Analysis of Checkpoint I/O Behavior

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. Nowadays, checkpoints have gained some relevance, given the increasing complexity of scientific applications for the use of many resources over a long period of time. Thus, in fault tolerance strategies, in addition to taking into account the impact that the application itself has on HPC systems, we must add the impact of the checkpoint. The checkpoint saves information about the application and the system in order to be able to restore the application, if necessary, in stable storage. The checkpoint can be considered as an intensive I/O application, so its storage need can have a great impact on the application. Therefore, in this paper, the analysis of the checkpoint's I/O behavior is presented. The number of checkpoints to be performed in an application is often related to the maximum overhead that you want to introduce in the application. If we know the maximum overload the user wants to pay for and the overhead that a checkpoint introduces, we can calculate the number of checkpoints to be performed. This overhead depends significantly on the I/O operations. The PIOM-PX tool was used to analyze the spatial and temporal I/O patterns of the checkpoint. Based on this analysis, a model was designed to predict their behavior. This information is used to calculate the number of checkpoints to be performed in an application given a maximum overhead predefined by the user. This will allow us to understand what happens when a checkpoint is created in an HPC system, in order to make decisions that adapt to the user's requirements.

Keywords: Checkpoint · Fault tolerance · I/O behavior · PIOM-PX

1 Introduction

Input/Output (I/O) is an important element that greatly affects the performance of parallel applications in High Performance Computing (HPC) systems. As it generates a lot of readings and writes, if they are very frequent they could impact significantly by collapsing storage and slowing down the execution of applications. Among the elements related to I/O behavior are the design of the same applications executed (I/O patterns), the HPC system and, especially, the storage subsystem (workload and resource management).

V. V. Krzhizhanovskaya et al. (Eds.): ICCS 2020, LNCS 12137, pp. 191–205, 2020.
https://doi.org/10.1007/978-3-030-50371-0_14

In [1], the input and output is differentiated in two ways: productive I/O and defensive I/O. Productive I/O is the writing of data that the user needs for actual science, such as visualization dumps and traces of key scientific variables over time. Defensive I/O is employed to manage a large application executed over a period of time much larger than the platform's Mean-Time-Between-Failure (MTBF). Defensive I/O is used for restarting a job in the event of application failure in order to retain the state of the computation, and hence the forward progress since the last checkpoint. Thus, one would like to balance the amount of resources devoted to defensive I/O and computation lost due to platform failure, which would require restarting the application. As the time spent on defensive I/O (mechanisms for fault tolerance) is reduced, the time spent on useful computations will increase. Checkpoints are a Fault Tolerance (FT) strategy which require intensive large-scale access to the storage system, through I/O operations.

Checkpoints can be differentiated into several types depending on how the processes involved in the execution of the application work with fault tolerance. In this way, if the processes are coordinated to create and store the checkpoint, they are said to be coordinated checkpoints. If each process performs the checkpoint independently, they are non-coordinated checkpoints and if they are coordinated by process groups the checkpoints are named semi-coordinated. In this work, in order to study I/O, coordinated checkpoints will be used because, when carrying out large amounts of simultaneous writes, they intensively access the storage system, so the file system must manage all this information, which can significantly affect the execution time of the application and influence its scalability. In this way, to analyze the influence of scalability, mapping on the size and time to make a checkpoint and having all the complete information of these accesses to the file system can help you manage them better. It is required to have the detailed information of the patterns generated to be able to replicate them without performing the computation, to achieve the most appropriate configuration and management of the file system and to be able to tune our applications with fault tolerance within the system, as well as to be able to carry out an analysis of the performance. This paper will focus on the analysis of the patterns generated by the checkpoint and its impact on the use of resources and generated overheads.

This article is structured as follows: In Sect. 2 related work related to this research will be presented. As a next step, in Sect. 3 we present the tools used to obtain this information and we will justify the selection of the tool used in this work. In Sect. 4 we will make a comparison between some I/O instrumentation tools, thus selecting the most suitable for our work. In Sect. 5 the Analysis of the checkpoint generated by the Distributed MultiThreaded CheckPointing (DMTCP) will be presented. In Sect. 6, a model to estimate the size and storage time of the checkpoint will be implemented. We finish with the conclusions and future work.

2 Related Work

Different authors have studied the I/O behavior and fault tolerance and its impact generating overhead in the applications. Below are some studies related to this research.

