Identificación del Comportamiento de Entrada Salida del Checkpoint Coordinado

Betzabeth León[1], Daniel Franco[1], Dolores Rexachs[1], Emilio Luque[1]

Resumen— **La redundancia es uno de los métodos utilizados en HPC, mediante el cual se emplea información, tiempo y recursos adicionales para asegurar el correcto funcionamiento del sistema, De esta manera uno de los protocolos utilizados son los *checkpoint*, los cuales permiten recuperar la información procesada antes del fallo y continuar con la ejecución de la aplicación. Este tipo de protocolos poseen diversos modelos que presentan a su vez características de comportamiento particulares. En el presente artículo se abordarán los *checkpoint* coordinados, exponiendo una descripción sobre su funcionamiento, características y patrones de E/S generados.**

Palabras clave—**Tolerancia a Fallos, HPC, *Checkpoint* Coordinado, *Rollback Recovery*.**

I. INTRODUCCIÓN

Hoy en día, con los avances tecnológicos que requieren de más prestaciones tanto en hardware como en software, la tolerancia a fallos se ha convertido en un elemento fundamental sobre todo en ambientes de Computación de Altas Prestaciones (HPC), debido a que su influencia permite a los sistemas seguir operando, manteniendo la disponibilidad y evitar fallos graves en el funcionamiento, pudiendo incluso anticiparse a condiciones de excepción y generando las soluciones adecuadas para hacerle frente a estas situaciones, a través de recuperaciones de estado e información obtenida de los procesos.

Según [1] la idea básica detrás de cualquier esquema de tolerancia a fallas es tener alguna forma de redundancia. Esta redundancia puede ser en forma de hardware, software adicional o redundancia temporal. De esta manera, la redundancia es uno de los métodos utilizados en HPC, mediante el cual se emplea información, tiempo y recursos adicionales para asegurar el correcto funcionamiento del sistema, esta redundancia es considerada la protección y una fase entre las técnicas de Tolerancia a Fallos, otras de las fases son la detección del error, la estimación de daños o diagnosis, la recuperación del error y la reconfiguración para poder continuar con el servicio del sistema.

En cuanto a los modelos de recuperación y reconfiguración se encuentran el de recuperación directa o hacia adelante, en donde se avanza desde un estado erróneo a uno correcto haciendo correcciones sobre partes del estado y la recuperación inversa o hacia atrás (*rollback recovery*) en donde se retrocede a un estado anterior correcto, guardado previamente, en este caso, es a través de los *checkpoints* que se guarda la información del estado de un proceso de manera periódica en un sistema de almacenamiento estable, suspendiendo la ejecución de este mientras lo salva y consumiendo recursos de Entrada y Salida (E/S), así como ancho de banda de la red.

En [2] describieron diferentes enfoques para protocolos de *rollback-recovery*, indicando que los protocolos de *checkpoint* requieren que los procesos tomen *checkpoint* periódicos con diversos grados de coordinación, entre estos grados de coordinación se encuentra el *checkpoint* coordinado. Por otro lado, existen diferentes alternativas en el momento de almacenar el checkpoint, utilizar almacenamiento local o global, comprimir o no comprimir, en el presente artículo se caracterizará el funcionamiento de diferentes alternativas que pueden ser configurados en el momento de realizar los *checkpoint* coordinados generados por la librería DMTCP (*Distributed MultiThreaded Checkpointing*), exponiendo sus características y funcionamiento. Este artículo se estructura de la siguiente forma:

La sección 2 se refiere a los *checkpoint* coordinados y sus características, en la sección 3 se abordan algunos de los trabajos relacionados con la presente investigación. En la sección 4 se describe la metodología para el análisis de los patrones de E/S de los *checkpoint* coordinados, luego en la sección 5 se refiere a los resultados de los experimentos sobre los patrones obtenidos. En la sección 6 presentamos las conclusiones, para culminar planteando los trabajos futuros.

II. CHECKPOINT COORDINADO

Como se mencionó anteriormente los *checkpoint* almacenan información periódicamente del estado de un proceso, en particular con los *checkpoint* coordinados todos los procesos deben sincronizarse para tener una línea de recuperación consistente y para crear un estado global coherente, esto simplifica la recuperación y disminuye el *overhead* que puede causar un esquema de gestión de basura, debido a que no es necesario en este tipo de protocolo. Sin embargo, la coordinación es un proceso costoso y pudiera representar problemas de escalabilidad.

Así mismo, como se observa en la Figura 1, el coordinador transmite una solicitud a todos los demás procesos para que realicen checkpoint. Al recibir una solicitud de checkpoint, los procesos cuando no existan mensajes en tránsito suspenden su ejecución, toman un *checkpoint* lo almacenan o envían a una memoria estable y se confirma el mensaje al coordinador. Después de que el coordinador recibe los mensajes de validación de todos los procesos, transmite un mensaje de confirmación. Al recibir este mensaje todos los procesos reemplazan el antiguo *checkpoint* con el *checkpoint* temporal y reanudan la ejecución.

[1] Departamento de Arquitectura del Computador y Sistemas Operativos, Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain {betzabeth.leon, daniel.franco, dolores.rexachs, emilio.luque}@uab.es

En este sentido, todos los procesos deben suspender su ejecución y esperar la confirmación de que el estado se ha guardado de manera estable para poder reanudar la ejecución de los procesos. De esta manera, el tiempo de almacenamiento en este tipo de protocolo es un factor crítico, debido a que puede afectar el rendimiento en grandes sistemas.

