Informe sobre la Creación de un Video a Partir de Imágenes Usando Ray en un Entorno Virtualizado

Juan Diego Collazos Mejia, Juan Sebastian Garizao

November 22, 2024

1 Introducción

Este informe describe el proceso realizado para la creación de un video a partir de un conjunto de imágenes descargadas de una página web. El trabajo fue realizado en un entorno virtualizado utilizando VirtualBox, con una distribución Kali Linux y empleando la librería Ray para paralelizar el procesamiento de las imágenes. Además, se utilizó el esquema maestro-esclavo para distribuir las tareas entre las máquinas virtuales.

2 Entorno de Trabajo

2.1 Configuración de la Red Interna en VirtualBox

Para permitir la comunicación entre las máquinas virtuales, se configuró una red interna en VirtualBox. Ambas máquinas virtuales deben estar conectadas a la misma red interna con IP estática. Por ejemplo:

- **Máquina Maestro:** IP 192.168.56.101
- **Máquina Esclavo:** IP 192.168.56.102

2.2 Inicialización de Ray en las Máquinas

Ray necesita ser iniciado en el nodo maestro y conectado en los nodos esclavos. A continuación, se muestran los comandos necesarios:

2.2.1 En el Nodo Maestro

Primero, inicia Ray en el nodo maestro con el siguiente comando:

```
ray start --head --node-ip-address=192.168.56.101 --port=6379
```

Este comando inicia el nodo maestro de Ray en la dirección IP y puerto especificados.

2.2.2 En los Nodos Esclavos

Para conectar los nodos esclavos al maestro, ejecuta este comando en cada nodo esclavo:

```
ray start --address='192.168.56.101:6379' --node-ip-address=192.168.56.102
```

Asegúrate de reemplazar `192.168.56.102` con la dirección IP correspondiente a cada nodo esclavo.

2.3 Ejecutando el Script en el Maestro

Una vez configurados los nodos, ejecuta el script Python en el nodo maestro. Ray se encargará de distribuir las tareas a los nodos esclavos automáticamente.

```
python main.py
```

El script se conectará al clúster de Ray y procesará las imágenes distribuyendo las tareas entre los nodos configurados.

2.4 Verificación del Estado del Clúster

Para verificar que los nodos están conectados correctamente al clúster, usa el siguiente comando en cualquiera de las máquinas:

```
ray status
```

Este comando muestra información sobre el estado del clúster, incluyendo los nodos activos y los recursos disponibles.

2.5 Configuración de VirtualBox

Las máquinas virtuales se configuraron en VirtualBox con las siguientes especificaciones:

- **Máquina Maestro:** Esta máquina es la encargada de coordinar el procesamiento y crear el video final.
- **Máquina Esclavo:** Se utiliza para procesar imágenes en paralelo mediante Ray.

Se configuró una red interna entre las máquinas virtuales para que pudieran comunicarse y usar Ray de manera efectiva.

2.6 Librerías y Recursos

Se utilizó la siguiente librería para realizar el procesamiento de imágenes y la creación del video:

- **Ray:** Ray es una librería de paralelización que permite distribuir tareas entre múltiples máquinas y núcleos.
- **PIL (Python Imaging Library):** Usada para la manipulación de imágenes (redimensionar, convertir a escala de grises).
- **OpenCV:** Para la creación de videos a partir de imágenes.
- **Requests y BeautifulSoup:** Para descargar imágenes desde una página web.

3 Proceso de Trabajo

El trabajo se dividió en dos scripts principales: uno para descargar las imágenes de una página web y otro para procesarlas y crear el video.

3.1 Descarga de Imágenes

El primer script descargó imágenes desde una página web utilizando las librerías `requests` y `BeautifulSoup`. Este script extrae todas las imágenes de la página y las guarda en una carpeta local para su posterior procesamiento.

```python
import os
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def download_image(image_url, save_folder):
    try:
        img_data = requests.get(image_url).content
        img_name = os.path.basename(image_url)
        img_path = os.path.join(save_folder, img_name)
        with open(img_path, 'wb') as file:
            file.write(img_data)
        print(f"Imagen guardada: {img_path}")
    except Exception as e:
        print(f"Error al descargar la imagen {image_url}: {e}")

def fetch_images_from_page(url, save_folder):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    img_tags = soup.find_all('img')
    if not os.path.exists(save_folder):
        os.makedirs(save_folder)
    for img_tag in img_tags:
        img_url = img_tag.get('src')
        if img_url:
            full_img_url = urljoin(url, img_url)
            if full_img_url.lower().endswith(('.jpg', '.jpeg', '.png')):
                download_image(full_img_url, save_folder)

if __name__ == "__main__":
    website_url = 'https://www.freeimages.com/search/dog'
    save_folder = 'input_images'
    fetch_images_from_page(website_url, save_folder)
```