In [2] the authors indicate that among the elements that determine the frequency of the checkpoint I/O depend on the choice of the checkpoint interval, the period between checkpoints and the number of checkpoint I/O operations performed by the application. This work is similar to the proposed objective, the authors analyze the I/O operations but they do not consider the behavior (spatial and temporal patterns of the I/O operations). The authors propose an analytical model designed for the simulation, while we propose a behavior model that allows the replication of the behavior in different systems or system configurations. In [3], a checkpointing technique is presented that significantly reduces the checkpoint overhead and is highly scalable. For this asynchronous checkpointing technique, a theoretical model is developed to estimate the checkpoint overhead. In this paper, we propose a model that permits replicating the behavior of the checkpoint in different systems or system configurations, so that with reduced resources we can obtain information that allows us to predict the overhead. In [4], the goal of this research has been to develop a strategy for optimizing parallel I/O over the wide variety of possible access patterns. They have demonstrated that, with the two-phase access strategy for parallel I/O, it is possible to obtain significant improvements in performance over previously used methods. Data distribution and storage distribution have been decoupled, enabling the most effective configuration to be used for parallel. Overlapping and parallel I/O strategies are also compatible with our proposal.

3 Previous Analysis

The coordinated checkpoint requires the coordination of the processes involved in the execution of the application with fault tolerance. For each process a file is generated relative to the work done by that process. There are several factors that influence the size of the checkpoint files that have been generated and the time they have used, among which is the number of accesses to the files and the pattern of these accesses, which is immersed in the workload handled. These patterns can be analyzed or described by their spatial behavior (related to the type of access), the mode (sequential, striped, random), the size of the access, among others, and their temporal behavior (related to the number and frequency of access). Therefore, the checkpoint storage time depends on the size and pattern, as shown below:

$$T_{storage_{ckpt}} = f(size, pattern) \tag{1}$$

This checkpoint storage time (see expression 1) is part of the total time (see expression 2) required to perform the coordinated checkpoint, as follows:

$$T_{ckpt} = T_{coord_{ckpt}} + T_{storage_{ckpt}} \tag{2}$$

The coordination time ($T_{coord_{ckpt}}$) consists of the preparation and management time at which the checkpoint starts, after this the storage time begins, where it carries out the storage of the content that concentrates the state of the process at a specific time. The storage stage is where the checkpoint spends more time. Another element that influences the storage time of the checkpoint is the place where it will be stored in the I/O subsystem, such as device type (SSD, HDD) and file system (ext3, NFS, PVFS). But the patterns generated are dependent on the application and therefore independent of the system. Once we have the pattern, we can analyze its behavior in different systems or different configurations. It is important to study it, as it does not only influence the size of the checkpoint, it is also important in relation to the pattern consisting of the frequency, size and amount of writes made in the time of storage, management files, mapping dependence, the number of nodes and the files generated. All these factors become important when the number of processes increases. In our case, one file is generated per process.

The characterization model of the checkpoint I/O patterns shows that it makes different writes of different sizes, both in the coordination and in the content that the checkpoint stores, and these form the total checkpoint size. This stored information is composed of the following three zones: (1) A data zone which depends on the application and where, if we increase the number of processes, the zone decreases. This is because when increasing the degree of parallelism, the same amount of data is split into smaller pieces for each process. (2) A library zone which remains fixed, because it depends on what the application needs functionally from the system. (3) A shared memory zone in which, as we increase the number of processes within the same node, the size of this zone increases. This is because, depending on the number of processes used, a variable amount of memory is reserved for inter-process communications within the same node. This memory reserve depends on the MPI implementation used.

Therefore, the storage times of each zone depend on the size of each write made. Adapting this ratio to the Pearson correlation coefficient (see Eq. 3), whilst the size of the checkpoint zones increases, so does storage time.

$$r_{xy} = \frac{\sum Z_x Z_y}{N} \tag{3}$$

In our case, "x" corresponds to the size of each write, "y" corresponds to the operation write time and "N" is the number of writes made. In this way, the larger the size, the more time it takes and this also depends on the congestion for the number of files. Likewise, for analysis, it is important to have the right tool that can implement all the necessary information to perform the required analysis of these patterns generated by fault tolerance. To get the information about the behavior, we must monitor the application's I/O. There are different trace and analysis tools that help describe the application's I/O behavior. In Sect. 4, a comparative study of three important tools capable of tracing the I/O of the applications is carried out the coordinated checkpoint will also be taken as the case of central use as an application with intensive access to the storage system.

