Greening the Video Transcoding Service with Low-Cost Hardware Transcoders

Peng Liu, ***University of Wisconsin—Madison;*** **Jongwon Yoon,** ***Hanyang University;***
Lance Johnson, ***University of Minnesota;*** **Suman Banerjee,** ***University of Wisconsin—Madison***

https://www.usenix.org/conference/atc16/technical-sessions/presentation/liu

This paper is included in the Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16).

June 22–24, 2016 • Denver, CO, USA

978-1-931971-30-0

Open access to the Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16) is sponsored by USENIX.

Greening The Video Transcoding Service With Low-Cost Hardware Transcoders[1]

Peng Liu[*], Jongwon Yoon[†], Lance Johnson[‡], and Suman Banerjee[*]

[*]University of Wisconsin-Madison, {pengliu,suman}@cs.wisc.edu
[†]Hanyang University, jongwon@hanyang.ac.kr
[‡]University of Minnesota, lmj@umn.edu

Abstract

Video transcoding plays a critical role in a video streaming service. Content owners and publishers need video transcoders to adapt their videos to different formats, bitrates, and qualities before streaming them to end users with the best quality of service. In this paper, we report our experience to develop and deploy VideoCoreCluster, a low-cost, highly efficient video transcoder cluster for live video streaming services. We implemented the video transcoder cluster with low-cost single board computers, specifically the Raspberry Pi Model B. The quality of the transcoded video delivered by our cluster is comparable with the best open source software-based video transcoder, and our video transcoders consume much less energy. We designed a scheduling algorithm based on priority and capacity so that the cluster manager can leverage the characteristics of adaptive bitrate video streaming technologies to provide a reliable and scalable service for the video streaming infrastructure. We have replaced the software-based transcoders for some TV channels in a live TV streaming service deployment on our university campus with this cluster.

1 Introduction

Video streaming service is one of the most popular Internet services in recent years. In particular, multimedia usage over HTTP accounts for an increasing portion of today's Internet traffic [47]. For instance, video traffic is expected to be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014 [5]. In order to provide high and robust quality of video streaming services to end users with various devices with diverse network connectivities, content owners and distributors need to encode the video to different formats, bitrates, and qualities. It is redundant to encode and store a source video to different variants for archiving purposes. The broad spectrum of varieties of bitrates, codecs and formats make it difficult for some video service providers to prepare all media content in advance. Therefore, video transcoding has been widely used for optimizing video data. For example, Netflix encodes a movie as many as 120 times before they stream the video to users [37]. Transcoders are also commonly used in the area of mobile device content adaptation, where a target device does not support the format or has a limited storage capacity and computational resource that mandate a reduced file size.

However, video transcoding is a very expensive process, requiring high computational power and resources. Thus, it is critical to have an energy efficient and low-cost video transcoding solution. In addition, video transcoders' performance is important to enhance the overall quality of video streaming services. Regarding the performance of transcoder, two metrics are important: quality and speed. Video quality with a given bitrate determines the amount of data needed to be transmitted in the network for a video. Transcoding speed determines the time to finish the transcoding. Thus, it is critical for live video streaming services. Other metrics, e.g., cost and power consumption, are also need to be considered in video transcoding system deployment.

Various video transcoding technologies are proposed and used, including cloud transcoding, software-based transcoding on local servers, and hardware transcoding with specialized processors. In this paper, we introduce VideoCoreCluster – a low-cost, energy efficient hardware-assisted video transcoder cluster to provide transcoding services for a live video streaming service. The cluster is composed of a manager and a number of cheap single board computers (Raspberry Pi Model B). We use the hardware video decoder and encoder modules embedded in the System on Chip (SoC) of a Raspberry Pi to facilitate video transcoding. With an optimized transcoding software implementation on the Raspberry Pi, each Raspberry Pi is able to transcode up to 3 Standard Definition (SD, 720x480) videos or 1 High

Definition (HD, 1280x720) and 1 SD videos in real time with very low power consumption. We also developed the cluster manager based on an IoT machine-to-machine protocol - MQTT [42] to coordinate the transcoding tasks and hardware transcoders in order to provide reliable transcoding service for video streaming services. Compared to software-based video transcoders, VideoCoreCluster has lower cost and higher energy efficiency.

Adaptive bitrate (ABR) video streaming over HTTP is a popular technology to provide robust quality of service to end users whose mobile devices have dynamic network connectivities. Multiple ABR technologies are used in the industry, but they share a similar idea. Media servers cut source videos into small segments and encode every segment to multiple variants with different bitrates. During playback, a video player selects the best variants for these segments base on the video player's performance, network connectivity, or user's preference. Streaming video over HTTP has many advantages [60], we chose it for our live video streaming service because it is easy to deploy a video player on web browsers of mobile devices with different operating systems, and we do not need to reconfigure middle-boxes (firewalls, NATs, etc.) in current campus network deployment for our service. We design VideoCoreCluster according to the requirements of ABR live video streaming so that it can provide robust transcoding service.

Contributions: The most important contribution of this paper is a practical implementation of real-time transcoding system based on energy efficient hardware-assisted video transcoders. Our contributions are multi-fold:

- We design and implement a cost-effective transcoding solution on a low-cost device, Raspberry Pi Model B.
- Our system is based on hardware video decoder and encoder, which is significantly more energy efficient than software-based transcoding system.
- We leverage characteristics of adaptive bitrate video streaming over HTTP to design a reliable and scalable system for live video streaming service. The system is easily deployable. We currently employ our VideoCoreCluster into an IP-based TV streaming service in the University of Wisconsin-Madison.

The paper is organized as follows. In Section 2, we introduce the background of our video transcoding system and the possible approaches to building it. Then we describe the architecture of VideoCoreCluster in Section 3. We evaluate the system in Section 4 and discuss the related work in Section 5. Section 6 concludes the paper.

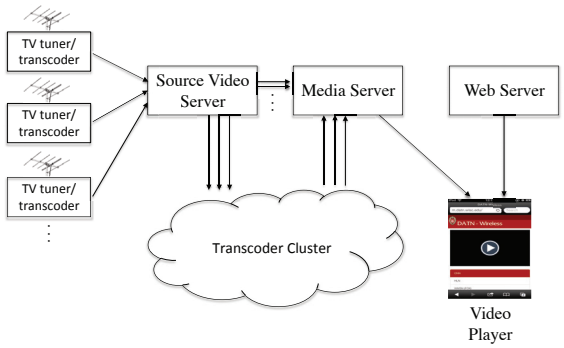


Figure 1: TV Streaming Service Architecture.

2 Background and Setup

We worked with the IT department of the University of Wisconsin-Madison to provide a free TV service for students on campus. Students can watch 27 (currently) TV channels with their mobile devices. Six of these channels are HD channels with a resolution 1280x720, 30fps. The other 21 channels are SD with resolutions up to 720x480, 30fps. In the most recent month (April 2016), there were more than 4000 view sessions and about total 480 watching hours. ABR video streaming techniques are used to provide high-quality video streaming service to mobile devices with diverse link capabilities and network dynamics. The system supports both Apple's HTTP Live Streaming (HLS) [13] and MPEG-DASH [60] to provide service to heterogeneous devices. Figure 1 shows the architecture of the system. Specifically, a TV frontend receives the TV signal and encodes the video stream into H.264 + AAC format, then the video stream is pushed to *source video server*. The *transcoder cluster* pulls videos from the *source video server* and pushes the transcoded results to *media server* in order to provide multiple variants of the video streams for every TV channel. The *source video server* and *media server* can be combined in deployment, so we will not discuss them separately in the following sections. The web server hosts web pages and video players for different web browsers.