Listing 1: Script para descargar imágenes

Este script utiliza `requests` para obtener el contenido HTML de la página y `BeautifulSoup` para extraer las URLs de las imágenes. Luego, descarga cada imagen en la carpeta `input_images`.

3.2 Procesamiento de Imágenes y Creación del Video

El segundo script se encargó de procesar las imágenes descargadas, convertirlas a escala de grises y redimensionarlas, si es necesario. Para esto, se utilizó Ray para paralelizar el procesamiento de las imágenes en los nodos esclavos.

```
import ray
from PIL import Image
import cv2
import os
import numpy as np

@ray.remote
def process_image(image_path, target_size=None):
    try:
        node_ip = ray._private.services.get_node_ip_address()
        print(f"Procesando {image_path} en nodo {node_ip}")
        image = Image.open(image_path)
        grayscale_image = image.convert("L")
        if target_size:
            grayscale_image = grayscale_image.resize(target_size)
        grayscale_cv2 = cv2.cvtColor(np.array(grayscale_image), cv2.COLOR_GRAY2BGR)
        return grayscale_cv2
    except Exception as e:
        raise ValueError(f"Error procesando la imagen {image_path}: {e}")

def create_video_from_images(image_paths, video_path, fps=1):
    if not image_paths:
        raise ValueError("No hay im genes para crear el video.")
    first_image = Image.open(image_paths[0])
    target_size = first_image.size
    image_refs = [process_image.remote(img_path, target_size) for img_path in image_paths]
    images = ray.get(image_refs)
    height, width = target_size[1], target_size[0]
    fourcc = cv2.VideoWriter_fourcc(*"XVID")
    video_writer = cv2.VideoWriter(video_path, fourcc, fps, (width, height))
```

```
    for idx, image in enumerate(images):
        try:
            video_writer.write(image)
            print(f"Imagen {idx + 1}/{len(images)} agregada al video.")
        except Exception as e:
            print(f"Error al agregar la imagen {idx + 1}: {e}")
            continue
    video_writer.release()
    print(f"Video creado exitosamente: {video_path}")

def get_image_paths_from_directory(directory, extensions=("jpg", "png", "jpeg")):
    return [os.path.join(directory, file) for file in os.listdir(directory) if file.lower().endswith(extensions)]

if __name__ == "__main__":
    ray.init(address='auto')  % Conectar con el nodo maestro autom ticamente
    try:
        image_directory = "input_images"
        output_video_path = "output_video.avi"
        image_paths = get_image_paths_from_directory(image_directory)
        image_paths.sort()
        create_video_from_images(image_paths, output_video_path, fps=0.5)
    finally:
        ray.shutdown()
```

Listing 2: Script para procesar imágenes y crear el video

En este script, el comando `ray.init(address='auto')` permite que las máquinas virtuales se conecten automáticamente al nodo maestro. El procesamiento de imágenes se distribuye entre las máquinas virtuales mediante Ray, y el video final se crea usando OpenCV.

4 Conclusiones

El uso de Ray permitió distribuir eficientemente el procesamiento de imágenes entre las máquinas virtuales configuradas en un esquema maestro-esclavo. La

paralelización de las tareas mejoró significativamente el rendimiento y permitió la creación de un video a partir de un gran número de imágenes en un tiempo razonable.

Además, se pudo aprovechar la capacidad de las máquinas virtuales para simular un entorno de procesamiento distribuido sin necesidad de hardware adicional. El proceso de trabajo se completó con éxito y el video final se generó correctamente.

5 Demostración del Funcionamiento

En esta sección se presenta un video demostrando el funcionamiento del sistema, que muestra cómo se descargan, procesan y generan los videos con las imágenes de manera distribuida utilizando Ray. El video está disponible en el siguiente enlace:

Video de demostración

6 Referencias

- Ray: A Unified Framework for Scaling Python
- Pillow (PIL Fork)
- OpenCV: Open Source Computer Vision Library
- BeautifulSoup: Web Scraping Library