Universidad EAFIT

Escuela de Ingeniería

Departamento de Ingeniería de Sistemas

Curso: Sistemas Operativos
Docente: Edison Valencia

Informe Técnico – Sistema de Backup y Restauración con Compresión Paralela

Luisa Álvarez

Juan Gómez

Miguel Hoyos

Santiago Neusa

Sebastián Restrepo

Fecha: 12/05/2025

1. Descripción General

Desarrollamos un sistema de respaldo y restauración de archivos diseñado para facilitar la compresión de directorios y su almacenamiento en distintos medios: localmente, en unidades USB y en la nube mediante Google Drive. También permite restaurar los respaldos desde cualquiera de estos orígenes. Todo el sistema opera a través de una interfaz de línea de comandos.

El sistema emplea compresión en formato .zip con soporte para cifrado. La estructura está organizada de manera modular, permitiendo gestionar desde el descubrimiento de archivos hasta la compresión y el almacenamiento, manteniendo la escalabilidad y facilidad de mantenimiento.

Uno de los principales enfoques fue garantizar eficiencia al procesar grandes volúmenes de archivos. Para ello, integramos procesamiento paralelo, tanto en la fase de exploración como en la de compresión, lo cual permitió reducir significativamente los tiempos de ejecución.

---

2. Paralelismo en el Sistema

Para mejorar el rendimiento del sistema ante directorios grandes, incorporamos procesamiento paralelo utilizando la biblioteca Dask. Esta decisión se tomó frente a alternativas como threading o multiprocessing, dado que Dask proporciona una API más robusta para construir flujos de trabajo paralelos y escalables.

En la fase de exploración de archivos, paralelizamos la tarea de recorrer recursivamente el sistema de archivos, dividiendo las carpetas en subárboles procesados en paralelo. Esto resultó en mejoras evidentes en directorios con miles de archivos.

En la compresión, implementamos una compresión paralela por bloques. Cuando el número de archivos supera un umbral definido, estos se dividen entre múltiples trabajadores, los cuales crean archivos comprimidos independientes que luego se consolidan en un solo archivo final. Esto mejora el rendimiento sin sacrificar la integridad de los datos.

---

3. Compresión y Cifrado

Utilizamos el algoritmo **DEFLATE**, el cual ofrece una buena relación entre velocidad de compresión y tamaño final. Para implementarlo empleamos la biblioteca pyzipper, que extiende zipfile con soporte para cifrado **AES-256**. Esto nos permite proteger los respaldos mediante contraseña, lo cual es útil en contextos donde se manejan archivos sensibles.

El sistema permite definir el nivel de compresión, lo cual facilita su adaptación a diferentes requerimientos de espacio y tiempo. También se mantiene la estructura original de los archivos, facilitando la restauración posterior.

---

4. Destinos de Almacenamiento

El sistema permite almacenar los respaldos en tres ubicaciones distintas:

- **Sistema local**: los archivos comprimidos se almacenan en el disco duro del equipo.
- **Unidades USB**: detectamos automáticamente dispositivos de almacenamiento extraíbles conectados al sistema y permitimos seleccionar uno como destino. Esto es útil para respaldos portables sin conexión a internet.
- **Google Drive**: mediante la API oficial, autenticamos al usuario y permitimos subir los respaldos a su cuenta. También listamos y descargamos respaldos existentes.

---

5. Interfaz de Usuario

El sistema se controla completamente desde consola, utilizando *argparse* para el procesamiento de argumentos. Incluimos comandos como:

- `--backup`: permite seleccionar el directorio origen, el destino y parámetros de compresión.
- `--restore`: permite restaurar desde un archivo comprimido existente.
- `--list-drive`: lista los respaldos almacenados en Drive.

Además, incorporamos retroalimentación visual del progreso para cada operación, lo cual mejora la experiencia de uso, especialmente en operaciones que pueden tomar varios minutos.