4 Tools Used

According to the parameters obtained above and to have information about their I/O behavior, different tools have been used. Below are the details of the information provided by each and a comparison, taking into account the type of information we need.

4.1 Experimentation Environment

The experiments have been carried out on: AMD Opteron™ 6200 @ CPU 1,56 GHz, Processors: 4, CPU cores: 16, Memory: 256 GiB, File system: ext3 and Disk type: HDD. The MPI implementation used was MPICH 3.2.1. For checkpoints the DMTCP-2.4.5 (Distributed MultiThreaded Checkpointing) has been used for this study [5].

4.2 Application I/O Behavior Tools

To perform the analysis of the checkpoint I/O patterns, it is necessary to have a tool that provides all the information required to study the checkpoint and its structure, how it is formed and what the information it stores is. In this way, there are a large number of tools that can trace the I/O of an application. In this work, we have used and compared three tools widely referenced in the literature used in HPC: Darshan, PIOM and Strace.

Darshan captures information about each file opened by the application. Rather than trace all operation parameters, however, Darshan captures key characteristics that can be processed and stored in a compact format. Darshan instruments POSIX, MPI-IO, Parallel netCDF, and HDF5 functions in order to collect a variety of information [6].

PIOM allows us to define an I/O behavior model based on the I/O phases of HPC applications at MPI-IO and POSIX-IO level. For every file used by the application, an I/O file is created. The spatial and temporal patterns are extracted from information contained in the I/O file [7].

Strace is a diagnostic, debugging and instructional user space utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. Strace works by using the ptrace system call which causes the kernel to halt the program being traced each time it enters or exits the kernel via a system call [8].

In the following tables we will make a comparison between Darshan, PIOM and Strace; and we show some general characteristics that are important for the user. In Table 1 we can observe the characteristics related to its installation and operation. The selected criteria were the following: Table 1 shows the elements that the tool traces. -File Access Type: This corresponds to the type of operation that is identified, for example: open, close, write, read, among others.

Table 1. Elements traced by the tool

Input				
Tool	File access type	Bursts	Trace the checkpoint	Trace the restart
Darshan	Yes	Together	Yes	No
PIOM	Yes	Independent	Yes	No
Strace	Yes	Independent	Yes	Yes

-I/O Bursts: A burst is the grouping of I/O operations of the same type. -Trace the checkpoint: This indicates the type of fault tolerance strategy being implemented. -Trace the restart: This indicates whether there is a trace if the starting point is a restart.

The three tools can identify the types of file access, but Darshan summarizes the total operations of the same type (burst) while PIOM and Strace show the operations separately, thus being able to identify the size and time of each one, as well as other indicators. With respect to the checkpoint, the three tools can trace their information, but with respect to the restart, only Strace can do it. However, this aspect does not present a major problem because the restart behaves in the same way as the checkpoint but instead of being "writings" they are "readings". Table 2 shows some elements of the monitoring considered. -Overhead: Corresponding to the overhead introduced by the tool. -Administrative Privileges: Corresponding to whether it is necessary for the tool to be executed with administrator privileges. -Transparent: Not interfering in the application execution. The overhead introduced by Darshan and PIOM is smaller than the one introduced by Strace, because it implements a lot of information related to the monitoring and manipulation of interactions between processes and the Linux kernel, which includes system calls, deliveries of signals and changes in the state of the process. Regarding the administration privileges for its use, once they are already installed, the three tools can be used without having administrator privileges, as none of the three hinder the execution of the application.

Table 3 shows the execution times of the NAS parallel benchmarks with one checkpoint, the execution times with fault tolerance, as well as the instrumentation tool used in each case. The purpose of this is to be able to check the time overhead generated by each of them.

Table 2. Monitoring tool

Monitoring			
Tool	Overhead	Administrative privileges	Transparent
Darshan	Small	No	Yes
PIOM	Small	No	Yes
Strace	Medium	No	Yes

Table 3. Overhead introduced by each of the tools

App	Time (sec.)			
	APP	DARSHAN	STRACE	PIOM
BT.B.4 N:1	139.93	140.12	142.15	141.52
BT.C.4 N:4	431.12	432.16	433.81	432.71
BT.D.25 N:1	2568.30	2834.91	3508.72	2641.45
SP.D.25 N:1	3810.71	4032.89	5221.62	4011.14
LU.D.25 N:1	3947.86	4186.34	4378.81	4097.10

In Table 3 it can be seen that the executions of the application BT, SP and LU applications class B [9] with fault tolerance implemented with Strace is the longest execution. As for the other two tools, Darshan and PIOM, it remained similar, that is, in the cases studied, half the time was less with PIOM and the other half with Darshan. This indicates that these two tools introduce less overhead when they are implementing the application with checkpoint.

