Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 7470234, 13 pages
https://doi.org/10.1155/2018/7470234

Research Article

VideoCoreCluster: Energy-Efficient, Low-Cost, and Hardware-Assisted Video Transcoding System

Peng Liu,[1] Jongwon Yoon,[2] Ha Ryung Kim,[2] and Suman Banerjee[1]

[1]*Computer Sciences Department, University of Wisconsin-Madison, Madison, WI, USA*
[2]*College of Computing, Hanyang University, Ansan, Republic of Korea*

Correspondence should be addressed to Jongwon Yoon; jongwon@hanyang.ac.kr

Received 23 November 2017; Revised 22 April 2018; Accepted 29 April 2018; Published 5 June 2018

Academic Editor: Giovanni Pau

Video streaming is one of the killer applications in recent years. Video transcoding plays an important role in the video streaming service to cope with the various purposes. Specifically, content owners and publishers heavily utilize video transcoders to reconfigure source video in a variety of formats, video qualities, and bitrate to provide end users with the best possible quality of service. In this paper, we present VideoCoreCluster, a low-cost and energy-efficient transcoder cluster that is suitable for live streaming services. We designed and implemented real-time video transcoder cluster using cheap ($35), powerful, and energy-efficient Raspberry Pi. The quality of transcoded video provided by VideoCoreCluster is similar to the best software-based video transcoder while consuming significantly less energy (<3 W). We have proposed a scheduling algorithm based on priority of video stream and transcoding capacity. Our cluster manager provides reliable and scalable streaming services, because it uses the characteristics of adaptive bitrate scheme. We have deployed our transcoding cluster to provide IP-based TV streaming services on our university campus.

1. Introduction

Video streaming service is one of the most popular Internet services in recent years. The volume of video traffic is expected to be 80 percent of all consumer Internet traffic in 2019, at 64 percent in 2014 [1]. Video content owners and distributors need to encode video in various formats, qualities, and bitrates to provide high and robust video streaming service quality to end users under varying network conditions. However, it is difficult for video service providers to prepare media contents in a variety of formats due to numerous bitrates, codecs, and formats. To overcome such difficulty and to optimize video data, video transcoding has been widely used. Despite the practical necessity of video transcoding, it is a compute-intensive process, and hence high computing power and resources are required. The performance of video transcoder directly affects the quality of the video streaming service. High-speed transcoding solution is important for providing real-time video streaming services. This is because the transcoding speed determines the time it takes to complete the transcoding task. In addition, we need to consider energy consumption and the transcoder's cost when designing and implementing video transcoding system.

In this paper, we introduce VideoCoreCluster, a low-cost, energy-efficient, and hardware-assisted video transcoder cluster to provide real-time transcoding solutions for handling live streams. VideoCoreCluster consists of cluster manager and video transcoders built with Raspberry Pi, a cost-effective single-board computer. To facilitate video transcoding, we use a hardware video decoder and encoder module embedded in Raspberry Pi SoC (System on Chip). In addition, we optimize the transcoding software implementation on Raspberry Pi; thus, a single Raspberry Pi can transcode up to three SD (720 × 480) videos or one HD (1280 × 720) and one SD videos in real-time. We have developed the cluster manager to coordinate transcoding tasks and hardware transcoders. Our design ensures reliable transcoding services for live streaming. Moreover, VideoCoreCluster provides low cost and high energy-efficiency compared to other software-based transcoders.

Adaptive bitrate (ABR) video streaming over HTTP is a popular technology that provides a better quality of service

for end users in dynamic network conditions. The media server divides the source video into several smaller segments and encodes them at different video bitrates. During playback, the video player chooses the best variant taking into account the performance of the video player, network connection, and user preference. We employ ABR to our video streaming service because of its benefits and easy deployment on various web browsers [2]. Toward this, we designed VideoCoreCluster to meet the requirements of ABR live video streaming, and hence VideoCoreCluster is able to support robust bitrate adaptation. Unlike UDP-based streaming solutions, our transcoding solution employs HTTP based streaming and is fully compatible with other systems; therefore, it is not necessary to reconfigure the middle-boxes in networks.

Contributions. The very contribution of this paper is the implementation of a real-time transcoding system in practice.

(i) We design and implement a cost-effective, energy-efficient transcoding solution with Raspberry Pi.

(ii) We use adaptive bitrate video streaming to design a reliable and scalable system for live streaming service.

