Jellyfin Smart Transcoding

Jellyfin is an open-source media server software that allows you to organize, stream, and access your personal media library from various devices. It provides a user-friendly interface, supports a wide range of media formats, and offers features like transcoding, remote access, and user customization. With Jellyfin, you can enjoy your favorite movies, TV shows, music, and photos anytime, anywhere. The Problem with Jellyfin is that it doesn't do well when it comes to transcoding media.

The purpose of this paper is to come up with some solutions that can make transcoding process in Jellyfin smarter and more effective by changing the “only transcode on demand” behavior. This behaviour comes with certain side effects, lower transcoded video quality, Hogging system resources, laggy playback on low-end servers.

Transcoding can be done in more smarter ways (which will be discussed later in the paper) but first of all lests see the problems.

Codecs:

Most videos in the internet are encoded with h.264 codec, but many newer codecs have showed up in recent years that provide better compression while maintaining same quality. But since they are now as widely adopted as H.264, some devices may not support them (or at least not support them in hardware level).

Jellyfin codec support

Sorted by efficency (excluding bit depth)	Chrome	Edge	Firefox	Safari	Android	Android TV	iOS	SwiftFin (iOS)	Roku	Kodi	Desktop
MPEG-4 Part 2/SP	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
MPEG-4 Part 2/ASP	✗	✗	✗	✗	✗	✗	✗	✓		✓	✓
H.264 8Bit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
H.264 10Bit	✓	✓	✗	✗	✓	✓	✗	✓	✗	✓	✓
H.265 8Bit	◆[8]	✓[7]	✗	◆[1]	◆[2]	✓[5]	◆[1]	✓[6]	◆[9]	✓	✓
H.265 10Bit	◆[8]	✓[7]	✗	◆[1]	◆[2]	◆[5]	◆[1]	✓[6]	◆[9]	✓	✓
VP9	✓	✓	✓	✗	✓[3]	◆[3]	✗	✗	✓	✓	✓
AV1	✓	✓	✓	✗	✓	◆[4]	✗	✗	✓	✓	✓

The table above shows us the codecs supported in each of the platforms, as we can see H.264 8bit format is supported on all platforms but H.265 has limited support. So it is safe to assume that there will be a transcoding from H.265 to H.264 every time we play a video on a device that does not support H.265 decoding; this approach works well as long as the server has powerful GPU and it is not being used for other demanding services at the same time; but most home servers are either using integrated graphics or have very low-end and old GPUs that probably does not support H.265 codec and has to rely on software decoding. This means some compromisation must be done during the transcoding to keep up with the vide playback. In my server this is the difference between direct playback and transcoded playback:

Figure 1 - Transcoded Video Playing on Firefox

Figure 2 - Direct Play

Both videos are hosted on the same server, the transcoding is done by Intel Sandybridge GPU using VAAPI on Linux.

Now compare this to the ffmpeg output I got from following command:

```
ffmpeg -vaapi_device /dev/dri/renderD128 -i test-1080p-h265.mkv -vf 'format=nv12,hwupload' -c:v h264_vaapi -preset ultrafast -b:v 5000k -c:a copy test-1080p-h264.mkv
```

Figure 3 - Manually Transcoded

Not as good as direct play but much better than Jellyfin transcoding, lets look at the resource usage:

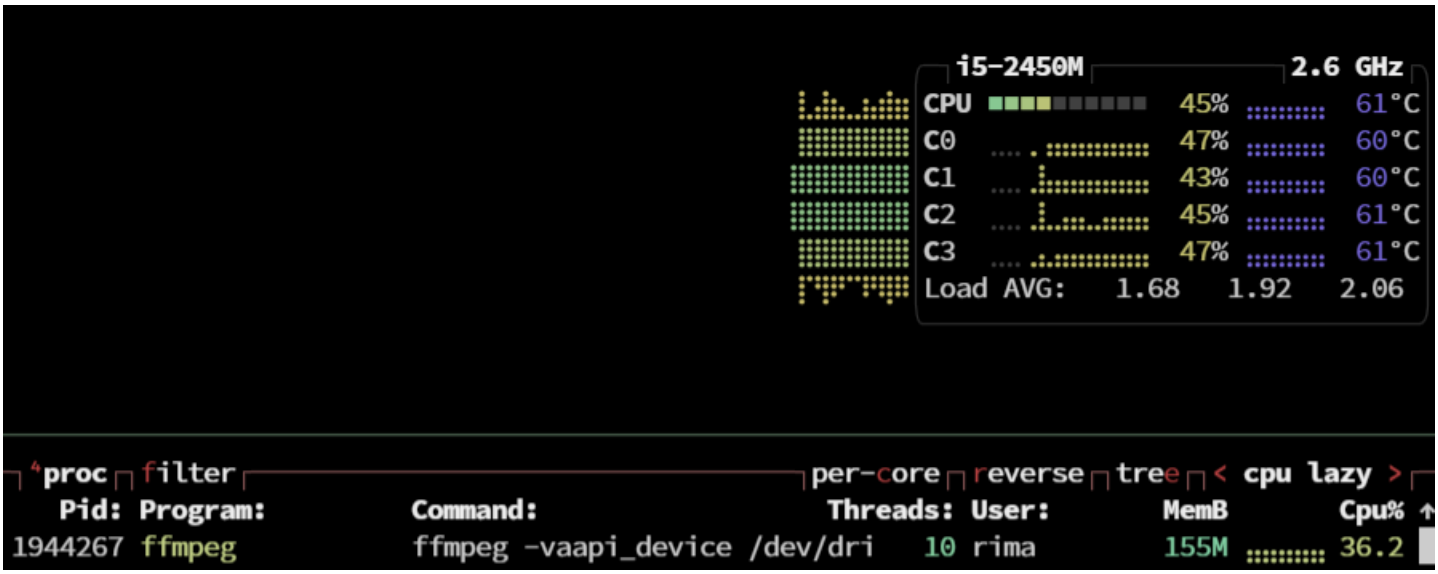


Figure 4 - Resource monitor during manual transcoding

And transcoding speed:

```
      _STATISTICS_WRITING_APP: mkvmerge v14.0.0 ('Flow') 64bit
      _STATISTICS_WRITING_APP-eng: mkvmerge v14.0.0 ('Flow') 64bit
      _STATISTICS_WRITING_DATE_UTC: 2017-07-29 07:09:14
      _STATISTICS_WRITING_DATE_UTC-eng: 2017-07-29 07:09:14
      _STATISTICS_TAGS: BPS DURATION NUMBER_OF_FRAMES NUMBER_OF_BYTES
      _STATISTICS_TAGS-eng: BPS DURATION NUMBER_OF_FRAMES NUMBER_OF_BYTES
      encoder         : Lavc58.134.100 ssa
frame=  455 fps= 42 q=-0.0 size=    3584kB time=00:00:19.18 bitrate=1530.8kbits/s speed=1.76x
```

Compare it to Jellyfin transcoding:

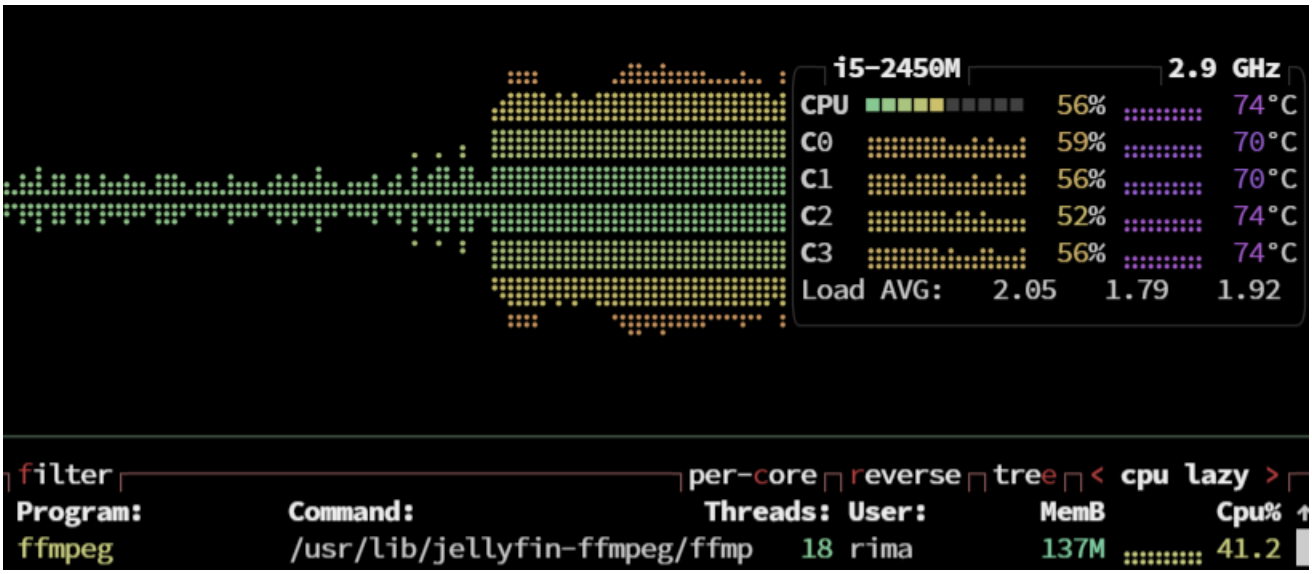


Figure 5 - Resource monitor during Jellyfin transcoding

The results are clearly indicating that Jellyfin is not aiming for efficiency or quality while transcoding, but it does transcode much faster. This makes sense since a playing movie is better than interrupted one. But, this is also raises the question, why not just transcode beforehand? Let's see the size difference between two files:

```
→ Test ls -l
total 2378400
-rw-rw-r-- 1 rima rima 1831195089 Nov 19 20:02 test-1080p-h264.mkv
-rwxrw-rw- 1 root root  604272268 Oct  4 17:13 test-1080p-h265.mkv
→ Test
```

Figure 6 - Original and converted file size

600Mb vs 1.8 Gb and I should point out that H.264 file is only half the length of the full movie which is in h265 file (I stopped the transcoding halfway since it was taking too long).

So, we are forced to choose between compatibility and space, at least until all devices start supporting h265.

What is the solution, what is next:

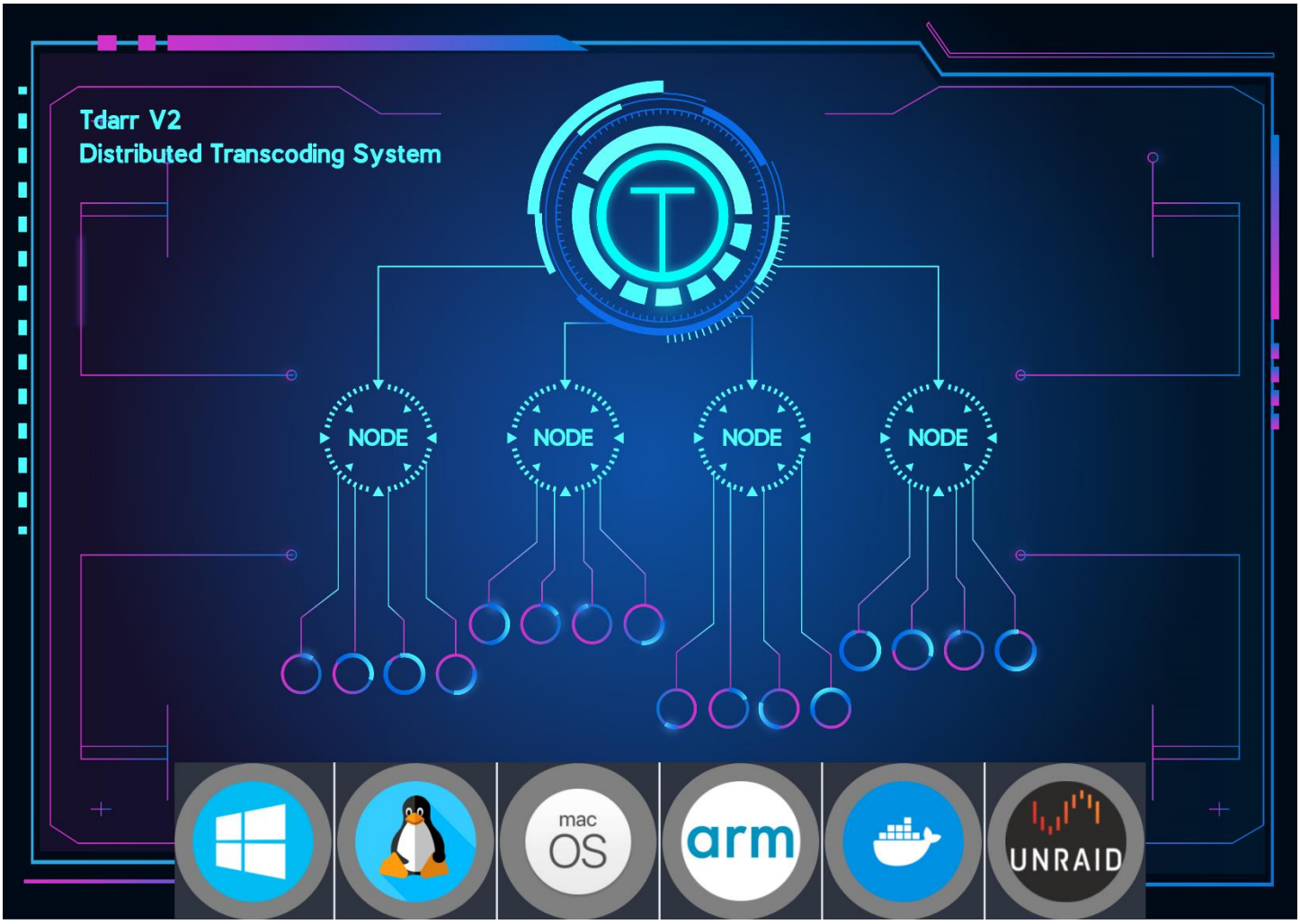
There is no straight forward solution, this project intended to provide an improvement idea to Jellyfin. This can be an integrated on Jellyfin or developed as an add-on.

