

Создание программного модуля для рекурсивного перекодирования видеофайлов в каталоге с помощью Python и FFmpeg

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В работе описывается реализация программы конвертации видео файлов из одного формата в другой. Процесс исполняет сканирование дочерних директорий на распознавание предварительно указанного расширения файла. Исходный код написан на языке Python с подключённым модулем ffmpeg-python. Результатом работы станет программа рекурсивного копирования дерева каталогов с изменением формата видео файлов.

Ключевые слова: FFmpeg, Python, кодирование видеоданных, сжатие видео, видеокодек.

Creating a software module for recursively transcoding video files in a directory using Python and FFmpeg

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The paper describes the implementation of a program for converting video files from one format to another. The process scans the child directories to recognize the previously specified file extension. The source code is written in Python with ffmpeg-python plug-in. The result of the work will be a program for recursively copying a directory tree with changing the format of video files.

Keywords: FFmpeg, Python, video encoding, video compression, video codec.

1 Введение

1.1 Актуальность

В современных условиях стремительного роста объёмов цифрового видеоконтента задача эффективного управления пространством для хранения данных приобретает критическую важность. Существующие файловые хранилища, особенно в контексте долгосрочного архивирования или работы с большими массивами видеозаписей, часто сталкиваются с ограничениями по ёмкости. В то же время значительная часть видео сохраняется в устаревших или неоптимальных с точки зрения сжатия форматах, таких как H.264/AVC, что приводит к нерациональному использованию дискового пространства. Переход к более современным стандартам кодирования, в частности к

H.265/HEVC, позволяет существенно сократить размер файлов при сопоставимом уровне визуального качества. Однако ручное применение средств конвертации к большому количеству файлов, распределённых по сложной иерархии каталогов, представляет собой трудоёмкий и подверженный ошибкам процесс, требующий значительных временных затрат. Отсутствие автоматизированного инструмента для пакетной обработки, сохраняющего исходную структуру директорий и предоставляющего детализированную аналитику по результатам конвертации, является существенным практическим ограничением.

1.2 Обзор исследований

Ю.П.Крючкова, Ф.М.Игнатов, И.В.Власюк провели анализ искажений изображений в современных видеокодеках [1]. И.А.Зубакин, В.В.Козлов, А.К.Цыцулин описали методику тестирования видеокодеков с учетом меры нестационарности изображений [2]. К.А.Невмержицкий исследовал отличительные особенности популярных аудио, видеокодеков и контейнеров [3]. В.Д.Попов, А.Е.Паршин разработали алгоритм поиска оптимальных по Парето параметров кодирования видеокодеков на основе объективных метрик качества видео [4]. К.О.Рагулина, В.Д.Попов, А.Е.Паршин сравнили алгоритмы анализа значений параметров кодирования видеокодеков [5].

1.3 Цель исследования

Цель исследования – описать реализацию программного модуля для рекурсивного перекодирования видеофайлов в каталоге с помощью языка программирования Python и набора инструментов FFmpeg.

2 Материалы и методы

В реализации используется алгоритм пакетного сжатия видеофайлов формата MP4 с использованием библиотеки FFmpeg, сопровождаемый генерацией отчета о результатах обработки в формате HTML. Алгоритм процесса строится на последовательном обходе всех файлов в указанной исходной директории и подкаталогах, при этом каждый найденный видеофайл соответствующий формату подвергается анализу и перекодированию.

3 Результаты и обсуждения

Пакетный модуль ffmpeg-python является совокупностью методов обёртки над низкоуровневыми операциями взаимодействия языка python и консольного набора инструментов ffmpeg. Сам по себе не содержит никаких функций для работы с медиа. В связи, с чем необходимо установить программный комплекс ffmpeg отдельно на компьютер и удостовериться, что его можно вызвать из любой директории. В случае с операционной системой Windows, нужно добавить путь к каталогу с бинарными файлами в переменную окружения path.

Далее в исходном коде python, можно вызвать методы для видео файла и измерить его длительность путём извлечения метаданных через метод probe.