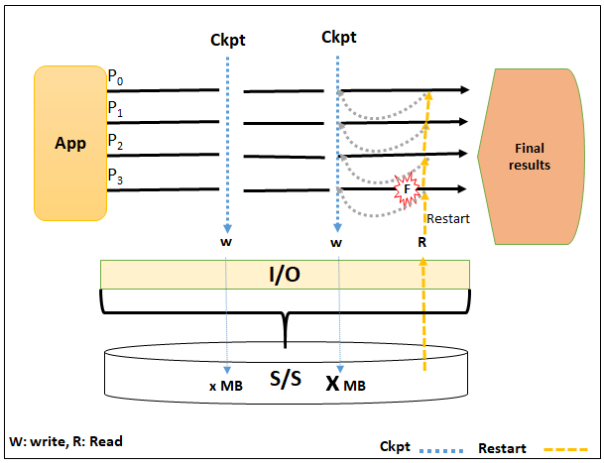

Fig. 1: Checkpoint Coordinado

En general los *checkpoint* como lo indica [3] son una operación de escritura intensiva, debido a que realizan más operaciones de entrada cada vez que requieren guardar de forma periódica el estado del *checkpoint.* Con respecto a las operaciones de lectura, se realizan exclusivamente para recuperar el proceso cuando existe un *restart*. En relación a los *checkpoint* coordinados la lectura, la vuelta atrás, la restauración se debe hacer a todo el conjunto completo de procesos.

Así mismo, la escalabilidad que presentan los *checkpoint* coordinados depende de todos los procesos que deben participar en cada *checkpoint* (escritura) y en cada recuperación (lectura) pudiendo generar un cuello de botella en los sistemas a gran escala. Asunto por el cual el sistema de ficheros debe lidiar con el número de procesos, debido a que cuando aumenta el número procesos, habrá más sobrecarga por todos requerir almacenar de manera simultánea.

III. TRABAJOS RELACIONADOS

Los protocolos de *rollback recovery* como estrategias de Tolerancia a Fallos son importantes para poder mantener la disponibilidad de los sistemas, pero esto implica un costo que se traduce en *overhead*, de esta manera existen una serie de trabajos de investigación que se han dedicado a estudiar el funcionamiento de estos protocolos y a proponer soluciones.

En [4] exponen sobre el *checkpoint* coordinado que tiene la ventaja de un *overhead* muy bajo, siempre y cuando la ejecución permanezca libre de fallas. Por el contrario, el *checkpoint* no coordinado debe complementarse con un protocolo de registro de mensajes (*log* de mensajes), que agrega una penalización significativa para todas las transferencias de mensajes. Los inconvenientes del *checkpoint* coordinado son el costo de sincronización antes de ser realizado el *checkpoint* y el costo de reinicio después de una falla. Así mismo, exponen que el *log* de mensajes no sufre de estos problemas, ya que procesan el *checkpoint* y reinician independientemente de los demás. Estas diferencias sugieren que el mejor enfoque depende de la frecuencia de la falla. Podemos observar que en el *checkpoint* coordinado todos los procesos escribirán o leerán en el sistema de almacenamiento concurrentemente. En el caso de los *checkpoints* no coordinados el momento de escribir o leer del intervalo de checkpoint de cada proceso y del patrón de comunicación.

En [5] indican que las técnicas actuales de tolerancia a fallas en HPC se centran en formas reactivas de mitigar fallas, es decir, a través de *Checkpoint* y *Restart* (C/R). Además de la sobrecarga de almacenamiento, la recuperación de fallas basada en C/R tiene un costo adicional en términos del rendimiento de la aplicación porque la ejecución normal se interrumpe cuando se ejecutan *Checkpoints.* Así mismo, indican que los estudios han demostrado que las aplicaciones que se ejecutan a gran escala pueden gastar más del 50% de su tiempo total almacenando *checkpoints*, reiniciando y rehaciendo el trabajo perdido.

En [6] presentan una implementación de *checkpoint* de múltiples niveles, combinando *checkpoints* en almacenamiento estable con tipos de *checkpoint* más bajos y menos resistentes, comparando los checkpoint almacenados en la memoria en otros nodos con los checkpoint almacenados localmente. De manera similar en [7] mostraron que para eliminar este cuello de botella originado por el almacenamiento centralizado, propusieron un esquema híbrido de *checkpoint* con *checkpoints* locales y globales.

Así mismo, en [8] presentan otra técnica utilizada para reducir la latencia del *checkpoint* utilizando el almacenamiento basado en pares, estas técnicas eliminan la dependencia del almacenamiento central mediante el uso de almacenamiento local de nodos en los sistemas pares para reducir el estrés en el sistema de archivos central.

