Cloud, a flexible environment to test HPC I/O configurations

Pilar Gomez-Sanchez[1], **Sandra Mendez**[2], **Javier Panadero**[1,3]
Aprigio Bezerra[4], **Dolores Rexachs**[1] **and Emilio Luque**[1]
[1]Computer Architecture and Operating Systems Department,
Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain
[2]High Performance Systems Division, Leibniz Supercomputing Centre (LRZ),
85748 Garching near Munich, Germany
[3]IN3 - Computer Science Department, Open University of Catalonia, Barcelona, Spain
[4]Departamento de Ciencias Exactas e Tecnológicas, Universidade Estadual de Santa Cruz, Bahia, Brasil
pilar.gomez@uab.es, sandra.mendez@lrz.de, javier.panadero@uab.es
aalbezerra@uesc.br, dolores.rexachs@uab.es, emilio.luque@uab.es

Abstract— *The Infrastructure as Service (IaaS), provided by cloud, has awoken interest among the scientific community over High Performance Computing (HPC) because Cloud offers the possibility to access resources with different features. In this paper, we show that Cloud can be considered an alternative for executing an application without waiting time until the system assigns the resources and it can act as a HPC I/O Test Bed environment. This platform allows us to change different components of the I/O stack and to evaluate the results without affecting the production environment or the workload. Finally, we present a methodology to evaluate the configurations of the I/O System and a methodology to guide user decisions when configuring the I/O parallel system in a virtual cluster in Cloud.*

Keywords: Cloud, I/O System, parallel I/O, I/O Phase

1. Introduction

In many research areas, there is the necessity to have a test environment *(Test Bed)* to evaluate proofs of concept, validate ideas, try implementations or configurations, or execute an application without affecting the production environment of High Performance Computing (HPC) systems. In many cases, the users and administrators have simple, basic and enclosed test environments for carrying out the first estimation, validation and/or testing of an idea and Cloud provides them with the possibility of creating different environments where it may be possible to carry out tests. Therefore, different authors have analyzed and evaluated the cloud platform to verify if it can be an alternative to traditional clusters and to check the suitability of this platform for running scientific applications, particularly High Performance Computing (HPC) applications. In this context, He et al. [1] present a case study where they analyze different benchmarks such as NAS Parallel Benchmark (NPB) on different cloud platforms such as Amazon EC2, GoGrid, and IBM Cloud. They conclude that the cloud platform can be an alternative for researchers that do not have applications to execute on supercomputers but they cannot execute these applications on a personal computer. In the same line of thought, Niu et al. [2] consider that the cloud platform can be an alternative for medium-scale or large-scale clusters. For this reason, they propose a Semi-Elastic Cluster (SEC) computing model which allows research groups to reserve and dynamically re-size a virtual cluster on cloud (Amazon EC2), to be shared by many users belonging to the same research group.

The cloud platform offers a lot of opportunities for researchers because they can create a cluster (public or private) on this platform, configure the I/O system and install different filesystems [3] without affecting the work of other users or the performance of other applications. In contrast, this is a difficult task in a traditional cluster, due to the changes that must be authorized by the administrator, or due to the need for administrator permissions when it is working in a production environment.

In this paper, we show how the cloud platform can be an alternative as an environment to test different I/O configurations and to execute parallel scientific applications using parallel I/O libraries. Given that the cloud platform provides different resources, the user has a lot of parameters to select and values to assign in order to configure the I/O system. We present a methodology to guide the user in the selection of storage resources on the cloud platform, to execute applications and we have adapted it when this platform is used as a *Test-Bed*. The proposed methodology is based on using the application I/O behavior information to carry out the tests during the system configuration. In our case, this behavior information is obtained with PIOM.

This article is organized as follows. Section 2 introduces our methodology to analyze the applications. In Section 3, we present the methodology to guide the user to work in cloud. In Section 4, we show the experimental results. Finally, in Section 5, we present the conclusions.

