Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Computación Tolerante a Fallas

Maestro: Michel Emanuel Lopez Franco

Juan Antonio Perez Juarez
Carrera: INCO
Código: 215660996

Application checkpointing

El checkpointing es una técnica que proporciona tolerancia a fallos en sistemas de computación. Básicamente, consiste en guardar una instantánea del estado de la aplicación, de modo que esta pueda reiniciarse desde ese punto en caso de fallo. Esto es particularmente importante para aplicaciones de larga duración que se ejecutan en sistemas de computación propensos a fallos.

En un entorno de computación distribuida, el checkpointing es una técnica que ayuda a tolerar fallos que, de otro modo, obligarían a que aplicaciones de larga duración se reinicien desde el principio. La forma más básica de implementar el checkpointing consiste en detener la aplicación, copiar todos los datos necesarios desde la memoria a un almacenamiento confiable (por ejemplo, un sistema de archivos paralelo) y luego continuar con la ejecución. En caso de fallo, cuando la aplicación se reinicia, no necesita comenzar desde cero. En su lugar, leerá el estado más reciente ("el checkpoint") desde el almacenamiento estable y continuará la ejecución desde allí. Aunque existe un debate en curso sobre si el checkpointing es la carga de trabajo de E/S dominante en los sistemas de computación distribuida, hay un consenso general de que el checkpointing es una de las principales cargas de trabajo de E/S.

Existen dos enfoques principales para el checkpointing en los sistemas de computación distribuida: el checkpointing coordinado y el checkpointing no coordinado. En el enfoque de checkpointing coordinado, los procesos deben asegurarse de que sus checkpoints sean consistentes. Esto se logra generalmente mediante algún tipo de algoritmo de protocolo de confirmación en dos fases. En el checkpointing no coordinado, cada proceso guarda su propio estado de forma independiente. Es importante destacar que simplemente forzar a los procesos a guardar su estado en intervalos de tiempo fijos no es suficiente para garantizar la consistencia global. La necesidad de establecer un estado consistente (es decir, sin mensajes perdidos o duplicados) puede obligar a otros procesos a retroceder a sus checkpoints, lo que a su vez puede causar que otros procesos retrocedan a checkpoints aún anteriores, lo que en el caso más extremo puede significar que el único estado consistente encontrado sea el estado inicial (el llamado efecto dominó).

Uno de los medios originales y ahora más comunes de checkpointing de aplicaciones fue la función de "guardar estado" en aplicaciones interactivas, en la cual el usuario de la aplicación podría guardar el estado de todas las variables y otros datos en un medio de almacenamiento mientras lo estaba usando. Luego, podía continuar trabajando o salir de la aplicación y, más tarde, reiniciar y restaurar el estado guardado. Esto se implementaba a través de un comando de "guardar" o una opción de menú en la aplicación. En muchos casos, se convirtió en una práctica

estándar preguntar al usuario si tenía trabajo no guardado al salir de la aplicación, y si quería guardar su trabajo antes de hacerlo.

Este tipo de funcionalidad se volvió extremadamente importante para la usabilidad en aplicaciones donde el trabajo en particular no podía completarse en una sola sesión (como jugar un videojuego que se espera dure decenas de horas, o escribir un libro o un documento largo que sumen cientos o miles de páginas) o donde el trabajo se realizaba durante un largo período de tiempo, como la entrada de datos en un documento, como filas en una hoja de cálculo.

El problema con la función de guardar estado es que requiere que el operador de un programa solicite el guardado. Para programas no interactivos, incluidos los trabajos automatizados o procesados en lotes, la capacidad de realizar un checkpoint de dichas aplicaciones también tuvo que ser automatizada.

A medida que las aplicaciones por lotes comenzaron a manejar decenas o cientos de miles de transacciones, donde cada transacción podía procesar un registro de un archivo contra varios archivos diferentes, la necesidad de que la aplicación fuera reiniciable en algún punto sin tener que ejecutar todo el trabajo desde el principio se volvió imperativa. Así nació la capacidad de "checkpoint/restart" (punto de control/reinicio), en la que después de que se hubieran procesado un número determinado de transacciones, se podía tomar una "instantánea" o "checkpoint" del estado de la aplicación. Si la aplicación fallaba antes del siguiente checkpoint, se podía reiniciar proporcionando la información del checkpoint y el último lugar en el archivo de transacciones donde una transacción se había completado con éxito. La aplicación podría entonces reiniciarse en ese punto.