Функции, представленные на рисунке 1, позволяют рассчитывать относительные метрики производительности. Фиксируется исходный размер файла в байтах.

```
def get_video_info(file_path):
    """Получает длительность видео в секундах и его размер."""
    try:
        probe = ffmpeg.probe(file_path)
        video_stream = next(s for s in probe["streams"] if s["codec_type"] == "video")
        duration = float(video_stream["duration"])
        return duration
    except Exception as e:
        print(f" Ошибка получения информации о {file_path}: {e}")
        return None

def get_file_size(file_path):
    """Получает размер файла в байтах."""
    return os.path.getsize(file_path)
```

Рис. 1. Извлечение информации о видео файле

Программная реализация формирует структуру выходной директории, зеркально повторяющую исходную, что обеспечивает сохранение организационной иерархии файлов.

В процессе обработки для каждого файла замеряется время, затраченное на сжатие, после чего вычисляется размер полученного файла. На основе этих данных определяются абсолютное время обработки, среднее время сжатия одной минуты видео, процент уменьшения размера файла и отношение размера файла с новым расширением к исходному файлу. Информация накапливается в памяти для последующего агрегирования (рис.2).

```
def compress_and_copy(src_folder, dest_folder, crf=28, preset="slow", audio_bitrate="128k"):
    report_data = [] # Данные для отчёта
    total_time = 0 # Общее время сжатия всех файлов
    total_video_minutes = 0 # Общее время всех видео в минутах
    total_compression_percentage = 0 # Общий процент сжатия для вычисления среднего

    for root, _, files in os.walk(src_folder):
        for file in files:
            if file.lower().endswith(".mp4"):
                src_path = os.path.join(root, file)

                # Получаем длительность видео
                duration = get_video_info(src_path)
                if duration is None:
                    continue # Пропускаем файл, если не удалось получить информацию

                duration_minutes = duration / 60 # Переводим секунды в минуты
                total_video_minutes += duration_minutes

                # Определяем путь в выходной папке
                relative_path = os.path.relpath(root, src_folder)
                output_dir = os.path.join(dest_folder, relative_path)
                os.makedirs(output_dir, exist_ok=True)
                output_path = os.path.join(output_dir, file)

                print(f" Сжатие: {src_path} → {output_path} (длительность: {duration:.2f} сек.)")

                # Получаем размер исходного файла
                original_size = get_file_size(src_path)

                # Засекаем время начала сжатия
                start_time = time.time()
```

Рис. 2. Инициализация параметров

Для примера, декодирование выполняется с применением кодера libx265, реализующего современный стандарт H.265/HEVC. Исходные файлы

предполагают расширение mp4. Параметры кодирования, такие как постоянный коэффициент скорости (CRF) и предустановка скорости обработки (preset), задаются пользователем и влияют на баланс между степенью сжатия, качеством и временем обработки. Аудиодорожка перекодируется в формат AAC с фиксированным битрейтом (рис.3).

```
# Кодирование видео
try:
    (
        ffmpeg
        .input(src_path)
        .output(output_path,
            vcodec="libx265", crf=crf, preset=preset, # Видео
            acodec="aac", audio_bitrate=audio_bitrate) # Аудио
        .run(overwrite_output=True)
    )
except ffmpeg.Error as e:
    print(f"✗ Ошибка сжатия {file}: {e.stderr.decode()}")
    continue
```

Рис. 3. Конвертация видео формата

По завершении обработки всех файлов производится расчёт суммарных и средних значений общего времени обработки, суммарной длительности всех видео в минутах, а также среднего процента сжатия по всем обработанным файлам. Также рассчитывается среднее время, требующее для сжатия одной минуты видеоматериала, что даёт оценку общей производительности процесса (рис. 4).

```
# Засекаем время окончания
end_time = time.time()
compression_time = end_time - start_time
total_time += compression_time # Добавляем в общий таймер

# Получаем размер сжатого файла
compressed_size = get_file_size(output_path)

# Среднее время сжатия за 1 минуту видео
avg_time_per_minute = compression_time / duration_minutes if duration_minutes > 0 else 0

# Рассчитываем процент сжатия
if original_size > 0:
    compression_percentage = ((original_size - compressed_size) / original_size) * 100
    size_ratio_percentage = (compressed_size / original_size) * 100
    total_compression_percentage += compression_percentage # Добавляем к общему проценту сжатия
else:
    compression_percentage = 0
    size_ratio_percentage = 0

# Записываем данные в отчет
report_data.append((file, duration, compression_time, avg_time_per_minute,
    original_size, compressed_size, compression_percentage, size_ratio_percentage))

print(f"📁 {file} сжато за {compression_time:.2f} сек. (среднее: {avg_time_per_minute:.2f} сек./мин.)")
print(f"📏 Размер исходного: {original_size / (1024**2):.2f} MB, Сжатого: {compressed_size / (1024**2):.2f} MB,"
    "Процент сжатия: {compression_percentage:.2f}%, Отношение размера: {size_ratio_percentage:.2f}%")

# Вычисляем общее среднее время сжатия за минуту видео
overall_avg_time_per_minute = total_time / total_video_minutes if total_video_minutes > 0 else 0

# Рассчитываем средний процент сжатия
avg_compression_percentage = total_compression_percentage / len(report_data) if len(report_data) > 0 else 0
```