(iii) Our system is implemented with hardware video decoder and encoder and consumes significantly less energy (<3 W) than software-based transcoding systems.

(iv) VideoCoreCluster is deployed and provides IP-based TV services at the University of Wisconsin-Madison.

The paper is organized as follows. In Section 2, we introduce the background of a video transcoding system and a feasible approach to realize it. The architecture and details of VideoCoreCluster are described in Section 3. We evaluate the performance of VideoCoreCluster in Section 5 and discuss the related work in Section 6. Section 7 concludes this paper.

2. Background and Motivation

2.1. Challenges of ABR Technique on Live Streaming. Packet delay (latency) has a significant impact on the quality of live streaming. The media server has to immediately generate video segments to provide the users with timely and continuous streaming service. ABR server must ensure that all video variants are available when user requests any of them and switches to other variants without pausing. In addition, an index file containing a list of all available video variants should be generated and updated in real time, consistent with the availability of videos. Several optimization techniques (e.g., high-throughput video transcoding, multipass encoding) are not applicable for live streaming, because of the strict requirements for low latency. It has to ensure the variants of video segments are generated synchronously to simplify the implementation of ABR algorithms. Moreover, a highly reliable system is desired to provide continuous streaming services to multiple users. Due to the reasons mentioned above, ABR technique is challenging to support live streams.

2.2. Video Transcoder. Given the difficulties of applying ABR technique on live streaming, video transcoding received attention as an alternative solution. Video transcoding encodes (converts) an input stream to various formats to overcome the diversity and compatibility. In detail, the basic pipeline for video transcoding is as follows: decode audio → filter → encode demultiplex → decode video → filter → encode. The decoding and encoding are the most time consuming processes (filtering, multiplexing, and demultiplexing are almost negligible). Even though, encoding complexity highly depends on the preset, profile, settings, etc., encoding complexity is much higher than decoding complexity.

The main part of a video transcoder consists of a video decoder and an encoder. There are several design choices for decoder and encoder, but we focus on a separate design of decoder and encoder because of their flexibility. The most popular video codecs have well-defined standards that precisely define the compliant bitstreams and operations of decoder. Hardware decoders are selected in most cases because of their superior performance. Unlike decoder, developers can freely design a customized video encoder as long as the output bitstream can be decoded by the reference implementation. This allows the encoder to generate multiple bitstreams of varying quality for the same video segment. The performance of video transcoder highly depends on that of encoder (e.g., encoder's video quality and speed). Hardware encoder outperforms software encoder; however, software encoder provides greater flexibility and makes it easier to adapt new encoding algorithm for improving video quality.

2.3. Video Standard (H.264). H.264 is a widely used video standard in many video applications (our VideoCoreCluster also supports H.264). The authors in [3] have compared the performance of multiple H.264 encoder implementations, e.g., software (x264, DivX H.264), GPU-accelerated, and hardware implementations. They have concluded that (i) x264 is one of the best codecs in terms of video quality, and (ii) hardware-based one is the fastest encoder [4]. Despite the high performance of x264 (among software H.264 encoders), x264 is inadequate for our purpose because of its in-efficiency. To overcome such in-efficiency, we have built a customized video transcoding system with high-efficiency hardware decoders and encoders while keeping costs low (details in next).

3. VideoCoreCluster Architecture

3.1. System Overview. The University of Wisconsin-Madison network provides free IP-based TV service on campus. Students can watch various TV channels (6 HD and 21 SD channels) everywhere on campus. All TV sessions are delivered using the HTTP Live Streaming (HLS) [5] and MPEG-DASH [6] standards (HLS and MPEG-DASH are widely used for video streaming). Figure 1 depicts the architecture of IP-based TV system. Once TV tuner encodes the received TV signals in H.264 (for video) and AAC (for audio) formats, the video streams are delivered to the source video server. The VideoCoreCluster fetches videos from the