2. Application analysis

A real parallel application can be executed on the cloud platform but there are several drawbacks such as the application execution time and the number of resources that are needed. In this section, we present the methodology that we use to analyze the MPI parallel applications at I/O level.This is because our interest is focused on the behavior of I/O parallel message passing (MPI) applications to attempt to analyze the impact of these applications on the I/O system and vice-versa.

2.1 Application I/O behavior model: PIOM

Firstly, it is necessary to know the parallel application I/O behavior. To that end, we have proposed PIOM, which is a model that allows us to show and understand the application I/O behavior (at MPI-IO level [4] and at POSIX-IO level PIOM-PX [5]) and to extract the application I/O requirements.

PIOM analyzes every file opened by the application and detects the I/O phases of access to every file. PIOM is a model based on the I/O phase concept, where an I/O phase is I/O operations consecutive sequence in a file. In Table 1, all the necessary characteristics are shown that we have considered to identify/define an application and for this reason, they are represented in the PIOM model to obtain the application I/O behavior. The characteristics have been classified depending on the level on which the information is provided: application, file and I/O phase.

Once the application I/O behavior has been obtained using PIOM, monitoring the application when it has been executed once in the cluster, a synthetic program with this I/O behavior is replicated. As this behavior is independent of the system in which the application is monitored, it can be used to replicate it in other systems. From this synthetic program, the I/O kernel of the parallel application in different HPC systems can be executed; which in our case is the cloud platform. The advantage of using the synthetic program is that it is not necessary to move the real data of the application or to execute the compute of application.

Depending on the analysis type to which the user wants to subject the application, it is only possible to replicate the more significant I/O phases, which have more impact on the application I/O behavior.

To analyze the results, there are different metrics to be considered such as data transfer rate, I/O time, I/O operations per second, performance and cost.

3. Building a Virtual Cluster on Cloud

No matter what the analysis goal of the application is, it is necessary to create the Virtual Cluster on Cloud (VCC) and the storage system to execute the application with I/O. The cloud platform provides a set of different instances that the user has to select. The instance selection is not trivial because an instance type is comprised of a combination of CPU, memory, storage (temporary and/or permanent), and networking capacity. For this reason, we have proposed a methodology (see Figure 1) to guide the user to select, create and configure a VCC.

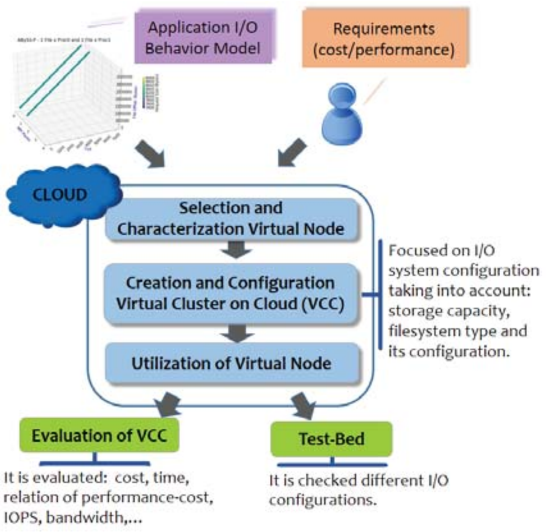

Application I/O Behavior Model
Requirements (cost/performance)
CLOUD
Selection and Characterization Virtual Node
Creation and Configuration Virtual Cluster on Cloud (VCC)
Utilization of Virtual Node
Focused on I/O system configuration taking into account: storage capacity, filesystem type and its configuration.
Evaluation of VCC
It is evaluated: cost, time, relation of performance-cost, IOPS, bandwidth,...
Test-Bed
It is checked different I/O configurations.

Fig. 1: Methodology to build an HPC-I/O subsystem for an application that it is executed in a VCC.

