilitando aislar y analizar esas fases una a una.

Con respecto a la plataforma Cloud, se ha propuesto una metodología para que el usuario pueda crear un Clúster Virtual en Cloud (VCC) ya que al trabajar en este entorno el número de parámetros a tener en cuenta para crearlo se ha visto incrementado. Además, se ha comprobado que el Cloud se puede utilizar como banco de pruebas (*Test Bed*), porque se puede cambiar la configuración del sistema de E/S sin afectar a la carga de trabajo.

Asimismo, con este trabajo de investigación se demuestra que extraer el modelo de comportamiento de la E/S de una aplicación paralela es útil por varias razones: primero, porque permite entender el patrón de acceso que utiliza la aplicación así como el tamaño de la petición que se está utilizando en ese patrón y, comprobar que influencia tiene la aplicación sobre un determinado sistema de E/S y viceversa. Segundo, poder evaluar diferentes sistemas HPC, en concreto diferentes configuraciones del sistema de E/S. Tercero, al obtener el modelo de comportamiento de la E/S de una aplicación también permite conocer los requerimientos de E/S que necesita una determinada aplicación para ejecutarse.

7.2. Líneas abiertas

Una línea futura de trabajo será analizar aplicaciones paralelas científicas escritas en otros lenguajes de programación tales como Java y Python para extraer su comportamiento de E/S. Seleccionando este tipo de aplicaciones, se quiere comprobar si las variables de entorno del sistema de E/S se tienen que tener en cuenta como en el caso de aplicaciones escritas en Fortran.

Otra línea futura sería analizar el comportamiento de E/S a nivel de sistema de ficheros para observar qué cantidad de datos es transferida a los dispositivos físicos y cómo son transferidos. Esto permitiría identificar que módulo del sistema de ficheros es más relevante, cómo gestiona realmente el sistema de ficheros los datos transferidos por otras interfaz.

A nivel de Cloud, una línea futura sería evaluar las aplicaciones paralelas científicas con diferentes sistemas de ficheros tales como Lustre para determinar que sistema de ficheros es más adecuado para una determinada aplicación, o para evaluar qué cambios se han de hacer en el sistema de ficheros de un sistema HPC tradicional para que ejecutando una aplicación se obtenga el rendimiento más adecuado.

7.3. Lista de publicaciones

Pilar Gomez-Sanchez, Sandra Mendez, Dolores Rexachs, Emilio Luque.**PIOM-PX: A Framework for Modeling the I/O Behavior of Parallel Scientific Applications.** HPC-IODC: HPC I/O in the Data Center Workshop at ISC 2017 (ISC Workshops 2017): 160-173

Pilar Gomez-Sanchez, Sandra Mendez, Dolores Rexachs, Emilio Luque. **"A parallel I/O behavior model for HPC applications using serial I/O libraries"**, Proceedings of the International Conference on High Performance Computing & Simulation (HPCS 2017): p.244-251.
http://ieeexplore.ieee.org/abstract/document/8035084/

Pilar Gómez Sánchez, Diego Encinas, Javier Panadero, Aprigio Bezerra, Sandra Mendez, Marcelo Naiouf, Armando De Giusti, Dolores Rexachs, Emilio Luque. **"Using AWS EC2 as Test-Bed infrastructure in the I/O system configuration for HPC applications"**. Journal of Computer Science & Technology; vol. 16, no. 2. p. 65-75. ISSN: 1666-6038. (2016).
http://sedici.unlp.edu.ar/handle/10915/57264

Pilar Gomez-Sanchez; Sandra Méndez; Dolores Rexachs; Emilio Luque. **"I/O performance evaluation of parallel scientific applications in a Cloud Environment"**. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA): p.653-659. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). (2015) CSREA Press. ISBN: 1-60132-400-6, 1-60132-401-4 (1-60132-402-2) (July 2015).
http://www.worldcomp-proceedings.com/proc/proc2015/PDPTA15_Final_Edition/PDPTA_Contents_Vol_1.pdf

Gomez-Sanchez P, Méndez S, Rexachs D, Luque E. **"Hopes and Facts in Evaluating the Performance of HPC-I/O on a Cloud Environment"**. Journal of Computer Science & Technology. 15 (01) pp 23-29. ISSN: 1666-6038. (2015).
http://journal.info.unlp.edu.ar/wp-content/uploads/JCST-Apr15-4.pdf

7.4. Agradecimientos

Esta investigación ha sido soportada por MINECO Spain bajo el contrato TIN2014-53172-P y TIN2017-84875-P y parcialmente soportada por CloudMas como competencia gubernamental de AMAZON Web Services (AWS).

El puesto de investigación del estudiante de doctorado Pilar Gomez ha sido financiado por un acuerdo de colaboración de investigación con la "Fundación Escuelas Universitarias Gimbernat".