2.1 Challenges of ABR Techniques on Live Video Streaming

Low latency is critical for a live video streaming service, in which a media server has to generate video data on-the-fly to provide continuous streaming service to the end user. A media server supporting ABR needs to guarantee all the variants of a video segment are available when a client requests one of them. Due to the strict requirement on low-latency transcoding, several optimization techniques (e.g., high throughput video transcoding,

multi-pass encoding for enhancing video quality) are not applicable for live streaming. Moreover, an index file containing a list of the available video variants should be generated in real-time, and it has to be updated on time and be consistent with the availability of these variants. We also need to make sure the variants of a video segment are generated synchronously to simplify the implementation of ABR algorithms on the video players. Furthermore, an efficient video transcoding system is desired and high reliability is required to provide 24/7 video streaming services to users without interruptions.

2.2 Video Transcoding

A video transcoder is composed of a video decoder and an encoder. Some researchers have discussed possible video transcoding designs, which mingle the modules of a video decoder and an encoder [40]. However, having separate video decoders and encoders provides more flexibility because we can easily have various decoder/encoder combinations to create different video transcoders for various purposes. In this work, we only discuss the video transcoder built by a separate video decoder and an encoder.

Most popular video codecs have well-defined standards. These standards strictly define the compliant bitstreams and decoder's behaviors. For many video coding techniques, including H.264 [46], different video decoder implementations are required to generate the identical video frames (output) with respect to the same input by their standards. This makes the selection of video decoder easy, and hence, hardware video decoders would be the best choice in most cases as long as it is available at low cost due to its high efficiency.

On the other hand, developers are free to design a video encoder's implementation as long as the generated bitstream can be decoded by the reference decoder implementation. Therefore, different encoder implementations can generate different bitstreams with various qualities for the same video frames. In this way, the interoperability is guaranteed while innovations on the encoder design are encouraged. As a result, we need to evaluate an encoder's performance on video quality in addition to the encoding speed when we select an encoder for a transcoder. Software video encoders are well known for their low efficiency. Hameed et al. [48] point out that application-specific integrated circuit (ASIC) implementation of the video encoder is 500 times more energy efficient than the software video encoder running on general purpose processors. They assume that both hardware and software implementations use the same algorithms for the corresponding procedures, e.g., motion estimation, intra-prediction, etc.. The high efficiency is only from the advantage of specialized hardware implementation. However, software video encoders are more flexible than hardware video encoders. It is much easier to try new algorithms on a software video encoder to improve video quality than a hardware video encoder. Video encoders on FPGA platform make a good trade-off on efficiency and flexibility, so they are also widely used in industry.

Different video applications have different requirements on video encoder. For mobile devices, the energy efficiency is crucial for battery life. Therefore, most of the SoCs for mobile phones include video decoder/encoder IPs [7, 36]. For broadcast applications, high video quality is desired while real-time encoding and flexibility are critical. Toward this, video encoders on the FPGA platform are widely used. For on-demand streaming applications, the low energy efficiency of software video encoders can be amortized by streaming one encoded result to many users. High latency is not a big concern because the service provider can encode video offline. Slow encoding speed can be overcome by launching a large number of instances in a cloud platform to run the video encoders in parallel to achieve very high throughput video encoding. The advantages of video quality and flexibility are the main reason that software video encoder is widely used to prepare video contents for on-demand streaming service. For instance, Netflix adapts the software-based transcoder because of its flexibility, after an unsuccessful deployment of a specialized hardware video transcoding system [23].

H.264 is a popular video standard widely used in diverse video applications. Since H.264 is the video coding standard that our system supports, we only discuss the available H.264 encoder implementations in this paper as shown in Table 1. The authors in [6] compared different H.264 encoder implementations, which includes software implementations (x264, DivX H.264, etc.), GPU-accelerated implementation (MainConcept CUDA), and hardware implementation (Intel QuickSync Video). Their conclusions include *(i)* x264 is one of the best codecs regarding video quality, and *(ii)* Intel QuickSync is the fastest encoder of those considered. Even though x264 is the best software H.264 encoder for our project, its low efficiency hinders its deployment. Given that we need to keep the system running 24/7 for continuous TV service in the campus network, it is not economical to rent cloud computing resource (e.g., Amazon EC2 instances) to execute the transcoding tasks. Building in-house transcoding system with highly efficient hardware video decoders and encoders is the best choice while keeping the cost low in our system.

We use the hardware H.264 decoder and encoder in a low-cost SoC – Broadcom BCM2835, which is the chip of a popular single board computer - Raspberry Pi Model B. VideoCoreCluster leverages its powerful GPU - VideoCore IV, which embeds hardware multimedia de-

Encoder Type	Explanations
Software	JM: The reference H.264 implementation from JVT[11]. It is widely used for research purpose and conformance test. It is too slow to be used in practical projects. x264: The most popular open source software H.264 encoder implementation. Compared to JM, it is about 50 times faster and provides bitrates within 5% of the JM reference encoder for the same PSNR[38, 57]. OpenH264: An open source H.264 implementation from Cisco[28]. It is optimized and the encoder runs much faster than JM, but it is slower and has fewer features than x264. Other proprietary implementations: MainConcept[17], Intel's IPP H.264 encoder[15], etc.
GPU-based	Three GPU vendors: Intel, NVIDIA, and AMD, all integrate hardware video codecs in their GPUs. They also provide GPU-accelerated video encoders, which leverage GPU's high throughput graphics engine to accelerate video encoding[27, 16]. For example, NVIDIA has two different versions of video encoder implementations: NVCUVENC and NVENC. NVCUVENC is a CUDA software-based implementation while NVENC is based on dedicated encoding hardware engine. NVCUVENC will not be available in the future because NVENC's improved performance and quality.
FPGA-based	Xilinx and its partners have professional solutions for broadcast applications[10]. They provide H.264 decoder and encoder IP for Xilinx's FPGA platforms. FPGA-based implementation is more flexible than ASIC implementation and more efficient than the software implementation. But it is more expensive than both of them.
Encoder IP in SoC	Many SoCs for mobile devices or other embedded devices have dedicated hardware decoders and encoders. For example, Qualcomm's chips for mobile phones [7]; Ambarella's chips for different video applications [2]; Broadcom's chip for TV set-top boxes.

Table 1: Different type of H.264 encoders

coders and encoders. The primary computation power of the cluster is from the VideoCore co-processors. The Raspberry Pi is a low cost (under $35) single board computer with very good software support [31, 33, 35]. Therefore, it is adequate to build a cost-effective video transcoder cluster.

3 VideoCoreCluster Architecture

The VideoCoreCluster is composed of a cluster manager and a number of transcoders. We use the MQTT protocol to transfer control signals between the cluster manager and the transcoders. MQTT is based on a publish/subscribe messaging pattern rather than a traditional client-server model, where a client communicates directly with a server. The cluster manager and the transcoders connect to an MQTT message broker. They exchange information by subscribing to topics and publishing messages to topics. The message payload is encoded with Google Protocol Buffer for minimal overhead [32]. RTMP [34] is used in the data path. Figure 2 shows the architecture of the VideoCoreCluster.