En [9] consideraron la atención que se le ha dado a la tolerancia a fallos debido al incremento y la complejidad en los sistemas de computación, así como la importancia de reducir la sobrecarga de *checkpoints*, para ello abordan en su trabajo que la información de los *checkpoint* comprende un mecanismo factible para reducir las latencias cometidas por los mismos y la sobrecarga de almacenamiento. A partir de un modelo simple para viabilidad de la compresión del *checkpoint*, exponen que: (1) la compresión de datos del *checkpoint* es factible para muchos tipos de aplicaciones científicas que se espera que se ejecuten en sistemas de escala extrema; (2) la viabilidad de la escala de compresión del checkpoint con respecto al tamaño del *checkpoint;* (3) el nivel de usuario versus nivel de sistema y (4) escalas de viabilidad de la compresión del *checkpoint* con el recuento de la aplicación del proceso. Por último, describen el impacto que la compresión del checkpoint podría tener en la futura generación de sistemas de escala extrema.

Vemos que en la literatura se están proponiendo diferentes estrategias de almacenamiento, centralizado, distribuido entre los nodos utilizando el almacenamiento

local, comprimido, sistemas híbridos con checkpoints locales y globales, entre otros, con el objetivo de mejorar el *overhead*. Nosotros proponemos una metodología para analizar el comportamiento de la E/S y poder comparar las diferentes alternativas para seleccionar la más adecuada en función de la aplicación y la configuración del sistema de cómputo

IV. METODOLOGÍA UTILIZADA PARA EL ANÁLISIS

Para realizar el análisis de los patrones de E/S generados por los checkpoint coordinados se siguieron los siguientes pasos:

1.- Ejecutar una aplicación con Tolerancia a Fallos:

En este primer paso se selecciona el tipo de protocolo de tolerancia a fallos ejecutándolo con la aplicación a la cual se quiere trazar para posteriormente obtener los patrones. Cuando se ejecuta la aplicación se obtiene información de los ficheros generados y una traza de los eventos que realizan cada uno de los procesos. Además de ejecutar la aplicación es necesario ejecutar un *restart* para analizar el acceso en el momento de lectura de los ficheros.

La monitorización es a nivel de aplicación, es una monitorización continua, que monitoriza a nivel de POSIX-IO y MPI-IO, se obtiene una traza por proceso y se hace un análisis de toda la traza.

2.- Análisis de la traza de la aplicación:

Se procede luego de ser ejecutada la aplicación con tolerancia a fallos, a seleccionar la información relevante para el análisis del comportamiento de la E/S para su procesamiento. Para ello se analiza el comportamiento a nivel de ficheros. Se identifican cuantos ficheros se generan, que procesos generan cada uno de los ficheros, en que momento lo generan y el patrón de acceso (operación, momento en el que ocurre, tamaño)

Luego de ejecutar cada una de las aplicaciones con tolerancia a fallos, se procede analizar la traza, como se observa en las figuras 2 y 3, entre la información obtenida se encuentra:

- Rank: Id del proceso.
- File: Id del fichero.
- Posix: Nombre de la operación.
- Namefile: Nombre del fichero.
- Ioopcounter: Contador de operaciones de E/S.
- Wtime: Tiempo.
- Rs: Request Size.
- Tick: Contador de eventos MPI.
- Subtick: Contador de eventos Posix.
- PID: Identificador del proceso.

Se analizan las trazas generadas al ejecutar la aplicación y al ejecutar el *restart* de la aplicación.

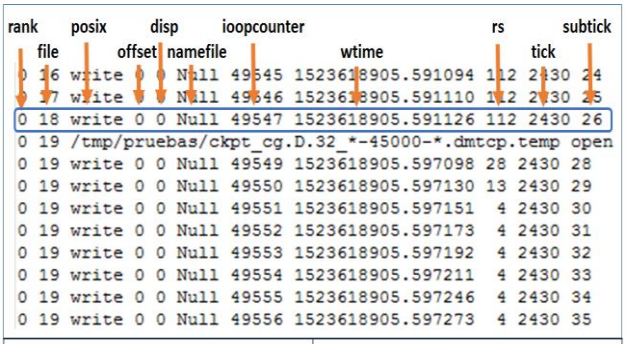

Fig. 2 Traza obtenida de la Herramienta PIOM-PX [10]

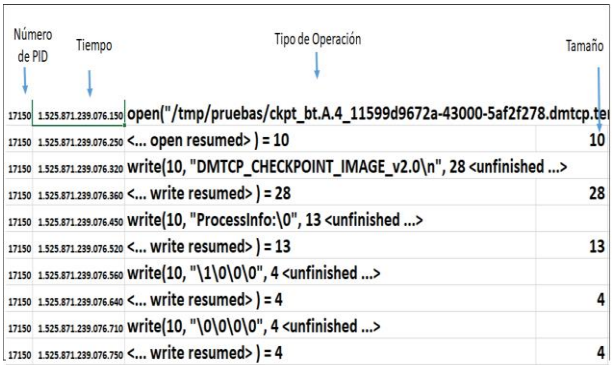

Fig. 3 Traza obtenida de la Herramienta STRACE

3.- Descripción del comportamiento de la tolerancia a fallos:

En esta etapa se hace una interpretación de los resultados obtenidos de las trazas. Se da información del número de ficheros, el tamaño de los ficheros, quien genera los ficheros, el impacto de comprimir o no, se analiza el comportamiento de E/S que permite identificar los tamaños de los ficheros y así analizar si caben en memoria o es necesario almacenarlos en el sistema de E/S, del mismo modo se puede observar:

- Número de procesos utilizados.
- Número de ficheros de *checkpoint* y de *restart* generados. Capacidad de almacenamiento requerida por la aplicación para la tolerancia a fallos.
- Tamaño de cada fichero de *checkpoint* y de *restart*.
- Tipo de Acceso.
- Número de ficheros por proceso.
- Número y comportamiento de las ráfagas identificadas.

