View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Servicio de Difusión de la Creación Intelectual

Journal of Computer Science & Technology, Volume 19, Number 2, October 2019

- ORIGINAL ARTICLE -

IaaS Cloud as a Virtual Environment for Experimentation in Checkpoint Analysis

IaaS Cloud como Entorno Virtual de Experimentación en el Análisis del Checkpoint

Betzabeth León, Pilar Gomez-Sanchez, Daniel Franco, Dolores Rexachs and Emilio Luque

Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain

{betzabeth.leon, pilar.gomez, daniel.franco, dolores.rexachs, emilio.luque}@uab.es

Abstract

Cloud Computing offers the possibility of computing resources, allowing remote access to software, storage and data processing through the Internet. Infrastructures as a Service (IaaS), it is a flexible space which can be used as an experimental environment, in which experiments can be carried out similar to a real environment, such as in a cluster can be carried out. Before making installations and changes in a production cluster or select resource in the cloud, it is important to analyze the impact of this change. For this reason we propose using the cloud to carry out the study of previous viability. In this paper, we observe the viability of using the cloud to analyze the behavior of the Checkpoint as one of the Fault Tolerance strategies, establishing the differences that exist in the information generated in a real environment (cluster) and a virtual environment (cloud). The results obtained show that due to the variability of the cloud, the impact on the benefits cannot be analyzed. However, the cloud is suitable for extracting the spatial and temporal behavior pattern of the checkpoint, which helps to characterize it and this will help us to know the right configuration and the development of methodologies and tools that simulate and predict the execution of the checkpoint in a real environment.

Keywords: Checkpoint, Cloud Computing, Fault Tolerance.

Resumen

El Cloud Computing ofrece la posibilidad de recursos informáticos, que permiten el acceso remoto a software, almacenamiento y procesamiento de datos a través de Internet; IaaS, es un espacio flexible que se puede utilizar como un entorno experimental, en el que se pueden llevar a cabo experimentos similares a los de un entorno real, como un clúster. Antes de realizar instalaciones y cambios en un clúster de producción o de seleccionar recursos en el cloud, es importante analizar el impacto de este cambio. Por este motivo se propone utilizar la nube para realizar el estudio de viabilidad previa. En este documento observamos la posibilidad de utilizar la nube para analizar el comportamiento del checkpoint como una de las estrategias de tolerancia a fallos, estableciendo las similitudes y diferencias que existen en la información generada en un entorno real (clúster) y un entorno virtual (nube). Los resultados obtenidos muestran que, debido a la variabilidad de la nube, no se puede analizar el impacto en las prestaciones, pero la nube es adecuada para extraer el patrón de comportamiento espacial y temporal del checkpoint. Caracterizar el comportamiento del checkpoint ayudará a configurar el sistema, teniendo en cuenta los recursos extra que se necesitan y el impacto en función de la aplicación y los recursos seleccionados.

Palabras claves: Checkpoint, Cloud Computing, Tolerancia a Fallos.

1. Introduction

Cloud Computing offers a large amount of computing resources with maintenance, flexibility and easy access, allowing us to have contact with software, storage and infrastructure. These can be accessed at any time and from anywhere, as well as having special privileges, which are limited in a real cluster. For these reasons, as indicated in [1], the cloud infrastructure has awoken interest among the High Performance Computing (HPC) scientific community because the IaaS (Infrastructure as Service) allows access to different resources, enabling us to create and configure our own virtual cluster in order to execute, analyze and evaluate parallel applications.

On the other hand, in the HPC environment, it is important to use fault tolerance so as to maintain the availability of systems, anticipate emergency conditions, generate solutions to these conditions and make state recoveries. The most common failure modes include machine faults in which hosts go down and get rebooted, and network faults, where links go down. Finding a single monolithic solution for fault tolerance that is acceptable to all user applications is unlikely [2]. Among the recovery models, there are strategies such as rollback recovery, in which you can go back to a previous correct state that has been previously saved. This is carried out through the checkpoint that keeps the information of the state of a process periodically in a stable storage system, suspending the execution of this while saving it and consuming I/O resources, as well as network bandwidth.

Therefore, using the cloud to have an experimental system and make decisions about which is the best strategy to protect a parallel application that uses MPI can be a good alternative that offers a flexible environment, allowing us to replicate the behavior of the checkpoint characterizing it. This will permit us to analyze its impact prior to its use and explore with greater freedom of privileges and greater options because in the cluster, if you do not have root permission for the use and analysis of checkpoint libraries and tools analysis is more limited, there may be limits on storage or for the same size of the test cluster.

In this way, as a first step we intend to correlate the data obtained from the execution of the checkpoint in a real cluster and a virtual machine in the cloud. By doing this, we can compare its behavior and establish which aspects are similar or different, establishing the parameters that should be used in this virtual environment of experimentation. Likewise, after this correlation, we analyze these patterns and establish a deeper base that provides the necessary information from various aspects of the checkpoint's characteristics. In this respect, we can ask ourselves the following questions: Does the virtual machine add extra information in the checkpoints? What and how much information? How is the mapping done? How does the checkpoint scale? What are the influential factors? Can they be analyzed using virtual machines? Can you use the public cloud with virtual machines to extract information applicable in a cluster HPC bare metal? Which cloud configuration is the most appropriate according to the type of experiments that we need to perform in order to resemble those carried out in the cluster?

In this paper a proposal for the systematic study of the checkpoint is presented. We carry out an evaluation of checkpoint strategies for applications, comparing checkpointing at two different scenarios: virtualized on Amazon Cloud and physical, running on physical machines (Cluster).

The structure of this paper is as follows: Section 2 presents a Literature review and Section 3 looks at the background with a general view of the checkpoint as a fault tolerance strategy. Section 4 shows analysis methodology. Section 5 describes the experimental environment, the characterization and analysis of checkpoint is made in Section 6 and then in Section 7, we continue with conclusions, future work and references.

2. Literature Review

Cloud computing [3] is a method which can be used for representing computing models where IT services are delivered via internet technologies. These have attracted millions of users. In order to make the most of these advantages offered by the cloud, it can be used as a remote laboratory experimentation environment [4] whose resources are in a location remote to the user interacting with them and whose nature can be real or simulated. Cloud computing has many advantages, including service scalability and flexibility. However, the most common concerns from the user's perspective are performance, reliability, control, security, and privacy. Cloud users expect services to be available at all times and to have access to their data from any location [5]. Furthermore cloud computing can acquire and release resources on-demand and they can configure and customize their own Virtual Cluster. For these reasons, parallel scientific applications can benefit from these platforms [6].

