Universitat Autònoma de Barcelona

Escola d Enginyeria

Departament d Arquitectura de Computadors i Sistemes Operatius

Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones

Tesis presentada por **Sandra Adriana Méndez** para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Este trabajo ha sido desarrollado en el departamento de Arquitectura de Computadores y Sistemas Operativos bajo la dirección de la Dra. Dolores Isabel Rexachs del Rosario.

Barcelona, July 2013

Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones

Tesis doctoral presentada por Sandra Adriana Méndez para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Trabajo realizado en el Departament d Arquitectura de Computadors i Sistemes Operatius de la Escola d Enginyeria de la Universitat Autònoma de Barcelona, dentro del Programa de Doctorado en Computación de Altas Prestaciones bajo la dirección de la Dra. Dolores Isabel Rexachs del Rosario.

Barcelona, Julio 2013

Director

Dra. Dolores Isabel Rexachs del Rosario

Dedicado

A la persona que más admiro en este mundo *mi querídisima Mami*, porque me has ensenado a luchar para alcanzar mis metas, porque a pesar de todas las adversidades de la vida me has dado todas la oportunidades para vivir dignamente. Todo lo que hoy soy, sin lugar a dudas, te lo debo a tí.

A mi *amada hija Valeria* por darme las fuerzas para seguir adelante, por hacerme sentir tu carino y respeto. Porque me has permitido cumplir con mis suenos, siempre entendiendome y nunca reprochandome. Sin tu ayuda y apoyo nunca lo hubiese logrado, gracias por ser ese motor que impulsa mi vida. Te quiero muchísimo!

A mis queridos hermanos América, Eusebio, Javier y Rubén, y por supuesto a mis sobrinos. Por cuidar de mi madre y mi hija. Por estar de aquel lado del mundo dandome fuerzas para luchar por lo que quiero.

A mi tío Mario por haber confiado en mi capacidad y haber colaborado con mi educación.

A la memoria de mi padre, mi tía y mis abuelos. Gracias por todo lo que han hecho por mi y mis hermanos. Porque siempre nos estan cuidando! Papá siempre estas con nosotros.

Gracias a todos por la paciencia, por acompanarme y entender cada locura que emprendo y por todo el amor que diariamente me hacen sentir.

Agradecimientos

En primer lugar quiero agradecer a Dios porque siempre me guía y porque ha puesto en mi camino una familia maravillosa y amigos generosos.

Quiero agradecer a todos los que me han acompanado en estos 4 anos y me han brindado su apoyo. Espero no olvidarme de nadie.

A Lola y Emilio por toda la paciencia que me tienen. Porque me han permitido trabajar a mi manera. Por haberme dado la oportunidad de aprender de este maravilloso mundo de la Investigación. A Remo Suppi porque sin conocerme me brindo su ayuda cuando recien llegue a Espana. A Dani Franco por sus consejos de investigación y por ayudarme a familiarizarme con CAOS cuando iniciaba mi larga estancia.

A mis queridos amigos de Jujuy: Marcelo, Hector, Monica, Nilda y Gracielita porque nunca me han faltado, por todo el carino que me han demostrado en cada email que me han enviado y en cada chat que hemos tenido. Y por supuesto a la familia de mis amigos que tanto han hecho por mi y por tenerme presente.

A César, Marcela C., Aprigio y Eduardo porque me han brindado su amistad y me han cuidado. Sin ustedes este camino hubiera sido solitario, gracias por haberme acompanado. A Andrea, Tharso, Carlitos, Joao, Javier P., Hugo, Gonzalo y Claudia por haber escuchado cuando lo he necesitado.

A Marisol, Alex, Carlitos y Eduardo por haber dedicado parte de su tiempo a corregir mi pésimo inglés. Muchas gracias!

Al Ing. José Lucas Sanchéz Mera decano de la Facultad de Ingeniería de la Universidad Nacional de Jujuy por confiar en mi capacidad y permitir mi crecimiento profesional. A Cecilia Lasserre, Sergio Martínez, Julio Tentor, Evelina Wulf, David Sanchéz Rivero por permitirme seguir con mi carrera docente en Jujuy.

A mis companeros de CAOS: Alvaro W., Alvaro Ch., Carlos B., Julio, Tomás A., Toni P., Hai, Arindam, Claudio, Alejandro, Abel, Manuel, Andres, Moni, Diego L., Ronal por hacer de CAOS un agradable lugar de trabajo. Al personal de CAOS: Gemma, Javier y Daniel por toda la ayuda brindada.

Abstract

English

The increase in processing units, the advance in speed and compute power, and the increasing complexity of scientific applications that use high performance computing require more efficient Input/Output(I/O) Systems. The performance of many scientific applications is inherently limited by the input/output subsystem. Due to the historical gap between the computing and I/O performance, in many cases, the I/O system becomes the bottleneck of parallel systems. Identification of I/O factors that influence on performance can help us to hide this gap .

This situation lead us to ask ourselves the next questions: Must adapt Scientific applications to I/O configurations?, How do I/O factors influence on performance? How do you configure the I/O system to adapt to the scientific application? Answering these questions is not trivial. To use the I/O system efficiently it is necessary to know its performance capacity to determine if it fulfills I/O requirements of application.

We present a methodology to performance evaluate of the I/O system on computer clusters. We propose the I/O system evaluation by taking into account requirements of the application and the I/O configuration. To do this we have defined a methodology with three stages: characterization, analysis of I/O and evaluation.

Furthermore, we define a methodology to obtain the I/O model of parallel application that has been utilized to compare different I/O subsystems. It allows us to determine how much of the system s capacity is being used taking into account the I/O phases of application. The methodology can be used to select the configuration with less I/O time from different I/O configurations. The I/O model of application is defined by three characteristics: metadata, spatial global pattern and temporal global pattern. We have evaluated the I/O system utilization of different configurations by considering the I/O model of application.

Keywords: Parallel I/O, I/O Architecture, I/O phases, I/O model, Access Patterns

Castellano

El aumento del número de unidades de procesamiento en los clúster de computadores, los avances tanto en velocidad como en potencia de los procesadores y las crecientes demandas de las aplicaciones científicas que utilizan cómputo de altas prestaciones trae mayores exigencias a los sistemas de Entrada/Salida (E/S) paralelos. El rendimiento de muchas aplicaciones científicas está inherentemente limitado por el sistema de E/S.

Debido al histórico gap entre las prestaciones del cómputo y la E/S, en muchos casos, el sistema de E/S se convierte en el cuello de botella de los sistemas paralelos. La identificación de los factores de E/S que influyen en el rendimiento puede ayudar a ocultar este gap . Esta situación lleva a hacer las siguientes preguntas: Deben las aplicaciones científicas adaptarse a la configuración de E/S, Cómo los factores E/S influyen en el rendimiento? Cómo se configura el sistema de E/S para adaptarlo a la aplicación científica? La respuesta a estas preguntas no es trivial. Para utilizar el sistema de E/S de manera eficiente es necesario conocer su capacidad de rendimiento para determinar si cumple con requisitos de aplicación.

En este trabajo se propone una metodología para evaluar las prestaciones de los sistemas de E/S en clusters de computadores. La evaluación considera la configuración del sistema de E/S y las características de E/S de la aplicación. Para ello se ha definido una metodología de tres etapas: caracterización, análisis de E/S y evaluación. En la fase de caracterización, se extrae las características de E/S de la aplicación y las características prestacionales del sistema de E/S.

Se ha definido una metodología para obtener el modelo de E/S de la aplicación paralela que se ha utilizado para comparar diferentes sistemas de E/S. El modelo de E/S de aplicación está definido por tres características: metadatos, patrón global espacial y patrón global temporal. Esto permite determinar cuanto de la capacidad del sistema se utiliza teniendo en cuenta las fases de E/S de la aplicación. La metodología se puede utilizar para seleccionar la configuración con menor tiempo de E/S de diferentes configuraciones de E/S.

Palabras clave: E/S paralela, Arquitectura de E/S, Patrones de Acceso, Fases de E/S

Índice general

0 European Doctor Mention (Thesis Resume, Introduction and Conclusions) 1

1 Introduction . . . 1
2 Motivation and Thesis Objectives . . . 2
3 Proposed Methodology . . . 3
3.1 Characterization . . . 3
3.2 Input/Output Analysis . . . 12
3.3 Evaluation . . . 15
4 Experimental validation . . . 16
4.1 Experimental Environment . . . 16
4.2 I/O system utilization . . . 18
4.3 Applying the I/O model . . . 21
5 Related Work . . . 31
6 Conclusions . . . 36
7 List of Publications . . . 38

1 Introducción 43

1.1 Motivación . . . 44
1.2 Objetivos . . . 46
1.2.1 Caracterización . . . 46
1.2.2 Análisis de E/S . . . 49
1.2.3 Evaluación . . . 50
1.3 Contribución . . . 51
1.4 Estructura de la Tesis . . . 52

2 Estado del Arte 55

2.1 Caracterización de la E/S de aplicaciones científicas paralelas . . . 56
2.1.1 Patrones de E/S en Aplicaciones Científicas . . . 57
2.1.2 Conceptos de MPI para la manipulación de ficheros . . . 59

2.2 Caracterización del sistema de E/S en clúster de computadores 63
2.2.1 Arquitectura de E/S Paralela 64
2.2.2 Stack de Software de E/S Paralela 71
2.3 Prestaciones de E/S paralela 78
2.4 Trabajos Relacionados 78

3 Metodología para la caracterización de la E/S de aplicaciones científicas paralelas 81
3.1 Metodología propuesta 82
3.1.1 Recopilación de Trazas de la aplicación paralela 82
3.1.2 Procesamiento de trazas y Extracción de los patrones de acceso . . 84
3.1.3 Identificación de fases de E/S y Definición del modelo de E/S . . . 89
3.2 Ejemplos de Modelos de E/S 91
3.2.1 NAS BT-IO 91
3.2.2 MADBench2 95
3.2.3 FLASH-IO Benchmark 97
3.3 Conclusiones 99

4 Metodología para la caracterización del sistema de Entrada/Salida 101
4.1 Metodología Propuesta 102
4.1.1 Identificación de Configuraciones de E/S 102
4.1.2 Evaluación de las características prestacionales 103
4.2 Experimentación 108
4.2.1 Identificación de Configuraciones 108
4.2.2 Evaluación de las características prestacionales 108
4.3 Conclusiones 111

5 Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones 113
5.1 Metodología Propuesta 115
5.1.1 Caracterización 115
5.1.2 Análisis de E/S 116
5.1.3 Evaluación 119
5.2 Experimentación 120
5.2.1 Caracterización del sistema de E/S 120
5.2.2 Caracterización de MadBench2 123
5.2.3 Análisis de E/S y Evaluación 124

5.3 Conclusiones 126

6 Evaluación Experimental **127**
6.1 Entorno de experimentación 127
6.2 FLASH-IO Benchmark 128
6.2.1 Análisis de E/S y Evaluación 130
6.3 Roms - Upwelling 132
6.3.1 Análisis de E/S y Evaluación 134
6.4 Conclusiones 136

7 Conclusiones **141**
7.1 Conclusiones Finales 141
7.2 Trabajo Futuro y Lineas Abiertas 142
7.3 Lista de Publicaciones 142

Bibliografía **147**

Índice de figuras

1 Methodology to performance evaluation of the I/O system in high performance computers 4
2 Format general of an I/O phase 5
3 Extracting I/O abstract model of application 6
4 Traces File ($TraceFile$) 7
5 Global Access Pattern 8
6 I/O Phases ($phase$) 8
7 I/O abstract model example for 4 processes 10
8 I/O abstract model example for 4 processes 10
9 I/O architecture for high performance computers 11
10 Example of I/O model of IOR 15
11 Performance characterized at I/O library on global filesystem 18
12 I/O model of MADBench2 for 36 processes, 40KPIX, and file type SHARED 20
13 I/O Phases for the FLASH-IO for 64 processes 23
14 I/O Phases for the FLASH-IO for 128 processes 24
15 I/O Phases for the FLASH-IO for 256 processes 25
16 I/O Phases for the FLASH-IO for 512 processes 26
17 I/O Model of Upwelling for the first file for 64 processes 28
18 I/O Model of Upwelling for the second file for 64 processes 29
19 I/O Model of Upwelling for the third file for 64 processes 30
20 I/O model of Upwelling for the fourth file for 64 processes 33

1.1 Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones 47
1.2 Componentes de Fases de E/S 48
1.3 Ejemplo de modelo de E/S para NAS BT-IO, 4 procesos, Clase A subtipo FULL 49
1.4 Sistema de E/S desde el punto de vista de la arquitectura y del camino de I/O 50

2.1 Sistema de E/S Paralelo 56

2.2 Patrones globales . . . 58
2.3 Patrón espacial de una aplicación . . . 59
2.4 Particionado de un fichero para varios procesos . . . 61
2.5 Operaciones MPI para acceso a los datos . . . 61
2.6 Sistema de E/S en clúster de computadores . . . 64
2.7 Cada procesador esta conectado a su propio disco local o a discos remotos 65
2.8 Nodo de E/S Independiente. Los nodos de cómputo acceden a disco por medio de los nodos de E/S. . . . 66
2.9 El nodo de E/S se encuentra integrado. Los nodos de Cómputo acceden a los discos por el nodo de E/S. . . . 67
2.10 Gestión del accesos a los dispositivos Centralizada. Un nodo (Nodo de E/S) gestiona el acceso a los discos. . . . 68
2.11 Gestión de Distribuida entre todos los procesadores . . . 69
2.12 Gestión de los dispositivos a cargo de un subconjunto de nodos de E/S. . . 69
2.13 Acceso a los datos de acuerdo a la ubicación y gestión los dispositivos de almacenamiento . . . 71
2.14 Stack de Software de E/S . . . 72
2.15 Librerías de E/S en clúster de computadores . . . 72
2.16 Componentes de Lustre . . . 75

3.1 Proceso para la extracción del modelo de E/S . . . 83
3.2 Traza lógica . . . 84
3.3 Componentes de Fases de E/S . . . 90
3.4 Ejemplo de patrón (LAP) . . . 92
3.5 Fases de E/S para el ejemplo ($phases$) . . . 92
3.6 Patrones de Acceso Espacial para los 4 procesos . . . 94
3.7 Patrones de Acceso Temporal para los 4 procesos . . . 95
3.8 Fases de NAS BTIO, clase C, 16 procesos . . . 96
3.9 Fases de NAS BTIO, clase D, 36 procesos subtipo Full . . . 97
3.10 Comportamiento temporal y espacial para MADBench2 para 16 procesos, 8KPIX, mode de acceso compartido y tipo de acceso secuencial . . . 98
3.11 Fases de E/S de FLASH-IO para 64 procesos . . . 99

4.1 Sistema de E/S desde el punto de vista de la arquitectura y del camino de E/S . . . 101
4.2 Caracterización del Sistema de E/S . . . 104
4.3 Caracterización para los niveles del sistema de E/S . . . 104

4.4 Caracterización a nivel de MPI-IO en OrangeFS del Sistema A 109
4.5 Caracterización a nivel de MPI-IO en NFS del Sistema B 109

5.1 Metodología para evaluar las prestaciones del sistema de E/S en clústeres de computadores 115
5.2 Algoritmo para la generación de la tabla de porcentaje usado, donde Block_size=rs y bw=BW 117
5.3 Algoritmo de búsqueda en tabla de prestaciones, donde Block_size=rs y bw=BW 118
5.4 Caracterización a nivel de MPI-IO en OrangeFS del Sistema A 121
5.5 Caracterización a nivel de MPI-IO en NFS del Sistema B 122
5.6 Fases de E/S para el modelo de E/S de MadBench2 para 36 processes, 40KPIX, y tipo de acceso SHARED 124

6.1 Fases de E/S de FLASH-IO para 64 procesos 130
6.2 Fases de E/S Upwelling para el primer fichero para 64 procesos 134
6.3 Fases de E/S Upwelling para el cuarto fichero para 64 procesos 137

Índice de tablas

1 Summary of notation 5
2 Notations for the I/O Benchmarks 12
3 Input Parameters for the IOR 13
4 Input Parameters for the IOzone 13
5 Output for the I/O Benchmarks 14
6 Description of the I/O subsystems of systems A and B 16
7 Description of the I/O Systems of the Clusters CAPITA, Finisterrae and Supernova 17
8 I/O phases description of MADBench2 for $np = 36$ processes with request size $rs = 352$ in MB 20
9 I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System A 21
10 I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System B 21
11 I/O model of FLASH-IO Write_at_all - First File 23
12 I/O model of FLASH-IO Write_at_all - Second File 24
13 I/O model of FLASH-IO Write_at_all - Third File 25
14 Error of I/O time estimation on Finisterrae for 64 and 128 processes for phase 5 of FLASH-IO 27
15 I/O time estimation on Finisterrae for 64 and 128 processes for phase 5 of FLASH-IO in 4 I/O node and Stripe 4MB 27
16 Phases of I/O model of Upwelling - Write_at_all - First File - Processes 0-7 and 56-63 31
17 Phases of I/O model of Upwelling Write_at_all - First File - Processes 8-55 32
18 Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 0-7 34
19 Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 8-55 35
20 Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 56-63 36

21 Error of I/O time estimation on Finisterrae for 64 processes of the Upwelling 36
22 Structure Lustre on Finisterrae 37

2.1 MPI-IO - File Manipulation 62
2.2 MPI-IO - View File 62

3.1 Resumen de la notación 82
3.2 Descripción de las fases de E/S 93
3.3 Descripción de fases de NAS BT-IO subtipo FULL para np procesos 94
3.4 Desripción de las fases de E/S de MADBench2 para np procesos 97
3.5 Desripción de fases de FLASH-IO para Write_at_all - Primer Fichero . . . 99
3.6 Desripción de fases de FLASH-IO para Write_at_all - Segundo Fichero . . . 100
3.7 Desripción de fases de FLASH-IO para Write_at_all - Tercer Fichero 100

4.1 Paramétros de entrada de IOR 106
4.2 Paramétros de entrada de IOzone 107
4.3 Output for the I/O Benchmarks 107
4.4 Descripción de los sistemas de E/S denominados A y B 108

5.1 Descripción de los sistemas de E/S denominados A y B 121
5.2 Descripción de las fases de MADBench2 para $np = 36$ procesos y tamano de request $rs = 352$ MB 125
5.3 Utilización del sistema, $BW_{(PK)}$ y $BW_{(MD)}$ en MB/segundo para MADBench2 para 36 procesos, tamano de fichero de 102 GB, RS=352MB y tipo de acceso SHARED en el sistema A 125
5.4 Utilización del sistema, $BW_{(PK)}$ y $BW_{(MD)}$ en MB/segundo para MADBench2 para 36 procesos, tamano de fichero de 102 GB, RS=352MB y tipo de acceso SHARED en el sistema B 125

6.1 Descripción de los sistemas de E/S de los clústeres CAPITA, Finisterrae and Supernova 128
6.2 Descripción de fases de FLASH-IO para Write_at_all - Primer Fichero . . . 129
6.3 Descripción de fases de FLASH-IO para Write_at_all - Segundo Fichero . . 130
6.4 Descripción de fases de FLASH-IO para Write_at_all - Tercer Fichero . . . 131
6.5 Tiempos de E/S en Finisterrae para 64 y 128 procesos para la quinta fase de FLASH-IO 131
6.6 Tiempos de E/S en Finisterrae para 64 y 128 procesos para la quinta fase de FLASH-IO en 4 nodos de E/S y tamano de Stripe de 4MB 132

6.7 Descripción de las fases del modelo de Upwelling - Write_at_all - Primer File - Procesos 0-7 and 56-63 135
6.8 Descripción de las fases del modelo de Upwelling - Write_at_all - Primer Archivo - Procesos 8-55 136
6.9 Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 0-7 138
6.10 Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 8-55 139
6.11 Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 56-63 140
6.12 Tiempo de E/S y error relativo en Finisterrae y Supernova para 64 procesos para Upwelling 140
6.13 Estructura de Lustre en Finisterrae - Cuatro OSTs por OSS 140

Chapter 0

European Doctor Mention (Thesis Resume, Introduction and Conclusions)

1. Introduction

The increase in processing units, the advance in speed and compute power, and the increasing complexity of scientific applications that use high performance computing require more efficient Input/Output(I/O) Systems. The performance of many scientific applications is inherently limited by the input/output system. Due to the historical gap between the computing and I/O performance, in many cases, the I/O system becomes the bottleneck of parallel systems. Identification of I/O factors that influence on performance can help us to hide this gap .

The architecture and software of the I/O system are chosen by system administrators. The system administrator or designer has the difficulty either to select components of the I/O system such as the type of devices (HD, SSD, capacity) and their configurations (JBOD, RAID level) or filesystem, interconnection network, among other factors; or to choose from different connection models (DAS, SAN, NAS, NASD) with different parameters to configure (redundancy level, stripe, among others).

The computer clusters are built to provide parallel computing to several applications. These applications have different I/O requirements and the same I/O system is not always appropriate for all applications. Even an application can have different I/O phases and each phase can be adjusted to a I/O system better than another. Programmers can modify their programs to efficiently manage I/O operations, but they need to know the I/O system, especially the I/O software stack.

The system administrators and programmers need information to answer the following questions, When is it convenient to use a local, parallel or distributed file system? When is it convenient to use I/O nodes for the Input/Output management? When is it convenient to use RAID, JBOD or a single disk? When is it convenient to use local storage or remote storage?

Users need information to answer questions like: 1) Is the I/O system a problem for the I/O access patterns of the application? 2) How much I/O system capacity is being used by the application? 3) How can the I/O system be used to improve the application I/O performance?

This situation lead us to ask ourselves the next questions: Must adapt the scientific application to the I/O configuration?, What I/O factors influence on performance? How to configure the I/O system to benefit to the access patterns of the scientific application? Answering these questions is not trivial. To use the I/O system efficiently, it is necessary to know its performance capacity to determine if it fulfills I/O requirements for an application.

This is the context of the present work. The rest of this chapter is organized as follows: Section 2 presents the motivation and the thesis objectives, Section 3 introduces the proposed methodology. In Section 4 the experimental validation is showed, in Section 5 the related works are reviewed, in Section 6 presents the conclusions and future work. Finally, Section 7 presents the list of publications.

2. Motivation and Thesis Objectives

The scientific applications are programmed to use the compute capacity of parallel computer, and exist several methods that can do it, however, there are not enough information to consider the I/O. Usually, the programmers avoid or do not manage explicitly the I/O.

The I/O system should be configured to provide performance taking into account the I/O access patterns of the application, however, this is not possible because the I/O systems are shared for several applications with different I/O access patterns.

If would be possible to adjust the specific parameters of the I/O system to increase the performance for a specific access pattern of the application, this would be a solution to hide the *gap* between the computing and I/O. In this case would be needed to extract the I/O access patterns of the application and to determine the parameters to configure in the I/O system. Identifying if it is possible to configure the I/O system at user or administrator level. This is the main motivation for the present work.

The general objective of this work is define a **Methodology to performance eval-**

uate of the I/O system on high performance computers taking into account I/O characteristics of the application and the I/O configuration.

To do this the following specific objectives are defined:

- Modeling the I/O behavior of the parallel scientific applications
- Defining the steps and the criteria needed to evaluate the I/O system taking into account the different configurations of the computer cluster.
- Defining the steps and criteria needed to analyze, to select and to configure the I/O system taking into account the behavior of the parallel scientific applications

To do this, we have defined a methodology with three phases: characterization, analysis of I/O configuration and evaluation. In the characterization phase, we extract the behavior of the application at I/O level. Also, we extract the performance characteristics of the different I/O systems.

Furthermore, we define a methodology to obtain the I/O model of the parallel application that has been utilized to compare different I/O subsystems. It allows us to determine how much of the system s capacity is being used taking into account the I/O phases of the application. The methodology can be used to select the configuration with less I/O time from different I/O configurations. The I/O model of application is defined by three characteristics: meta-data, spatial global pattern and temporal global pattern.

We have evaluated the I/O system utilization of different configurations by considering the I/O model of application and the I/O system.

3. Proposed Methodology

The proposed methodology is composed of three stages: Characterization, I/O analysis and Evaluation. Figure 1 shows the proposed methodology. Next, we explain each stage.

3.1. Characterization

The characterization is applied to the I/O system and parallel scientific application. This stage has two objectives: i) Extracting the I/O model of application; and ii) Extracting the performance characteristics of the different I/O configurations of the I/O system. These activities are independent. The characterization of application is done off-line and the application I/O model can be applied to analyze different target systems.

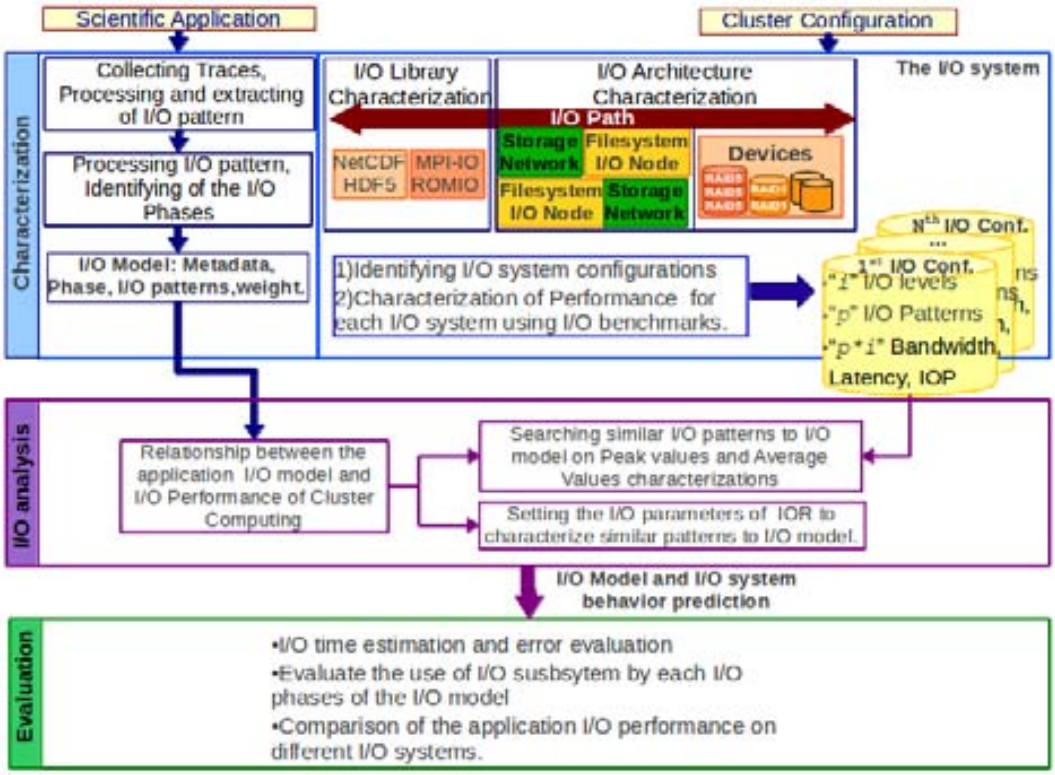


Figure 1: Methodology to performance evaluation of the I/O system in high performance computers

Extracting the I/O model of Scientific Application

The I/O model of application is defined by three characteristics: meta-data, spatial global pattern and temporal global pattern. The application is characterized off-line and once at I/O library level because it provides two important benefits. First, to obtain a model of the application s I/O independent from the execution environment, i.e. the computer cluster. Second, to analyze and evaluate the behavior of the application with different I/O systems, avoiding the overhead of the tracing tool.

The notation utilized to explain the extraction of I/O abstract model is shown in Table 1.

The I/O model of the application is expressed by I/O phases, where an I/O phase is a repetitive sequence of same pattern on a file for a number of processes of the parallel application. A phase will be significant depending on data transferred. A phase is composed by:

- Identifier of phase($IdPh$)
- Identifier of the processes of the phase ($IdsP$)

Table 1: Summary of notation

Notation	Description
np	number of process of parallel application ;
$IdFile$	File identifier ($0 < idF < nF$) ;
$displacement$	Displacement in a position relative of the file (in $Bytes$) ;
rs	request size for the operation (in $Bytes$);
rep	number of repetitions of an access pattern;
$tick$	Logical time unit;
LAP	Local Access Pattern
AP	Access Pattern, similar LAPs for a group of processes
$weight_{(ph)}$	weight of phase $ph = \ rep(simLAP) * rs(simLAP)$

- Number of processes that compose the phase (np)
- request size (rs), type of operation (w/r), distance ($dist$), displacement ($disp$) and $tick$
- number of repetitions (rep), number of I/O operations ($\#iop$) and weight ($weight$).

Figure 2 shows the general format of an I/O phase, also the temporal pattern and spatial pattern that composes the phase.

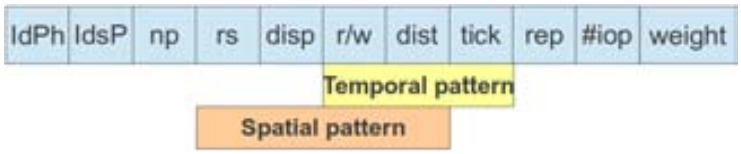


Figure 2: Format general of an I/O phase

The weight depends on the volume of data acceded in the I/O operation during the phase. The unit of the weight is expressed in Bytes. Where the weight is calculated with the equation 1.

$$Weight(IdPh) = np * rs * rep \tag{1}$$

The methodology to extract I/O abstract model of application is presented in Figure 3. This is composed by three steps:

- Collecting Traces of the parallel application
- Processing of traces and extraction of the access patterns

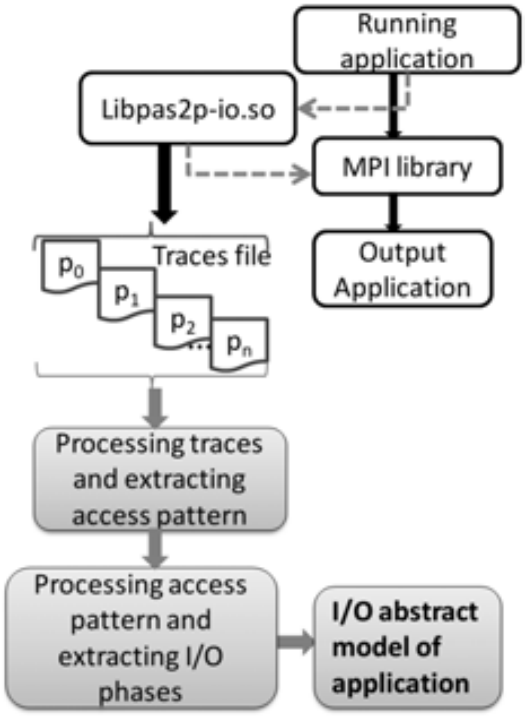


Figure 3: Extracting I/O abstract model of application

- Processing of access patterns, Identification of the I/O phases and definition of the I/O model

The meta-data considered for the I/O model is:

- Access Mode: Strided, Sequential and Random.
- Access Type: Unique (file-per-process) or Shared (shared-process)
- Number of file used for the parallel application
- Data of type of operation related with positioning (explicit offset, individual file pointer, shared file pointer), synchronism (blocking or non-blocking), and coordination (collective or non-collective).

In order to consider the order of occurrence of the events of message-passing parallel applications, it is used the concept of tick. A tick is defined as a logical unit time, and it is incremented by each communication event and I/O event for each parallel process. This tick is adjusted for all the process to order the I/O event for the parallel application. An I/O event is a segment of application where the I/O operation is called. The I/O event is composed of an ID file, mpi-io operation, offset, displacement, size of etype, size of filetype,

name of file, number of event, size of request, logical time, duration, count of datatype, and size of datatype.

The format of traces file is presented in Figure 4. A logical time unit named *tick* (bold in the Figure 4) is used to order the communication and I/O events of MPI. In this case, it only is shown the I/O operation type $MPI\_File\_write\_at\_all$.

IdP	IdF	MPI-Operation	Offset	tick	RequestSize	time	duration
0	1	MPI_File_write_at_all	0	**148**	10612080	22.198392	0.131034
0	1	MPI_File_write_at_all	265302	**269**	10612080	39.101632	0.159706
0	1	MPI_File_write_at_all	530604	**390**	10612080	55.983115	0.131610
0	1	MPI_File_write_at_all	795906	**511**	10612080	72.863806	0.156971
...							
IdP	IdF	MPI-Operation	Offset	tick	RequestSize	time	duration
1	1	MPI_File_write_at_all	0	**147**	10612080	22.198267	0.130836
1	1	MPI_File_write_at_all	265302	**268**	10612080	39.101493	0.160048
1	1	MPI_File_write_at_all	530604	**389**	10612080	55.982996	0.131376
1	1	MPI_File_write_at_all	795906	**510**	10612080	72.863678	0.158459
...							

Figure 4: Traces File ($TraceFile$)

LAP is obtained from traces file for each process MPI. Figure 5 shows the format for the access patterns. Each line in Figure 5 is obtained taking into account the similarity of I/O operation parameters and the *tick*. For example, the first line in bold in Figure 5 means that the process 0 has done 40 writing operations with the same request size (10612080 bytes), displacement (10612080 bytes) and initial offset 0 in its file view, next, the line second for the process 0 shows 40 reading operations with same parameters as writing operations. We can observe this behavior in the four processes of the application. These access patterns allows us to identify the I/O similar operations and the order of occurrence for each process.

The next step is obtain the global behavior of the application. The concept of I/O phase is used to express the global behavior. The *tick* and LAP are used to define the I/O phases. Figure 6 shows the I/O phases for the example. Phase 1 is composed for the four processes MPI with similar access pattern $simLAP$ and similar *tick*. This phase has $weight = 40MB$. Phase 2 is similar to Phase 1 that occurs past 122 *tick* of Phase 1. The difference is the offset that is calculated with the displacement *disp* and the $initOffset$. We identify $f(initOffset)$ to express the $initOffset$ of each process and this function is used in the I/O abstract model of the application.

Global access pattern is obtained from I/O phases of *np* processes for the application *app* for each nF files.

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
0	**1**	**MPI_File_write_at_all**	**40**	**10612080**	**265302**	**0**
0	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
1	**1**	**MPI_File_write_at_all**	**40**	**10612080**	**265302**	**0**
1	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
2	1	MPI_File_write_at_all	40	10612080	265302	0
2	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
3	1	MPI_File_write_at_all	40	10612080	265302	0
3	1	MPI_File_read_at_all	40	10612080	265302	0

Figure 5: Global Access Pattern

Phase 1

IdP	IdF	MPI-Operation	Offset	tick	RequestSize
0	1	MPI_File_write_at_all	0	**148**	10612080
1	1	MPI_File_write_at_all	0	**147**	10612080
2	1	MPI_File_write_at_all	0	**147**	10612080
3	1	MPI_File_write_at_all	0	**147**	10612080

Phase 2

IdP	IdF	MPI-Operation	Offset	tick	RequestSize
0	1	MPI_File_write_at_all	265302	**269**	10612080
1	1	MPI_File_write_at_all	265302	**268**	10612080
2	1	MPI_File_write_at_all	265302	**268**	10612080
3	1	MPI_File_write_at_all	265302	**268**	10612080

...

Figure 6: I/O Phases (*phase*)

Spatial global pattern is represented by the $f(initOffset)$, displacement, and request size.

Temporal global pattern is represented by the *tick* and the local access patterns LAP. Due to that LAP was ordered by *ticks* we also can obtain the temporarily of LAP in each process and the ordering of LAP for the np processes.

An I/O phase is one part of Global access pattern, where LAP are similar. LAP are similar if LAP has the similar value for a number of processes of the application, except to *initial Offset* because each process usually works in different part of file to take advantage of parallel I/O. The significance of an I/O phase in the application is valued through its *weight*. The *weight* of a phase depends on repetitions of operations rep, number of process in the phase, and request size rs. The algorithm 1 show the process to identify the

I/O phases.

Algorithm 1: Identifying the I/O phases.

```
Algorithm Identifying the I/O phases
[Input: Files of Local Access Patterns of the processes
 of the application]
[Output: Files of the I/O phases]

Open Files of Local Access Patterns
IdPh = 1;
While (!EOF(files))
 if (firt LAP)
   if (Similar LAP and tick )
    LAP[IdPh] = LAP[i]
    tick[IdPh] = tick
    repPh[IdPh] = 1
   endif
 else
  if (Similar ( LAP and tick) ) and
  (LAP is similar to LAP[i-1] ) then
   repPh[IdPh] = rep[IdPh] + 1
  else
    LAP[IdPh] = LAP[i]
    tick[IdPh] = tick
    repPh[IdPh] = 1
 endif
 endAlgorithm
```

Figure 7 shows the spatial pattern and Figure 8 shows the temporal pattern of the I/O model of the example, where the global access pattern is shown through its spatial local pattern, spatial global pattern, temporal local pattern, and temporal global pattern. Also, we show the global access pattern in three dimensional space, where *file Offset* indicates the position where the *process* p is accessing in the logical time *tick* with a request size *rs*. In the global access pattern, first red dot of the four processes represent the Phase 1, the second red dot represent Phase 2 and so on to Phase 40. Next, the four processes do 40 reading operations in a phase (blue in Figure 8). We have considered a phase of reading operations because there are not other MPI events between the reading operations. Due

this the Phase 41 is similar to a vertical blue line.

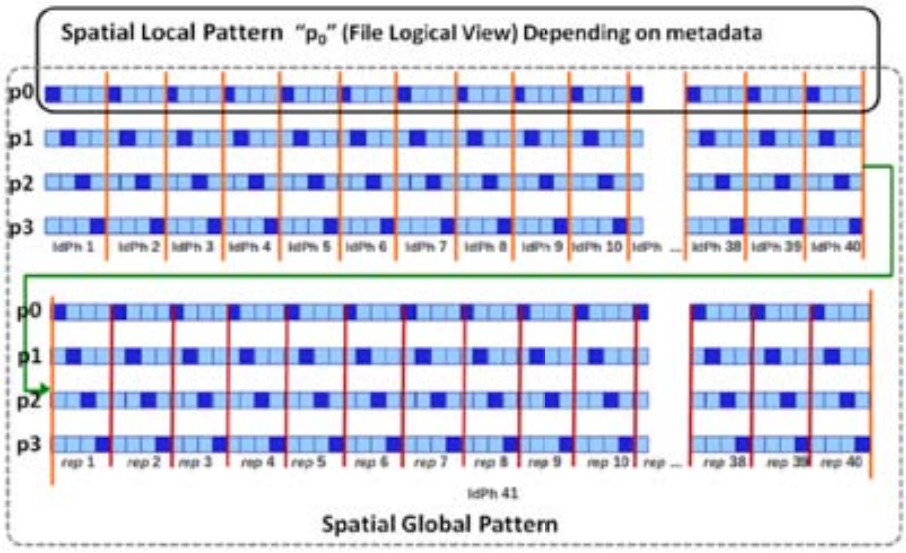


Figure 7: I/O abstract model example for 4 processes

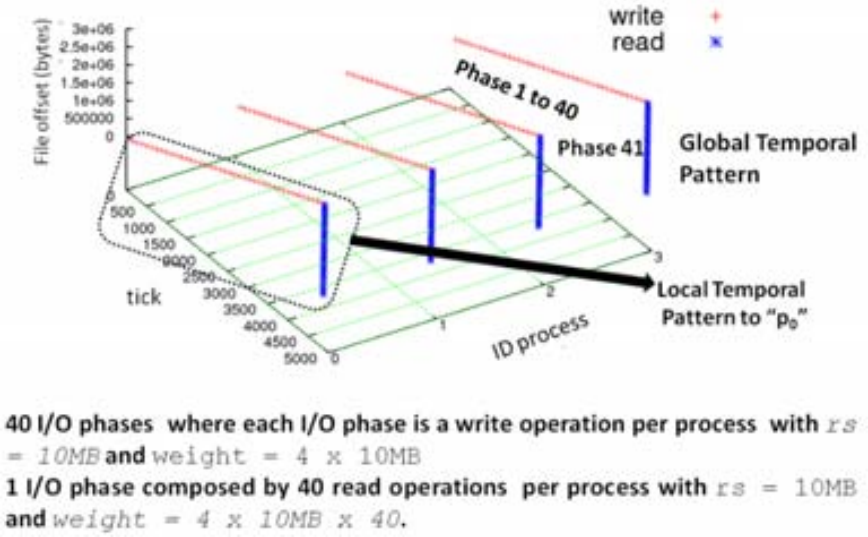


Figure 8: I/O abstract model example for 4 processes

Furthermore, in the extracting of local access patterns, we also consider the meta-data of each file of the application because this allows us to define the different logical views of the files and it obtains the logical global view of the file for the processes of application. For the example, the access mode is "strided"because the application uses $MPI\_File\_set\_view$ for the four processes. Therefore, the offset for each process is about its logical view. For example, it is identified the spatial pattern of Figure 7, where in Phase 1 (#1) each process writes a portion of the file (blue boxes), in Phase 2 (#2) each process writes in the position

$offset + disp$ but this is in the file logical view of each process for this reason it can observe a strided access in the spatial access pattern.

I/O System

The I/O system is structured as a hierarchical scheme to give ordering to the evaluation process. This structure takes into account the hardware and software stack of the I/O system of the high performance computers (Figure 9)

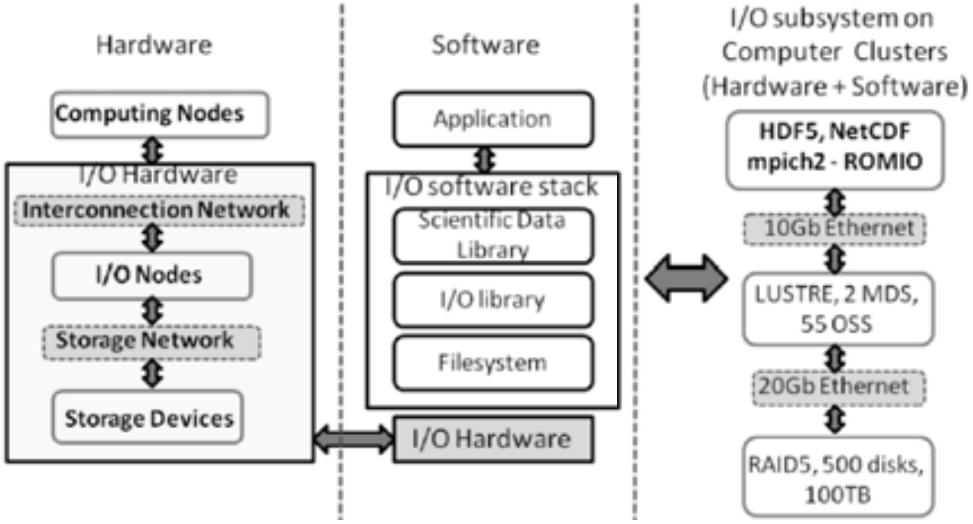


Figure 9: I/O architecture for high performance computers

Table 2 shows the notation used in this stage.
In the I/O system characterization, the following steps are applied:

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

- Setting input parameters for the Benchmarks: IOR [1] is applied at I/O library level and Global Filesystem level. IOzone [2] is applied at I/O devices on local filesystem. Parameters values are selected according to the characteristics of the configurations identified. We have selected different parameters of IOR and IOzone for our methodology. Table 3 shows parameters selected to IOR.

