Using AWS EC2 as Test-Bed infrastructure in the I/O system configuration for HPC applications

Pilar Gomez-Sanchez[1], Diego Encinas[2], Javier Panadero[1], Aprigio Bezerra[3], Sandra Mendez[4], Marcelo Naiouf[2], Armando De Giusti[2], Dolores Rexachs[1], and Emilio Luque[1]

[1] *Computer Architecture and Operating Systems Department,Universitat Autònoma de Barcelona,Campus UAB,Edifici Q,08193 Bellaterra(Barcelona), Spain.*
[2] *Informatics Research Institute LIDI, National University of La Plata, La Plata, Buenos Aires, Argentina*
[3] *Departamento de Ciências Exatas e Tecnológicas, Universidade Estadual de Santa Cruz, Ilhéus, Bahia, Brasil*
[4] *High Performance Systems Division, Leibniz Supercomputing Centre (LRZ), D-85748, Garching bei München, Germany.*

Abstract

In recent years, the use of public cloud platforms as infrastructure has been gaining popularity in many scientific areas and High Performance Computing (HPC) is no exception. These kinds of platforms can be used by system administrators as Test-Bed systems for evaluating and detecting performance inefficiencies in the I/O subsystem, and for taking decisions about the configuration parameters that have influence on the performance of an application, without compromising the performance of the production HPC system. In this paper, we propose a methodology to evaluate parallel applications by using virtual clusters as a test system. Our experimental validation indicates that virtual clusters are a quick and easy solution for system administrators, for analyzing the impact of the I/O system on the I/O kernels of the parallel applications and for taking performance decisions in a controlled environment.

Keywords: Cloud Computing, Parallel File System, PVFS2, MPI applications,

1 Introduction

Due to the complex interaction between the parallel I/O subsystem and the parallel message-passing applications, the I/O subsystem can turn into a bottleneck in parallel systems, bringing about performance inefficiencies in the parallel applications.

In order to use the system in an efficient way, improving the performance, users and system administrators have focused their efforts on tuning the I/O subsystem with parallel applications.

Unfortunately, the high number of configuration parameters, together with the fact that the right combination of parameters is highly dependent on the scientific application, results in making it a complex task to modify any configuration parameter in computation centers, where the HPC system is in production 24/7.

Cloud computing is gaining popularity in many areas, including HPC. Unlike on traditional HPC platforms, on a virtual cluster, users are their own administrators, making it easy to change the I/O parameters, such as the choice of the number of I/O nodes or the stripe size, in order to fit the I/O requirements of the MPI application. Moreover, as a main advantage, the changes carried out in the I/O subsystem do not have any influence on the production HPC system.

In this paper, we discuss how a virtual HPC system, deployed in cloud platform, can be employed as a Test-Bed system to detect performance inefficiencies and to tune the MPI application with the I/O subsystem, so as to achieve high performance. In this way, system administrators can replicate its HPC platform as virtual clusters, in order to evaluate and take decisions about the configuration parameters that have an influence on the performance of an application, without compromising overall system performance.

To guide users in this evaluation, we propose a methodology that strives on using a bounded time to evaluate the I/O subsystem. To achieve this requirement, we have focused both on a reduced analysis time and a bounded time to deploy the virtual cluster. To achieve a reduced analysis time, the I/O patterns are analyzed to extract an I/O kernel of a parallel application, using a physical HPC system. Once this kernel is obtained, it is replicated in the virtual system using the IOR benchmark or a synthetic program, to evaluate the I/O execution time and the bandwidth of the application as performance indexes. In this way, users can evaluate the performance of the application in the I/O subsystem in a quick and efficient way, in order to tune the configuration I/O system parameters.

On the other hand, to deploy the I/O virtual subsystem, we have developed a plugin for the StarCluster tool [1], which allows us to deploy the PVFS2 [2] file system quickly and automatically.

This paper is organized as follows. Section 2 presents the related work. Section 3 introduces our proposed methodology. In Section 4 we present the experimental evaluation, and in Section 5 the experimental validation. Finally, in the Section 6, we discuss the conclusions and future work.

2 Related work

Although traditionally the research efforts in virtual HPC systems have been focused on evaluating computation and communication, in recent years, there has been increase of work focused on assessing the parallel I/O subsystems in cloud environments.

Liu et al. [3] demonstrate that the unique configurability advantage offered by public clouds may bring opportunities for HPC applications to achieve significant performance or cost improvement. In particular, they illustrate the impact of virtual cluster configurability by assessing the impact of I/O system customization on the Amazon EC2 system. There are other works which have been focused on optimizing cloud platform configurations on the Amazon EC2 system. Gideon et al. [4] study the impact of different data sharing options for scientific workflows on Amazon EC2. Exposito et al. [5] evaluate the I/O storage subsystem on the Amazon EC2 CC platform to determine its suitability for scientific applications with high I/O performance requirements . Vecchiola et al. [6] execute a fMRI brain imaging workflow on Amazon EC2 using the Amazon Simple Storage Service (S3) for storage, and analyzed the cost varying the number of nodes.

Moreover, there are some tools which are focused on configurating the I/O subsystem in virtual clusters. Zhai et al. [7] propose ACIC (Automatic Cloud I/O Configurator), one of the first tools to optimize the I/O system for parallel applications in the cloud. Given a parallel application to run on a given cloud platform, ACIC automatically searches for optimized I/O system configurations from many candidate settings. Herodotou et al. [8] propose Elastisizer, which can select the proper cluster size and instance types for MapReduce workloads running in the cloud.