---

6. Justificación Técnica

Elegimos Python como lenguaje de desarrollo por su ecosistema maduro y la disponibilidad de bibliotecas para procesamiento paralelo, compresión, manejo de archivos y servicios en la nube.

La elección de Dask nos permitió escalar el sistema de forma eficiente sin complicar el código base. A diferencia de multiprocessing, Dask administra automáticamente la

distribución de tareas y permite ajustar la cantidad de trabajadores según la carga del sistema.

El uso de pyzipper fue clave para ofrecer compresión segura, mientras que argparse nos permitió estructurar una CLI robusta y amigable para el usuario.

Frente a otras opciones más complejas como interfaces gráficas o integración con gestores de archivos, priorizamos un enfoque sencillo, portable y fácil de mantener.

---

7. Cómo usar el sistema

Este sistema permite realizar copias de seguridad comprimidas en formato ZIP utilizando compresión paralela adaptativa. Las copias pueden guardarse localmente, en unidades USB o directamente en Google Drive. También permite restaurarlas desde cualquiera de estos medios.

7.1 Requisitos

Antes de ejecutar el sistema, asegúrese de contar con:

- Python 3.10 o superior.
- Las dependencias instaladas con:

pip install -r requirements.txt

- Si desea utilizar Google Drive, debe tener el archivo credentials.json previamente autenticado.

7.2 Realizar respaldo

A. A una ubicación local o USB

Para comprimir una carpeta y guardar el archivo de respaldo de forma local o en una USB:

python -m src.main backup /ruta/a/la/carpeta --output nombre_respaldo.zip --password tuClave123

- `/ruta/a/la/carpeta`: Ruta a la carpeta que desea respaldar.
- `--output`: (opcional) Nombre del archivo de respaldo. Por defecto se usa el formato `backup_YYYYMMDD_HHMMSS.zip`.
- `--password`: (opcional) Clave para encriptar el archivo con AES-256.

Luego de hacer el procesamiento del zip y almacenamiento local, aparecerá un menú que facilita el almacenamiento en dispositivos externos.

A.B Google Drive

Para guardar el respaldo directamente en una carpeta de Google Drive:

python -m src.main backup /ruta/a/la/carpeta --output nombre.zip --password tuClave123

Luego, cuando el programa se esté ejecutando aparecerá un menú dinámico donde se ingresa el campo de id de la carpeta donde se quiere almacenar, y tomando la información registrada en el *settings.yaml* con las credenciales de la cuenta, se procede a hacer el respaldo.

7.3 Restaurar respaldo

A. Desde archivo local o USB

Si el respaldo está en un archivo .zip local (incluyendo USB):

python -m src.main restore local --file-path /ruta/al/respaldo.zip --output-dir /ruta/destino --password tuClave123

- `--file-path`: Ruta al archivo `.zip` de respaldo.
- `--output-dir`: Carpeta donde se restaurarán los archivos.
- `--password`: (opcional) Clave usada para cifrar el respaldo, si aplica.

B. Desde Google Drive

Para restaurar un archivo almacenado en Drive:

python -m src.main restore drive --file-id FILE_ID --output-dir /ruta/destino --password tuClave123

- `--file-id`: ID del archivo en Drive que desea restaurar.
- `--output-dir`: Carpeta de destino donde se extraerán los archivos.

---

8. Conclusiones

El sistema que desarrollamos permite realizar respaldos rápidos, seguros y versátiles gracias al uso de procesamiento paralelo, compresión eficiente y múltiples destinos de almacenamiento. La inclusión de cifrado y soporte para dispositivos USB extiende sus casos de uso en ambientes que requieren seguridad y portabilidad.

El uso de Dask resultó fundamental para mejorar el rendimiento en cargas altas, validando su pertinencia para tareas de sistema operativo orientadas a rendimiento. Además, la modularidad del sistema facilita su extensión futura, por ejemplo, para incluir almacenamiento en otros servicios como Dropbox o S3.