As it is a flexible environment in which various applications can be executed, this platform can also be used to investigate the behavior of some fault tolerance strategies, such as check pointing/restart. This technique is very efficient for long-running applications. In this task-level fault tolerance technique, whenever some task failure occurs, it is provisioned to restart from the state which was most recently checked [7].

In this way, cloud computing makes use of the virtual machines [8] being implemented using specialized hardware, software or both. It provides services in the same way that the real physical machine does. The Cloud service provider can have direct access to the virtual machine. From this, you can perform experiments with root privileges and you can have full access checkpoint-restart approaches achieving fault tolerance by periodically saving the global state of the application persistently to stable storage and restarting from an intermediate state in case of failures [9].

In [10] the authors evaluated the performance of two of the most commonly used checkpoint/restart techniques (Distributed Multithreaded Checkpointing (DMTCP) and Berkeley Lab Checkpoint/Restart library (BLCR) integrated into the OpenMPI framework). In this paper the authors aimed to test their validity and evaluate their performance in both local and Amazon Elastic Compute Cloud (EC2) environments. The findings proved that DMTCP performs better than BLCR for checkpoint and restart speed, data scalability and compute processes scalability experiments. This paper is related to this research because we also use the DMTCP as a library for coordinated checkpoints and study its behavior in the cloud as an experimental environment.

3. Background

Fault Tolerance (FT) has now become a fundamental element to ensure the availability and operation of systems in High Performance Computing (HPC) environments, avoiding serious malfunctions through prevention and recovery protocols. The checkpoint is a FT technique in computer systems that is responsible for storing the global status of each process. According to [11], global checkpoint consists of taking a snapshot of the entire system state regularly (not necessarily periodically), so that when a failure occurs in any process, all the system rolls back to the latest checkpoint image to continue the computation.

There are several types of checkpoints, such as coordinated, uncoordinated and semi-coordinated, each of which depends on how the processes are coordinated amongst themselves to store the checkpoint [12]. With coordinated checkpoints (Fig. 1), all processes are coordinated to perform the checkpoint at the same time, generating a global state. In the case of uncoordinated checkpoints, each process performs the checkpoint independently, without the need for any coordination. In the case of semi-coordinated checkpoints, the processes are coordinated by groups.

In this way, each process generates a checkpoint file, which must be stored in a stable storage system.

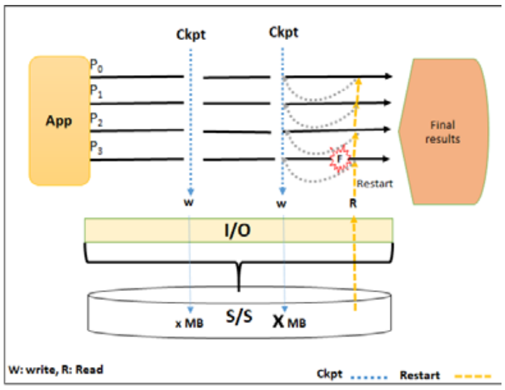

Fig. 1 Coordinated Checkpoint

The size of each checkpoint file depends on several aspects, such as the size of the application data and mapping, which must be taken into account when managing the fault tolerance in applications. This is because it generates an overhead in terms of storage time, in addition to the space that it occupies and therefore it must be managed efficiently. In this paper a coordinated checkpoint will be used, as explained in [13], as it is an effective technique for crash recovery support in software distributed shared memory (SDSM). It creates a checkpoint, during the synchronization process where a globally consistent state of execution is established to save all computation that SDSM has performed, until just prior to checkpoint creation. The Distributed MultiThreaded Checkpointing (DMTCP) library will be used for the generation of the checkpoint, which as indicated in [14] is a transparent user-level checkpointing package for distributed applications. DMTCP includes a checkpointing library that is injected into each process of the target application. This library creates a checkpoint thread in each process, to communicate with the coordinator and to copy process memory and other states to a checkpoint image [15].

Checkpointing [11] has a significant overhead increasing the application execution time, so it is important to deepen its operation to find a way to reduce this overhead and this implies observing its behavior in the generation of patterns of space, time, structure, among other relevant characteristics. This is because coordinated checkpointing represents a very effective solution to assure the continuity of distributed and parallel applications in the occurrence of failures [16].

4. Analysis Methodology

In order to establish similarities and differences in the behavior of the checkpoint coordinated, between an experimentation environment in a cluster and in a virtual experimentation environment such as the cloud, its characteristics were observed regarding files sizes, zones and I/O behavior. For this, following methodology was followed:

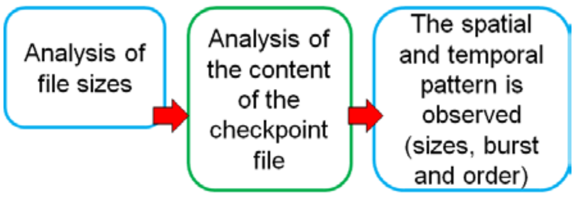
Analysis of file sizes
Analysis of the content of the checkpoint file
The spatial and temporal pattern is observed (sizes, burst and order)

Fig. 2 Methodology for Checkpoint Analysis in the Cloud

Figure 2 shows the steps that were followed to perform the analysis of the checkpoint in the cloud based on its comparison with the information on the execution of the checkpoint in the cluster. In this way, as an initial step, an analysis of the size of the files generated in the cloud is performed. If they are of similar sizes with the size obtained in the cluster, it continues analyzing the contents of the checkpoint file, where the information that is stored in the image of the checkpoint is observed. This is composed mainly of application data, libraries and information related to shared memory or communications within the node. Next, an analysis of the spatial and temporal pattern of the files is carried out, identifying the number of zones, the writing bursts, the order and the time involved in writing the checkpoint snapshot.

Elements necessary to configure the checkpoint in the cloud

Below are some necessary elements to take into account to execute the checkpoint in the cloud:

1. Select instance type.
2. Select the number of instances according to the number of processes to use.
3. Choose the storage device for the checkpoint.
4. Calculate the checkpoint interval.
5. Run the app with fault tolerance.

For point 1 and 2, the impact behavior of the checkpoint must be taken into account, it is important to analyze how the size increases depending on the mapping.

With respect to point 3, it should be remembered that the storage of the checkpoint is temporary, because when the application finishes running, the checkpoint files no longer have any use. But in addition to the protection offered by the checkpoint, the image of the same EBS volume could be captured, as indicated by [10], which says: For additional reliability boost, EBS volume can be backed up to the highly reliable object-based Amazon Simple Storage Service (S3) by creating a snapshot of that volume.