Since we can achieve a better-quality video with less pinning on resource, why not transcode beforehand? this does not have to be always h265 to h264 transcode, users can have the free will to choose what they want to optimize their content for (space or compatibility, quality, or responsiveness (less bitrate?)). An even better solution would be smart choice of content to transcode; This can include saving the transcoded version of most played media or media with high probability to be played next (next episodes on a series) or transcoding all media to the suitable codec for the most used client devices.
all these smart transcoding operations can happen when system is sitting at idle to avoid pining down the resources.

Already existing services:

There are many different GUIs for FFmpeg that can provide a partial solution for the problem above, what they lack is the information about the preferred codec for client devices and playback info. This is a partial solution because it is not smart. An example of this kind of software is Tdarr, it automatically scans your media library and can use multiple GPUs (with ffmpeg on the background) to transcode your media to suitable encoding and also gives a very nice stats on the progress.

Tdarr is also distributed, it means you can offload some of the processing power needed from your server to other devices in your home lab.



But how do we make it smart? Well, that's where Jellyfin integration should come in, this can be a optional plugin or a built in feature. Jellyfin collects the playback and client data and based on them it can control the transcoding process of Tdarr. The distributed nature of Tdarr also means that we don't have to wait for server to be idle to start transcoding, if any other system in the network is available it can handle the transcoding. Also the sophisticated stats from Tdarr can help us monitor the smart transcoding process and intervene if needed.

Fully integrated in Jellyfin:

Here is a sample script that transcodes all the media in given directory to HVEC if CPU usage is less than 10%

```
import os
import subprocess
import psutil
import time

def check_cpu_idle(threshold=10):
    cpu_percent = psutil.cpu_percent(interval=1)
    return cpu_percent < threshold

def convert_to_hevc(input_dir, output_dir):

    # Get a list of all video files in the input directory
    video_files = [
        file for file in os.listdir(input_dir)
        if file.lower().endswith(('.mp4', '.avi', '.mkv', '.mov'))
    ]

    # Iterate over each video file and convert it to HEVC
    for file in video_files:
        while not check_cpu_idle(threshold=10):
            time.sleep(100)  # Wait for 100 seconds before checking CPU usage again

        input_path = os.path.join(input_dir, file)
        output_path = os.path.join(output_dir, f"{os.path.splitext(file)[0]}.hevc.mp4")
        command = [
            "ffmpeg",
            "-i", input_path,
            "-c:v", "libx265",
            "-crf", "23",
            "-preset", "medium",
            "-c:a", "copy",
            output_path
        ]


        subprocess.run(command)

    print("Conversion complete.")
```

```
input_directory = "/path/to/input/directory"
output_directory = "/path/to/output/directory"

convert_to_hevc(input_directory, output_directory)
```

a more sophisticated version of this script can be integrated into Jellyfin to take advantage of playback data and predict which videos should be prioritize for transcoding.

References and sources:

https://jellyfin.org/docs/general/server/transcoding/

https://home.tdarr.io/

https://ffmpeg.org/documentation.html