I/O Information: Other important information to be analyzed is that generated by the I/O when the checkpoint is being stored. In the case of the PIOM tool, spatial information (number and size of writes), type of access (sequential, random) and temporal information (frequency of accesses) are analyzed in order to identify the bursts or consecutive accesses to the disk. Darshan gives profile information with a number of writes grouped by size and does not allow us to identify bursts.

Table 4 shows the difference in the I/O behavior identified in the BT, SP and LU class B applications with 4 processes with DMTCP, identified by the PIOM and Darshan tools. It also shows the writes identified, which present a difference between both tools of less than 4% and for the size of less than 3%. In addition, it shows the information per file (each process generates a file), that is, for BT, SP and LU class B applications with 4 processes executed on a single node (N: 1), the information for each of the four files (Fx). As all four files have similar behavior, to present the information in the table, only two files are shown for each application.

Table 4. I/O information identified by PIOM and Darshan by file

App	Files	PIOM	Darshan	Difference %Num. writes	PIOM	Darshan	Difference %MiB
		No. writes			MiB		
BT.B.4	F0	209	216	3.24	155.14	156.59	0.93
N:1	F1	209	216	3.24	154.56	156.00	0.92
SP.B.4	F0	215	228	4.44	139.55	141.02	1.05
N:1	F1	221	226	1.34	138.99	140.43	1.02
LU.B.4	F0	219	225	2.67	93.16	95.57	2.52
N:1	F1	217	224	3.13	93.16	94.68	1.60

In Table 5, a comparison is made between PIOM and Strace with respect to the "writes" generated and the bursts, considering a burst as a number of continuous operations of the same type, in this case of "writes", as well as the size of each of them. Table 5 shows that, in terms of the first burst, both tools detect the same number of "writes" 44 writes and a size of 0.0078 MiB for the checkpoint. On the other hand, for burst two there is a difference between 0 to 5% between the number of writes that detect these tools and in terms of size, it is very similar, so this aspect is less than 1% difference.

Table 5. I/O information identified by PIOM and Strace by file

App	File	PIOM			Strace			Difference
		Burst 1	Burst 2	Total	Burst 1	Burst 2	Total	%Num.
		No. writes			No. writes			No.writes
BT.B.4	F0	44	165	209	44	155	199	5.03
N:1	F1	44	165	209	44	157	201	3.98
	File	MiB			MiB			%MiB
	F0	0.0078	155.13	155.14	0.0078	154.52	154.48	0.43
	F1	0.0078	154.55	154.56	0.0078	154.56	154.49	0.05
SP.B.4	File	No. writes			No. writes			%No. writes
N:1	F0	44	171	215	44	173	217	0.92
	F1	44	177	221	44	173	217	1.84
	File	MiB			MiB			%MiB
	F0	0.0078	139.54	139.55	0.0078	138.97	138.98	0.41
	F1	0.0078	138.98	138.99	0.0078	138.97	138.98	0.01
LU.B.4	File	No. writes			No. writes			%No. writes
N:1	F0	44	175	219	44	167	211	3.79
	F1	44	173	217	44	166	210	3.33
	File	MiB			MiB			%MiB
	F0	0.0078	93.22	93.23	0.0078	93.15	93.16	0.07
	F1	0.0078	93.22	93.23	0.0078	93.14	93.15	0.08

Table 6 shows some aspects related to the output report: Postprocessing (It implies what must be done after the tool is executed in order to analyze the information), Temporary Pattern (Informing on the order that the access events are carried out to the file), Space Pattern (Providing information about what file is used by the application, which process accessed a file and where (File positions) it was accessed by the process during the application execution), Generated File (Referring to the amount of trace files generated) and Size Generated File (Referring to the size of trace files generated).

With respect to the reports generated by these three tools, Darshan does not need a post-processing, because with the same utility programs that it features, several ready reports can be generated that show the information in a summary way. With respect to PIOM, post-processing is medium because it presents an orderly report with each of the operations identified, as well as generating a report for each process executed with all its information. However,

certain calculations must be made to obtain the total operations, total size and time. As for Strace, post-processing must be carried out by the user, because it is only a monitoring tool and it is much more complex because it generates a lot of information and as it issues a single report, the required information must be thoroughly searched. The three tools show the temporal and spatial patterns, PIOM and Strace show them by operation, although Darshan shows them summarized. In relation to the size of the reports generated, the smallest is Darshan, on average. In the case of PIOM and Strace, as they generate so much information, the size of the report is larger.