For point 4 it is necessary to characterize the checkpoint time, the time depends on the size, and the size depends on the selected system and the mapping.

5. Experimentation environment:

For the execution of the experiments in the present investigation the following environment was used:

In the cluster:

In the cluster, experiments have been carried out on different types of machines, with different architectures and different file systems:

A. AMD Opteron™ 6200 @ CPU 1,56 GHz, Processors: 4, cpu cores: 16, Memory: 256 GiB – File system: ext3. (HDD)
B. AMD Athlon(™) II X4 610e CPU 800.000MHz, Processors: 1, cpu cores: 4. Memory: 16 GiB - File system: NFS. (HDD)
C. AMD Athlon(™) II X4 610e CPU 800.000MHz, Processors: 1, cpu cores: 4. Memory: 16 GiB - File system: PVFS. (SSD)

In the cloud:

The following Amazon instances were selected for having a cheaper cost and for having different virtual CPU numbers:

1. T2.micro, virtual CPU: 1, Memory: 1 GiB. Storage Instance: 1x8GiB SSD (EBS). Performance Network: from low to moderate.
2. C3.xlarge, virtual CPU: 4, Memory: 7,5GiB. Storage Instance: 2x40 GiB SSD. Performance Network: moderate.
3. C3.2xlarge, virtual CPU: 8, Memory: 15GiC3.2xlargeB. Storage Instance: 2x80 GiB SSD. Performance Network: high.

In relation to the software used:

- Mpich-3.2.1 was used for the execution of the applications.
- DMTCP-2.4.5 for the checkpoint library.
- The application used: the MPI version of the NASA Advanced Supercomputing (NAS) product, which is called NAS Parallel Benchmarks (NPB). We used Block Tri-diagonal solver (BT) benchmarks of the NPB, Class B and C [17].
- The strace tool and the readdmtcp.sh script were used for the instrumentation.
- OS:
    - Cloud: Ubuntu Server 16.04 LTS.
    - Cluster: Centos 5.8.

6. Checkpoint Characterization and Analysis

6.1 Number of files and checkpoint file sizes:

To observe the behavior (size) of the checkpoint in different environments, Table 1 shows an example for the BT.B.4 app with two different mappings. 4 nodes with 1 process in each node (4N x 1P) and 1 node with 4 processes (1N x 4P). When comparing the size of the files generated from the checkpoints, we find they are the same or very similar between the executions carried out in different machines and with different file systems, but there is a greater difference when running the app with different mapping. In this way, we can see that with the different execution modes, mapping is an element that can significantly influence the size of the files. As long as there are more processes within the same node, the checkpoint file is bigger.

It is also important to indicate that although it is the same application with the same class, this distribution influences the size of the checkpoint. As you can see, the size of the checkpoint is smaller in the cases where the processes are in different nodes rather than in a single node. This is because the memory shared within a node significantly affects the size of the checkpoint, causing it to be larger when there are more processes within the same node, which occurs because of the MPI implementation used which is MPICH.

Likewise, in the execution of checkpoints we can see that several files are generated, every checkpoint consists of a shared data segment, a (local) data segment and a stack segment. The checkpoint creation time corresponds to the time used by each process to create a checkpoint in coordination.

Table 1 Comparison of the size of checkpoint files with execution in different types of cluster

App: BT.B.4				
Cluster	**Processes**	**No. Files**	**Size per ckpt file (MiB)**	**Total Size Ckpt (MiB)**
A	4N x 1P	4	143	572
	1N x 4P	4	157	628
B	4N x 1P	4	141	564
	1N x 4P	4	155	620
C	4N x 1P	4	141	564
	1N x 4P	4	155	620

The checkpoints generate a file for each process executed, in addition to generating other smaller files related to the execution, communication between nodes and to the restart. Table 2 shows several examples for app BT, Class C, with different mappings in the cloud. It is observed that the size of the checkpoint files is maintained even if the instance changes.

Table 2 Comparison of the size of checkpoint files with execution in different types of instances

Cluster	**Processes**	**No. Files**	**Size per ckpt file (MiB)**	**Total Size stored (MiB)**
t2.micro	**App: BT.C.4**			
	4I x 1P	4	444	1909.80
c3.xlarge	**App: BT.C.4**			
	4I x 1P	4	444	1909.80
	1I x 4P	4	458	1880
	App: BT.C.16			
	4I x 4P	16	161	2709.80
c3.2xlarge	**App: BT.C.4**			
	4I x 1P	4	444	1909.80
	1I x 4P	4	458	1880
	App: BT.C.16			
	4I x 4P	16	161	2709.80
	2I x 8P	16	180	2956.60

Table 3 shows a comparison between the cluster and the cloud of the sizes of the files generated by the checkpoints. For this experiment, the instance c3.2xlarge was used in the cloud and machine "A" was used as the cluster. In this Table 3, all the generated files are shown.

Table 3 Comparison of Checkpoint file sizes generated in the Cluster and in the Cloud

App BT.B.4	**Name of the file**	**CLUSTER (A)**		**CLOUD (C3.2xlarge)**	
		No. files	**Size files (MiB)**	**No. files**	**Size files (MiB)**
1Nx4P	ckpt_bt	4	155	4	157
	pmi_proxy	1	19	1	26
	mpiexec	1	19	1	26
4Nx1P	ckpt_bt	4	141	4	145
	pmi_proxy	1	19	1	24
		3	2.7	3	2.9
	mpiexec	1	19	1	24
	ssh	3	18	3	23
	sshd	3	2.5	3	2.7

The BT application was used with 4 process and a different number of nodes. The symbology used in Table 3 is as follows: 1N x 4P = means in the cluster environment to 1 node with 4 processes and in the environment of the cloud 1 instance with 4 virtual CPU; 4N x 1P = in the cluster environment it refers to 4 nodes and 1 process per node and in the cloud environment it means 4 instances and 1 virtual CPU per instance. The files ckpt_bt_* are the main checkpoint files where the status of the process is saved at the moment the checkpoint was executed. It can be seen that when comparing the size of these files in the cluster and in the cloud, they are similar. For example, for the BT.B.4 (1N x 4P) the size of the checkpoint file in the cluster was 155 MiB and in the cloud of 157 MiB this size is independent from the infrastructure. In another case, with a different number of nodes, with the BT.B.4 (4N x 1P) in the cluster the file size is 141 MiB and in the cloud 145 MiB. This similarity is observed in the following examples shown in the table, but although they are similar, it can be observed that the checkpoint files in the cloud are larger in all the observed cases, even in the rest of the files related to the execution.