V. PATRONES DE E/S GENERADOS POR LOS CHECKPOINT COORDINADOS

Para observar el comportamiento de estos ficheros tanto en formato comprimido como en no comprimido, se realizaron experimentos con tres aplicaciones NAS [11] para poder conocer y comparar sus características. El ambiente de Experimentación utilizado fue el siguiente: Procesador: Intel® Xeon® CPU E5430 @ 2.66 Ghz quadcore 6Mb L2, Nro. Procesadores: 2.

Memoria: 16 GB Fully Buffered DIMMs (FBD) 667 MHz. Trabajando en los experimentos con 1 y 5 nodos.

Siguiendo los pasos de la metodología indicada anteriormente, para poder observar el comportamiento de los patrones generados por este tipo de checkpoint, se obtuvieron los siguientes resultados presentados a continuación:

1. Selección de una aplicación con tolerancia a fallos:

Se utilizó la librería DMTCP [12] para realizar los *checkpoint* coordinados a las aplicaciones elegidas, esta librería realiza los *checkpoint* con un formato comprimido de manera predeterminada, pero incluyendo algunos parámetros se pueden ejecutar de forma no comprimida. Los *checkpoint* fueron generados con ambos formatos tanto comprimidos como no comprimidos, un *checkpoint* desde la misma aplicación para observar la diferencia en el comportamiento.

Luego de ejecutar la aplicación podemos agrupar la información generada desde dos vertientes:

a. Descripción de los ficheros de tolerancia a fallos generados:

En la Tabla I, se muestra la información que se genera al realizar un *checkpoint* coordinado con la DMTCP a la aplicación NAS BT, este experimento se realizó con 4 procesos en 1 nodo, los ficheros de *checkpoint* generados corresponden con el número de procesos, es decir un fichero por proceso, más tres ficheros relativos al *restart* y a la gestión de los procesos.

Tabla I: Checkpoint Coordinado – Programa NAS BT Class A (4 Procesos)

Ficheros generados por la Tolerancia a Fallos Programa NAS bt Class A (Nro. Procesos:4, Nro. Nodos:1)		
No-gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_bt.a.4_*	1 3	74 MiB 73 MiB
ckpt_hydra_pmi_proxy_*	1	19 MiB
ckpt_mpiexec.hydra_*	1	19 MiB
dmtcp_restart_script_*	1	6,5 KiB
Gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_bt.A.4_*	4	28 MiB
ckpt_hydra_pmi_proxy_*	1	2,4 MiB
ckpt_mpiexec.hydra_*	1	2,5 MiB
dmtcp_restart_script_*	1	6,5 KiB

En la Tabla II, se muestra el resultado al ejecutar la aplicación CG con tolerancia a fallos, es decir con un *checkpoint* coordinado con la DMTCP. Este experimento se realizó con 4 procesos en 1 nodo. De igual forma presenta el mismo número de ficheros que en el experimento anterior, debido a que el número de procesos y nodos utilizado fue el mismo.

Tabla II: Checkpoint Coordinado – Programa NAS CG Class A (4 Procesos)

Ficheros generados por la Tolerancia a Fallos Programa NAS cg Class A (Nro. Procesos:4, Nro. Nodos:1)		
No-gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_cg.a.4_*	4	58 MiB
ckpt_hydra_pmi_proxy_*	1	19 MiB
ckpt_mpiexec.hydra_*	1	19 MiB
dmtcp_restart_script_*	1	6,5KiB
Gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_cg.a.4_*	4	14 MiB
ckpt_hydra_pmi_proxy_*	1	2.4 MiB
ckpt_mpiexec.hydra_*	1	2,5 MiB
dmtcp_restart_script_*	1	6,5 KiB

De manera similar, en la Tabla III, se muestra la información obtenida al ejecutar un *checkpoint* coordinado con la DMTCP a la aplicación NAS SP este experimento se realizó también con 4 procesos en un nodo, para así poder observar el comportamiento de estas tres diferentes aplicaciones NAS con tolerancia a fallos.

Tabla III: Checkpoint Coordinado – Programa NAS SP Class A (4 Procesos)

Ficheros generados por la Tolerancia a Fallos Programa NAS sp Class A (Nro. Procesos:4, Nro. Nodos:1)		
No-gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_sp.a.4_*	4	69 MiB
ckpt_hydra_pmi_proxy_*	1	19 MiB
ckpt_mpiexec.hydra_*	1	19 MiB
dmtcp_restart_script_*	1	6,5KiB
Gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_sp.a.4_*	4	22 MiB
ckpt_hydra_pmi_proxy_*	1	2,4 MiB
ckpt_mpiexec.hydra_*	1	2,5 MiB
dmtcp_restart_script_*	1	6,5 KiB

Como se puede observar en las tres tablas presentadas anteriormente, al ejecutar un *checkpoint* en cada una de las aplicaciones mencionadas, se generó un fichero por cada proceso, así como otros ficheros de menor tamaño que son dependientes de las librerías utilizadas.