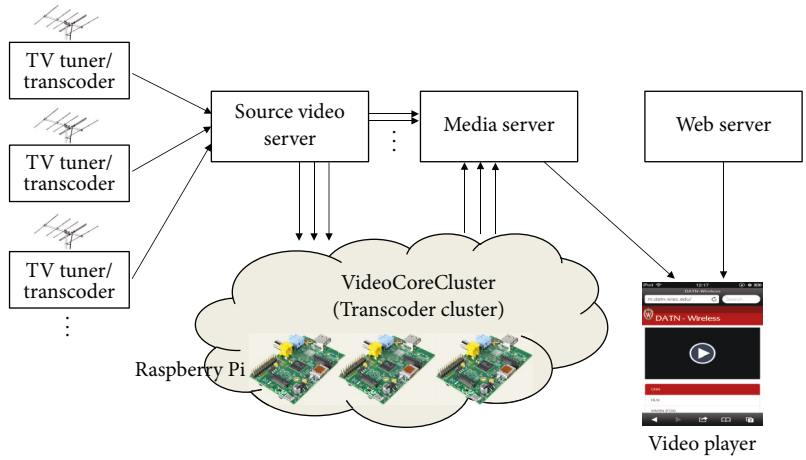


Figure 1: The architecture of the IP-based TV service; VideoCoreCluster (grey cloud) is integrated to provide video transcoding service.

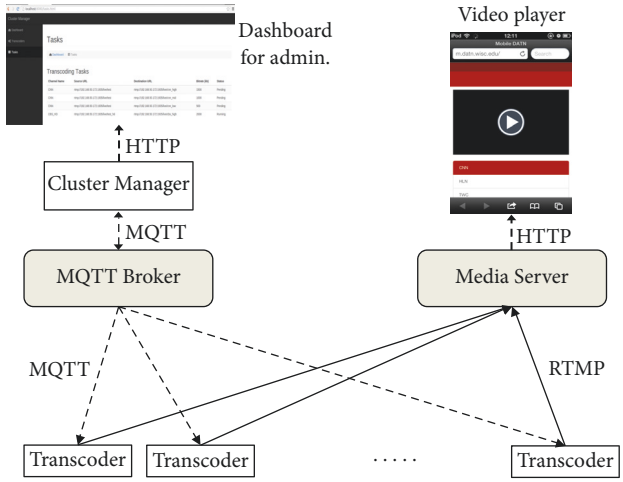


Figure 2: VideoCoreCluster architecture.

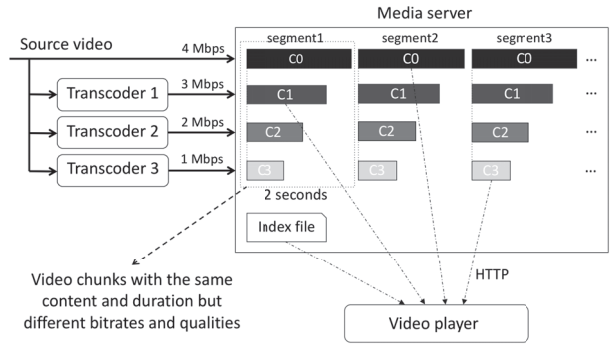


Figure 3: Media server is responsible for (i) splitting video streams into multiple chunks having the same duration (i.e., IDR interval) and (ii) updating the index file according to the status of the transcoded streams.

source video server, runs video transcoding procedures, and delivers transcoded stream to the media server. Like ABR, VideoCoreCluster provides multiple variants of video streams (e.g., different bitrate and resolution) for all TV channels to cope with various link capacity and network dynamics. Users can simply access the web page using their preferred browser and watch TV streams using any device.

Figure 2 shows the architecture of the VideoCoreCluster that consists of a cluster manager and multiple transcoders. The transcoder maintains two connections, MQTT and RTMP [7]. MQTT is used for control flow and RTMP is used for data flow. Both the cluster manager and the transcoder are connected to the MQTT message broker, and the MQTT protocol is used to exchange control signals between them. The transcoder receives the transcoding tasks from the cluster manager via MQTT and sends the transcoded output to the media server through RTMP. Dashboard on the cluster manager provides an interface for administrator to monitor the status of VideoCoreCluster and update/manage transcoding tasks. The following subsections describe each component in detail.

3.2. Design of a Media Server. The media server supports various streaming techniques such as RTMP, HLS, and MPEG-DASH. In our system, HLS and MPEG-DASH are used for the video player (front-end), and RTMP is used in the transcoder (back-end). We employ RTMP to minimize delivery latency because RTMP is designed to transmit real-time multimedia data. We are able to reduce delivery delays of the source video to the transcoders and the transcoded output to the media server. Session control for multimedia application in RTMP is used to implement interactions between the media server and the transcoders.

