System rozproszonego przetwarzania plików wideo

Autorzy: Artur Bauer, Jakub Gajda, Łukasz Kożuszek

github

Założenia projektowe

- Podstawowym celem projektu było opracowanie i implementacja programowa architektury rozproszonej, mającej za zadanie interakcję z użytkownikiem oraz wykorzystanie zasobów obliczeniowych rozproszonych węzłów, koniecznych do konwersji wideo.
- Implementacja korzysta z biblioteki *ffmpeg* oraz *ffprobe,* użytkownik ma możliwość konwersji pliku wejściowego do wybranych standardów oraz rozdzielczości.
- Po dokonaniu konwersji w sposób rozproszony, system powiadamia użytkownika o statusie operacji oraz umieszcza gotowy plik w lokalizacji docelowej wskazanej przez użytkownika systemu.

Przypadki użycia

Projektowanie systemu rozpoczęto od zdefiniowania przypadków użycia zarówno z punktu widzenia systemu zarządzającego konwersją, jak również użytkownika dokonującego konwersji pliku.

1. **Przypadek użycia I**

   Aktor główny: Użytkownik
   Aktor pomocniczy: Klasa obsługująca linię poleceń [UserCLI]
   Opis: Użytkownik przesyła do systemu wybrany plik, wybiera docelowy format i rozdzielczość, system poprzez odpowiedni status i komunikat zawiadamia o wyniku operacji [FileData].
   Wyzwalacz: Użytkownik rozpoczyna przesyłanie pliku do systemu poprzez wpisanie odpowiedniego polecenia wraz z wymaganymi opcjami.
   Warunki początkowe:
   - Użytkownik ma połączenie z serwerem nadzorczym [Server]
   - Użytkownik wybiera plik wejściowy o właściwym rozszerzeniu [fileExtension]
   - Użytkownik wybiera właściwe opcje konwersji [resolution]

   Warunki końcowe:
   - System jest w stanie przetworzyć żądanie konwersji

   Przepływ normalny:
   - Użytkownik w linii poleceń uruchamia program
   - Użytkownik wybiera wejściowy plik do konwersji [location]
   - Użytkownik wybiera opcje konwersji [fileExtension] [resolution]
   - System przetwarza żądanie [convertFile()]
   - System zwraca odpowiedź dot. statusu operacji załadowania pliku

   Wyjątki:

   - Plik wybrany przez użytkownika ma nieobsługiwany format lub za duży rozmiar
     - System informuje użytkownika, że format wybranego pliku jest nieobsługiwany lub że ma za duży rozmiar (komunikat błędu

pokazuje dostępne formaty i maksymalny rozmiar pliku) i przerywa procedurę
        - Użytkownik może wybrać inny plik
    - Użytkownik wpisuje zbyt duże albo zbyt małe wartości docelowe
        - System informuje użytkownika, że rozdzielczość docelowa [resolution] jest nieprawidłowa (komunikat błędu pokazuje przedziały, w jakich musi się zawrzeć rozdzielczość)
        - Użytkownik może wpisać inne wartości rozdzielczości
- Użytkownik przerywa pracę z systemem w czasie przetwarzania
    - Następuje czyszczenie zawartości programu. Program kończy swoje działanie
- Błąd systemu
    - System informuje użytkownika o błędzie i prosi o ponowienie próby za kilka minut

2. **Przypadek użycia II**

Aktor główny: Serwer

Aktor(aktorzy) pomocniczy: Worker

Opis: Serwer po otrzymaniu pliku od użytkownika i załadowaniu do pamięci, dzieli plik na części po czym wysyła poszczególne częsci do rozproszonych węzłów, które są odpowiedzialne za przetworzenie pliku do zadanego formatu i rozdzielczości.

Wyzwalacz: Serwer ładuje nowy plik do pamięci**.**

Warunki początkowe:

- Serwer otrzymał właściwy plik [FileData].
- Serwer posiada dostępne zdalne węzły [Worker].
- Co najmniej jeden z węzłów jest w stanie przyjąć plik do przetworzenia [getFreeQueueSize()].

Warunki końcowe:

- Wszystkie części pliku [ConvFile] zostają wysłane do przetworzenia

Przepływ normalny:

- Serwer sprawdza liczbę dostępnych zasobów (węzłów) [checkWorkers()] oraz ich pojemności dot. dostępnego miejsca w kolejkach [getFreeQueueSize()].
- Na podstawie 1. ustalana jest liczba części, na które zostaje podzielony plik wejściowy [splitFile()]
- Przy pomocy wybranego protokołu przesyłany jest plik wraz z danymi do konwersji [sendSplitedFiles()] w postaci zdalnego wywołania procedury.
- Wszystkie pliki zostają wysłane.
- Serwer przechodzi w stan nasłuchiwania na nadchodzące przetworzone fragmenty [receiveSplitedFiles()].