To create a VCC, the user has to take into account the application I/O requirements and the user requirements (the cost that the user can spend or the time that the user can wait). We have presented a methodology to guide the user in resource selection to configure the HPC-I/O subsystem,

- Using PIOM to obtain the application requirements and the application I/O behavior model.
- Selecting the instance, the user has to take into account the I/O node characteristics that he will employ in the cluster (reference system) to look for similar instances to create the VCC and validate the similarity with IO-zone [6] for the I/O patterns observed in the application.
- Creating the VCC, taking into account the computing needs, the strategy followed is 1 process per core. For the I/O, different configuration changes are carried out such as resources (number of instances, instance types, dedicated or shared instances) and customer mapping.
- Installing the filesystem (NFS, PFS, number of data servers of PFS).
- Characterizing the HPC-IO subsystem using a synthetic program.

From the point of view of hardware architecture, the analysis possibilities use temporary or permanent storage, independent nodes to do I/O and computing or share the nodes to do I/O and computing.

Table 1: PIOM Model Characteristics

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

3.1 Characterization and selection of node or instance to build the Virtual Cluster.

We apply IOzone benchmark to obtain the average values for the transfer rate at local filesystem level. IOzone is a filesystem benchmark tool that generates and measures a variety of file operations. We only analyze the write/rewrite and read/reread operations.

Taking into account the application I/O requirements obtained with PIOM, in particular, the request size (rs) that the application uses is obtained. This information allow us to evaluate the instance in a specific range determined by the request size used by the application. As a consequence, the evaluation time of instances is reduced. This step allows us to estimate the bandwidth that can be obtained from 1 data server.

3.2 Creation of the Virtual Clusters

A Virtual Cluster is represented by the components shown in Table 2. The components that can be selected by the user are identified with (*), the components that the user must configure manually are marked with (-), and the components that the user cannot change because they are default, depending on instance type, are indicated with (+) [7].

The baseline software on a VCC for each compute node depends on the Machine Image selected. Similar to physical HPC systems, the Linux operating system is more commonly used in the HPC on cloud, especially for the I/O Software Stack. The previous components such as instance type and storage capacity allow a decision to be made considering the I/O requirements for the application. However, as the user pays for the utilization, the cost is one of the main restrictions to the creation of a Virtual Cluster.

Table 2: Components of Virtual Cluster configurable

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

3.3 Configuration of the HPC I/O subsystem in the Virtual Cluster

When they use shared files, parallel applications need a global file system to share the input and output data. Usually the HPC-I/O subsystem must be configured by the user in

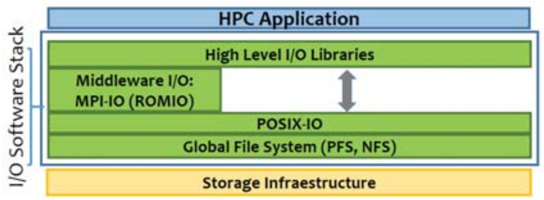


Fig. 2: I/O Software Stack

a cloud environment. The I/O Software Stack (see Figure 2) is comprised of a Scientific Data Library (like HDF5 or NetCDF), I/O Middle-ware (like MPI-IO), POSIX-IO and Parallel File System (like GPFS, Lustre or PVFS2) [8].

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 or PVFS2[9].

In general, parallel file systems have several parameters that affect the performance.

3.4 Deploying a Virtual Cluster

In order to deploy the virtual cluster, we use the StarCluster tool [10]. This tool helps users and system administrators to create a virtual cluster using instance of Amazon's EC2 platform [11]. StarCluster deploys a Linux cluster with NFS, while the user must install and configure the parallel file system.

Firstly, we select a small instance such as t2.micro. From an image (AMI) of StarCluster such as ami-6b211202, a little cluster is created to install the software that the user needs and it is checked that all software runs correctly.

To configure the PVFS2 file system, users must specify all configuration parameters of PVFS2, as explained in [12]. They must also specify the placement of the MDS and DFs within 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.

Secondly, a new image is created. Finally, with the new AMI and the instance selected by the user, the VCC is created which is used as a Test-Bed or virtual cluster.