3.1 Media Server

The media server supports HLS, MPEG-DASH, and RTMP. HLS and DASH are for the video players, whereas RTMP is for the transcoders. HLS and DASH are two popular adaptive bitrate streaming standards to stream video over HTTP and they are widely supported by mobile operating systems and web browsers. RTMP is designed to transmit real-time multimedia data, and thus, it guarantees to transfer source video to the transcoders and the transcoded video to the media server with minimum latency. RTMP's session control features for media applications are used to implement the interactions between media server and transcoders in our configuration.

Figure 3 shows the responsibilities of the media server in our system. For every TV channel, there are multiple transcoded video streams (variants) with different bitrates and qualities. Each video stream is split to chunks with the same duration on the media server. It is important to align the boundaries of those chunks from different variants even different workers may not be exactly synchronized. To ensure synchronization, we set the Instantaneous Decoder Refresh (IDR) interval of a source video and the transcoded video to 2 seconds. Then the media server uses the IDR frame as the boundary to split the streams. It uses the timestamp of the IDR frame with 2 seconds (IDR interval) granularity to define the segment number. So that we can keep the chunks to be synchronized from a video player's view as long as the progress offset of different workers for the same source video is under 2 seconds (IDR interval). We can set the IDR interval to a larger value to obtain higher tolerance on transcoding speed variation. The index file for a channel has the information about all variants of the video. It is dynamically updated by the media server based on the

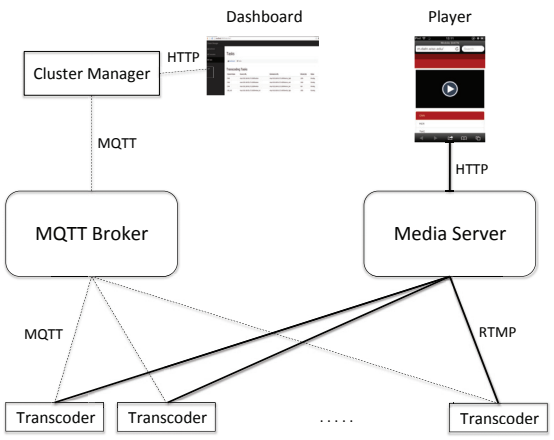


Figure 2: The components and connections between the components in the VideoCoreCluster. Every transcoder node maintains two separate network connections for the control flow and data flow respectively. Administrator monitors the status of VideoCoreCluster and updates transcoding tasks through the dashboard of the cluster manager.

availability of the video streams. One worker may temporarily fail to generate the corresponding stream, and the transcoder cluster can migrate the failed task from one worker to another within a second. RTMP specification requires the initial timestamp of a stream to be 0 [34]. To ensure all other RTMP streams for the same channel as the failed stream have the same timestamp, we need to reset their RTMP connections as well. And the media server can always generate a consistent index file so that we can guarantee the reliability of the video streaming service.

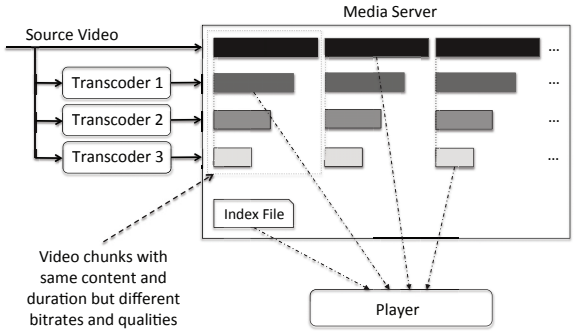


Figure 3: Overview of the Media Server's responsibilities: (i) Splitting video streams to chunks with the same duration (IDR interval). (ii) Refreshing the index file according to the status of the transcoded streams.

3.2 Transcoder Design

The Raspberry Pi Model B has a weak ARM CPU but a powerful GPU (VideoCore IV). Given that, our design strategy is to run simple tasks on the CPU, and offload compute-intensive tasks to the GPU. There are two types of processes in a transcoder: cluster agent and transcoding worker. There is one cluster agent on the board, and from 0 to 3 transcoder workers depending on the transcoding task's requirement on the computing resource. Figure 4 shows the relationship between these two types of processes.

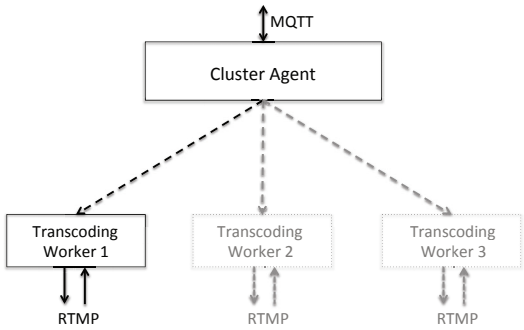


Figure 4: Two types of processes on a transcoder. A transcoding worker is the child process of the cluster agent process.

A cluster agent executes on the ARM processor only. It has two responsibilities: (i) Reporting the status of the board and workers to the cluster manager. (ii) Accepting commands from the cluster manager and launching or killing transcoding worker processes. A cluster agent will report the available resource to the cluster manager when it registers to the cluster manager. The cluster manager dispatches transcoding tasks to a transcoder based on the available resources of the transcoder. A cluster agent maintains an MQTT connection to the MQTT message broker by sending keep-alive packets if no information flows between the transcoder and the MQTT message broker for a predefined interval. If a cluster agent is disconnected from the broker or the cluster manager is offline, the transcoder will stop all transcoding workers. The cluster manager can revoke a transcoding task from a transcoder if it wants to assign the task to another transcoder. Failed transcoding tasks will be reported to the cluster manager, which can assign them to other transcoders.

A transcoding worker is a process to transcode a video stream. It only transcodes video data but passes audio data. Video transcoding tasks primarily execute on the VideoCore IV co-processor. The ARM processor is responsible for executing the networking protocol stack, video streams demuxing/muxing, and audio data pass

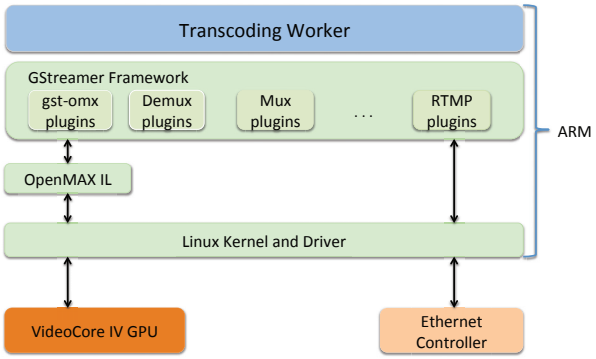


Figure 5: Software architecture of a transcoding worker. Compute-intensive video transcoding task executes on VideoCore IV GPU, whereas ARM processor is responsible for coordination and data parsing/movement.