Our proposal differs from these approaches, because our objective is to use the virtual cluster as a Test-Bed platform, to take decisions about the production HPC system.

3 Proposed Methodology

We propose a methodology that is composed of a set of steps that allow us to extract the I/O behavior of the parallel application and define its I/O kernel, define different configurations of the I/O system and evaluate them in a Cloud environment.

3.1 Application Selection

Recent advances in different scientific and engineering areas are generating large amounts of data that need to be stored and analyzed efficiently. These huge volumes of data must be processed in a satisfactory time to enable decision-making and new discoveries. So we have a group of applications with different types of requirements. Applications where the main requirement is not the need for processing, but managing the huge amounts of data within a reasonable time. Storing, transfering and processing large volumes of data become determining factors for systems that handle these types of applications, and a challenge for high performance computing.

Reconciling these needs with requests from traditional applications is a major challenge for administrators of multi-user clusters. In this type of environment it is necessary to consider the sources of I/O contention when the cluster administrator defines the job scheduling policy: CPU, Memory, Storage I/O, Network I/O, System bus I/O and Blade chassis I/O.

It is necessary to examine whether the application triggers waiting times of I/O. When this occurs, the make-span times tend to increase (measured time from sending the first work to cluster until the end of last work released). The cluster administrator needs to assess where the bottleneck is, if memory makes swap, if the I/O system collapses or both of these things.

3.2 Analyzing the I/O properties

HPC applications perform I/O operations by using two common I/O strategies: serial and parallel. The second I/O strategy should be used to take advantage of the performance capacity of the modern HPC-I/O systems. Parallel I/O can be done by using specialized I/O libraries or statements provided by the different programming languages. Understanding the behavior of the operation into file is crucial for detecting I/O bottlenecks and for proposing I/O tuning techniques.

In this section, we present the main I/O characteristics of parallel applications that are portable to different I/O systems. These are selected to help the user to evaluate and understand the I/O behavior.

Two main properties impact on the I/O performance: the I/O concurrence degree and the data amount transferred into the file system. To represent these two issues, we extract the following information:

- At parallel application level

  - The number of files
  - The number of MPI processes
  - Storage capacity required
  - Data to be transferred

- At file level
  - Access Mode: for p MPI processes a file can be accessed using a strided, sequential or random pattern.
  - Access Type: Unique (a file per MPI process) or Shared (a shared File for all or a set of MPI processes).
  - Data access type: File can be write only (W), read only(R) or write-read (W/R).
  - Number of I/O processes
  - The count of concurrent accesses into the file system: the temporal pattern is extracted considering the time stamp of opens and data access operations within the different files.
  - Data access operation properties: request size and the frequency of the I/O operations.

HPC tracing and profiling tools can be applied to extract the I/O properties. However, unlike the compute performance analysis, the I/O performance at file level is not a typical characteristic provided by the performance analysis tool. Due to this, we select the Darshan [9] tool to extract the I/O profiling and apply our experience on the performance analysis to define the portable I/O characteristics that represent the behavior of the parallel application.

3.3 Replicating the I/O properties

Once the application I/O behavior has been obtained, it is replicated by using a synthetic program or a benchmark with flexibility to reproduce the I/O kernel of the parallel applications.

An I/O benchmark that allows us to set up different I/O properties is IOR [10],which is developed to be used in tests on parallel file systems in high-performance cluster.

IOR can be used on any platform that provides an MPI implementation. It also offers the ability to test aggregate I/O rates through several typical middle-ware libraries, including MPI-IO, HDF5, NetCDF and POSIX libraries.

Input parameters allow us to vary the overall size of the I/O operation, the individual size of each transfer, the mode of access to the file and if the data is accessed sequentially or randomly. These inputs can be used to reproduce I/O patterns of HPC applications.

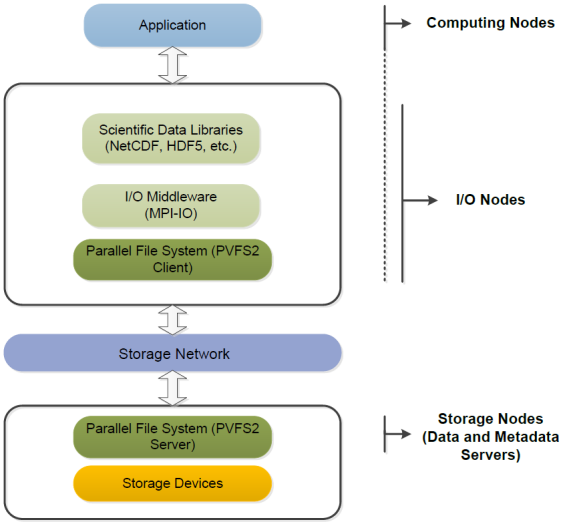


Figure 1: HPC I/O System

3.4 Deploying the Test Environment on Cloud

On Cloud, a HPC cluster can be created in similar way to physical cluster because it offers individual HPC resources. The Cloud platforms provide different types of instances that include a specific number of CPU, memory, storage and the networking performance. Furthermore, a cloud environment usually provides image with a stack software to deployed Linux clusters.