Wyjątki:

- Plik jest zbyt duży aby został obsłużony
    - System informuje użytkownika, że nie jest w stanie konwertować tak dużych plików [showError()].
    - Użytkownik może wybrać inny plik [FileData].
- Plik jest relatywnie mały, koszt jego przetworzenia lokalnie jest mniejszy niż całkowity nakład pracy na dzielenie, przetworzenie i składanie
    - Serwer dokonuje konwersji lokalnie [convertFile()]
    - Plik jest zapisywany w lokalizacji wskazanej przez użytkownika [saveLocation].
- Użytkownik przerywa pracę z systemem w czasie przetwarzania
    - Następuje czyszczenie zawartości programu [clearData()]. Program kończy swoje działanie.
- Błąd systemu wynikający z nieosiągalności węzłów
    - System informuje użytkownika o błędzie w zdalnym połączeniu i prosi o ponowienie próby za kilka minut.

3. **Przypadek użycia III**

Aktor główny: Serwer

Aktor pomocniczy: Worker

Opis: Serwer po otrzymaniu komunikatu o gotowości Workera do wysłania pliku po konwersji przyjmuje pliki [convFile] i scala je w jedną całość [mergeSplitedFiles()]

Wyzwalacz: Komunikat od poszczególnych węzłów rozproszonych

Warunki początkowe:

- Serwer ma połączenie ze zdalnymi węzłami [Worker].
- Każdy z węzłów [workerList] zakończył konwersję przydzielonej mu części pliku.

Warunki końcowe:

- System jest w stanie scalić gotowy plik po konwersji [mergeSplitedFiles()]

Przepływ normalny:

- Serwer w trybie nasłuchiwania oczekuje na komunikaty od zdalnych węzłów [checkWorkers()].
- Po otrzymaniu komunikatu przyjmuje porcję danych [convFile] od każdego z węzłów.
- Dane przechowywane są aż do otrzymania wszystich części [convertedFiles].
- System scala wszystkie otrzymane pliki [mergeSplitedFiles()].
- System zwraca odpowiedź dot. statusu operacji do użytkownika

Wyjątki:

- Nie ma połączenia z którymś z węzłów

        - System informuje użytkownika [showError()], że ze względu na błąd połaczenia sieciowego nie jest w stanie przeprowadzić konwersji
        - Po upłynięciu ustalonego timeout, w którym sprawdzana jest możliwość połączenia, program kończy działanie
    - Użytkownik przerywa pracę z systemem w czasie przetwarzania
        - Następuje czyszczenie zawartości programu [clearData()]. Program kończy swoje działanie.
    - Błąd systemu
        - System informuje użytkownika o błędzie [showError()] i prosi o ponowienie próby za kilka minut.

4. **Przypadek użycia IV**

    Aktor główny: Worker

    Aktor pomocniczy: Serwer

    Opis: Worker otrzymuje nowy plik [convFile] na wewnętrzną kolejkę przetwarzania [conversionFiles]. Zapamiętane są adresy źródłowe [serverData] oraz komenda [conversionData], którą należy wykonać, aby skonwertować plik.

    Wyzwalacz: Otrzymanie fragmentu pliku od serwera centralnego [receiveFile()]

    Warunki początkowe:
    - Węzeł ma połączenie z serwerem [connectToServer()].
    - Węzeł ma wolne miejsce w kolejce plików do konwersji [getFreeQueueSize()].

    Warunki końcowe
    - Węzeł odsyła plik po konwersji do centralnego serwera [sendFile()]

    Przepływ normalny
    - Węzeł otrzymuje komunikat z procedurą, którą należy wykonać do konwersji pliku [conversionData]
    - Węzeł otrzymuje część pliku przeznaczoną do konwersji [ConvFile]
    - Węzeł wykonuje konwersję pliku przy pomocy gotowej metody [convertFile()]
    - Węzeł wysyła komunikat o gotowości pliku do przesłania
    - Węzeł wysyła plik przy pomocy protokołu [sendFile()]

    Wyjątki
    - Nie ma połączenia z serwerem
        - System informuje użytkownika, że ze względu na błąd połączenia sieciowego nie jest w stanie przeprowadzić konwersji [sendError()]
        - Po upłynięciu ustalonego timeout, w którym sprawdzana jest możliwość połączenia, program kończy działanie

