Realización del producto de matrices usando checkpoint

Arquitecturas Tolerantes a Fallos

Curso 2012/2013

Pereira Guerra, Adrián <adrian.pereira@udc.es>
https://github.com/adrisons/ATF

He decidido implementar el checkpoint mediante dos ficheros:

`result.csv` Que almacena cada posición de la matriz resultado en formato csv

`checkPoint.txt` Que va almacenando la última posición almacenada en `result.csv`

Por cada posición que se calcula de la matriz resultado, su valor se almacena en el fichero resultado y su posición en el fichero de checkpoint.

El fichero de checkpoint sólo existe mientras el calculo de la matriz resultado no se ha realizado por completo y, además, almacena el path de las dos matrices que se están multiplicando, para controlar que sólo se realice la recuperación si las matrices a multiplicar son las mismas.

Si la ejecución termina inesperadamente, cuando se intenta ejecutar el programa con las mismas matrices, se restaura el estado del sistema utilizando el checkpoint y se continúa ejecutando.

Durante la ejecución del programa se pueden dar errores relacionados con los ficheros `result` y `checkpoint`. Por ejemplo, si tratamos con matrices de gran volumen, podría pasar que se guardara el estado en `checkpoint` pero que un error externo interrumpiese mientras se guarda en el fichero `result` y tener valores incorrectos.
Con el objetivo de reducir la posibilidad de este error es por lo que se guarda cada posición calculada de la matriz resultado en `result`, en vez de almacenar toda la matriz resultado en cada iteración. De este modo las escrituras son verdaderamente rápidas, ya que sólo almacenan un número en cada iteración. Para añadir más tolerancia a fallos, se comprueba además la matriz resultado almacenada hasta el momento. Si el nº de filas y de columnas coinciden con los datos del checkpoint, se continúa como antes, si no, se continúa en donde se ha quedado la matriz resultado.

Como vemos en la figura , al tener que almacenar la información de checksum empeoran los tiempos de ejecución.

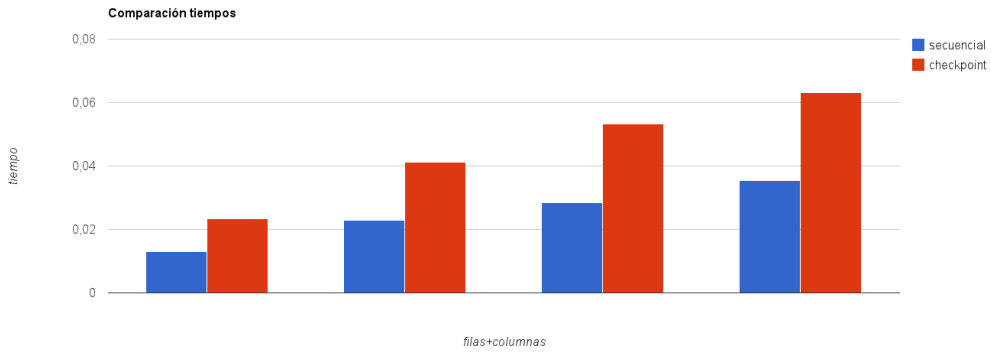

Figura 1: Comparación tiempos secuencial sin/con checksum sin fallo

Cuando hay un fallo y el programa tiene que cargar los datos a partir del checksum generado previamente, la carga realizada también tiene una repercusión sobre el tiempo de ejecución. Esto quiere decir que, si este coste es muy alto, en algunos casos será conveniente comenzar de cero en vez de reconfigurar el sistema.

En nuestro caso en particular, por la propia definición de la estructura de los ficheros `checkopoint` y `result`, sólo es necesaria la última fila de éstos para la reconfiguración, y los nombres de los ficheros de matrices para el checkpoint, que son las dos primeras filas; por lo que no se trata con grandes volúmenes de datos. Debería ser muy efectiva la reconfiguración y ser conveniente en la mayoría de los casos. En los casos que puede ser menos conveniente realizar la reconfiguración es en los que se han procesado pocos datos, por lo tanto es más eficiente calcular esos datos que reestaurar el sistema.

Esta teoría se prueba ejecutando el programa con matrices de $25 * 25$. Son matrices relativamente pequeñas y, en las pruebas realizadas, se ha ido parando el programa en diferentes puntos, calculando el tiempo de recuperación.

El coste de reconfigurar el sistema es del orden de $3 * 10^{-4}$. En la implementación final del sistema, se decide guardar en el fichero de checkpoint, no sólo el último estado, si no todos los estados guardados. Para detectar fallos en caso de error. Con esta modificación, el fichero de checkpoint se hace más grande resultando en variaciones en el tiempo de recuperación, como se muestra en la figura

Pero lo importante es que, si guardamos los últimos estados del sistema (no todos), el rendimiento es bueno (tiempos de recuperación del orden de $3 * 10^{-4}$).

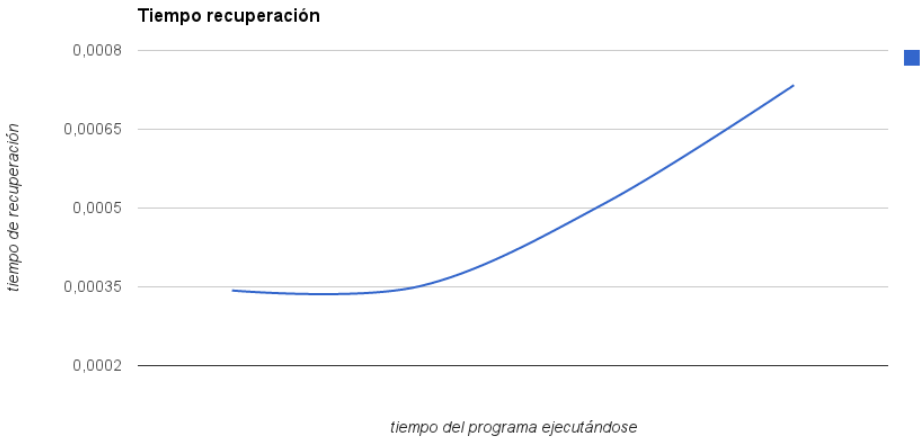

Figura 2: Tiempos de recuperación con fichero checksum con histórico