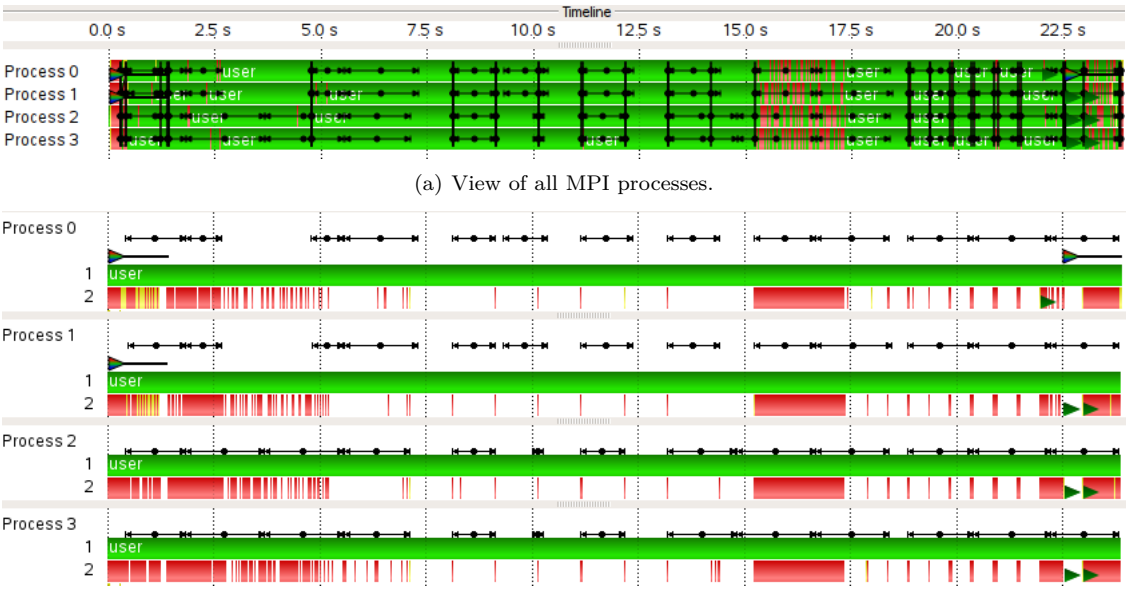
An automatic tool to create Linux Clusters is StarCluster [1], which helps users and system administrators to create a virtual cluster using instance of Amazon's EC2 platform. We select this tool to create a base HPC cluster.

Usually the HPC-I/O subsystem must be configured by the user in a Cloud environment. The HPC-I/O system is represented as is shown in Figure 1. The I/O software stack comprises Scientific Data Library (like HDF5 and NetCDF), I/O Middle-ware (like MPI-IO) and Parallel File System (like GPFS, Lustre and PVFS2) [11].

In this point, we create an image on AWS EC2 that implements the I/O software stack and this is used to deploy a cluster using Starcluster tool. The user can select the type of instance and select our image to have a fully functional HPC-I/O system.

3.5 Configuring a Parallel Filesystem

Parallel applications need a global file system to share the input and output data. Depending on the application I/O pattern, the type of file system can have a negative impact on the I/O performance. The most common global file systems in HPC cluster are NFS and parallel file systems like Lustre, GPFS, PVFS2[12].



(a) View of all MPI processes.

(b) View for each MPI process

Figure 2: Vampir Trace for ABYSS-P using 4 MPI processes. Test case for two small input files and k=25. User events in green, MPI events in red, and I/O events in dark green triangles and yellow bars.

Starcluster tool deploys a cluster with NFS, therefore, in order to use a parallel file system, this must be installed and configured by the user.

In this case, we have chosen PVFS2, which is an open source file system developed to support efficient read and write operations of large amounts of data. PVFS2 is designed as a client-server architecture where the server provides the storage and the client contains the access logic to the distributed storage. Servers can be classified into datafile (DF) and metadata servers (MDS). The former keep parts of logical files while the latter keep attributes of the logical file system objects (files and directories in PVFS). Usually every I/O node is either a DF or an MDS.

In general, parallel file systems have several parameters that affect the performance. However, in this work we focus on number of DFs, the stripe size and the placement of the DFs and MDS into the architecture of the HPC cluster. The number of DF defines the file stripping and stripe size is the number of bytes written on one DF before cycling to the next DF.

3.6 Analyzing Results

Parallel application performance can be affected for several issues like memory, compute, communication or I/O into files. The user normally focuses on obtaining the results in the least possible time. However, the administrator expects to have high throughput at the HPC system level.

In this work, we focus our analysis on the I/O into files to identify and extract the I/O patterns of parallel applications that are portable to other HPC-IO systems. The metrics considered to analyze the results are data transfer rate, I/O time and I/O operations per second. A parallel application I/O kernel is replicated through the I/O patterns identified and IOR benchmark on different I/O subsystems.

4 Experimental Evaluation

In this section, we present the evaluation of the parallel application ABySS (Assembly By Short Sequences) [13], which reports slow running on the CACAU [14] cluster of the NBCGIB (Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas, Brazil). The user provides us with the input data and specific parameters to evaluate ABYSS. We perform an I/O pattern analysis in LRZ's HPC systems [15] and define a set of different configurations of the I/O system to be implemented in a Cloud environment.

4.1 ABySS Application Description

ABySS is a parallelized sequence assembler. It is specialized in assembling large genomes from short sequences, using a pair-end (forward and reverse) method. ABYSS is an example of a *de novo* assembling algorithm, where the construction of a genome occurs in its pure form, without consulting previously resolved references genomes.

The traditional short read *de novo* assemblers are single-threaded applications designed to run on a single processor. However, these assemblers cannot process the total number of base pairs

sequenced by massively parallel sequencing platforms, such as the Illumina, Inc. Genome Analyzer or Applied Biosystems SOLiD System, due to the computation time and memory constraint limits.