Seguidamente para observar las diferencias o similitudes con aplicaciones que manejen mayor volumen de información, de procesos y de nodos, se ejecutó nuevamente un *checkpoint* coordinado con la aplicación NAS SP, pero esta vez con la Clase D y con 36 procesos, distribuidos en 5 nodos (Tabla IV). Al tener muchos más datos, el tamaño de los *checkpoint* aumentó considerablemente con respecto a los resultados mostrados anteriormente.

Tabla IV: Checkpoint Coordinado – Programa NAS SP Class D (36 Procesos- 5 Nodos)

Ficheros generados por la Tolerancia a Fallos Programa NAS sp Class D (Nro. Procesos:36, Nro. Nodos:5)		
No-gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_sp.d.36_*	3 5 4 24	683 MiB 682 MiB 678 MiB 677 MiB
ckpt_hydra_pmi_proxy_*	1 4	23 MiB 2,5 MiB
ckpt_mpiexec.hydra_*	1	23 MiB
dmtcp_restart_script_*	1	15 KiB
ckpt_dmtcp_ssh	4	21 MiB
ckpt_dmtcp_sshd	2 2	2,2 MiB 2,3 MiB
Gzip		
Nombre	**Número de Ficheros**	**Tamaño**
ckpt_sp.d.36_*	4 5 13 14	480 MiB 479 MiB 478 MiB 477 MiB
ckpt_hydra_pmi_proxy_*	1 4	4,1 MiB 2,5 MiB
ckpt_mpiexec.hydra_*	1	4,2 MiB
dmtcp_restart_script_*	1	15 KiB
ckpt_dmtcp_ssh	4	3,9 MiB
ckpt_dmtcp_sshd	2 2	2,2 MiB 2,3 MiB

Como se observa en la tabla IV, además de generar un *checkpoint* por proceso con ficheros grandes, de tamaños similares, también se generaron quince ficheros de menor tamaño dependientes de las librerías, que sirven para la coordinación de los *checkpoint* en caso de que se tuviese que iniciar un proceso de recuperación, [13] además de esto se genera el fichero de *restart* (dmtcp_restart_script_*.sh), el cual se encarga de restaurar el sistema y es escrito por el coordinador en su propio directorio local. Otro fichero generado al igual que en los experimentos anteriores fue el ckpt_hydra_pmi_proxy_*.dmtcp, en donde los *proxy* envían información de E/S desde los procesos de la aplicación a un *proxy* principal o al administrador de procesos. Otros ficheros que se generan al momento de crear el *checkpoint* son el ckpt_mpiexec.hydra.*.dmtcp . y cuando se trabaja con varios nodos como fue el caso de este experimento en que se utilizaron cinco nodos, en el coordinador adicionalmente se crea el fichero ckpt_dmtcp_ssh_*.dmtcp y en los clientes se origina el fichero ckpt_dmtcp_sshd_*.dmtcp, los cuales almacenan información referente a la comunicación. El tamaño de estos ficheros adicionales es pequeño con respecto al tamaño de los *checkpoint* generados.

Así mismo, se pudo observar que en los ficheros hydra_pmi_proxy los de mayor tamaño corresponden a los que se generaron en el nodo en donde fue lanzado el *checkpoint* con la aplicación, del mismo modo se puede observar que el número de ficheros de tipo ckpt_dmtcp_sshd y Nº de ckpt_dmtcp_ssh = Nº de nodos -1.

Por lo tanto, la distribución de estos ficheros es lógica con respecto al entorno de experimentación utilizado, así como el tamaño es pequeño en comparación con el tamaño generado por los ficheros de *checkpoint*. También se puede observar que existe gran diferencia en cuanto al tamaño de los ficheros entre los que fueron ejecutados en formato comprimido (Gzip), como los que se realizaron de manera no comprimida (No-gzip).

2. Análisis de la Traza de la Aplicación:

Luego de ejecutar cada una de las aplicaciones mencionadas anteriormente con tolerancia a fallos, se procedió analizar la traza. Entre la información obtenida se encuentra:

Traza del Programa NAS BT:

- Número de procesos utilizados: 4
- Número de ficheros de *checkpoint* generados: 4 ficheros uno por cada proceso, (más 2 de gestión y restauración) generados por los procesos: 17135, 17138, 17147, 17148, 17149 y 17150.
- Capacidad de almacenamiento requerida por la aplicación para la tolerancia a fallos: 331 MiB
- Nombre del fichero del *checkpoint*: ckpt_bt.A.4.*
- Tamaño de cada fichero de *checkpoint:* entre 73 MiB y 74 MiB.
- Tipo de Acceso: escritura (*checkpoint*) y lectura (*restart*)
- Número de ficheros por proceso: 1 fichero por proceso.
- Número de Ráfagas identificadas en la escritura: 2.
- Número de Ráfagas identificadas en la lectura: 2.

Traza del Programa NAS CG:

- Número de procesos utilizados: 4
- Número de ficheros de *checkpoint* generados: 4 ficheros uno por cada proceso, (más 2 de gestión y restauración) generados por los procesos: 17220, 17223, 17232, 17233, 17234 y 17235.
- Capacidad de almacenamiento requerida por la aplicación para la tolerancia a fallos: 270 MiB
- Nombre del fichero del *checkpoint*: ckpt_cg.A.4.*
- Tamaño de cada fichero de *checkpoint*: 58 MiB.
- Tipo de Acceso: escritura (*checkpoint*) y lectura (*restart*)
- Número de ficheros por proceso: 1 fichero por proceso.
- Número de Ráfagas identificadas en la escritura: 2.
- Número de Ráfagas identificadas en la lectura: 2.

