Escuela Superior de Informática

Sistemas Distribuidos

Youtube Downloader

Miembros:
Juan Perea Campos
Rodrigo Díaz-Hellín Valera

Repositorio:
https://github.com/jupcan/youtube-dl

January 28, 2019

Contents

1 Youtube Downloader

1.1 Objetivo Principal

El objetivo principal del proyecto es diseñar un sistema cliente-servidor que permita la extracción de ficheros de audio a partir de la URL de clips de youtube. Este sistema debe ser escalable, permitiendo la creación bajo demanda de nuevos servidores encargados de las tareas de descarga y extracción de los audios. Además, estos servidores deben interconectarse y sincronizarse de forma automática entre sí, proporcionando de esta manera un sistema de alta disponibilidad[2]

1.2 Arquitectura del Proyecto

El proyecto se compone de tres tipos de componentes: **servidores**, **clientes** y **canales de eventos**, y cuya arquitectura se muestra en la Figura 1.

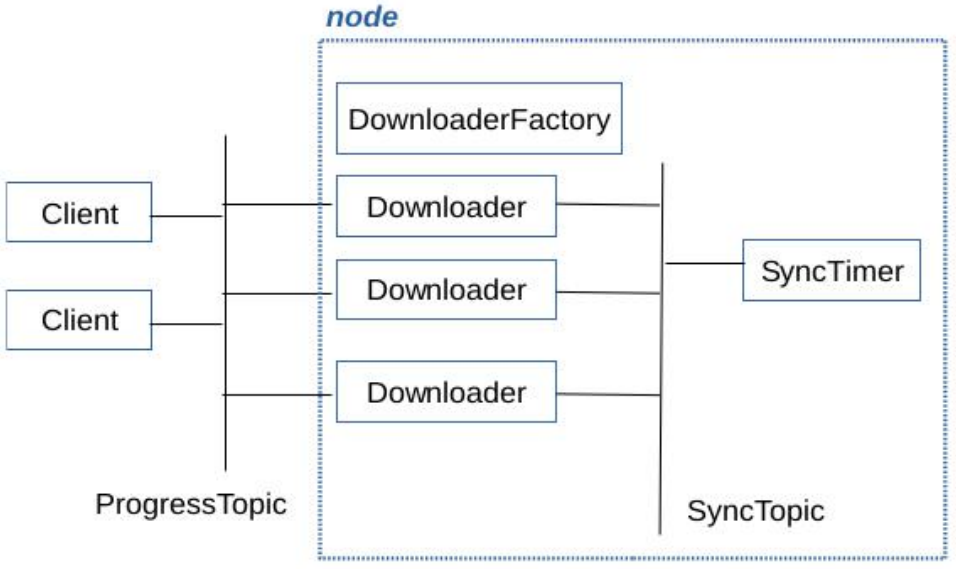

Figure 1: Diagrama del sistema a implementar

La gestión de todo el servicio ha sido realizada a través de *IceGrid*[3].

1.2.1 Servidores - factory.py

El despliegue de los servidores se ha realizado de manera **dinámica** utilizando para ello una factoría *SchedulerFactory*. Esta factoría los crea y elimina a través de los métodos *make(name, current=None)* y *kill(name, current=None)* respectivamente.

Implementan la interfaz *DownloadScheduler* y realizan las siguientes tareas:

- Recepción de las **peticiones de descarga** de los clientes sin retornar valores y que comunican la información de la descarga a través de un canal de eventos.
- **Almacenamiento** de los archivos obtenidos tras la extracción y descarga del audio de la url deseada en un directorio local al servidor.
- **Transferencia** al cliente de dichos archivos en formato *.mp3*.
- Obtención de un **listado** de los archivos almacenados en cada servidor.

1.2.2 Clientes - client.py

Capaces de hacer uso de los servidores del sistema y realizar las siguientes tareas correspondientes a los métodos de la interfaz *downloader.ice* facilitada:

- Crear nuevo servidor de descargas a través de la factoría.
- Eliminar servidor de descargas.
- Ver listado de archivos descargados en un servidor específico.
- Solicitar descargas[1] de una URL de YouTube.
- Obtención del archivo de audio deseado desde el servidor.
- Ver listado de servidores actualmente desplegados.
- Salir de la aplicación.