One advantage of the ABySS compared with other short read *de novo* assemblers is that it allows a distribution of the assembly algorithm in a parallel way (master/worker architecture). It is possible because it uses a distributed representation of a *de Bruijn* graph [16]. This kind of graph is a simple way to represent DNA sequences where every read corresponds to a vertex, and two vertices are connected by an edge if the corresponding reads overlap. The main feature of a *de Bruijn* graph is to allow the representation of sequence regions that repeat as a unique edge. Thus, the problem of fragment assembly is reduced to find a path visiting every edge of the graph exactly once.

The ABySS algorithm proceeds in two phases. The first phase loads short read sequences, breaking each into k-mers of length k. Then, it finds adjacent (overlapping) k-mers and it removes k-mers resulting from read errors. In the second phase, mate-pair information is used to remove variant sequences (bubbles) and to generate contiguous sequences (contigs).

However, there are reports that indicate problems in ABySS algorithm scalability. The costly communication and load-balancing inefficiencies emerge from larger scale. It is because the algorithm executes master-worker model with asynchronous message/work complexes queues, and uses MPI P2P and collective communication that varies by execution stage. Furthermore, there are extreme run-to-run execution time variations (which is likely to be mostly due to file I/O).

4.2 Analyzing the ABySS I/O properties

Considering the `ABYSS-P` description a first analysis is performed to identify the I/O phases. We select VampirTrace to start with a small test and Darshan for larger number of MPI processes and I/O workload. VampirTrace tool allows us to observe the full behavior of the application and Darshan tool extracts the I/O patterns.

We start the analysis of ABySS-P for a small test by using 4 MPI processes, to assemble paired reads in two files named `reads1.fastq` and `reads2.fastq` into contigs in a file named `test-bubbles.fa`; and k=25.

The test case is shown in Figure 2. This small case allows us to identify two I/O phases at the beginning of the running and at the ending. Figure 2(a) shows the timeline for the 4 MPI processes. In Figure 2(b) a trace view for each process is depicted, where row 1 corresponds to user functions and row 2 represents the I/O events (yellow bars)

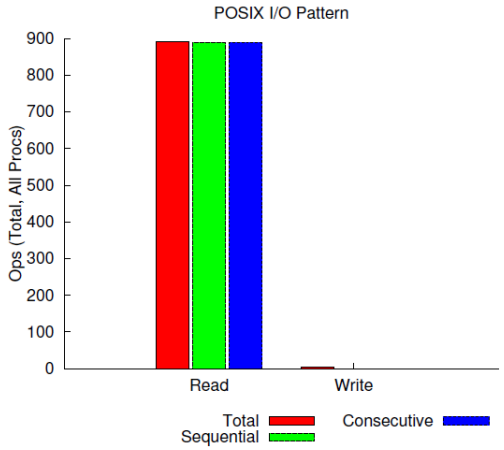


Figure 3: Access mode for ABYSS-P using 4 MPI processes. Sequential: An I/O operation issued at an offset greater than where the previous I/O operation ended. Consecutive: An I/O operation issued at the offset immediately following the end of the previous I/O operation.

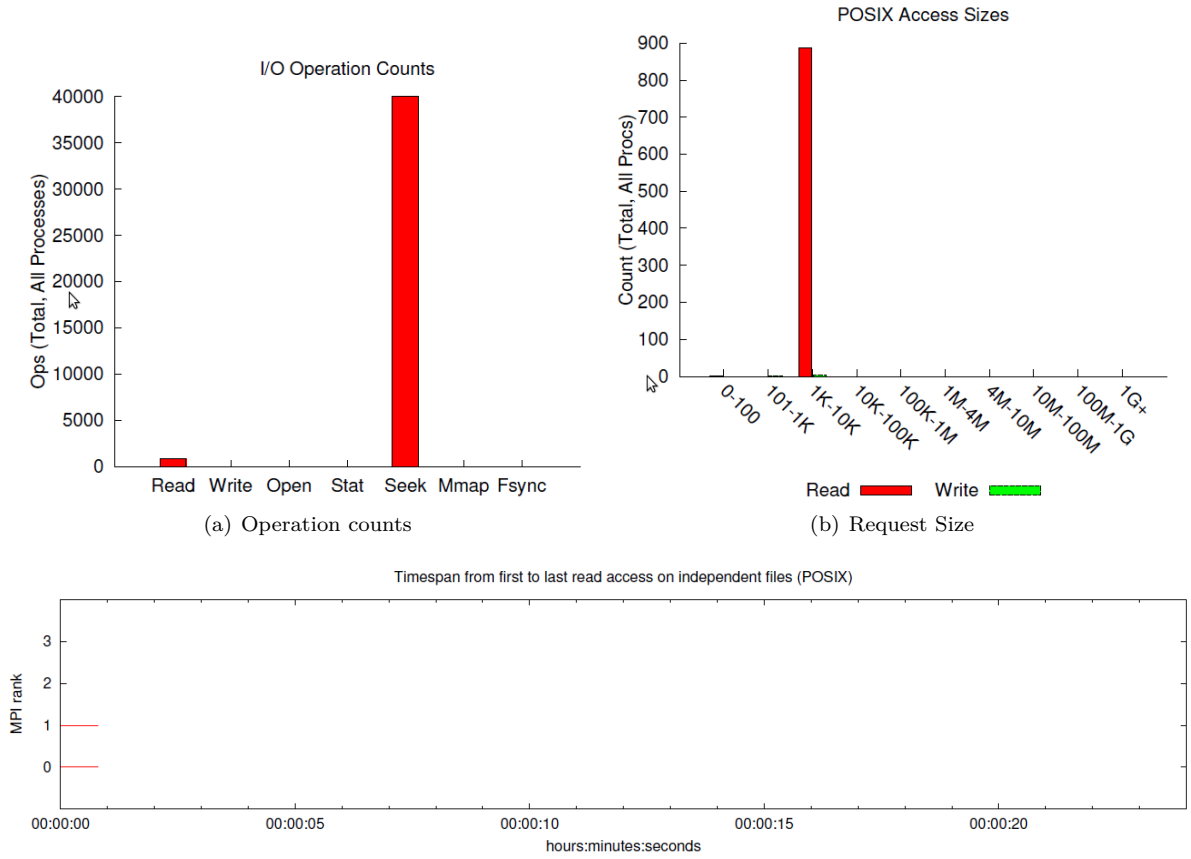
and MPI events (red bars), green triangles represent the file open and close, and the red/green triangles represent multiple I/O event at the same time interval. In the first I/O phase, the processes with rank 0 and 1 perform I/O activities; and in the second phase (at running end) all MPI processes write to output files.