Pilar Gomez recibió una beca de mobilidad para la investigación de la institución SEBAP (Societat Econòmica Barcelonesa d'Amics del Pais) para financiar su estancia de investigación de tres meses en

el Centro de Supercomputación Leibniz (LRZ, Alemania).

Especial agradecimiento a los recursos proporcionados por el Centro de Supercomputación de Galicia (CESGA, España) y el Centro de Supercomputación de Leibniz (LRZ, Alemania).

Se agradece a Eduardo Almeida Costa, estudiante de doctorado del Programa de Biología y Biotecnología de Microorganismos en la Universidad Estatal de Santa Cruz (Brasil), quien proporcionó la información de los datos para la aplicación ABySS. También, se agradece a Pawel Wichary por su colaboración en la implementación del *plugin PVFS2* para Cloud.

Bibliografía

[1] H. Luu, B. Behzad, R. Aydt, and M. Winslett. A multi-level approach for understanding i/o activity in hpc applications. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–5, Sept 2013.

[2] Steven A. Wright, Simon D. Hammond, Simon J. Pennycook, Robert F. Bird, J. A. Herdman, I. Miller, A. Vadgama, Abhir Bhalerao, and Stephen A. Jarvis. Parallel file system analysis through application i/o tracing. *Computer Journal*, Volume 56(Number 2):141–155, 2013.

[3] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu. On the root causes of cross-application i/o interference in hpc storage systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 750–759, May 2016.

[4] Amazon Web Services and HPC wire.Tabor Custom Publishing. 'cloud-driven hpc', Last Access:2017.

[5] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. An ephemeral burst-buffer file system for scientific applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 69:1–69:12, Piscataway, NJ, USA, 2016. IEEE Press.

[6] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[7] HDF group. High level introduction to hdf5, 2016.

[8] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netcdf: A high-performance scientific i/o interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, pages 39–, New York, NY, USA, 2003. ACM.

[9] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM.

[10] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh. Passion runtime library for parallel i/o. In *Proceedings Scalable Parallel Libraries Conference*, pages 119–128, Oct 1994.

[11] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. Pidx: Efficient parallel i/o for multi-resolution multi-dimensional scientific datasets. In *2011 IEEE International Conference on Cluster Computing*, pages 103–111, Sept 2011.

[12] W. Frings, F. Wolf, and V. Petkov. Scalable massively parallel i/o to task-local files. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, Nov 2009.

[13] MPI-Forum. Mpi-2 standard, 2009.

[14] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing mpi-io portably and with high performance. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, IOPADS '99, pages 23–32, New York, NY, USA, 1999. ACM.

[15] ROMIO Project. Users guide for romio: A high-performance, portable mpi-io implementation.

[16] IEEE and The Open Group. Posix 1003.1.

[17] Prabhat and Quincey Koziol. *High Performance Parallel I/O*. Chapman & Hall/CRC, 1st edition, 2014.

[18] Lustre software release 2.x: Operations manual.

[19] Sandra Mendez, Dolores Rexachs, and Emilio Luque. Analyzing the parallel i/o severity of mpi applications. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '17, pages 953–962, Piscataway, NJ, USA, 2017. IEEE Press.

[20] John M. May. *Parallel I/O for High Performance Computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[21] S. J. Kim, S. W. Son, W. k. Liao, M. Kandemir, R. Thakur, and A. Choudhary. Iopin: Runtime profiling of parallel i/o in hpc systems. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 18–23, Nov 2012.

[22] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. Pattern-driven parallel i/o tuning. In *Proceedings of the 10th Parallel Data Storage Workshop*, PDSW '15, pages 43–48, New York, NY, USA, 2015. ACM.

[23] Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: transputer and occam developments*, volume 44, pages 17–31. IOS Press, 1995.

[24] Barcelona Supercomputing Center. Paraver: a flexible performance analysis tool. *URL http://www.bsc.es/computer-sciences/performance-tools/paraver/general-overview*, 2014.

[25] Karthik Vijayakumar, Frank Mueller, Xiaosong Ma, and Philip C. Roth. Scalable i/o tracing and analysis. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 26–31, New York, NY, USA, 2009. ACM.

[26] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, May 2006.

[27] Julian M. Kunkel, Michaela Zimmer, Nathanael Hübbe, Alvaro Aguilera, Holger Mickler, Xuan Wang, Andriy Chut, Thomas Bönisch, Jakob Lüttgau, Roman Michel, and Johann Weging. *The SIOX Architecture – Coupling Automatic Monitoring and Optimization of Parallel I/O*, pages 245–260. Springer International Publishing, Cham, 2014.