Table 2: Notations for the I/O Benchmarks

Notation	Description
FZ	File Size: size of file to test. $minimumsize = 2 * RAMsize$, RAM size of node where the benchmark will be executed;
RS	Request Size. RS can be from KB to GB depending on file size and transfer rate;
IOP	Input/Output operations type. $value = write, read, meta - oper$;
$\#IOP$	Number of I/O operations;
NP	Number of Processes that will do the IOP;
AM	Access Mode: Form of access to a file. $Value = sequential, strided, random$;
AT	Access Type: Number of file per process. $Value = shared, unique$, where shared is one file for all processes, unique is one file per process;
CIO	Collective I/O;

Table 4 shows selected parameters to IOzone. Where ION is I/O Node, CN is Computing Node.

- Selecting output for the Benchmarks: Output parameters depend on I/O requirements. The metrics are the I/O time, number of operations, transfer rate, and latency. Each benchmark has different options to select metrics. Table 5 shows outputs for the two benchmarks.

- Identifying I/O devices to monitor: when it is necessary to know the I/O operations at I/O devices level it can define the units to monitor. We have selected the Linux monitoring tool *iostat* for the disks monitoring that is done in each I/O node for each partition with *iostat -x -p 1*.

3.2. Input/Output Analysis

In this step, we search the similarity between the LAP of the I/O model phases and the patterns used to characterize the performance of the I/O system obtained in the characterization. This is done to obtain the performance characteristics of the I/O system for a specific I/O model. To search the similarity, the following steps are performed when there is an exhaustive characterization:

Table 3: Input Parameters for the IOR

Parameters	Parameters Values
FZ	$NP * s * b$; where $-s$ $IntegerNumber$ sets the number of segments and $-b$ $Number(k, m, g)$ sets contiguous bytes to write per process.
RS	$-t$ $Number(k, m, g)$; where $-t$ sets the request size.
NP	$mpirun -n$ NP $-a$ $MPIIO$ sets the API to MPIIO mode for NP processes. The API for I/O can be [$POSIX$ $MPIIO$ $HDF5$ $NCMPI$]
AM	$Default$ to Sequential, $Not\ Working$ to Strided, $-z$ to Random
AT	$Default$ to Shared , $-F$ to Unique
IOT	$Default$ without collective I/O, $-c$ to enable collective I/O.

Table 4: Input Parameters for the IOzone

Parameters	Parameters Values
FZ	$-s$ FZ where $-s$ sets the file size to FZ.
RS	$-y$ RS sets Minimum request Size to RS.
AM	$-i$ 0 $-i$ 1 to Sequential, $-i$ 0 $-i$ 5 to Strided and $-j$ # Set stride of file accesses to $(\# * RS)$, $-i$ 0 $-i$ 2 to Random.
AT	$Default$ to Shared , $-F$ to Unique

- Opening the file of performance and setting the variable found to stop the searching when the values are found.
- If the operation type, access mode, and access type is equal to a value in the performance table, and the request size of the I/O phase is:
  - less than minimum request size of the performance file then it selects the transfer rate corresponding to minimum request size.
  - greater than maximum request size of performance table then, it selects the transfer rate corresponding to the maximum request size.
  - equal to a request size of the performance table then it selects the transfer rate corresponding to such request size.
  - a value between the characterized values then it selects the closest upper value to the searched value.
- When the search finishes then the performance table is closed and the transfer rate is returned.

From this process, it obtains the expected transfer rate denoted by $BW_{(CH)}$ for the different access patterns that compose I/O phases. With this information it is posible estimate the I/O time $Time_{io(CH)}$ for the application.

Table 5: Output for the I/O Benchmarks

Metric	Notation	Unit
Mean Writing Time	$T_{io(w)}$	seconds
Mean Reading Time	$T_{io(w)}$	seconds
Mean Writing operations per second	$IOPs_w$	*double number*
Mean Reading operations per second	$IOPs_r$	*double number*
Mean Transfer rate for the writing operations	BW_w	MB/sec
Mean Transfer rate for the reading operations	BW_r	MB/sec

When there is not a characterization of I/O system for similar patterns as phases of I/O model, it uses IOR to achieve similarity with the I/O model of the application. The I/O model is used to set up the input parameters of the benchmark IOR [3]. The benchmark is configured and run for the phases with higher weight or for the phases that represent the 90 % of I/O.

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

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

The metric selected of IOR is transfer rate expressed in MB/sec and named $BW_{(CH)}$. The I/O time estimate by I/O phase for each IOP type for the application is calculated by expression (2).

$$Time_{io} = \sum_{i=1}^{n} Time_{io}(phase[i]) \tag{2}$$

Where the $Time_{io}(phase[i])$ is calculated by expression (3).

$$Time_{io}(phase[i]) = \frac{weight_{(phase[i])}}{BW_{(CH)}(phase[i])} \tag{3}$$

For instance, Figure 10 shows an I/O model of IOR, where are identified a writing phase and a reading phase into the global access pattern.

The I/O model is used to determine the I/O capacity of system and determine what system can provide best performance to the application.

To determine how much of the I/O subsystem capacity is being used, we measure the peak value of each I/O configuration that is named $BW_{(PK)}$. That is obtained by benchmark IOzone which is executed in each I/O configuration identified. Peak value is

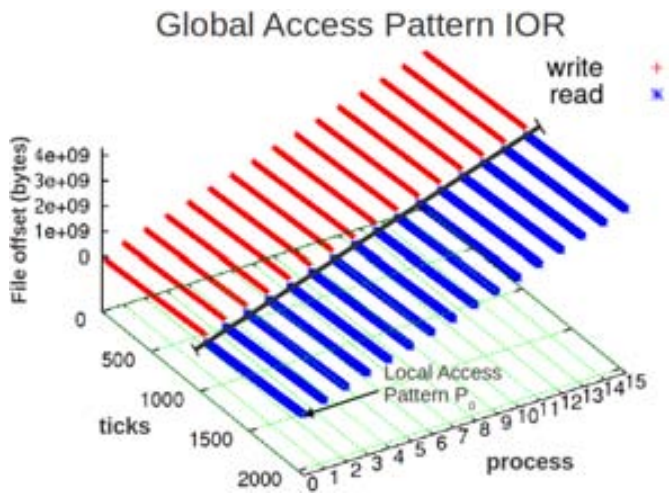


Figure 10: Example of I/O model of IOR

the maximum value obtained by the IOzone for the different access patterns and a peak value is calculated by IOP.

Peak value is calculated by expression (4);

$$BW_{(PK(ION[i])} = maxBW(ION_{(i)}) \tag{4}$$

where i $0..\#NIO$, $(\#NIO)$ is the number of I/O nodes, and $maxBW(ION_{(i)})$ is the maximum value obtained by IOzone for the IOP. In I/O configurations with parallel filesystems the peak value is calculated by expression (5).

$$\left(BW_{(PK)} = \sum_{i=0}^{\#NIO} maxBW(ION_{(i)})\right) \tag{5}$$

3.3. Evaluation

It evaluates the utilization of I/O system by the relation between the bandwidth characterized $BW_{(PK)}$ at I/O devices level and measured $BW_{(MD)}$ expressed in equation 6.

$$SystemUsage(phase[i]) = \frac{BW_{(MD)}(phase[i])}{BW_{PK(IOP(fase[i]))}} * 100 \tag{6}$$

It also evaluates the relative error produced by the I/O time estimation. This is necessary to evaluate how much our configuration selecting can be deviated. Relative error is calculated by expression (7);

$$error_{rel} = 100 * (\frac{error_{abs}}{BW_{MD}(phase[i])}) \tag{7}$$

Table 6: Description of the I/O subsystems of systems A and B

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

where absolute error is calculated by the expression (8).

$$error_{abs} = BW_{(CH)}(phase[i]) - BW_{MD}(phase[i]) \quad (8)$$

When a phase has two or more I/O operations then the $BW_{(CH)}$ is calculated how the average of the $BW_{(CH)}$ of each I/O operation that composes the I/O phase.

4. Experimental validation

4.1. Experimental Environment

It presents the experiments in two part: 1) it extracts the I/O model of MadBench2 [4] and it is applied to evaluate the utilization of the two I/O subsystems. This approach is adequate when there is an exhaustive characterization of the I/O subsystem. 2) It extracts the I/O model of Flash-IO benchmark [5] and it is used to tune in parameters of IOR benchmark in order to evaluate the I/O performance for the I/O model. This approach is adequate when there is not an exhaustive characterization of the I/O subsystem. Table 6 shows the I/O systems of the System A and B.

System A is composed of 14 computing nodes:

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

System B is composed of 32 IBM x3550 Nodes:

Table 7: Description of the I/O Systems of the Clusters CAPITA, Finisterrae and Supernova

I/O Element	CAPITA	Supernova	Finisterrae
I/O library	mpich2	mvapich2	mpich2, HDF5
Communication Network	1 Gbps Ethernet	Infinibad 20 Gbps	Infinibad 20 Gbps
Storage Network	1 Gbps Ethernet	Infinibad 20 Gbps	Infinibad 20 Gbps
Filesystem Global	OrangeFS	Lustre 2.1.5	Lustre (HP SFS)
I/O nodes	10	8 OSS	18 OSS
Metadata Server	1	2 MDS	2 MDS
Filesystem Local	Linux ext3	Linux ext3	Linux ext3
Level Redundancy	-	RAID 6	RAID 6
Capacity of I/O Devices	500 GB	485TB	216TB
Stripe Size	64 KB	1 MB	4MB
Mounting Point	/mnt/orangefs	/lustre/scratch	$HOMESFS

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

The I/O models are presented in a progressive way, starting as a single I/O model and finally present a complex I/O model from a real application. The I/O model has been extracted in subsystem A and the I/O model it is applied in Finisterrae [6]. Table 7 shows the I/O architecture of two systems.

Cluster CAPITA is composed of 14 computing nodes:

- 4 cores of AMD Phenom™ II (8MB cache) or Athlon™ II (2MB cache)
- 4 DIMM slots for up to 16GB DDR3 UDIMM ECC; 2 DIMMs per channel
- 500GB SATA Disk

Finisterrae is composed of 143 computing nodes:

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

Supernova is composed of 570 compute nodes:

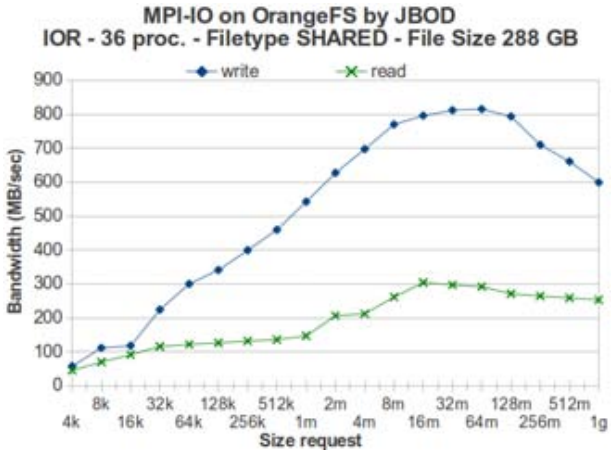
- 126 Intel Xeon E5345 2.33 GHz nodes with 8 (2x quad-core) and 16GB of memory.
- 40 Intel Xeon L5420 2.5 GHz nodes with 8 (2x quad-core) and 16GB of memory.
- 404 Intel Xeon X5650 2.67 GHz nodes with 12(2x six-core) and 24GB of memory.

4.2. I/O system utilization

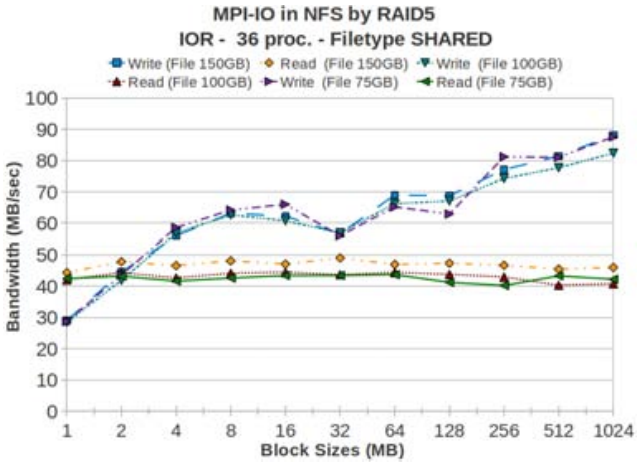
This section is applied to MadBench2 in the system A and B

I/O system Characterization

Figure 11 shows performance measured with IOR at I/O library level for the I/O system of A and B.



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



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

Figure 11: Performance characterized at I/O library on global filesystem

The I/O subsystem A (Figure 11(a)) shows a increasing I/O performance with the increment of request sizes for the writing operations and reading operations. For the request sizes upper to 16 MB we can observe a performance drop for reading operations. For the writing operations we can observe a drop in performance for the request sizes upper to 64MB. The I/O subsystem B (Figure 11(b)) shows a regular behavior for the

reading operations regardless of the file size and request sizes. However, for the writing operations the request size is the I/O factor with the highest impact in performance. We can observe greater transfer rates for bigger request sizes ($> 256MB$), we also can observe a drop in performance for request size of 32 MB.

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

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

Characterization of MadBench2

To evaluate the system usage we analyze the I/O phases to MADBench2 in the I/O subsystems of system A and B. MADbench2 is a tool for testing the overall integrated performance of the I/O, communication and calculation subsystems of massively parallel architectures under the stress of a real scientific application. MADbench2 is based on the MADspec code, which calculates the maximum likelihood angular power spectrum of the Cosmic Microwave Background radiation from a noisy pixelated map of the sky and its pixel-pixel noise correlation matrix.

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

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

We have obtained the I/O model of MADBench2 for 36 processes. Figure 12 shows I/O model of MADBench2 for 36 processes, 40KPIX, and file type SHARED.

Table 8 shows the five phases identified.

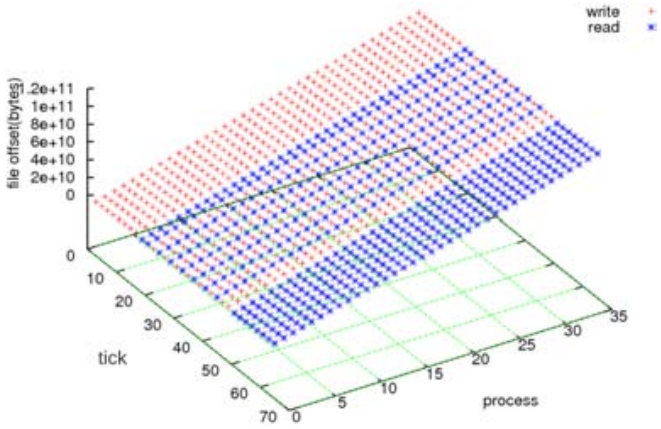


Figure 12: I/O model of MADBench2 for 36 processes, 40KPIX, and file type SHARED

Table 8: I/O phases description of MADBench2 for $np = 36$ processes with request size $rs = 352$ in MB

ID_Phase	np	**Operation**	rs	rep	$weight$
1	36	write	352MB	8	102GB
2	36	read	352MB	2	25GB
3	36	write	352MB	6	75GB
	36	read	352MB	6	75GB
4	36	write	352MB	2	25GB
5	36	read	352MB	8	102GB

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

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

Table 10 shows the utilization of the I/O subsystem of the system B. It also shows the amount of data transferred in each I/O phase ($weight$), the number and type of I/O operation (W=write, R=read, W-R=write-read), $BW_{(MD)}$ and $BW_{(PK)}$ in MB/second.

Table 9 shows the utilization of the I/O subsystem A for the MADBench2 to 40KPIX and 36 processes. It can be observed that the I/O phases 1 and 4 (with writing operations) have utilized greater performance capacity than phases 2, 3 and 5 (phases with reading operations or composed). The third phase has used at about 31 % of the performance capacity, a percentage similar to phases with reading operations that have used at about 20 %.

Table 10 shows the utilization of the I/O subsystem for the MADBench2 to 40KPIX and 36 processes. It can be observed that the I/O phases 1 and 4 (with writing operations)

Table 9: I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System A

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

Table 10: I/O system utilization, $BW_{(PK)}$ and $BW_{(MD)}$ in MB/second for MADBench2 with 36 processes, file size 102 GB, RS=352MB and a shared file on System B

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

have utilized higher I/O performance capacity than phases 2, 3 and 5 (phases with reading operations or composed). The third phase has used at about 16 % of the performance capacity, a percentage similar to phases with reading operations that have used at about 15 %.

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

4.3. Applying the I/O model

In this section we present the I/O model of two applications. The aim of each experimentation is to extract the I/O model and select I/O phases with more weight to evaluate in other I/O subsystem. Also, to determine how this info can be useful for the user to choose the right I/O subsystem.

FLASH-IO Benchmark

FLASH-IO Benchmark [5] simulates the I/O employed by FLASH for the purposes of benchmarking the code. FLASH is a block-structured adaptive mesh hydrodynamics code. The computational domain is divided into blocks which are distributed across the processors. Typically a block contains 8 zones in each coordinate direction (x,y,z) and a perimeter of guard-cells (presently 4 zones deep) to hold information from the neighbors. The code will produce a checkpoint file (containing all variables in 8-byte precision) and two plot files (4 variables, 4-byte precision, one containing corner data, the other containing cell-centered data). The checkpoint and plot file routines are identical to those used in the FLASH Code.

Extracting the I/O model

It has obtained the following metadata of FLASH-IO in the parallel HDF5 version with our tool:

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

FLASH-IO does only I/O operations and there are not communication events. I/O operations of parallel HDF5 are converted in MPI primitives. Table 11 shows the description of I/O phase. Where IdPh. is Identifier of Phase, Oper. is Type of operation, *RS* is Size of request, Iter. is the number of iterations, Dist. is Distance, Disp. is Displacement, and 1°Tick is the first tick (1°Tick) of the phase. Distance is the number of events of communication/IO between two phases. The first tick is the number of tick where is done the first pattern of the phase. Displacement defines the location where a view begins, this is the file displacement.

It can be observed five phases for the first file. In this case the are several MPI_File_set_view to achieve strided access. There are two types of writing operations: MPI_File_write_at and MPI_File_write_at_all. It shows phases for the collective operations because represent the 90 % of I/O. The I/O Phases for the FLASH-IO are shown in Figure 13 for 64 processes, Figure 14 for 128 processes, Figure 15 for 256 processes, and Figure 16 for 512 processes. It observes that the behavior is similar to the different numbers of processes.

The I/O model shows three file used during the run of application. Files are enumerated taking into account the order that were opened.

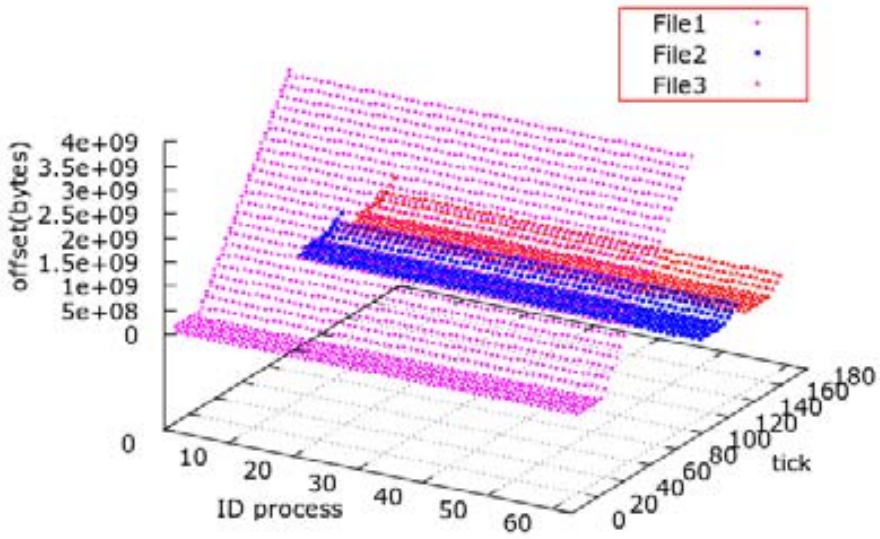


Figure 13: I/O Phases for the FLASH-IO for 64 processes

For the following experiments were applied to 64 processes. Table 11 shows description of phases for first file, Table 12 shows description of phases for second file, and Table 13 shows description of phases for third file. We can observe small request size for the phases 1 to 4 for the three files, 2MB for first file and 1MB for the second and third file.

Table 11: I/O model of FLASH-IO Write_at_all - First File

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	1920	2	3	124392	122KB
4	64	write	3840	1	3	439012	245KB
5	64	write	2621440	24	3	169869312	4GB

It uses the I/O model of FLASH-IO to set the parameters of IOR. It selects phase 5 to configure IOR because is phase with more weight. The parameters are set as follows:

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

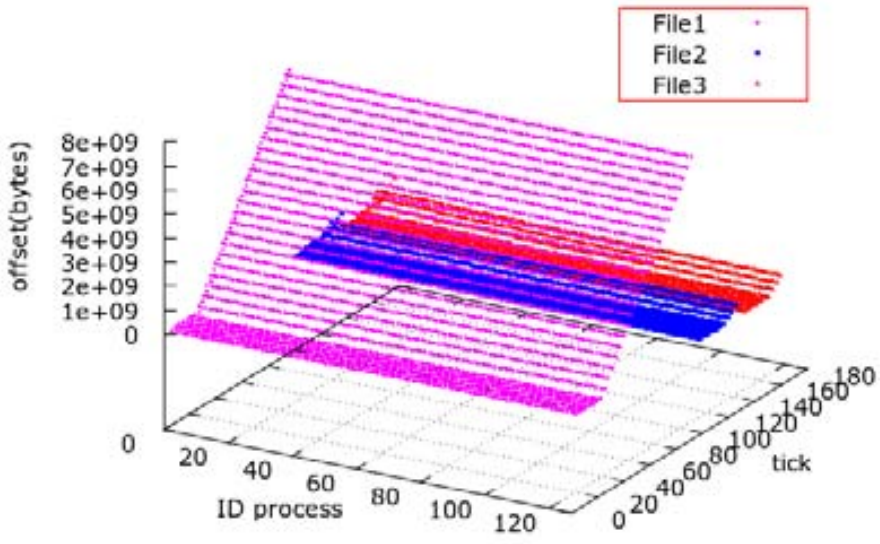


Figure 14: I/O Phases for the FLASH-IO for 128 processes

Table 12: I/O model of FLASH-IO Write_at_all - Second File

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	128KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1310720	4	3	84934656	335MB

Due to that the order of access in files is File1, File2 and File3, we run IOR three times in that order.

We have evaluated the I/O time of IOR and I/O time of flash in the cluster Finisterrae. Table 14 show the I/O time obtained with IOR $Time_{io(CH)}$, I/O time for FLASH-IO $Time_{io(MD)}$, and relative error $error_{rel}$. We can observe high errors for 64 processes in the File 2 and File 3, this is due to the size of Files (400MB). However, we can observe that error is decreased in 128 processes for the File 1 that is the file with higher size. Furthermore, we have evaluated the impact of number of I/O nodes to FLASH-IO (Table 15). Analyzing the I/O of FLASH-IO, we have executed FLASH-IO with writing enabled to work in four I/O nodes and stripe size of 4MB in Lustre. We consider a configuration appropriate of 4 nodes where sixteen processes have accessed in each I/O node. Request

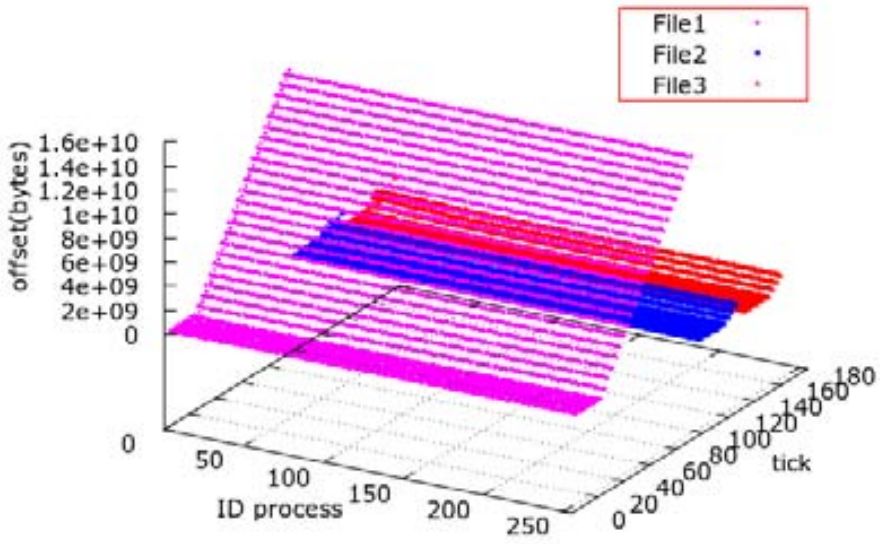


Figure 15: I/O Phases for the FLASH-IO for 256 processes

Table 13: I/O model of FLASH-IO Write_at_all - Third File

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	122KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1572160	4	3	101974016	402MB

size selected is of the phase with higher weight. We can observe that the I/O time with this I/O configuration is decreased in a 33 % for the File1, 12 % for the File2, and 11 % for the File3 in 68 processes. For 128 processes the I/O time is reduced in 23 % for the File1, 31 % for the File2, and 35 % for the File3. Also, we can observe the similar decreasing for the mimic with IOR in 128 processes.

Roms - Upwelling

ROMS [7] is a free-surface, terrain-following, primitive equations ocean model widely used by the scientific community for a diverse range of applications. ROMS includes accurate and efficient physical and numerical algorithms and several coupled models for biogeochemical, bio-optical, sediment, and sea ice applications. It also includes several

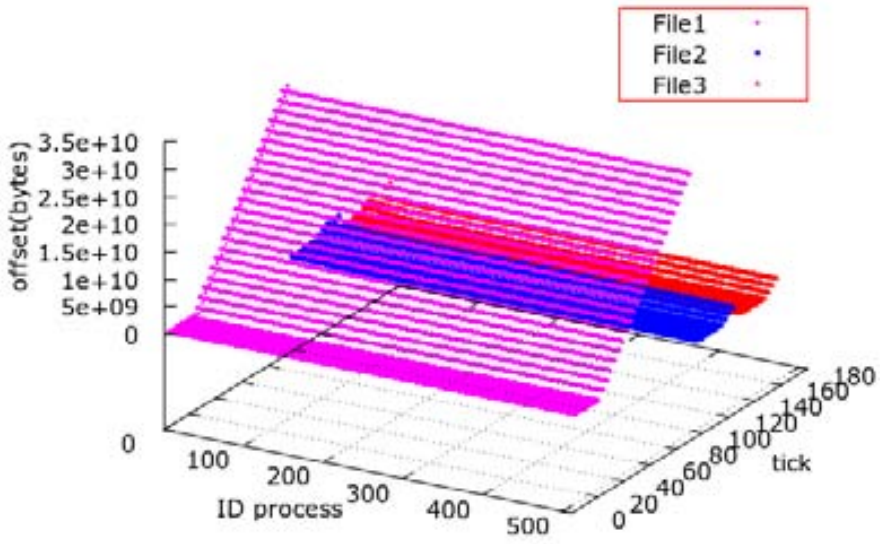


Figure 16: I/O Phases for the FLASH-IO for 512 processes

vertical mixing schemes, multiple levels of nesting and composed grids.

We have evaluated the upwelling that consists of a periodic channel with shelves on each side. There is along-channel wind forcing and the Coriolis term leads to upwelling on one side and downwelling on the other side.

We have used Upwelling with enabled parallel I/O option. This option enables writing parallel through NetCDF parallel. NetCDF parallel work on MPI-IO. Upwelling writes four file each five step. We have configured Upwelling to 64 processes (NtileI == 8, NtileJ == 8) and for 9600 iterations (NTIMES==9600).

Extracting the I/O model

We have obtained the following meta-data of Upwelling with our tool:

- Explicit offset, Blocking I/O operations, Collective operations and Non-collective.
- Strided access mode, Shared access type for four files (ocean_avg.nc, ocean_dia.nc,ocean_his.nc, and ocean_rst.nc).
- MPI-IO routine MPI_Set_view with different etype and filetype for collective and non-collective operations.

Table 14: Error of I/O time estimation on Finisterrae for 64 and 128 processes for phase 5 of FLASH-IO

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

Table 15: I/O time estimation on Finisterrae for 64 and 128 processes for phase 5 of FLASH-IO in 4 I/O node and Stripe 4MB

	$Time_{io(CH)}$	$Time_{io(MD)}$	$error_{rel}$
64p			
File 1	30.39	35.18	13 %
File 2	1.97	4.06	50 %
File 3	2.35	4.27	44 %
128p			
File 1	79.52	78.12	2 %
File 2	5.67	5.78	2 %
File 3	8.11	7.20	12 %

- File 1, 2 and 3 have an offset incremental for each I/O operation. However, fourth file has the same offset for the different I/O operations. We can observe this behavior in its I/O model. In fact, size of fourth file is 4.8MB, although the total size moved for this file is 73MB.
- The four files make a MPI_File_sync before of each iteration for the I/O phases with more weight.

First, second (Figure 18) and third (Figure 19) file have a similar I/O model where the difference is the weight and offset. We have selected first and fourth file to show in detail their phases.

The I/O model for the first file is shown in Figure 17. We can observe that the first eight processes and eight last processes writing in similar tick, request and offset. The processes 8 to 55 are grouped in other phase due to difference of patterns. Table 16 shows description in detail for the processes 0-7 and 56-63 where we can observe that phase 12 is

performed 133 times. In this case there is a MPI_File_sync before each iteration of this phase.

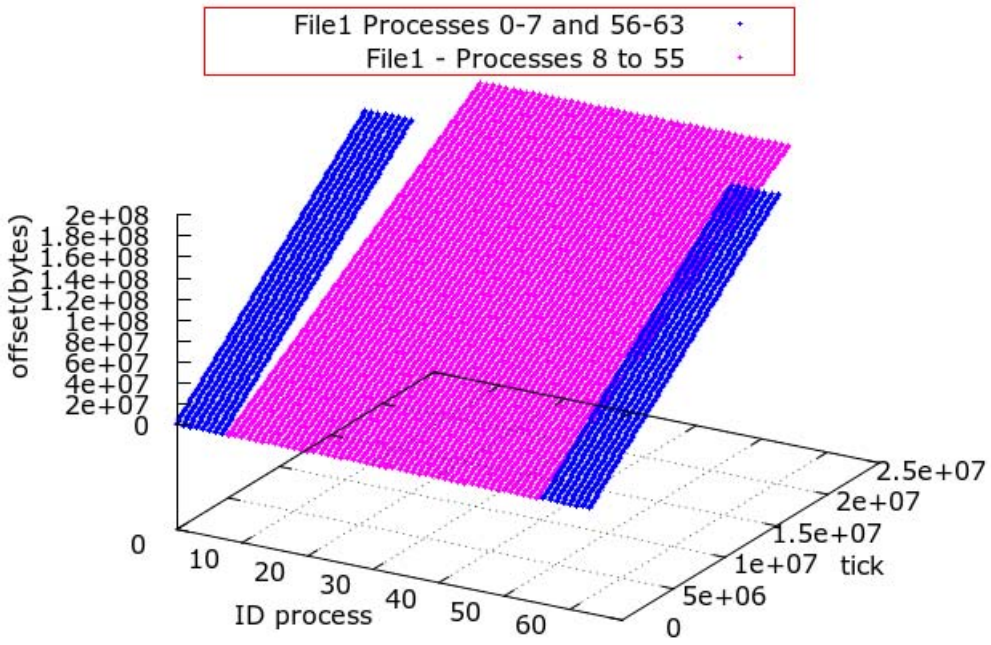


Figure 17: I/O Model of Upwelling for the first file for 64 processes

Table 17 shows phases for the processes 8 to 55, with similar behavior and a MPI_File_sync before each iteration of phase 12. The offset for each iteration of Table 16 and Table 17 is calculated by expression

(9).

$$OInit_{(12.i,iter)} = OInit_{(i,iter-1)} * i * 1416004 \quad (9)$$

Where i $1 - 12$ and $iter$ $2 - 133$.

The I/O model for fourth file is shown in Figure 20. In this case the offset is the same in each iteration. We can observe that first eight processes and eight last processes write in similar ticks, request and offset. The processes 8 to 55 are grouped in other phase due to difference of patterns.

Table 18 and 20 shows description in detail for the processes 0-7 and 56-63 where we can observe that phase 12 is performed 16 times. Table 19 shows phases for the processes 8 to 55, with similar behavior and a MPI_File_sync before each iteration of phase 12.

The I/O model of Upwelling is used to set the parameters for the IOR. We have selected phase 12 to mimic the I/O model because it is the phase with more weight. The

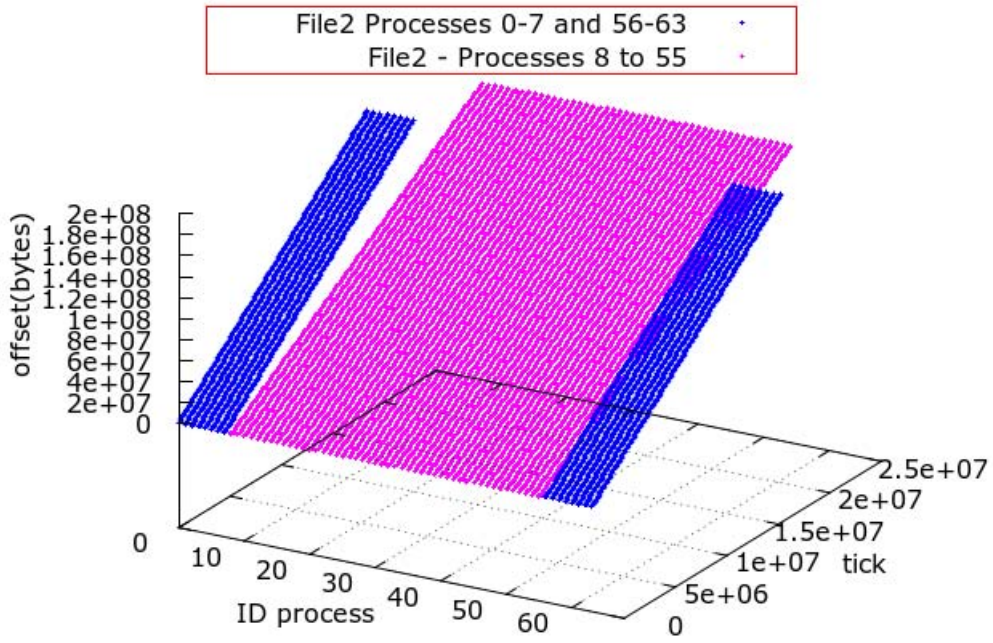


Figure 18: I/O Model of Upwelling for the second file for 64 processes

parameters are set as follows:

- File 1: -np 16 -a MPIIO -c -s 133 -b 35904 -t 4488; -np 48 -a MPIIO -c -s 133 -b 32640 -t 4080
- File 2: -np 16 -a MPIIO -c -s 133 -b 33792 -t 4224; -np 48 -a MPIIO -c -s 133 -b 32640 -t 4080
- File 3: -np 16 -a MPIIO -c -s 133 -b 261888 -t 4224, -np 48 -a MPIIO -c -s 133 -b 238080 -t 3840
- File 4: -np 16 -a MPIIO -c -s 16 -b 135168 -t 8448; -np 48 -a MPIIO -c -s 16 -b 122880 -t 7680

It sets request size with higher request of phase 12 and block size (-b) is set to number of I/O patterns of each iteration multiplied by request size, e.g. for the File 1 $-b32640$ is equal to $4080 * 8$. Segment count is put to number of iteration of the phase 12. Due to that the four file are accessed in similar ticks we run IOR with at the same time for the four file. It selects higher I/O time as I/O time for the application.

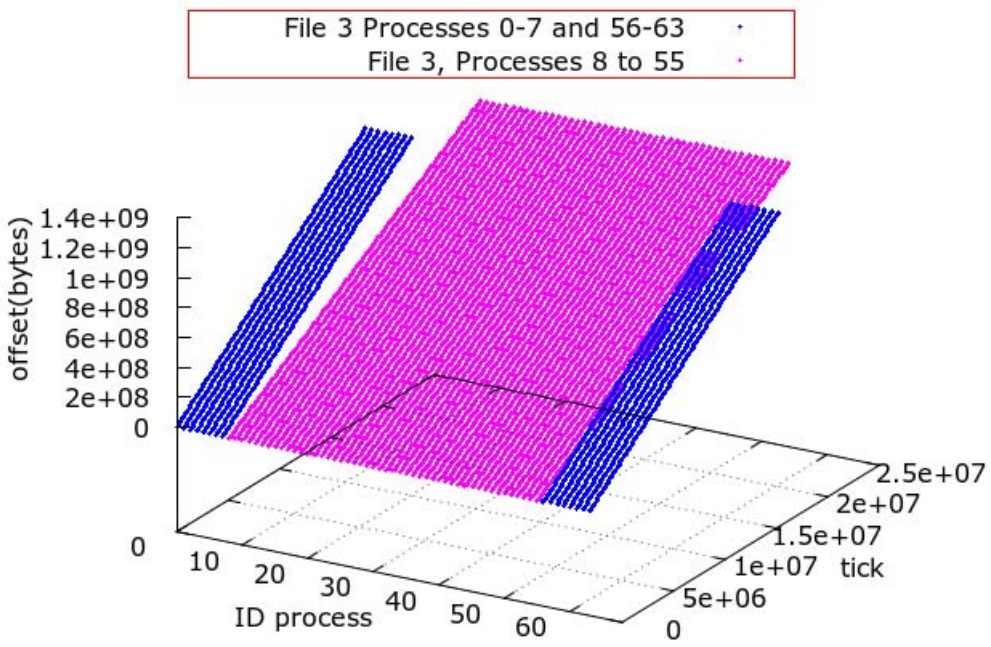


Figure 19: I/O Model of Upwelling for the third file for 64 processes

We have evaluated the I/O time with IOR for the Upwelling in the cluster Finisterrae. Table 21 show the I/O time total obtained with IOR $Time_{io(CH)}$, I/O time for Upwelling in Finisterrae $Time_{io(MD)}$, and relative error $error_{rel}$.

We can observe that size of the four files of Upwelling are lower than 1GB and the request sizes are small: 200 bytes to 4480 bytes to the first file, 240 bytes to 4224 bytes to the second file, 240 bytes to 4224 bytes to the third file, and 200 bytes to 8448 bytes to the fourth file.

In this case, an important point to reduce the I/O impact in I/O time for Upwelling is set where the 4 files will be stored. Due to file sizes and request sizes of highest phase is more convenient to store the four files in the same OSS (a file by each OST of the same OSS).

To can do this we need to know the I/O configuration of Lustre. Table 22 shows the structure of the filesystem Lustre in Finisterrae. Upwelling are tested in four scenarios:

- The four Files by default configuration: Where each file is stored in different OST. With this configuration the I/O time represent the 24 % of the total time.
- The four Files in the same OSS and different OSTs: where the I/O time represent the 26 %.

Table 16: Phases of I/O model of Upwelling - Write_at_all - First File - Processes 0-7 and 56-63

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
11	16	write	4224	2	3	225664	135KB
12				133	137776		55.8MB
12.1	16	write	264		3	14104	4KB
12.2	16	write	264		3	13776	4KB
12.3	16	write	240		3	13932	3.8KB
12.4	16	write	4224		3	220416	67KB
12.5	16	write	3840		3	222912	61KB
12.6	16	write	4488		9	239768	71KB
12.7	16	write	4488		3	239768	71KB
12.8	16	write	4224		3	225664	67KB
12.9	16	write	4224		3	225664	67KB

- The four Files in the same OSTs: where the I/O time represent the 20 %.
- The four Files in the 4 OSS: where the I/O time represent the 21 %.

We can observe that the worst configuration is when each files is stored in an OST of the same OSS. Probably this situation overload the OSS causing the decrease in the performance, therefore, increasing the I/O time. We can observe that the better configuration is when the four files are stored in the same OST. Probably it is the best result due to the sizes of files and the number of process parallel.

5. Related Work

This section present related work with the performance evaluation of the I/O system and the I/O characterization of the parallel scientific applications.

Although following articles are focused on the supercomputer s I/O system, the factors of I/O system configuration that have an impact on the performance are applicable both to small and medium computer clusters. There are various papers that present different I/O configurations for parallel computers and how these configurations are used to performance evaluate of the I/O systems.

Laros presented in [8] the I/O performance analysis developed in the Sandia National Laboratories over the Red Storm platform. To evaluate the I/O system performance the authors evaluate theoretical and empirically the performance. To do this, in first place, they estimate peak theorical for the single data path and for the aggregate controller. Furthermore, they define a methodology to evaluate the performance that consider the

Table 17: Phases of I/O model of Upwelling Write_at_all - First File - Processes 8-55

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
11	48	write	3840	2	3	225664	369KB
12	...			133	183608	...	154MB
12.1	48	write	240		3	14104	11.5KB
12.2	48	write	240		3	13776	11.5KB
12.3	48	write	240		3	13932	11.5KB
12.4	48	write	3840		3	220416	184KB
12.5	48	write	3840		3	222912	184KB
12.6	48	write	4080		11	239768	196KB
12.7	48	write	4080		3	239768	196KB
12.8	48	write	3840		3	225664	184KB
12.9	48	write	3840		3	225664	184KB

single data path, test for file-per-process, and shared-file. The test are done with IOR benchmark. Their results show that the file system and I/O library reduce the performance that provide the I/O devices (Theorical estimation). Furthermore, the authors mentioned that this type of test require an important time to do it.

Yu in [9] presented an in-depth evaluation of parallel I/O software stack of the Cray XT4 platform at Oak Ridge National Laboratory (ORNL). The Cray XT4 I/O subsystem was provided through 3 servers Lustre filesystems. The evaluation covers the performance of a variety of parallel I/O interfaces, including POSIX IO, MPI-IO, and HDF5. Furthermore, a user-level perspective is presented in [10] to empirically reveal the implications of storage organization of parallel programs running on Jaguar at the ORNL. The authors described the hierarchical configuration of the Jaguar Supercomputer Storage System. They evaluated the performance of individual storage components. In addition, they examined the scalability of metadata, and data-intensive benchmarks on Jaguar, and they showed that the file distribution pattern can have an impact on the aggregated I/O bandwidth.