Figure 3 depicts how the media server works. On all TV channels, VideoCoreCluster transcodes source videos into multiple streams (variants) of different bitrate and quality. In detail, the media server first divides the video stream into a plurality of chunks having the same duration. It is important to align the chunks that have the same video segments but different bitrate and quality because the transcoders may not be perfectly synchronized. To ensure synchronization, we set

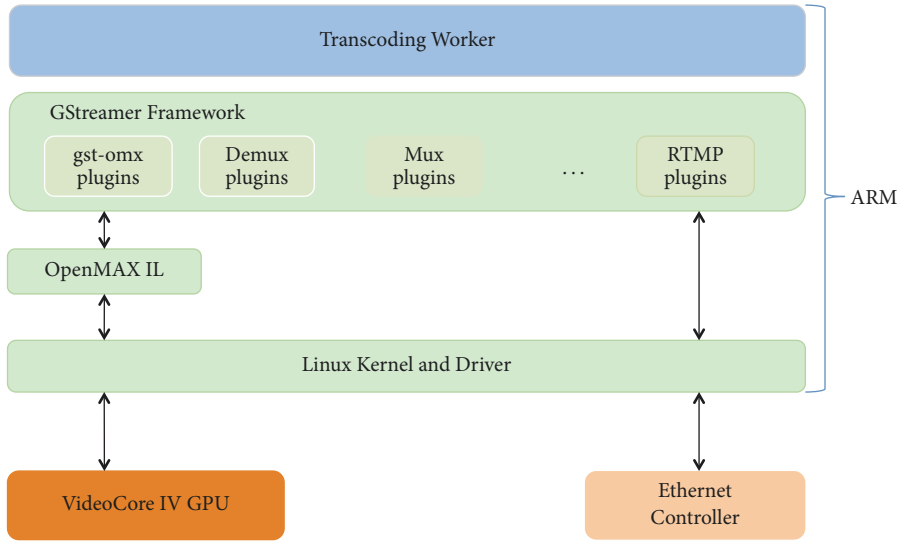


Figure 4: VideoCore IV GPU performs computationally intensive transcoding tasks, while ARM processor is responsible for coordination and data passing.

both the Instantaneous Decoder Refresh (IDR) interval of the source video and the duration of transcoded segments to 2 seconds. These IDR frames are used as boundaries when splitting the streams. The timestamp of the IDR frame (i.e., IDR interval) is used to define the segment number. In this manner, chunks can be synchronized if the transcoder's progress offset for the same video source is less than 2 seconds (i.e., IDR interval). This feature enables bitrate adaptation while switching between the variants without any interruptions. We can also set a larger IDR interval to achieve high tolerance for variations in transcoding speed.

The channel's index file contains information regarding all variations of available streams. The media server instantaneously updates the index file whenever transcoded video streams are available. The client player reads the index file and switches bitrates without pausing or gaps if the video quality improves. The reliability of the video streaming service is guaranteed, because the media server always generates a consistent index file.

3.3. Design of a Transcoder

3.3.1. Overview. The Raspberry Pi (model B) has both a CPU (900 MHz ARM Cortex-A7 quad core) and a powerful GPU (VideoCore IV). Using this architecture, our design strategy is to execute light-weight operations on the CPU and run computationally intensive tasks on the GPU. We ran Debian Linux on Raspberry Pi and reserved 128 MB (out of 512 MB RAM) for the VideoCore (i.e., 128 MB is dedicated for video transcoding process). There are two types of processes running on the transcoder: cluster agent and transcoding worker. There is one cluster agent running on the board, and there are 1~3 transcoding workers (depending on the transcoding task requirements) running on the computing resource.

3.3.2. Cluster Agent. The cluster agent is executed on an ARM processor and has two responsibilities: (i) report the status of the board and transcoding workers to the cluster manager and (ii) manage the transcoding workers by launching/killing transcoding processes. When the cluster agent registers with the cluster manager, it reports the available resource too. The cluster manager assigns the transcoding task to the cluster agent based on the available resources of the transcoder. The cluster agent maintains the MQTT connection by sending a keep-alive packet to the MQTT message broker, if no information flows between the transcoder and the MQTT message broker during the predefined interval. If the cluster agent is disconnected from the broker or the cluster manager is offline, the transcoder stops all transcoding workers. Failed (or aborted) transcoding tasks are reported to the cluster manager; thus, the cluster manager cancels the failed task and reassigns it to another transcoder.