[28] Sandra Méndez, Dolores Rexachs, and Emilio Luque. A methodology to characterize the parallel i/o of the message-passing scientific applications. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 436. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[29] Sandra Méndez, Javier Panadero, Alvaro Wong, Dolores Rexachs, and Emilio Luque. A New approach for Analyzing I/O in Parallel Scientific Applications. In *CACIC 12, Congreso Argentino de Ciencias de la Computación*, pages 337–346, 2012.

[30] Alvaro Wong, Dolores Rexachs, and Emilio Luque. *PAS2P Tool, Parallel Application Signature for Performance Prediction*, pages 293–302. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[31] Babak Behzad, Surendra Byna, Stefan M. Wild, Mr. Prabhat, and Marc Snir. Improving parallel i/o autotuning with performance modeling. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 253–256, New York, NY, USA, 2014. ACM.

[32] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 24/7 Characterization of Petascale I/O Workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

[33] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The vampir performance analysis tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer, 2008.

[34] Pilar Gomez-Sanchez, Sandra Mendez, Dolores Rexachs, and Emilio Luque. *PIOM-PX: A Framework for Modeling the I/O Behavior of Parallel Scientific Applications*, pages 160–173. Springer International Publishing, Cham, 2017.

[35] A. Wong, D. Rexachs, and E. Luque. Parallel application signature for performance analysis and prediction. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2009–2019, July 2015.

[36] William Loewe, Tyce McLarty, and Christopher Morrone. IOR Benchmark, 2012. Accessed: 2016-05-14.

[37] The Centre of Supercomputing of Galicia. Finisterrae ii, 2017.

[38] Leibniz Supercomputing centre. Overview of lrz hpc resources, 2017.

[39] opengroup. The open group base specifications issue 7, 2016.

[40] M. Kerrisk. *The Linux Programming Interface*. No Starch Press Series. No Starch Press, 2010.

[41] MADbench2. https://crd.lbl.gov/departments/computational-science/c3/c3-research/madbench2/, Accessed June 2016.

[42] Parkson Wong and Rob F. Van Der Wijngaart. Nas parallel benchmarks i/o version 2.4. Technical report, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, 2003.

[43] J H Chen, A Choudhary, B de Supinski, M DeVries, E R Hawkes, S Klasky, W K Liao, K L Ma, J Mellor-Crummey, N Podhorszki, R Sankaran, S Shende, and C S Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science Discovery*, 2(1):015001, 2009.

[44] Mingliang Liu, Jidong Zhai, Yan Zhai, Xiaosong Ma, and Wenguang Chen. One Optimized I/O Configuration Per HPC Application: Leveraging the Configurability of Cloud. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, pages 15:1–15:5. ACM, 2011.

[45] RobertoR. Expósito, GuillermoL. Taboada, Sabela Ramos, Jorge González-Domínguez, Juan Touriño, and Ramón Doallo. Analysis of I/O Performance on an Amazon EC2 Cluster Compute and High I/O Platform. *Journal of Grid Computing*, 11(4):613–631, 2013.

[46] Gideon Juve, Ewa Deelman, G. Bruce Berriman, Benjamin P. Berman, and Philip Maechling. An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2. *J. Grid Comput.*, 10(1):5–21, March 2012.

[47] Qais Noorshams, Samuel Kounev, and Ralf Reussner. *Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems*, pages 63–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[48] S. Sivathanu, L. Liu, M. Yiduo, and X. Pu. Storage management in virtualized cloud environment. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 204–211, July 2010.

[49] William D. Norcott. IOzone Filesystem Benchmark, 2006.

[50] AWS-EC2. Amazon Elastic Compute Cloud, Instance Types, 2017.

[51] StarCluster. An Open Source Cluster-Computing Toolkit for Amazon's Elastic Compute Cloud (EC2), 2014.

[52] Sandra Méndez, Dolores Rexachs, and Emilio Luque. Modeling Parallel Scientific Applications through their Input/Output Phases. In *CLUSTER Workshops*, volume 12, pages 7–15, 2012.

[53] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable i/o forwarding framework for high-performance computing systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, Aug 2009.

[54] Thomas Ilsche, Joseph Schuchart, Jason Cope, Dries Kimpe, Terry Jones, Andreas Knüpfer, Kamil Iskra, Robert Ross, Wolfgang E. Nagel, and Stephen Poole. Enabling event tracing at leadership-class scale through i/o forwarding middleware. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 49–60, New York, NY, USA, 2012. ACM.

[55] Spencer Shepler, David Noveck, and Mike Eisler. Network file system (nfs) version 4 minor version 1 protocol. *Network*, 2010.

[56] T. Haynes. Network file system (nfs) version 4 minor version 2 protocol. *Network*, 2016.

[57] Julian Borrill, Leonid Oliker, John Shalf, Hongzhang Shan, and Andrew Uselton. HPC Global File System Performance Analysis Using a Scientific-application Derived Benchmark. *Parallel Comput.*, 35(6):358–373, June 2009.

[58] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inan Birol ç. ABySS: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.