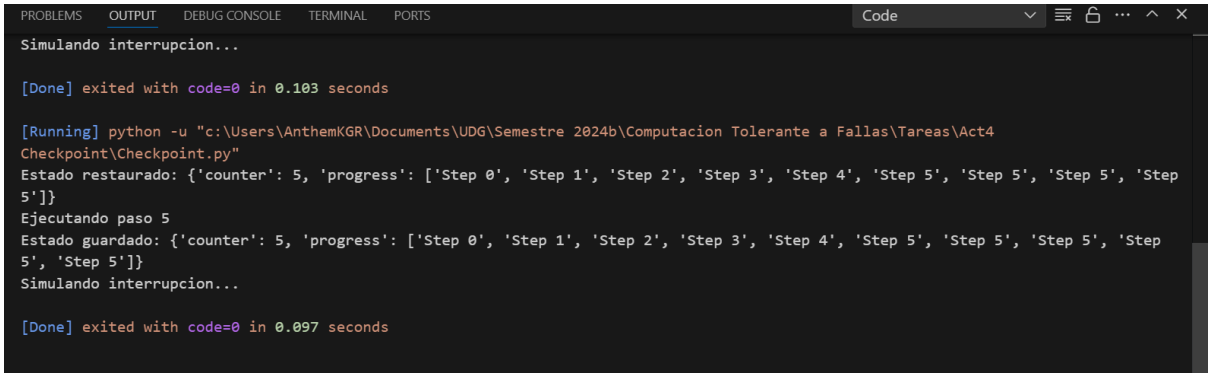
El checkpointing tiende a ser costoso, por lo que generalmente no se realizaba con cada registro, sino en un punto intermedio razonable entre el costo de un checkpoint y el valor del tiempo de computadora necesario para procesar un lote de registros. Por lo tanto, la cantidad de registros procesados para cada checkpoint podría variar entre 25 y 200, dependiendo de factores de costo, la complejidad relativa de la aplicación y los recursos necesarios para reiniciar la aplicación con éxito.

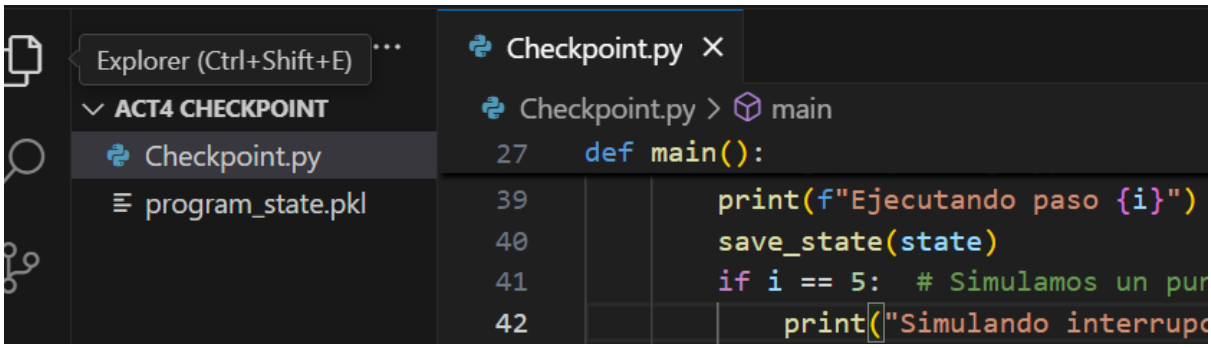
Cree este pequeño programa que hace un intento de checkpoint, me inspiré en su mayoría de un código de stackoverflow.

```
import pickle
import os

# Archivo donde se guardará el estado
STATE_FILE = 'program_state.pkl'

# Estado inicial del programa
state = {
    'counter': 0,
```

```
    'progress': []
}

def save_state(state):
    with open(STATE_FILE, 'wb') as f:
        pickle.dump(state, f)
    print(f"Estado guardado: {state}")

def load_state():
    if os.path.exists(STATE_FILE):
        with open(STATE_FILE, 'rb') as f:
            state = pickle.load(f)
        print(f"Estado restaurado: {state}")
        return state
    else:
        return None

def main():
    global state

    # Intentar restaurar el estado anterior
    previous_state = load_state()
    if previous_state:
        state = previous_state

    # Simulación de ejecución del programa
    for i in range(state['counter'], 10):
        state['counter'] = i
        state['progress'].append(f"Step {i}")
        print(f"Ejecutando paso {i}")
        save_state(state)
        if i == 5:  # Simulamos un punto de restauración en el paso 5
            print("Simulando interrupción...")
            break

if __name__ == "__main__":
    main()
```

Salida en consola:





Referencias:
Wikipedia contributors. (2024, July 31). Application checkpointing. Wikipedia. https://en.wikipedia.org/wiki/Application_checkpointing

Stack Overflow - where developers learn, share, & build careers. (n.d.). Stack Overflow. https://stackoverflow.com/