This increase in size in the checkpoint files could be due to the fact that it indicates [13] the state of the computation defined at each moment in time by two main components: (1) the state of each of the Virtual Machine (VM) instances; and (2) the state of the communication channels between them (open sockets, in transit network packets, virtual topology, etc.). Thus, the general case implies saving both the state of all VM instances and the state of all active communication channels among them. Therefore, the state of the Virtual Machine could be influencing in some way the increase in the size of these files.

Characteristics of the checkpoint file:

The generated checkpoint files are described below:

- ckpt_bt.* : This file is generated one per process and contains information on the content of the checkpoint snapshot, which is structured by three different types of information:
  Data of the app, Libraries, Shared memory/ communications. This information can be observed inside the checkpoint files in different memory zones.
- ckpt_hydra_pmi_proxy_*: the application uses hydra, which is a process management system to start parallel jobs, creating a proxy process in each node. The proxy, which is a process administrator, divides the MPI processes. In this way, the proxy sends I/O information from the application's processes to a main proxy or process administrator.
- ckpt_mpiexec.hydra_*: It is related to the management of processes.
- ckpt_dmtcp_ssh_* and ckpt_dmtcp_sshd_*: when working with several nodes in the coordinator, the ssh file is created and in the client, the sshd file is originated. In these files, information related to communication is stored.
- dmtcp_restart_*: script to restore the application from the last performed checkpoint.

Content of the checkpoint file:

In order to read the contents of the checkpoint file, a "readdmtcp.sh" script must be executed, which is part of the utilities that the DMTCP library has. The way to use it is after the checkpoint files have been generated. This script is executed for some checkpoint files and in this way you get all the information regarding the content that the checkpoint has saved. Thus, if we rely on the contiguity of the memory addresses used, we can say that the following zones have been detected:

a) Data Zone:

In Table 4, we can see the data zone of the checkpoint file. In the first three columns, we can see the zone of the checkpoint snapshot with the start and end memory address, as well as the size of each operation and the total size.

Table 4 Data zone for a checkpoint file of the BT.B.4 app

CLUSTER		CLOUD	
Listing ckpt image area	**Total Size (Bytes)**	**Listing ckpt image area**	**Total Size (Bytes)**
400000-418000	98304	400000-418000	98304
618000-619000	4096	617000-618000	4096
619000-7388000	114749440	618000-619000	4096
		619000-7388000	114749440
92e3000-9347000 [heap]	409600	8b8e000-8bd6000 [heap]	294912
Bytes	**115261440**	**Bytes**	**115150848**
MiB	**109.92**	**MiB**	**109.82**

The total size of the data zone for the checkpoint of the application BT.B.4 is 109 MiB, and this size is the same for the two experimental environments such as the cluster (A) and the cloud (c3.2xlarge).

b) Libraries Zone and Shared Memory Zone:

Table 5 shows part of the information shown in the file generated by "readdmtcp.sh" in relation to the content of the checkpoint for a BT.B.4, in the zone of libraries and shared memory.

Comparison of the Input and Output (I/O) behavior of the Checkpoint file:

As noted in the previous point, the image stored in the checkpoint has a structure, which consists of a fixed part (libraries and shared memory). These depend on the system and the implementation of MPI. Likewise, the checkpoint has a variable part dependent on the data of the application. In this respect, the information regarding how the checkpoint is related to the I/O system also has a behavior, which we need to study if we want to characterize a checkpoint in a complete way.

Table 5 Libraries and Shared Memory Zone for a checkpoint file of the BT.B.4 app

CLUSTER		CLOUD	
Listing ckpt image area	**Total Size (Bytes)**	**Listing ckpt image area**	**Total Size (Bytes)**
3079a00000-3079a20000	131072	7f27cc000000-7f27cc021000	135168
3079c1f000-3079c20000	4096	7f27cc021000-7f27d0000000	66973696
3079c20000-3079c21000	4096	7f27d044c000-7f27d07d2000	3694592
…		…	
7f4fe3209000-7f4fe320a000	4096	7f27d517b000-7f27d517c000	4096
7f4fe320a000-7f4fe320d000	12288	7f27d517c000-7f27d517e000	8192
7fff1086f000-7fff1125e000 [stack]	10416128	7fffef6e8000-7fffefec9000 [stack]	8261632
Bytes	**37494784**	**Bytes**	**36179968**
MiB	**34.50**	**MiB**	**34.75**

Therefore, in the cloud it is worthwhile observing the behavior referred to the operations of input and output that the checkpoint performs when it generates the snapshot, in order to compare it with its behavior in the cluster, as show in Table 6. For example, comparing of the bursts generated (one burst being a contiguous number of write or read operations) for a checkpoint of the app BT.B.4 (1N x 4P), in the cluster it has been observed that the coordinated checkpoint has an I/O behavior in which two bursts of writing are observed. For BT.B.4, a first one is observed in which 44 consecutive writings are made with a weight of 8192 bytes and a second burst of 163 writings of 155.12 MiB.

In the case of the cloud, two bursts were also identified with a first burst having a weight of 8192 bytes, as in the cluster, but it made more writes (51 writes). In the case of the second burst in the cloud (172 writes), it also made more writes than in the cluster (163 writes) and has a slightly larger size of 158.77 MiB compared to 155.12MiB in the cluster.

6.2. Times of the applications executed with a Checkpoint:

With regard to times, the execution times of the application and the times of the application with a checkpoint. The times were measured in two ways, with the clock of the app itself and the command "time" of linux. In Table 7, the same experiment was executed ten times in the cluster. The experiment was done with BT.C.4 with two different distributions of processes, 1 process in each node and 4 processes in a same node. The BT.C.16 app was also used, to which the processes were also distributed in two different ways. Four processes distributed one in each node and eight processes in two nodes each.

The different distributions of the processes to put fault tolerance in the app affect the time of the application. In Table 7, it is observed that the BT.C app was executed more quickly when 16 processes distributed in two nodes (8 processes each node) were used.

The time also depends on the architecture of each experimental environment. Below is a comparison of the times obtained in some of the instances of the cloud, to get an idea of the differences and similarities that may exist in the times between instances when introducing fault tolerance to the application.