We consider that the three tools used adequately instrument the I/O of the applications with checkpoint. But based on the results of these experiments and the qualitative comparison, we consider that PIOM is the tool of instrumentation that most adapts to the study we want to perform in relation to the analysis of the generated patterns of the I/O of the applications with fault tolerance.

5 Analysis of the Checkpoint Generated by the DMTCP

5.1 Spatial and Temporal Pattens

Figure 1 shows the information identified by PIOM of a checkpoint of a BT.D.25 application. We can observe the moment in which the checkpoint began to be stored and each of the zones that compose it, that is, the zone of data (DTAPP), the library zone (LB) and the shared memory zone (SHMEM), as well as the size of each zone. Knowing this, we can know the time it took to store each zone and the total time to store all the zones.

Likewise, PIOM also provides us with information about the total execution time of the application plus the checkpoint and the total size that will be stored. All this information has been validated with the information generated by the DMTCP, which is the following: Checkpoint time: 138.57 s. Checkpoint size: 1205.51 MiB (This size is per file, as is a BT.D.25 must be multiplied by 25 processes, to obtain the total stored). App time + checkpoint: 2641.45 s.

Table 6. Output tool

Output					
Tool	Post-processing	Show temporary pattern	Show space pattern	Generated file	Size generated file
Darshan	Easy	Summarized	Summarized	1	Small
PIOM	Medium	By operation	All	1 per process	Medium
Strace	Complex	By operation	All	1	Large

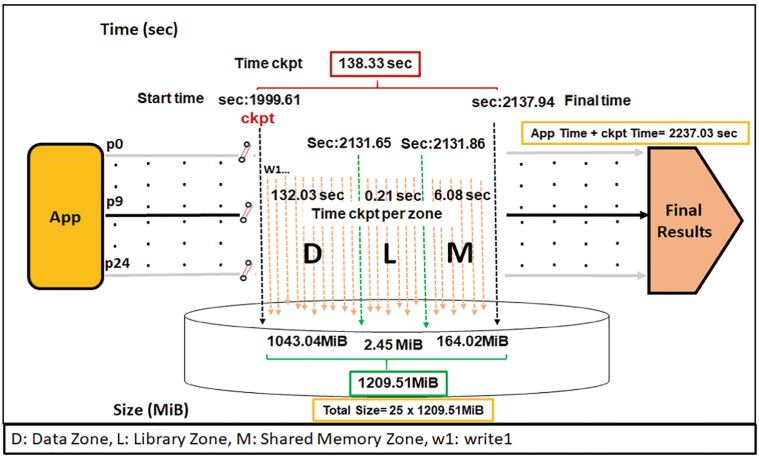

Fig. 1. Spatial and temporal patterns identified by PIOM

In the case of this example, it indicates that the time to store the checkpoint is 138.57 s, which is similar to that shown by PIOM when adding the three zones 138.33 s. In addition to the total size stored by each checkpoint file 1205.51 MiB and is also similar to the information identified by PIOM 1209.51 MiB. In this sense, the total time of the application with fault tolerance in this case is 2641.45 s and the one identified by PIOM for this example was 2237.03 s. With this information generated by PIOM, we obtain important details of the patterns, being able to identify and compare them with great congruence with what the DMTCP generates.

Table 7 shows a comparison between the times and sizes of each zone in the execution of the BT benchmark with class D, with 64 and 16 processes and with I/O. In this table we can see some elements which impact the execution time of the application with fault tolerance.

Table 7. Elements that affect the temporal and spatial pattern

App		Ckpt zones		
		Data zone	Library zone	Shared Memory zone
BT.D.64.mpi_io_full	Time	327.96	2.67	537.28
	Size per file	478.16	2.53	533.86
	Total size	30602.25	161.75	34167.04
BT.D.16.mpi_io_full	Time	135.28	0.12	3.01
	Size per file	1662.89	2.52	108.76
	Total size	26606.24	40.32	1740.16

If we analyze the results presented by each zone, we can observe the following: the data zone with 64 processes occupied a larger amount of time than in the other case with 16 processes, although the workload was equal in all two cases. When distributing it among the 64 processes, the size to handle for each process was smaller. Therefore, this data zone is impacted by the amount of information that must be stored from the application itself and the number of processes that it handles, producing congestion in the storage system.