To obtain detailed information about the I/O pattern in each phase we analyze ABySS using Darshan tool. A total of eleven files are used by ABYSS-P that corresponds to 2 files per each MPI process, two read files and 1 output file. I/O patterns are represented by the sequential access mode (see Figure 3) with mainly consecutive accesses for read operations, small request size (see Figure 4(b)), read and seeks operations (see Figure 4(a)) that correspond to operations into the input files. The I/O is serial by using the C I/O library.

Figure 4(c) depicts the temporal pattern for `ABYSS-P` where we can observe that only two first processes perform read operations into `reads1.fastq` and `reads2.fastq` files. Write operations take a short time, so for this reason we cannot observe them in the timeline.

Once the pattern for this case has been analysed, we need to define the general phases and the I/O pattern that will be applied to evaluate ABYSS-P for real input files and number of MPI processes. To do this, we select the Darshan tool due to its light overhead and small trace file size compared to other tracing tools.

ABYSS-P uses a total of $np * 2$ write files, two input files and 1 output file, where np is the number of MPI processes for the parallel execution. Input files are read in a first phase where only the



(a) Operation counts

(b) Request Size

(c) Temporal Pattern ABYSS-P using 4 MPI processes. The y-Axis represents the MPI ranks and the x-Axis the time in hours:minute:seconds. A total of 11 files was open but only the read files are shown because are representing the I/O with more impact on the performance.

Figure 4: ABYSS-P's I/O characteristics reported by Darshan tool for 4 MPI processes and test input files.

Table 1: Read Phase for ABYSS-P by using **np** MPI processes for two real input files. Two files of 17 GiB are read by two I/O processes (I/O proc) that corresponds to rank 0 and rank 1. Request size (rs) and access mode are independent of the input files.

np	**rs(B)**	**I/O proc**	**File per I/O proc**	**#iop**	**Data x I/O proc**	**Phase Data**
24	8191	2	1	2135744 x 2	17 GiB	33 GiB
32	8191	2	1	2135744 x 2	17 GiB	33 GiB
64	8191	2	1	2135744 x 2	17 GiB	33 GiB
128	8191	2	1	2135744 x 2	17 GiB	33 GiB
512	8191	2	1	2135744 x 2	17 GiB	33 GiB

rank 0 and 1 read a file each one independently of number of MPI processes. The Read phase moves more than 90% of the data and I/O operations of ABYSS-P. Table 1 shows detailed information by using different number of MPI processes only focus on read phase.

4.3 Replicating the ABYSS-P's I/O properties

We design a set of experiments to replicate the ABYSS-P's I/O properties by using the IOR benchmark (IOR 2.10.3) and only focus on the read phase showed in Table 1.

IOR benchmark is configured as follow:

```
mpirun -np NP ./IOR -r -N 1 -a POSIX
-b 17496014848 -t 8192 -k
```

Where:

- `NP` represents the number of MPI processes to evaluate.
- `-r` selects only read test because we are only analyzing the ABYSS-P's read phase.
- `-N` sets up the number of I/O processes, in this case 1.
- `-a` sets up the I/O library.
- `-b` defines the contiguous data in bytes to read for each I/O process that corresponds to the total size of an input file of ABYSS-P.
- `-t` defines the request size in bytes that in this case is 8192 because IOR does not allows the value 8191.
- `-k` indicates that the read file has not be removed when the test finishes.

To replicate the I/O concurrency into two input files, two concurrent executions of IOR with the previous configuration are run on different configurations of the I/O system.

4.4 Deploying a virtual cluster

In order to deploy the virtual cluster we use the StarCluster tool. As currently StarCluster does not deploy a HPC cluster with a parallel filesystem, a plugin has been developed to be integrated with the StarCluster tool, which allows us to deploy the PVFS2 file system automatically.

This plugin has been integrated using the standard plugin method provided by StarCluster. To configure the PVFS2 file system, the plugin provides a configuration file, as is shown in Figure 5. In this configuration file, users can choose between two installation ways: *regular* and *expert* mode.