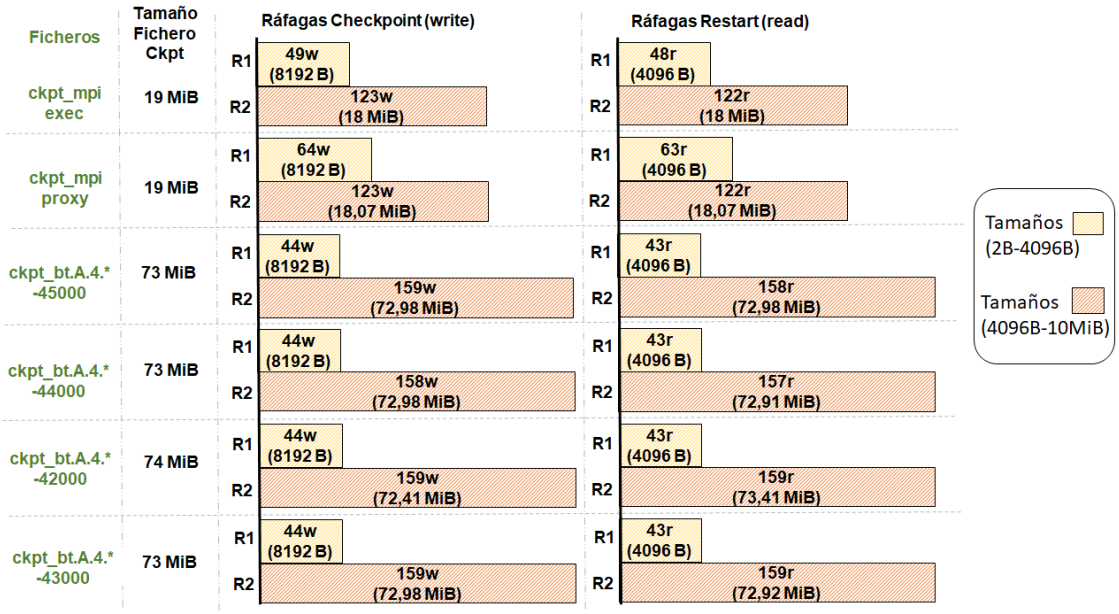

Fig. 5. Patrones de E/S Checkpoint Coordinado (No-gzip) – Restart, Programa NAS BT

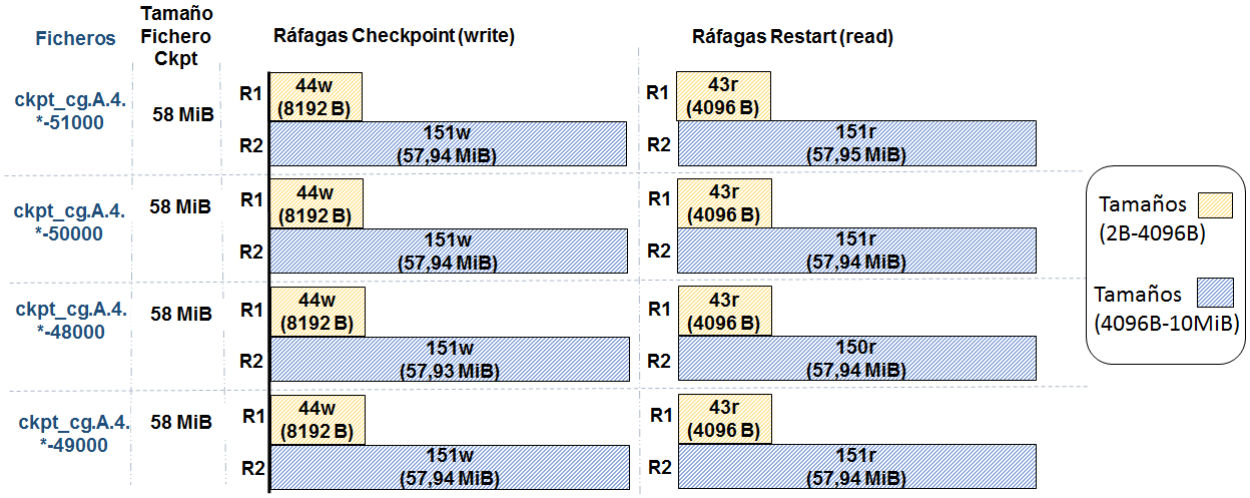

Fig. 6. Patrones de E/S Checkpoint Coordinado (No-gzip)– Restart, Programa NAS CG