- Użytkownik przerywa pracę z systemem w czasie przetwarzania
    - Następuje czyszczenie zawartości programu [clearData()]. Program kończy swoje działanie.
- Błąd systemu
    - System informuje użytkownika o błędzie i prosi o ponowienie próby za kilka minut.[sendError()]

Aplikacja

Aplikacja została napisana w Pythonie, przy użyciu m.in. bibliotek socket, threading, subprocess, PythonVideoConverter i queue. Aplikacja wykorzystuje metody tych bibliotek do:

- Socket – zapewnia komunikację między Workerami oraz Serverem.
- Threading – tworzy i obsługuje osobne wątki do komunikacji z każdym Workerem. W Workerze z kolei tworzone są wątki do nasłuchiwania na zapytania od strony Servera oraz do konwersji plików.
- PythonVideoConverter – nakładka do ffmpeg ułatwiająca jego wykorzystanie przez skrypt Pythona.
- Queue – synchroniczne kolejki plików do przetworzenia oraz tych już przetworzonych, zarówno po stronie Workera, jak i Servera.
- subprocess - pozwala na wywołanie procedur konwersji po stronie zdalnych węzłów oraz scalania gotowych plików po stronie serwera poprzez wykonanie gotowch procedur (wykorzystuje ffmpeg i ffprobe)

Węzły systemu są reprezentowane przez procesy, identyfikowane przy pomocy numerów PID, komunikują się za pośrednictwem socketów. Dane o plikach przeznaczonych do konwersji oraz tych skonwertowanych są przechowywane w kolejkach FIFO, natomiast dzielenie, konwersja i łączenie plików jest wykonywana przy pomocy frameworku FFMPEG, do którego API zapewnia biblioteka PythonVideoConverter

Komunikacja

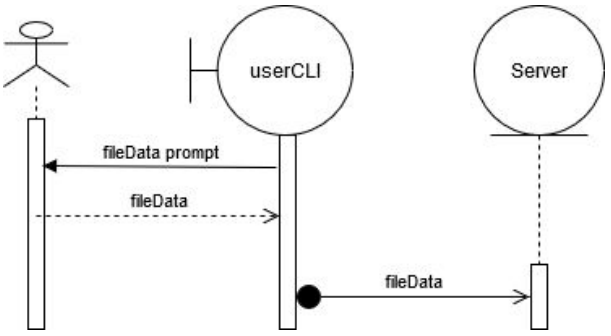
- Po uruchomieniu Workerzy zgłaszają się do Serwera (wiadomość JOIN)
- Serwer zapisuje dane każdego nowego Workera i przesyła do niego potwierdzenie zaakceptowania (wiadomość JOIN ACCEPT)
- Co 5 sekund Serwer odpytuje Workerów o liczbę plików, jakie są w danej chwili zdolne przetworzyć (wiadomość FREE SPACE REQUEST)
- Workerzy odpowiadają wiadomością FREE SPACE ANSWER
- W razie otrzymania nowego pliku od użytkownika Serwer dzieli go na tyle części, ile wynosi suma wartości zadeklarowanych przez Workerów w wiadomościach FREE

SPACE ANSWER, jednak co najwyżej na tyle, aby długość każdej części nie wynosiła mniej niż 10 sekund

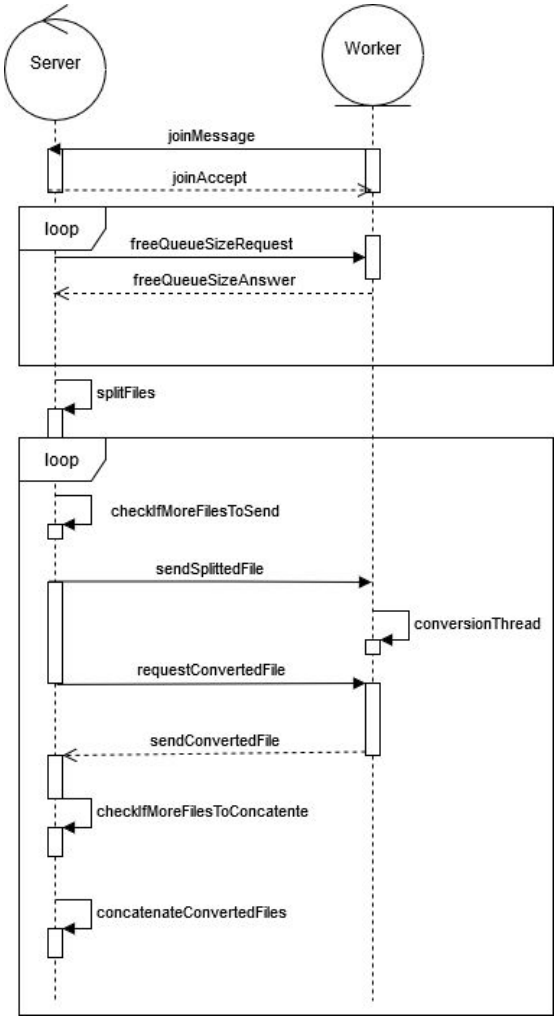
- Po podzieleniu pliku na części Serwer przesyła je do Workerów (wiadomość CONVERT FILE)
- Serwer co 5 sekund odpytuje Workerów, czy dokonały konwersji swoich plików (wiadomość SEND FILE)
- Jeżeli Worker ukończył konwersję, to odpowiada na tę wiadomość przesłaniem pliku do Serwera (wiadomość CONVERTED)