Рис. 4. Расчёт значений для отчёта

Финальным этапом является формирование структурированного HTML-отчёта, содержащего таблицу с результатами по каждому файлу и сводными данными в конце документа. Отчёт сохраняется в выходной

директории. В консоль также выводятся итоговые метрики, позволяющие быстро оценить эффективность проведённого сжатия (рис. 5).

```
# Запись отчёта в HTML
report_path = os.path.join(dest_folder, "compression_report.html")
with open(report_path, "w", encoding="utf-8") as report_file:
    report_file.write("<html><head><title>Отчёт о сжатии видео</title></head><body>")
    report_file.write("<h1>Отчёт о сжатии видео</h1>")
    report_file.write("<table border='1' cellpadding='5' cellspacing='0'>")
    report_file.write("<thead><tr><th>Файл</th><th>Длительность (сек.)</th><th>Время сжатия (сек.)</th>")
    report_file.write("<th>Среднее (сек./мин.)</th><th>Исходный размер (МБ)</th><th>Сжатый размер (МБ)</th>")
    report_file.write("<th>Процент сжатия (%)</th><th>Отношение сжатого к исходному (%)</th></tr></thead>")
    report_file.write("<tbody>")

    for file, duration, time_taken, avg_time_per_minute, original_size, compressed_size, compression_percentage, size_ratio_percentage in report_data:
        report_file.write(f"<tr><td>{file}</td><td>{duration:.2f}</td><td>{time_taken:.2f}</td>")
        report_file.write(f"<td>{avg_time_per_minute:.2f}</td><td>{original_size / (1024**2):.2f}</td>")
        report_file.write(f"<td>{compressed_size / (1024**2):.2f}</td><td>{compression_percentage:.2f}</td>")
        report_file.write(f"<td>{size_ratio_percentage:.2f}</td></tr>")

    report_file.write("</tbody>")
    report_file.write(f"<p><strong>Среднее время сжатия за минуту видео (по всем файлам): "<strong>")
    report_file.write(f"<strong>{overall_avg_time_per_minute:.2f} сек.</strong></p>")
    report_file.write(f"<p><strong>Средний процент сжатия (по всем файлам): {avg_compression_percentage:.2f}%</strong></p>")
    report_file.write("</body></html>")

print(f"<br> Отчет сохранен в формате HTML: {report_path}")
print(f"<br> Среднее время сжатия за минуту видео: {overall_avg_time_per_minute:.2f} сек.")
print(f"<br> Средний процент сжатия: {avg_compression_percentage:.2f}%")
```

Рис. 5. Генерация HTML-отчёта

Для проверки корректности работы программы взят курс видео лекций по управлению сервисом Docker. Каталог содержит 16 директорий с видео файлами в кодировке H.264. Всего в курсе содержится 96 лекций с разной временной длительностью. Отчет, сгенерированный программой после выполнения конвертации с указанными в данной работе характеристиками, представлен на рисунке 6 .

Отчёт о сжатии видео

Среднее время сжатия за минуту видео (по всем файлам): 27.42 сек.
Средний процент сжатия (по всем файлам): 86.96%