Lang in [11] a case study of the I/O challenges to performance and scalability on the IBM Blue Gene/P system at the Argonne Leadership Computing Facility was presented. The authors evaluated both software and hardware of I/O system and a study of PVFS and GPFS at filesystem level is presented. They evaluate the I/O system for the NAS BT-IO, MadBench2, and Flash3 I/O benhcmarks.

Carns in [12] presented a multilevel application I/O study and a methodology for system-wide, continuous, scalable I/O characterization that combines storage device instrumentation, static filesystem analysis, and a new mechanism for capturing detailed application-level behavior.

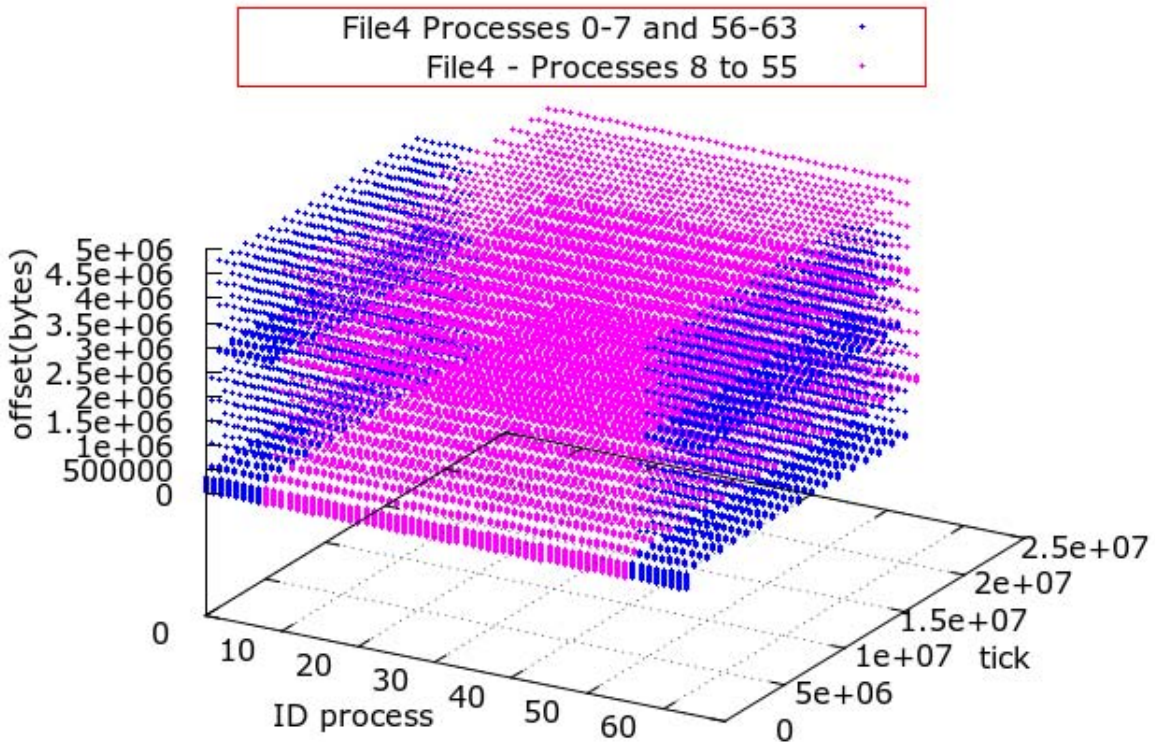


Figure 20: I/O model of Upwelling for the fourth file for 64 processes

From these works the following I/O factors are identified how high impact in the I/O perfomance: filesystem, number of I/O nodes, number of clients, I/O access patterns and the I/O library. The performance evaluation of the I/O system is a long process and usually is done for benchmark that represent a specific workload for the full I/O system.

However, the real applications have different patterns and with different I/O requirements. The application performance depends on its I/O access patterns and the I/O system configuration. Therefore, it is important to extract the I/O access patterns. There are several tracing tools to I/O operations of parallel applications, although most of them are not available for external users. These tools can help to identify application behavior.

Carns in [13] presented the Darshan tracing tool for the I/O workloads characterization of the petascale. Darshan is designed to capture an accurate picture of the application I/O behavior, including properties such as patterns of access within files, with minimum overhead.

Kim et. al. [14] presents a tracing tool for I/O software stack which has been applied to PVFS2 and MPI-IO. They provide a tracing tool which extracts I/O metrics through

Table 18: Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 0-7

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	551015		11MB
12.1	8	write	528		3	28208	4KB
12.2	8	write	528		3	27552	4KB
12.3	8	write	480		3	27864	3.8KB
12.4	8	write	8448		3	440832	67KB
12.5	8	write	7680		3	445824	61KB
12.6	8	write	8448		3	451328	67KB
12.7	8	write	8448		3	451328	67KB
12.8	8	write	8448		3	-4197200	67KB
12.9	8	write	528		551015	28208	4KB
12.10	8	write	528		3	27552	4KB
12.11	8	write	480		3	27864	3.8KB
12.12	8	write	8448		3	440832	67KB
12.13	8	write	7680		3	445824	61KB
12.14	8	write	8448		3	451328	67KB
12.15	8	write	8448		3	451328	67KB
12.16	8	write	8448		3	451328	67KB

the automatic instrumentation of source code of applications. The user must select the appropriate metric and determine that portion of code to trace.

Nakka et. al. [15] presented a trace tool to extract MPI-IO operations from very large applications running at full scale in production systems. This trace tool is specific to their system.

Byna et. al. [16] used I/O signatures for parallel I/O prefetching. They presented a classification of I/O patterns for parallel application, I/O signatures at local process level and applying of signature to prefetching technical. We use their propose to identify access patterns. However, we have identified the global access pattern because we need the I/O for the parallel application. From local access patterns and by similarity, we have defined the global access pattern, then global access patterns are divided in the I/O phases.

H. Shan and J. Shalf in [17] present a proposal to characterize three applications and they utilized IOR to mimic the I/O pattern of these parallel scientific applications. They used this mimic to predict the performance of the applications in different I/O systems.

Darshan has been used in beginning of this research. However, it has decided to change to PAS2P [18] tracing tool due to that was more appropriate to identify the I/O phases of parallel application. The traces library of the PAS2P tool has been extended to trace MPI-IO routines of MPI-2 standard [19], through an instrumentation automatic that

Table 19: Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 8-55

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	734350		10MB
12.1	48	write	480		3	28208	3.8KB
12.2	48	write	480		3	27552	3.8KB
12.3	48	write	480		3	27552	3.8KB
12.4	48	write	7680		3	440832	61KB
12.5	48	write	7680		3	440832	61KB
12.6	48	write	7680		3	440832	61KB
12.7	48	write	7680		3	440832	61KB
12.8	48	write	7680		3	-4197200	61KB
12.9	48	write	480		734350	28208	3.8KB
12.10	48	write	480		3	27552	3.8KB
12.11	48	write	480		3	27552	3.8KB
12.12	48	write	7680		3	451328	61KB
12.13	48	write	7680		3	451328	61KB
12.14	48	write	7680		3	451328	61KB
12.15	48	write	7680		3	451328	61KB
12.16	48	write	7680		3	451328	61KB

interposes to MPI-IO functions.

A similar proposal to [16] is used to identify access patterns. They work at local access pattern level. In the present proposal, we have included the identification of the global access pattern because it is need to understant the parallel application I/O behavior. From local access patterns and by similarity, the global access pattern have been defined, then global access patterns are divided in the I/O phases.

IOR has been configured to reproduce similar patterns to the I/O model phases of the application. It has defined relationships between the I/O phases and the IOR parameters. These definitions are applicable to parallel scientific applications that use MPI-IO, parallel HDF5 and parallel NetCDF. Also, the proposal of this thesis do not require the source code of the application.

In the perfomance evaluation of the I/O system most of these researches are aimed at specific supercomputers, while the strategy proposed in this thesis is focused on computers clusters. Also, these researcher are focused in the analyze the application executing time. However, the main difference is that the proposed methodology is focused in obtain an I/O abstract model of scientific application that can be applied on different I/O systems without that the application is executed in the target I/O system.

Table 20: Phases of I/O model of Upwelling Write_at_all - Fourth File - Processes 56-63

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	551005		11MB
12.1	8	write	528		3	28208	4KB
12.2	8	write	528		3	27552	4KB
12.3	8	write	528		3	27552	4KB
12.4	8	write	8448		3	440832	67KB
12.5	8	write	8448		3	440832	67KB
12.6	8	write	8448		3	440832	67KB
12.7	8	write	8448		3	440832	67KB
12.8	8	write	8448		3	-4197200	67KB
12.9	8	write	528		551005	28208	4KB
12.10	8	write	528		3	27552	4KB
12.11	8	write	528		3	27552	4KB
12.12	8	write	8448		3	451328	67KB
12.13	8	write	8448		3	451328	67KB
12.14	8	write	8448		3	451328	67KB
12.15	8	write	8448		3	451328	67KB
12.16	8	write	8448		3	451328	67KB

Table 21: Error of I/O time estimation on Finisterrae for 64 processes of the Upwelling

Phase	$Time_{io(CH)}$	$Time_{io(MD)}$	$error_{rel}$
Finisterrae	429	418	2 %
Supernova	988	882	12 %

6. Conclusions

A methodology to analyze I/O performance of parallel computers has been proposed and applied. Such methodology encompasses the characterization of the I/O system at different levels: device, I/O system and application. We analyzed and evaluated the configuration of different elements that impact performance by considering the application and the I/O architecture. This methodology was applied in different clusters for the NAS BT-IO benchmark, MadBench2, FLASH-IO and the application Upwelling of the framework Roms.

The characteristics of the I/O systems were evaluated, as well as their impact on the performance of the application. Furthermore, a methodology to obtain an I/O model of the parallel scientific applications has been proposed and tested. The application I/O model is defined by three characteristics: metadata, spatial and temporal global pattern.

Table 22: Structure Lustre on Finisterrae

OSS	**OSTs**	**OSS**	**OSS**
1	0-3	10	36-39
2	4-7	11	40-43
3	8-11	12	44-47
4	12-15	13	48-51
5	16-19	14	52-55
6	20-23	15	56-59
7	24-27	16	60-63
8	28-31	17	64-67
9	32-35	18	68-71

The application is traced to obtain the access patterns and these are analyzed to identify the I/O phases. This instrumentation is performed at MPI-IO level and it does not require the source code.

We have obtained the I/O model of FLASH-IO and Upwelling programs in ROMS. We have evaluated them taking into account the most significant I/O phase. We have used the I/O model to estimate the I/O time.

We only have evaluated the I/O phase that represent 90 % of I/O for the applications. The results show that this approach is more accurate for I/O intensive applications.

The I/O model can be used to evaluate the performance for the application without executing the application. This is very useful particularly for real application that usually need several libraries in order to be executed. This is the case of Upwelling that need the libraries NetCDF parallel, and their HDF5 parallel configured with a parallel filesystem. With the I/O model we can evaluate the behavior only with MPI-IO.

On the other hand, we have configured the number of I/O nodes for FLASH-IO taking into account the number of processes that do I/O operations. The result showed a reduction of about 30 % in the I/O time for the most relevant phase.

On the other hand, we have configured the number of I/O node for FLASH-IO taking into account the number of processes that do I/O. The result show a reduction around of 30 % in the I/O time for the most relevant phase.

Analyzing the I/O model of Upwelling, we observed that the files placement is a factor that can improve the performance. For a Grid of 0041x0080x016, 64 processes and a Tiling of 008x008 the best configuration, in Finisterrae, is an OSTs for the four files. However, the size of files increases when the grid is higher. If the I/O is more significant (phase with higher weight) then it is convenient consider also the number of I/O nodes for each file to increase the transfer rate and reduce the I/O time.

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

7. List of Publications

1. **Sandra Méndez, Dolores Rexachs y Emilio Luque. A Methodology to characterize the parallel I/O of the message-passing scientific applications, In Proceeding of the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 13). In Press. July 2013.**

   This work proposes a methodology to characterize the parallel I/O of scientific applications. The message-passing applications is represented through an I/O model. The model allows us to evaluate the I/O subsystem taking into account the I/O phases of the application.

2. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Evaluating Utilization of the I/O System on Computer Clusters, In Proceeding of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 12), pp. 366-372, Las Vegas, Nevada, USA, July 2012.**

   This paper propose a methodology to evaluate the I/O system utilization taking into account parallel applications behavior and the I/O system performance capacity. To analyze the I/O system we evaluate the I/O path of computer clusters. It uses a hierarchical scheme of the I/O system with its performance capacity at I/O library level and at I/O device level.

   It proposes a method to identify I/O phases of parallel scientific applications depending on its temporal and spatial patterns. It has evaluated the I/O system utilization taking into account the I/O phases behavior of applications in I/O devices.

3. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Modeling Parallel Scientific Applications through their Input/Output Phases, In pro-**

ceeding of the Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS 2012) in CLUSTER 2012, pp. 07-15, Beijing, China, September 2012.

Here, it proposes a methodology to evaluate the I/O system performance capacity through an I/O model of the parallel application independent of the I/O subsystem. This I/O model is composed of I/O phases representing where.[a]nd when the I/O operations are performed into application logic.

This approach encompasses the I/O subsystem evaluation at I/O library level for the application I/O model. The I/O phases are replicated by benchmark IOR which is executed in the target subsystem. This approach was used to estimate the I/O time of an application in different subsystems. This approach was also utilized to select the I/O subsystem that provide less I/O time for the application.

4. **Sandra Méndez, Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. A new approach for Analyzing I/O in parallel scientific applications, In Proceeding of the XVIII Congreso Argentino en Ciencias de la Computación (CACIC 2012), pp. 337-346, Bahía Blanca, Argentina, October 2012.**

   In this paper, it presents the concepts used in the PAS2P methodology, which have been adapted for MPI-IO applications.

5. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Methodology for Performance Evaluation of the Input/Output System, In Proceeding of the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 11), pp. 886-889, Las Vegas, Nevada, USA, July 2011.**

   This paper proposes a methodology for I/O system performance evaluation for computers clusters. The approach encompasses the characterization of the application, I/O system, and devices. We select and evaluate the I/O factors that have an impact on the application performance by considering the application and the I/O architecture configuration. During the analysis of the I/O configuration, we identify configurable factors that impact the I/O system performance. In the evaluation phase, we evaluate the application under different I/O configurations and we determine the inefficiency by analyzing the difference between measured values and characterized values.

6. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Methodology for Performance Evaluation of the Input/Output System on Computer Clusters in Proceeding of the Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS 2011) in CLUSTER 2011, pp. 474-483, Austin, Texas, USA, September 2011.**

   This paper proposes a methodology to evaluate I/O performance on computer clusters under different I/O configurations. This evaluation is useful to study how different I/O subsystem configurations will affect the application performance.

   This approach encompasses the characterization of the I/O system at three different levels: application, I/O system and I/O devices. We select different system configuration and/or I/O operation parameters and we evaluate the impact on performance by considering both the application and the I/O architecture. During I/O configuration analysis we identify configurable factors that have an impact on the performance of the I/O system. In addition, we extract information in order to select the most suitable configuration for the application.

7. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Efficiency Evaluation of the Input/Output System on Computer Clusters, In Proceeding of the Congreso Argentino en Ciencias de la Computación, pp. 261-270, La Plata, Argentina, October 2011.**

   This paper proposes the efficency evaluation of the I/O system on computer clusters. This evaluation is useful to study how different I/O system will affect the application performance. This approach encompasses the characterization of the computer cluster at three different levels: devices, I/O system and application. We select different system and we evaluate the impact on performance by considering both the application and the I/O architecture. During I/O configuration analysis we identify configurable factors that impact the performance of I/O system. Furthermore, we extract information in order to determine the used percentage of I/O system by an application on a given computer cluster.

8. **Javier Panadero, Sandra Méndez, Dolores Rexachs y Emilio Luque. Characterizing energy efficiency in I/O system for scientific applications, In Proceeding of the Fifth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2011), pp. 106-112, Lisboa, Portugal, November 2011.**

   This paper proposes a methodology to characterize the energy efficiency of the

Input/Output system, considering the Input/Output system at a device level, I/O library and file system. The methodology provides a wide range of I/O system parameters that have an impact on the energy efficiency. Furthermore, we evaluate the impact of Input/Output intensive applications in energy efficiency.

Capítulo 1

Introducción

El aumento del número de unidades de procesamiento en los clúster de computadores, los avances tanto en velocidad como en potencia de los procesadores y las crecientes demandas de las aplicaciones científicas que utilizan cómputo de altas prestaciones trae mayores exigencias a los sistemas de Entrada/Salida (E/S) paralelos.

Las mejoras de prestaciones del acceso a los datos (latencia y ancho de banda) ha sido mas lenta que las prestaciones de los procesadores. La velocidad de las unidades de disco se ha incrementado únicamente en un 7 % por ano en las pasadas dos décadas, lo cual es significativamente bajo comparado con alrededor de un 50 % por ano para las prestaciones de los procesadores [20]. Esto trae como consecuencia que en muchos casos, el cuello de botella de los sistemas paralelos sea la E/S dada las exigencias que debe afrontar y las limitaciones propias de los dispositivos.

Todo esto ejerce una gran presión en los sistemas de E/S, especialmente para la computación de altas prestaciones (HPC) donde el rendimiento es clave.

La E/S Paralela es esencial para emparejar el avance de las arquitecturas de los procesadores y el rápido crecimiento de la capacidad computacional. Aunque la arquitectura jerárquica de memoria multinivel puede evitar grandes perdidas de prestaciones debido a los retardos de acceso a disco, la capacidad de memoria es limitada. Además, como la capacidad computacional aumentará, la disponibilidad de memoria por core decrecerá, especialmente si la escala de los sistemas de HPC se proyecta a millones de cores o más. Varias simulaciones científicas y de ingeniería de áreas críticas de investigación, tales como la nanotecnología, astrofísica, clima y energía física están convirtiéndose en aplicaciones intensivas de datos[21]. Estas aplicaciones realizan un gran número de accesos de E/S, donde una gran cantidad de datos son almacenados y recuperados desde los sistemas de almacenamiento. Usualmente este conjunto de datos son mayores que la memoria de la que se dispone y necesitan sistemas de E/S paralelos para cumplir con sus demandas[22].

El diseno de los modernos sistemas de cómputo de altas prestaciones separan los recursos de cómputo de los recursos de almacenamiento, por varias razones, como la utilización del consumo energético, confiabilidad de recursos de cómputo, y facilitar la gestión del almacenamiento. Sin embargo, esta estructura también introduce la necesidad de capas adicionales para gestionar la transferencia de datos entre el sistema de cómputo y el sistema de E/S. Estas capas proveen herramientas para utilizar los recursos disponibles de HPC, sin necesidad de que conozcan el funcionamiento interno del sistema de E/S. Sin embargo, estas capaz adicionales también incrementan la complejidad del sistema, haciendo el análisis de prestaciones del sistema de E/S más difícil[23].

Muchas aplicaciones requieren un gran número de nodos de cómputo y operan con un gran volumen de datos que deben ser transferidos entre los procesos por medio de mensajes. La interfaz de paso de mensaje MPI (Message Passing Interface) es la librería de paso de mensajes más ampliamente usada para proveer comunicación en los computadores paralelos. MPI provee una interfaz estándar para operaciones tales como comunicación punto a punto, comunicación colectiva, sincronización, y operaciones de E/S. Durante las operaciones de E/S, los procesos frecuentemente acceden a un conjunto de datos compartidos para realizar las operaciones de E/S de diversos tamanos. Dependiendo del patrón de acceso de E/S esto se puede convertir en un cuello de botella para el sistema de cómputo de altas prestaciones.

En este capítulo se presenta la motivación de este trabajo, el objetivo general y los objetivos específicos, la contribuciones y finalmente se presenta la organización de este documento.

1.1. Motivación

Para poder analizar y disminuir la brecha entre las prestaciones de CPUs y la E/S se deben identificar los factores que influyen en las prestaciones y proponer nuevas soluciones. Esto lleva a plantear las siguientes preguntas:

- Qué factores de E/S influyen en el rendimiento?
- Son los factores de E/S configurables a nivel de usuario y/o a nivel de administrador?
- Deben los programadores cambiar las aplicaciones para adaptarse a los sistemas de E/S?

- Deben los disenadores de sistemas configurar el sistema de E/S para que se adapte de forma transparente a los características de E/S de las aplicaciones?

Responder a estas preguntas no es trivial. Las aplicaciones tienen un comportamiento de acuerdo a sus propósitos y si bien los programadores o disenadores pueden realizar las modificaciones necesarias para que la aplicación realice de forma eficiente las operaciones de E/S, estas modificaciones están realizadas para un computador paralelo en particular. Por otro lado, sacar el mayor provecho al sistema de E/S requiere que el programador conozca los detalles de los sistemas de E/S, lo que puede resultar difícil.

Los usuarios necesitan información para responder a preguntas como: Es el sistema de E/S un factor limitante para la aplicación? Cuánta capacidad del sistema de E/S está siendo utilizado por la aplicación? A la vez los administradores de sistemas necesitan meta-información de las aplicaciones para poder hacer una mejor gestión del sistema de E/S.

Para aprovechar las prestaciones de los sistemas de almacenamiento sería conveniente que la aplicación hubiera sido programada teniendo en cuenta la estructura del sistema E/S. Las aplicaciones científicas de HPC son programadas para aprovechar el capacidad de cómputo de los computadores paralelos, y existen muchas teorías que ayudan, pero no existe tanta información ni formación para considerar la E/S, muchos programadores intentan evitarla, o no la gestionan de un modo explícito.

Otra opción sería que los sistemas de E/S estén configurados para dar mejores prestaciones a los patrones de accesos de la aplicación, esta propuesta tampoco es aplicable ya que los sistemas de E/S son compartidos por diferentes tipos de aplicaciones con diferentes patrones de acceso.

Una tercera posibilidad sería poder ajustar determinados parámetros del sistema de E/S que tengan impacto en las prestaciones que perciba la aplicación para lo cuál sería necesario identificar estos factores y ver si se pueden configurar a nivel de usuario o administrador. En este caso también es necesario analizar la aplicación y sus patrones de comportamiento respecto a la E/S.

Este contexto es la principal motivación de este trabajo. Es necesario proporcionar, a los usuarios y administradores de sistemas, información y pautas que les permitan analizar tanto la aplicación como el sistema de E/S para poder identificar los cuellos de botella y las ineficiencias y dar alternativas que permitan mejorar las prestaciones de E/S que la aplicación percibe en base a la selección del sistema de E/S que mejor se ajuste a la aplicación.

1.2. Objetivos

Teniendo en cuenta este escenario el presenta trabajo de investigación tiene como objetivo general definir una Metodología para evaluar las prestaciones de los sistemas de Entrada/Salida de computadores de altas prestaciones considerando la configuración de la E/S y las características de la Aplicación .

Para concretar este objetivo se plantean los siguiente objetivos específicos:

- Modelar el comportamiento de E/S de las aplicaciones científicas.
- Definir los pasos y criterios necesarios para evaluar el sistema de E/S teniendo en cuenta las diferentes configuraciones de los actuales clúster de computadores.
- Definir los pasos y criterios necesarios para analizar, seleccionar o configurar el sistema de E/S teniendo en cuenta el comportamiento de las aplicaciones científicas.

Para concretar el objetivo planteado y sus objetivos específicos se ha definido una metodología (Figure 1.1) que consta de tres etapas: Caracterización de los patrones de E/S de la aplicación y de la infraestructura de E/S utilizada, Análisis de la E/S y Evaluación. La metodología es aplicable a aplicaciones científicas paralelas que usan MPI-IO y que tienen un comportamiento determinístico. En cuanto al sistema de E/S este trabajo solo se centra en los sistemas de ficheros más utilizados en clúster de computadores como NFS, LUSTRE y PVFS2 (OrangeFS).

1.2.1. Caracterización

La etapa de caracterización se aplica a la aplicación paralela y al sistema de E/S. Esta tiene dos objetivos:

- Identificación de las configuraciones del sistema de E/S y caracterización del rendimiento de cada configuración por medio de benchmark de E/S.
- Extracción del modelo de comportamiento de E/S de aplicaciones científicas paralelas.

Aplicación Científica Paralela

La metodología incluye una etapa inicial de caracterización que permite obtener el modelo de E/S de la aplicación paralela. El modelo de E/S está definido por tres características: los metadatos, el patrón de acceso espacial y el patrón de acceso temporal.

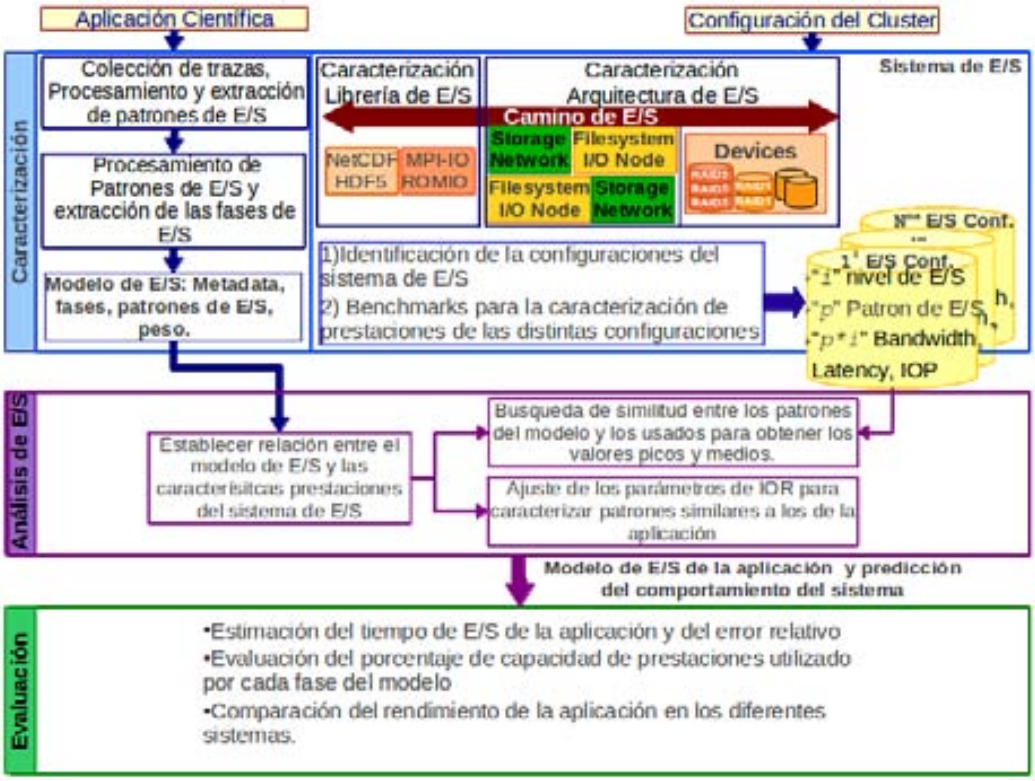


Figura 1.1: Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones

Los patrones de E/S usualmente se repiten en fases de E/S. Una fase de E/S es una secuencia repetitiva de patrones de acceso. Esta fase se puede repetir varias veces en la ejecución.

Una fase será más o menos significativa en función de la cantidad de datos transferidos (su peso). El peso de una fase depende del volumen de información a la que accede las operaciones de E/S durante las fases. El peso de una fase se expresa en Bytes.

Una fase de E/S esta compuesta por:

- Identificador de Fase (IdF)
- Identificadores de los Procesos que forman parte de la fase ($IdsP$)
- Número de procesos que forman la fase (np)
- request size (rs), tipo de operación (w/r), distancia ($dist$), desplamiento ($disp$) y $tick$
- Número de repeticiones (rep), número de operaciones de E/S ($\#iop$) y peso ($weight$).

La Figura 1.2 muestra los elementos que componen una fase y el patrón temporal y espacial que la componen.

Figura 1.2: Componentes de Fases de E/S

Donde el peso es expresado como:

$$Peso(fase[i]) = np * rs * rep \tag{1.1}$$

Donde np es el número de procesos de la $fase[idF]$, rs el tamano de la operación y rep el número de repeticiones del patrón.

Se usa el concepto de patrón local y global para la aplicación científica paralela y dentro de estos patrones el concepto de patrón temporal y espacial. El patrón espacial local muestra como un proceso de la aplicación accede a un fichero. El patrón temporal local muestra el orden en el que se hacen los accesos a un archivo. Los patrones globales espaciales y temporales muestran como los np procesos de la aplicación paralela acceden a un archivo compartido así como el orden en el que lo hacen. La Figura 1.3 un ejemplo de modelo de E/S. Donde podemos observar el patrón local y global tanto temporal como espacial. Además de esta información tenemos en cuenta la meta-iformación para el modelo de E/S. La meta información permite identificar el modo de acceso (Strided, Secuencial, Random), tipo de acceso (Unico, compartido).

Sistema de E/S

Además en la etapa de caracterización se analiza el hardware y el software del sistema de E/S para identificar las diferentes configuraciones de E/S que existen en el clúster objeto de análisis. Para este proceso el sistema de E/S es estructurado en un esquema jerárquico para dar un orden al proceso de caracterización. En la figura 1.4 se presenta el esquema del sistema de E/S. Éste es analizado desde el punto de vista de la arquitectura de E/S y del camino de E/S de los datos sobre un clúster. Se ejecutan los benchmarks para la obtención de las métricas de prestaciones se aplica al nivel de dispositivos de E/S y de librería de E/S en cada configuración para diferentes patrones de acceso.

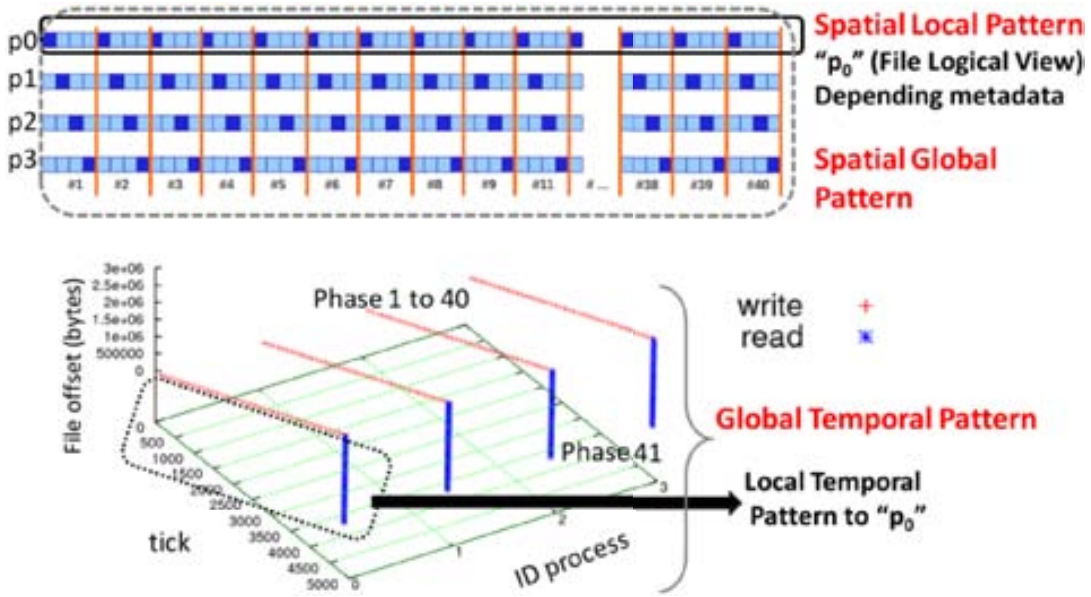


Figura 1.3: Ejemplo de modelo de E/S para NAS BT-IO, 4 procesos, Clase A subtipo FULL

1.2.2. Análisis de E/S

En la etapa de Análisis de E/S se utiliza la información del modelo de comportamiento de la aplicación y las métricas obtenidas con los benchmarks en las diferentes configuraciones de E/S. En el caso de no contar con una caracterización para un patrón similar a las fases del modelo de E/S se sintoniza el benchmark IOR de la siguiente manera:

- Modo de acceso Strided: $s = Iter$; $b = RS_{(IdPh)}$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$; $-F$ si el tipo de acceso es Único y sin $-F$ para un tipo de acceso Compartido. $-c$ si las operaciones de E/S son colectivas.

- Modo de acceso Secuencial: $s = 1$; $b = weight(IdPh)$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$; $-F$ si el tipo de acceso es Único y sin $-F$ para un tipo de acceso Compartido. $-c$ si las operaciones de E/S son colectivas.

Se ejecuta el benchmark IOR se obtiene un ancho de banda caracterizado $BW_{(CH)}$ y a partir de ésta obtenemos el tiempo de I/O caracterizado que denominamos $Time_{io(CH)}$.

Para determinar como se comporta la aplicación en dicho sistema se establece una relación entre el patrón de acceso de cada fase de E/S de la aplicación con los patrones de acceso de los benchmark usados para la obtención de las métricas. Por ejemplo para determinar que tasa de transferencia obtendría una fase de E/S se busca el patrón en la base de datos de valores medidos por los benchmark y se extrae la tasa de transferencia caracterizada. En la etapa de evaluación se opta por seleccionar como métrica de selección

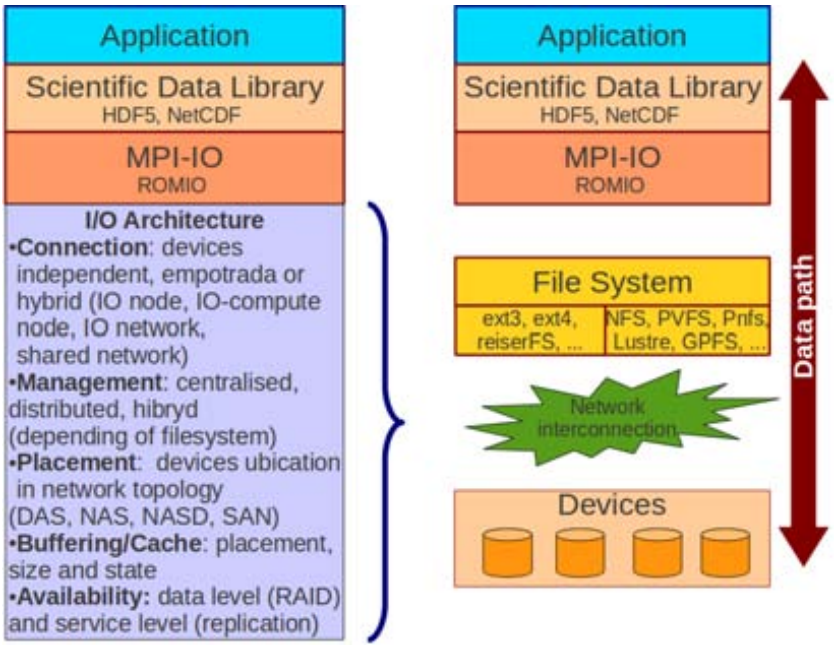


Figura 1.4: Sistema de E/S desde el punto de vista de la arquitectura y del camino de I/O

el tiempo de E/S. De esta forma aquella configuración que proporcione menor tiempo de E/S será la seleccionada para la aplicación.

El tiempo de E/S se expresa como $Tiempo_{E/S} = \sum_{i=1}^{n} Tiempo_{E/S}(fase[i])$

Donde el $Tiempo_{E/S}(fase[i]) = \frac{Peso(fase[i])}{TasaTransferencia(Patron_{CH})}$

1.2.3. Evaluación

Una vez que se tiene la información estimada de cómo se comportan cada una de las fases de la aplicación en el sistema, se pasa a la fase de evaluación. Esta etapa se incluye para evaluar el uso que se hace del sistema de E/S y el tiempo de E/S. Utilizando la información de las prestaciones de cada configuración, se evalúa cuanto de esa capacidad usa la aplicación. Para esto se establece una relación entre la tasa de transferencia pico y la tasa de transferencia alcanzada por cada fase de la aplicación.

$$UsoSistema(fase[i]) = \frac{TasaTransferencia_{MD}(fase[i])}{TasaTranserencia_{CH}(fase[i])} * 100 \tag{1.2}$$

Además también evaluamos el error cometido en la estimación del tiempo de E/S. Para ello usamos:

$$error_{abs} = BW_{CH}(fase[i]) - BW_{MD}(phase[i]) \tag{1.3}$$

$$error_{rel} = 100 * (\frac{error_{abs}}{BW_{MD}(fase[i])}) \quad (1.4)$$

Cabe aclarar que no usamos el tiempo de E/S para hacer una predicción del tiempo total de E/S de la aplicación ya que no se están considerando los tiempos de apertura y cierre de los ficheros, en este caso se utiliza el tiempo para ver el comportamiento de la aplicación en diferentes sistemas de E/S.

1.3. Contribución

A partir de la metodología propuesta las principales contribuciones de este trabajo se resumen a continuación:

- Una metodología para evaluar las prestaciones de los sistema de E/S en clúster de computadores [24]. Esta metodología da un orden a la evaluación de prestaciones en clúster de computadores. Indicando que parámetros se deben considerar al momento de determinar las valores picos y medios de prestaciones.

- Una metodología para la caracterización de los requisitos de E/S de las aplicaciones científicas paralelas [25], donde se presenta la definición de un modelo de E/S que se centra en las fases de E/S de la aplicación y en el orden de ocurrencia de eventos de E/S y de comunicación. Las fases de E/S están representadas por los patrones de E/S y el peso de la fase que indica la cantidad de datos que la aplicación moverá en el sistema de E/S.

- Una herramienta para la extracción de fases de E/S. Ésta realiza el trazado a nivel de MPI de las operaciones de E/S identificando los parámetros necesarios para la elaboración de un modelo de E/S que sea independiente de los tiempos de E/S del sistema donde se extraen las trazas [26].

- Una metodología que da información estructurada del sistema de E/S que permite comparar sistemas, analizar cuellos de botella y provee información para seleccionar y configurar un sistema de E/S teniendo en cuenta el modelo de E/S de las aplicaciones. A partir del modelos de E/S de la aplicación se obtienen características de E/S, estas se analizan para determinar que recursos de E/S serían los más adecuados para estas características para lograr aumentar la tasa de transferencia y reducir el tiempo de E/S [25][26].

1.4. Estructura de la Tesis

De acuerdo al objetivo presentado y para un mejor entendimiento de este trabajo, este documento se ha estructurado de la siguiente manera:

- Capítulo 2: Estado del Arte

  Éste capítulo presenta los trabajos relacionados y los conceptos de E/S. Se presenta en primer lugar una explicación del concepto de patrones de E/S para aplicaciones científicas paralelas, los conceptos del sistema de E/S, y finalmente los trabajos relacionados para la caracterización de las aplicaciones y para la evaluación de prestaciones de los sistemas de E/S..

- Capitulo 3: Metodología para la caracterización de la E/S de aplicaciones científicas paralelas

  En este capítulo se presenta el proceso para la extracción de los requisitos de E/S de las aplicaciones científicas paralelas expresados en un modelo de E/S que se centra en las fases de E/S de la aplicación. Se presenta una parte experimental para mostrar los conceptos aplicados a los benchmark de E/S. Este modelo es aplicado a diferentes sistemas y muestra la independencia del modelo de la arquitectura del sistema de E/S.

- Capítulo 4: Metodología para la caracterización del sistema de Entrada/Salida en computadores de altas prestaciones

  En este capítulo se explica el proceso para caracterizar el sistema de E/S. Esta caracterización se realiza para identificar las diferentes configuraciones del sistema de E/S y evaluar sus características prestacionales. Se presenta una parte experimental sobre diferentes configuraciones de E/S para mostrar la aplicación de la metodología.

- Capítulo 5: Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones

  En este capítulo se explica el proceso para seleccionar y configurar el sistema de E/S en función del modelo de E/S planteado en el capítulo 4. Se presenta una parte experimental sobre dos clúster con diferentes configuraciones de E/S para mostrar la aplicación de la metodología.

- Capítulo 6: Evaluación Experimental

En este capítulo se presenta la evaluación experimental aplicada a dos clúster de computadores en producción donde se muestra el uso de la metodología para la selección y configuración del sistema de E/S con kernel de aplicaciones y aplicaciones reales paralelas.

- Capítulo 7: Conclusiones y Trabajos Futuros

  Finalmente en este capítulo se presentan las conclusiones en base a los experimentos realizados y las líneas abiertas.

Capítulo 2

Estado del Arte

Para mejorar la eficiencia de la E/S de las aplicaciones científicas paralelas es necesario conocer el sistema de E/S donde se va evaluar la aplicación y para esto se necesita conocer la estructura del sistema de E/S del clúster de computadores. Además se deben conocer los patrones de acceso de las aplicaciones científicas paralelas y ver cómo estos serán gestionados por el sistema de E/S.

El sistema de E/S Paralelo comprende dos niveles: nivel de Librería de E/S Paralela y nivel de Arquitectura de E/S. Pero considerar al Sistema de E/S Paralelo sin tener en cuenta la aplicación, no es apropiado debido a que las prestaciones de la E/S paralela para una configuración de E/S varía de acuerdo al patrón de E/S de la aplicación, por esta razón se incorpora un nivel de aplicación. Debido a que no existe un consenso sobre los componentes de un sistema de E/S paralelo, es necesario establecer una estructura del sistema de E/S que será la base para establecer la metodología para la evaluar las prestaciones. En la figura 2.1 se muestra la estructura del sistema de E/S Paralelo para este trabajo y sus componentes.

Este capítulo se estructura de la siguiente manera: en la Sección 2.1 se presentan los conceptos usados en el capítulo 3 de caracterización de la aplicación, la Sección 2.2 se presenta los conceptos para entender la caracterización del sistema de E/S, 2.2.1 presenta los conceptos que se deben tener en cuenta para disenar el sistema de E/S desde el punto de vista de la arquitectura, 2.2.2 se presenta la estructura de la pila de software del sistema de E/S, y finalmente en sección 2.4 se presentan los trabajos relacionados con el trabajo presentado en esta memoria.