3.3.3. Transcoding Worker. The transcoding worker is a process to transcode an input stream. The transcoding worker only transcodes the video data and simply bypasses the audio data because the size of audio data (126 kbps) is almost negligible compared to the video data (several Mbps). VideoCore IV coprocessor primarily performs the video transcoding task. The ARM processor is responsible for executing the networking protocol stack and demultiplexing and multiplexing the video streams (note that audio data is simply bypassed). In BCM2835, OpenMAX IL interfaces are provided that allows application layer software to access hardware video decoder and encoder [8]. Instead of calling the OpenMAX IL interfaces directly, we built an application using the GStreamer framework [9]. GStreamer is a widely used open-source multimedia framework for building media processing application and has a well-designed plugin (filter) system. For instance, gst-omx is a GStreamer OpenMAX IL wrapper plugin that is used to access hardware video decoder and encoder resources [10]. Figure 4 shows the software

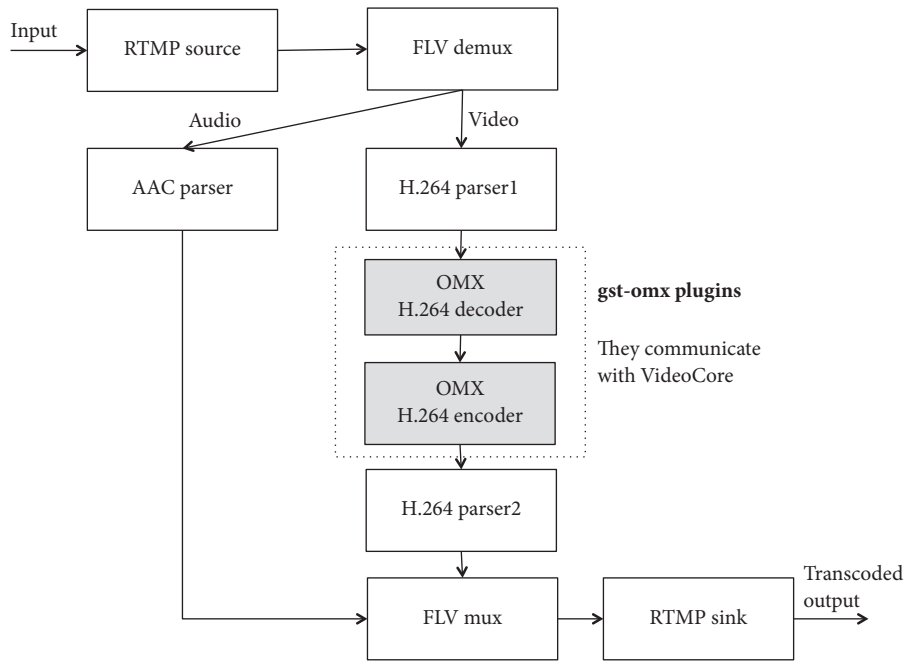


Figure 5: The architecture of GStreamer pipeline. After demuxing, video data is transcoded through pipelines, whereas audio data is bypassed.

architecture of the transcoding worker in our implementation.

3.3.4. GStreamer. The transcoding worker creates a pipeline of the GStreamer plugins as shown in Figure 5. Video data is processed sequentially by the plugins. All plugins are executed on ARM processor except H.264 decoder and H.264 encoder plugins. The default behavior of the GStreamer plugin is summarized in three steps: (i) read data from the source pad, (ii) process data, and (iii) write data to the sink pad. The GStreamer pipeline moves data and signals between connected source and sink pads. The plugins operate independently and process the data sequentially. Both H.264 decoder and encoder send and receive large amounts of data between the ARM processor's memory and the VideoCore IV GPU; however, such data exchange wastes CPU cycles. To address this issue, we have extensively modified gst-omx implementation and have created hardware tunneling between the encoder and decoder to minimize the CPU load. This hardware tunneling is important for real-time video transcoding, because it allows transcoder workers to transcode multiple videos simultaneously (e.g., transcoding one HD and one SD videos). Figure 6 illustrates data flow in a pipeline without hardware tunneling, whereas Figure 7 illustrates it with hardware tunneling.