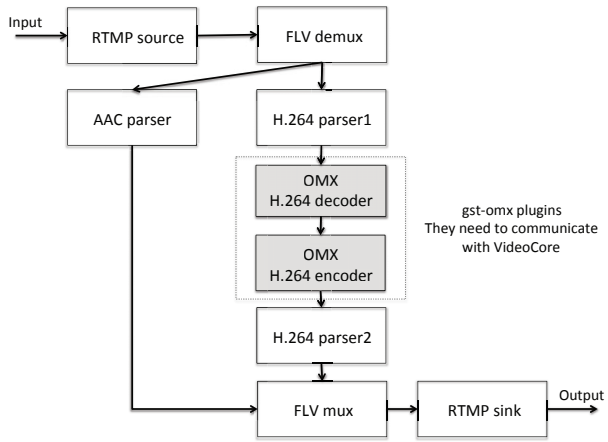


Figure 6: GStreamer pipeline of the transcoding worker process. Some trivial plugins, e.g., buffering, synchronization, etc. are not shown here.

through. In BCM2835, an application layer software can access the hardware video decoder and encoder through the OpenMAX IL interfaces [31, 29]. Rather than calling the OpenMAX IL interfaces directly, we built the program with the GStreamer framework [9]. GStreamer is an open source multimedia framework widely used to build media processing applications. It has a well-designed filter (plugin) system. The gst-omx is the GStreamer OpenMAX IL wrapper plugin that we use to access the hardware video decoder and encoder resources. Figure 5 shows the software architecture of a transcoding worker.

A transcoding worker creates a pipeline of GStreamer plugins. Video data are processed by the plugins one by one, sequentially. Figure 6 shows the structure of a pipeline. All plugins except H.264 decoder and H.264 encoder plugins fully execute on the ARM processor.

The default behavior of a GStreamer plugin can be summarized as 3 steps: (i) Read data from the source pad. (ii) Process the data. (iii) Write data to the sink pad. The GStreamer pipeline moves data and signals between the connected source pad and sink pad. Plugins work separately and process the data sequentially. This means both the H.264 decoder and H.264 encoder need to move a large amount of data back and forth between the memories for the ARM processor and the VideoCore IV GPU, which wastes CPU cycles. We modified the gst-omx implementation to enable hardware tunneling between the decoder and encoder, which significantly reduce the CPU load [8]. Without the hardware tunneling, a transcoder worker cannot support real-time transcoding of a 1280x720, 30fps video. Whereas after we enable it, a transcoder worker can simultaneously transcode one 1280x720 video and one 720x480 video in real-time. Figure 7 illustrates the data movement in the pipeline without hardware tunneling. Figure 8 illustrates the data movement in the pipeline with hardware tunneling.

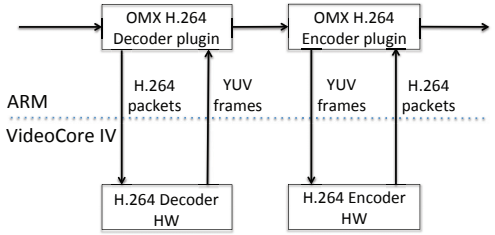


Figure 7: The decoder and encoder plugins work independently. YUV frames need to be moved from the VideoCore IV memory to the ARM memory by the decoder plugin, then they need to be moved from the ARM memory to the VideoCore IV memory by the encoder plugin.

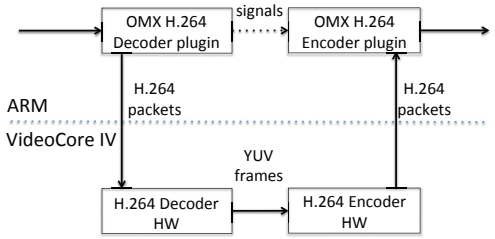


Figure 8: A hardware tunnel is created between the decoder and encoder. Plugins do not need to touch the YUV frames. Therefore, the workload of the ARM processor is reduced.

3.3 Cluster Manager Design

The cluster manager maintains a task pool and a transcoder pool. Its major responsibilities are assigning the transcoding tasks to the transcoders, and migrating the failed tasks from one transcoder to another. Both transcoding tasks and transcoders have priorities, which need to be considered when the cluster manager schedules the tasks. The cluster manager maintains a series of lists of tasks with different priorities. Every transcoding task has the following information: { ID, channel name, command, bitrate, resource, priority, state }. The resource is an integer representing the required computational resource for the task. Its value depends on the source video type (SD or HD) and target bitrate. Each task has four possible states: idle, assigning, revoking, and running. The cluster manager employs an event-driven design. When anything happens to the tasks or transcoders, e.g., task failure, a new worker joining, a worker leaving, etc., the cluster manager will check whether rescheduling is necessary and do so if it is.

When a new worker sends a register message to the cluster manager, the cluster manager will add a record to the worker list and keep tracking its status with a predefined interval. Transcoders will send the status of themselves and tasks running on them to the cluster manager periodically. The transcoder status includes CPU load, VideoCore IV load, temperature, and free memory. MQTTs *last will message* mechanism is used to implement the online status tracking of the transcoders. The cluster manager also embeds a web server to provide a dashboard for administrators to monitor the cluster's status and change the configurations. We have three design goals for the cluster manager: scalability, reliability, and elasticity.

Scalability: Scalability is ensured by an event-driven design and minimum control flow information. The cluster manager only maintains critical information about the transcoders. The information that frequently exchanges between the cluster manager and transcoders is only about the status. Media servers manage the source videos and transcoded videos. We can replicate the media server if its workload is too high. The separation of data flow and control flow ensures the cluster manager's scalability.

Reliability: We ensure the reliability of the system by real-time status monitoring coupled with low latency scheduling to migrate the failed tasks to working transcoders. The media server updates the index file on-the-fly to consistently list correctly transcoded video streams. A temporary failure will not affect the video players.

Elasticity: We define priorities for tasks and transcoders. An important characteristic of adaptive bitrate video streaming is that every video program has multiple versions of encoded videos. The more variants of a video available, the more flexible the players can optimize the user experience. Our system leverages that characteristic to implement an elastic transcoding service. When the available transcoders have more resources to run all the transcoding tasks, all the tasks will be assigned to transcoders. But if not, only the high priority tasks will be scheduled and executed. We can easily extend the transcoder cluster by adding more transcoders with this elastic design to support more TV channels.

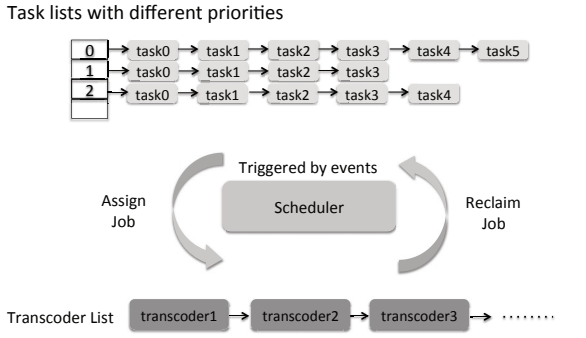


Figure 9: Overview of the cluster manager. The scheduler is an event-driven design.

Figure 9 shows the internal data structures of the cluster manager. The scheduler makes decisions based on the capacities of the transcoders and the resource requirements of the transcoding tasks. The capacity and resource requirement are both positive integers. Their values determine what kind of tasks and how many tasks can run on a transcoder in real-time.

