PROYECTO

SISTEMAS DISTRIBUIDOS

PROFESOR: GUILLERMO MONROY RODRIGUEZ

ALUMNO: JAVIER ARTURO CIRILO VARGAS
MATRICULA: 2233030596

Alumno: Javier Arturo Cirilo Vargas Matricula: 2233030596

UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Cuajimalpa

PROBLEMÁTICA

Construir un sistema distribuido que permita intercambiar fragmentos de video entre nodos de forma P2P. Los fragmentos se distribuyen entre varios nodos, que pueden solicitarse mutuamente los archivos faltantes.

OBJETIVOS ESPECIFICOS

- Dividir un video en 10 fragmentos para facilitar su distribución y manejo en el sistema.
- Implementar nodos que permitan el intercambio directo de fragmentos de video entre sí, siguiendo un modelo peer-to-peer (P2P).
- Desarrollar un microservicio o API que gestione y registre qué fragmentos posee cada nodo en la red.
- Implementar un sistema de publicación/suscripción (Pub/Sub) para notificar a los nodos sobre la disponibilidad y actualización de fragmentos.
- Crear una interfaz básica, mediante consola o registros de logs, que permita solicitar y recibir fragmentos de video entre nodos.
- Utilizar contenedores Docker para facilitar el despliegue y la ejecución aislada de los nodos y servicios, y documentar la API mediante Swagger para una mejor gestión y pruebas.

Justificación

El modelo P2P (peer-to-peer) se ha consolidado como una arquitectura ampliamente empleada en sistemas de intercambio de archivos y plataformas de streaming distribuido, gracias a su eficiencia, escalabilidad y capacidad de resiliencia ante fallos. En el presente proyecto, se busca aplicar estos principios de manera práctica mediante la implementación de un sistema que permita el intercambio de fragmentos de video entre nodos autónomos. Esta dinámica simula el funcionamiento de redes colaborativas de transmisión multimedia, donde los datos son gestionados de forma descentralizada y eficiente.

Metodología de Desarrollo

Se utilizó un enfoque **incremental**, donde se fue construyendo cada modulo y probando cada uno a la ves para de esta forma después integrarlos:

1. Desarrollo del **video splitter** con este se generó el dividir el video en fragmentos a partir del video original, en este caso decidimos dividirlo en 10 partes.
2. Creación de APIs en **node1** y **node2** para que se gestionen y servir los fragmentos.
3. Implementación de la comunicación **TCP** entre nodos para el intercambio directo de los fragmentos del video.
4. Añadido del sistema **Pub/Sub** para notificaciones automáticas.
5. Configuración de **Dockerfiles** y **docker-compose.yml** para el despliegue de nuestros contenedores.
6. Documentación del API con **Swagger**.

ESTRUCTURA

```
Proyecto_Sistemas/
│
├── docker-compose.yml
├── node1/
│   ├── app.py
│   ├── pubsub.py
│   ├── Dockerfile
│   └── fragments/
├── node2/
│   ├── app.py
│   ├── pubsub.py
│   ├── Dockerfile
│   └── fragments/
├── video_splitter/
│   ├── splitter.py
│   └── fragments/
├── shared/
│   └── tcp_client.py
└── swagger/
    └── swagger.yaml
```

COMPONENTES

splitter.py Fragmentation del video

Ubicación: video_splitter/splitter.py

- **Que hace:** Toma un archivo de video original (.mp4) y lo divide en un número configurable que nosotros elegimos de fragmentos más pequeños para facilitar su transmisión y almacenamiento.
- **Tecnologías empleadas:**

  Python 3.11
  fragment (llamadas ffmpeg) sirve para cortar el video sin pérdida de calidad.
- **Flujo interno:**

1. Se define la ruta del video de entrada y el número de partes en que se desea dividir que fueron los fragmentos decididos.
2. El script llama a la funcionalidad de fragment/ffmpeg para procesar el archivo.
3. Se generan los archivos de salida fragment_0.mp4, fragment_1.mp4, etc., en la carpeta fragments/.

- **Interacción con otros módulos:** Los fragmentos generados son distribuidos a los nodos (node1 y node2) para su posterior intercambio.

app.py API REST

- **Ubicación:** node1/app.py y node2/app.py
- **Que hace:** Proporciona un servicio HTTP para **subir**, **descargar** y **listar** fragmentos disponibles en cada nodo.
- **Tecnologías empleadas:**

  1. **Flask** = Framework para implementar las rutas REST.
  2. **requests** = Para enviar fragmentos entre nodos cuando se reciben peticiones.
- **Rutas principales:**

  1. POST /upload_fragment = Recibe un archivo y lo guarda en fragments/.
  2. GET /get_fragment/<nombre> = Devuelve el fragmento solicitado.
  3. GET /list_fragments = Lista todos los fragmentos.