3.4. Design of a Cluster Manager. The cluster manager keeps both task pool and transcoder pool. The task pool contains a list of transcoding requests and the transcoder pool maintains a list of available transcoders. The primary responsibility of the cluster manager is to assign the transcoding task to the transcoder and to migrate the failed

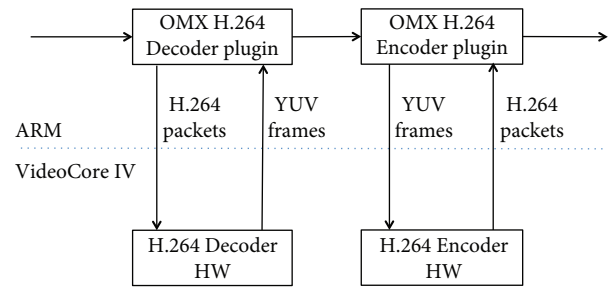


Figure 6: The decoder and encoder plugins work independently. Decoder plugin moves the YUV frames from VideoCore IV memory to ARM memory; then encoder plugin moves them back to VideoCore IV memory.

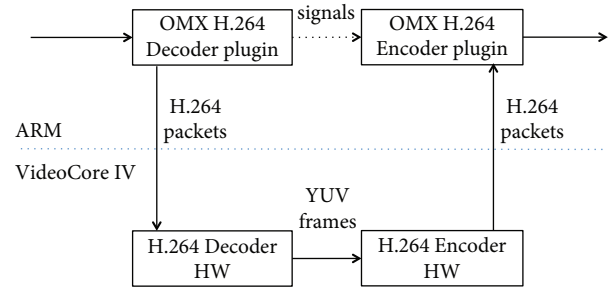


Figure 7: We have created hardware tunnel between H.264 decoder and H.264 encoder to reduce CPU load on ARM processor. Both the decoder and the encoder plugins are not required to process YUV frames.

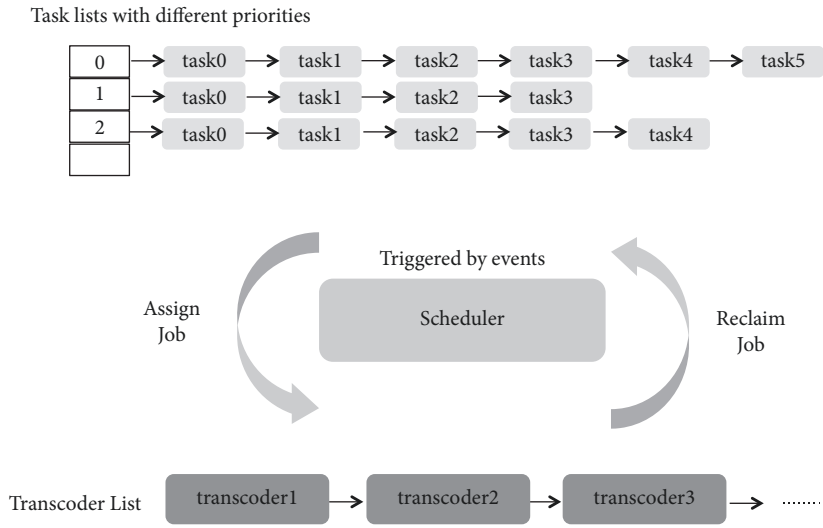


Figure 8: The scheduler assigns/reclaims tasks based on both the priorities and the available resources. The scheduler works in an event-driven manner.

task from one transcoder to another transcoder. When scheduling the transcoding task, the type and priority of the task are taken into account. The cluster manager maintains a list of tasks with different priorities. The transcoding task includes the following information; {*ID*, *state*, *channel and bitrate*, *command*, *resource*, *priority*}. We have *resource* as integer value that indicates the amount of computation resources (determined by type and bitrate, e.g., SD or HD) required to execute the task. Task can have the following four states: {*idle*, *assigning*, *revoking*, *and running*}. We adopt an event-driven design for the cluster manager. For example, when an event occurs (task fails, a new worker is created, a worker completes, etc.), the cluster manager checks if a rescheduling is needed.

When registering a new worker with the cluster manager, the cluster manager adds records to the transcoder list and tracks the transcoder's state at predefined intervals. Transcoder periodically sends both status information (including CPU load, VideoCore IV load, and available memory) and running tasks to the cluster manager. We use MQTT *last will message* to track the transcoder's status in real-time. The cluster manager provides a dashboard that can monitor the status of the cluster and make configuration changes. When designing the cluster manager, we focus on three goals.