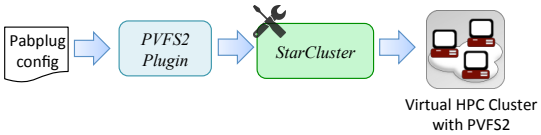


Figure 5: Integrating the pvfs2 plugin with StarCluster

Table 2: Description of the CACAU cluster

Components	**CACAU**	
	CACAU0	**CACAU1**
Compute Nodes	20	3
Processor (per node)	2 Intel Xeon E5430 (2.66GHz)	2 Intel Xeon E5-2400 (2.40 GHz)
CPU cores (per node)	8	12
GPU cores (per node)	-	2498
RAM Memory (per node)	16 GB	48 GB
Local FileSystem	ext4	ext4
Capacity of Local Storage (per node)	160 GB	500 GB
Device Type of Local Storage	HDD	HDD
MPI Library	OpenMPI 1.10.2	
Operating System	RedHat Linux Server 6.3	
Global FileSystem	NFS v4	
Capacity of Global FileSystem	2 TB	
Storage Total	8 TB	
Storage Network	4 Gbps Ethernet	

Using the *regular* mode, users only have the possibility to specify the I/O nodes and the metanode, the rest of pvfs2 parameters are set up by default. On the other hand, using the *expert* mode, users have full control of the plugin. In this way, users can specify all configuration parameters of PVFS2 explained in [17]. Moreover, users can specify the placement of the MDS and DFs into HPC cluster architecture. In this case we define two placements for the DFs: 1) a DF defined on each compute node of the cluster or, 2) DFs defined on independent nodes, which means that these will only be I/O nodes.

4.5 Configuring the different I/O subsystems on Cloud

A set of experiments is designed by using the IOR configuration defined in Section 4.3, and these experiments are executed in the following scenarios:

- NFS: i) Mapping an I/O process in differ-

Table 3: Characteristics of the Amazon's Instances Selected

Instances	Processor	vCPUs	RAM (GiB)	Storage (GB)	Network	$xHora
t2.medium	Intel Xeon (Turbo up to 3.3GHz)	2	4	only EBS	Low to Moderate	0.052
c3.2xlarge	Intel Xeon E5-2680 v2 (Ivy Bridge)	8	15	2x80 (SSD)	High	0.42

ent compute nodes and ii) Mapping two I/O processes on the same compute node.

- PVFS2: i) Mapping an I/O process in different compute nodes and ii) Mapping two I/O processes on the same compute node. Each one on different data layout:
  - Each input file placed in 1 different DF.
  - Each input file stripped into different numbers of DFs.

The I/O processes mapping is considered because each process reads a 17 GB file, which has an impact on the memory utilization and it can impact on the application performance, especially for compute nodes with less than 33 GB of RAM.

To select the instance type for the testing, we try to use an instance with similar characteristics to the compute nodes of the CACAU cluster. The CACAU features are shown in Table 2, which is composed of two segments. We focus on the CACAU0 segment.

We create clusters with six nodes: 1 master and five compute nodes. The AWS EC2 instance similar to the CACAU0 segment is the c3.2xlarge (see Table 3). We also select the t2.medium (see Table 3) to check it is possible to demonstrate the hypothesis working with instances that have lower features. Two Virtual Clusters are configured with these two types of instances that are described in Table 4.

For each VCC we use 6 instances to evaluate NFS and PVFS2 file systems. For PVFS2 we use a maximum of 3 DFs be able to evaluate the two DFs placement on the HPC-VCC's architecture.

4.6 Analyzing the I/O behavior on different scenarios

We evaluate the I/O kernel of the ABYSS-P in the scenarios defined in section 4.5 and we obtain different performance metrics on PVFS2 and NFS. In this case, we present the result of data transfer rate and I/O operations per second (IOPs). Figure 6(a) shows results by using instance t2.medium. The x-axis represents the different configurations for the file system, PVFS2 by using different number of DFs and NFS. The mapping an I/O process in different compute nodes is represented by the label 1PxCN, and 2PxCN for the mapping the two I/O processes on the same compute node.

Figure 6(b) presents results for the c3.2xlarge instance. In this case, the configuration PVFS-1DFx1F means that each input file is striped to 1 DF but different DF for each one.

The results show that NFS reports more data transfer rate than PVFS. This result is the same for the different instances and it can be observed that the bandwidth does not improve using more DFs.

On the NFS file system, the mapping is impacting on data transfer rate and this is more significant for VCC with compute nodes with 4 GiB RAM. On PVFS2, we can observe an improvement for 1PxCN. The data layout 1DFx1F reports more performance than other PVFS2 I/O configurations.

Discussion ABYSS-P is a parallel application that is reading from two files by using an I/O process for each file. This behavior is the same independently of the number of MPI processes. This I/O pattern cannot take advantage of the performance capacity provided by a parallel filesystem like PVFS2. The results in Figure 6 show this problem. Although we increase the number of DFs, the I/O performance presents similar values.

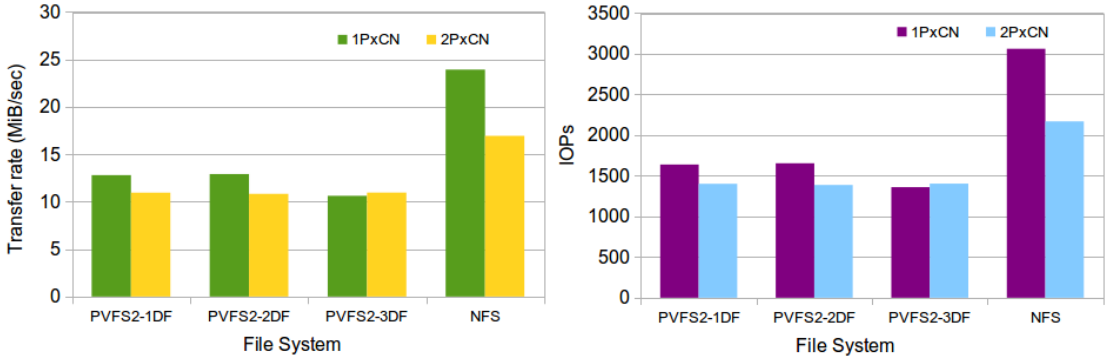
Furthermore, the small request size impacts on the performance of PVFS2 when only one process is carrying out I/O. We can observe that NFS can provide an acceptable I/O performance. However, we must to take care in this point because the NFS filesystem for this kind of application should be different to the NFS for home user accounts.