Table 2 presents several task examples. A Raspberry Pi Model B's capacity is 20, so it can run one HD transcoding task, e.g., 4 or 5, or two SD transcoding tasks (1 and 2), or three SD transcoding tasks (2, 3, and 7), or one HD transcoding task and 1 SD transcoding task (3 and 6). When a new transcoder registers to the cluster manager, the idle task in the highest priority task list will be assigned to it. When a task fails and returns to the cluster manager, the task manager will try to assign it to another transcoder. If the cluster manager can find a transcoder that has enough capacity for it, it will assign the failed task to that transcoder. If not, the cluster manager will try to revoke tasks with lower priorities from a transcoder and then assign this task to that transcoder. If the cluster manager can not find a running task which has lower priority than the failed task, the cluster manager will not do anything. As discussed in section 3.1, if we successfully reschedule a failed task, we need to send messages to transcoders to reset the RTMP connections of those streams corresponding to the same channel as

ID	Channel	Command	Resource	Priority
1	A	transcode rtmp://192.168.1.172:1935/live/A_src rtmp://192.168.1.172:1935/live/A_800k 800	10	0
2	A	transcode rtmp://192.168.1.172:1935/live/A_src rtmp://192.168.1.172:1935/live/A_600k 600	8	1
3	A	transcode rtmp://192.168.1.172:1935/live/A_src rtmp://192.168.1.172:1935/live/A_400k 400	6	0
4	B	transcode rtmp://192.168.1.172:1935/live/B_src rtmp://192.168.1.172:1935/live/B_2400k 2400	20	0
5	B	transcode rtmp://192.168.1.172:1935/live/B_src rtmp://192.168.1.172:1935/live/B_1600k 1600	16	1
6	B	transcode rtmp://192.168.1.172:1935/live/B_src rtmp://192.168.1.172:1935/live/B_800k 800	14	0
7	C	transcode rtmp://192.168.1.172:1935/live/C_src rtmp://192.168.1.172:1935/live/C_400k 400	6	0

Table 2: Transcoding task examples

the failed task.

3.4 Implementation

We implemented both the cluster manager and transcoders on the Linux operating system. We extensively used open source software in this project, including Apache [3], Nginx [24], node.js + Express [26], mqtt.js [22], paho MQTT client library [21], GStreamer, and Google protobuf. We used Mosquitto [20] as the MQTT message broker.

Media Server: We built the media server with Nginx + Nginx_RTMP_module [25] and Apache. The RTMP protocol is implemented by Nginx, whereas the HTTP interface is provided by Apache. The media server is installed on a server with Ubuntu 14.04 LTS.

Cluster Manager: The cluster manager is written in Javascript, and built on node.js + Express. We use mqtt.js to develop the MQTT client module that subscribes and publishes messages related to the transcoders. The cluster manager is also installed on a server with Ubuntu 14.04 LTS.

Transcoder: The transcoder's two components – transcoder worker and cluster agent are implemented in C/C++. They depend on GStreamer, paho MQTT client library, and Google protobuf. We built the SDK, root disk and Linux kernel for Raspberry Pi with buildroot [4], then we built the two components with the customized SDK.

3.5 Deployment

We deploy VideoCoreCluster in an incremental way. Currently, we leverage a hybrid approach to provide transcoding service for the live video streaming service. The deployment has a small scale VideoCoreCluster with 8 Raspberry Pi Model Bs and a transcoder cluster composed of five powerful servers with Intel Xeon processors. We plan to extend the size of VideoCoreCluster and eventually replace all the servers in the deployment.

4 Evaluations

We evaluate VideoCoreCluster on video quality and transcoding speed with benchmark tests. We also analyze the power consumption of VideoCoreCluster and compare it with a transcoder cluster built with general-purpose processors.

4.1 Video Quality Test

The H.264 decoder module of VideoCore IV can support real-time decoding of H.264 high profile, level 4.0 video with resolutions up to 1920x1080 with very low power consumption. In addition, the decoding result is exactly same as the reference H.264 decoders. So the quality loss of our transcoding system is only from the encoder. Many hardware video encoders have some optimizations to simplify the hardware design while sacrificing video quality, especially for the video encoder modules in SoCs for mobile devices, because of the stringent restriction on power consumption. In order to have a clear idea about the video quality of the VideoCore's H.264 video encoder, we conducted benchmark tests on it and compared its performance with x264.

Both subjective and objective metrics are available for evaluating the video quality. Subjective metrics are desired because they reflect the video quality from users' perspective. However, their measurement procedures are complicated [18, 30]. In contrast, objective metrics are easy to measure; therefore, they are widely used in video encoder developments, even though there are arguments about them. Two metrics, Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM) Index, are

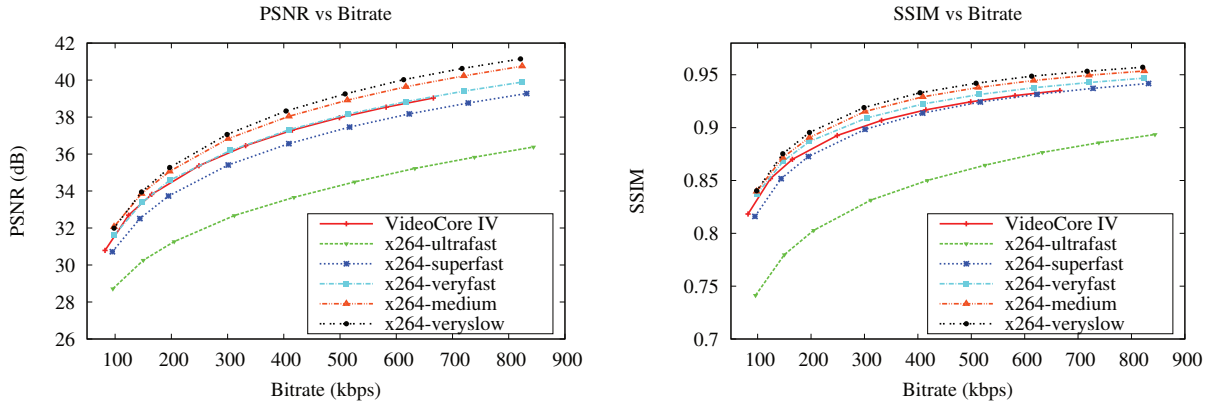


Figure 10: Video quality test result of VideoCore IV and x264 with different presets. We used foreman(352x288, 25fps) YUV sequence to test the encoders, set IDR to 2 seconds, and disabled B-frame support of x264.

widely used to compare video encoders. PSNR is the traditional method, which attempts to measure the visibility of errors introduced by lossy video coding. Huynh-Thu et al. showed that as long as the video content and the codec type are not changed, PSNR is a valid quality measure [50]. SSIM is a complementary framework for quality assessment based on the degradation of structural information [63]. We evaluated the hardware video encoder's performance with both the PSNR and SSIM.

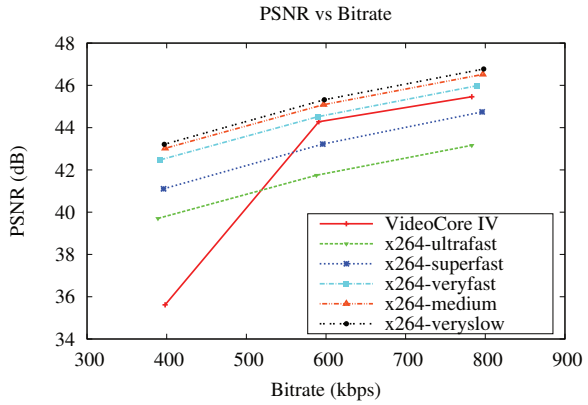
The x264 has broad parameters to tune, which are correlated and it is hard to achieve the optimal configuration. Rather than trying different parameter settings, we used the presets provided by x264 developers to optimize the parameters related to video encoding speed and video quality. The x264 has ten presets: ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow, and placebo (the default preset is medium). They are in descending order of speed and ascending order of video quality. As the video quality increases, encoding speed decreases exponentially.