(i) Reliability. Real-time monitoring of the transcoders status can ensure the reliability of the system. We have combined the monitoring system with low latency scheduling that can migrate failed tasks to other transcoders. The media server updates the index file in real-time to keep the list of transcoded streams consistent. Even if the update of the index file temporarily fails, the user can still watch the video stream because it only hinders bitrate switching

(ii) Scalability. Event-driven design and minimization of control flow ensure scalability of our system. In order to minimize the overhead of control flow, the cluster manager and transcoder exchange only critical information (e.g., transcoder's status). In addition, separation of data and control flows ensures scalability of the cluster manager. The media server hosts both the source (original) and the transcoded videos and they can be easily replicated when the workload is above the threshold.

(iii) Elasticity. To support adaptive bitrate streaming, every channel keeps multiple versions of encoded video at various bitrate. The more variations of available video with different bitrate, the better the video quality the system can provide to users. VideoClusterCore uses this feature and task priority to provide elastic video transcoding services. If the resource is not sufficient for all requested transcoding tasks, the higher-priority tasks are scheduled. Lower-priority tasks have to wait for resources to be granted after completing a higher-priority task. In addition, we expand the transcoder cluster by easily adding more transcoders to this elastic design and support more TV channels.

3.5. Transcoding Scheduler. Figure 8 shows how the scheduler works (in the cluster manager). According to the capacity of the transcoder and the resource requirements of the task, the scheduler determines the transcoding schedule. The capacity and resource requirement are expressed as an integer representing the type and maximum number of tasks that can be executed by the transcoder. The examples of the transcoding tasks (including ID, required resource, and priority) are shown in Table 1. If the capacity of the VideoCore IV is 20 (in terms of resource), it can execute 1 HD task (e.g., ID 4 or 5), or 2 SD tasks (ID 1 and 2), or 3 SD tasks (ID 2, 3, and 7), or 1 HD task and 1 SD task (ID 3 and 6). When a new

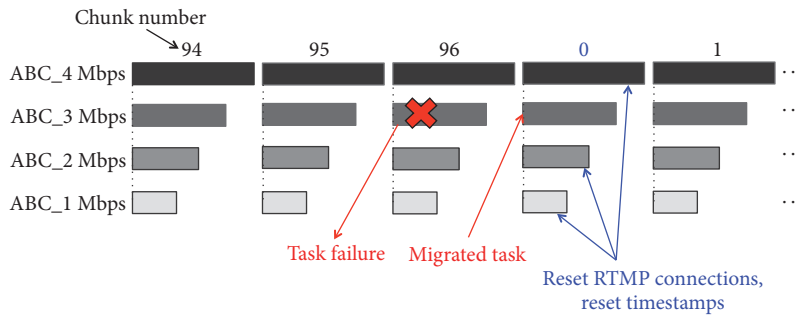


Figure 9: After migrating task, transcoder resets RTMP and timestamp.

Table 1: Examples of the transcoding tasks.

ID	Channel	Resource	Priority
1	A	10	0
2	A	8	1
3	A	6	0
4	B	20	0
5	B	16	1
6	B	14	0
7	C	6	0

transcoder is registered with the cluster manager, the highest priority idle or unassigned task is immediately scheduled. When the task fails and is returned to the cluster manager, the cluster manager immediately (within a second) migrates the failed task to another transcoder with sufficient capacity. If the cluster manager fails to find an appropriate transcoder, it cancels the lower-priority running task and attempts to assign the failed task to this transcoder. When the cluster manager cannot find an executing task with a lower-priority than the failed task, it waits until the task execution is completed.

It is required to reset the RTMP connections when migrating task from one transcoder to another. RTMP specification requires that the stream's initial timestamp be zero [7]. As depicted in Figure 9, cluster manager sends messages to the transcoders to reset their RTMP connections of the failed streams to ensure all other RTMP connections have the same timestamp; therefore, transcoded streams with different bitrate are synchronized.

4. Implementation and Deployment

Implementation. We utilize several open-source frameworks to implement our video transcoding system. We have extensively modified the vanilla design of these frameworks. In particular, Nginx, Apache, GStreamer, mqtt.js, node.js and Express, Google protobuf, and paho MQTT client library are used in our implementation [11–13]. Both the video transcoders and cluster manager run on Linux system. Mosquitto [14] is used as MQTT message broker to enable communication between components. VideoCoreCluster implementation involves the following three components.