Файл	Длительность (сек.)	Время сжатия (сек.)	Среднее (сек./ мин.)	Исходный размер (МБ)	Сжатый размер (МБ)	Процент сжатия (%)	Отношение сжатого к исходному (%)
001 Введение.mp4	348.43	124.97	21.52	107.17	14.08	86.86	13.14
002 Что такое Docker и почему его стоит использовать.mp4	175.97	52.55	17.92	48.88	5.94	87.85	12.15
003 Что такое контейнер.mp4	30.03	9.70	19.39	8.36	1.05	87.45	12.55
004 Компоненты Docker.mp4	114.03	33.46	17.61	33.95	3.39	90.00	10.00
005 Как работают контейнеры.mp4	351.60	109.95	18.76	106.67	10.98	89.71	10.29
006 Что такое образы и их строение.mp4	333.90	105.27	18.92	100.99	11.01	89.10	10.90
007 Что такое репозиторий.mp4	120.27	35.68	17.80	35.69	3.70	89.62	10.38
008 Какие программы нужны для этого курса.mp4	392.77	122.22	18.67	110.45	14.15	87.19	12.81
009 Основные команды Docker.mp4	142.40	40.62	17.11	36.34	4.47	87.69	12.31
010 Создание контейнера hello-world.mp4	613.77	187.44	18.32	157.31	22.76	85.54	14.46
011 Создание контейнера busybox.mp4	237.10	66.19	16.75	60.66	7.45	87.71	12.29
012 Есть ли доступ в интернет внутри контейнеров.mp4	287.40	100.90	21.06	80.62	10.27	87.26	12.74
013 Создание контейнера NGINX.mp4	261.03	85.64	19.69	66.44	9.40	85.85	14.15
014 Запуск контейнера в фоновом режиме.mp4	257.23	81.39	18.99	68.24	9.80	85.63	14.37
015 Остановка контейнеров.mp4	89.57	27.49	18.41	22.26	2.89	87.02	12.98
016 Запуск дополнительных процессов в работающем контейнере.mp4	237.50	71.10	17.96	61.18	7.31	88.05	11.95
017 Создание имени для контейнера.mp4	124.17	35.23	17.02	33.42	3.72	88.87	11.13
018 Публикация портов контейнера.mp4	239.73	75.79	18.97	64.54	7.94	87.70	12.30
019 Разные порты для разных контейнеров.mp4	159.70	57.26	21.51	39.21	5.64	85.60	14.40
020 Подключение томов.mp4	163.07	55.86	20.55	42.19	5.19	87.71	12.29
021 Создание собственной HTML-страницы для	671.80	211.72	18.91	180.29	21.60	88.02	11.98

Рис. 6. HTML-отчёт результата конвертирования

Полученный средний процент сжатия составил почти 87%, после конвертации в кодировку HEVC. При этом диапазон значений варьируется от 82 до 89 процентов.

4 Выводы

В текущем исследовании была рассмотрена реализации программы пакетного сжатия видеофайлов на примере MP4 и кодировщиков H.264 и HEVC. Описан способ взаимодействия языка программирования Python с консольной утилитой FFmpeg. Представлены основные параметры работы с видео и приведен пример программного выполнения.

Библиографический список

1. Крючкова Ю.П., Игнатов Ф.М., Власюк И.В. Анализ искажений изображений в современных видеокодеках // Фундаментальные проблемы радиоэлектронного приборостроения. 2009. Т. 9. № 4. С. 182-185.
2. Зубакин И.А., Козлов В.В., Цыцулин А.К. Методика тестирования видеокодеков с учетом меры не стационарности изображений // Телекоммуникации. 2012. № 8. С. 29-33.
3. Невмержицкий К.А. Отличительные особенности популярных аудио, видеокодеков и контейнеров // Двигатель. 2012. № 4 (82). С. 60-62.
4. Попов В.Д., Паршин А.Е. Алгоритм поиска оптимальных по Парето параметров кодирования видеокодеков на основе объективных метрик качества видео // Новые информационные технологии в автоматизированных системах. 2009. № 12. С. 30-38.
5. Рагулина К.О., Попов В.Д., Паршин А.Е. Алгоритмы анализа значений параметров кодирования видеокодеков // Новые информационные технологии в автоматизированных системах. 2009. № 12. С. 39-50.
6. Александров А.А., Умняшкин С.В. Модификация алгоритма кодирования разностного кадра для видеокодека на базе дискретного вейвлет-преобразования // Известия высших учебных заведений. Электроника. 2012. № 6 (98). С. 92-94.
7. Калистратов Д.С. Влияние параметров поисковых алгоритмов компенсации движения на показатели качества современных видеокодеков // Молодой ученый. 2014. № 5 (64). С. 20-23.
8. Погорелов К.В. Видеокодек для системы многоточечной видеоконференцсвязи // Известия ТРТУ. 2004. № 1 (36). С. 73-74.
9. Скуднев Д.М., Толчеев В.А. Видеокодеки и 4К-видео // В сборнике: Информационные технологии в процессе подготовки современного специалиста. Межвузовский сборник статей. Ответственный редактор Зияутдинов В.С., Липецк, 2015. С. 100-105.