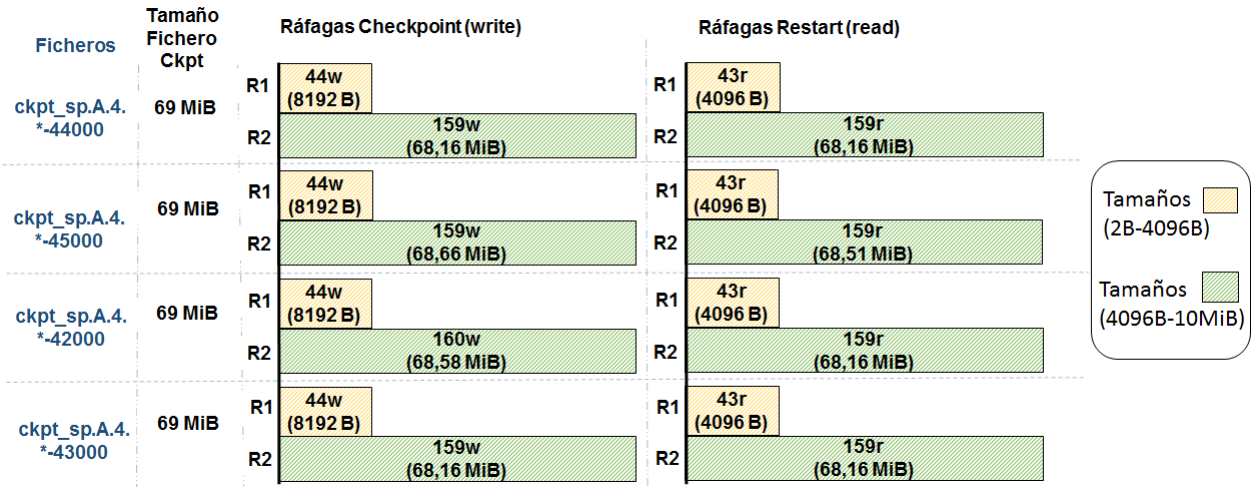

Fig. 7. Patrones de E/S Checkpoint Coordinado (No-gzip)– Restart, Programa NAS SP

Traza del Programa NAS SP:

- Número de procesos utilizados: 4
- Número de ficheros de *checkpoint* generados: 4 ficheros uno por cada proceso, (más 2 de gestión y restauración) generados por los procesos: 16283, 16286, 16295, 16296, 16297 y 16298.
- Capacidad de almacenamiento requerida por la aplicación para la tolerancia a fallos: 314 MiB
- Nombre del fichero del *checkpoint*: ckpt_sp.A.4.*
- Tamaño de cada fichero de *checkpoint*: 69 MiB.
- Tipo de Acceso: escritura (*checkpoint*) y lectura (*restart*)
- Número de ficheros por proceso: 1 fichero por proceso.
- Número de Ráfagas identificadas en la escritura: 2.
- Número de Ráfagas identificadas en la lectura: 2.

3. Descripción del comportamiento de la Tolerancia a fallos:

A partir del número de procesos utilizados, el número de ficheros generados la capacidad de almacenamiento requerida analizando el tamaño de cada fichero de *checkpoint* y *restart*, el tipo de acceso y el número de ráfagas generadas, a continuación se hace una descripción detallada del comportamiento de E/S emanado de cada uno de los ficheros, para los programas NAS BT, CG y SP, Clase A, representando de cada aplicación la siguiente información:

Con respecto al programa BT Clase A, ejecutado en un nodo, se observaron dos ráfagas, tanto en el checkpoint como en el *restart*. Como se observa en la figura 5, los ficheros ckpt_mpiexec y ckpt_mpi_proxy, la ráfaga 1 (R1) mantiene el mismo tamaño de las ráfagas 1 de los ficheros de checkpoint, es decir de 8192B para el checkpoint y 4096B para el *restart.* Del mismo modo se comportaron las ráfagas 1 de estos mismos ficheros de ckpt_mpiexec y ckpt_mpi_proxy en los programas NAS CG y SP, por lo cual no se reflejaron en las figuras 6 y 7, debido a que son iguales.

En el caso de la ráfaga 2 del programa BT Clase A si existe diferencia de tamaños entre los dos ficheros nombrados anteriormente (ckpt_mpiexec y ckpt_mpi_proxy) y los ckpt_bt.A.4-* con aproximadamente un tamaño de 73MiB, siendo este último de un tamaño mayor. Esta diferencia de tamaños resalta que los más pequeños son ficheros que dependen de las librerías necesarias para ejecutar la aplicación y de las librerias de Tolerancia a Fallos y los más grandes, son los *checkppoint* en sí, que se encargan de guardar el estado completo del proceso y dependen del tamaño de los datos de la aplicación.

También es importante acotar que el número de *writes* (escritura) en la primera ráfaga en todos los casos del checkpoint se mantuvo en 44 *writes* y en el caso de los *restart* para todos los programa se mantuvo en 43 *read* (lectura).

En el caso de los programas NAS CG y SP, figuras 6 y 7, se puede observar que los tamaños de sus segundas ráfagas oscilan para el CG en 57 MiB, y para el SP los tamaños son de 68 MiB. De manera similar, como se observa en las tres figuras en el caso de las segundas ráfagas de los *restart (read)*, los tamaños son equivalentes a los de las segundas ráfagas de los *checkpoint (write)* y también al tamaño de los ficheros de *checkpoint*. Esto muestra una correlación entre lo que ha sido escrito para almacenar la información del estado del proceso y lo que se ha leído para restaurarlo.

Así mismo, los patrones de E/S también presentaron un comportamiento similar, al identificarse dos ráfagas en cada proceso de escritura en los *checkpoint* y de lectura en los *restart*.