Regarding the cloud, the instance T2.micro presented a great variability over time in the ten executions, as show Table 8. It was the most variable instance. While the others remained more constant, but according to their characteristics differences are observed among them, where the app with checkpoint is faster than in other cases. At this point, the number of processes used and the number of instances must also be taken into account, since this also has an influence when storing the checkpoint.

In Fig. 3 it can be seen that there are differences between sizes and times with respect to the distribution of the processes used in one and in four instances. The size of each checkpoint of app BT.C.4, when we distribute the processes in 4 instances (4I x 1P), is 444 MiB and in 1 instance (1I x 4P) the size is 458 MiB. The total stored 1909.80 MiB (4I x 1P) and 1880 MiB (1I x 4P), including the files that the DMTCP generates of comunication and management when doing each checkpoint:

- ckpt_bt.C.4_*,
- ckpt_hydra_pmi_proxy_*,
- ckpt_mpiexec.hydra_*,
- ckpt_dmtcp_ssh_*,
- ckpt_dmtcp_sshd_*

Table 6 Comparison of the size, number and order of events of writes generated in the cluster and in the cloud

BURST 1 (Bytes)		BURST 2 (Bytes)		BURST 2 (Bytes)		BURST 2 (Bytes)		BURST 2 (Bytes)	
CLUSTER	**CLOUD**	**CLUSTER**	**CLOUD**	**CLUSTER**	**CLOUD**	**CLUSTER**	**CLOUD**	**CLUSTER**	**CLOUD**
28	28	4096	4096	94208	4096	4096	4096		4096
13	13	98304	98304	4096	4096	90112	4096		4096
4	4	4096	4096	4096	4096	4096	4096		4096
4	4	4096	4096	4096	4096	4096	4096		8192
4	4	4096	4096	4096	4096	4096	102400		4096
4	4	114749440	4096	4096	28672	503808	4096		8269824
4	4	4096	4096	16384	4096	4096	4096		4096
4	4	430080	114749440	4096	4096	12288	4096	**Total Size (Byte)**	
4	4	4096	4096	8192	4096	4096	4096	**162652160**	**148799488**
4	4	4071424	274432	4096	4096	20480	4096	**Total No. Write**	
4	4	4096	4096	4096	4096	4096	512000	**163**	**172**
6	6	17567744	135168	4096	98304	53248	4096		
4	4	4096	4096	4096	4096	4096	8192		
5	40	135168	4096	4096	4096	4096	4096		
4	4	4096	32768	1613824	4096	4096	4096		
12	6	4096	4096	4096	4096	110592	4096		
4	4	266240	266240	16384	4096	4096	16384		
12	4	4096	4096	4096	16384	4096	4096		
24	4	49152	266240	4096	4096	4096	65536		
24	4	4096	4096	4096	937984	401408	4096		
24	24	4096	266240	20480	4096	4096	4096		
4	24	4096	4096	4096	32768	12288	4096		
4	8	4096	3694592	258048	4096	4096	4096		
4	8	4096	4096	4096	8192	4096	4096		
4	8	4096	49152	4096	4096	4096	106496		
8	8	4096	4096	4096	86016	8192	4096		
8	24	4096	4096	94208	4096	4096	4096		
8	4	1048576	4096	4096	1830912	4096	4096		
8	4	4096	4096	4096	4096	4096	4096		
8	4	4096	4096	4096	16384	4096	4096		
8	4	4096	17567744	536576	4096	4096	413696		
8	8	4096	4096	4096	8192	4096	4096		
4	4	4096	4096	4096	4096	4096	8192		
12	8	131072	4096	4096	20480	131072	4096		
4	8	4096	8388608	4096	4096	4096	4096		
57	8	983040	4096	4096	81920	266240	4096		
4	8	4096	4096	1351680	4096	4096	4096		
57	8	8192	4096	4096	4096	266240	4096		
12	8	4096	4096	8192	4096	4096	8192		
4	4	4096	4096	4096	4096	1413120	4096		
7	5	4096	212992	221184	4096	4096	4096		
4	4	28672	4096	4096	1060864	16384	4096		
3663	52	4096	4096	4096	4096	4096	4096		
4096	4	4096	4096	4096	4096	45056	4096		
	52	4096	4096	2498560	4096	4096	8192		
	12	4096	4096	4096	4096	4096	4096		
	4	4096	2322432	73728	4096	4096	4096		
	7	1691648	4096	4096	1118208	4096	4096		
	4	4096	57344	237568	4096	4096	4096		
	3615	40960	4096	4096	4096	4096	4096		
	4096	4096	16384	45056	4096	4096	143360		
Total Size (Byte)		8192	4096	4096	8192	10416128	4096		
8192	**8192**	4096	237568	4096	4096	4096	2015232		
Total No. Write		16384	4096	4096	221184		4096		
44	**51**	4096	12288	4096	4096		65536		

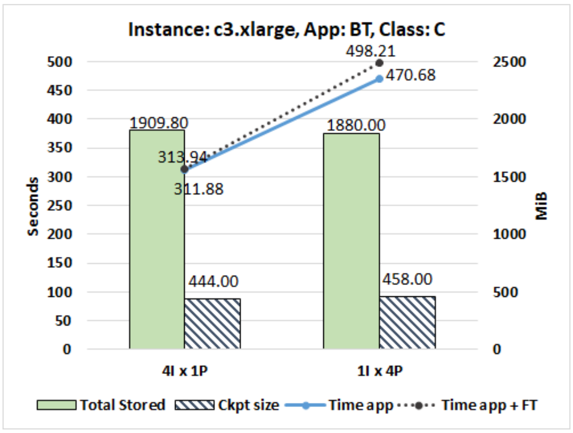

Fig. 3 Time in the c3.xlarge instance of the app and the app with checkpoint

However, it seems that the storage of these files, since they are smaller than the checkpoint, although there are more of them, does not affect the final time of the execution of the application with the checkpoint.

The same behavior was observed with the instance c3.2xlarge (Fig. 4), the size of the checkpoint remains the same as in the previous experiment 444 MiB and 458 MiB. With respect to the time, this almost did not present differences between the time of the app and the time of the application with tolerance to failures. In this case it is important to indicate that this instance has better features than the previous one. For that reason the time improved when concentrating the four processes in a single node.