4. Experimental Results

When the user needs to analyze how a change in the user's HPC I/O subsystem can affect a parallel application, they can use cloud to carry out this analysis before modifying the production cluster.

We present the evaluation of the parallel application ABySS (Assembly By Short Sequences) [13], which reports slow running using NFS on the CACAU [14] (reference system).The user provides us with the input data and specific parameters to evaluate ABYSS.

In this work, we focus on comparing the configuration impact with NFS and PVFS2[15] to analyze if PVFS2 is an alternative to NFS. PVFS2 is an open source file system developed to support efficient read and write operations for large amounts of data. PVFS2 is designed as a client-server architecture where the server provides the storage and the client has 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.

4.1 ABySS Application Description

ABySS is a parallelized sequence assembler. 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 genome references. ABySS enables a distribution of the assembly algorithm in a parallel way (master/worker architecture). This is possible because it uses a distributed representation of a *de Bruijn* graph [16].

However, there are reports that indicate problems in ABySS algorithm scalability. Costly communication and load-balancing inefficiencies emerge from the larger scale. This is because the algorithm executes a master-worker model with asynchronous message/work complex queues, and it uses MPI P2P and collective communication that varies by execution stage.

Analyzing the ABySS I/O properties

We perform an I/O pattern analysis in LRZ's HPC systems [17] where PIOM is installed and we define a set of different configurations of the I/O system to be implemented in a cloud environment. ABySS-P uses a total of $np*2$ temporary write files, two input files and one 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 rank 0 and 1 read a file, each one independently from the number of MPI processes. The read phase moves more than 90% of the data and I/O operations of ABYSS-P. Table 3 shows detailed information by using a different number of MPI processes which only focus on the read phase. Two files of 17 GiB are read by two I/O processes (I/O proc) that correspond to rank 0 and rank 1. Request size (rs) and access mode are independent of the number of MPI processes.

Figure 3 depicts the temporal pattern for `ABySS-P`, where we can observe that only the first two processes perform read operations in the two input files (`reads1.fastq` and `reads2.fastq` files, test input files). The application is executed by 4 processes but only the processes to rank 0 and 1 read the test input files. It can be observed that PIOM shows the event order ($Tick$) of I/O operations and the file offset. The colour corresponds to the request size of I/O operations. Write operations take a short time, so for this reason we cannot observe them on the timeline.

Table 3: Read Phase for ABYSS-P by using **np** MPI processes for two real input files.

np	rs(B)	I/O proc	File x I/O proc	#iop	Data x I/O proc (GiB)	Phase Data Data (GiB)
4	8191	2	1	2135744 x 2	17	33
32	8191	2	1	2135744 x 2	17	33
64	8191	2	1	2135744 x 2	17	33
128	8191	2	1	2135744 x 2	17	33
512	8191	2	1	2135744 x 2	17	33

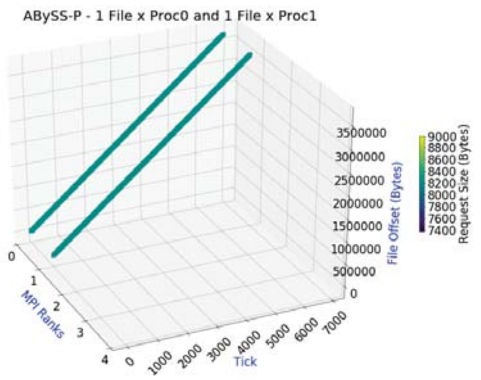


Fig. 3: ABYSS-P's I/O characteristics reported by PIOM tool for 4 MPI processes and test input files.

Once the pattern for this case has been analyzed, we need to define the general phases and the I/O pattern that will be applied to replicate the behavior of ABYSS-P. Table 4 shows the concept at phase level for the two real input files.

Table 4: ABySS Phases for the input files.

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

We use IOR benchmark [18] as a synthetic program. However, it cannot replicate all applications' I/O patterns. IOR (IOR 2.10.3) was customized to replicate ABYSS-P where each process reads a file of 17 GB. However, moving this data volume to the Cloud impacts on time and cost. For this reason, we use IOR because it allows us to replicate the application I/O behavior, omitting the computing and communication time. For this application, we only focus on the read phase (see Table 3).