1.2.3 Canales de Eventos - synctimer.py

Como mencionamos previamente, la comunicación entre cliente y servidores de descarga se realiza a través de un canal de eventos *ProgressTopic* que tiene cuatro tipos de llamadas del tipo notify, correspondiéndose con los cuatro estados en los que puede hallarse una descarga: **Pending**, **InProgress**, **Done** o **Error**.

[1] Al hacerlo, éste recibe notificaciones del estado de la misma mediante *ProgressTopic*

A su vez están también sincronizados entre ellos mediante otro canal de eventos *SyncTopic*, del que los servidores son subscriptores; el publicador es un componente especial -el código del mismo se encuentra en el binario **synctimer.py**- con un temporizador que realiza las solicitudes para sincronizar objectos *DownloadScheduler*.

1.3 Extensiones Opcionales Implementadas

Hemos incluido la eliminación de servidores siempre que éstos no estén en uso y extendido la factoría para mostrar el número total de servidores desplegados, ambas son opciones listadas en el menú del cliente como veíamos en la sección 1.2.2.

1.4 Manual de Usuario

Para la ejecucción del sistema, los pasos a seguir son los siguientes:

1. Asegurarnos de que no hay ningún proceso de **ZeroC Ice** en ejecucción.
2. Ejecutar el bash script **makefile.sh** facilitado, que copiará los binarios del proyecto al directorio temporal del sistema y pondrá en ejecucción el node1.
3. Entrar en la carpeta src y lanzar **icegridgui** configurando una nueva conexión en localhost tal como se muestra en la Figura 2, se deberá visualizar el node1 del paso anterior, de lo contrario la conexión no se habrá realizado bien.

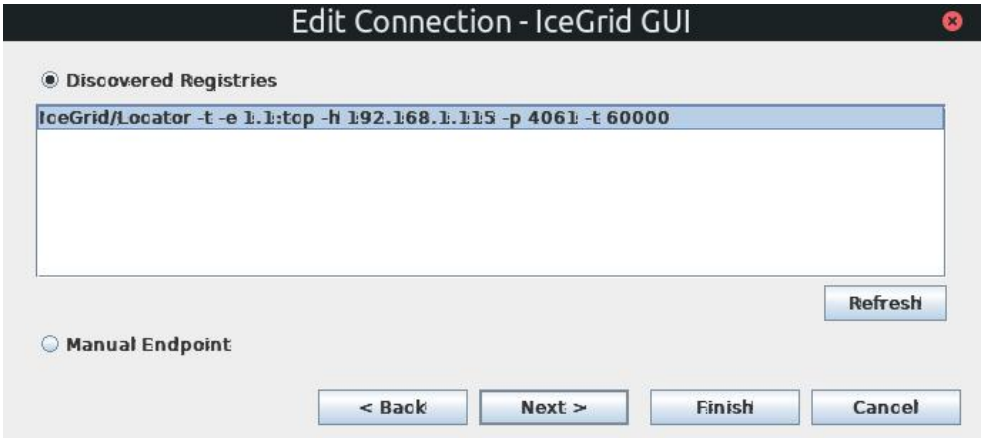

Figure 2: Conexión en localhost detectando endpoints

4. Una vez dentro, pulsaremos "abrir applicación desde archivo" y seleccionaremos **practica.xml** para posteriormente cargarla al registry y desplegarla a los servidores con *Tools - Application - Patch Distribution*.

5. Si todos los pasos se han realizado correctamente, podremos ver el servidor **factoría** y el de **sincronización** junto con los necesarios para el funcionamiento *IcePatch2* y *IceStorm*. Pulsamos botón derecho sobre **Factoria**, lo lanzamos y posteriormente hacemos click en "retrieve stdout" para visualizar el log del mismo, se nos abrirá una nueva pestaña mostrándolo en la que estará un string que es el proxy necesario para que el cliente pueda conectarse a él.