In relation to the library zone, it can be observed that when it is handling 16 processes, the size per file is the same, regardless of the workload, but when increasing the number of processes to 64, the time also increases, as does the total to be stored in to this zone.

With respect to the shared memory zone, it can be seen that this is directly impacted by the number of processes used within the same node. In the case of 64 processes, it can be observed that the size is much larger than that of 16 processes, although the workload handled is the same. Therefore this zone is independent of the workload handled, so the time increases as the number of processes increases, because the size also grows.

In the case shown in Table 7, all the files generated by the checkpoint have been stored locally and an ext3 file has been used (Experiments have been carried out on several file systems, but we have illustrated this with ext3). In this way, the number of processes used is an element that significantly impacts the runtime of the application with fault tolerance, because a bottleneck is formed when all processes try to access the storage system at the same time because it is a coordinated checkpoint.

5.2 A Comparative analysis of the I/O Patterns of Applications with Fault Tolerance

In order to compare the I/O of an application with the I/O of the checkpoint of the same application, the first graph in Fig. 2 shows the behavior of the I/O of BT.C.16.mpi_io_full. In this case, it made a total of 440 writes in forty bursts of 10 writes, each write of 16 MiB, and the last of each burst 2.18 MiB. Therefore, the number of writes and their sizes exhibit regular behavior. In the second graph in Figure 2, the I/O behavior of a checkpoint executed in BT.C.16.mpi_io_full can be observed. In this case, it made a total of 241 writes, and we can observe that the writes have different sizes; there are a great number of very small 4 KiB ones and few large ones of up to 111.73 MiB. In the same way, the time varies if we compare when making the small writes, which can take thousandths of seconds, whereas a big write can take more than 12 s. Therefore, the I/O behavior of the checkpoint is not regular. In both cases, we can observe that the time depends on the writing size.

After observing the trace generated by PIOM, we need to know the correlation between the size of each writing with the time it took to make. In this way, we can find this if we use the Pearson correlation coefficient, which is defined as the covariance between two typified variables. By applying Eq. 3 to our traces to understand the correlation between the size and time of the checkpoint scripts,

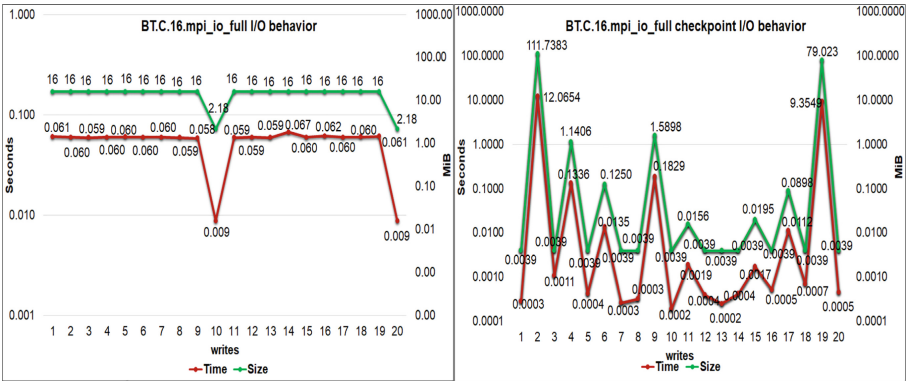

Fig. 2. I/O behavior (writes size and time)

we can see an association level of 0.992. Therefore, a positive correlation between both variables is observed whereas the write size increases the time. Therefore, the size of each write is an important element to study because it significantly affects the time of the checkpoint.

5.3 Checkpoint Storage Time Estimation

In [10] a study was conducted to predict the size of the checkpoint. In this article, equations were obtained to calculate the size of the areas that make up the checkpoint. In the case of the DTAPP Zone, it was calculated through an equation that was made from the number of processes and the size of the zone. In this way, the size of this zone could be estimated with a different number of processes. With respect to the LB zone in all cases, the size of this zone of 2.45 MiB was obtained. In this case, it was recommended to identify the area in a run and verify the size, because if it is the same application it does not change even if the number of processes and the workload change. With regards to the SHMEM zone, as the size of this zone depends on the number of processes, the equation obtained was as follows:

$$SHMEM_{Size} = 0.0617x^2 + 3.9983x + 25.47 \tag{4}$$

The storage time of the checkpoint depends on the size and as noted above, we can already predict the size of each of the checkpoint zones. Therefore, we can estimate the storage time approximately. In order to try to get closer to the value of time, we have calculated the regression equation for each of the zones, obtaining the following equations:

$$DTAPP_{Time} = 1.46041308E - 07x - 0.00404367 \tag{5}$$

$$LB_{Time} = 1.3205654E - 07x - 0.00014156 \tag{6}$$

$$SHMEM_{Time} = 1.5359626E - 07x - 0.0109242 \tag{7}$$

In these equations "x" corresponds to the size of each of the zones. So as to validate the three equations shown above 6, 7, 5, the execution of the following applications was executed and measured with a checkpoint: BT.C.16, BT.C.25, BT.C.36, BT.C.49, BT.C.64 , BT.D.25, BT.D.36, BT.D.49, BT.D.64, SP.C.64, LU.C.36 on one node and BT.D.64 on two nodes. These applications were used

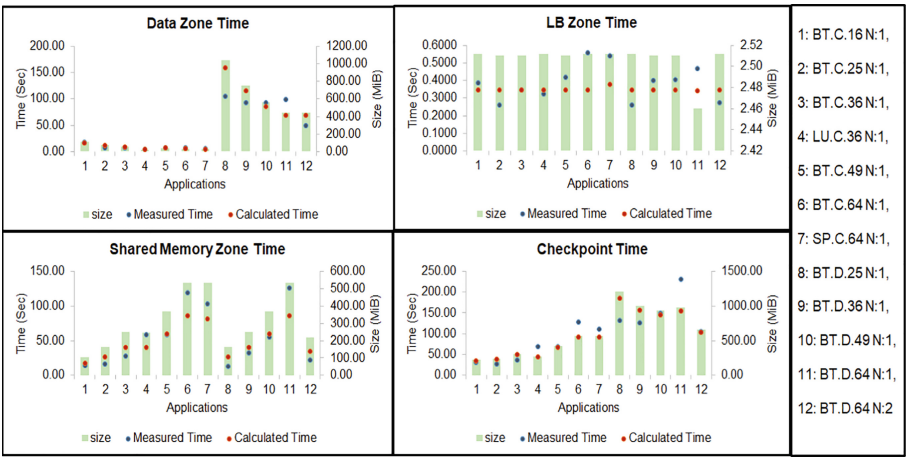

Fig. 3. Comparison between measured and calculated time

with different workloads, number of processes and nodes, in order to be able to vary the sizes of the checkpoint zones and check the validity of the equations. In Fig. 3 you can see the time in which the data zone was stored, the library zone, the shared memory zone and the total time of the checkpoint. In general, it is observed that the time calculated with the equations increases as the size grows; with respect to the measured time this also remains similar, but in some cases there is variability that occurs because it is being stored on the hard disk and this can impact on the variability of the storage time.

In this way the calculated results obtained from the equations presented an approximation with the actual measured values. The variability in the times can be considered acceptable to estimate the storage time of the checkpoint.

5.4 Approximate Model the Number of Checkpoints to Execute

This section presents an approximate model for obtaining the number of checkpoints in a given execution (Fig. 4). For this, the checkpoint size must first be calculated, identifying and calculating the zones that integrate it. The DTAPP zone, depending on the application, should look for the regression equation, the LB zone is constant and once verified it is sufficient and the SHMEM zone is calculated using the equation indicated in the model. As a next step, the storage time of each zone must be calculated, since the time depends on the size of the