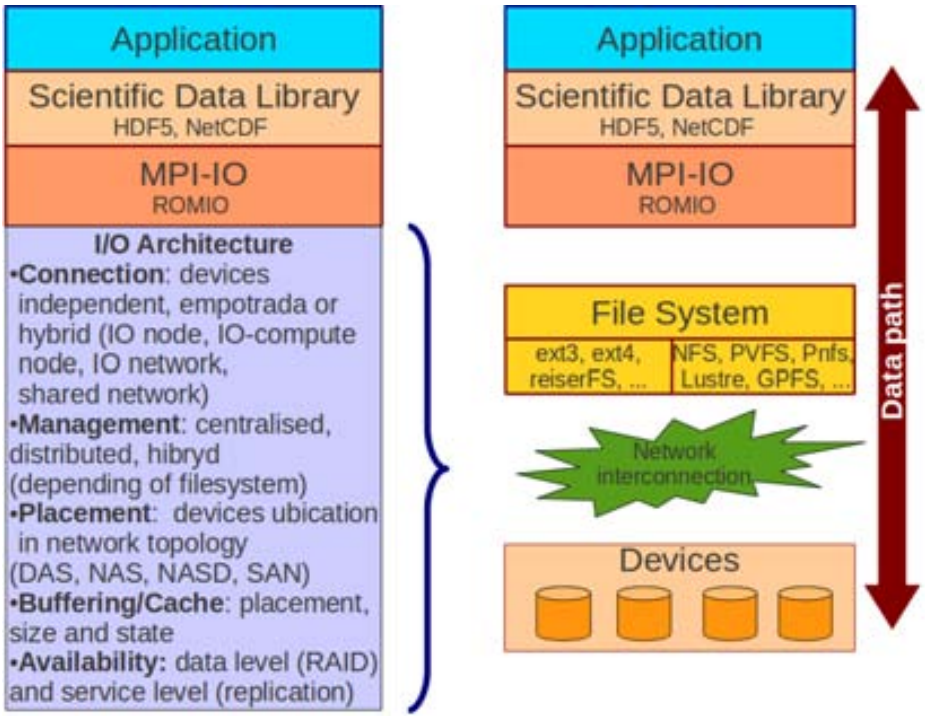


Figura 2.1: Sistema de E/S Paralelo

2.1. Caracterización de la E/S de aplicaciones científicas paralelas

Las prestaciones que una aplicación consigue del sistema de E/S depende de sus patrones de acceso y de la configuración del sistema de E/S. Para poder extraer los patrones de E/S de las aplicaciones paralelas se pueden usar las herramientas como mpe, tau, etc. Sin embargo, estas herramientas no ofrecen la información requerida para el análisis de los patrones de E/S que el presente trabajo propone. La herramienta DARSHAN [13], desarrollada por Philip Carns en el Laboratorio Nacional de Argonne de Estados Unidos, es una de las primera orientadas a la caracterización de I/O de aplicaciones científicas paralelas y de libre distribución. Esta fue usada al inicia de esta investigación para la caracterización de I/O pero debido a que no proporcionaba la información por evento de E/S ni el orden de ocurrencia se optó por una extensión de una herramienta de trazado de aplicaciones MPI desarrollada en nuestro grupo de investigación libpas2p [18]. Libpas2p ha sido modificada para extraer las operaciones de E/S del estándar MPI-2 [19]. Esta herramienta no necesita la instrumentación del código fuente de la aplicación ya que utiliza el concepto de interposición de funciones. Ésta librería es cargada por LD_PRELOAD y sólo requiere que la librería de paso de mensajes esté configurada como librería dinámica.

Muchas aplicaciones científicas leen y escriben datos en fases bien definidas: el programa se carga, hace cómputo, escribe datos. Usualmente, un programa continuará procesando en varios pasos, escribiendo datos cada vez que pasan un número específico de pasos. En una aplicación paralela, las tareas individuales en un trabajo escribirán sus datos al mismo tiempo. Estos patrones de E/S pueden ser predecibles o aleatorios y generalmente se centran más en operaciones de escritura.

Las aplicaciones científicas cuya tarea central es el cómputo, se asume que los datos están en la memoria por lo que muchas veces no son programadas pensando en la E/S. El movimiento de los datos entre la memoria y el almacenamiento pasa a ser un problema secundario durante el diseno de la aplicación.

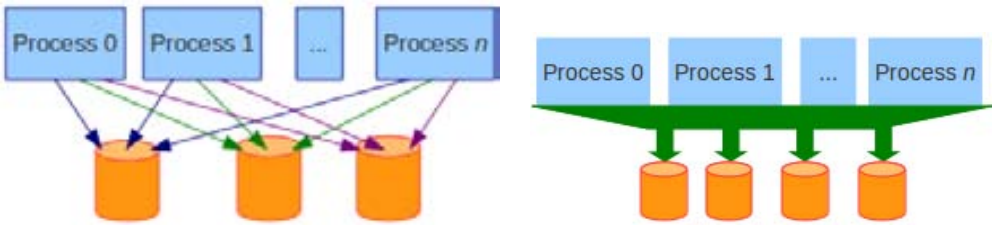
Sin embargo cada vez hay más aplicaciones que requieren acceder a un gran volumen de datos, como por ejemplo: las aplicaciones científicas de áreas como estudios del clima, fusión y dinámica molecular usan E/S para varios propósitos, tales como la obtención inicial de condiciones y parámetros de ejecución, como una forma persistente de almacenamiento de las salidas del programa, y como resguardo contra fallos del sistema. Este último propósito está teniendo un crecimiento importante: el deseo de alcanzar una capacidad computacional marcada en los sistemas de cómputo de altas prestaciones ha producido una tendencia a sistemas con un número incremental de componentes, y la confiabilidad de estos sistemas disminuye a medida que el número de componentes aumenta.

Teniendo en cuenta este contexto los esfuerzos por mejorar las prestaciones de la E/S deben empezar por entender los patrones de E/S. Para esto en primer lugar se debe conocer las características de los patrones de E/S de las aplicaciones, definir un método para extraerlos y representarlo de una forma que el usuario pueda analizar y el administrador pueda utilizar en la gestión de los sistemas de E/S.

A continuación se presenta una clasificación de patrones de E/S de aplicaciones científicas paralelas y los conceptos de MPI para la manipulación de archivos.

2.1.1. Patrones de E/S en Aplicaciones Científicas

Muchos archivos son pequenos y temporales, que son creados y borrados en un corto período de tiempo. Además en muchos archivos son más frecuentes las lecturas que las escrituras. Muchas lecturas son secuenciales y recorren todo el archivo. Para estos archivos el caching y buffering trabajan bien. Desafortunadamente, las aplicaciones científicas de HPC no son programas UNIX típicos, y los patrones de E/S de computadores de memoria compartida difieren de los de memoria distribuida [27]. Este comportamiento de la E/S en las aplicaciones de HPC trae como consecuencia que sea complicado realizar una taxonomía de los patrones de acceso. En este trabajo se ha usado parte de la taxonomía de Byna [16]

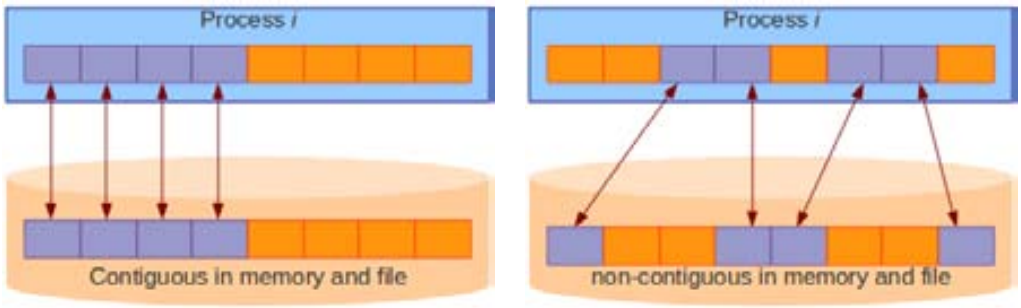


(a) Entrada/Salida independientes (b) Entrada/Salida colectivas

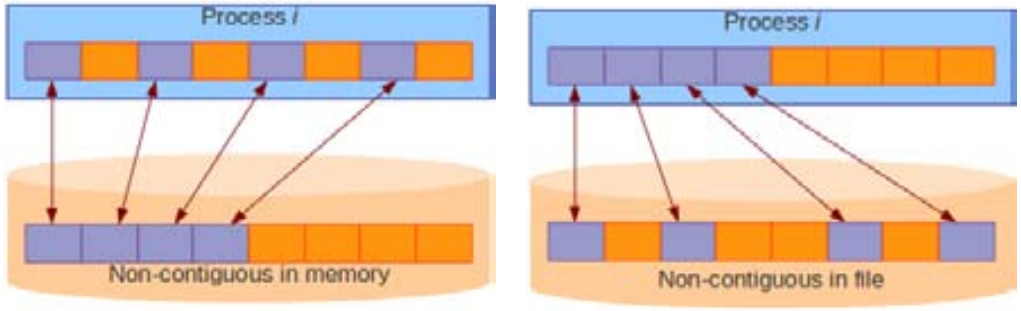
Figura 2.2: Patrones globales

para los patrones de E/S locales y para los patrones globales se ha considerado además de los patrones locales, la temporalidad y meta-información de las operaciones de E/S. Los patrones de aplicaciones paralelas pueden ser divididas en patrones locales y patrones globales. Los patrones locales están determinados por el proceso (o thread), mostrando como un archivo es accedido por un proceso local. Los patrones globales (figura 2.2) son sobre la aplicación paralela, representando como los múltiples procesos acceden a un archivo. En [16] se hace una clasificación en cinco dimensiones para un proceso local. Las cinco dimensiones son espacial, tamano de solicitud, comportamiento repetitivo, temporal y tipo de operación de E/S. A continuación se explica las 5 dimensiones:

- Operación de E/S: las operaciones de E/S pueden ser de *write only*, *read only* y *read/write*.
- Patrón espacial: la secuencia en la que se accede a posiciones de un archivo se denomina patrón espacial de una aplicación. Ellos pueden ser contiguos (figura 2.3(a)) o discontiguos (Fig. 2.3(b)) o una combinación de ambos (Fig. 2.3(c), 2.3(d)). Los accesos discontiguos hacen referencia a huecos en los accesos a un archivo. Estos huecos pueden ser de tamano fijo o variable. Los huecos variables pueden seguir un patrón de dos o más dimensiones. Otro patrón puede ser con *strides* decrecientes. Algunos accesos no tienen un patrón regular, los *strides* son aleatorios.
- Tamano de la Solicitud: este puede ser pequeno (*small*), mediano (*medium*) y grande (*large*). El tamano de una solicitud puede ser fija o variable. Una solicitud se considera pequena cuando es solo una fracción de una página. Y será grande cuando el tamano sea varias veces mayor al tamano de una página. Dada la alta latencia de la E/S, las solicitudes pequenas pueden causar cuellos de botella en las prestaciones si se accede a los discos por una pequena cantidad de bytes.
- Comportamiento Repetitivo: Una aplicación tiene un comportamiento repetitivo



(a) Accesos contiguos (b) Accesos discontinuos



(c) Accesos discontinuos en memoria (d) Accesos discontinuos en archivo

Figura 2.3: Patrón espacial de una aplicación

cuando los bucles o una función con bucles tiene operaciones de E/S. Cuando un patrón de accesos es repetitivo, el *caching* y el *prefetching* pueden efectivamente enmascarar la latencia de los accesos.

- Intervalo Temporal: los patrones temporales capturan las regularidad de las ráfagas de E/S en una aplicación. Estos pueden ocurrir periódicamente como irregularmente.

2.1.2. Conceptos de MPI para la manipulación de ficheros

Para entender el proceso de detección de patrones de acceso para aplicaciones MPI se presentan los conceptos usados para la manipulación de ficheros. MPI-IO en su estándar MPI-2, define conceptos para la manipulación de ficheros. A continuación se presenta una breve descripción de estos:

- *file*: Un archivo MPI es un colección ordenada de ítems de un tipo de dato. MPI soporta accesos random o accesos secuenciales. Un fichero es abierto colectivamente por un grupo de procesos y todas las llamadas colectivas en un fichero son colectivas en ese grupo.

- *file size and end of file*: El tamano de un archivo de MPI se mide en bytes desde el comienzo del archivo. Un archivo recién creado tiene un tamano de cero bytes.

- *file pointer*: Un puntero de archivo es un *offset* implícito mantenido por MPI. Los *Individual file pointers* son apuntadores de archivo que son locales a cada proceso que abrió el archivo. Un *shared file pointer* es un puntero a un archivo que es compartido por el conjunto de procesos que abrió el archivo.

- *file handle*: Un identificador de archivo es un objeto opaco creado por MPI_FILE_-OPEN y liberado por MPI_FILE_CLOSE. Todas las operaciones en un fichero abierto se hacen a través de un *handle* de archivo.

- *displacement*: es un desplazamiento en una posición de bytes absoluta relativa al inicio del fichero. El desplazamiento define dónde inicia una vista (view). El *file displacement* es diferente al *typemap displacement*.

- *etype*:Es un tipo de dato elemental (*elementary datatype*), es la unidad de acceso y posicionamiento. Puede ser cualquier tipo predefinido o tipo de dato derivado. El acceso a los datos es realizado en unidades de *etype*, leyendo y escribiendo ítems de datos del tipo *etype*. Los *offsets* son expresados en unidades de *etype*, los punteros de ficheros apuntan al inicio de esta unidad. Dependiendo del contexto, el término *etype* se utiliza para describir uno de los tres aspectos de un tipo de dato elemental: un tipo particular MPI, un elemento de datos de ese tipo, o una medida de ese tipo.

- *filetype*: Es la base para la partición de un archivo entre procesos y define una plantilla para acceder al archivo. Es un único etype o un tipo de datos MPI derivado construido a partir de varias instancias del mismo *etype*. Además, la extensión de cualquier *hole*

  en el *filetype* debe ser un múltiplo de la medida de etype. Los desplazamientos en el *typemap* del tipo de archivo no están obligados a ser diferentes, pero tienen que ser no negativos y monolítico no decreciente.

- *view*: Una vista define el conjunto actual de datos visibles y accesibles a partir de un archivo abierto como un conjunto ordenado de *etypes*. Cada proceso tiene su propio punto de vista del fichero, definida por tres cantidades: un desplazamiento, un *etype*, y un *filetype*. El patrón descrito por un *filetype* se repite, comenzando en el desplazamiento, para definir la vista. Las vistas pueden ser modificados por el usuario durante la ejecución del programa. La vista predeterminada es

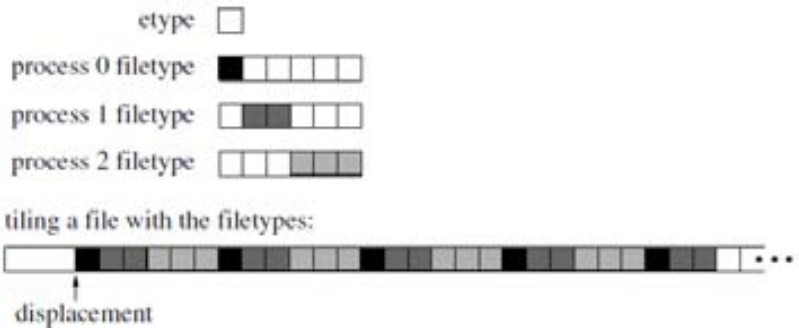


Figura 2.4: Particionado de un fichero para varios procesos

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking & split collective	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking & split collective	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking & split collective	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

Figura 2.5: Operaciones MPI para acceso a los datos

una secuencia de bytes lineal (el desplazamiento es cero, y el *etype* y *filetype* son iguales a MPI_BYTE).

- *offset*: Es una posición en el archivo relativo a la vista actual, expresada en número de etypes. Los *holes* en el *filetype* de la vista se omiten al calcular esta posición. Un valor 0 es la ubicación del primer etype visible en la vista (después de saltarse el desplazamiento y los *holes* iniciales en la vista). Por ejemplo, un *offset* de 2 para el proceso 1 en la Figura 2.4 es la posición del octavo etype en el archivo después del desplazamiento. Un *explicit offset* es un desplazamiento que se utiliza como un parámetro formal en las rutinas de acceso a datos explícitos.

El estándar MPI-2 muestra las operaciones de I/O en tres grupos: manipulación de ficheros (Tabla 2.1), vista de ficheros (Tabla 2.2) y accesos de datos (Figura 2.5) para operaciones colectivas y no colectivas.

Tabla 2.1: MPI-IO - File Manipulation

Operation Type	**Parameters**
MPI_File_open	(MPI_Comm comm, char * filename,int amode, MPI_Info info, MPI_File * fh)
MPI_File_close	(MPI_File * fh)
MPI_File_delete	(char * filename, MPI_Info info)
MPI_File_set_size	(MPI_File fh, MPI_Offset size)
MPI_File_preallocate	(MPI_File fh, MPI_Offset size)
MPI_File_get_size	(MPI_File fh, MPI_Offset * size)
MPI_File_get_group	(MPI_File fh, MPI_Group * group)
MPI_File_get_amode	(MPI_File fh, int * amode)
MPI_File_set_info	(MPI_File fh, MPI_Info info)
MPI_File_get_info	(MPI_File fh, MPI_Info * info_used)
MPI_File_seek	(MPI_File fh, MPI_Offset offset, int whence)
MPI_File_get_position	(MPI_File fh, MPI_Offset * offset)
MPI_File_get_byte_offset	(MPI_File fh, MPI_Offset offset, MPI_Offset * disp)
MPI_File_seek_shared	(MPI_File fh, MPI_Offset offset, int whence)
MPI_File_get_position_shared	(MPI_File fh, MPI_Offset * offset)
MPI_File_get_type_extent	(MPI_File fh, MPI_Datatype datatype, MPI_Aint * extent)
MPI_File_set_atomicity	(MPI_File fh, int flag)
MPI_File_get_atomicity	(MPI_File fh, int * flag)
MPI_File_sync	(MPI_File fh)

Tabla 2.2: MPI-IO - View File

Operation Type	**Parameters**
int MPI_File_set_view	(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char * datarep, MPI_Info info)
int MPI_File_get_view	(MPI_File fh, MPI_Offset * disp, MPI_Datatype * etype, MPI_Datatype * filetype, char * datarep)

2.2. Caracterización del sistema de E/S en clúster de computadores

Para mejorar las prestaciones de E/S a través de la configuración del sistema de E/S es necesario conocer los requisitos de E/S de las aplicaciones científicas paralelas y el sistema de E/S subyacente. El sistema de E/S debe ser caracterizado tanto a nivel prestacional como estructural. Para lograr esto es necesario conocer los conceptos de prestaciones de E/S, E/S paralela de las aplicaciones científicas, y sistemas de E/S para clúster de computadores. Un sistema de E/S debe satisfacer los siguientes requisitos[28]:

- Proporcionar el ancho de banda y capacidad que le exigen la aplicaciones actuales. Las prestaciones del sistema de E/S debe ser balanceado con la velocidad de cómputo y el ancho de banda de las redes de interconexión de los computadores paralelos.
- Minimizar la latencia de los accesos para reducir el tiempo de los procesos en espera. Esto es posible conectando los nodos de E/S a través de una red de altas prestaciones.
- Proporcionar fiabilidad frente a fallas, y mantener la disponibilidad de los datos cuando algún elemento falle o sea reemplazado.
- Conseguir efectividad de los Costos. En muchas ocasiones, agregar hardware de E/S de altas prestaciones es muy costoso. Sin embargo, el costo puede ser justificado si el sistema de E/S permite al computador, como una unidad, usarse más efectivamente, especialmente cuando consideramos que los sistemas de E/S internos reducen el espacio de los servidores de datos externos.
- Soportar operaciones complejas. Los sistemas de ficheros deben controlar de diseno de datos de los archivos, proporcionar la habilidad de particionar los datos entre los procesos de la aplicación y soportar la compartición de datos entre los procesos. Estas características deberían estar también en los modelos de programación y de particionado de datos usado en los computadores paralelos.

Además un sistema de E/S paralelo debe tener en cuenta la Arquitectura de E/S y también las librerías de E/S necesarias para realizar los accesos en forma eficiente. Debido a que las prestaciones que una aplicación pueda obtener en una configuración de E/S está muy influenciada por sus patrones de acceso se debe también considerar el tipo de aplicación y los patrones de accesos.

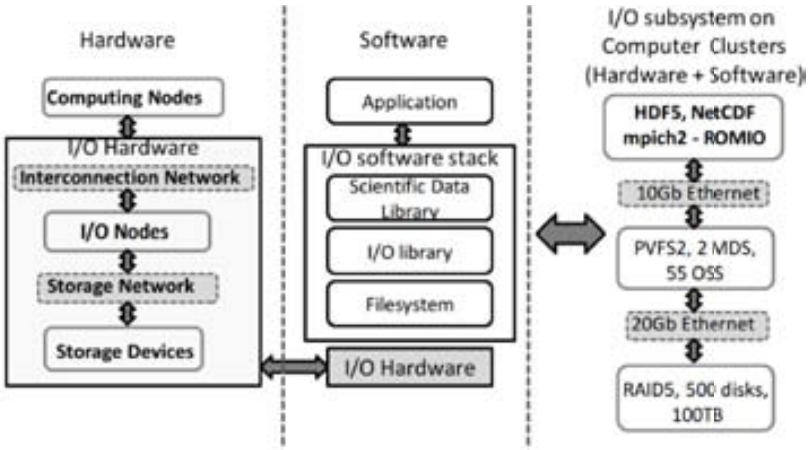


Figura 2.6: Sistema de E/S en clúster de computadores

El sistemas de E/S de los sistemas de cómputo de altas prestaciones esta compuesto tanto por hardware como por software de E/S. La Figura 2.6 muestra una estructura estándar que sigue un esquema jerárquico en función del camino que siguen los datos para ser escritos o leídos en el clúster de computador.

2.2.1. Arquitectura de E/S Paralela

Las arquitecturas de los computadores paralelos deben proporcionar un balance entre cómputo, ancho de banda y capacidad de memoria, capacidad de comunicación y E/S. En el pasado, mucha parte del esfuerzo del diseño se centraba en el hardware y software del cómputo básico y la comunicación. Esto ha llevado a computadores desbalanceados con pobres prestaciones de E/S. En la actualidad se dedica esfuerzo en el diseño de hardware y software para el sistema de E/S en los computadores paralelos. Las arquitecturas que surgieron para los computadores paralelos se basan en sistemas de E/S acompañados con una colección de nodos de E/S dedicados, donde cada uno gestiona y proporciona acceso a un conjunto de discos. Los nodos de E/S están conectados a otros nodos por una red de interconexión de la misma forma que lo nodos de cómputo se conectan con otros nodos. En esta sección se explica que se entiende por arquitectura de E/S paralela según Kotz [29].

Conexión

Una red de interconexión es necesaria para mover los datos entre los múltiples dispositivos de E/S (o nodos de E/S, controladores, etc) y las múltiples memorias. Existen dos modelos básico para la conexión:

- cada procesador está conectado a su disco local o a discos compartidos. Cualquier

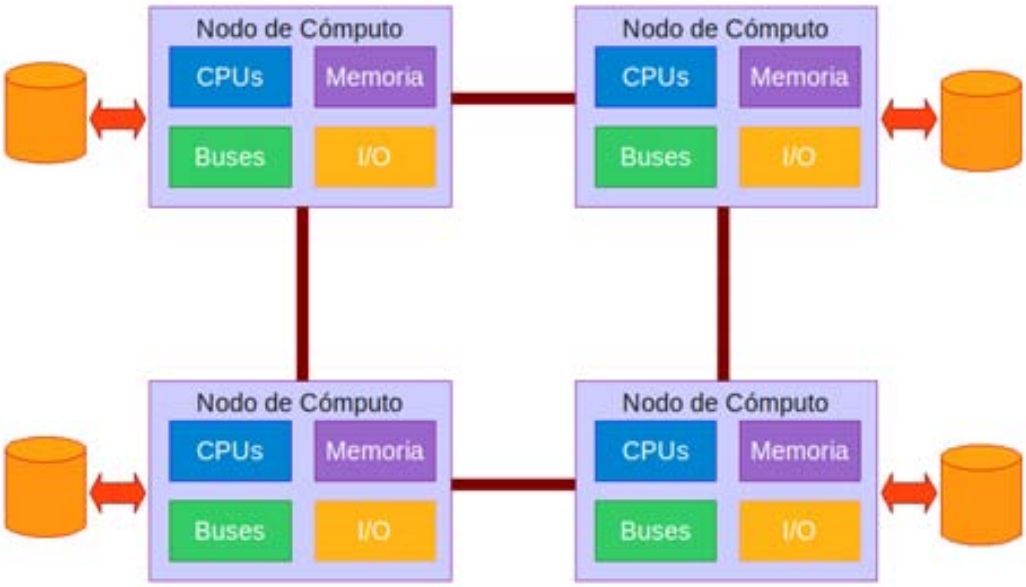


Figura 2.7: Cada procesador esta conectado a su propio disco local o a discos remotos

procesador puede acceder a su disco local o a un disco remoto en un procesador remoto. Si un procesador está conectado con un disco local, el sirve como controlador del disco local (ver Figura 2.7)

- exiten dos clases de nodos: nodo de cómputo y nodo de E/S. Dependiendo de la posición de los nodos de E/S, el subsistema de E/S puede ser visible o independiente. El ratio de los nodos de E/S a los nodos de cómputo es muy importante para mantener el balance entre cómputo y subsistemas de E/S. Este ratio afecta el rendimiento de E/S global (y el de la aplicación). Los nodos de E/S pueden tener el mismo hardware que los nodos de cómputo y pueden no ejecutar cómputo. Cada nodo de E/S actúa como un controlador dedicado para un conjunto de discos asociado. En este modelo, cada acceso a disco por un nodo de cómputo requiere intercambio de mensajes con el nodo de E/S controlando el disco. Cada nodo de E/S está conectado a un conjunto de discos a través de una red de interconexión.

  El subsistema de E/S está formado por los nodos de E/S, discos y red de interconexión. Dependiendo de la posición de los nodos de E/S, el subsistema de E/S puede ser, respecto de los nodos de cómputo, independiente o empotrado. En un subsistema de E/S independiente, los nodos y los discos están organizados físicamente como unidades separadas. El subsistema de E/S está conectado con el resto de la máquina (o parte de cómputo) a través de un canal de E/S (figura

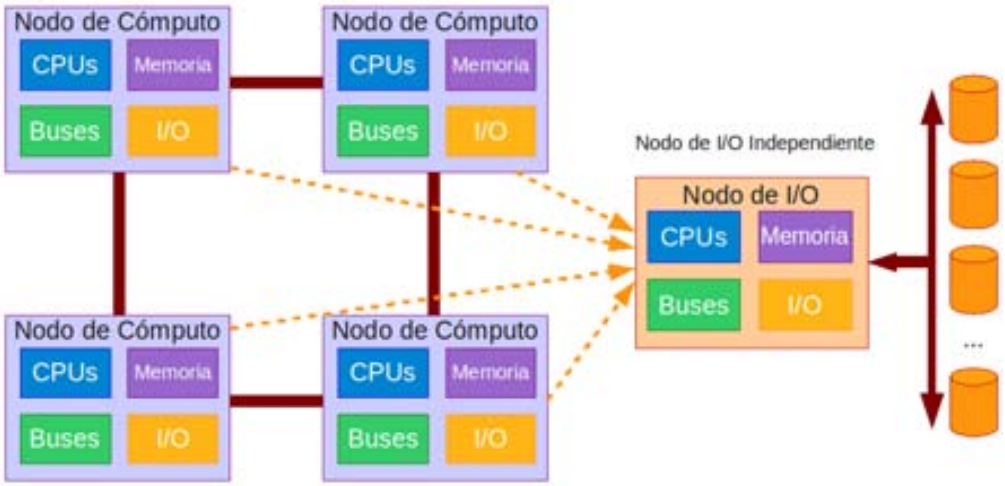


Figura 2.8: Nodo de E/S Independiente. Los nodos de cómputo acceden a disco por medio de los nodos de E/S.

2.8) En un subsistema de E/S empotrado, la E/S y los nodos de cómputo están organizados en una única unidad física. Los nodos de E/S utilizan la misma red de interconexión que los nodos de computo (Figura 2.9). Si múltiples procesadores envían peticiones de E/S al mismo controlador de E/S simultáneamente, resultará una congestión. Por tanto, es necesario proporcionar varios controladores de E/S los cuales pueden compartir la carga de trabajo.

Hay cuestiones a tener en cuenta en la conexión de los dispositivos de E/S y los nodos de cómputos:

- Hay una red o una subred dedicada al tráfico de la E/S? El tráfico de E/S comparte la red de interconexión interprocesos?
  Un extremo es conectar los nodos de E/S, o incluso los dispositivos de E/S, directamente a la red de cómputo. Otro extremo es proporcionar una red dedicada a la E/S. También se puede conectar cada nodo de E/S a unos pocos puntos en la red de cómputo usando enlaces extras. El tráfico de la E/S es diferente del tráfico de la red de cómputo. Los mensajes de E/S tienden a ser grandes y a ráfagas, mientras que los mensajes entre procesos tienden a ser pequenos. El *Throughput* es la meta de la comunicación relacionada con la E/S, mientras que la latencia es lo importante para los mensajes interprocesos. Cada tipo de comunicación puede causar congestión o contención que impactará negativamente en las prestaciones de la otra.

- La interfaz de la red incluye DMA o memoria compartida? Acepta accesos

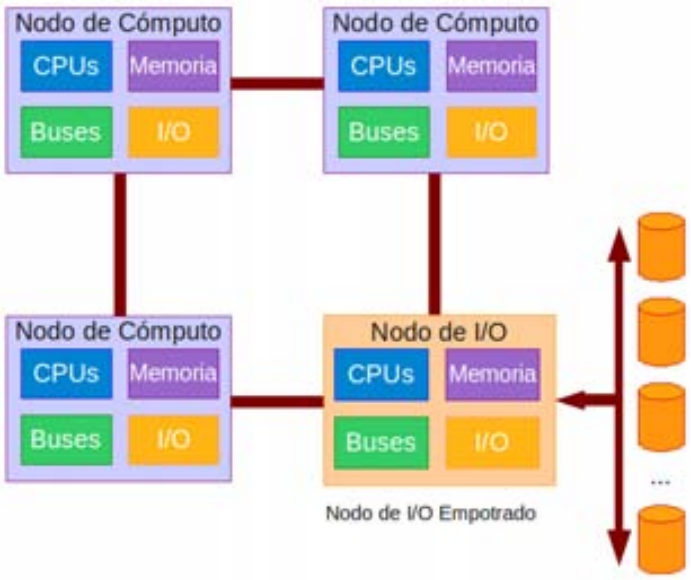


Figura 2.9: El nodo de E/S se encuentra integrado. Los nodos de Cómputo acceden a los discos por el nodo de E/S.

a nivel de usuario o requiere privilegios de usuarios? El adaptador de E/S está adjuntado directamente a la red de interconexión o al nodo de E/S?
Esto es crítico debido a que los sistemas de E/S dependen de la habilidad de mover datos. En la actualidad el computador y la tarjeta se comunican entre sí para que puedan proceder al intercambio de información. De esta manera, el computador asigna parte de su memoria a las tarjetas que tienen DMA (acceso directo a memoria). La interfaz de la tarjeta indica que otro nodo está solicitando datos del nodo. El bus del nodo transfiere los datos de la memoria del nodo a la tarjeta de red. Si los datos se desplazan demasiado rápido como para que el adaptador proceda a su procesamiento, se colocan en la memoria del buffer de la tarjeta (RAM), donde se almacenan temporalmente mientras se siguen enviando y recibiendo los datos.

Gestión

La E/S implica el movimiento de los datos entre la memoria y los dispositivos periféricos y viceversa. En los computadores paralelos hay varias memorias y varios dispositivos periféricos. Una cuestión clave, entonces, es la Gestión, Qué procesadores gestionan el acceso a los discos? Hay tres soluciones comunes, donde la Gestión puede ser:

- Centralizada en un procesador: El enfoque centralizado (figura 2.10)es común en sistemas SPMD, donde la mayoría de la gestión es centralizada y el modelo de

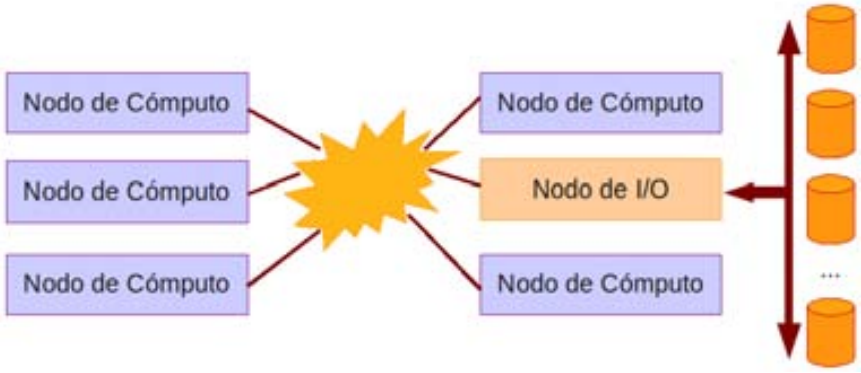


Figura 2.10: Gestión del accesos a los dispositivos Centralizada. Un nodo (Nodo de E/S) gestiona el acceso a los discos.

programación es síncrono. En grandes sistemas MPMD, sin embargo, representa un potencial cuello de botella, especialmente cuando son usados con un modelo de programación asíncrono.

- Distribuida entre todos los procesadores: Pocos sistemas eligen la gestión distribuida entre todos los procesadores (figura 2.11), prefiriendo concentrar el hardware de E/S en un subconjunto de nodos procesador que son usualmente dedicados a actividades de E/S. Cuando la gestión de los dispositivos es distribuida entre todos los nodos:
  - Un procesador está conectado a su propio disco local.
  - Cualquier procesador puede acceder a su propio disco local o a un disco remoto en un procesador remoto
  - Mejora la localidad si cada procesador puede enfocar su actividad de E/S en sus dispositivos locales
  - Es útil para soporte de memoria virtual
  - Es difícil caracterizar el impacto si cambia el workload
  - Mejora la localidad de accesos, balanceo de carga y fiabilidad
  - Posiblemente se presente interferencia con computaciones de usuario.
- Distribuida entre un subconjunto de procesadores que están dedicados a la E/S (figura 2.12): El hardware de E/S se concentra en un subconjunto de nodos que estan dedicados a las actividades de E/S. La concentración de hardware de E/S en nodos de E/S tiene algunas ventajas sobre la distribución total:

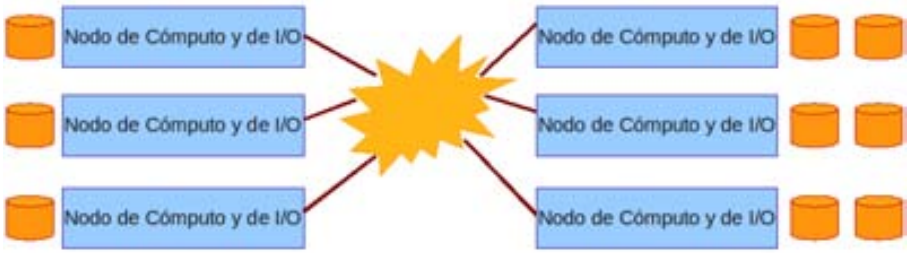


Figura 2.11: Gestión de Distribuida entre todos los procesadores

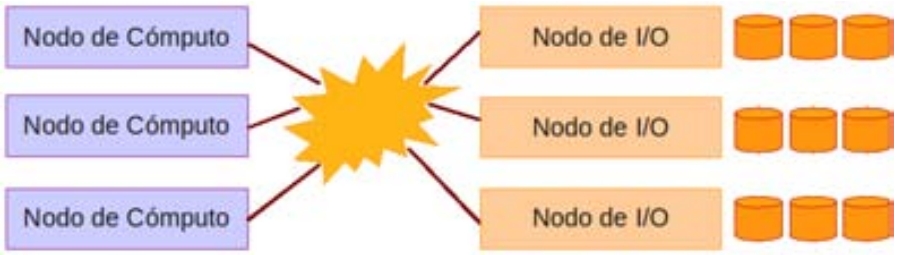


Figura 2.12: Gestión de los dispositivos a cargo de un subconjunto de nodos de E/S.

- El número de nodos de E/S y dispositivos pueden ser elegidos independientemente del número de nodos de cómputo, permitiendo una configuración del sistema más flexible.
- Los nodos de E/S pueden ser diferentes, es decir, con una CPU diferente, más o menos memoria, hardware DMA especializado y desde luego adaptadores para periféricos y buses de E/S.
- Pueden necesitarse menos adaptadores.
- El sistema puede ser más simple, puesto que los nodos de cómputo pueden tener características físicas diferentes que los nodos de E/S. Además cada uno puede encontrarse en diferentes tipos de racks.
- La actividad del servicio de E/S no impacta en aplicaciones de cómputo por robo de ciclos o memoria, o causando interrupciones inesperadas.
- Por otro lado, la gestión de la E/S distribuida entre todos los procesadores podría lograr mejor localidad, si cada procesador pudiera enfocar su actividad de E/S en sus dispositivos locales. Es difícil caracterizar el impacto en el rendimiento de su localidad dada la gran variedad de workloads y arquitecturas de redes de interconexión, pero parece adecuado que los discos locales sean útiles para paginación y otras formas de soporte de memoria virtual para computaciones *out-of-core*.

Ubicación

Todos los computadores paralelos tienen una red de interconexión con una topología determinada. La latencia de comunicación, el ancho de banda y la contención en estas redes a menudo depende de la posición relativa del destino de la comunicación. Entonces la posición de los nodos o dispositivos de E/S en la topología de la red pueden tener un impacto significativo en el rendimiento del sistema de E/S. Existen 3 enfoques típicos:

- La posición es ignorada; los nodos de E/S están colocados en cualquier sitio.
- Todos los nodos de E/S están agrupados en su propia partición de la red.
- Los nodos de E/S están distribuidos alrededor de la red, pero en posiciones elegidas cuidadosamente.

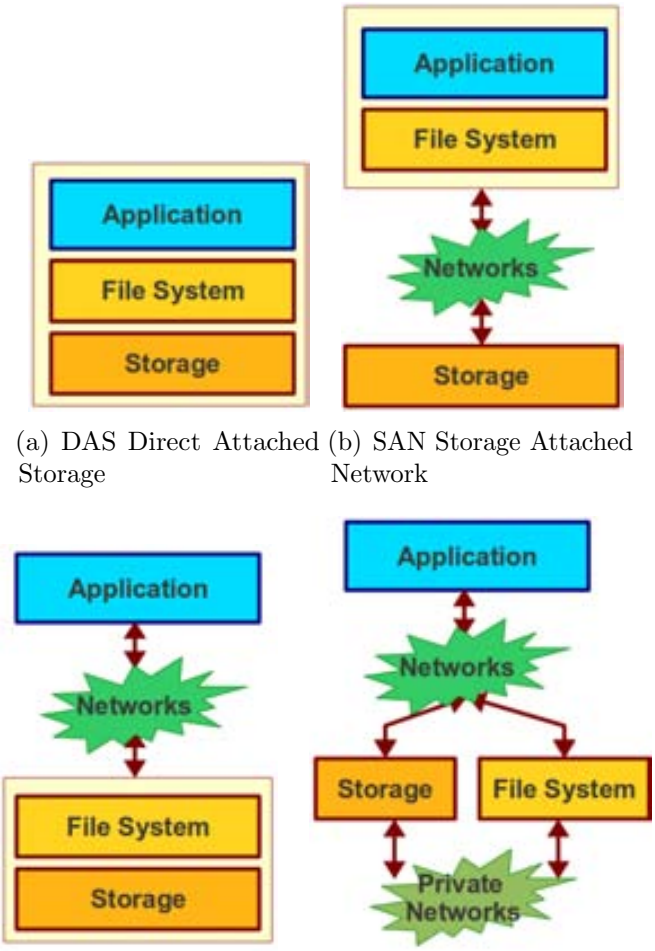
El desarrollo de las redes y de Internet han hecho que el compartir archivos sea algo sencillo, común y requerido. La posición de los discos o dispositivos de almacenamiento si estamos conectados a una red pueden tener un impacto significativo en el rendimiento del sistema. Hay 4 formas de acceder a los datos de acuerdo a la ubicación y gestión los dispositivos de almacenamiento: Direct Attached Storage (DAS) que se muestra en la Figura 2.13(a)), Storage Attached Network (SAN) que se muestra en la Figura 2.13(b), Network Attached Storage (NAS) que se muestra en la Figura 2.13(c) y y Network Attached Storage Devices (NASD) que se muestra en la Figura 2.13(d).

Buffering/Caching

El buffering y caching es un factor importante en cualquier sistema de E/S. El buffering es importante, por ejemplo, entre la unidad de disco y red de interconexión, para compensar las diferencias de velocidad y las diferentes granularidades (bloques o paquetes), y las ráfagas dadas por las características de los dispositivos (búsqueda en los discos) o la carga (congestión de la red).
Un buffer/cache, almacena un pool de los bloques usados recientemente, este puede evitar accesos completos. Un buffer cache puede ser particularmente importante en los nodos de E/S de los computadores paralelos, porque pueden tomar ventaja de la localidad interproceso, cuando múltiples procesadores están accediendo a diferentes partes del mismo bloque.

Disponibilidad

Un computador paralelo esta formado por varios componentes, usados en paralelo para mejorar las prestaciones. Cuando los datos de los archivos son distribuidos sobre



(a) DAS Direct Attached Storage

(b) SAN Storage Attached Network

(c) NAS Network Attached Storage

(d) NASD Network Attached Storage Devices

Figura 2.13: Acceso a los datos de acuerdo a la ubicación y gestión los dispositivos de almacenamiento

múltiples dispositivos de almacenamiento, la falla de cualquier dispositivo causa la perdida del archivo. Por lo tanto, la distribución de los datos sobre múltiples dispositivos aumenta las prestaciones pero decrementa la fiabilidad. Para la falla en los discos normalmente es enmascarada con los sistemas RAID. La falla en los nodos de E/S pueden ser enmascarado con la redundancia de componentes.

2.2.2. Stack de Software de E/S Paralela

Muchas avances se han hecho en los sistemas de E/S tanto en la comunidad de HPC como en grupos externos. Una categoría clave de mejora ha sido la organización del software de E/S y la definición de interfaces estándar para varias capas, tanto en software como en hardware. Otro avance ha sido el desarrollo y despliegue de implementaciones de múltiples

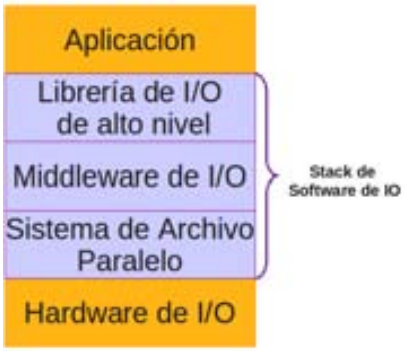


Figura 2.14: Stack de Software de E/S

Figura 2.15: Librerías de E/S en clúster de computadores

sistemas de archivos paralelos.
El software de E/S ha pasado de ser una librería secuencial monolítica a una *stack* de software en tres niveles: librerías de E/S de alto nivel, middleware o librería de bajo nivel y los sistemas de archivos paralelos. Estas capas se muestran en la figura 2.14. Cada una de estas capas proporciona un conjunto de funcionalidades; y para comprender el rol del sistema de archivo paralelo en este sistema multicapa, primero se debe ver *stack* de software como un todo [30].