Moreover, we have not evaluated the impact that the stripe size and the number of MDS can have. Other parallel file systems like Lustre or GPFS show more bandwidth for this kind of I/O pattern. However, in this work, we select PVFS2 for the simple installation and free distribution.

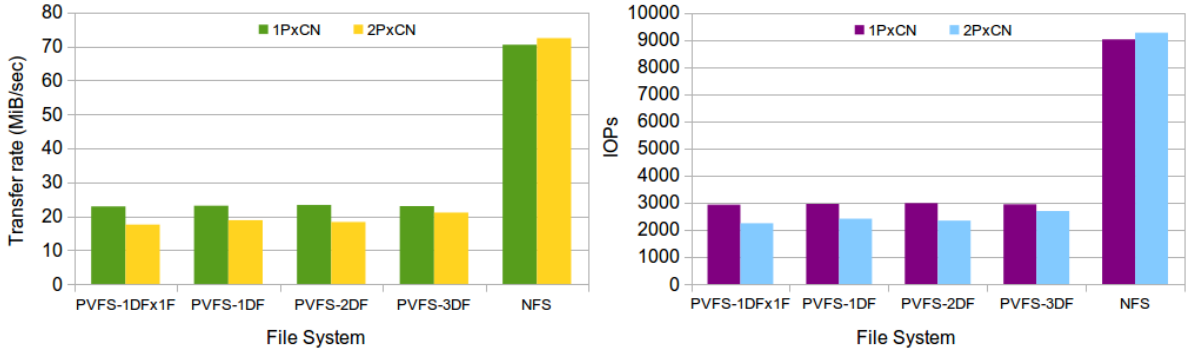
During the process of the I/O pattern extraction needed to define the I/O kernel of ABYSS-P, we have observed that the I/O pattern is a problem for the scalability. Due to the fact only two processes are reading the input files and sending read data to the rest of the processes, this is clearly an inefficient I/O pattern and an obstacle for the ABYSS-P scalability.

Table 4: Descriptive Characteristics of the Virtual Clusters on cloud (VCC) configured equal for the experiments.

I/O components	VCC 1	VCC 2	VCC 3	VCC 4
Instance	t2.medium	t2.medium	c3.2xlarge	c3.2xlarge
Number of Instances	6	6	6	6
Temporal Storage	-	-	Ephemeral	Ephemeral
Persistent Storage	EBS	EBS	EBS	EBS
Temporal Device	-	-	SSD	SSD
Persistent Device	SSD(GP 192/3000)	SSD(GP 300/3000)	SSD(GP 192/3000)	SSD(GP 300/3000)
Capacity of Persistent Storage	100GB	64GB	100GB	64GB
File system Local	ext4	ext4	ext4	ext4
File system Global	NFS	PVFS2 (2.8.8)	NFS	PVFS2 (2.8.8)
Parallel Storage Capacity	-	500GB	-	400GB
Number of data servers	-	5	-	5
Number of Metadata Server	-	1	-	1
Stripe Size	-	64KB	-	64KB
MPI library	mpich-3.2	mpich-3.2	mpich-3.2	mpich-3.2



(a) Instance t2.medium



(b) Instance c3.2xlarge

Figure 6: Data transfer rate and IOPs for IOR configured to ABYSS-P's I/O Kernel.

Table 5: Description of the CAPITA cluster

Components	CAPITA	
	CAP0	CAP1
Compute Nodes	13	13
CPU cores (per node)	4	4
RAM Memory	16GB	16GB
Capacity of Local Storage (per node)	46GB	46GB
Device Type of Local Storage	SSD	SSD
Local Filesystem	Linux ext3	Linux ext3
Global Filesystem	NFS	PVFS2 (2.8.7)
Capacity of Global FileSystem	46GB	175GB
MPI-Library	mpich2 (1.5)	mpich2 (1.5)
Number of data servers	-	4
Number of Metadata Server	-	1
Stripe Size	-	64K
Network	1 Gbps Ethernet	1 Gbps Ethernet

5 Experimental Validation

In this section, we validate the results obtained in a Cloud environment by replicating the experiments defined in Section 4.5, in the physical cluster CAPITA. This is described in Table 5, the NFS and PVFS2 file spaces are configured using SSD devices. The NFS file space is a special partition only used for intensive I/O that is different to the NFS used as HOME file space. Due to physical nodes being more comparable to instances C3.2xlarge, we focus on the validation comparing the CAPITA results with VCCs experiments that use instances C3.2xlarge.

In previous IOR experiments on a Cloud environment (see Section 4.6), we have shown that the PVFS2 provides less I/O performance than NFS. These results are also observed in the CAPITA cluster. Experimental results are shown in Figure 7, where the same behavior for the ABYSS's I/O kernel both in VCCs (see Figure 6(b)) as in the CAPITA cluster can be observed. Therefore, we can say that the Cloud environment can be used as Test-Bed infrastructure as long as the virtual cluster is designed considering the main components of the physical cluster and the requirements of the application's I/O kernel.