VI. CONCLUSIONES

Los *checkpoint* coordinados son una forma de redundancia temporal que utiliza la tolerancia a fallos para mantener en funcionamiento los sistemas en caso de que ocurra un fallo. En el presente documento se hizo una descripción del comportamiento de este tipo de *checkpoint*, los cuales fueron ejecutados utilizando la librería DMTCP, con formatos comprimidos y no comprimidos y se observó que existe gran diferencia con respecto a los tamaños de los ficheros generados. Así mismo, fueron identificadas dos ráfagas de escrituras continuas de diversos tamaños de bloque (*request size*) para la primera ráfaga pequeños (<512B) y medianos (<4kB) y para la segunda ráfaga de tamaños medianos (>4kB) y grandes (<372 MB) , en el caso de los *checkpoint* y dos ráfagas de lectura continua en el caso del *restart* de similares tamaños a los generados con los *checkpoint.* De esta manera, se pudo observar una gran regularidad en el comportamiento de estos patrones, al coincidir el tamaño de las operaciones de entrada (escritura) de los *checkpoint*, con el tamaño de las operaciones de salida (lectura) del *restart* y con el tamaño de los ficheros de *checkpoint* que fueron generados en todos los casos estudiados.

VII. TRABAJOS FUTUROS

Conocer en profundidad la gestión del almacenamiento estable es importante, para así poder aplicar las acciones necesarias que tengan que ver con la reducción del *overhead* al aplicar la tolerancia a fallos, en este trabajo nos hemos centrado en la caracterización de *checkpoints* coordinados, y estamos también observando la caracterización de otros tipos de patrones generados por la tolerancia a fallos, para así poder identificar su comportamiento y poder ofrecer métodos que ayuden a optimizar la gestión del almacenamiento de este tipo de patrones.

AGRADECIMIENTOS

Esta investigación ha sido financiada mediante MICINN/MINECO España bajo contrato TIN2014-53172-P y TIN2017-84875-P.

REFERENCIAS

[1] Sohi, G. S., Franklin, M., & Saluja, K. (1989). A Study of Time-Redundant Fault Tolerance Techniques for Hig-Performance Pipelined Computers. The Nineteenth International Symposium on

Fault-Tolerant Computing. Digest of Papers. Obtenido de http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=105616

[2] Elnozahy, E. M., Alvisi, L., Wang, Y.-M., & Johnwon, D. B. (2002). A Survey of Rollback-Recovery Protocols in Message-Passing Systems. Computer Science. The Univesity of Texas at Austin, 275-408. Obtenido de https://www.cs.utexas.edu/~lorenzo/papers/SurveyFinal.pdf

[3] Al-Kiswany, S., Ripeanu, M., Vazhkudai, S. S., & Gharaibeh, A. (2008). Stdchk: A Checkpoint Storage System for Desktop Grid Computing. The 28th International Conference on Distributed Computing Systems. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4595934

[4] Bouteiller, A., Lemarinier, P., Krawezik, G., Cappello, F. (2003) Coordinated checkpoint versus message log for fault tolerant MPI. Cluster Computing, 2003. Proceedings. 2003. IEEE International Conference. Obtenido de: https://ieeexplore.ieee.org/abstract/document/1253321/

[5] James, E., Kharbas, K., Fiala, D., Mueller, F., Ferreira, K., & Engelmann, C. (2012). Combining Partial Redundancy and Checkpointing for HPC. IEEE 32nd International Conference on Distributed Computing Systems. Obtenido de http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6258034&tag=1

[6] Moody, A., Bronevetsky, G., Mohror, K., Supinski, B. (2010) Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. http://ieeexplore.ieee.org/document/5645453/

[7] Dong, X., Xie, Y., Muralimanohar, N., Jouppi, N. Hybrid Checkpointing Using Emerging Nonvolatile Memories for Future Exascale Systems. (2011). https://www.researchgate.net/publication/220170043_Hybrid_Checkpointing_Using_Emerging_Nonvolatile_Memories_for_Future_Exascale_Systems

[8] Hursey, J., Lumsdaine, A. (2010) A Composable Runtime Recovery Policy Framework Supporting Resilient HPC Applications. https://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR686

[9] Ibtesham, D., Arnold, D., Bridges, P. G., Ferreira, K. B., & Brightwell, R. (2012). On the Viability of Compression for Reducing the Overheads of Checkpoint/Restart-based Fault Tolerance. 41st International Conference on Parallel Processing. Obtenido de http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6337576&tag=1

[10] Gómez-Sánchez, P., Méndez, S., Rexachs, D., Luque, E. (2017). PIOM-PX: A Framework for Modeling the I/O Behavior of Parallel Scientific Applications. , pages 160–173. Springer International Publishing, Cham, 2017

[11] Bailey, D., Barszcz. E., Barton J., Browning, D. (1991). The NAS Parallel Benchmarks. International Journal of Supercomputer Applications.Volume 5, No. 3, pp. 63-73.

[12] DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. Jason Ansel, Kapil Arya and Gene Cooperman.
23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09). 12 pages, Rome, Italy. May, 2009.

[13] Ibtesham, D., Arnold, D., Bridges, P. G., Ferreira, K. B., & Brightwell, R. (2012). On the Viability of Compression for Reducing the Overheads of Checkpoint/Restart-based Fault Tolerance. 41st International Conference on Parallel Processing.