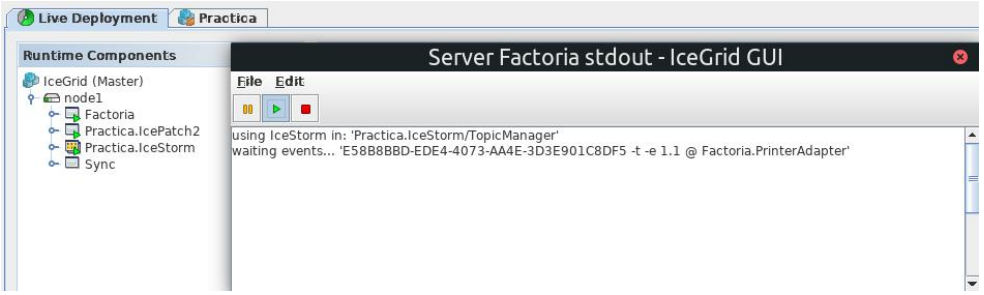

Figure 3: App desplegada en IceGrid con ejemplo de proxy

6. Para que la lista de canciones se sincronice entre servidores deberemos también levantar el servidor **Sync** pulsando también botón derecho y a continuación "Start". El stdout del mismo muestra las sync request de los servidores.

7. Dentro de la carpeta src, ejecutamos el cliente con su respectivo archivo de configuración y pasándole el proxy de la factoría que hemos obtenido en el paso anterior: **./client –Ice.Config=locator.config str_proxy**[2].

8. Se mostrará el menú del cliente para interactuar con todas las tareas disponibles mencionadas en la sección 1.2.2, siendo todas ellas funcionales y pidiendo la primera vez, el nombre para la creación de un servidor de descargas.

```
introduzca el nombre de un servidor: server
1.crear nuevo servidor
2.eliminar servidor
3.ver lista de canciones
4.descargar cancion
5.obtener cancion
6.servidores desplegados
7.salir de la aplicacion
introduzca una opcion: 4
```

[2]En el ejemplo mostrado en la Figura 3, el proxy para el cliente sería la cadena *'E58B8BBD-EDE4-4073-AA4E-3D3E901C8DF5 -t -e 1.1 @ Factoria.PrinterAdapter'* al completo.

1.5 Ejemplo de Ejecución

Tras haber puesto el marcha el sistema como hemos visto en la sección 1.4, un ejemplo de ejecucción del mismo podría ser el que mostramos a continuación.

```
1.crear nuevo servidor
2.eliminar servidor
3.ver lista de canciones
4.descargar cancion
5.obtener cancion
6.servidores desplegados
7.salir de la aplicacion
introduzca una opcion: 1
introduzca el nombre del servidor: s1
introduzca una opcion: 4
servidor al que realizar la solicitud: s1
introduce la url: https://www.youtube.com/watch?v=iX-QaNzd-0Y
descargando...
[descargada]
introduzca una opcion: 3
servidor al que realizar la solicitud: s1
obteniendo lista de canciones...
[obtenida]
./Milky Chance - Stolen Dance (Album Version).mp3
introduzca una opcion: 1
introduzca el nombre del servidor: s2
introduzca una opcion: 3
servidor al que realizar la solicitud: s2
obteniendo lista de canciones...
[obtenida]
./Milky Chance - Stolen Dance (Album Version).mp3
introduzca una opcion: 5
servidor al que realizar la solicitud: s1
introduce el nombre: ./Milky Chance - Stolen Dance (Album Version).mp3
transfiriendo...
[transferida]
nuevo nombre de la cancion: milky
introduzca una opcion: 6
2 servidor/es: ['s1', 's2']
introduzca una opcion: 7
finalizando ejecucion
```

References

[1] UCLM. Sistemas distribuidos. Moodle, 2018.

[2] Profesores Sistemas Distribuidos UCLM. proyecto. URL, 2018.

[3] ZeroC. Icegrid documentation. Website, 2018.

[4] ZeroC. Zeroc ice documentation. Website, 2018.