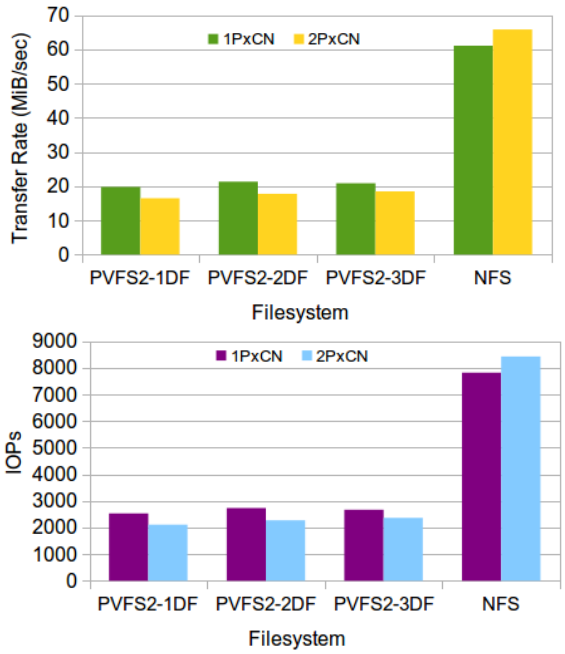


Figure 7: Data transfer rate and IOPs for IOR configured to ABYSS-P's I/O Kernel. Results obtained in CAPITA cluster.

6 Conclusion

In this paper, we have analyzed the real parallel application ABYSS-P on different configurations of the I/O subsystem without changing the I/O system of CACAU physical cluster.

In Cloud environments the time consumed by the instances is fundamental because we pay for the time, for this reason we only evaluate the ABYSS-P's I/O kernel. This is obtained in a physical cluster and it can be executed on different HPC systems.

Our work shows that the Cloud allows different I/O configurations to be deployed automatically and their influence on the I/O performance of the parallel application to be evaluated quickly.

From this evaluation we can suggest the user tries to modify the I/O pattern because it will not be able to take advantage of a parallel file system.

On other hand, the administrator must try to allocate the application on compute nodes with more memory, especially for the I/O processes that usually need more memory for the input files.

Experimental results showed that the parallel file system selected using different configurations has less performance than NFS. The mapping an I/O process per compute node on a NFS file system was the most suitable configuration for this kind of pattern.

Acknowledgements

This research has been supported by the MINECO (MICINN) Spain under contract TIN2014-53172-P and partially supported by the CloudMas as Government Competency of AMAZON Web Services (AWS). The research position of the PhD student P. Gomez has been funded by a research collaboration agreement, with the "Fundación Escuelas Universitarias Gimbernat". The authors thankfully acknowledge the resources provided by the Centre of Supercomputing of Galicia (CESGA, Spain) and the Leibniz Supercomputing Centre (LRZ, Germany).

We thank Eduardo Almeida Costa, PhD student of the Program in Biology and Biotechnology of Microorganisms at the State University of Santa Cruz (Brazil), who provided us with the input data for the ABySS application. We would also like to show our gratitude to Pawel Wichary for collaborating in the implementation of the PVFS2 plugin.

References

[1] "Starcluster by mit." http://http://star.mit.edu/. Accessed: 2016-05-02.

[2] N. Miller, R. Latham, R. Ross, and P. Carns, "improving cluster performance with pvfs2," *ClusterWorld Magazine*, vol. 2, no. 4, 2004.

[3] M. Liu, J. Zhai, Y. Zhai, X. Ma, and W. Chen, "One optimized i/o configuration per hpc application: Leveraging the configurability of cloud," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, pp. 15:1–15:5, 2011.

[4] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Data sharing options for scientific workflows on amazon ec2," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–9, 2010.

[5] R. R. Expósito, G. L. Taboada, S. Ramos, J. González-Domínguez, J. Touriño, and R. Doallo, "Analysis of i/o performance on an amazon ec2 cluster compute and high i/o platform," *Journal of grid computing*, vol. 11, no. 4, pp. 613–631, 2013.

[6] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pp. 4–16, 2009.

[7] J. Zhai, M. Liu, Y. Jin, X. Ma, and W. Chen, "Automatic cloud i/o configurator for i/o intensive parallel applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3275–3288, 2015.

[8] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pp. 18:1–18:14, 2011.

[9] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *Trans. Storage*, vol. 7, pp. 8:1–8:26, Oct. 2011.

[10] W. Loewe, T. MacLarty, and M. C, *IOR Benchmark*, 2012. Accessed: 2016-05-14.

[11] 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, pp. 501+, 2005.

[12] Prabhat and Q. Koziol, *High Performance Parallel I/O*. Chapman & Hall/CRC Computational Science, 1 ed., 2014.

[13] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: a parallel assembler for short read sequence data," *Genome research*, vol. 19, no. 6, pp. 1117–1123, 2009.

[14] CACAU, "Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas," tech. rep., Universidade Estadual de Santa Cruz, 2016.

[15] LRZ, "Leibniz Supercomputing Centre," tech. rep., Bayerischen Akademie der Wissenschaften, 2016.

[16] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to dna fragment assembly," *Proc Natl Acad Sci USA*, vol. 98, pp. 9748–53, Aug. 2001.

[17] PVFS2, "Config File Description," tech. rep., PVFS2, 2016.