Librerías de E/S de Alto Nivel

Las librerías de E/S son interfaces que se proporcionan, a nivel de software, que permiten realizar las operaciones de E/S de forma más eficiente. Existen dos tipos de Librerías de E/S las de bajo nivel y las de alto nivel. La figura 2.15 muestra la jerarquía de las librerías de E/S. A continuación se presentan una breve descripción de las más usadas en la computación de altas prestaciones.

- Bajo Nivel

Las librerías de E/S de bajo nivel soportan los accesos a archivos paralelos e implementan las diferentes técnicas de E/S. Estas interfaces trabajan en diferentes niveles de abstracción, son interfaces de bajo nivel en el sentido que no necesitan ninguna información acerca de los datos que se van a almacenar, y la aplicación debe seguir la pista de los datos que residen en el archivo. Uno de las interfaces de E/S más usados y un estándar en E/S Paralela es MPI-IO.

MPI-IO es un extensión del estándar MPI-2, es una interfaz disenada específicamente para E/S de altas prestaciones y portable. Permite al usuario especificar patrones de accesos discontiguos y leer o escribir todos los datos con una simple llamada de función de E/S. También permite al usuario especificar colectivamente las solicitudes de E/S de un grupo de procesos, proporcionando mayor información de acceso y alcance de optimización. Una de las implementaciones más usada de MPI-IO es ROMIO [31].

- Alto Nivel

  Las librerías de alto nivel, también conocidas como librerías de datos científicos, gestionan los datos a un nivel más alto. Los programas pueden manipular estructuras de datos complejas directamente, y la librería registra automáticamente el tipo de información y otros metadatos útiles. Las librerías de datos científicos ofrecen otras características útiles, por ejemplo, las aplicaciones pueden consultar los archivos para determinar el tamano y forma de las estructura de datos antes de leerlo en memoria.

  Las librerías más usadas en el campo de la computación de altas prestaciones son HDF5 [32] y NetCDF [33] tanto en su versión paralela como serial.

  Parallel netCDF [34] es una extensión de la librería netCDF desarrollada en Unidata [33], éste proporciona para el almacenamiento multidimensional, un conjunto de tipos de datos en un formato portable, así como almacenamiento de atributos en estos datos.

  NetCDF (Network Common Data Form) es un conjunto de librerías de formato de datos independiente de las máquinas y autodescritptivo que soporta la creación, acceso y compartición de datos científicos orientado a *arrays*. El formato de dato es *self-describing*, esto significa que hay una cabecera en la cual se describe el diseno del resto del archivo, en particular de los arreglos de datos, así como los metadatos de los archivos en la forma nombre/valor de los atributos.

  HDF5 [32] simplifica la estructura de un archivo al incluir dos tipos principales

de objetos:

- *Datasets*, que son arreglos multidimensionales de un tipo homogéneo y
- *Groups*, que son estructuras contenedoras que pueden almacenar *Datasets* y otros *groups*.

HDF5 Paralelo abre un archivo con un comunicador. Éste retorna un manejador para ser usado para futuras accesos al archivo.

Middleware de E/S

Este componente es responsable de proporcionar la base para que las librerías de alto nivel puedan ser construidas, pero también se las puede usar para hacer las operaciones de E/S, pero requiere un nivel más de experiencia. Esta capa proporciona un mapping desde interfaces relativamente simples de los sistemas de ficheros paralelos en interfaces que aprovechan conceptos de modelos de programación, tales como comunicadores, tipos de datos del modelo de programación MPI. Uno de los mejores ejemplos es la interfaz MPI-IO.

Sistema de Archivos Paralelo

El sistema de archivos paralelo es el responsable de gestionar todos los componentes hardware de almacenamiento. Éste presenta una simple vista lógica de este hardware que puede ser aprovechado por otras capas de software. También hace cumplir un modelo de consistencia para que los resultados de los accesos concurrentes sean correctos. El sistema de archivos paralelo debe escalar para un gran número de clientes, manejar muchas operaciones concurrentes y aparentemente independientes. Debe presentar una interfaz en la cual los componentes de las implementaciones de las interfaces de alto nivel puedan ser construidas. Destacamos tres sistemas de archivos paralelos que se pueden encontrar en entornos de HPC. El Sistema de Archivo Paralelo General (GPFS) de IBM que surgió de el sistema de archivo multimedia *Tiger Shark* [35] y que ha sido ampliamente usado en plataformas AIX. El sistema de archivo Lustre [36] que es uno de los sistemas más usados entre los supercomputadores del TOP 500. El sistema de ficheros paralelo PVFS2 que es le segundo sistema de archivo desarrollado por los autores de [37]. Este difiere de las versiones anteriores porque es una versión libre, abierta a la comunidad para construir sistemas de ficheros paralelos para HPC y sirve no sólo para una producción de un sistema de archivo paralelo sino también para investigación en conceptos de E/S paralela y está disponible para clúster Linux. En este trabajo se ha trabajado con dos de estos sistemas de fichero

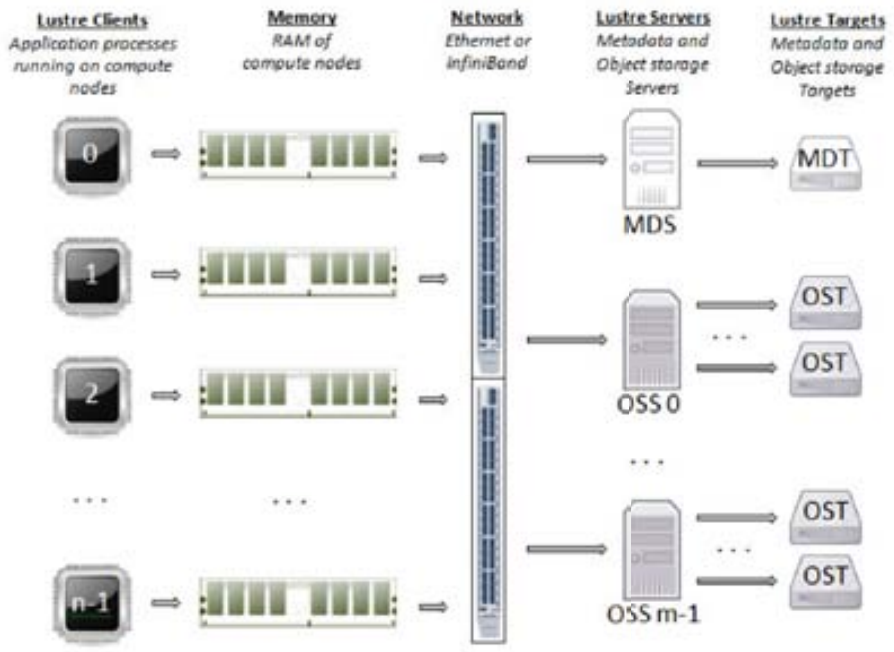


Figura 2.16: Componentes de Lustre

paralelos: Lustre y PVFS2. Son estos dos sistemas los que se explican a continuación con mayor detalle para poder contextualizar el trabajo realizado en el capítulo 4.

- Sistema de Ficheros Paralelo Lustre

  El sistema de archivos Lustre se compone de un conjunto de servidores de E/S denominados *Object Storage Servers* (OSSs) y los dispositivos de almacenamiento llamados *Object Storage Targets (OSTs)*. Los *metadatos* son controlados por un servidor de metadatos denominado *Metadata Server* (MDS) y almacenados en un *Metadata Target* (MDT). Un sistema Lustre simple normalmente tiene 1 MDS y un MDT. La Figura 2.16 muestra la interacción entre los componentes de Lustre en un simple clúster.

  A continuación se describe la función que cumplen cada uno de estos elementos:

  - Object Storage Servers (OSSs): gestionan un pequeño conjunto de OST controlando los accesos de E/S y las solicitudes de la red que les llegan. Los OSSs contienen algunos metadatos sobre los archivos almacenados en sus OST. Por lo general se sirven entre 2 y 8 OST, hasta 16 TB de tamaño cada uno.
  - Object Storage Targets (OSTs): son dispositivos de almacenamiento de bloque que almacenan los datos del archivo del usuario. Un OST puede

ser pensado como un disco virtual, aunque a menudo se compone de varios discos físicos, por ejemplo en una configuración RAID. Los datos del archivo del usuario se almacenan en uno o más objetos, con cada objeto almacenado en un OST separado. El número de objetos por archivo es configurable por el usuario y se puede ajustar para optimizar el rendimiento de una carga de trabajo determinada.

- Metadata Server (MDS): Es un único nodo de servicio que asigna y sigue las ubicaciones de almacenamiento con cada archivo con el fin de dirigir las solicitudes de E/S para el conjunto de OST y los OSS correspondientes. Una vez que se abre un archivo, el MDS no se involucra en la E/S del fichero. Esto es diferente de muchos de los sistema de ficheros de los clústeres basados en bloques donde el MDS controla la asignación de bloques, eliminándolo como fuente de contención para el fichero.
- The Metadata Target (MDT): almacena los metadatos (por ejemplo, nombres de archivos, directorios, los permisos y la estructura del archivo) en el almacenamiento conectado a un MDS. Almacenar los metadatos en un MDT proporciona una eficiente división del trabajo entre los recursos computacionales y de almacenamiento. Cada archivo en el MDT contiene la disposición *layout* del archivo de datos asociado, incluyendo el número de OST y el identificador de objeto y los punteros a uno o más objetos asociados con el archivo de datos.

Cuando un nodo de cómputo necesita crear o tener acceso a un archivo, se solicita la ubicación del almacenamiento al MDS y al MDT asociado. Las operaciones de E/S se realizan directamente con los OSSs y OSTs asociado al archivo sin pasar por el MDS. Para las operaciones de lectura, los datos del archivo se mueven desde los OSTs a memoria. Cada OST y MDT mape a un subconjunto distinto de dispositivos RAID. La capacidad total de un sistema Lustre es la suma de las capacidades de los OSTs.

- Sistema de Fichero Paralelo PVFS2

  PVFS2 es un sistema de ficheros paralelos de código abierto y escalable dirigido a entornos de computación de altas prestaciones. La arquitectura es modular, lo que facilita la inclusión de nuevos soportes de hardware y nuevos algoritmos. Esto hace de PVFS uno sistema indicado para la investigación de sistemas de ficheros paralelos.

Un sistema de fichero PVFS2 puede estar formado por los siguientes componentes: el servidor PVFS2 *pvfs2-server*, la interfaz de sistema *system interface* , el gestor de interfaz *management interface*, el driver del kernel de linux *Linux kernel driver* , clientes pvfs2 *pvfs2-client* y el device PVFS2 de ROMIO *ROMIO PVFS2 device*. A continuación se describe cada uno de estos componentes:

- *pvfs2-server*: es un demonio del sistema de fichero que se ejecuta completamente en espacio de usuario. Puede haber varias instancia de *pvfs2-server* ejecuntándose en diferentes máquinas. Cada instancia podría actuar como un metadata server, un servidor de datos (*I(O-server*), o ambos a la vez. Los servidores de datos almacenan los datos asociados con cada archivo, típicamente distribuido en múltiples servidores en una forma round-robin. Los metada servers almacenan la meta información de los ficheros, tales como permisos, timestamps y parámetros de distribución. Los metadata servers también almacenan la jerarquía de directorio.
- *system interface*: es un API en espacio de usuario de bajo nivel que provee acceso al sistema de fichero PVFS2. Esta es implementada como simple librería, llamada *libpvfs2*. Ésta API no mapea directamente con funciones POSIX. Particularmente, es un API sin estado que no tiene conceptos de *open()*, *close()*, o descriptores de archivos. Sin embargo, esta abstrae la tarea de comunicación con varios servidores simultáneamente.
- *management interface*: es una implementación similar a la interfaz de sistema. Es una API suplementario que agrega funcionalidad que normalmente no es vista por todos los usuarios del sistema de fichero. Esta funcionalidad esta destinada a los administradores, y para aplicaciones como fsck o monitorización de rendimiento que requiere información del sistema de fichero de bajo nivel.
- *Linux kernel driver*: es un módulo que puede ser cargado en el kernel de linux sin modificarlo para proporcionar soporte VFS para PVFS2. Es un componente que permite que las aplicaciones de Unix estándares (incluyendo utilidades como ls y cp) trabajen en PVFS2. Este también requiere el uso de una aplicación de ayuda en espacio de usuario llamada *pvfs2-client*.
- *pvfs2-client*: es un demonio en espacio de usuario que maneja la comunicación entre los servidores de PVFS2 y el driver del kernel. Su principal función es convertir las operaciones VFS en operaciones de la interfaz del sistema. Un demonio *pvfs2-client* debe ser ejecutado en cada cliente que desee acceder al

sistema de fichero a través de la interfaz VFS.

- *ROMIO PVFS2 device*: es un componente de la implementación ROMIO de MPI-IO (distribuido por separado) que provee soporte para PVFS2.

2.3. Prestaciones de E/S paralela

Cuando se habla de las prestaciones de los sistemas de E/S se hace referencia a la tasa de transferencia, latencia y operaciones de E/S por segundo. Para evaluar las prestaciones de E/S este trabajo utiliza los conceptos definidos por Parallel I/O Benchmarking Consortium [38](PIO-BENCHMARK) que provee un conjunto de benchmark para caracterizar las prestaciones de los sistemas de ficheros paralelos. Se centran específicamente en aplicaciones paralelas que usan MPI para la comunicación y MPI-IO o POSIX para los accesos de E/S.

Debido a que el presente trabajo se centra en aplicaciones que realizan la E/S paralela con MPI-IO se ha decidido seleccionar esta definición de conceptos. La caracterización de la E/S de las aplicaciones científicas paralelas se realiza teniendo en cuenta los conceptos usados por los benchmark para que esto sea comparable entre diferentes sistemas de E/S.

- *Transfer rate* o *bandwidth* medida en MB/sec. La tasa de transferencia es calculada teniendo en cuenta el tiempo empleado para abrir, escribir/sync o leer y cerrar el fichero.
- tiempo de escritura o lectura (write/read time): el tiempo para todas las llamadas de escritura o lectura para los datos a transferir (no hace referencia a una operación individual de escritura o lectura).
- tiempo de apertura o cierre (open/close time): tiempo entre el inicio de la llamada colectiva para abrir o cerrar el fichero y su terminación.

Cuando se habla de mejorar las prestaciones de E/S para la aplicación se espera aumentar la tasa de transferencia para los patrones de acceso de la aplicación para reducir los tiempos de E/S.

2.4. Trabajos Relacionados

Existen varios paper que presentan la evaluación de prestaciones de los sistemas de E/S para computadores de altas prestaciones. Aunque la mayoría están centrados en supercomputadores, los factores de E/S presentados en estos trabajos también impactan en las prestaciones de los sistemas de E/S de clúster pequenos y medianos.

Laros en [8] presenta un análisis de prestaciones del sistema de I/O del supercomputador Red Storm del Sandia National Laboratories. Presenta en primer lugar una estimación teórica de los valores pico y además presentan un metodología para evaluar las prestaciones de Red Storm que consiste en probar una camino simple de datos para archivo compartidos y para un fichero por proceso. El estudio demuestra que las prestaciones se decrementan significativamente respecto del valor pico teórico cuando se aumentan el número de procesos por cada OSS.

Yu en [9] realiza una evaluación empírica de prestaciones para varias interfaces de I/O (POSIX IO, MPI-IO, y HDF5). Evalúan el rendimiento de los componentes individuales de almacenamiento. Además, examinan la escalabilidad de los metadatos, y los benchmark intensivos en I/O en Jaguar, y demuestran que el patrón de distribución de archivos puede tener un impacto en el ancho de banda agregado de E/S.

Lang en [11] presenta un caso de estudio de los desafíos de la E/S para el rendimiento y la escalabilidad en IBM Blue Gene/P system en el Argonne Leadership Computing Facility. Evalúan tanto el hardware como el software del sistema de E/S a nivel de sistema de fichero PVFS2 y LUSTRE.

Carns en [12] presenta una metodología para la caracterización de la I/O, escalable y para el sistema completo que combina la instrumentación del dispositivo de almacenamiento, el análisis del sistema de archivos estático y un nuevo mecanismo para la captura del comportamiento a nivel de aplicación.

A partir de estos trabajos, lo primero que se definió, es la estructura del sistema de E/S. Después se determino la forma de conocer los valores picos de prestaciones. En los casos anteriores los valores picos se obtuvieron de la descripción técnica de los componentes hardware o con comandos linux tipo *dd* o *sci_dd*.

En esta tesis se utiliza el benchmark IOzone a nivel de sistema de fichero local para calcular los valores picos de prestaciones. Se hace esta elección ya que ofrece la posibilidad de ajustar más factores que comandos linux simples . Además se consideró necesario contar con valores medios de prestaciones para diferentes patrones de E/S a nivel de Librería de E/S y para ello se optó por el benchmark IOR que es muy usado en área de E/S paralela.

Debido a que las prestaciones que una aplicación consigue en un sistema específico depende de sus patrones de E/S también se ha evaluado el estado del arte en el área de caracterización de E/S de las aplicaciones científicas paralelas.

Carns en [13] presenta a DARSHAN una herramienta de trazas para cargas de E/S de los sistemas de la era de la Peta escala. Darshan está disenado para capturar una imagen exacta del comportamiento de E/S, incluyendo características tales como los patrones de acceso a los archivos, con una sobrecarga mínima de aplicación.

Byna en [16] presenta el concepto de firmas de E/S y lo usan para prefetching. Es un trabajo que presenta una interesante clasificación de patrones locales. A partir de estos se puede conocer las características de E/S para determinar el comportamiento de la aplicación.

Shan and Shalf en [17] presentan una propuesta para caracterizar la E/S de tres aplicaciones y usan esta información para configurar IOR y predecir los tiempo de E/S totales de esas tres aplicaciones.

Para la definición del modelo de E/S este trabajo ha usado parte de la definición de patrones de acceso local de Byna.

Aunque en este trabajo se utilizo IOR de forma similar a Shan and Shalf, la principal diferencia del presente trabajo, es que se ha establecido un modelo de E/S que no requiere que el usuario estudie el código fuente de la aplicación y se ha definido la configuración de IOR para varios tipo de aplicaciones y no sólo para las que se presentan en este documento.

En el presente trabajo se presenta una metodología que permite evaluar las prestaciones del sistema de E/S teniendo en cuenta valores picos y medios del sistema de E/S y los patrones de la aplicación expresado en un modelo de E/S. Dado que este modelo de E/S es obtenido a partir de la lógica algorítmica es independiente de la arquitectura de E/S. Además, tanto el modelo como las valores picos y medio son expresados de manera que la E/S de la aplicación se pueda relacionar con las prestaciones del sistema de E/S.

Capítulo 3

Metodología para la caracterización de la E/S de aplicaciones científicas paralelas

Cuando se habla de las prestaciones de los sistemas de E/S se hace referencia a la tasa de transferencia, latencia y operaciones de E/S por segundo. Sin embargo, Cómo se expresa esta información para la aplicación? La aplicación realiza operaciones de apertura, escritura, lectura, cierre en sus ficheros, y la métrica que le interesa de la E/S es el tiempo de E/S de manera que este tiempo no lleve a tener procesos ociosos en la aplicación paralela.

Para poder establecer una relación entre la E/S de las aplicaciones paralelas y las prestaciones que ofrecen los sistemas de E/S es necesario expresar el comportamiento de E/S de la aplicación de una forma que se pueda usar en diferentes sistemas de E/S y donde las prestaciones sean comparables.

El modelo de comportamiento de E/S de la aplicación debería ser obtenida en un nivel que sea independiente de la parte física del sistema de E/S. Si la E/S de la aplicación fuera expresada en base a la lógica algorítmica de la aplicación esta se podría usar en otros sistemas de E/S.

En este capítulo se presenta la metodología que extrae los patrones de acceso de E/S y metadatos de la aplicación expresándolos en un modelo de E/S. Este modelo está en función de los patrones de acceso de la aplicación, donde son agrupados en fases de E/S y en base a estas fases se analizan las prestaciones para la aplicación en diferentes sistemas de E/S. Debido a que el modelo de E/S se obtiene a nivel de librería de E/S, éste es independiente de la arquitectura de E/S permitiendo usarlo en sistemas con diferentes configuraciones. La Tabla 3.1 muestra la notación utilizada en este capítulo.

Tabla 3.1: Resumen de la notación

Notación	Descripción
np	número de procesos de la aplicación paralela ;
IdF	Identificador de Archivo;
$displacement$	desplazamiento en la posición relativa del archivo (en *Bytes*) ;
rs	tamano de *request* para la operación (en *Bytes*);
rep	repeticiones del patrón de acceso;
$tick$	uniddad de tiempo lógica;
LAP	patrón de acceso local (Local Access Pattern)
AP	patrón de acceso (Access Pattern),
	LAPs similares para un conjunto de procesos
$weight_{(ph)}$	peso de la fase ph

3.1. Metodología propuesta

La metodología propuesta consta de los siguientes pasos (Figura 3.1):

- Recopilación de Trazas de la aplicación paralela,
- Procesamiento de trazas, Extracción de los patrones de acceso y del orden lógico de los eventos,
- Identificación de fases de E/S y Definición del modelo de E/S.

El modelo de E/S de la aplicación esta definido por tres caracterísitcas: metadatos, patrón espacial global y patrón temporal global. La caracterización se realiza off-line y una única vez a nivel de librería de MPI. Esto tiene dos ventajas, primero, el modelo de E/S es independiente del entorno de ejecución y, segundo, podemos evaluar el comportamiento de la aplicación a través del modelo en diferentes sistemas de E/S evitando el *overhead* de la herramienta de trazado.

El modelo es expresado por medio de fases de E/S, donde cada fase es una secuencia repetitiva del mismo patrón de acceso en un fichero para un número de procesos de la aplicación paralela.

A continuación se presenta la descripción de cada paso de la metodología y los conceptos usados para el modelo de E/S.

3.1.1. Recopilación de Trazas de la aplicación paralela

Para poder identificar los diferentes patrones de acceso de las aplicaciones paralelas es necesario definir qué información es necesario trazar y seleccionar la herramienta. En este

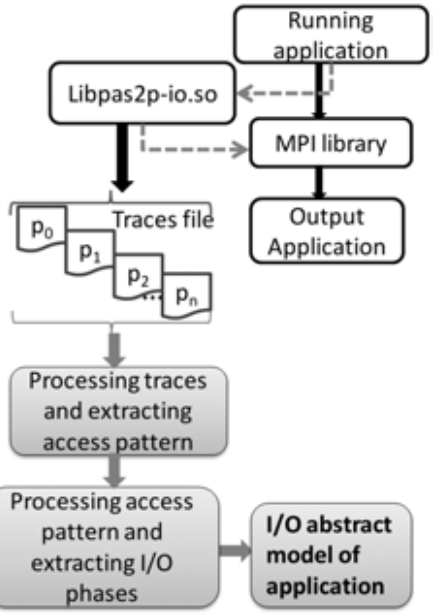


Figura 3.1: Proceso para la extracción del modelo de E/S

trabajo se optó por una librería de la herramienta PAS2P desarrollada en el Departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona (CAOS-UAB) [18] que es utilizada para trazar las operaciones de comunicación de aplicaciones paralelas de paso de mensaje. Esta librería fue extendida para trazar las operaciones de E/S del estandar MPI-2.

Para trazar las operaciones del estandar MPI-2 se ha definido un nuevo evento llamado evento de E/S, que es una parte de la tarea del proceso donde una operación es llamada. Un evento de E/S esta compuesto por *ID file, mpi-io operation, offset, displacement, size of etype, size of filetype, name of file, order of ocurrence, size of request, logical time, duration, count of datatype, and size of datatype*.

La Figura 3.2 muestra un ejemplo de la traza física original de PAS2P más los eventos de E/S. Un concepto que usa PAS2P para dar orden a los eventos de un aplicación paralela es un *tick*, que es definido como una unidad lógica que permite la sincronización lógica o dar un orden lógico a los eventos de comunicación y de E/S de los distintos procesos de la aplicación paralela.

Los siguientes pasos son realizados para obtener las trazas de la aplicación:

1. Compilación de la librería *libpas2p-io.c*.

   ```
   mpicc -shared -fPIC libpas2p-io.c -o bin/libpas2p-io.so -ldl
   ```

   Se usar una distribución MPI configurada con la opción *enable-shared*.

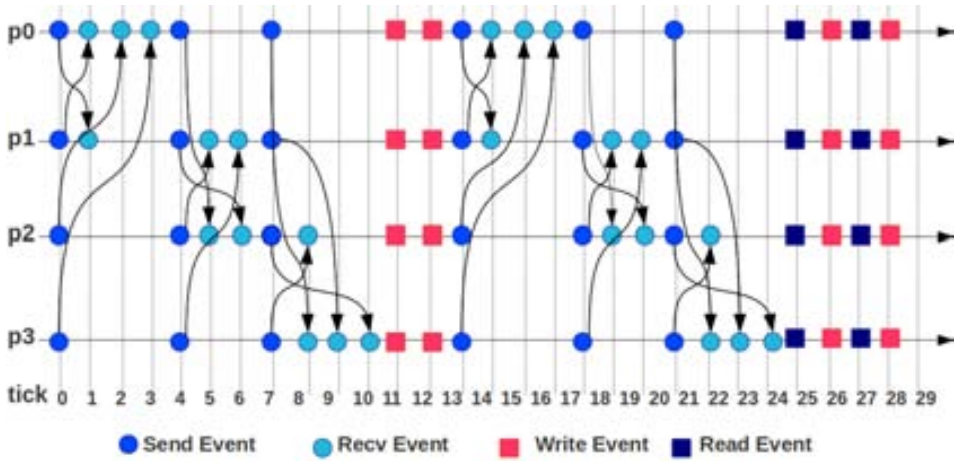


Figura 3.2: Traza lógica

2. Cargar la Librería de libpas2p-io.so.

   Para aplicaciones escritas en fortran usar:

   ```
   export LD_PRELOAD=libfmpich.so:$HOME/libpas2p/bin/libpas2p-io.so
   ```

   Y para aplicaciones que no están escritas en fortran usar:

   ```
   export LD_PRELOAD=$HOME/libpas2p/bin/libpas2p-io.so
   ```

3. Ejecutar la aplicación normalmente. La aplicación debe estar compilada con la misma distribución MPI que la *libpas2p-io.c*.

4. Durante la ejecución se generan tantos ficheros como procesos tiene la aplicación. Los ficheros de trazas son nombrados con *num.tmp_pas2p_2* donde *num* corresponde al *rank* del proceso paralelo. Los archivos de trazas son almacenados por defecto en el mismo lugar donde la aplicación es ejecutada.

3.1.2. Procesamiento de trazas y Extracción de los patrones de acceso

El proceso seguido para la extracción de los patrones de accesos local sigue los siguientes pasos:

1. Se identifican los diferentes ficheros usados por cada proceso de la aplicación paralela y se genera un fichero con las operaciones de E/S por cada fichero. El algoritmo 3.1 es usado para la extracción de las operaciones de E/S.

Algoritmo 3.1: Extracting the IOPs by file for each process of the application.

```
Algorithm Extractiong I/O operations
/* Applied to each traces file of each process of the
application */
   Input: Traces File of application
   Output: An I/O operations file for each file
   indentified by each process

Extracting_IOP(Number Processes)

 for each process of the application
  Extracting of the Name and the ID of Files
 en for

 for each ID of File of the processes
   Extract the different I/O operations
   /*There are 48 IOP defined*/
 end for
end Extracting_IOP
```

2. Determinación de offset explícitos, desplazamiento (*displacement*) y distancia (*distance*) para las operaciones de lectura y escritura.

   La estructura de datos usada para este proceso es la siguiente:

```
typedef struct record_iop

   long long offset;
   long long sizeDatatype;
   int sizeRequest;
   int tick;
   double time;
   double duration;
   int countDatatype;
   int distance;
   long long disp;
```

```
  line_iop;
line_iop write_read;
```

Para identificar los patrones de E/S espaciales es necesario determinar los offset explícitos de las operaciones de acceso a datos. Para esto las operaciones MPI_File_set_view y MPI_File_set_seek son identificadas y guardadas en una estructura de datos pila para poder usarlas en la determinación del offset.

Para las operaciones que usan punteros es necesario identificar las operaciones de vistas y seek para conocer el offset de las operaciones de escritura y lectura. Aunque se debe tener en cuenta que las operaciones MPI_File_set_view afectan también a las operaciones con offsets explicitos.

La distancia entre las operaciones de E/S se calcula por la diferencia de *ticks* y el desplazamiento con la diferencia de offsets ambas diferencias es entre dos operaciones de E/S consecutivas.

El algoritmo 3.2 *Updating the I/O operations* describe los pasos para realizar el calculo de offset, el desplazamiento y la distancia. Se puede observar que se utiliza una estructura de datos pila para las operaciones de vistas y seek para determinar los nuevos offset de las operaciones de lectura y escritura.

Algoritmo 3.2: Updating the I/O operations.

```
Algorithm Updating the I/O operations
[Input: Files of I/O operations of the processes
 of the application]
[Output: Files of I/O operations of the processes
    of the application with new offset]
for each record of files of I/O operations
   if (operation == MPI_File_set_view) then
      push(operation, stack_view)
  endif
  if (operation == MPI_File_set_seek) then
      push(operation, stack_seek)
  endif
  if (operation == read or write) then
    if (not_empty(stack_seek)) then
      offset_seek = pop(stack_seek)
      write_read.offset = offset_seek
```

```
    else
        if (not_empty(stack_view)) then
          displacement_view = pop(stack_view)
          write_read.offset= write_read.offset + disp_view
        endif
      endif
    endif
   write_read.displacement = write_read.offset(actual)
   - write_read.offset(actual-1)
   write_read.distance = write_read.tick(actual)
   - write_read.tick(actual-1)
   Add write_read record to files of I/O operations
  endfor
endAlgorithm
```

La pila (*stack*) registra cada operación MPI_File_set_view o MPI_File_seek y al momento de evaluar las operaciones de escritura y lectura el *offset* es calculado con la ecuación 3.1 si la operación inmediata anterior a una lectura o escritura fue MPI_File_seek y es calculado con la ecuación 3.2 si fue MPI_File_set_view.

$$offset\_operacion = offset\_seek \tag{3.1}$$

$$offset\_operacion = offset\_operacion + displacement\_view \tag{3.2}$$

Los archivos generados en este paso son utilizados para la identificación de patrones de E/S locales.

3. Extracción de Patrones

   Para obtener el patrón de acceso local temporal de cada operación de calcula la diferencia de *ticks* entre las operaciones que se denomina distancia. Esto nos permite determinar la cantidad eventos de comunicación entre dos operaciones de E/S. A partir de este proceso se obtienen las operaciones de E/S separadas por tipo con su offset real y su distancia.

   A partir de los archivos generados en el proceso anterior se identifican los patrones de acceso locales. Se realizan dos recorridos en los ficheros uno donde se identifican los patrones simples consecutivos exactos y otro donde se identifican los patrones similares que estan compuestos por varios patrones simples.

Un operación de E/S formará parte de un patrón simple si tiene el mismo tamano de request, tamano del tipo de dato, distancia y desplazamiento; y se encuentran en un orden consecutivo. A partir de esto se obtiene la cantidad de repeticiones de un patrón simple denominado *iterSep*. Y este patrón simple es considerado una operación de E/S con el nuevo atributo *iterSep* para la identificación del patrón de acceso local (LAP).

Un patrón compuesto esta formado por patrones simples similares. Una operación de E/S es considerada como parte de un patrón si tienen operaciones de E/S similares y orden similar.

Teniendo en cuenta lo anterior un LAP esta representado por *tipo de operación, iteración simple,tamaño de request, tamaño del tipo de dato, distancia, desplazamiento, iteración global y tick*. La estructura de datos usada para los LAP es la siguiente:

```
typedef struct record_pattern

  int iterSeq;
  int requestSize;
  long long sizeDatatype;
  int distance;
  long long disp;
  int iterGlobal;
  int tick;
line_pattern;
```

Donde *iterSeq* es la repetición del patrón simple después del primer recorrido y *iterGlobal* es la cantidad de veces que se repite el patrón después del segundo recorrido. El algoritmo para identificar un patrón se basa en la igualdad del tipo de operación y la similitud del *request size*, *displacement*, y la *distance*. La distancia es el número de *ticks* entre dos operaciones consecutivas y el desplazamiento es la diferencia del offset de dos operaciones consecutivas. La similitud de un patrón esta dada por la relación expresada en las ecuaciones 3.3,3.4 y 3.5 para el tamano de request, distancia y desplazamiento.

$$Request\_Similar = \frac{requestSize\_Patron}{requestSize\_NuevoEvento} \tag{3.3}$$

$$Displacement\_Similar = \frac{displacement\_Patron}{displacement\_NuevoEvento} \tag{3.4}$$

$$Distance\_Similar = \frac{distance\_Patron}{distance\_NuevoEvento} \tag{3.5}$$

Si el resultado de las ecuaciones 3.3,3.4 y 3.5 es $>$ 0,80 y menor que $<$ 1,1 se considera al nuevo evento como una nueva ocurrencia del patrón que se está analizando. Al finalizar este proceso se tiene un fichero por cada proceso de la aplicación con sus patrones locales.

Para obtener el patrón de acceso global se analizan los patrones obtenidos en el paso anterior. Un patron global espacial esta compuesto por LAP similares. Para que un LAP se considere similar se aplican las ecuaciones 3.3, 3.4 y 3.5 y la misma relación establecida para la identificación de los LAPs. Para el patrón de acceso global se considera la similitud entre los ticks de los diferentes LAPs para todos los procesos paralelos y se aplica la ecuación 3.6. Se asigna a *tick_Patron* el tick del LAP del primer proceso de la fase y se evalúa su similitud con los otros procesos de la fase, si la ecuación 3.6 es $<$ 1,2 y $>$ 0,8 entonces el LAP pasa a formar parte del patrón global, este proceso se aplica a todos los procesos con LAPs similares.

$$tick\_Similar = \frac{tick\_Patron}{tick\_NuevoEvento} \tag{3.6}$$

El patrón global espacial esta representado por el offset, desplamiento, y el tamano del size. El patrón global temporal es representado por el *tick* y el patrón de acceso local (LAP). Debido a que los LAP fueron ordenados por los *ticks* también se obtiene la temporalidad de los LAP en cada proceso y el orden de los LAP para los np procesos.

3.1.3. Identificación de fases de E/S y Definición del modelo de E/S

Para la definición del modelo se usa el concepto de fase y peso de E/S. Una fase de E/S está compuesta por un conjunto de procesos que tienen similares LAP y ticks. El modelo de E/S solo considera los eventos de E/S considerando los eventos de comunicación únicamente para obtener la frecuencia de las operaciones de manipulación de ficheros. Una fase de E/S ésta compuesta por:

- Identificador de Fase (IdF)
- Identificadores de los Procesos que forman parte de la fase ($IdsP$)

- Número de procesos que forman la fase (np)
- request size (rs), tipo de operación (w/r), distancia ($dist$), desplamiento ($disp$) y $tick$
- Número de repeticiones (rep), número de operaciones de E/S ($\#iop$) y peso ($weight$).

La Figura 3.3 muestra los elementos que componen una fase y el patrón temporal y espacial.

Figura 3.3: Componentes de Fases de E/S

Una fase es parte del patrón global, donde los LAP son similares si tiene valores similares para un número procesos excepto para el offset inicial ya que cada proceso suele trabajar en posiciones diferentes del fichero para aprovechar la ventaja de la E/S paralela.

El peso de una fase depende del número de procesos, request size del LAP y el núpmero repeticiones de cada LAP que es parte de la fase. Éste se expresa en MB/seg. para determinar la cantidad de datos que cada fase utilizará. La importancia de una fase está dada por su peso. Donde el peso se obtiene aplicando la ecuación 3.7.

$$Peso(fase[i]) = np * rs * rep \tag{3.7}$$

Donde np es el número de procesos de la $fase[i]$, rs el tamano de la operación y rep el número de repeticiones del patrón. El modelo depende de la fase y el peso, permitiendo conocer Cuando y Cómo el sistema de E/S será utilizado.

La metainformación que forma parte del modelo de E/S es la siguiente :

- Número de fichero usados por la aplicación paralela
- Mode acceso: Strided, Secuencial o Segmentado y Aleatorio.
- Tipo de Acceso: Unico (un fichero para cada proceso) o Compartido (un fichero compartido por n procesos)
- Información del tipo de operación relacionado con el modo de acceso. Esta información esta relacionada con el tipo de posicionamiento (offset explícito,

individual file pointer, shared file pointer), sincronismo (blocking y non-blocking) y coordinación (collective y non-collective).

Además, se adjunta al modelo una tabla con la decripción de las fases, donde se detalla el identificador de fase, el peso, request size y el número de procesos.

3.2. Ejemplos de Modelos de E/S

Para mostrar la aplicación de la metodología en esta sección diferentes modelos para diferentes benchmarks de E/S. Se parte NAS BT-IO que tiene un modelo de I/O con fases simples sobre un fichero compartido, se sigue con MADBench2 que presenta varias fases donde la tercera fase es un fase compleja y finalmente se presenta el modelo de FLASH-IO que tiene varias fases de I/O en tres archivos.

Se analizarán siguiendo la metodología: Recopilacion de Trazas de la aplicacion paralela, Procesamiento de trazas y Extraccion de los patrones de acceso y finalmente la identicacion de fases de E/S y definición del modelo de E/S.

3.2.1. NAS BT-IO

El NAS BT-IO [39], es una extensión del benchmark NAS BT, simula los requisitos de E/S de BT. BTIO presenta un patrón de particionado tridiagonal en una arreglo de tres dimensiones en un número cuadrado de procesos. Cada proceso es responsable de varios subconjuntos cartesianos del conjunto de datos completo, cuyo número se incrementa como la raíz cuadrada de los procesos participantes en el cómputo. En BTIO, 40 arreglos son escritos consecutivamente a un archivo compartido agregando uno después de otro. Cada arreglo debe ser escrito en un formato canónico y por filas en el archivo. Los cuarentas arreglos son leídos nuevamente para verificación usando el mismo particionado de datos. El tamano del archivo es fijo, sin tener en cuenta el número de procesos. Por lo tanto la cantidad de E/S se irá decrementando para cada proceso a medida que se aumenten los procesos. El patrón de acceso al archivo de cada proceso es discontiguo con *strides* variables. El patrón de acceso del NAS BT-IO es típico de las aplicaciones científicas, por esta razón es uno de los benchmarks más usado [40, 11, 41] en la evaluación de las prestaciones de la E/S.

Para mostrar el proceso de extracción del modelo para NAS BT-IO se partirá de un ejemplo para $np = 4$ procesos. La Figura 3.4 se obtiene después de procesamiento de los archivos de trazas y esta muestra un ejemplo del formato del patrón de acceso. Cada línea en la Figura 3.4 se obtiene teniendo en cuenta los patrones y los ticks. Por ejemplo, la

primera línea en negrita significa que el proceso 0 ha realizado 40 operaciones de escritura con el mismo request size (10612080 bytes), displacement y offset inicial 0 en su vista del fichero. La segunda linea para el proceso 0 muestra 40 lecturas con el mismo patrón que las escrituras. Se puede observar este comportamiento para los cuatro procesos de la aplicación.

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
0	**1**	**MPI_File_write_at_all**	**40**	**10612080**	**265302**	**0**
0	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
1	**1**	**MPI_File_write_at_all**	**40**	**10612080**	**265302**	**0**
1	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
2	1	MPI_File_write_at_all	40	10612080	265302	0
2	1	MPI_File_read_at_all	40	10612080	265302	0

IdP	IdF	MPI-Operation	Rep	RequestSize	Disp	OffsetInit
3	1	MPI_File_write_at_all	40	10612080	265302	0
3	1	MPI_File_read_at_all	40	10612080	265302	0

Figura 3.4: Ejemplo de patrón (LAP)

La Figura 3.5 muestra las fases para el ejemplo. La fase 1 esta compuesta por $np = 4$ procesos con similares patrones de accesos. El peso de esta fase 1 es igual a 40MB ($np * rs$ donde $np = 4$ y $rs = 10MB$), la fase 2 es similar y se produce pasados 122 ticks (o sea 122 eventos mpi). Esta distancia de 122 tick es lo que lleva a considerar 40 fase de E/S ya que la carga al sistema de E/S será de 40MB cada 122 tick y no una ráfaga de $40 * 40MB$ para el sistema de E/S.

Phase 1

IdP	IdF	MPI-Operation	Offset	tick	RequestSize
0	1	MPI_File_write_at_all	0	**148**	10612080
1	1	MPI_File_write_at_all	0	**147**	10612080
2	1	MPI_File_write_at_all	0	**147**	10612080
3	1	MPI_File_write_at_all	0	**147**	10612080

Phase 2

IdP	IdF	MPI-Operation	Offset	tick	RequestSize
0	1	MPI_File_write_at_all	265302	**269**	10612080
1	1	MPI_File_write_at_all	265302	**268**	10612080
2	1	MPI_File_write_at_all	265302	**268**	10612080
3	1	MPI_File_write_at_all	265302	**268**	10612080

...

Figura 3.5: Fases de E/S para el ejemplo (*phases*)

Las Figura 3.6 y 3.7 muestran el patrón global para el ejemplo donde el patrón global se muestra a través del patrón espacial tanto local como global (Figure 3.6). También se

muestra en un gráfico de tres dimensiones el patrón temporal (Figura 3.7), tanto local como global donde el *file Offset* indica la posición donde el proceso p esta accediendo en el tick t.

Además es necesario tener la siguiente metainformación:

- Explicit offset, Blocking I/O operations, Collective operations.
- Modo de acceso es Strided y el tipo de acceso es Shared.
- MPI_Set_view con tamano de etype de 40bytes y tamano de filetype igual a 40MB.

La tabla 3.2 muestra una descripción de las fases para completar el modelo de E/S.

Tabla 3.2: Descripción de las fases de E/S

IdPh	*rep*	#Phase	*weight*	*np*
1 to 40	1	40	40MB	4
41	40	1	6400MB	4

En la figura 3.7 se puede observar a los primeros 4 puntos rojos para los cuatro procesos que representan la fase 1, los segundos puntos rojos representa la fase 2 y así sucesivamente hasta la fase 40, estas fases se ven como una linea roja casi horizontal. Lo 40 puntos azules para los cuatro procesos representan la fase 41, que se observa como una línea vertical. Se ha considerado a las operaciones lectura como una única fase debido a que no hay otros eventos MPI entre las operaciones por esta razón la fase 41 se ve como una linea azul (tick consecutivos).

La meta información analizada permite determinar que se accede a un fichero, donde el modo de acceso es Strided y el tipo de acceso es Shared. El acceso Strided se consigue con el uso de vistas de MPI. Por lo tanto, cada offset es respecto a la vista de cada proceso. En este ejemplo existe un único fichero que es accedido por todos los procesos de la aplicación.