We used two sets of video sequences for evaluating the encoders. The first one is the YUV sequences commonly used in video coding research [39], so the results can be easily reproduced and compared with other researcher's results. The second one is captured from our deployment that reflects the encoder's performance in practical deployment. We natively compiled the x264-snapshot-20150917-2245 on a desktop with Ubuntu 14.04 LTS. We also cross-compiled the libraries, drivers and firmware for Raspberry Pi from [33, 35] on the same machine. One thing we notice is that the H.264 encoder of VideoCore IV does not support B-frames. The reason is that the target applications of the SoC are real-time communication applications, e.g., video chatting and conference, where low latency is a strict requirement. B-frame leads to high encoding/decoding latency. Thus, the H.264 encoder of VideoCore IV does not support it. For a fair comparison, we test x264 with and without B-frame support to check the impact of B-frame support on video quality vs. bitrate.

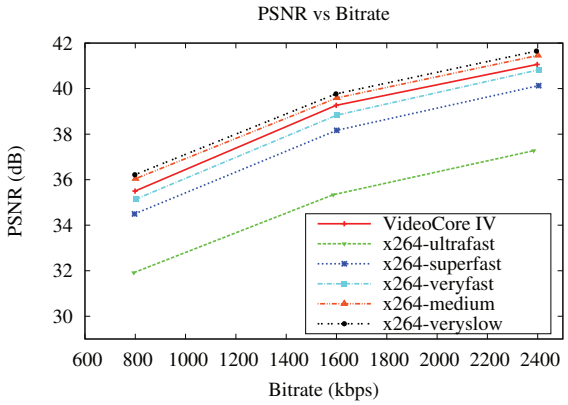
For all the video sequences we tested in the first set, we obtained similar results, though the exact numbers vary. We also found that the VideoCore's video encoder generated very low-quality video when the target bitrate was very low. That could be a bug in the video encoder's implementation. For brevity, we only show the results for foreman_cif here. We omit the results of x264 with B-frame because B-frame does not have a significant impact on the quality in our encoding settings. From figure 10, we can see VideoCore IV has similar or better performance regarding video quality comparing to the x264 with preset superfast. Since the purpose of the second test set is to evaluate the video encoder's performance in practical deployment, we only evaluated the encoder's performance with the typical bitrates. Figures 11a and 11c indicate that the VideoCore has poor performance on low bitrate settings. However, we believe it is not a big concern for deployment. When the players have to use that low bitrate version of the video streams, the player's network performance must be very low. We do not expect that will be a common condition. As shown in Figures 11b and 11d, VideoCore's video quality is good for high bitrate settings. Its quality is close to x264 with preset medium.

4.2 Transcoding Speed Test

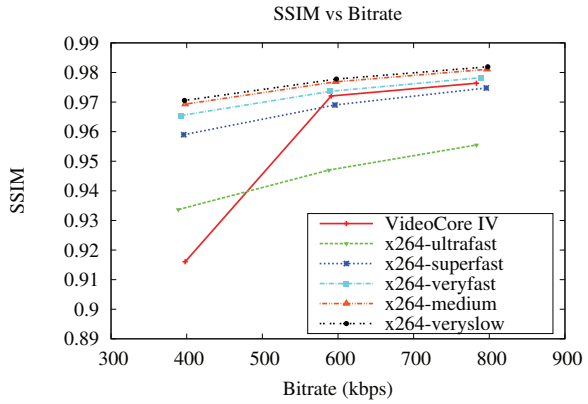
We measured the transcoder's performance under stress. We transcoded the video streams captured from an SD channel and an HD channel offline and recorded the time for transcoding. There are overheads on demuxing and muxing in the process, but because of the high complex-



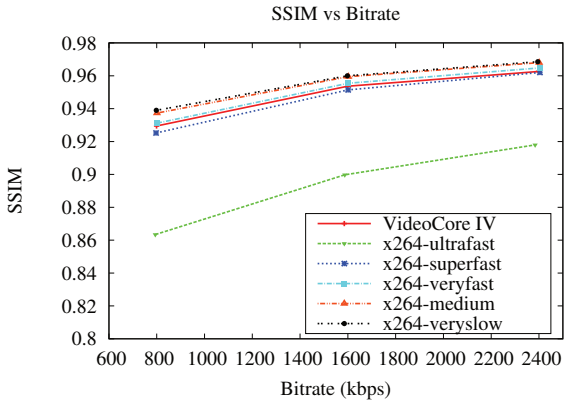
(a) PSNR values of an SD channel (720x480, 30fps)



(b) PSNR values of an HD channel (1280x720, 30fps)



(c) SSIM values of an SD channel (720x480, 30fps)



(d) SSIM values of an HD channel (1280x720, 30fps)

Figure 11: Video quality test result of VideoCore IV and x264 with different presets. We maintained the same configurations (IDR is 2 seconds, B-frame support of x264 is disabled).

ity of transcoding (decoding + encoding), the bottleneck of the process is on video transcoding. The detail specifications of the test videos are (1) SD channel: 720x480, 30fps, H.264 high profile, level 4.0, 1.2 Mbps. (2) HD channel: 1280x720, 30fps, H.264 high profile, level 4.0, 4Mbps. For some SD channels with lower resolutions in our deployment, the transcoding speed is higher than the results shown here. We transcoded the SD and HD videos to 800kbps and 2.4Mbps respectively and kept other parameters the same.

The hardware video encoder in VideoCore IV can support encoding 1920x1080, 30fps, H.264 high profile video in real-time. But when we run the decoder and encoder at the same time, the performance is not sufficient to support such high-resolution transcoding in real-time because the video decoder and encoder share some hardware resources. Even with the optimization described in section 3, the transcoder can only support transcoding video with resolution up to 1280x720 in real-time.

We also measured software transcoder's speed for comparison. The software video transcoder we used is FFmpeg, which has built-in H.264 video decoder. We linked it with libx264 to provide H.264 video encoding. The desktop we used to run FFmpeg has Intel Core i5-4570 CPU @ 3.20GHz and 16GB RAM. We built FFmpeg and libx264 with all the CPU capabilities (MMX2, SSE2Fast, SSSE3, SSE4.2, AVX, etc.) to accelerate the video transcoding. We tested superfast (similar quality as the video encoder of VideoCore IV) and medium (default) presets of x264. Figure 12 shows that when the output video qualities are similar, software transcoder executing on powerful Intel i5 CPU runs about 5.5x and 4x faster than the video transcoder running on Raspberry Pi for SD and HD channel respectively. For our transcoding system, which transcodes video in real-time, that means we can run 5.5x (SD) or 4x (HD) more transcoding tasks on a desktop than a Raspberry Pi. However, a Raspberry Pi is much cheaper and consumes much less power than a desktop with an Intel i5 CPU.