The IOR benchmark is configured as follows:

```
mpirun -np 2 ./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. In this case, POSIX because this application does not use MPI-IO functions.
- `-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. In this case, 17496014848 bytes.
- `-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 in two input files, two concurrent executions of IOR with the previous configuration are run on different configurations of the I/O system.

4.2 Creation of VCC

To select the instance type for the testing, we try to use an instance with similar characteristics to the compute nodes of the CACAU0 cluster [14]. CACAU0 has 8 CPU cores per node and 16GiB RAM memory per node.

The AWS EC2 instance similar to the CACAU0 is the c3.2xlarge (see Table 5) because this instance has 8 CPU cores per node and 15 GiB of RAM memory. Two Virtual Clusters are configured with this type of instance that are described in Table 6.

For each VCC we use six instances (six nodes) to evaluate NFS and PVFS2 file systems. We create clusters with one master and five compute nodes and/or I/O nodes. This number of nodes allows us to change the configuration: to have the compute nodes and I/O nodes shared or to have the compute nodes independent of I/O nodes. The c3.2xlarge instance provides two SSD as temporal storage (*ephemeral*). We use a persistent storage to install the software and if we change the instance only we have to add this storage to the new instance. We do not need to install once again the software.

4.3 Configuring the different HPC I/O subsystems on cloud

A set of experiments is designed by using the IOR configuration defined in Section 4.1. To analyze the contention problem in the node we use two mappings: a process in different nodes (1PxCN) and two processes in the same node (2PxCN). These experiments are executed in the following scenarios:

Table 5: Characteristics of the Amazon's Instance Selected

Instances	Processor	vCPUs	RAM(GiB)	Storage(GB)	Network
c3.2xlarge	Intel Xeon E5-2680 v2 (Ivy Bridge)	8	15	2x80 (SSD)	High

Table 6: Descriptive Characteristics of the Virtual Clusters on cloud (VCC) configured equal for the experiments.

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

- NFS: NFS is in the master node. We use three nodes: two compute and one I/O node.
- PVFS2:
  - Each input file placed in a different DF (1DFx1F).
  - Each input file stripped into different numbers of DFs (1DF, 2DF, 3DF).

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

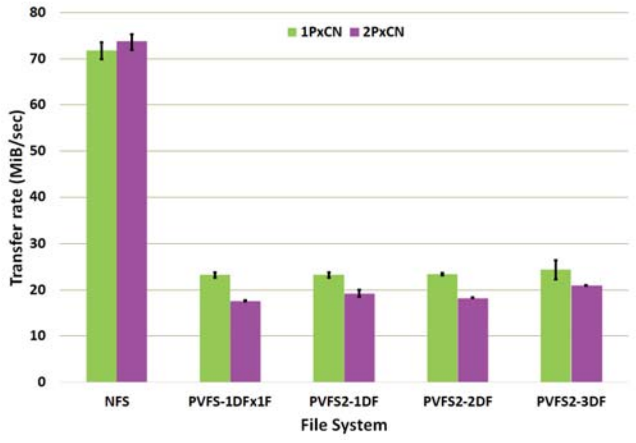
4.4 Analyzing the I/O behavior on different scenarios

We evaluate the I/O kernel of the ABYSS-P in the scenarios defined in section 4.3 and we obtain different performance metrics on NFS (filesystem that is using the reference system) and PVFS2. In this case, we present the result of the data transfer rate and I/O operations per second (IOPs). Figure 4 presents results for the c3.2xlarge instance. The x-axis represents the different configurations for the file system NFS and PVFS2 by using a different number of DFs. Mapping an I/O process in different compute nodes is represented by the label 1PxCN, and 2PxCN represents the mapping of the two I/O processes in the same compute node. The configuration PVFS-1DFx1F means that each input file is striped to one DF but different DFs for the two files.

The results show that NFS reports more data transfer rate and IOPS than PVFS and it can be observed that the bandwidth does not improve by using more DFs.