Diagramy sekwencji

1. Użytkownik wprowadza plik do konwersji:



2. System dokonuje konwersji przy pomocy zdalnych węzłów:



Instrukcja użytkownika

W celu używania programu należy zapewnić niezbędne oprogramowanie do działania programu:

- Python 3.x
- Pakiety Pythona

```
pip install -r requirements.txt
```

- Ffmpeg
- Export ścieżki ffmpeg do zmiennej środowiskowej

```
export ffmpeg = “/usr/bin/ffmpeg;/usr/bin/ffprobe”
```

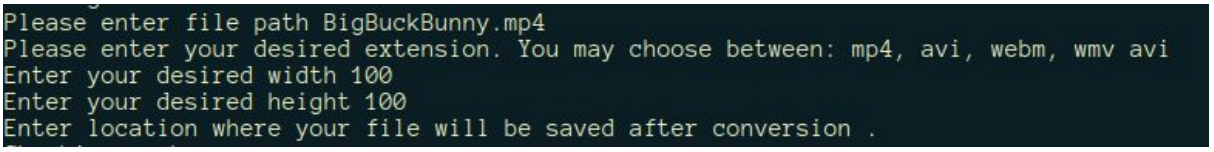
Po spełnieniu wymagań podstawowych należy uruchomić serwer:

```
python Server.py
```

Następnie uruchomić Workery, w zależności od potrzeb (min. 1).

```
python Worker.py
```

Należy podać nazwę pliku, rozszerzenie, rozdzielczość oraz lokalizację zapisu w Serwerze:



Po konwersji plik zostanie zapisany w wybranej przez użytkownika lokalizacji (w przykładzie w aktualnym katalogu) pod nazwą output z wybranym rozszerzeniem.

Podczas konwersji pliku możliwe jest dodanie dodatkowych Workerów, po nawiązaniu połączenia zostanie im przydzielony fragment pliku do konwersji

```
python Worker.py
```

Wnioski

Przy wykorzystaniu systemu rozproszonego dodatkowym problemem byłoby przesyłanie plików między serwerem oraz workerami, z uwagi na lokalne testy niezbędne było wysyłanie tylko ścieżek do plików.

Podczas pracy na różnych systemach trzeba było uwzględnić różne konwencje systemów plików, aby właściwie przechować fragmenty filmu. Wykonany program posiada cechy przenośności, jego prawidłowe działanie zostało przetestowane na platformach Windows oraz Linux.

W czasie realizacji programowej całej architektury napotkano również typowe problemy, które należy rozwiązać podczas implementacji typowego systemu rozproszonego. Problem jednoczesnego dostępu do sekcji krytycznej został rozwiązany poprzez wykorzystanie klasy Queue, posiadającej wbudowane mechanizmy gwarantujące wzajemne wykluczenie. Utrzymanie komunikacji w sposób ‘jeden do wielu’ zostało wykonane przy użyciu specjalizowanych wątków, których zadaniem była komunikacja z węzłami zdalnymi oraz wywołanie koniecznych akcji po stronie serwera. Dzięki wykorzystaniu biblioteki Socket architektura może pracować na różnych platformach w rzeczywisty rozproszony sposób (zmiany wymagał by wtedy sposób przesyłania plików pomiędzy węzłami).

Wykorzystanie zewnętrznego oprogramowania (ffmpeg) pozwoliło na ograniczenie pracy z samym przetwarzaniem plików oraz ich fragmentacją i pozwoliło skupić się w projekcie na pracy systemu rozproszonego w praktyce.

Wykorzystanie wątków w Workerach pozwoliło na jednoczesną komunikację z Serwerem nadzorującym oraz przetwarzanie plików.