A continuación se muestra el modelo de E/S para 16 y 36 procesos para las clases C y D respectivamente. Esto también permite demostrar que el modelo de la aplicación es independiente del sistema de E/S y que una vez obtenido se lo pueda usar en otro sistema de E/S: La metainformación se obtiene fue (igual que la que se explico en el ejemplo de 4 procesos):

- La aplicación trabaja sobre un archivo
- Explicit offset, Blocking I/O operations, Collective operations.
- Modeo de acceso Strided y tipo de de acceso Shared.

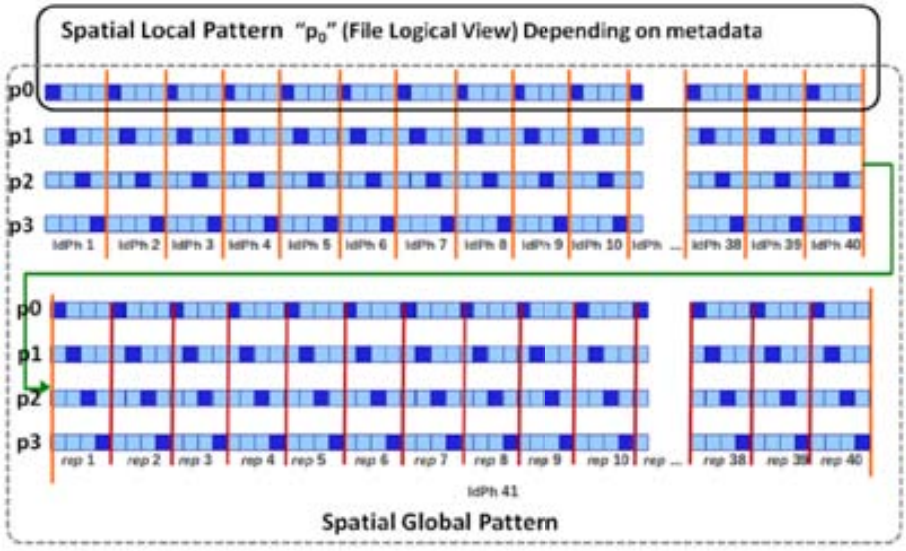


Figura 3.6: Patrones de Acceso Espacial para los 4 procesos

- MPI_Set_view con etype of 40 y Request size de 10MB.

La Figura 3.8 muestra las fases para 16 procesos, para la clase C en su subtipo Full.

También se obtuvo el modelo para clase D 36 procesos. Figure 3.9 muestra el modelo, este es similar para las diferentes clases y número de procesos. La diferencia entre las clases se puede observer en el peso de las fases.

La Tabla 3.3 muestra la descripción de las fases donde $IdPh$ es el Identificador de fase, rs es el request size, np el número de procesos de la fase $IdPh$, idP es el rank del proceso MPI. Este modelo ha sido obtenido también para 64 y 121 y se ha observado un comportamiento similar.

Tabla 3.3: Descripción de fases de NAS BT-IO subtipo FULL para np procesos

IdPh	Operación	Offset inicial	rep
Class C			
1-40	np W in each phase	$rs * idP + (rs * (IdPh - 1) + + (rs * (np - 1)) * (IdPh - 1))$	1
41	np R	$rs * idP + (rs * (rep - 1) + + (rs * (np - 1)) * (rep - 1))$	40
Class D			
1-50	np W in each phase	$rs * idP + (rs * (IdPh - 1) + + (rs * (np - 1)) * (IdPh - 1))$	1
51	np R	$rs * idP + (rs * (rep - 1) + + (rs * (np - 1)) * (rep - 1))$	50

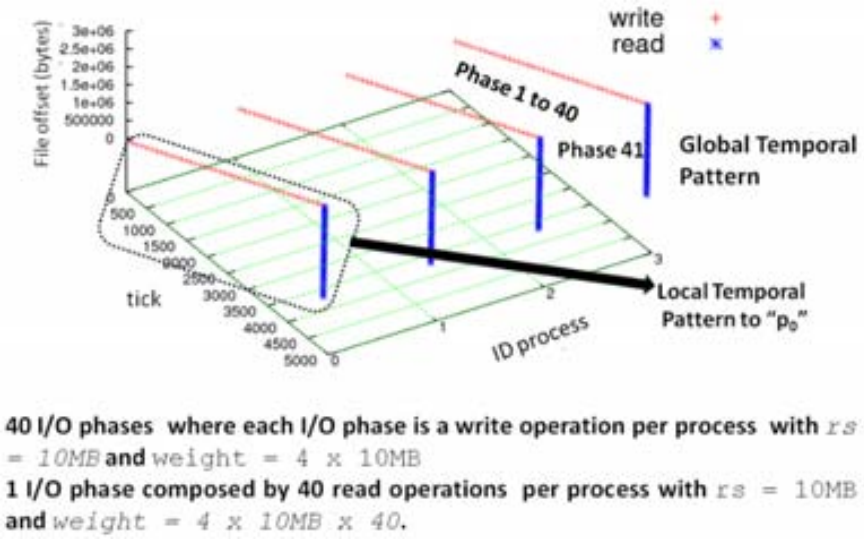


Figura 3.7: Patrones de Acceso Temporal para los 4 procesos

3.2.2. MADBench2

MADBench2 es un derivado del código de análisis de datos MADspec. El MADspec estima el espectro de energía regular de la microondas de la radiación cósmica en el cielo de un conjunto de datos pixelados. Como parte de sus cálculos, el MADspec realiza muchas operaciones sobre una matriz *out-of-core* , requeriendo sucesivas escrituras y lecturas de una gran cantidad de datos contiguos de archivos compartidos o individuales. Debido a grandes, contiguas y mezcladas operaciones de lectura y escritura que MADbench2 realiza y su capacidad para probar una variedad de parámetros, éste es elegido como benchmark en la comunidad de la E/S paralela [11, 42]. MADBench2 realiza cuatro pasos

- construye recursivamente una secuencia del polinomio de *Legendre* basado en la matriz componente de la correlación pixel-pixel y la senal CMB (*Cosmic Microwave Background*), escribiendo cada uno a disco.
- Forma e invierte la matriz de componentes senalCMB+ruido (cálculo y comunicación)
- A su vez, lee cada componente de la matriz de correlación senal CMB del disco, multiplicando este por la matriz de correlación de datos CMB inversa, y escribe la matriz resultante a disco.
- A la vez, lee cada par de resultados de la matriz desde disco y calcula el signo de su producto.

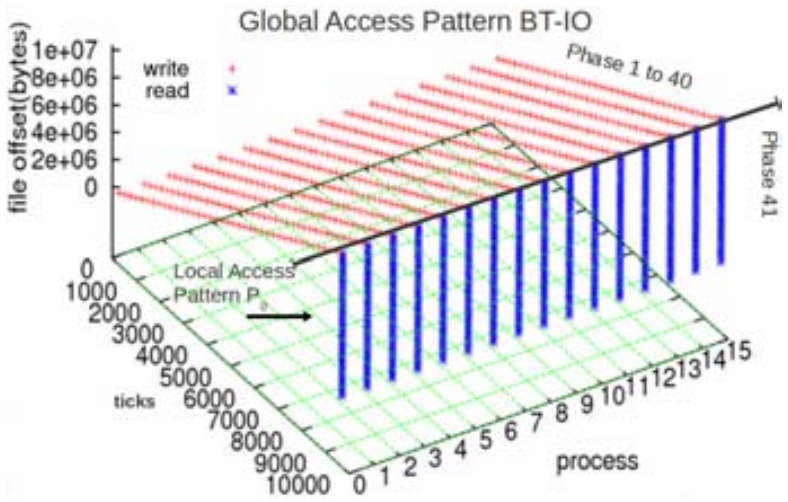


Figura 3.8: Fases de NAS BTIO, clase C, 16 procesos

Debido a la gran cantidad de memoria del dominio computacional en los cálculos reales, todas las matrices requeridas generalmente no entran en memoria simultáneamente. De esta forma, un algoritmo *out-of-core* es utilizado, el cual requiere suficiente memoria como para almacenar solo 5 matrices en cualquier momento.

En la Figura 3.10 se muestra el modelo de comportamiento para el Benchmark MADBench2 donde se puede observar el comportamiento espacial y temporal para la aplicación agrupadas en fases de E/S para 16 procesos.

La metainformación de MADBench2 es:

- Todos los procesos acceden a un único fichero
- Modo de acceso Secuencial y tipo de acceso es compartido,
- Con Individual file pointers, Offset implícito, operaciones Non-collective y Blocking I/O,
- Para el manejo de la posición de lo punteros usa la directiva MPI_File_seek.

A partir de este modelo podemos obtener una forma general del comportamiento espacial en las diferentes fases. En la Tabla 3.4 se muestra la forma general de las cinco fases donde destacamos el offset inicial a partir de donde cada proceso empieza a trabajar el fichero. Donde rs es el tamano de request. Esto permite definir los valores para los diferentes benchamrk que se usaran para evaluar las métricas que interesan de acuerdo al requisito solicitado: prestaciones, disponibilidad, escalabilidad o consumo energético.

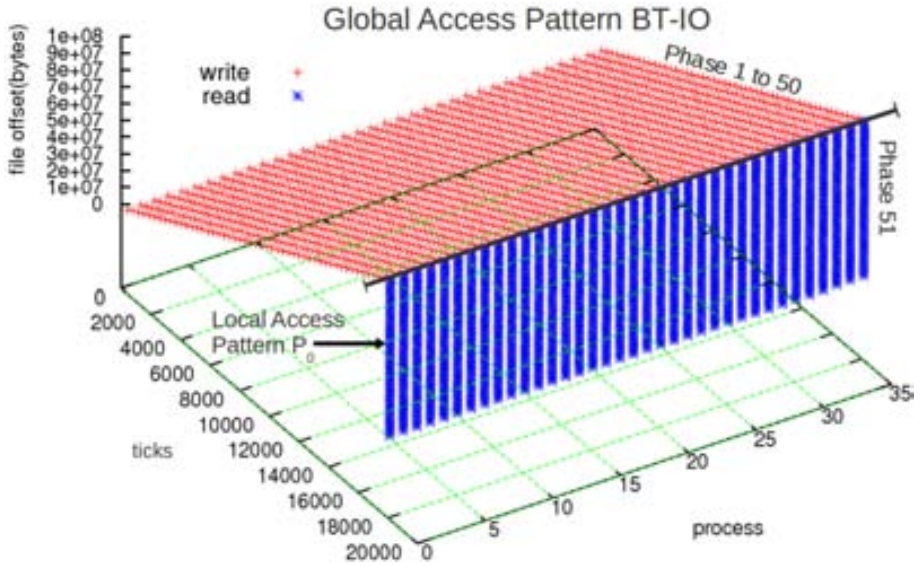


Figura 3.9: Fases de NAS BTIO, clase D, 36 procesos subtipo Full

Tabla 3.4: Desripción de las fases de E/S de MADBench2 para np procesos

IdPh	Numero de Operaciones	Offset Inicial	*rep.*	Peso
1	np write	$IdP * 8 * rs$	8	$np * rs * 8$
2	np read	$IdP * 8 * rs$	2	$np * rs * 2$
3	np write np read	$IdP * 8 * rs$ $IdP * 8 * rs+$ $+2 * rs$	6 6	$np * rs * 6$ $np * rs * 6$
4	np write	$IdP * 8 * rs-$ $-2 * rs$	2	$np * rs * 2$
5	np read	$IdP * 8 * rs$	8	$np * rs * 8$

3.2.3. FLASH-IO Benchmark

El Benchmark FLASH-IO [5] es un kernel de la aplicación FLASH, un código de hidrodinámica de una maya adaptiva de bloque estructurado que resuelve ecuaciones de hidrodinámica reactiva, desarrollado principalmente para estudio de los flashes de los neutrones de las estrellas y las enanas blancas. El dominio computacional es dividido en bloques en un número de procesos. Un bloque es un arreglo de tres dimensiones con cuatro elementos adicionales que hacen de celdas protectoras en cada dimensión para almacenar información de sus vecinos. FLASH IO produce un archivo de checkpoint y dos archivos conteniendo datos centrales y de las esquinas. El archivo de mayor tamano es el de checkpoint, el tiempo de E/S que domina el benchamrks. FLASH IO usa una librería de alto nivel de E/S (HDF5) para almacenar los datos con sus metadatos.

La siguiente metainformación se obtuvo para FLASH-IO en su versión parallel HDF5:

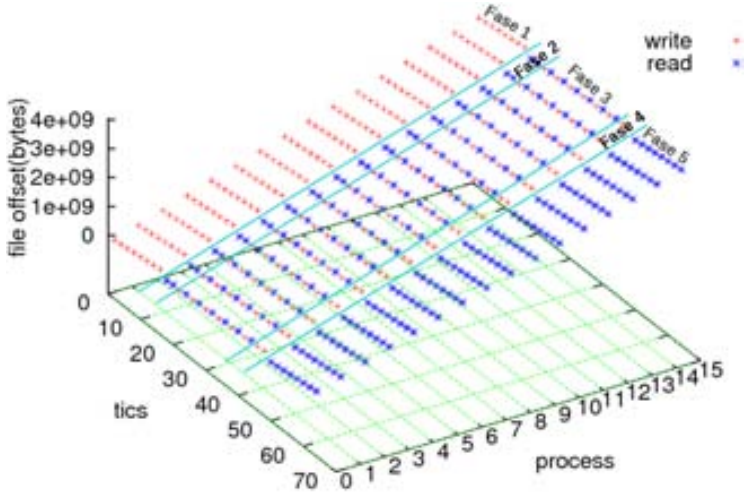


Figura 3.10: Comportamiento temporal y espacial para MADBench2 para 16 procesos, 8KPIX, mode de acceso compartido y tipo de acceso secuencial

- Todos los procesos acceden a 3 ficheros
- Explicit offset, operaciones Blocking, Collective y Non-collective.
- Modo de acceso Strided y tipo de acceso Shared en tres ficheros usados por todos los procesos de la aplicación.
- MPI_Set_view con diferentes etypes y filetype para las operaciones collective y non-collective.

FLASH-IO realiza únicamente operaciones de escrituras a través de HDF5 paralelo que son convertidas a operaciones de MPI-IO. La Figura 3.11 muestra las fases de FLASH-IO para los tres ficheros. Podemos observar que los tres ficheros son accedidos en forma consecutiva. Para el análisis se han enumerado teniendo en cuenta su orden de apertura durante la ejecución de la aplicación. En esta caso hay dos tipos de operaciones usadas MPI_File_write_at and MPI_File_write_at_all. Debido a que las operaciones collectivas (MPI_File_write_at_all) representan más del 90 % de la E/S para el análisis sólo mostramos la fases para este tipo de operaciones.

La Tabla 3.5 muestra la descripción de las fases del primer fichero.

Donde **IdPh** es el identificador de la fase, *rs* es el tamano del request, *rep* son las repeticiones, *dist.* es la distancia, y *disp.* es el desplazamiento. Podemos observar que hay 5 fase sdonde la quinta fase es la de mayor peso, y esta realiza 24 operaciones de 2MB que son realizadas por todos los procesos.

Ls Tabla 3.6 muestra la descripción para el segundo fichero podemos observar también cinco fases donde la quinta fase es la de mayor pero pero en este caso son solo 4 operaciones realizadas por todos los procesos de la aplicación.

Tabla 3.5: Desripción de fases de FLASH-IO para Write_at_all - Primer Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	1920	2	3	124392	122KB
4	64	write	3840	1	3	439012	245KB
5	64	write	2621440	24	3	169869312	4GB

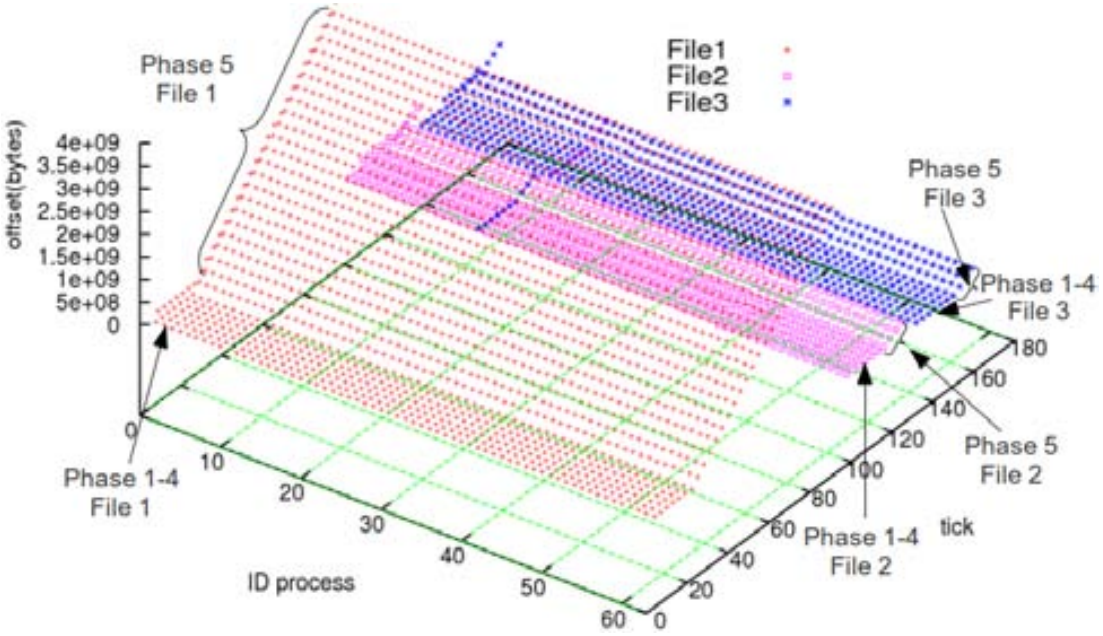


Figura 3.11: Fases de E/S de FLASH-IO para 64 procesos

La Tabla 3.7 muestra las fases para el tercer fichero que tiene un comportamiento similar a las fases del segundo fichero con diferencia en el peso de la quinta fase.

En este caso las fases se ordenan de la siguiente forma Id_File.Id_Fase: 1.1, 1.2, 1.3, 1.4. 1.5, 2.1, 2.2, 2.3, 2.4, 2.5,3.1.3.2,3.3,3.4 y 3.5. Por lo tanto, FLASH-IO tiene 15 fases de E/S.

3.3. Conclusiones

Una metodología para caracterizar la E/S de aplicaciones científicas paralelas ha sido propuesta y aplicada. Se ha definido un modelo de E/S por medio tres características: metadatos, patrón global espacial y patrón global temporal. Para obtenerlo se captura la información relevante de E/S por medio de una librería que traza todas la primitivas de MPI-IO. Este procedimiento no necesita la intrumentación del código fuente debido a

Tabla 3.6: Desripción de fases de FLASH-IO para Write_at_all - Segundo Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	128KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1310720	4	3	84934656	335MB

Tabla 3.7: Desripción de fases de FLASH-IO para Write_at_all - Tercer Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	122KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1572160	4	3	101974016	402MB

que usa el concepto de interposición de funciones. Se ha aplicado la metodología a NAS BT-IO, MadBench2 y FLASH-IO. Este proceso de caracterización se ha definido para dar soporte a la metodología presentada en el capítulo 5.

Capítulo 4

Metodología para la caracterización del sistema de Entrada/Salida

Los sistema de E/S de los computadores de altas prestaciones se pueden ver desde el punto de vista de la arquitectura como de la pila se software. La pila de software provee lo necesario para que la manipulación de las operaciones de E/S sean transparente al usuario. El hardware de E/S se representa como una arquitectura a la que se puede acceder por medio de la pila de software. La Figura 4.1 muestra el esquema del sistema de E/S.

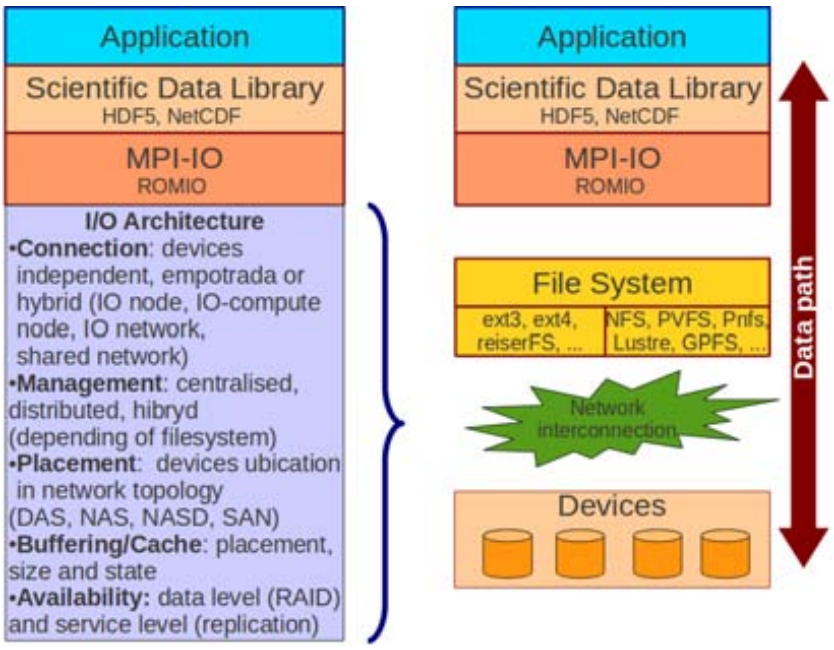


Figura 4.1: Sistema de E/S desde el punto de vista de la arquitectura y del camino de E/S

Los sistemas de E/S de los clúster de computadores actuales por lo general tienen distintas configuraciones que llevan a un proceso de evaluación de prestaciones complejo. Por

eso se propone un proceso que da orden a la evaluación considerando tanto la arquitectura como la pila de software del sistema de E/S.

Los clúster de computadores normalmente ofrecen una pila de software para la gestión de las operaciones de E/S para los diferentes niveles del sistema. Los sistemas de ficheros compartidos como NFS y sistemas de ficheros paralelos como LUSTRE, GPFS, PVFS2 y sistemas distribuidos como GLUSTER o Hadoop para aquellas aplicaciones que requieran E/S. También hay diferentes librerías de E/S como HDF5 y NetCDF, a su vez existen diversas distribuciones de MPI. Cuando el usuario accede a un clúster suele enviar su trabajo al sistema de colas y no considera el sistema de E/S que su aplicación usará. En este capítulo se presenta un proceso de caracterización del sistema de E/S que ayuda tanto al usuario como al administrador a conocer la estructura del sistema de E/S y evaluar las características prestacionales.

4.1. Metodología Propuesta

Se analiza el hardware y el software del sistema de E/S para identificar las diferentes configuraciones de E/S que existen en el computador de altas prestaciones. Para este proceso el sistema de E/S es estructurado (Figura 4.1) en un esquema jerárquico para dar un orden al proceso de caracterización. Este es analizado desde el punto de vista de la arquitectura de E/S y del camino de E/S de los datos sobre un clúster.

4.1.1. Identificación de Configuraciones de E/S

En este paso se identifican las diferentes configuraciones de E/S que existen en el clúster. Para esto se consideran los siguientes componentes del sistema de E/S: tipo de sistema de fichero (local, distribuido y paralelo), tipo y número de redes de interconexión (uso dedicado o compartido con el cómputo), número y ubicación de nodos de E/S , tipo de dispositivos (RAID, JBOD, SSD).

Para obtener esta información en un clúster linux se puede usar estas tres opciones:

- *df -Th*: esto se utiliza para tener la lista de todos los sistemas de ficheros, tipo de sistemas, además de su capacidad y
- *mount* y *cat /etc/fstab*: Este brinda también la información anterior y otros características más específicas de los diferentes sistemas de ficheros del sistema.

A partir los comandos anteriores se obtendrá la información de los diferentes sistemas montados, capacidad, porcentaje utilizado, permisos habilitados y punto de montaje. Esto

es muy útil para que el usuario vea en dónde están montados los diferentes sistemas y de esa forma elegir donde trabajar.

En el caso de disponer de sistemas de fichero paralelos. Si el sistema dispone de Lustre se puede obtener la información de la cantidad de MDS (Metadata Server), MST (Metadata Target), OSS (Object Storage Server), OST (Object Storage Target). La cantidad de OST en el sistema se puede obtener con el comando *lfs df -h* siendo usuario del sistema. Para ver la cantidad de OSS y MDS es necesario ser usuario con permisos de administrador o consultar la estructura del sistema de fichero paralelo en la documentación técnica que ofrecen los clúster de computadores. Para conocer el stripe por configurado por defecto en Lustre se usa *lfs getstripe -f /mnt/lustre/file*.

Si el sistema paralelo que ofrece el clúster es un PVFS2 o su versión OrangeFS se puede ver la configuración con *cat /etc/pvfs2-fs.conf*. De esta forma se obtiene el número de MetaDataServer y DataServer que se están usando. Para ver el stripe configurado por defecto en PVFS2 se usa *pvfs2-viewdist -f /mnt/pvfs2/file*.

Para ficheros paralelos es importante conocer el tamano de stripe ya que este es útil para entender problemas de prestaciones en la E/S de las aplicaciones.

Al final de este proceso se debería disponer de una tabla de configuraciones del sistema de E/S con la siguiente información: Librerías de E/S, Red de comunicación, Red de almacenamiento, Sistema de fichero global, Nodos de E/S, Metadata Server , Sistema de Fichero Local, Nivel de redundancia, Número de dispositivos, Capacidades de almacenamiento, Punto de montaje, tamano de stripe.

4.1.2. Evaluación de las características prestacionales

El sistema de E/S es evaluado en tres niveles: Librería de E/S, Nodo de E/S (global filesystem) y dispositivos (local filesystem). Se caracteriza el ancho de banda (bandwidth BW), la latencia (latency l) y la cantidad de operaciones de E/S por segundo ($IOPs$) para cada nivel como se puede observar en la Figura 4.2.

La Figura 4.3 muestra Qué y Con qué se puede obtener esta información para los tres niveles. A partir de esta evaluación se obtiene una tabla de prestaciones para diferentes patrones de acceso y para las diferentes configuraciones identificadas y por cada nivel de E/S. Este proceso es aplicable en clúster pequenos y medianos (entre 64 y 512 nodos de cómputo). Hay que tener en cuenta que un proceso de evaluación de prestaciones implica una sobrecarga en el sistema de E/S que puede afectar a los usuarios del sistema por lo tanto este proceso se debe hacer en forma exclusiva o en horarios donde el sistema no esta siendo utilizado.

Del estudio realizado de las diferentes herramientas linux y benchmarks para la evalu-

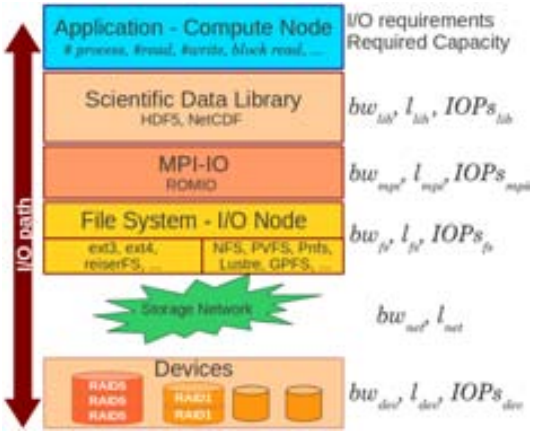


Figura 4.2: Caracterización del Sistema de E/S

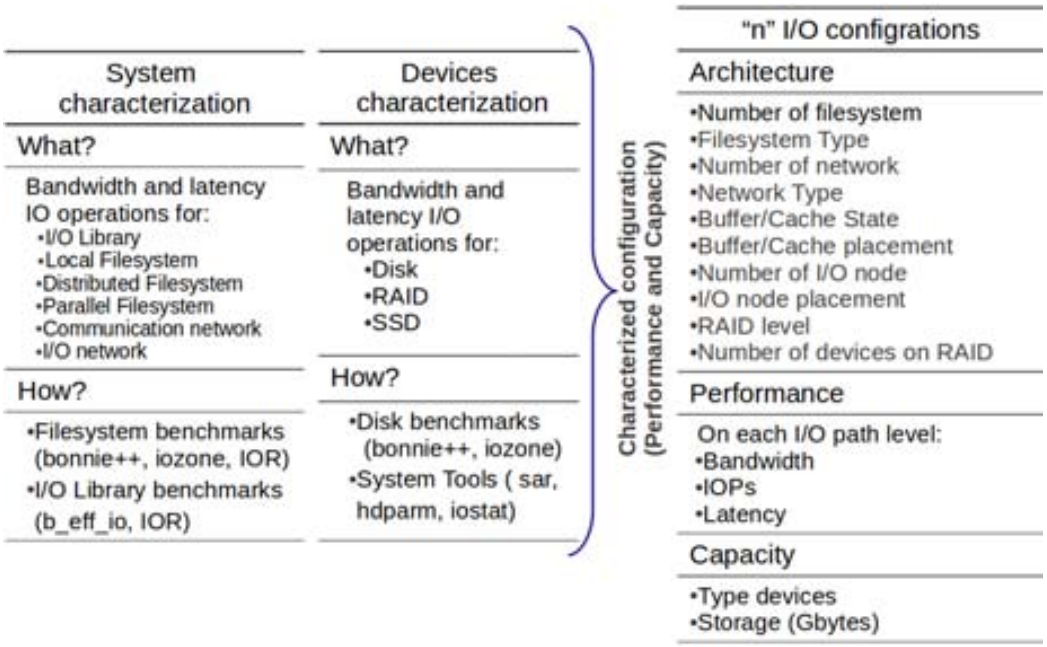


Figura 4.3: Caracterización para los niveles del sistema de E/S

ación de prestaciones se ha seleccionada IOzone [2] para evaluar las prestaciones a nivel de sistema de fichero local y el sistema de fichero compartido NFS.

Una alternativa para ver las prestaciones de los dispositivos se ha utilizado *hdparm -Tt /dev/sda* para obtener rápidamente la velocidad de cada dispositivo, este se puede usar en discos simples aunque en dispositivos RAID esto no es posible. Este requiere tener permisos de administrador. Para los casos en los que no dispone de permiso de administrador se uso IOzone a nivel de sistema de fichero local. Para el caso de dispositivos como RAID, JBOD y SSD se utilizo IOzone a nivel de fichero local.

En el caso del sistema de fichero paralelos se ha elegido el benchmark IOR [3], uno de los benchmark más usados en cómputo de altas prestaciones para evaluar E/S. Este nos permite evaluar para cuatro tipos de librerías de E/S: POSIX-IO, HDF5 paralelo, Parallel NetCDF y MPI-IO.

Debido a que los buffer y cache pueden afectar las prestaciones que se quieren evaluar para el almacenamiento para evitar su efecto lo que se ha hecho es configurar todos los benchmark para que escriban y lean de un fichero que tiene el doble del tamano de la memoria RAM del nodo donde se ejecuta el benchmark. El proceso de caracterización lo podemos resumir en los siguiente pasos.

- Configurar los parámetros de los diferentes benchmark: La Tabla 4.1 muestra los parámetros seleccionados de IOR y los valores que deben asignarse según las características de las configuraciones.

  La Tabla 4.2 muestra los parámetros para IOzone.

- Seleccionar las métricas de salidas para los diferente benchmarks: Las métricas que se pueden obtener son el tiempo de E/S, el número de operaciones, la tasa de transferencia y la latencia. La Tabla 4.3 muestra las salidas típicas para los benchmark de E/S.

- Identificar los dispositivos a monitorizar: En algunas situaciones necesario monitorizar el comportamiento de los benchmark a nivel de dispositivo para observar para qué patrones de E/S el dispositivo esta siendo ocupado a un 100 %. Para monitorizar a nivel de disco se usa *iostat -x -p 1* que nos permite ver la como se están procesando las diferentes peticiones a nivel bloque. Si lo que se quiere es monitorizar la cantidad de datos que se escribe en el disco es conveniente usar *iostat -x -p 60* para realizar la monitorización cada 60 segundos para reflejar la cantidad de datos que se están leyendo y/o escribiendo en el dispositivo que se está analizando.

Tabla 4.1: Paramétros de entrada de IOR

Parámetro	Descripción y valor recomendable
tamano del fichero	El tamano del fichero a escribir es calculado como $NP*s*b$; donde $-s$ $IntegerNumber$ ajusta el número de segmento y $-b$ $Number(k,m,g)$ ajusta una cantidad de bytes contiguos a escribir por un proceso de la aplicación paralela. Esto depende de los patrones que se quieran evaluar. Esto tiene que ver con que tipo de aplicación se ejecuta normalmente en el clúster objeto de la caracterización. Para evitar el efecto del buffer/cache en las operaciones de lectura se debe asegurar que cada proceso escriba y lea el doble de la memoria que le corresponde en cada nodo de cómputo.
rs	$-t$ $Number(k,m,g)$; donde $-t$ ajusta el tamano del request. Este puede estar en bytes, MB y GB
np	Número de procesos que participarán en el test. Para esto se usa $mpirun$ $-n$ NP $-a$ $MPIIO$ que ajusta la API a modo MPIIO para NP procesos. La API puede ser [$POSIX$ $MPIIO$ $HDF5$ $NCMPI$]
Mode de acceso	Modeo de acceso donde el $Default$ para un acceso secuencial, no trabaja para el Strided, $-z$ para el acceso Random. Para que trabaje en modo strided se debe ajustar el tamano de bloque y el request al mismo valor y el segmento al número de strided que presenta el patrón.
Tipo de acceso	Tipo de acceso por $Default$ es Shared , y se usa $-F$ el tipo único
Tipo de operación	Esto permite evaluar las prestaciones para operaciones que usan Colectivas de E/S, el $Default$ trabaja son Collective I/O, con $-c$ se habilita esta opción.

En el caso de sistemas de E/S con sistemas de fichero como Lustre que manejan un conjunto jerárquico de dispositivos lo conveniente es usar *collectL* que permite monitorizar las operaciones de E/S que procesa cada OST. Con *collectl -sndcl -oT* se obtiene información de la CPU, Discos, Red de interconexión de cada OST.

- Determinar los valores picos: Para evaluar la capacidad de prestaciones del sistema de E/S, se utiliza las medidas obtenidas por el benchmark IOzone a nivel de fichero de sistema local o dispositivo. El valor pico $BW_{(PK)}$ es el máximo valor obtenido para los diferentes patrones evaluados por IOzone y se calcula para cada tipo de operación de E/S (lectura, escritura, etc.). Este se calcula con la ecuación 4.1, donde $ION[i]$ es un nodo de E/S y i $0..\#NIO$, $(\#NIO)$ es el número de nodos de E/S y $maxBW(ION_{(i)})$ es el máximo valor obtenido por IOzone.

$$BW_{(PK(ION[i])} = maxBW(ION_{(i)}) \tag{4.1}$$

Esto es aplicado para sistemas tipo NFS o con discos locales. En el caso de los sistema paralelos el análisis es más complejo. En este trabajo se ha tomado

Tabla 4.2: Paramétros de entrada de IOzone

Parámetro	Descripción y valor recomendable
tamano de fichero	Para definir el tamano del fichero se usa $-s$ seguido del tamano que se quiere evaluar.
rs	El tamano de request es puesto por default desde 2KB hasta 64 MB. Si se quiere poner otro valor mínimo se debe usar $-y$ y el tamano de request deseado rs. En sistemas tipo NFS un tamano de request 1K o 2K puede provocar la saturación del sistema de E/S por esta razón se recomienda evaluar el ancho de banda de la red de E/S y la carga del servidor NFS para no provocar un degradación de prestaciones para lo otros usuarios cuando se esta realizando el test.
modo de acceso	Para el modo de acceso secuencial se debe colocar $-i$ 0 $-i$ 1, para el modo strides $-i$ 0 $-i$ 5 y $-j$ # que ajusta el tamano del stride a $(\# * RS)$, ara el modo de acceso random $-i$ 0 $-i$ 2.
tipo de acceso	El tipo de acceso Shared es el por $Default$ y para el tipo único se debe colocar $-F$.

Tabla 4.3: Output for the I/O Benchmarks

Métrica	Notación	Unidad
Tiempo medio de escritura	$T_{io(w)}$	segundos
Tiempo medio de lectura	$T_{io(r)}$	segundos
Media de operaciones de escritura por segundo	$IOPs_w$	*double number*
Media de operaciones de lectura por segundo	$IOPs_r$	*double number*
Tasa de transferencia media para las operaciones de escritura	BW_w	MB/sec
Tasa de transferencia media para las operaciones de lectura	BW_r	MB/sec

un valor pico optimista para los sistemas paralelos que esta representado en la ecuación 4.2, donde el valor pico es la suma de todos los valores máximos de los diferentes nodos de almacenamiento que forman parte del sistema de ficheros paralelo. En este caso se asume que la prestación pico es proporcional al número de nodos de almacenamiento que se tienen, es una alternativa optimista porque no se está considerando la influencia de la red de interconexión ni la gestión de los diferentes nodos.

$$\left(BW_{(PK)} = \sum_{i=0}^{\#NIO} maxBW(ION_{(i)})\right) \tag{4.2}$$

Tabla 4.4: Descripción de los sistemas de E/S denominados A y B

Emenento de E/S	Sistema A	Sistema B
Librería de E/S	mpich2	OpenMPI
Red de comunicación	1 Gbps Ethernet	1 Gbps Ethernet
Red de almacenamiento	1 Gbps Ethernet	1 Gbps Ethernet
Sistema de fichero global	OrangeFS	NFS Ver 3
Nodos de E/S	10	32 DAS and 1 NAS
Metadata Server	1	1
Sistema de Fichero Local	Linux ext3	Linux ext4
Nivel de redundancia	-	RAID 5
Número de dispositivos	11 disks	5 disks
Capacidad	500 GB	1.8 TB
Punto de montaje	/mnt/orengafs	/home

4.2. Experimentación

La metodología propuesta se aplica a dos sistemas de E/S: uno paralelo OrangeFS y un sistema compartido NFS.

4.2.1. Identificación de Configuraciones

El proceso de identificación de la configuración de los sistemas de E/S se presenta en la Tabla 4.4 muestra la configuración de los dos sistemas de E/S.

Es sistema A es compuesto de 14 nodos de cómputo con la siguiente descripción técnica:

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

Es sistema B es compuesto de 32 IBM x3550 nodos de cómputo con la siguiente descripción técnica:

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

4.2.2. Evaluación de las características prestacionales

El benchamrk IOR ha sido configurado para escribir y leer en un arhivo compartido por *np* procesos. Para diferentes tamano de requests. La Figura 4.4 muestra las prestaciones medidas con IOR a nivel de MPI-IO para el sistema A y la Figura 4.5 muestra las prestaciones medidas en el sistema B.

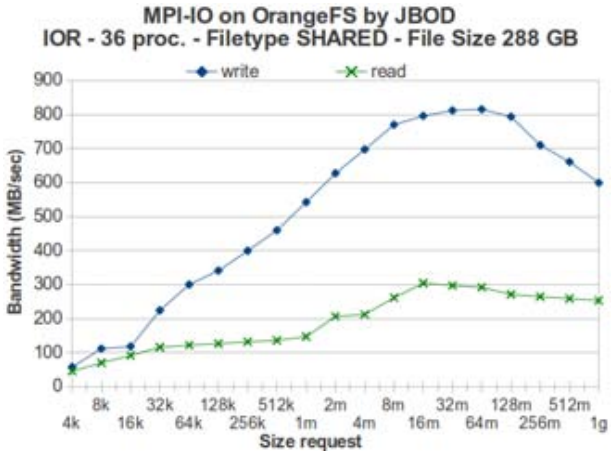


Figura 4.4: Caracterización a nivel de MPI-IO en OrangeFS del Sistema A

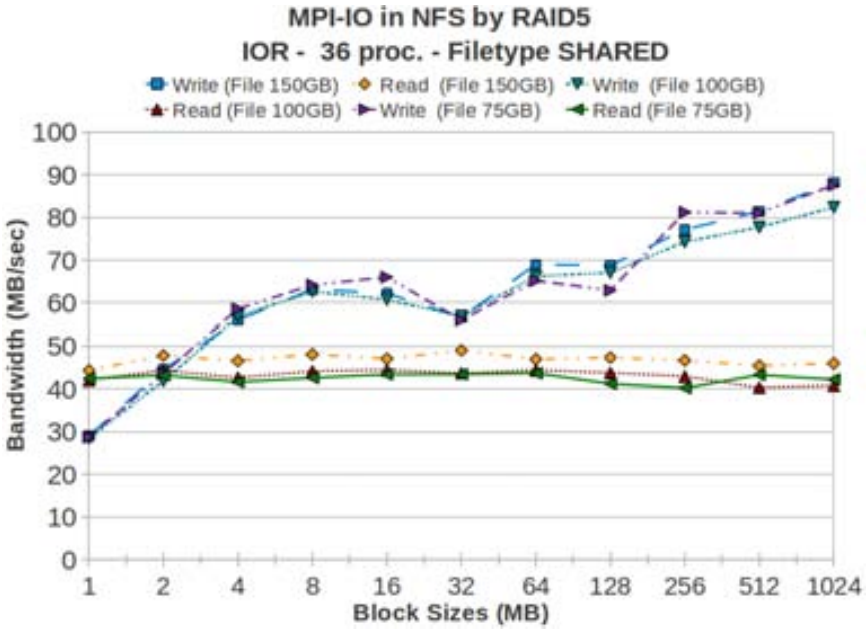


Figura 4.5: Caracterización a nivel de MPI-IO en NFS del Sistema B

En el sistema A (Figure 4.4) se puede observar un incremento en las prestaciones a medida que aumenta el tamano de request para las operaciones de escritura y lectura. Aunque este incremento es mayor para las operaciones de escritura. También se puede observar que a partir de los 64MB las prestaciones caen ligeramente para las lecturas y con mayor diferencia en las escrituras. Claramente en este sistema una aplicación con caracteristicas de E/S similares al usado en este caso se beneficiará si sus fases de E/S con mayor peso son fases compuesta por operaciones de escritura.

En el sistema B (Figure 4.5) se puede observar prestaciones similares para los diferentes tamanos de request en las operaciones de lectura. Esto nos muestra que en este sistema el tamano de request no es un factor de E/S que impacte en las prestaciones. En el caso de las escrituras vemos que las prestaciones se incrementan a medida que el tamano de request se incrementa, excepto para un request de 32 MB donde se puede observar una caída en las prestaciones. Debido a esto en esta caso se ha evaluado para diferente tamanos de fichero y podemos ver que este comportamiento se repite lo que nos indica que es un comportamiento regular en este sistema y se debe a la carga temporal del sistema de E/S. Además en el caso del sistema B la evaluación se ha realizado para request mayores a un MB debido a que tamanos menores provocan un degradación en las prestaciones y llevan un tiempo considerable de evaluación. En muchos de los test los tamanos de request del orden de KB provocan la saturación del servidor NFS.