Table 7 Executions made in different instances in the cluster and the measured times (seconds)

Cluster: A

Name of the app: BT CLASS: C

Np	Mt	1	2	3	4	5	6	7	8	9	10	A	Sd
4N x 1P	Ta_A	769.92	768.72	769.07	838.89	**951.83**	835.97	769.76	768.96	**766.89**	858.54	809.86	58.42
	Ta_L	781.09	**776.04**	776.38	846.29	**959.30**	843.34	777.09	776.30	774.20	865.98	817.60	58.22
	Ta+ FT_A	848.85	866.63	**878.63**	850.03	862.48	**844.95**	850.04	847.35	848.98	846.14	854.41	10.52
	Ta+ FT_L	858.56	875.63	**887.65**	859.02	871.51	**853.92**	859.02	871.66	857.95	855.13	865.00	10.47
1N x 4P	Ta_A	**487.23**	537.86	542.93	497.62	496.76	**551.23**	507.3	493.03	494.66	494.85	510.35	22.73
	Ta_L	**492.26**	542.09	547.24	501.86	501.00	**555.42**	511.65	497.25	498.89	499.08	514.67	22.65
	Ta+ FT_A	496.97	**485.54**	508.79	511.95	496.80	492.75	513.46	**553.01**	501.70	499.72	506.07	17.68
	Ta+ FT_L	502.36	**490.74**	514.16	517.36	502.02	498.03	518.84	**558.70**	507.11	505.05	511.44	17.81
4N x 4P	Ta_A	346.75	**453.54**	311.32	386.96	377.05	325.59	**284.26**	292.32	357.69	304.98	344.05	49.30
	Ta_L	353.91	**457.87**	317.09	390.30	389.05	330.31	**289.00**	301.99	362.49	309.29	350.13	49.10
	Ta+ FT_A	308.22	369.99	334.98	440.12	427.95	427.95	324.22	388.83	**442.25**	**270.18**	298.76	58.47
	Ta+ FT_L	325.68	378.47	342.64	**463.37**	436.36	436.36	341.30	394.83	448.27	**280.38**	307.38	58.07
2N x 8P	Ta_A	170.39	179.58	155.82	168.96	**148.11**	175.56	176.76	154.91	**186.40**	175.07	169.16	11.68
	Ta_L	172.99	182.06	157.73	170.96	**149.98**	177.59	178.66	156.86	**188.59**	177.08	171.25	11.80
	Ta+ FT_A	183.45	184.15	184.38	183.21	186.42	**178.87**	**192.75**	180.09	187.24	187.06	184.76	3.73
	Ta+ FT_L	186.96	187.59	188.03	186.54	189.98	**182.16**	**196.25**	183.49	190.81	190.48	188.23	3.79

Ta: Time app	Ta+FT_A: Time app + FT (App time)
Ta_L: Time app (Linux time)	Ta+FT_L: Time app + FT (Linux time)
Ta_A: Time app (App time)	Mt: Time measured in seconds
Ta+FT: Time app + FT	P: Number of processes
A: Average	Sd: Standard deviation
I: Instance	

Table 8 Executions made in different instances in the cloud and the measured times (seconds)

Name of the app: BT   CLASS:C

I	**P**	**Mt**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**A**	**Sd**
t2.micro	4I x 1P	Ta_A	325.5	325.68	**320.56**	329.61	329.90	1779.66	**3234.99**	361.44	360.54	358.11	772.6	926.5
		Ta_L	328.5	328.67	**323.57**	332.93	332.93	1785.54	**3267.30**	364.64	363.70	361.48	778.9	934.5
		Ta+ FT_A	361.8	1442.07	3386.99	2752.65	**3456.08**	**322.61**	335.89	331.35	331.07	330.42	1305	1292.9
		Ta+ FT_L	366.1	1449.75	3394.37	2760.21	**3463.58**	**327.20**	340.02	335.65	335.49	334.75	1310.7	1294.3
c3.xlarge	4I x 1P	Ta_A	309.8	311.25	310.36	310.86	310.60	311.02	310.21	310.41	310.21	310.58	310.5	0.4
		Ta_L	312.9	314.17	313.28	313.77	313.52	313.93	313.12	313.33	310.58	313.49	313.2	0.9
		Ta+ FT_A	311.3	311.4	311.89	311.19	311.91	314.41	312.67	310.72	312.51	310.89	311.9	1.03
		Ta+ FT_L	315.5	315.47	315.95	315.27	315.98	318.48	316.73	314.77	316.59	314.97	315.9	1.03
c3.xlarge	1I x 4P	Ta_A	467.3	468.29	469.27	469.26	468.07	467.85	468.97	468.89	468.54	468.96	468.5	0.62
		Ta_L	471.9	472.55	473.52	473.52	472.30	472.09	473.20	473.14	472.77	473.22	472.8	0.56
		Ta+ FT_A	474.7	475.54	472.78	476.77	472.66	477.29	471.85	477.79	471.60	477.09	474.8	2.29
		Ta+ FT_L	480.1	480.88	478.10	482.08	477.96	482.61	477.15	483.10	491.60	482.41	481.6	3.90
c3.xlarge	4I x 4P	Ta_A	123.3	123.04	122.73	123.49	122.59	122.89	123.80	122.89	123.29	123.44	123.1	0.36
		Ta_L	124.8	124.38	124.11	124.91	123.99	124.31	125.20	124.30	124.70	124.83	124.5	0.37
		Ta+ FT_A	125	124.53	128.97	130.62	127.77	128.46	132.58	130.35	132.92	131.60	129.2	2.75
		Ta+ FT_L	128	127.15	131.55	153.47	145.47	131.12	135.21	133.01	135.58	134.35	135.5	7.68
c3.2xlarge	1I x 4P	Ta_A	315.1	318.07	313.73	313.69	313.7	315.24	313.05	311.61	313.71	314.6	314.2	1.61
		Ta_L	317.9	320.82	316.45	316.40	316.43	317.98	315.76	314.31	316.42	317.34	316.9	1.63
		Ta+ FT_A	324.6	280.12	293.83	315.39	301.91	315.53	300.58	309.98	304.13	313.9	306	12.14
		Ta+ FT_L	328.4	283.78	297.36	319.09	305.51	319.22	304.18	313.66	307.74	317.60	309.6	12.18
c3.2xlarge	4I x 1P	Ta_A	306.4	306.48	306.94	306.95	306.62	306.42	306.86	306.64	306.38	307.2	306.6	0.25
		Ta_L	309.6	309.36	309.81	309.83	309.49	309.30	309.74	309.54	309.25	310.03	309.6	0.24
		Ta+ FT_A	307.4	307.6	306.91	306.62	306.62	310.69	308.98	307.67	307.76	307.1	307.7	1.18
		Ta+ FT_L	311.5	311.62	310.92	310.63	310.67	314.71	313.01	311.69	311.78	311.16	311.7	1.17
c3.2xlarge	4I x 4P	Ta_A	89.8	89.13	89.33	89.84	90.03	89.67	89.4	89.84	89.56	89.84	89.6	0.27
		Ta_L	90.8	90.16	90.35	90.87	91.04	90.74	90.48	90.92	90.66	90.86	90.7	0.27
		Ta+ FT_A	90.1	90.02	89.71	95.19	90.93	90.35	94.8	92.84	90.25	89.99	91.4	1.97
		Ta+ FT_L	92.3	92.20	92.02	97.40	93.13	93.13	96.98	95.05	92.45	92.23	93.6	1.94
c3.2xlarge	2I x 8P	Ta_A	140.6	140.67	141.69	140.23	141.08	140.66	139.67	141.70	141.70	141.00	140.9	0.64
		Ta_L	142.1	142.23	143.19	141.74	142.60	142.21	142.21	141.19	143.24	142.56	142.3	0.58
		Ta+ FT_A	153.3	138.58	136.53	140.85	137.60	141.99	138.71	141.56	137.61	141.66	140.8	4.55
		Ta+ FT_L	156	141.33	139.20	143.55	140.30	144.66	141.42	144.25	140.31	144.35	143.5	4.57