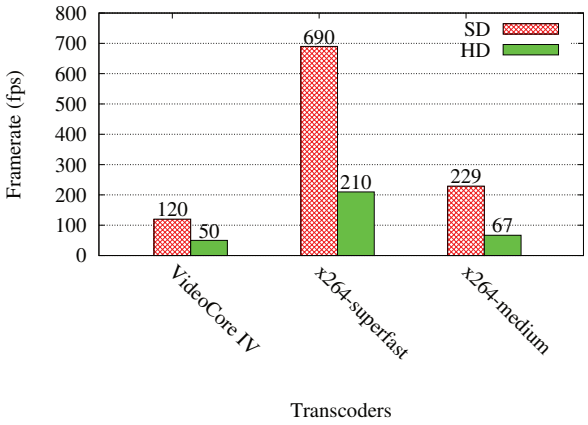


Figure 12: Transcoding speed of VideoCore IV and x264.

4.3 Power Consumption Analysis

We can see the superiority of Raspberry Pi regarding power efficiency from the transcoding speed test. An Intel Core i5-4570 processor has an average power consumption of 84W [14]. If we include the power consumption of other components, e.g., RAM and Hard Disk, the power consumption of a server would be higher than 100W. Raspberry Pi Model B in a configuration without any peripherals except Ethernet has typical power consumption about 2.1W [45]. The desktop consumes more than 40 times power than the Raspberry Pi Model B while it can do about 5.5x SD video transcoding or 4x HD video transcoding. We can see the VideoCoreCluster is more energy efficient than a transcoder cluster built with general-purpose processors to provide the same transcoding capacity. We omit the power consumption analysis on the network switches in our system deployment because we can use the same switches for the different video transcoders.

5 Related Work

Video Transcoding: Video transcoding is critical for video streaming service deployment. Different approaches and architectures have been proposed to implement it for various use cases. Vetro et al. discussed the transcoding of block-based video coding schemes that use hybrid discrete cosine transform (DCT) and motion compensation (MC) [62]. Xin et al. discussed several techniques for reducing the complexity and improving video quality by exploiting the information extracted from the input video bit stream [65]. Unlike their research, we believe that the cascaded decoder and encoder approach is much more straightforward and flexible. As the hardware video encoder improving quality and efficiency and reducing the cost, a cascaded pixel-domain approach is more suitable for practical deployments. For a particular scenario, Youn et al. showed that for point-to-multipoint transcoding, a cascaded video transcoder is more efficient since some parts of the transcoder can be shared [67].

Cloud Transcoding: Li et al. introduced a system using cloud transcoding to optimize video streaming service for mobile devices [54]. Video transcoding on a cloud platform is a good solution to transcode a large volume of video data because of its high throughput. For instance, Amazon, Microsoft, and Telestream Cloud provide cloud transcoding service for users [19, 1, 12]. Netflix also deployed their video transcoding platform on Amazon's cloud [23].

Specialized Hardware for Video Applications: Efficiency issue of general-purpose processors on multimedia applications, including video decoding and encoding, has attracted a lot of research efforts. Various approaches have been studied to improve the hardware efficiency, including specialized instructions [58, 44], specialized architectures [52, 59, 64], GPU offloading [55, 43], application-specific integrated circuit(ASIC) [56], and FPGA-based accelerators [53].

Adaptive Bitrate Video Streaming: Adaptive bitrate video streaming is a widely used technique by video streaming service providers to provide high-quality video streaming services. Designing a robust and reliable algorithm to switch bitrate is challenging. Many researchers have proposed adaptation algorithms to achieve a better video quality in dynamic network environments [51, 49, 66]. These works focus on the client side implementation, whereas our paper concentrates on the server side.

Computer Cluster: Computer cluster is a well-known scheme of distributed system used to provide high throughput computing. For example, Condor is a distributed system for scientific applications [61]. Our system is unique in the sense that the target application is real-time computing, and the computing nodes are specialized, low cost and highly efficient hardware. FAWN is also a cluster built with low-power embedded CPUs. However, it is a system only for the data-intensive computing [41]. Our system is both the data-intensive and computation-intensive.

6 Conclusion and Future Work

High-quality video transcoding is critical to ensure high-quality video streaming service. We implemented VideoCoreCluster, a low-cost, highly efficient video transcoder system for live video streaming service. We built the system with commodity available, low-cost single board computers embedding high performance and low power

video encoder hardware module. We implemented the cluster based on a network protocol for IoT for the control path and RTMP for the data path. This separation design can get low latency in video transcoding and data delivery. Our system has much higher energy efficiency than the transcode cluster built with general-purpose processors, and it does not sacrifice quality, reliability, or scalability. We can use VideoCoreCluster on other live video streaming services, and we can further improve the system on capability and energy efficiency by upgrading the transcoders.

7 Acknowledgements

We thank Derek Meyer for his help in the system deployment. We are grateful to our shepherd, Anthony Joseph, and the anonymous reviewers whose comments helped bring the paper to its final form. All authors are supported in part by the US National Science Foundation through awards CNS-1555426, CNS-1525586, CNS-1405667, CNS-1345293, and CNS-1343363.

References

[1] Amazon elastic transcoder. https://aws.amazon.com/elastictranscoder/.

[2] Ambarellas broadcast infrastructure solutions. http://www.ambarella.com/products/broadcast-infrastructure-solutions#S3.

[3] Apache http server project. https://httpd.apache.org/.

[4] Buildroot - making embedded linux easy. https://buildroot.org/.

[5] Cisco visual networking index: Forecast and methodology, 2014-2019 white paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[6] Eighth mpeg-4 avc/h.264 video codecs comparison - standard version. http://www.compression.ru/video/codec_comparison/h264_2012/.

[7] Enabling the full 4k mobile experience: System leadership. https://www.qualcomm.com/documents/enabling-full-4k-mobile-experience-system-leadership.

[8] gst-omx. https://github.com/pliu6/gst-omx.

[9] Gstreamer: open source multimedia framework. http://gstreamer.freedesktop.org/.

[10] H.264 4k video encoder. http://www.xilinx.com/products/intellectual-property/1-4iso3h.html.

[11] H.264/avc reference software. http://iphome.hhi.de/suehring/tml/download/.

[12] High quality video transcoding in the cloud. https://cloud.telestream.net/.

[13] Http live streaming. https://developer.apple.com/streaming/.

[14] Intel core i5-4570 processor. http://ark.intel.com/products/75043/Intel-Core-i5-4570-Processor-6M-Cache-up-to-3_60-GHz.

[15] Intel integrated performance primitives. https://software.intel.com/en-us/intel-ipp.

[16] Introducing the video coding engine (vce). http://developer.amd.com/community/blog/2014/02/19/introducing-video-coding-engine-vce/.

[17] Mainconcept. http://www.mainconcept.com/.

[18] Methodology for the subjective assessment of the quality of television pictures. https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf.

[19] Microsoft azure media services. https://azure.microsoft.com/en-us/services/media-services/encoding/.

[20] Mosquitto - an open source mqtt v3.1/v3.1.1 broker. http://mosquitto.org/.

[21] Mqtt c++ client for posix and windows. https://eclipse.org/paho/clients/cpp/.

[22] Mqtt.js. https://github.com/mqttjs.

[23] Netflix's encoding transformation. http://www.slideshare.net/AmazonWebServices/med202-netflixtranscodingtransformation.

[24] Nginx. https://www.nginx.com/.

[25] nginx-rtmp-module. https://github.com/pliu6/nginx-rtmp-module.