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



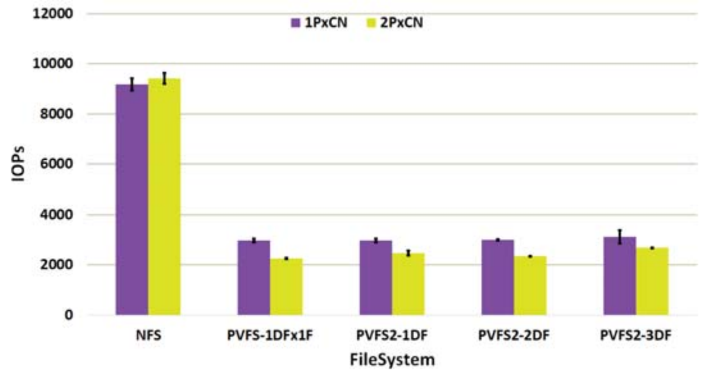


Fig. 4: Data transfer rate and IOPs for IOR configured to ABYSS-P's I/O Kernel using instance c3.2xlarge.

file. This behavior is the same independent 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 4 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 can have.

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 that 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.

5. Conclusions

The cloud platform has awoken interest among the High Performance Computing (HPC) scientific community because the IaaS service (Infrastructure as Service) allows access to different resources, enabling us to create and configure our own virtual cluster to execute, analyze and evaluate the parallel applications. We have shown that cloud is an alternative for users to evaluate the HPC I/O subsystem configurations without affecting the production environment of HPC systems.

6. Acknowledgements

This research has been supported by the MICINN/MINECO Spain under contracts TIN2014-53172-P and TIN2017-84875-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).

References

[1] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running hpc applications in public clouds," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 395–401. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851535

[2] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, "Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2013, pp. 1–12.

[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*. ACM, 2011, pp. 15:1–15:5.

[4] S. Méndez, D. Rexachs, and E. Luque, "A methodology to characterize the parallel i/o of the message-passing scientific applications," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 436.

[5] P. Gomez-Sanchez, S. Mendez, D. Rexachs, and E. Luque, *PIOM-PX: A Framework for Modeling the I/O Behavior of Parallel Scientific Applications*. Cham: Springer International Publishing, 2017, pp. 160–173.

[6] W. D. Norcott. (2006) IOzone Filesystem Benchmark. [Online]. Available: http://www.iozone.org/

[7] P. Gómez Sánchez, S. Méndez, D. Rexachs del Rosario, and E. Luque, "Hopes and facts in evaluating the performance of hpc-i/o on a cloud environment," *Journal of Computer Science & Technology*, vol. 15, 2015.

[8] 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.

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

[10] StarCluster. (2014) An Open Source Cluster-Computing Toolkit for Amazon Elastic Compute Cloud (EC2). [Online]. Available: http://star.mit.edu/cluster/

[11] AWS-EC2. (2017) Amazon Elastic Compute Cloud, Instance Types. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

[12] PVFS2, "Config File Description," PVFS2, Tech. Rep., 2016. [Online]. Available: http://www.pvfs.org/cvs/pvfs-2-8-branch-docs/doc//pvfs-config-options.php

[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," Universidade Estadual de Santa Cruz, Tech. Rep., 2016. [Online]. Available: http://nbcgib.uesc.br/nbcgib/

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

[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, no. 17, pp. 9748–53, Aug. 2001.

[17] LRZ, "Leibniz Supercomputing Centre," Bayerischen Akademie der Wissenschaften, Tech. Rep., 2016. [Online]. Available: http://www.lrz.de/services/compute

[18] W. Loewe, T. McLarty, and C. Morrone. (2012) IOR Benchmark. Accessed: 2016-05-14. [Online]. Available: https://github.com/chaos/ior/blob/master/doc/USER_GUIDE