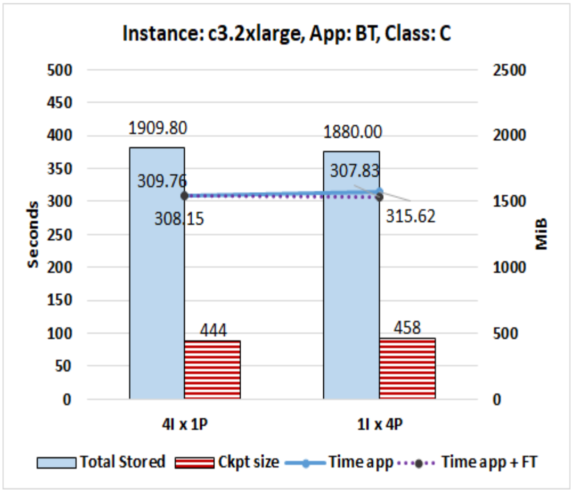

Fig. 4 Time in the c3.2xlarge instance of the app and the app with checkpoint

When executing the application and the checkpoint in the instance C3.2xlarge with 16 processes (4I x 4P) in four instances, a shorter time was observed. The second best time was also observed with the same number of instances (4) and processes (16) as the previous one and with the instance C3.xlarge, as show in Fig. 5.

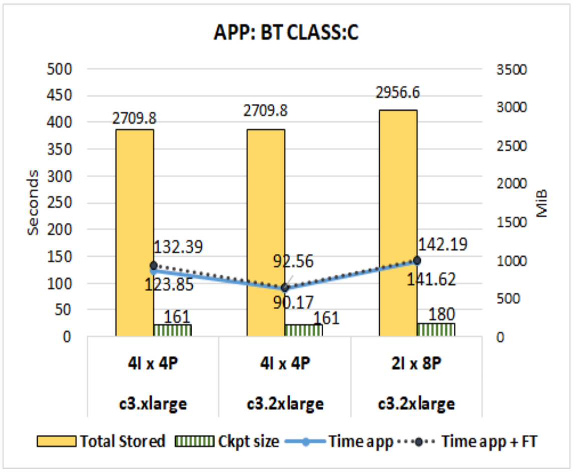

Fig. 5 Time of app BT.C.16 in the c3.xlarge and c3.2xlarge instances with checkpoint and without checkpoint

Figure 5 shows the graphic representation of the implementation of the app BT.C.16 in two different instances c3.xlarge (1 mapping: using four instances) and c3.2xlarge (2 mapping: using four instances and two instances). The observed behavior refers mainly to the variation in the times between the instances, with the execution in the instance c3.2xlarge being faster. As well as this, the proper selection of resources is important, it is observed that the best distribution of the processes was in 4 instances with four processes in each instance, in comparison with the execution in two instances with eight processes in each one. This makes us confirm that mapping is an element that has a significant influence when putting fault tolerance in an application.

Depending on the configuration of the checkpoint, the overhead can increase or decrease. Therefore, it is important to know in advance how your behavior will be in order to configure it in the most appropriate way.

Given the large service diversity, selecting an appropriate virtual cluster configuration for an application with FT is a non-trivial challenge. While functional properties can be compared by studying provider information, non-functional properties, such as performance, need to be quantified tediously [18].

As for the instances used in the cloud, these can have an influence according to their performance characteristics. For example, experiments using the instance c3.2xlarge showed a better behavior in terms of time used than with the instance c3.xlarge.

In this way, we can observe that the sizes of the checkpoints remained constant in all cases, while the times decreased when using the instance c3.2xlarge, which has more memory, CPU and storage resources than the other two instances used: t2.micro and the c3.xlarge.

Table 9 Difference of the percentage of time of an app and the time of an app with FT in several instances

App: BT.C	I	Time app (s)	Difference App + FT Time (%)	I	Time app (s)	Difference App + FT Time (%)
4I x 1P	c3.xlarge	311.88	0.66% ↑	c3.2xlarge	308.15	0.52% ↑
1I x 4P		470.68	5.85% ↑		315.62	2.47% ↓
4I x 4P		123.85	6.9% ↑		90.17	2.65% ↑
2I x 8P					141.62	0.4% ↑

Table 9 shows the percentage difference in time when fault tolerance is applied to an app in the instances of the cloud used. It was observed that in most cases the time i ncreased by a small percentage and in one of them it decreased.

7. Discussion of the results

Table 10 summarizes the results in terms of the patterns obtained when executing a coordinated checkpoint in the cluster and in the cloud.

From this information, we have an overview of the behavior of the coordinated checkpoint executed with the DMTCP library, the BT application has been used to show the results, which allows us to know that there is a great similarity in behavior in both environments (cluster and cloud). This will contribute to being able to use the cloud in order to know the most appropriate configuration of the checkpoint before taking it to an environment of a cluster in production.