[26] Node.js. https://nodejs.org/en/.

[27] Nvidia video codec sdk. https://developer.nvidia.com/nvidia-video-codec-sdk.

[28] Openh264. http://www.openh264.org/.

[29] Openmax integration layer application programming interface specification. https://www.khronos.org/registry/omxil/specs/OpenMAX_IL_1_1_2_Specification.pdf.

[30] P.910 : Subjective video quality assessment methods for multimedia applications. https://www.itu.int/rec/T-REC-P.910/en.

[31] Programming audiovideo on the raspberry pi gpu. https://jan.newmarch.name/RPi/.

[32] Protocol buffers. https://developers.google.com/protocol-buffers/?hl=en.

[33] Raspberry pi firmware. https://github.com/raspberrypi/firmware.

[34] Real-time messaging protocol (rtmp) specification. http://www.adobe.com/devnet/rtmp.html.

[35] Source code for arm side libraries for interfacing to raspberry pi gpu. https://github.com/raspberrypi/userland.

[36] State of the art of video on smartphone. http://ngcodec.com/news/2014/3/13/state-of-the-art-of-video-on-smartphone.

[37] To stream everywhere, Netflix encodes each movie 120 times. https://gigaom.com/2012/12/18/netflix-encoding/.

[38] x264. http://www.videolan.org/developers/x264.html.

[39] Yuv video sequences. http://trace.eas.asu.edu/yuv/.

[40] AHMAD, I., WEI, X., SUN, Y., AND ZHANG, Y.-Q. Video transcoding: an overview of various techniques and research issues. *Multimedia, IEEE Transactions on 7*, 5 (2005), 793–804.

[41] ANDERSEN, D. G., FRANKLIN, J., KAMINSKY, M., PHANISHAYEE, A., TAN, L., AND VASUDEVAN, V. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 1–14.

[42] BANKS, A., AND GUPTA, R. Mqtt version 3.1. 1. *OASIS Standard* (2014).

[43] CHEN, W.-N., AND HANG, H.-M. H. 264/avc motion estimation implmentation on compute unified device architecture (cuda). In *Multimedia and Expo, 2008 IEEE International Conference on* (2008), IEEE, pp. 697–700.

[44] DAIGNEAULT, M.-A., LANGLOIS, J. P., AND DAVID, J. P. Application specific instruction set processor specialized for block motion estimation. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on* (2008), IEEE, pp. 266–271.

[45] DICOLA, T. Embedded linux board comparison. `https://learn.adafruit.com/downloads/pdf/embedded-linux-board-comparison.pdf`.

[46] DRAFT, I. recommendation and final draft international standard of joint video specification (itu-t rec. h. 264— iso/iec 14496-10 avc). *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050 33* (2003).

[47] GEBERT, S., PRIES, R., SCHLOSSER, D., AND HECK, K. Internet Access Traffic Measurement and Analysis. In *Proc. of ACM TMA* (2012).

[48] HAMEED, R., QADEER, W., WACHS, M., AZIZI, O., SOLOMATNIKOV, A., LEE, B. C., RICHARDSON, S., KOZYRAKIS, C., AND HOROWITZ, M. Understanding sources of inefficiency in general-purpose chips. In *ACM SIGARCH Computer Architecture News* (2010), vol. 38, ACM, pp. 37–47.

[49] HUANG, T.-Y., JOHARI, R., MCKEOWN, N., TRUNNELL, M., AND WATSON, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM* (2014), ACM, pp. 187–198.

[50] HUYNH-THU, Q., AND GHANBARI, M. Scope of validity of psnr in image/video quality assessment. *Electronics letters 44*, 13 (2008), 800–801.

[51] JIANG, J., SEKAR, V., AND ZHANG, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (2012), ACM, pp. 97–108.

[52] KIM, S. D., AND SUNWOO, M. H. Asip approach for implementation of h.264/avc. *Journal of Signal Processing Systems 50*, 1 (2008), 53–67.

[53] LEHTORANTA, O., SALMINEN, E., KULMALA, A., HÄNNIKÄINEN, M., AND HÄMÄLÄINEN, T. D. A parallel mpeg-4 encoder for fpga based multiprocessor soc. In *Field Programmable Logic and Applications, 2005. International Conference on* (2005), IEEE, pp. 380–385.

[54] LI, Z., HUANG, Y., LIU, G., WANG, F., ZHANG, Z.-L., AND DAI, Y. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video* (2012), ACM, pp. 33–38.

[55] LIN, Y.-C., LI, P.-L., CHANG, C.-H., WU, C.-L., TSAO, Y.-M., AND CHIEN, S.-Y. Multi-pass algorithm of motion estimation in video encoding for generic gpu. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on* (2006), IEEE, pp. 4–pp.

[56] LIN, Y.-K., LI, D.-W., LIN, C.-C., KUO, T.-Y., WU, S.-J., TAI, W.-C., CHANG, W.-C., AND CHANG, T.-S. A 242mw, 10mm 2 1080p h. 264/avc high profile encoder chip. In *Proceedings of the 45th annual Design Automation Conference* (2008), ACM, pp. 78–83.

[57] MERRITT, L., AND VANAM, R. Improved rate control and motion estimation for h. 264 encoder. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on* (2007), vol. 5, IEEE, pp. V–309.

[58] PELEG, A., AND WEISER, U. Mmx technology extension to the intel architecture. *Micro, IEEE 16*, 4 (1996), 42–50.

[59] SEO, S., WOH, M., MAHLKE, S., MUDGE, T., VIJAY, S., AND CHAKRABARTI, C. Customizing wide-simd architectures for h.264. In *Systems, Architectures, Modeling, and Simulation, 2009. SAMOS'09. International Symposium on* (2009), IEEE, pp. 172–179.

[60] STOCKHAMMER, T. Dynamic adaptive streaming over http–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems* (2011), ACM, pp. 133–144.

[61] THAIN, D., TANNENBAUM, T., AND LIVNY, M. Distributed computing in practice: The condor experience. *Concurrency-Practice and Experience 17*, 2-4 (2005), 323–356.

[62] VETRO, A., CHRISTOPOULOS, C., AND SUN, H. Video transcoding architectures and techniques: an overview. *Signal Processing Magazine, IEEE 20*, 2 (2003), 18–29.

[63] WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (2004), 600–612.

[64] WOH, M., SEO, S., MAHLKE, S., MUDGE, T., CHAKRABARTI, C., AND FLAUTNER, K. Anysp: anytime anywhere anyway signal processing. In *ACM SIGARCH Computer Architecture News* (2009), vol. 37, ACM, pp. 128–139.

[65] XIN, J., LIN, C.-W., AND SUN, M.-T. Digital video transcoding. *Proceedings of the IEEE 93*, 1 (2005), 84–97.

[66] YIN, X., JINDAL, A., SEKAR, V., AND SINOPOLI, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), ACM, pp. 325–338.

[67] YOUN, J., XIN, J., SUN, M.-T., AND ZHANG, Y.-Q. Video transcoding for multiple clients. In *Visual Communications and Image Processing 2000* (2000), International Society for Optics and Photonics, pp. 76–85.

Notes

[1] During the course of this work, J. Yoon was a graduate student at the University of Wisconsin-Madison, and L. Johnson was a staff at the University of Wisconsin-Madison.