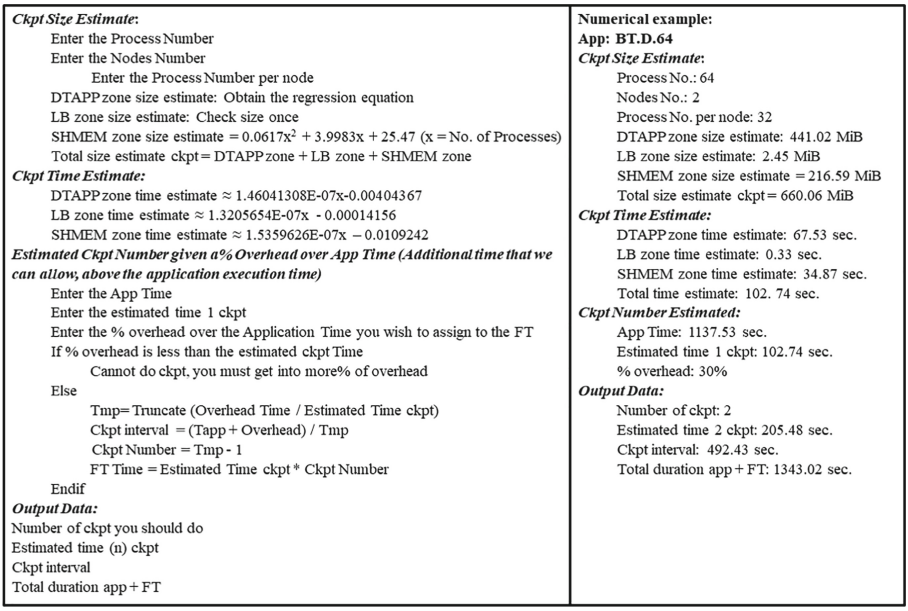

Fig. 4. Approximate model of the number of checkpoints to execute

zone, three regression equations have been obtained which are approximate when there is no congestion in the node, because if there were congestion the values could vary a lot. In order to calculate the estimated checkpoint number given an overhead percentage over the application time, the application execution time, the estimated checkpoint storage time obtained with the previous equations must be known. Then you enter the percentage of overhead you want to have on the execution of the application and you get the total time of the application with fault tolerance, the number of checkpoints that must be performed in a given time interval.

6 Conclusions and Future Work

The defensive I/O that generates fault tolerance directly affects the application, increasing the execution time. We have observed in this article that the pattern generated by the checkpoint is very irregular, which makes it difficult to predict. The storage time of the checkpoint depends on the size and patterns. In this way, knowledge of these spatial and temporal patterns will allow us to predict the size and storage time of the checkpoint so that with this information you can calculate the number of approximate checkpoints that can be made in a given time, when there is no congestion in the node. In addition, it allows us to establish policies and develop tools that help to replicate their behavior in any system as well as being able to establish configuration methods and strategies

to reduce the overload generated by the fault tolerance I/O. In this way, as a future work, we plan to obtain a more exact model of prediction, taking into consideration the elements that affect the congestion of the node. In addition, we will continue to research with other types of fault tolerance strategies and develop utilities that focus on the behavior of defensive I/O in order to assess its impact and reduce it.

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) UE and partially funded by a research collaboration agreement with the Fundación Escuelas Universitarias Gimbernat (EUG).

References

1. Subramaniyan, R., Grobelny, E., Studham, S., George, A.D.: Optimization of checkpointing -related I/O for high-performance parallel and distributed computing. J. Supercomput. **46**, 150–180 (2008)
2. Arunagiri, S., Daly, J.T., Teller, P.J.: Modeling and analysis of checkpoint I/O operations. In: Al-Begain, K., Fiems, D., Horváth, G. (eds.) ASMTA 2009. LNCS, vol. 5513, pp. 386–400. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02205-0_27
3. Shahzad, F., Wittmann, M., Zeiser, T., Hager, G., Wellein, G.: An evaluation of different I/O techniques for checkpoint/restart. In: 2013 IEEE International Symposium on Parallel Distributed Processing, pp. 1708–1716 (2013)
4. del Rosario, J.M., Bordawekar, R., Choudhary, A.: Modeling and analysis of checkpoint I/O operations. ACM SIGARCH Comput. Archit. News **21**, 31–38 (1993)
5. Jason, A., Arya, K., Cooperman, G.: DMTCP: transparent checkpointing for cluster computations and the desktop. In: 23rd IEEE International Parallel and Distributed Processing Symposium (2007)
6. Carns, P., et al.: Understanding and improving computational science storage access through continuous characterization. ACM Trans. Storage (TOS) **7**, 1–26 (2011)
7. Gomez-Sanchez, P., Mendez, S., Rexachs, D., Luque, E.: PIOM-PX: a framework for modeling the I/O behavior of parallel scientific applications. In: Kunkel, J.M., Yokota, R., Taufer, M., Shalf, J. (eds.) ISC High Performance 2017. LNCS, vol. 10524, pp. 160–173. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67630-2_14
8. The strace developers. Strace, 2001–2019. https://strace.io/
9. Bailey, D.H., et al.: The NAS parallel benchmarks. Int. J. High Perform. Comput. Appl. **5**, 63–73 (1991)
10. Leon, B., Franco, D., Rexachs, D., Luque, E.: Impact of the checkpoint on the scalability of the parallel applications. In: The 2019 International Conference on High Performance Computing & Simulation (HPCS 2019)(Accepted) (2019)