Table 10 General Characteristics of the Observed Patterns

Observed elements	Cluster	Cloud
Checkpoint file sizes	You can correlate files that are generated and their sizes.	
Characteristics of the content of the checkpoint file	The content information of the checkpoint is composed of Data, Libraries and Shared Memory.	
Memory mapping, memory location	Distribute information in 3 memory zones	Distribute information in 2 memory zones
Comparison in the I/O behavior of the Checkpoint file	Two bursts of writes	Two bursts of writes
	The pattern of the writes and the order is different	
Times of applications executed with Checkpoint	Similar time	Variable time
Infrastructure	Mapping affects: Size and time. Instance affects: Time	

With regard to the selection of resources in the cloud, the number of instances, the distribution of the processes must be taken into account. In the examples shown in this paper, we consider that the best option is to choose 4 instances (few processes per node, and nodes with more features). It is also better to choose 4 instances with fewer benefits, than to choose 2 with more features but grouping more processes, because it increases the checkpoint size and storage time.

8. Conclusions and Future Works

The cloud is a flexible environment that has allowed us to execute the checkpoint in an environment of experimentation similar to the cluster, with greater freedom of choice. The behavior of the checkpoint has been similar in most of the elements that have been taken into account to evaluate what the size of the checkpoint files is, the content and its structure. In this way, the cloud has served to perform an analysis of the abstract behavior of the spatial and temporal application. That analysis in turn provides us with information to select the resources and it can be done with a limited set of resources. We were also able to observe that in the case of the study of benefits, as well as in the case of time, there is some variability depending on instance. Therefore, in our case, the cloud is better suited to the study of checkpoint behavior.

As future work, we intend to continue using the cloud to execute new fault tolerance strategies such as uncoordinated or semi-coordinated checkpoints that may need to use tools that need root privileges, since they use special system resources. In this way, it will allow us to know the impact prior to its use in a production cluster. As well as this, in the future we can attain the most global behavior of these strategies, being able to generate tools that mimic the I/O behavior of the checkpoint without having to execute it and being able to take them to the cluster and carry out a deeper study of the response of the system before the behavior of the checkpoint will enable us to reduce the overhead caused by the I/O of these fault tolerance strategies.

Acknowledgment

This publication is supported under contract TIN2017-84875-P, funded by the Agencia Estatal de Investigación (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) EU and partially funded by a research collaboration agreement with the Fundación Escuelas Universitarias Gimbernat (EUG). Appreciation to the CloudMas as Government Competency of AMAZON Web Services (AWS) by the access and use of its computation resources.

Competing interests

The authors have declared that no competing interests exist.

References

[1] P. Gomez, S. Mendez, J. Panadero, B. Aprigio, D. Rexachs and E. Luque, “Cloud, a flexible environment to test HPC I/O configurations,” Int'l Conf. Par. and Dist. Proc. Tech. and Appl. PDPTA'18, pp. 197-203, 2018.

[2] J. Weissman, “Fault Tolerant Wide-Area Parallel Computing,” International Parallel and Distributed Processing Symposium, pp. 1214-1225, 2000.

[3] D. Mittal and N. Agarwal, "A review paper on Fault Tolerance in Cloud Computing," 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2015, pp. 31-34.

[4] F. Santamaría, J. Ballesteros and J. González, “Plataforma cloud computing como infraestructura tecnológica para laboratorios virtuales, remotos y adaptativos -Cloud computing as technologic infrastructure for virtual, remote and adaptive labs”, Revista Científica, 3(23), pp. 98-110, 2016.

[5] A. Mohammad, Al-Rousan Mohammad, Y. Eman and E. Hanem, “A Study on Fault Tolerance Mechanisms in Cloud Computing,”

International Journal of Computer Electrical Engineering, pp. 62-71, 2017.

[6] P. Gómez and D. Rexachs, “Methodology to select a I/O configuration (hardware resources and stack software) in cloud platform,” BSC Doctoral Symposium, 2nd ed. Barcelona: Barcelona Supercomputing Center, pp. 143-144, 2015.

[7] D. Kochhar and H. Jabanjalin, “An approach for fault tolerance in cloud computing using machine learning technique,” International Journal of Pure and Applied Mathematics, volume 117, No. 22, pp. 345-351, 2017.

[8] K. Devi and D. Paulraj, "Multi level fault tolerance in cloud environment," International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, pp. 824-828, 2017.

[9] B. Nicolae and F. Cappello, "BlobCR: Efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots," SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seatle, WA, pp. 1-12, 2011.

[10] B. Azeem and M. Helal, "Performance evaluation of checkpoint/restart techniques: For MPI applications on Amazon cloud," 9th International Conference on Informatics and Systems, Cairo, pp. 49-57, 2014.

[11] A. Bouteiller, P. Lemarinier, G. Krawezik and F. Capello, "Coordinated checkpoint versus message log for fault tolerant MPI," Proceedings IEEE International Conference on Cluster Computing, Hong Kong, China, pp. 242-250, 2003.

[12] L. Fialho, D. Rexachs and E. Luque, "What is Missing in Current Checkpoint Interval Models?," 31st International Conference on Distributed Computing Systems, Minneapolis, MN, pp. 322-332, 2011.

[13] A. Kongmunvattana, S. Tanchatchawal and Nian-Feng Tzeng, "Coherence-based coordinated checkpointing for software distributed shared memory systems," Proceedings 20th IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, pp. 556-563, 2000.

[14] A. Jason, A. Kapil and G. Cooperman, “DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop,” 23rd IEEE International Parallel and Distributed Processing Symposium, 2007.

[15] J. Cao, K. A. Kapil, R. Garg, S. Matott, D. Panda, H. Subramoni, J. Vienne, G. Cooperman, "System-Level Scalable Checkpoint-Restart for Petascale Computing," IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, pp. 932-941, 2016.

[16] L.M. Silva and J.G. Silva, “An Experimental Evaluation of Coordinated Checkpointing in a Parallel Machine,” EDCC-3. EDCC Lecture Notes in Computer Science, vol 1667. Springer, Berlin, Heidelberg, 1999.

[17] D. Bailey, “The Nas Parallel Benchmarks,” International Journal of High Performance Computing Applications, pp. 63-73, 1991.

[18] J. Scheuner and P. Leitner, “Estimating Cloud Application Performance Based on Micro-Benchmark Profiling” IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 90-97, 2018.

Citation: B. León, P. Gómez-Sánchez, D. Franco, D. Rexachs and Emilio Luque. *IaaS Cloud as a Virtual Environment for Experimentation in Checkpoint Analysis.* Journal of Computer Science & Technology, vol. 19, no. 2, pp. 110-122, 2019.
DOI: 10.24215/16666038.19.e11
Received: April 1, 2019 **Accepted:** August 16, 2019.
Copyright: This article is distributed under the terms of the Creative Commons License CC-BY-NC.