Los valores picos para los sistemas de E/S son: Write=1120 MB/sec, Read=1260MB/sec para el sistema A y Write=204 MB/sec, Read=300MB/sec para el sistema B.

De esta caracterización se puede observar claramente que las prestaciones del sistema A son superiores que las del sistema B. El sistema B nos es adecuado para aplicaciones intensivas en E/S no solo por el sistema de fichero global sino también por la prestaciones de su red de interconexión. Este sistema tiene una degradación de prestaciones muy severo para request pequenos por lo que una aplicación que escriba o lea en request de KB tenga una penalización muy alta en sus tiempos de E/S. Este es un clúster configurado para servir a aplicaciones que realizan cómputo y comunicación, y E/S temporal en discos locales no esta configurado para aplicación que necesitan realizar E/S paralela en un fichero compartido.

En el caso del sistema A tiene problemas de prestaciones para request size superiores a 512 MB, llevando a una caida prestaciones en el sistema de fichero provocando en varios test la caída de la ejecución del test debido a time out que el propio sistema de fichero impone. El sistema A ésta configurado para aplicaciones que requieran E/S paralela por medio del uso de librerías como ROMIO, parallel HDF5 y parallel NetCDF.

4.3. Conclusiones

Un metodología que da un orden al proceso de caracterizar las prestaciones de los sistemas de E/S ha sido propuesto y aplicado. A partir del esquema jerárquico del sistema de E/S se identifican las configuraciones del sistema de E/S y se realizan las evaluaciones prestaciones para diferentes patrones. El proceso de caracterización prestacional del sistema de E/S es un proceso que consume tiempo y sobrecarga el sistema de cómputos de altas prestaciones. En este capítulo se presento la aplicación de la metodología para dos pequenos clústeres para los sistemas globales: NFS y OrangeFS (PVFS2). Esta metodología ha sido definida para dar soporte a la metodología presentada en el capítulo 5.

Capítulo 5

Metodología para la evaluación de prestaciones del sistema de Entrada/Salida en computadores de altas prestaciones

Los sistemas de E/S son configurados para conseguir prestaciones aceptables para todo el computador de altas prestaciones. Esto es los más conveniente debido a que los sistemas de E/S no son exclusivos y deben proveer servicios a varios tipos de aplicaciones.

Sin embargo, hay aplicaciones que tienen patrones de acceso que se ven perjudicados por las opciones por defecto. En estos casos sería conveniente que el sistema se pudiera ajustar para que la aplicación obtenga mayores prestaciones de acuerdo a sus patrones de acceso.

Los sistemas de E/S de los clústers de computadores actuales por lo general están configurados de distintas formas. La mayoría de los parámetros son configurados a nivel de administrador y algunos también son configurables a nivel de usuario. El usuario en un mismo computador paralelo puede seleccionar distintos recursos de E/S y configurar algunos de sus parámetros. La cantidad de factores a configurar lleva a encontrar sistemas de E/S con varias configuraciones y esto provoca que el proceso de evaluación de prestaciones sea complejo y que lleva tiempo ya que las prestaciones varían dependiendo de la configuración del sistema.

Una información interesante para el usuario sería conocer si el sistema de E/S esta limitando sus prestaciones, cómo saber si se puede obtener mas prestaciones, cómo se puede saber si el problema es que la aplicación no esta usando eficientemente el sistema de E/S. Estas cuestiones no son fáciles de responder. En este capítulo se trata de dar

información necesaria para ayudar a encontrar las respuestas.

En primer lugar, para saber si la E/S es un problema para la aplicación es necesario conocer el sistema de E/S y su capacidad de prestaciones. En segundo lugar, se debe conocer los requerimientos de E/S de la aplicación para determinar si el sistema de E/S puede cumplir con estos. Esto permitirá saber si el sistema de E/S será un factor limitante en las prestaciones que la aplicación pueda obtener.

Por otro lado para lograr una mayor tasa de transferencia en un sistema de E/S es necesario tener la posibilidad de asignar recursos como nodos de E/S, seleccionar dónde se quiere hacer la E/S y poder ajustar el tamano de stripe al más conveniente para el tipo de operación que más influye en las prestaciones de E/S que podría obtener la aplicación. Esto es posible con los sistemas de ficheros paralelos de los clúster de computadores actuales que ofrecen la posibilidad de ajustar el número de nodos de E/S, afinidad de nodos de E/S, y cambio de stripe a nivel de usuario. Estos sistemas ofrecen la posibilidad al usuario de enviar la configuración de estos parámetros cuando envía sus trabajos a cola. La configuración de estos factores permitirán mejorar las prestaciones de E/S.

Dentro de este contexto se definió una metodología para evaluar las prestaciones del sistema de E/S de computadores de altas prestaciones que:

- Caracteriza la E/S de la aplicación por medio de la metodología presentada en el capitulo 3 y del sistema de E/S aplicando la metodología presentada en el capítulo 4
- Analiza la E/S estableciendo una relación entre la aplicación y las características del sistema de E/S. Esta relación se establece en función a la similitud entre el patrón de acceso de las fases del modelo de E/S que se obtiene de la aplicación y los patrones de E/S usados en la etapa de caracterización del sistema de E/S.
- Evalúa la información analizada a partir de la cuál se puede:
  - estimar el tiempo de E/S para la aplicación en diferentes sistemas de E/S y evaluar el error,
  - evaluar el porcentaje de utilización de la capacidad de prestaciones del sistema de E/S para cada fase del modelo de E/S de la aplicación y
  - comparar sistemas de E/S para el modelo de E/S de la aplicación

5.1. Metodología Propuesta

La metodología (Figura 5.1) propuesta consta de tres etapas: Caracterización, Análisis de la E/S y Evaluación.

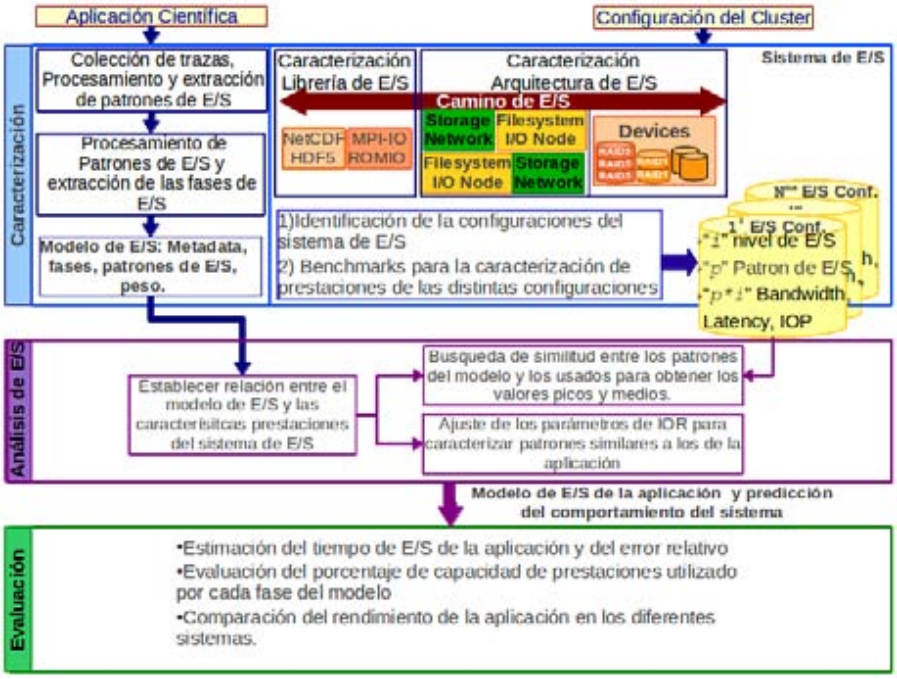


Figura 5.1: Metodología para evaluar las prestaciones del sistema de E/S en clústeres de computadores

5.1.1. Caracterización

La etapa de caracterización se aplica a la aplicación paralela y al sistema de E/S. Ésta tiene dos objetivos:

- Identificación de las configuraciones del sistema de E/S y caracterización del rendimiento de cada configuración.
- Extracción del modelo de comportamiento de E/S de aplicaciones científicas paralelas.

Este capítulo se utiliza la metodología para la caracterización del sistema de E/S presentada en el capítulo 4. Para la caracterización de la aplicación se ha utiliza metodología presentada en el capítulo 3

5.1.2. Análisis de E/S

En la etapa de Análisis de E/S se utiliza la información del modelo de comportamiento de la aplicación y las métricas obtenidas con los benchmarks en las diferentes configuraciones de E/S. El proceso de obtención de las métricas se aplica al nivel de dispositivos de E/S y de librería de E/S en cada configuración para diferentes patrones de acceso.

En este paso tenemos dos escenarios para evaluar la relación entre la caracterización realizada a la aplicación y a las diferentes configuraciones. En el escenario 1 existe un evaluación exhaustiva de las prestaciones del sistema de E/S para patrones de acceso similar al del modelo de E/S de la aplicación. En el escenario 2 no existe una caracterización del sistema de E/S para el patrón de acceso del modelo de E/S de la aplicación. A continuación se presenta el análisis de la relación entre el modelo de E/S de la aplicación y la caracterización del sistema de E/S en estos dos escenarios.

Análisis de E/S con caracterización exhaustiva

Cada configuración identificada tiene una tabla de prestaciones para los tres niveles de E/S: librería, sistema de fichero global y sistema de fichero local(dispositivos).

La Figura 5.2 muestra los pasos algorítmicos para la generación de la tabla de porcentaje usado. El camino lógico se presenta a continuación:

- Se toma como entrada la información del patrón de acceso de cada fase que forma parte del modelo de E/S de la aplicación que se esta analizando y la tasa de transferencia obtenida por cada fase cuando se ejecuto la aplicación.
- Se busca en la tabla de prestaciones un patrón similar al de las fases del modelo y se extrae la tasa de transferencia que le correspondería a un patrón similar.
- El porcentaje de prestaciones que la aplicación ha usado es calculado por cada nivel de E/S que ha sido caracterizado para un patrón similar.

El algoritmo usado para buscar la tasa de transferencia en cada nivel de E/S se muestra en la Figura 5.3, este es aplicado en la etapa de búsqueda de la Figura 5.2. Los siguientes pasos explican la Figura 5.3:

- Se abre la tabla de prestaciones y se setea la variable found que es usada para detener el proceso de búsqueda cuando se encuentra un patrón similar.
- Si el tipo de operación, modo de acceso, tipo de acceso es igual al valor que se encuentra en tabla de prestaciones y el tamano del request de la fase que se está analizando es:

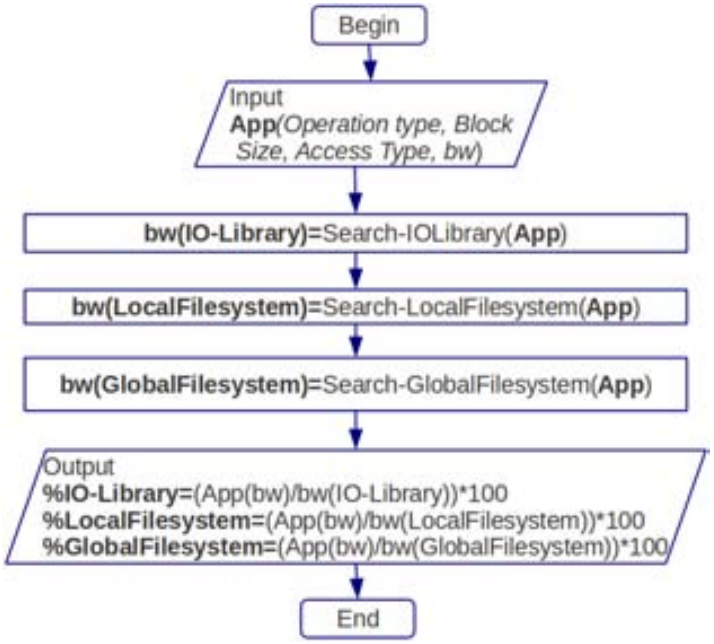


Figura 5.2: Algoritmo para la generación de la tabla de porcentaje usado, donde Block_size=rs y bw=BW

- menor que el mínimo request evaluado en la tabla de prestaciones entonces se selecciona la tasa de transferencia correspondiente al mínimo tamano de request usado durante la etapa de caracterización.
- mayor que el máximo request de la tabla de prestaciones, se selecciona la tasa de transferencia correspondiente al máximo valor de request evaluado en la etapa de caracterización.
- igual a un tamano de request de la tabla de prestaciones entonces se selecciona la tasa de transferencia correspondiente a ese valor.
- si esta entre dos valores de la tabla de prestaciones entonces se selecciona la tasa de transferencia del valor inmediato superior al valor que se esta buscando.

- Cuando la búsqueda termina se cierra la tabla de prestaciones y se devuelve la tasa de transferencia caracterizada.

Se debe tener en cuenta que los valores obtenidos durante la etapa de caracterización ha sido obtenido bajo una carga del sistema de E/S. Cuando una aplicación no hace E/S que necesite el acceso a disco probablemente obtenga una tasa de transferencia mayor a los valores que hemos caracterizado esto se debe a la influencia del buffer/cache. Además, aunque una aplicación escriba un archivo del doble de tamano de la memoria principal (por lo que se eliminaría el efecto del buffer/cache) si las operaciones no son realizadas en

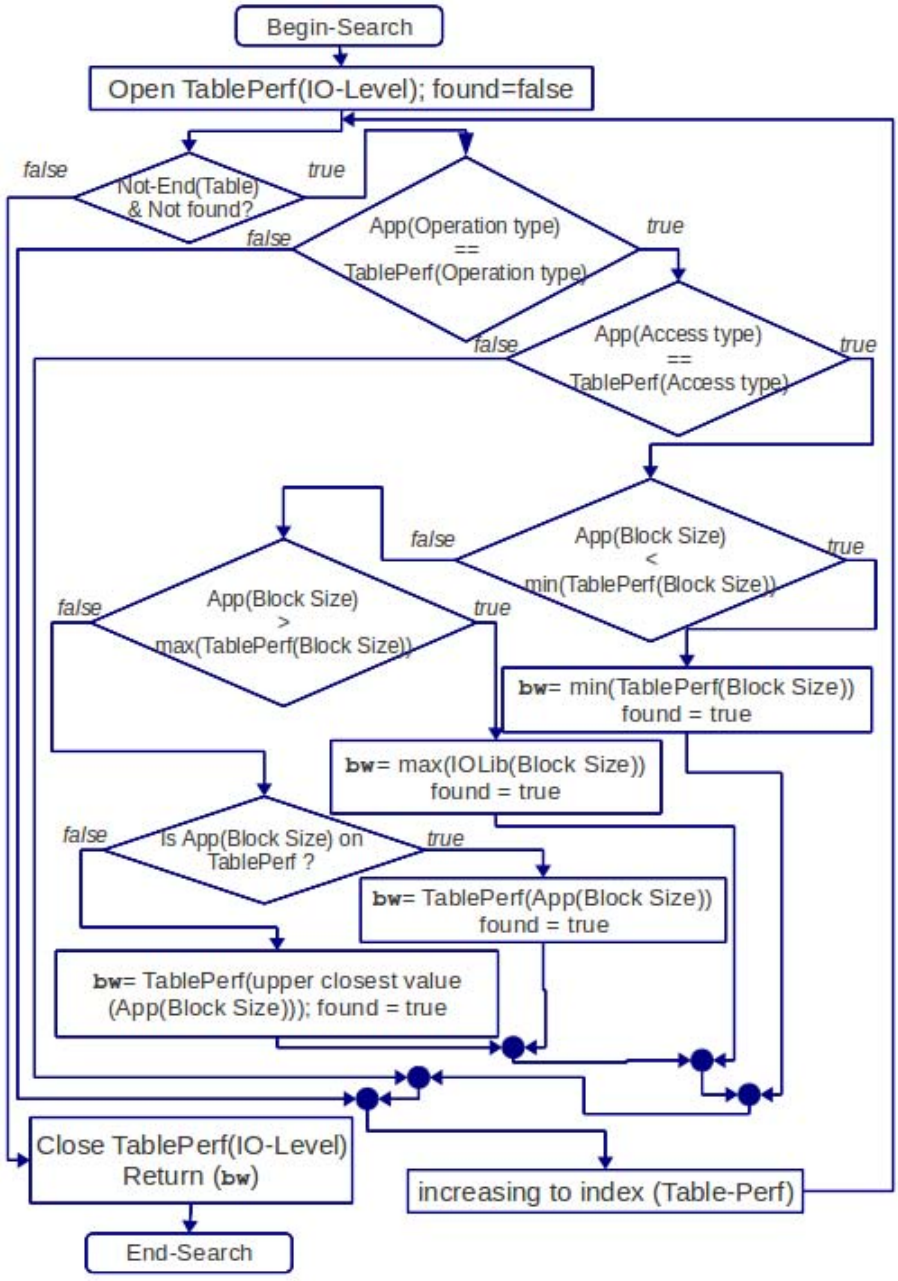


Figura 5.3: Algoritmo de búsqueda en tabla de prestaciones, donde Block_size=rs y bw=BW

forma consecutiva como las realizan los diferentes benchmark de E/S probablemente la tasa de transferencia también será superior al valor que hemos obtenido en la etapa de caracterización. Los benchmark evalúan las prestaciones para un determinado patrón que solo esta haciendo operaciones de E/S. Estos benchmarks representan mejor a un workload de un sistema de E/S que aun patrón de acceso de una aplicación paralela específica. El pico de prestaciones que puede conseguir una aplicación esta dado por la capacidad que los dispositivos de E/S pueden ofrecer. Aunque es muy difícil que una aplicación consiga este valor pico porque debe pasar por las diferentes capas de E/S que llevan a disminuir las prestaciones que la aplicación puede obtener.

Análisis de E/S sin caracterización exhaustiva

En el caso de no contar con una caracterización para un patrón similar a las fases del modelo de E/S se sintoniza el benchmark IOR de la siguiente manera:

- Modo de acceso Strided: $s = Iter$; $b = RS_{(IdPh)}$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$; $-F$ si el tipo de acceso es Único y sin $-F$ para un tipo de acceso Compartido. $-c$ si las operaciones de E/S son colectivas.
- Modo de acceso Secuencial: $s = 1$; $b = weight(IdPh)$; $t = RS_{(IdPh)}$; $NP = np_{(IdPh)}$;si el tipo de acceso es Único y sin $-F$ para un tipo de acceso Compartido. $-c$ si las operaciones de E/S son colectivas.

Se ejecuta el benchmark IOR se obtiene un ancho de banda caracterizado $BW_{(CH)}$ y a partir de esta obtenemos el tiempo de I/O caracterizado que denominamos $Time_{io(CH)}$.

Para determinar que configuración de E/S cumple con los requisitos de la aplicación se establece una relación entre el patrón de acceso de cada fase de E/S de la aplicación con los patrones de acceso de los benchmarks usados para la obtención de las métricas. Por ejemplo para determinar que tasa de transferencia obtendría una fase de E/S se busca el patrón en la base de datos de valores medidos por los benchamrk y se extrae la tasa de transferencia caracterizada. En este caso se opta por seleccionar como métrica de selección el tiempo de E/S. De esta forma aquella configuración que proporcione menor tiempo de E/S será la seleccionada para la aplicación.

El tiempo de E/S se expresa como $Tiempo_{E/S} = \sum_{i=1}^{n} Tiempo_{E/S}(fase[i])$

Donde el $Tiempo_{E/S}(fase[i]) = \frac{Peso(fase[i])}{TasaTransferencia(PatronSimilar_{CH})}$

5.1.3. Evaluación

Una tercera etapa se incluye para evaluar el uso que se hace del sistema de E/S. Utilizando la información de las prestaciones que cada configuración tiene se evalúa cuanto

de esa capacidad usa la aplicación. Para esto se establece una relación entre la traza de transferencia pico y la tasa de transferencia alcanzada por cada fase de la aplicación.

$$UsoSistema(fase[i]) = \frac{TasaTransferencia_{MD}(fase[i])}{TasaTranserencia_{CH}(fase[i])} * 100 \quad (5.1)$$

Además también evaluamos el error cometido en la estimación del tiempo de E/S. Para ello usamos:

$$error_{abs} = BW_{CH}(fase[i]) - BW_{MD}(phase[i]) \quad (5.2)$$

$$error_{rel} = 100 * (\frac{error_{abs}}{BW_{MD}(fase[i])}) \quad (5.3)$$

Cabe aclarar que no usamos el tiempo de E/S para hacer una predicción del tiempo de E/S de la aplicación solo se usa para hacer la selección de la configuración.

5.2. Experimentación

En esta sección vamos a presentar la evaluación de prestaciones de tres sistemas de E/S. Mostrando los diferentes componentes identificados y un análisis de los valores obtenidos. En el capitulo 6 se presenta la aplicación de la metodología considerando tanto la aplicación como el sistema de E/S. En este experimento vamos a usar la metodología propuesta para determinar cuanto de la capacidad del sistema esta siendo utilizado por la aplicación. Se presenta la caracterización de MadBench2 [4] y se evalúa la utilización de este en dos sistemas de E/S. En este caso se cuenta con una caracterización exhaustiva del sistema de E/S. La Tabla 5.1 muestra la configuración de los dos sistemas de E/S.

Es sistema A es compuesto de 14 nodos de cómputo con la siguiente descripción técnica:

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

Es sistema B es compuesto de 32 IBM x3550 nodos de cómputo con la siguiente descripción técnica:

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

5.2.1. Caracterización del sistema de E/S

La Figura 5.4 y la Figura 5.5 muestran las prestaciones medidas con IOR a nivel de MPI-IO para ambos sistemas.

Tabla 5.1: Descripción de los sistemas de E/S denominados A y B

Emenento de E/S	Sistema A	Sistema B
Librería de E/S	mpich2	OpenMPI
Red de comunicación	1 Gbps Ethernet	1 Gbps Ethernet
Red de almacenamiento	1 Gbps Ethernet	1 Gbps Ethernet
Sistema de fichero global	OrangeFS	NFS Ver 3
Nodos de E/S	10	32 DAS and 1 NAS
Metadata Server	1	1
Sistema de Fichero Local	Linux ext3	Linux ext4
Nivel de redundancia	-	RAID 5
Número de dispositivos	11 disks	5 disks
Capacidads	500 GB	1.8 TB
Punto de montaje	/mnt/orengafs	/home

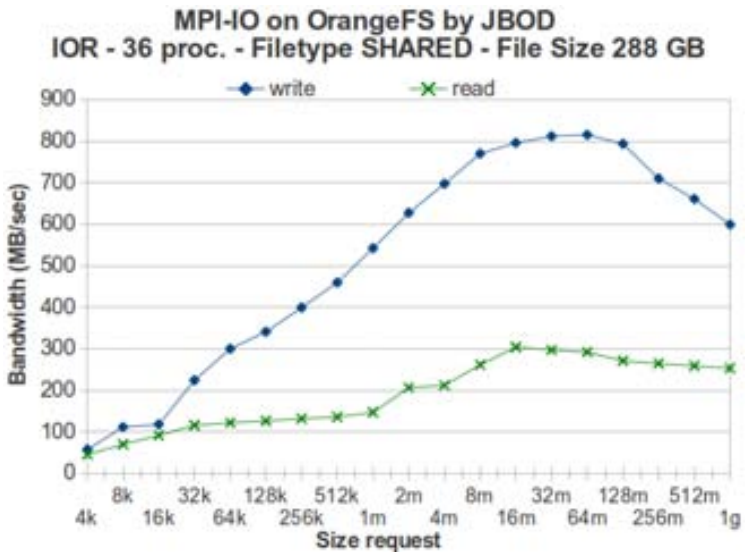


Figura 5.4: Caracterización a nivel de MPI-IO en OrangeFS del Sistema A

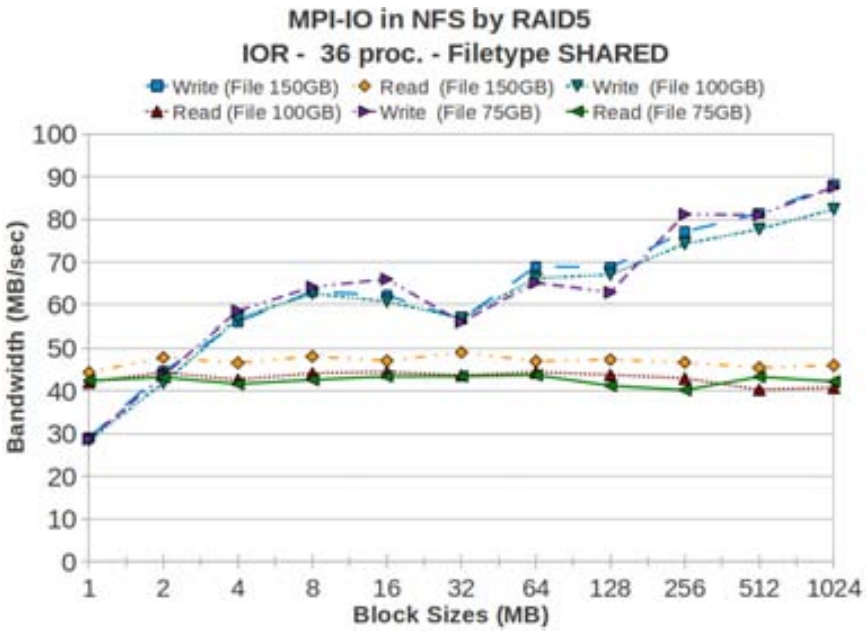


Figura 5.5: Caracterización a nivel de MPI-IO en NFS del Sistema B

En el sistema A (Figure 5.4) se puede observar un incremento en las prestaciones a medida que aumenta el tamano de request para las operaciones de escritura y lectura. Aunque este incremento es mayor para las operaciones de escritura, también se puede observar que a partir de los 64MB las prestaciones caen ligeramente para las lecturas y con mayor diferencia en las escrituras. Claramente en este sistema una aplicación con características de E/S similares al usado en este caso se beneficiará si sus fases de E/S con mayor peso son fases compuesta por operaciones de escritura.

En el sistema B (Figure 5.5) se puede observar prestaciones similares para los diferentes tamanos de request en las operaciones de lectura. Esto nos muestra que en este sistema el tamano de request no es un factor de E/S que impacte en las prestaciones. En el caso de las escrituras vemos que las prestaciones se incrementan a medida que el tamano de request se incrementa, excepto para un request de 32 MB donde se puede observar una caída en las prestaciones. Por ello en este caso se ha evaluado para diferente tamanos de fichero y se puede observar que este comportamiento se repite lo que nos indica que es un comportamiento regular en este sistema y no es consecuencia de una carga temporal del sistema de E/S. Además en el caso del sistema B la evaluación se ha realizado para request mayores a 1 MB debido a que tamanos menores provocan un degradación en las prestaciones y llevan un tiempo considerable de evaluación. En muchos de los test tamano de request del orden de KB provocan la saturación del servidor NFS.

Los valores picos para los sistemas de E/S son: Write=1120 MB/sec, Read=1260MB/sec

para el sistema A y Write=204 MB/sec, Read=300MB/sec para el sistema B.

De esta caracterización se puede observar claramente que las prestaciones del sistema A son superiores que las del sistema B. El sistema B no es adecuado para aplicaciones intensivas en E/S no sólo por el sistema de fichero global sino también por la prestaciones de su red de interconexión. Este sistema tiene una degradación de prestaciones muy severo para request pequenos por lo que una aplicación que escriba o lea en request de KB tenga una penalización muy alta en sus tiempos de E/S. Éste es un clúster configurado para servir a aplicaciones que realizan cómputo y comunicación, y E/S temporal en discos locales no esta configurado para aplicación que necesitan realizar E/S paralela en un fichero compartido.

En el caso del sistema A, éste tiene problemas de prestaciones para tamanos de request superiores a 512 MB, llevando a una caída de prestaciones provocando en varios test la caída de la ejecución del test debido al time out que el propio sistema de fichero impone. El sistema A esta configurado para aplicaciones que requieran E/S paralela por medio del uso de librerías como ROMIO, parallel HDF5 y parallel NetCDF.

5.2.2. Caracterización de MadBench2

MADBench2 es un derivado del código de análisis de datos MADspec. El MADspec estima el espectro de energía regular de la microondas de la radiación cósmica en el cielo de un conjunto de datos pixelados. Como parte de sus cálculos, el MADspec realiza muchas operaciones sobre una matriz *out-of-core* , requiriendo sucesivas escrituras y lecturas de una gran cantidad de datos contiguos de archivos compartidos o individuales. Debido a grandes, contiguas y mezcladas operaciones de lectura y escritura que MADbench2 realiza y su capacidad para probar una variedad de parámetros, éste es elegido como benchmark en la comunidad de la E/S paralela [11, 42]. MADBench2 realiza cuatro pasos

- construye recursivamente una secuencia del polinomio de *Legendre* basado en la matriz componente de la correlación pixel-pixel y la senal CMB (*Cosmic Microwave Background*), escribiendo cada uno a disco.
- Forma e invierte la matriz de componentes senal CMB+ruido (cálculo y comunicación)
- A su vez, lee cada componente de la matriz de correlación senal CMB del disco, multiplicando este por la matriz de correlación de datos CMB inversa, y escribe la matriz resultante a disco.

- A la vez, lee cada par de resultados de la matriz desde disco y calcula el signo de su producto.

Debido a la gran cantidad de memoria del dominio computacional en los cálculos reales, todas las matrices requeridas generalmente no entran en memoria simultáneamente. De esta forma, un algoritmo *out-of-core* es utilizado, el cual requiere suficiente memoria como para almacenar solo 5 matrices en cualquier momento. La metainformación de MAdBench2 es: Modo de acceso Secuencial y el tipo de acceso es compartido. Usa Individual file pointers, operaciones Non-collective y Blocking I/O. Para el manejo de la posición de lo punteros usa la directiva MPI_File_seek.

En este caso se ha caracterizado MadBench2 para 36 procesos, 40 KPIX y para un tipo de acceso SHARED. La Figura 5.6 muestra las fases de E/S para el modelo de E/S de MadBench2 y la Tabla 5.2 la descripción de las fases.

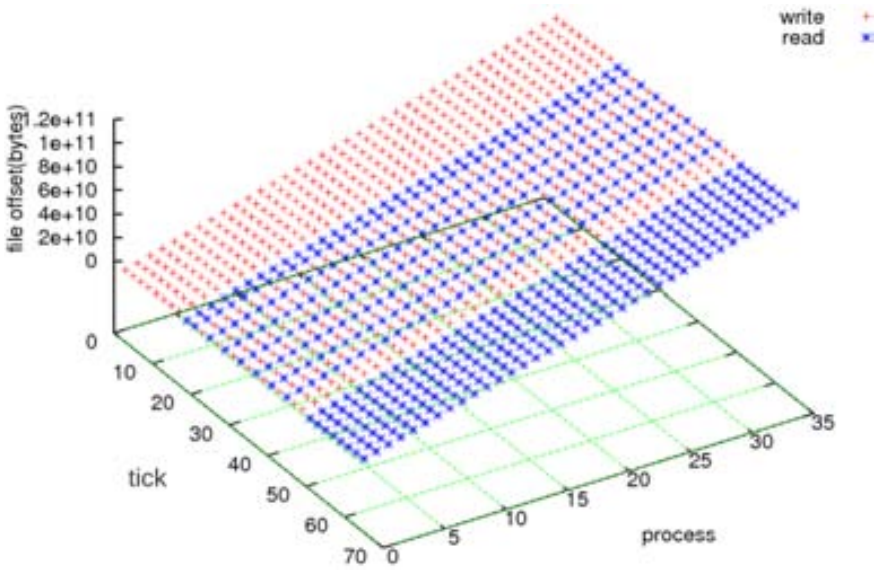


Figura 5.6: Fases de E/S para el modelo de E/S de MadBench2 para 36 processes, 40KPIX, y tipo de acceso SHARED

5.2.3. Análisis de E/S y Evaluación

La tabla 5.4 muestra la utilización de la capacidad de prestaciones del sistema B. Se puede observar el peso de las fases que es cantidad de datos transferidos en cada fase (*weight*), el número y tipo de operación (W=write, R=read, W-R=write-read), y la tasa de transferencia medido $BW_{(MD)}$ y caracterizado $BW_{(PK)}$ en MB/segundo.

La Tabla 5.3 muestra la utilización para el sistema A. Se puede observar que las fases 1 y 4 (con operaciones de escritura) han utilizado mayor capacidad de las prestaciones

Tabla 5.2: Descripción de las fases de MADBench2 para $np = 36$ procesos y tamano de request $rs = 352$ MB

ID_Phase	np	**Operation**	rs	rep	$weight$
1	36	write	352MB	8	102GB
2	36	read	352MB	2	25GB
3	36	write	352MB	6	75GB
	36	read	352MB	6	75GB
4	36	write	352MB	2	25GB
5	36	read	352MB	8	102GB

Tabla 5.3: Utilización del sistema, $BW_{(PK)}$ y $BW_{(MD)}$ en MB/segundo para MADBench2 para 36 procesos, tamano de fichero de 102 GB, RS=352MB y tipo de acceso SHARED en el sistema A

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

que las 2, 3 y 5 (fases con lecturas o compuestas). La tercera fase ha usado alrededor del 31 % de la capacidad, las operaciones de lectura utilizan un porcentaje similar que esta alrededor del 20 %.

Tabla 5.4: Utilización del sistema, $BW_{(PK)}$ y $BW_{(MD)}$ en MB/segundo para MADBench2 para 36 procesos, tamano de fichero de 102 GB, RS=352MB y tipo de acceso SHARED en el sistema B

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

La Tabla 5.4 muestra la utilización para el sistema B. Las fases 1 y 4 utilizan una mayor capacidad del sistema de E/S como ocurre en el sistema A. La fase 3 ha usado alrededor del 16 % y las 2 y 5 han usado alrededor de 15 %.

Para MadBench2 las fases con mayor impacto en el sistema de I/O son aquellas que

realizan las operaciones en forma consecutiva y con operaciones de escritura. Las fases 2, 3 y 5 no impactan en el sistema de E/S debido a que se realizan con una frecuencia suficiente como para que las operaciones de E/S no sean un cuello de botella para la aplicación.

5.3. Conclusiones

Una metodología para analizar el rendimiento de E/S de las computadoras paralelas se ha propuesto y aplicado. Dicha metodología incluye la caracterización del sistema de E/S en los diferentes niveles: el dispositivo, librería de E/S y sistema de fichero global. Se analizó y evaluó la configuración de los diferentes elementos que influyen en el rendimiento teniendo en cuenta la aplicación y la configuración de E/S. Esta metodología se aplicó en dos clúster de computadores para el benchmark MadBench2. Se evaluaron las características de ambos sistemas de E/S, así como su impacto en el rendimiento de la aplicación. También se observa que la misma aplicación tiene diferentes tasas de transferencia para operaciones de escritura y lectura en la misma configuración de E/S. Esta situación afecta a la selección de la configuración apropiada, ya que será necesario analizar la operación con más peso para la aplicación. MadBench2 es un ejemplo que muestra el impacto de la misma configuración en diferentes fases de la aplicación como se pudo observar en la experimentación.

Capítulo 6

Evaluación Experimental

En este capítulo se aplica la metodología propuesta en el capítulo 5 a FLASH-IO y a Upwelling del framework ROMS. Estos se evalúan en tres clúster de computadores. Se analizan los modelos de E/S y se seleccionan las fases con mayo peso para evaluar las prestaciones en los tres sistemas de E/S solo centrándose en las fases con mayor peso. Además analizamos de acuerdo a las características de E/S de ambas aplicaciones y a los recursos de E/S de cada clúster se seleccionar la configuración más adecuada para mejorar las prestaciones de las aplicaciones evaluadas.

6.1. Entorno de experimentación

Los modelos de E/S han sido obtenidos en CAPITA y en Finisterrae[6] y se han evaluado en tres sistemas de E/S. La Tabla 6.1 la descripción del sistema de E/S de los clúster que usamos en la parte experimental.

El Cluster CAPITA esta compuesto por 14 nodos de cómputo con la siguiente descripción técnica:

- 4 cores AMD Phenom^{TM} II (8MB cache) or Athlon^{TM} II (2MB cache)
- 4 DIMM slots en 16GB DDR3, y
- Discos de 500GB SATA

El Cluster Finisterrae esta compuesto por 143 nodos de cómputo con la siguiente descripción técnica:

- 142 HP Integrity rx7640 nodos con 16 Itanium Montvale cores y 128 GB de memoria RAM.

Tabla 6.1: Descripción de los sistemas de E/S de los clústeres CAPITA, Finisterrae and Supernova

I/O Element	CAPITA	Supernova	Finisterrae
I/O library	mpich2	mvapich2	mpich2, HDF5
Communication Network	1 Gbps Ethernet	Infinibad 20 Gbps	Infinibad 20 Gbps
Storage Network	1 Gbps Ethernet	Infinibad 20 Gbps	Infinibad 20 Gbps
Filesystem Global	OrangeFS	Lustre 2.1.5	Lustre (HP SFS)
I/O nodes	10	8 OSS	18 OSS
Metadata Server	1	2 MDS	2 MDS
Filesystem Local	Linux ext3	Linux ext3	Linux ext3
Level Redundancy	-	RAID 6	RAID 6
Capacity of I/O Devices	500 GB	485TB	216TB
Stripe Size	64 KB	1 MB	4MB
Mounting Point	/mnt/orangefs	/lustre/scratch	$HOMESFS

- 1 nodo HP Integrity Superdome con 128 Itanium Montvale cores y 1,024 GB de memoria RAM.

Supernova esta compuesto por 570 nodos de cómputo con la siguiente descripción técnica:

- 126 Intel Xeon E5345 2.33 GHz nodos con 8 (2x quad-core) y 16GB de memoria RAM.
- 40 Intel Xeon L5420 2.5 GHz nodos con 8 (2x quad-core) y 16GB de memoria RAM.
- 404 Intel Xeon X5650 2.67 GHz nodos con 12(2x six-core) y 24GB de memoria RAM.

6.2. FLASH-IO Benchmark

El Benchmark FLASH-IO [5] es un kernel de la aplicación FLASH, un código de hidrodinámica de una maya adaptiva de bloque estructurado que resuelve ecuaciones de hidrodinámica reactiva, desarrollado principalmente para estudio de los flashes de los neutrones de las estrellas y las enanas blancas. El dominio computacional es dividido en bloques en un número de procesos. Un bloque es un arreglo de tres dimensiones con cuatro elementos adicionales que hacen de celdas protectoras en cada dimensión para almacenar información de sus vecinos. FLASH IO produce un archivo de checkpoint y dos archivos conteniendo datos centrales y de las esquinas. El archivo de mayor tamanos es el

de checkpoint, el tiempo de E/S que domina el benchamrks. FLASH IO usa una librería de alto nivel de E/S (HDF5) para almacenar los datos con sus metadatos.

La siguiente meta-información se obtuvo para FLASH-IO en su versión parallel HDF5:

- Explicit offset, operaciones Blocking, Collective y Non-collective.
- Modo de acceso Strided y tipo de acceso Shared en tres ficheros usados por todos los procesos de la aplicación.
- MPI_Set_view con diferentes etypes y filetype para las operaciones collective y non-collective.

FLASH-IO realiza únicamente operaciones de escrituras a través de HDF5 paralelo que son convertidas a operaciones de MPI-IO. La Figura 6.1 muestra las fases de FLASH-IO. Podemos observar que los tres ficheros son accedidos en forma consecutiva. Para el análisis se los han enumerado teniendo en cuenta su orden de apertura durante la ejecución de la aplicación. En esta caso hay dos tipo de operaciones usadas MPI_File_write_at and MPI_File_write_at_all. Debido a que las operaciones collectivas (MPI_File_write_at_all) representan más del 90 % de la E/S para el análisis solo mostramos la fases para este tipo de operaciones.

La Tabla 6.2 muestra la descripción de las fases del primer fichero.

Tabla 6.2: Descripción de fases de FLASH-IO para Write_at_all - Primer Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	1920	2	3	124392	122KB
4	64	write	3840	1	3	439012	245KB
5	64	write	2621440	24	3	169869312	4GB

Donde IdPh. es el identificador de la fase, rs es el tamano del request, rs es el número de repeticiones, *dist.* es la distancia, *disp.* es el desplazamiento.

Podemos observar que hay 5 fases donde la quinta fase es la de mayor peso esta tiene 24 operaciones de 2MB que son realizadas por todos los procesos.

La Tabla 6.3 muestra la descripción para el segundo fichero podemos observar también cinco fases donde la quinta fase es la de mayor pero pero en este caso son solo 4 operaciones realizadas por todos los procesos de la aplicación.

La Tabla 6.4 muestra las fases para el tercer fichero que tiene un comportamiento similar a las fases del segundo fichero con diferencia en el peso de la quinta fase. En este

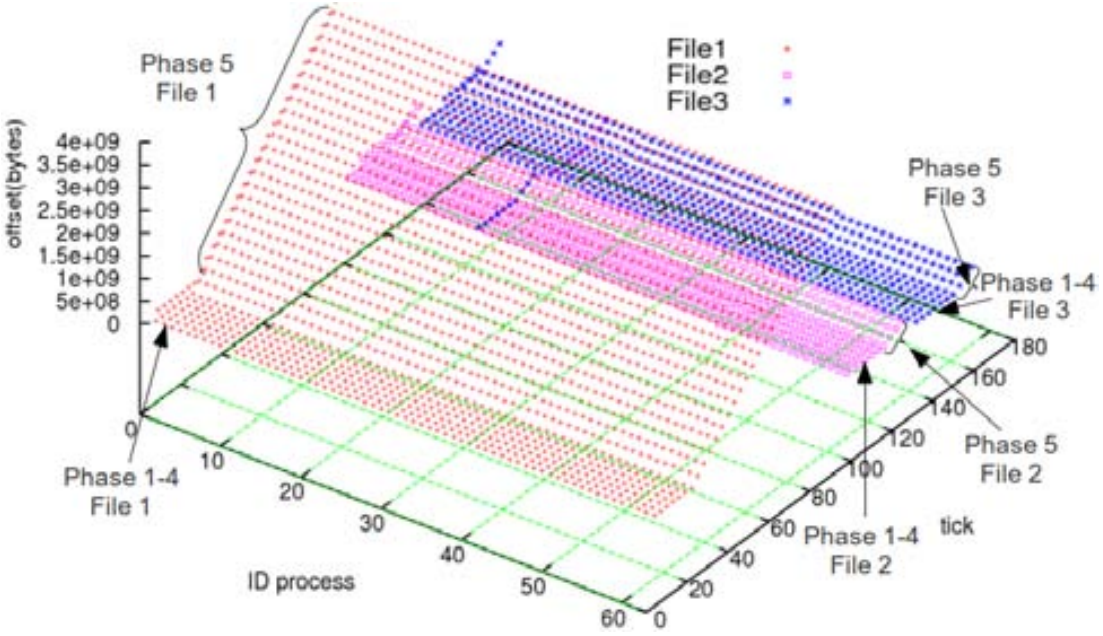


Figura 6.1: Fases de E/S de FLASH-IO para 64 procesos

Tabla 6.3: Descripción de fases de FLASH-IO para Write_at_all - Segundo Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	128KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1310720	4	3	84934656	335MB

caso las fases se ordenan de la siguiente forma Id_File.Id_Fase: 1.1, 1.2, 1.3, 1.4. 1.5, 2.1, 2.2, 2.3, 2.4, 2.5,3.1.3.2,3.3,3.4 y 3.5. Por lo tanto, FLASH-IO tiene 15 fases de E/S.