- **Interacción con otros módulos:**
    1. Invoca pubsub.py para notificar a otros nodos cuando llega un nuevo fragmento.
    2. Puede llamar a tcp_client.py para solicitar fragmentos que no posee.

pubsub.py

- **Ubicación:** node1/pubsub.py y node2/pubsub.py
- **Que hace:** Notificar a los demás nodos cuando un fragmento nuevo es añadido.
- **Tecnologías empleadas:**
    1. **Flask** para recibir notificaciones HTTP.
    2. **Sockets TCP** para enviar avisos en tiempo real.
- **Flujo interno:**

1. Un nodo publica un mensaje indicando que tiene un nuevo fragmento.
2. Los nodos suscritos reciben la notificación y pueden iniciar la descarga del archivo.

- **Interacción con otros módulos:** Está directamente conectado con app.py para integrarse en el momento en que un fragmento se sube.

tcp_client.py

- **Ubicación:** shared/tcp_client.py
- **Que hace:** Solicitar a otro nodo un fragmento específico utilizando el protocolo **TCP**.
- **Tecnologías empleadas:**
    1. **socket** para conexión punto a punto.
    2. **file streaming** lectura y escritura en bloque para transferir archivos grandes.

- **Flujo interno:**
    1. Se establece una conexión TCP con el puerto del nodo remoto.
    2. Se envía el nombre del fragmento requerido.
    3. Se recibe el archivo y se guarda en fragments/ local.
- **Interacción con otros módulos:** Se ejecuta desde app.py o manualmente para obtener archivos faltantes.

Dockerfile

- **Ubicación:** node1/Dockerfile y node2/Dockerfile
- **Que hace:** Define cómo construir la imagen de cada nodo para su despliegue en contenedores.
- **Contenido:**
    1. **Imagen base:** python:3.11-slim
    2. Instalación de dependencias (Flask, requests, moviepy, etc.).
    3. Copia del código fuente al contenedor.
    4. Definición del puerto expuesto.
- **Interacción con otros módulos:** Asegura que cada nodo tenga el mismo entorno de ejecución sin conflictos de versiones.

docker-compose.yml

- **Ubicación:** raíz del proyecto.
- **Que hace:** Levantar los dos nodos (node1 y node2) y establecer los puertos de comunicación.
- **Características principales:**
    1. Define puertos HTTP (8000/8001) y TCP (5001/5002).
    2. Monta volúmenes para persistir fragments/.
    3. Facilita la comunicación entre contenedores usando una red interna de Docker

Alumno: Javier Arturo Cirilo Vargas Matricula: 2233030596

Proceso de Pruebas

Pruebas de Fragmentación

Se utilizó un video de prueba (3770775687-preview.mp4) que fue dividido en 10 fragmentos.
Resultado: Archivos fragment_0.mp4 a fragment_9.mp4 generados correctamente en la carpeta de salida.

Pruebas de API

Se verificaron endpoints de subida y descarga de fragmentos con herramientas como **Postman** y **cURL**.
Resultado: Respuestas HTTP 200 y transferencia correcta de archivos.

Pruebas de Comunicación P2P

Con node1 y node2 ejecutándose en contenedores, se solicitó un fragmento inexistente en node1, pero presente en node2.
Resultado: Transferencia exitosa vía **TCP** y almacenamiento en node1/fragments/.

Pruebas de Pub/Sub

Se notificó a los nodos cuando un nuevo fragmento fue agregado a otro nodo.
Resultado: Mensaje de disponibilidad recibido en tiempo real.

Resultados

1. Video fragmentado exitosamente en 10 partes.
2. Intercambio de fragmentos entre nodos confirmado.
3. Notificaciones automáticas funcionando correctamente.
4. Contenedores desplegados y comunicándose entre sí.

Alumno: Javier Arturo Cirilo Vargas Matricula: 2233030596

UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Cuajimalpa

Evidencias

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

 ✓ node2                                  Built
                0.0s
 ✓ node1                                  Built
                                                  17.6s
 ✓ Container proyecto_sistemas-node1-1  Recreated
               18.5s
Attaching to node1-1, node2-1
node1-1  |  * Serving Flask app 'app'
node1-1  |  * Debug mode: off
node1-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node1-1  |  * Running on all addresses (0.0.0.0)
node1-1  |  * Running on http://127.0.0.1:8010
node1-1  |  * Running on http://172.19.0.2:8010
node1-1  | Press CTRL+C to quit
node2-1  | [TCP Server] Escuchando en puerto 5002
node2-1  |  * Serving Flask app 'app'
node2-1  |  * Debug mode: off
node2-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node2-1  |  * Running on all addresses (0.0.0.0)
node2-1  |  * Running on http://127.0.0.1:8011
node2-1  |  * Running on http://172.19.0.3:8011
node2-1  | Press CTRL+C to quit
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
```