6.2.1. Análisis de E/S y Evaluación

Debido al modelo de E/S de FLASH-IO se va aplicar una análisis de E/S con una caracterización a medida para esto ajusta a IOR con la siguiente configuración para obtener un patrón similar a la fase 1.5, 2.5 y 3.5 que son las de mayor peso para cada fichero.

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

Tabla 6.4: Descripción de fases de FLASH-IO para Write_at_all - Tercer Fichero

IdPh	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*peso*
1	64	write	320	2	3	20732	40KB
2	64	write	4800	1	3	310980	307KB
3	64	write	960	2	3	62196	122KB
4	64	write	1920	1	3	301260	122KB
5	64	write	1572160	4	3	101974016	402MB

Como los fichero son accedidos en un orden secuencial y no hay solapamiento en la manipulación de los ficheros se ejecuta IOR con las tres configuraciones en el orden dado por el número de fichero File1, File2 y File3. Al ejecutar IOR en Finisterrae se obtiene la tasa de transferencia para este patrón y se calcula los tiempo de E/S que en este caso representaran el tiempo de E/S caracterizado. La Tabla 6.5 muestra el tiempo calculado con IOR $Time_{io(CH)}$, tiempo de E/S para FLASH-IO $Time_{io(MD)}$, and el error relativo $error_{rel}$. Se puede observar errores grandes para 64 procesos en los fichero 2 y 3. Esto es debido a que estos son ficheros muy pequenos (400MB) y pueden estar influenciados por los efecto buffer y cache. Aunque para el propósito del análisis lo que interesa es que tengan la misma tendencia y esto se cumple. El error decrece si aumenta la E/S que la aplicación realiza como ocurre en el caso de 128 procesos.

Tabla 6.5: Tiempos de E/S en Finisterrae para 64 y 128 procesos para la quinta fase de FLASH-IO

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

Analizando el modelo de E/S de FLASH-IO se puede observar que el primer fichero tiene mayor peso este crece a medida que aumente le números de procesos de la aplicación. FLASH-IO tiene el mismo patrón de E/S para sus fases de modelo independientemente de la cantidad de nodos que se usen lo que lleva a que el peso de las fases sea proporcional al número de procesos. Por lo tanto, la E/S puede llegar a ser un cuello de botella para un número mayor de procesos (mayor a 1024 procesos). Como el tipo de acceso es compartido

y el fichero 1 crece a medida que agregamos procesos un alternativa para aumentar la tasa de transferencia y reducir el tiempo de E/S es aumentar el número de nodos de E/S para los tres ficheros manteniendo el stripe por defecto. Esto se realizo para FLASH-IO y para su versión IOR (solo para la fase 5), el resultado se muestra en la Tabla 6.6. Se puede observar que la tendencia es similar. Estos es útil porque se puede usar la versión de IOR para evaluar diferentes configuraciones. Esto reduce los tiempos de evaluación ya que la aplicación completa (no solo la E/S) siempre consumirá más tiempo y si solo evaluamos la E/S se podrá encontrar la mejor configuración en menor tiempo.

Tabla 6.6: Tiempos de E/S en Finisterrae para 64 y 128 procesos para la quinta fase de FLASH-IO en 4 nodos de E/S y tamano de Stripe de 4MB

Phase	$Time_{io(CH)}$	$Time_{io(MD)}$	$error_{rel}$
64p			
File 1	30.39	35.18	13 %
File 2	1.97	4.06	50 %
File 3	2.35	4.27	44 %
128p			
File 1	79.52	78.12	2 %
File 2	5.67	5.78	2 %
File 3	8.11	7.20	12 %

Los tiempos de E/S con esta configuración se decrementa en un 33 % para el File1, 12 % para el File2, y 11 % para el File3 in 64 procesos. Para 128 procesos el tiempo de E/S se reduce en un 23 % para el File1, 31 % para el File2, y 35 % para el File3. Un tendencia similar se puede observar para la versión con IOR.

6.3. Roms - Upwelling

ROMS [7] es ampliamente usado en el modelado de océanos en la comunidad científica para un diversidad de aplicaciones. Este incluye un algoritmo numérico y físico eficiente y preciso y varios modelos para biogeochemical, bio-optical, sediment, y sea ice. También incluye varios esquemas de mezcla vertical, múltiples niveles de grid anidados y compuestos.

En este trabajo se ha evaluado Upwelling con E/S paralela. Esta opción habilita la escritura a través de NetCDF parallel. Upwelling escribe sobre 4 ficheros cada 5 pasos. Lo hemos configurado para que trabaje para 64 procesos (NtileI == 8, NtileJ == 8) y 9600 iteraciones (NTIMES==9600).

Esta es la metainformación obtenida:

- Cuatro ficheros para todos los procesos.
- Operaciones: con Explicit offset, Blocking I/O, Collective operations y Non-collective.
- Modo de accesp: Strided y Tipo de acceso: Shared.
- MPI_Set_view con diferente etype y filetype para las operaciones collective y non-collective.
- Los archivos 1, 2 y 3 tiene un offset incremental para cada operación de I/O. El cuatro archivo tiene el mismo offset para las diferentes operaciones de E/S. Este comportamiento se ha observado en su modelo de E/S. De hecho, el tamano del cuarto archivo es 4.8MB, aunque el tamano total desplazado was 73MB.
- Los cuatro archivos realizan un MPI_File_sync antes de cada iteración de la fase 12.

El primer, segundo y tercer fichero, tienen un comportamiento similar pero con diferente offset y pesos. Aquí solo se presenta la descripción de las fases con mayor peso para el primer y cuarto fichero.

Las fases para el primer fichero se muestra en la Figura 6.2. Se puede observar que los 8 primeros y últimos procesos escriben en tick similares, además tiene un request y offset similar. Los procesos 8 a 55 están agrupados en otra fase ya que escriben en otros ticks llevando a un patrón diferente para los otros 16 procesos.

La Tabla 6.7 muestra la descripción de la fase nº 12 para los procesos 0-7 y 56-63, donde esta se repite 133 veces. En este caso hay un MPI_File_sync antes de cada iteración esto asegura que antes de iniciar cada nueva iteración el dato debe estar escrito en disco.

La Tabla 6.8 muestra las fases para los procesos 8 a 55, también con un MPI_File_sync antes de cada iteración de su fase 12. El offset para cada iteración de las Tablas 6.7 y 6.8 es calculado por la expresión (6.1).

$$OInit_{(12.i,iter)} = OInit_{(i,iter-1)} * i * 1416004 \tag{6.1}$$

Donde i $1 - 12$ y $iter$ $2 - 133$.

Las fases para el cuarto fichero se muestran en la Figura 6.3. En este caso el offset es el mismo en cada operación. Nuevamente los primeros y últimos procesos trabajan en ticks, request y offset similares. Los procesos 8 a 55 son agrupados en otra fase por trabajar en diferentes ticks por lo que siguen un patrón diferente.

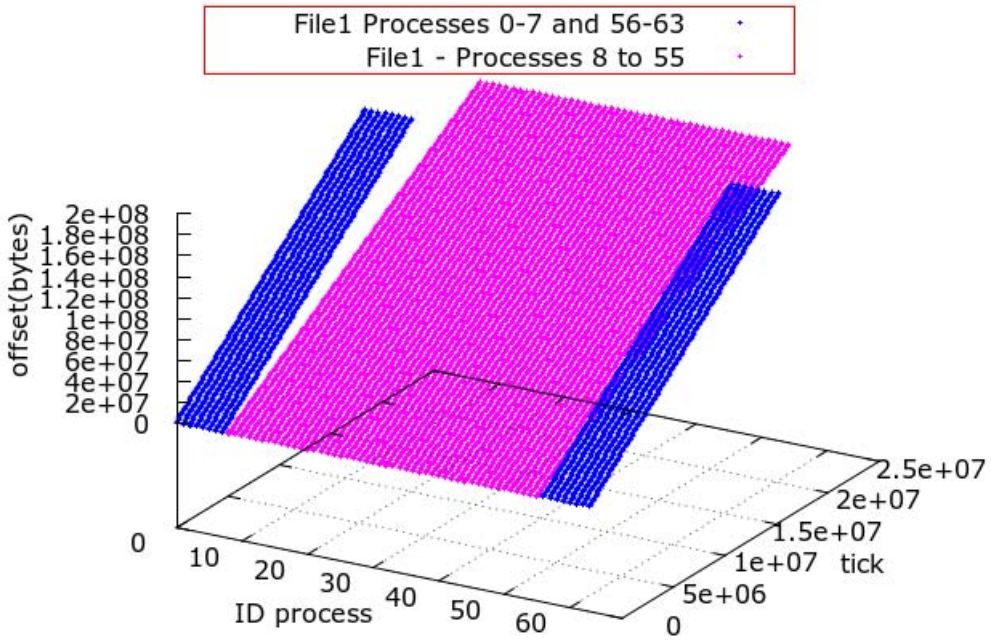


Figura 6.2: Fases de E/S Upwelling para el primer fichero para 64 procesos

La Tabla 6.9 y 6.11 muestran la descripción de las fases para los procesos 0-7 y 56-63 donde la fase 12 se repite 16 veces. La Tabla 6.10 muestra las fases para los procesos 8 a 55. En ambos casos se realiza MPI_File_sync antes de cada iteración de la fase 12.

6.3.1. Análisis de E/S y Evaluación

En este caso se ha seleccionado la fase 12 de cada fichero para sintonizar los parámetros de IOR y obtener las prestaciones para la fase con mayor peso para Upwelling. Los parámetros son seteados como sigue:

- File 1: -np 16 -a MPIIO -c -s 133 -b 35904 -t 4488; -np 48 -a MPIIO -c -s 133 -b 32640 -t 4080
- File 2: -np 16 -a MPIIO -c -s 133 -b 33792 -t 4224; -np 48 -a MPIIO -c -s 133 -b 32640 -t 4080
- File 3: -np 16 -a MPIIO -c -s 133 -b 261888 -t 4224, -np 48 -a MPIIO -c -s 133 -b 238080 -t 3840

Tabla 6.7: Descripción de las fases del modelo de Upwelling - Write_at_all - Primer File - Procesos 0-7 and 56-63

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
11	16	write	4224	2	3	225664	135KB
12				133	137776		55.8MB
12.1	16	write	264		3	14104	4KB
12.2	16	write	264		3	13776	4KB
12.3	16	write	240		3	13932	3.8KB
12.4	16	write	4224		3	220416	67KB
12.5	16	write	3840		3	222912	61KB
12.6	16	write	4488		9	239768	71KB
12.7	16	write	4488		3	239768	71KB
12.8	16	write	4224		3	225664	67KB
12.9	16	write	4224		3	225664	67KB

- File 4: -np 16 -a MPIIO -c -s 16 -b 135168 -t 8448; -np 48 -a MPIIO -c -s 16 -b 122880 -t 7680

Como la fase 12 tiene diferentes tamanos de request, se elige el mayor request de la fase 12, el tamano de bloque es puesto al número de patrones de E/S de cada iteración multiplicado por el request. El segmento es puesto el número de iteraciones de la fase.

Como los 4 fichero son accedidos en forma concurrente para calcular la tasa de transferencia se toma aquella que produce el mayor tiempo de E/S para las configuraciones de IOR. Es este tiempo caracterizado para la aplicación.

Esto se ha evaluado en Finiserrae y en Supernova. La Tabla 6.12 muestra el tiempo obtenido con IOR $Time_{io(CH)}$, tiempo de E/S de Upwelling $Time_{io(MD)}$, y el error relativo $error_{rel}$.

Se puede observar que el tamano de los archivos de Upweeling son menores a 1 GB y que los tamanos de request son pequenos: 200 bytes a 4480 bytes para le primer archivo, 240 bytes a 4224 bytes para el segundo archivo, 240 bytes a 4224 bytes para el tercero y 200 bytes a 8448 bytes para el cuarto archivo.

En este caso, un importante factor es la ubicación de los cuatro archivos. Dado el tamano de los ficheros y los tamanos de request de la fase con mayor peso, es conveniente almacenar los 4 archivos en el mismo OSS, aunque se debe evaluar si será en el mismo OST o distintos OSTs.

Para evaluar esta situación se ha analizado Upwelling en cuatro configuraciones de Finisterrae para evaluar cuál es la más adecuada. La estructura del sistema de fichero Lustre de Finisterrae se muestra en la Tabla 6.13 Upwelling se ha evaluado en las siguientes

Tabla 6.8: Descripción de las fases del modelo de Upwelling - Write_at_all - Primer Archivo - Procesos 8-55

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
11	48	write	3840	2	3	225664	369KB
12	...			133	183608	...	154MB
12.1	48	write	240		3	14104	11.5KB
12.2	48	write	240		3	13776	11.5KB
12.3	48	write	240		3	13932	11.5KB
12.4	48	write	3840		3	220416	184KB
12.5	48	write	3840		3	222912	184KB
12.6	48	write	4080		11	239768	196KB
12.7	48	write	4080		3	239768	196KB
12.8	48	write	3840		3	225664	184KB
12.9	48	write	3840		3	225664	184KB

cuatro configuraciones:

- los cuatro ficheros con la configuración por defecto: es este caso cada archivo se almacena en un OST. En esta configuración el tiempo de E/S represento el 24 % del tiempo total de ejecución.
- los cuatro archivos en el mismo OSS y diferentes OSTs: en este caso el tiempo de E/S representa en 26 % del tiempo total.
- los cuatro archivos en el mismo OST: en este caso el tiempo de E/S representa el 20 %.
- los cuatro archivos en 4 OSSs: en este caso el tiempo de E/S representa el 21 % del tiempo total de ejecución.

Se puede observar que el peor resultado se da para la configuración donde cada fichero es almacenado en un OST del mismo OSS. Probablemente se debe a una sobrecarga para el OSS causando un caida de prestaciones e incrementando el tiempo de E/S.

EL menor porcentaje se puede observar para la configuración donde los cuatro archivos son almacenados en el mismo OST. Esto puede deberse al tamano de los archivos y al número de procesos paralelos que hacen E/S.

6.4. Conclusiones

Se ha aplicado la metodología para evaluar las prestaciones para una aplicación real y para un kernel de aplicación científica. Estos presentan patrones de E/S complejos y

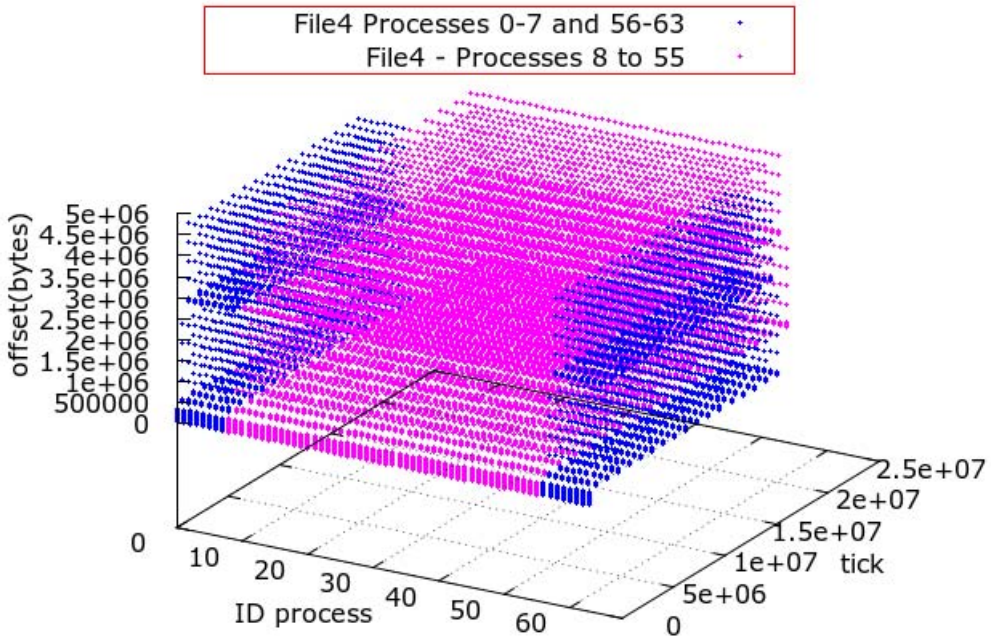


Figura 6.3: Fases de E/S Upwelling para el cuarto fichero para 64 procesos

además usan varios ficheros durante la ejecución de la aplicación.

Se ha identificado las fases con mayor peso y la evaluación de prestaciones se ha realizado únicamente para estas fases. Además de acuerdo a las características de la aplicación se ha podido determinar que configuración de E/S sería la adecuada para mejorar las prestaciones. Este proceso se ha realizado en tres sistemas de E/S.

En el caso de FLASH-IO debido a que sus ficheros son accedidos por todos los procesos de la aplicación y crecen a medida que se aumentan los procesos, es recomendable que se envíe cada fichero a más de un nodo de E/S. En este trabajo se usaron 4 nodos de E/S. Para Upwelling se trabaja con la afinidad de los nodos de E/S, ubicando los ficheros en diferentes nodos de almacenamiento pero en el mismo OSS.

Se puede observar que la metodología es aplicable a clúster en producción y que se puede realizar una caracterización a medida en entornos de producción donde no es adecuado evaluar prestaciones con caracterizaciones exhaustivas porque no se puede disponer del clúster para evaluación de prestaciones de E/S exclusivamente.

Tabla 6.9: Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 0-7

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	551015		11MB
12.1	8	write	528		3	28208	4KB
12.2	8	write	528		3	27552	4KB
12.3	8	write	480		3	27864	3.8KB
12.4	8	write	8448		3	440832	67KB
12.5	8	write	7680		3	445824	61KB
12.6	8	write	8448		3	451328	67KB
12.7	8	write	8448		3	451328	67KB
12.8	8	write	8448		3	-4197200	67KB
12.9	8	write	528		551015	28208	4KB
12.10	8	write	528		3	27552	4KB
12.11	8	write	480		3	27864	3.8KB
12.12	8	write	8448		3	440832	67KB
12.13	8	write	7680		3	445824	61KB
12.14	8	write	8448		3	451328	67KB
12.15	8	write	8448		3	451328	67KB
12.16	8	write	8448		3	451328	67KB

Tabla 6.10: Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 8-55

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	734350		10MB
12.1	48	write	480		3	28208	3.8KB
12.2	48	write	480		3	27552	3.8KB
12.3	48	write	480		3	27552	3.8KB
12.4	48	write	7680		3	440832	61KB
12.5	48	write	7680		3	440832	61KB
12.6	48	write	7680		3	440832	61KB
12.7	48	write	7680		3	440832	61KB
12.8	48	write	7680		3	-4197200	61KB
12.9	48	write	480		734350	28208	3.8KB
12.10	48	write	480		3	27552	3.8KB
12.11	48	write	480		3	27552	3.8KB
12.12	48	write	7680		3	451328	61KB
12.13	48	write	7680		3	451328	61KB
12.14	48	write	7680		3	451328	61KB
12.15	48	write	7680		3	451328	61KB
12.16	48	write	7680		3	451328	61KB

Tabla 6.11: Descripción de las fases del modelo de Upwelling - Write_at_all - Cuarto File - Procesos 56-63

ID_Phase	*np*	**Operation**	*rs*	*rep*	**dist.**	**disp.**	*weight*
...	...	...	...	...	...	...	...
12				16	551005		11MB
12.1	8	write	528		3	28208	4KB
12.2	8	write	528		3	27552	4KB
12.3	8	write	528		3	27552	4KB
12.4	8	write	8448		3	440832	67KB
12.5	8	write	8448		3	440832	67KB
12.6	8	write	8448		3	440832	67KB
12.7	8	write	8448		3	440832	67KB
12.8	8	write	8448		3	-4197200	67KB
12.9	8	write	528		551005	28208	4KB
12.10	8	write	528		3	27552	4KB
12.11	8	write	528		3	27552	4KB
12.12	8	write	8448		3	451328	67KB
12.13	8	write	8448		3	451328	67KB
12.14	8	write	8448		3	451328	67KB
12.15	8	write	8448		3	451328	67KB
12.16	8	write	8448		3	451328	67KB

Tabla 6.12: Tiempo de E/S y error relativo en Finisterrae y Supernova para 64 procesos para Upwelling

Phase	$Time_{io(CH)}$	$Time_{io(MD)}$	$error_{rel}$
Finisterrae	429	418	2 %
Supernova	988	882	12 %

Tabla 6.13: Estructura de Lustre en Finisterrae - Cuatro OSTs por OSS

OSS	**OSTs**	**OSS**	**OSS**
1	0-3	10	36-39
2	4-7	11	40-43
3	8-11	12	44-47
4	12-15	13	48-51
5	16-19	14	52-55
6	20-23	15	56-59
7	24-27	16	60-63
8	28-31	17	64-67
9	32-35	18	68-71

Capítulo 7

Conclusiones

La E/S paralela es un área del cómputo de altas prestaciones que implica no solo el análisis de las aplicaciones científica sino también el estudio de la estructura del sistema de E/S. Esto se ha visto reflejado en este trabajo donde hemos presentado un análisis de la E/S tanto para el sistema de altas prestaciones como de las aplicaciones.

7.1. Conclusiones Finales

En este trabajo se ha presentado una metodología que permite evaluar las prestaciones de los sistemas de E/S de los computadores de altas prestaciones. Esta metodología da un orden a la evaluación de prestaciones indicando que parámetros se deben considerar al momento de determinar las valores picos y medios de prestaciones.

Para considerar las características de las aplicaciones se ha definido una metodología para la caracterización de la E/S de las aplicaciones científicas paralelas [25]. Se presenta la definición de un modelo de E/S que se centra en las fases de E/S de la aplicación y orden de ocurrencia de eventos de E/S y de comunicación. Las fases de E/S están representadas por los patrones de E/S y el peso de la fase que indica la cantidad de datos que la aplicación moverá en el sistema de E/S.

Se realiza la extracción de trazas a nivel de MPI de las operaciones de E/S identificando los parámetros necesarios para la elaboración de un modelo de E/S que es independiente de los tiempos de E/S del sistema donde se extraen las trazas [26].

Además se brinda información de la E/S que permite analizar y seleccionar el sistema de E/S teniendo en cuenta el modelo de E/S de las aplicaciones.

A partir del modelo de E/S de la aplicación se obtiene los requisitos de E/S, estos se analizan para determinar que recursos del sistema de E/S son necesarios y se pueden ajustar para cumplir con estos requisitos y lograr aumentar la tasa de transferencia y

reducir el tiempo de E/S.

La metodología se ha aplicado a diferentes kernels de aplicaciones científicas y a una aplicación real sobre 5 clústeres de computadores para los sistemas de ficheros paralelos Lustre, PVFS2 y para el sistema compartido NFS. Las aplicaciones que se han analizado demuestran que la E/S no solo es un factor que afecta las prestaciones para cunado los archivos son de gran tamano (decenas de GB) sino también a aplicaciones que utilizan varios pequenos ficheros compartidos (cientos de MB). La afinidad de nodos de E/S es una alternativa para mejorar las prestaciones para aplicaciones que usan varios ficheros compartidos. Cuando los archivos son compartidos por varios procesos y tienen un tamano del orden de los GB una alternativa es aumentar el número de nodos de almacenamiento.

7.2. Trabajo Futuro y Lineas Abiertas

A continuación se presentan la lineas abiertas y trabajos futuros de este trabajo:

- Extender la utilización de la metodología para un mayor número de aplicaciones reales que presenten patrones complejos de E/S.
- Incorporar aplicaciones paralelas que usen POSIX-IO esto permitirá cubrir un mayor número de aplicaciones del campo del cómputo de altas prestaciones.
- Utilizar la información de la aplicación para hacer la selección de recursos en entornos cloud y para gestión de colas en clústeres de computadores.
- Usar simulación para usando la información de la aplicación y del sistema de E/S evaluar el efecto de los cambios de los factores con mayores influencia en las prestaciones como el número de nodos de E/S, la afinidad de nodos de E/S y el tamano de stripe.
- Simular la arquitectura de la E/S para evaluar el efecto de dimensionar el sistema de E/S o cambiar componentes como la red de almacenamiento, dispositivos de E/S.
- Extender la metodología para que considere requisitos de escalabilidad y consumo energético.

7.3. Lista de Publicaciones

Las principales aportaciones de esta tesis están en las siguientes publicaciones realizadas:

1. **Sandra Méndez, Dolores Rexachs y Emilio Luque. A Methodology to characterize the parallel I/O of the message-passing scientific applications, in Proceeding de PDPTA 13 - The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications. Pendiente de Publicación. Julio 2013.**

   En este trabajo se presenta una metodología que permite extraer los patrones de acceso y metadatos de la aplicación expresándolos en un modelo de E/S. Este modelo esta en función de los patrones de acceso de la aplicación, donde estos patrones son agrupados en fases de E/S y es en base a estas fases que se analiza las prestaciones para la aplicación en diferentes sistema de E/S. Debido a que el modelo de E/S que se obtiene es a nivel de librería de E/S este es independiente del sistema de E/S permitiéndonos usarlos en sistemas con diferentes configuraciones.

2. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Modeling Parallel Scientific Applications through their Input/Output Phases, in Proceeding de IASDS 12 - Workshop on Interfaces and Architectures for Scientific Data Storage in CLUSTER 2012. Páginas 07-15. Septiembre 2012.**

   Se presenta un modelado de las aplicaciones científicas paralelas. Se obtiene un modelo de E/S de la aplicación en función de sus fases de E/S que se usan para seleccionar la configuración de E/S que mejor se ajuste a los requisitos de E/S de la aplicación.

3. **Sandra Méndez, Javier Panadero, Alvaro Wong, Dolores Rexachs y Emilio Luque. A new approach for Analyzing I/O in parallel scientific applications, in Proceeding de CACIC 2012 - XVIII Congreso Argentino de Ciencias de la Computación. Páginas 337-346. Octubre 2012.**

   En este trabajo se presenta una metodología para analizar las aplicaciones científicas paralelas considerando: cómputo, comunicación, operaciones de E/S y el comportamiento temporal y espacial a nivel del sistema de almacenamiento.

4. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Evaluating Utilization of the I/O System on Computer Clusters, in Proceeding de PDPTA 12 - The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications. Páginas 366-372. Julio 2012.**

Se presenta una metodología para determinar cuanto de la capacidad del sistema de E/S es usado por una aplicación paralela. Se establece una relación formal entre los patrones de E/S de las aplicaciones con las medidas de prestaciones de los sistemas de E/S.

5. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Methodology for Performance Evaluation of the Input/Output System on Computer Clusters, in Proceeding de IASDS 11 - Workshop on Interfaces and Architectures for Scientific Data Storage in CLUSTER 2011. Páginas 474-483. Septiembre 2011.**

   En este trabajo se presenta una metodología para evaluar las prestaciones del sistema de E/S en clúster de computadores. Se presenta la caracterización de la aplicación a nivel de fases. Se presenta un método algorítmico para la evaluación del uso del sistema de E/S. Se seleccionan los factores de E/S que afectan las prestaciones que una aplicación percibe en un clúster determinado. Se refina la evaluación de prestaciones al incorporar el tamano, patrón y tipo de acceso de las operaciones de E/S logrando de esta forma una evaluación más precisa del uso del sistema de E/S.

6. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Efficiency Evaluation of the Input/Output System on Computer Clusters, in Proceeding de CACIC 2011 - XVII Congreso Argentino de Ciencias de la Computación. Páginas 261-270. Octubre 2011.**

   En este trabajo presenta un proceso para la evaluación de la eficiencia del uso del sistema de E/S. Se destacan la estructuras de datos para las prestaciones de las configuraciones de E/S tanto para la caracterización del sistema de E/S como para los valores evaluados.

7. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Methodology for Performance Evaluation of the Input/Output System, in Proceeding de PDPTA 11 - The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications. Páginas 886-889. Julio 2011.** En este trabajo se presenta un metodología para la evaluación del sistema de I/O. Se destaca la etapa de caracterización tanto de la aplicación como del sistema paralelo. La evaluación de prestaciones se hace de acuerdo a valores promedios de las operaciones de E/S en cada configuración de E/S evaluada. Las etapas de análisis de la configuración y evaluación son presentados en forma

empírica.

8. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Metodología para Analizar y Evaluar los Sistemas de Entrada/Salida Paralelos, in Proceeding de las Jornadas de Paralelismo 2011. Páginas 569-574. Septiembre 2011.**

   En este trabajo se presenta un análisis de la caracterización de las aplicaciones científicas como parte de una metodología de evaluación de prestaciones del sistema de entrada/salida paralelo. Además con la caracterización de la aplicación se presenta un modelo preliminar de entrada/salida de una aplicación científica.

9. **Sandra Méndez, Dolores Rexachs y Emilio Luque. Impacto de la Entrada/Salida en los Computadores Paralelos, in Proceeding de CACIC 2010 - XVI Congreso Argentino de Ciencias de la Computación. Páginas 491-495. Octubre 2010.**

   En este trabajo se propone una metodología para el análisis de E/S en los clústeres de computadores, que permita examinar cómo afectan las diferentes configuraciones a la aplicación.

10. **Javier Panadero, Sandra Méndez, Dolores Rexachs y Emilio Luque. Evaluación del impacto de la eficiencia energética en el sistema de Entrada/Salida, en Proceeding de CACIC 2011 - XVII Congreso Argentino de Ciencias de la Computación. Páginas 271-281. Octubre 2011.**

    En este trabajo se presenta una metodología para caracterizar, analizar y evaluar el impacto de las aplicaciones intensivas de E/S en la eficiencia energética. Para ello, se consideran dos niveles del sistema de E/S: el nivel de dispositivo y el nivel de sistema. Aporte como Co-Autor: En este trabajo colabore en la elaboración del artículo. Además de la selección de los componentes del sistema de E/S que se debían tener en cuenta para el consumo energético. Análisis y selección de aplicaciones, benchmarks de E/S para la evaluación del impacto del sistema de E/S en el consumo energético.

11. **Javier Panadero, Sandra Méndez, Dolores Rexachs y Emilio Luque. Characterizing energy efficiency in I/O system for scientific applications, in Proceeding de ADVCOMP 2011- The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences. Páginas 106-112. Noviembre 2011.**

En este trabajo presenta una metodología para caracterización de la eficiencia energética en el sistema de E/S. La metodología considera tanto las prestaciones como la energía, y extrae información del ancho de banda y consumo energético por medio de diferentes benchmarks. Aporte como Co-Autor: En este trabajo colabore con la evaluación de las prestaciones del sistema de E/S, las operaciones de E/S que se deben considerar, la selección de los diferentes benchmarks usados. Además colabore en la elaboración de la metodología.

Bibliografía

[1] T. M. William Loewe and C. Morrone. (2012) Ior benchmark. [Online]. Available: https://github.com/chaos/ior/blob/master/doc/USER_GUIDE

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

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

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

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

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

[7] O. M. Group. (2013) Regional ocean modeling system (roms). [Online]. Available: http://www.myroms.org/

[8] J. H. Laros, L. Ward, R. Klundt, S. Kelly, J. L. Tomkins, and B. R. Kellogg, Red storm io performance analysis, in *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 50 57.

[9] W. Yu, S. Oral, J. Vetter, and R. Barrett, Efficiency evaluation of cray xt parallel io stack, 2007.

[10] W. Yu, H. S. Oral, R. S. Canon, J. S. Vetter, and R. Sankaran, Empirical analysis of a large-scale hierarchical storage system, in *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 130 140.

[11] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, I/o performance challenges at leadership scale, in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1 12.

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

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

[14] S. Kim, Y. Zhang, S. Son, R. Prabhakar, M. Kandemir, C. Patrick, W.-k. Liao, and A. Choudhary, Automated tracing of i/o stack, in *Recent Advances in the Message Passing Interface*, ser. LNCS. Springer Berlin/Heidelberg, 2010, vol. 6305, pp. 72 81.

[15] N. Nakka, A. Choudhary, W. Liao, L. Ward, R. Klundt, and M. Weston, Detailed analysis of i/o traces for large scale applications, in *Intl. Conf. on High Performance Computing (HiPC)*, dec. 2009, pp. 419 427.

[16] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, Parallel i/o prefetching using mpi file caching and i/o signatures, in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, 15-21 2008, pp. 1 12.

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

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

[19] M. P. I. Forum. (2009) Mpi: A message-passing interface standard. [Online]. Available: http://www.mpi-forum.org/docs/mpi-2.2

[20] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[21] M. U. R. Ross, R. Latham and B. Welch., Paralelli/o in practice, tutorial in the acm/ieee supercomputing conference, Tech. Rep., 2009.

[22] Y. Chen, X.-H. Sun, R. Thakur, P. Roth, and W. Gropp, Lacio: A new collective i/o strategy for parallel i/o systems, in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 794 804.

[23] K.-L. M. J. C. K. I. Carmen Sigovan, Chris Muelder and R. Ross, A visual network analysis methodology for large scale parallel i/o systems, in *Proceedings of IPDPS 2013*, May 2013, (Accepted for publication and presentation).

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

[25] , Modeling parallel scientific applications through their input/output phases, in *CLUSTER Workshops'12*, 2012, pp. 7 15.

[26] S. Méndez, J. Panadero, A. Wong, D. Rexachs, and E. Luque, A new approach for analyzing i/o in parallel scientific applications, in *CACIC 12, CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION*, 2012, pp. 337 346.

[27] J. M. May, *Parallel I/O for high performance computing.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[28] D. G. Feitelson, P. F. Corbett, S. Johnson Baylor, and Y. Hsu, Parallel i/o subsystems in massively parallel supercomputers, *IEEE Parallel Distrib. Technol.*, vol. 3, no. 3, pp. 33 47, 1995.

[29] D. Kotz, Introduction to multiprocessor i/o architecture, in *Input/Output in Parallel and Distributed Computer Systems, chapter 4.* Kluwer Academic Publishers, 1996, pp. 97 123.

[30] R. Ross, R. Thakur, and A. Choudhary, Achievements and challenges for i/o in computational science, *Journal of Physics: Conference Series*, vol. 16, no. 1, p. 501, 2005. [Online]. Available: http://stacks.iop.org/1742-6596/16/i=1/a=069

[31] R. Thakur, W. Gropp, and E. Lusk, Data sieving and collective i/o in romio, in *Frontiers of Massively Parallel Computation, 1999. Frontiers '99. The Seventh Symposium on the*, 21-25 1999, pp. 182 189.

[32] T. T. of HDF at NCSA, Hdf5 wins 2002 r&d 100 awards, 2002.

[33] R. Rew and G. Davis, Netcdf: an interface for scientific data access, *Computer Graphics and Applications, IEEE*, vol. 10, no. 4, pp. 76 82, jul 1990.

[34] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, Parallel netcdf: A high-performance scientific i/o interface, in *Supercomputing, 2003 ACM/IEEE Conference*, 15-21 2003, pp. 39 39.

[35] F. Schmuck and R. Haskin, Gpfs: A shared-disk file system for large computing clusters, in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2002, p. 19.

[36] P. J. Braam and Others, The Lustre storage architecture, *White Paper, Cluster File Systems, Inc., Oct*, vol. 23, 2003.

[37] R. B. Ross and R. Thakur, Pvfs: A parallel file system for linux clusters, in *In Proceedings of the 4th Annual Linux Showcase and Conference*. MIT Press, 2000, pp. 391 430.

[38] R. L. e. a. Rob Ross, Rajeev Thakur, Parallel i/o benchmarking consortium, http://www.mcs.anl.gov/research/projects/pio-benchmark/pio-benchmark.pdf, Tech. Rep., 2002.

[39] P. Wong and R. F. V. D. Wijngaart, Nas parallel benchmarks i/o version 2.4, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, Tech. Rep., 2003.

[40] A. Nisar, W.-k. Liao, and A. Choudhary, Scaling parallel i/o performance through i/o delegate and caching system, in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1 12.

[41] W.-k. Liao and A. Choudhary, Dynamically adapting file domain partitioning methods for collective i/o based on underlying parallel file system locking protocols, in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1 12.

[42] J. Borrill, L. Oliker, J. Shalf, and H. Shan, Investigation of leading hpc i/o performance using a scientific-application derived benchmark, in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1 12.

[43] P. Carns, Y. Yao, K. Harms, R. Latham, R. Ross, and K. Antypas, Production i/o characterization on the cray xe6, in *Proceedings of the Cray User Group meeting 2013 (CUG 2013)*, May 2013.

[44] P. Carns, K. Harms, D. Kimpe, R. Ross, J. Wozniak, L. Ward, M. Curry, R. Klundt, G. Danielson, C. Karakoyunlu, J. Chandy, B. Settlemeyer, and W. Gropp, A case for optimistic coordination in hpc storage systems, in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, ser. SCC 12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 48 53. [Online]. Available: http://dx.doi.org/10.1109/SC.Companion.2012.19

[45] R. Coker, Bonnie++ filesystem benchmark, Tech. Rep., 2001. [Online]. Available: http://www.coker.com.au/bonnie++/

[46] R. Rabenseifner and A. E. Koniges, Effective file-i/o bandwidth benchmark, in *Euro-Par '00: Procs from the 6th Int. Euro-Par Conference on Parallel Procs*. London, UK: Springer-Verlag, 2000, pp. 1273 1283.

[47] R. Jain, J. Werth, and J. C. Browne, Eds., *Input/Output in Parallel and Distributed Computer Systems*. USA: Kluwer Academic Publishers, 1996.

[48] P. Brezany, Ed., *Input/Output Intensive Massively Parallel Computing*. Berlin, Heidelberg, New York: Springer-Verlag, 1997.

[49] H. Jin, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, R. Buyya and T. Cortes, Eds. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[50] T. Ludwig, Research trends in high performance parallel input/output for cluster environments, in *Proceedings of the 4th International Scientific and Practical Conference on Programming UkrPROG2004, Pages 274-281, National Academy of Sciences of*, 2004.

[51] S. J. Baylor, C. Benveniste, and Y. Hsu, Performance evaluation of a massively parallel i/o subsystem, *SIGARCH Comput. Archit. News*, vol. 22, no. 4, pp. 5 10, 1994.

[52] J. Li, L. Yan, Z. Gao, and D. Hei, A task-pool parallel i/o paradigm for an i/o intensive application, in *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*, 10-12 2009, pp. 679 684.

[53] T. M. Madhyastha and D. A. Reed, Learning to classify parallel input/output access patterns, *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 8, pp. 802 813, 2002.

[54] E. L. Miller and R. H. Katz, Input/output behavior of supercomputing applications, in *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 1991, pp. 567 576.

[55] P. C. Roth, Characterizing the i/o behavior of scientific applications on the cray xt, in *PDSW '07: Proceedings of the 2nd international workshop on Petascale data storage*. New York, NY, USA: ACM, 2007, pp. 50 55.

[56] H. Yu, R. Sahoo, C. Howson, G. Almasi, J. Castanos, M. Gupta, J. Moreira, J. Parker, T. Engelsiepen, R. Ross, R. Thakur, R. Latham, and W. Gropp, High performance file i/o for the blue gene/l supercomputer, in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, 11-15 2006, pp. 187 196.

[57] A. Núnez, J. Fernández, J. D. Garcia, L. Prada, and J. Carretero, Simcan: a simulator framework for computer architectures and storage networks, in *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1 8.

[58] K. Coloma, A. Ching, A. Choudhary, W. keng Liao, R. Ross, R. Thakur, and L. Ward, A new flexible mpi collective i/o implementation, in *Cluster Computing, 2006 IEEE International Conference on*, 25-28 2006, pp. 1 10.

[59] A. Ching, A. Choudhary, W. keng Liao, L. Ward, and N. Pundit, Evaluating i/o characteristics and methods for storing structured scientific data, in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 25-29 2006, p. 10 pp.

[60] R. Ross, D. Nurmi, A. Cheng, and M. Zingale, A case study in application i/o on linux clusters, in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM, 2001, pp. 11 11.

[61] J. Piernas, J. Nieplocha, and E. J. Felix, Evaluation of active storage strategies for the lustre parallel file system, in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, 10-16 2007, pp. 1 10.

[62] M. Fahey, J. Larkin, and J. Adams, I/o performance on a massively parallel cray xt3/xt4, in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 14-18 2008, pp. 1 12.

[63] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Sclatter Ellis, and M. Best, File-access characteristics of parallel scientific workloads, *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 10, pp. 1075 1089, oct 1996.

[64] D. Kotz, Disk-directed i/o for mimd multiprocessors, *ACM Trans. Comput. Syst.*, vol. 15, no. 1, pp. 41 74, 1997.

[65] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett, Server-directed collective i/o in panda, in *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM, 1995, p. 57.

[66] J. M. del Rosario, R. Bordawekar, and A. Choudhary, Improved parallel i/o via a two-phase run-time access strategy, *SIGARCH Comput. Archit. News*, vol. 21, no. 5, pp. 31 38, 1993.

[67] A. Choudhary and Y. Ishikawa, I/o systems. draft the international exascale software project roadmap, www.exascale.org, Tech. Rep., 2009.

[68] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi, Passion: Optimized i/o for parallel applications, *Computer*, vol. 29, pp. 70 78, 1996.

[69] R. Ross, Parallel i/o benchmarking consortium - mpi-tile-io, 2001.

[70] D. A. Patterson, G. Gibson, and R. H. Katz, A case for redundant arrays of inexpensive disks (raid), in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1988, pp. 109 116.

[71] D. A. Patterson, G. A. Gibson, and R. H. Katz, A case for redundant arrays of inexpensive disks (raid), Berkeley, CA, USA, Tech. Rep., 1987.

[72] K. Salem and H. Garcia-Molina, Disk striping, in *Proceedings of the Second International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1986, pp. 336 342.

[73] M. Livny, S. Khoshafian, and H. Boral, Multi-disk management algorithms, *SIGMETRICS Perform. Eval. Rev.*, vol. 15, no. 1, pp. 69 77, 1987.

[74] G. Santos, A. Duarte, Dolores, and E. Luque, Increasing the performability of computer clusters using radic ii, in *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security.* Washington, DC, USA: IEEE Computer Society, 2008, pp. 653 658.