```
node2-1  | Press CTRL+C to quit
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:56:52] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:58:05] "GET /fragments HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [08/Aug/2025 23:58:29] "GET /fragments HTTP/1.1" 200 -
node2-1  | 172.19.0.2 - - [08/Aug/2025 23:59:10] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.2 - - [08/Aug/2025 23:59:10] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.2 - - [08/Aug/2025 23:59:10] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.2 - - [08/Aug/2025 23:59:10] "POST /upload_fragment HTTP/1.1" 200 -
node2-1  | 172.19.0.2 - - [08/Aug/2025 23:59:11] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [08/Aug/2025 23:59:11] "GET /send_all_fragments HTTP/1.1" 200 -
node1-1  | 172.19.0.3 - - [09/Aug/2025 00:00:08] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.3 - - [09/Aug/2025 00:00:08] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.3 - - [09/Aug/2025 00:00:08] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.3 - - [09/Aug/2025 00:00:08] "POST /upload_fragment HTTP/1.1" 200 -
node1-1  | 172.19.0.3 - - [09/Aug/2025 00:00:08] "POST /upload_fragment HTTP/1.1" 200 -
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

  699 kb/s, 50 fps, 50 tbr, 12800 tbn (default)
      Metadata:
        handler_name    : VideoHandler
        vendor_id       : [0][0][0][0]
        encoder         : Lavc59.37.100 libx264
Press [q] to stop, [?] for help
[out#0/mp4 @ 000001e152f23440] video:471KiB audio:0KiB subtitle:0KiB other streams:0KiB global headers:0KiB muxing overhead: 0.8755
22%
frame=  277 fps=0.0 q=-1.0 Lsize=     475KiB time=00:00:01.01 bitrate=3835.2kbits/s speed=  52x
Guardado fragmento 9: fragments\fragment_9.mp4
[✓] Enviado correctamente → fragment_0.mp4
[✓] Enviado correctamente → fragment_1.mp4
[✓] Enviado correctamente → fragment_2.mp4
[✓] Enviado correctamente → fragment_3.mp4
[✓] Enviado correctamente → fragment_4.mp4
[✓] Enviado correctamente → fragment_5.mp4
[✓] Enviado correctamente → fragment_6.mp4
[✓] Enviado correctamente → fragment_7.mp4
[✓] Enviado correctamente → fragment_8.mp4
[✓] Enviado correctamente → fragment_9.mp4
PS C:\Users\Javier\Documents\Proyecto_Sistemas\video_splitter> curl http://localhost:8010/fragments


StatusCode        : 200
```

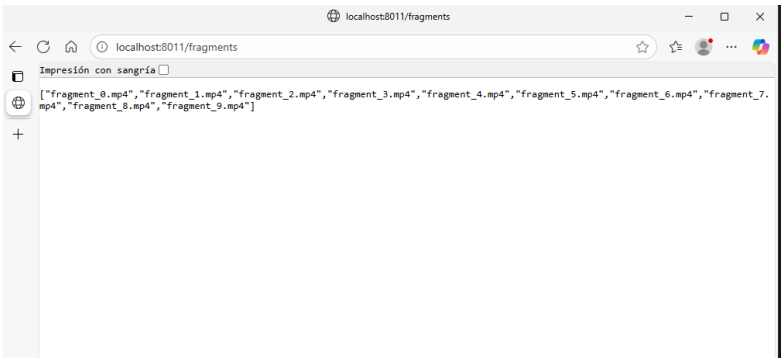
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\Javier\Documents\Proyecto_Sistemas> curl http://localhost:8010/send_all_fragments


StatusCode        : 200
StatusDescription : OK
Content           : {"enviados":["fragment_0.mp4","fragment_1.mp4","fragment_2.mp4","fragment_3.mp4","fragment_4.mp4"],"fallidos":
                    []}

RawContent        : HTTP/1.1 200 OK
                    Connection: close
                    Content-Length: 114
                    Content-Type: application/json
                    Date: Fri, 08 Aug 2025 23:59:11 GMT
                    Server: Werkzeug/3.1.3 Python/3.10.18

                    {"enviados":["fragment_0.mp4","fr...
Forms             : {}
Headers           : {[Connection, close], [Content-Length, 114], [Content-Type, application/json], [Date, Fri, 08 Aug 2025
                    23:59:11 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 114
```



CONCLUSION

En este proyecto se cumplieron los objetivos planteados, logrando la implementación de un sistema P2P funcional para el intercambio de fragmentos de video. Por otro lado, el uso de Docker facilitó tanto el despliegue como la replicación del entorno, mientras que la integración de Swagger permitió una documentación clara y la realización de pruebas eficientes sobre la API.

Asimismo, se comprobó que el uso del patrón Pub/Sub mejora significativamente la eficiencia del sistema al permitir la notificación en tiempo real sobre la disponibilidad de los recursos.

Nos esforzamos por desarrollar un proyecto lo más completo posible en relación con los requerimientos establecidos, avanzando paso a paso para garantizar su correcto funcionamiento, libre de errores y con un enfoque en la calidad y estabilidad. Consideramos que el resultado final cumple con los estándares esperados y confiamos en que sea de su agrado.

Posibles Mejoras

- Implementar autenticación y cifrado en la transmisión.
- Mejorar la interfaz para monitorear el intercambio de fragmentos.
- Añadir balanceo de carga en redes con múltiples nodos.
- Implementar recuperación de fragmentos desde múltiples fuentes simultáneamente.