(i) Media Server. Apache is used to build the media server and both Nginx and Nginx_RTMP_module are executed on the media server [15]. As described, the media server maintains HTTP and RTMP flows that are implemented by in Apache and Nginx, respectively.

(ii) Cluster Manager. Node.js Express is used to build the cluster manager. The cluster manager requires the MQTT client module for subscribing and publishing messages with the transcoder. We implemented mqtt.js for the message exchange. Both the cluster manager and the media server are installed on dedicated server running Ubuntu 14.04 LTS.

(iii) Transcoder. Video transcoder consists of the transcoder worker and the cluster agent. Both components are written in C/C++ and rely heavily on our modifications of GStreamer, paho MQTT client library, and Google protobuf. Specifically, we first used buildroot [16] to build SDK, the root disk, and the Linux kernel for Raspberry Pi and then created the above-mentioned two components in the customized SDK. All components are connected via Gigabit ethernet; therefore, there is no bottleneck in communication and data exchanges.

Deployment. We adopt a hybrid approach for deployment. VideoCoreCluster is deployed in an incremental manner. We first deploy all core components and then increase the number of Raspberry Pis (transcoding workers) to support all channels. The deployment consists of a small-scale VideoCoreCluster (including eight Raspberry Pis) and a transcoder cluster with five Intel Xeon servers. Specifically, our system provides a real-time transcoding service along with the certain IP-based TV channels. We are in the process of extending the size of VideoCoreCluster to handle a large number of IP-based TV channels (e.g., over 100) with various video bitrates.

5. Evaluations

5.1. Video Quality

5.1.1. Metrics. In this subsection, we evaluate the performance of the encoder in terms of video quality. It is critical to maintain high video quality to ensure the performance of the video transcoder. In general, subjective and objective metrics are used to evaluate video quality. Subjective video

quality is quantified by human experience and cannot be repeated; therefore, the results may vary depending on test conditions and the equipment [17, 18]. Unlike subjective manner, objective metrics have been widely adopted to access the performance of video encoders because they are repeatable and easy to handle. Specifically, PSNR (Peak Signal to Noise Ratio) and SSIM index (Structural Similarity) are used to evaluate the video qualities of the encoder. PSNR measures the visibility of *errors* introduced by lossy video coding. The authors in [19] showed that PSNR is a valid quality measure when the video contents and the codec types have not changed. SSIM is a framework for accessing video quality based on degradation of structural information [20]. In other words, SSIM algorithm assesses the *similarity* of two video images (max. SSIM is 1.0). We used both PSNR and SSIM to quantify the performance of the video encoders since their objectives are different (error versus similarity).

5.1.2. Comparison. VideoCore IV has a powerful H.264 decoder that provides real-time decoding of H.264 HD video (e.g., resolution of 1920×1080 and level 4.0) while consuming little power. Besides the performance, it retains the same quality decoding results compared to other reference H.264 decoders; therefore, quality loss in our transcoding system occurs only in the encoding part. Simplifying the hardware design is also important for the hardware vendors. Despite the quality loss, the vendors apply several optimizations to the hardware design. For instance, the video encoders on SoCs in mobile devices degrade video quality due to strict limits on power consumption. We have conducted several performance measurements on VideoCore IV encoder to quantify video quality against the performance of x264. The x264 has various parameters to configure for encoding. It is hard to achieve the optimal configuration because the parameters are highly correlated with each other and have impact on the encoding performance. Instead of setting various parameters, we optimized the parameters related to video encoding speed and video quality using presets provided by the x264 developer. There are 10 presets provided by the x264 in descending order of speed and in ascending order of video quality: "ultrafast", "superfast", "veryfast", "faster", "fast", "medium", "slow", "slower", "veryslow", and "placebo". The "medium" is used for default preset. A slower preset will provide better compression (compression is quality per file size). As video quality increases, the encoding speed decreases exponentially. For instance, it takes about 29.24 seconds to encode video (500 frames, 1280×720, 50 fps, YUV 4:2:0) in a "fast" preset and 14.66 seconds for a "veryfast" preset (slower preset takes almost double). With the same video source, "veryslow" requires about 140 times more than a "ultrafast" preset. Despite the high quality, slow configurations are inadequate for transcoding live streams because of the rigid timing requirements for real-time playback.

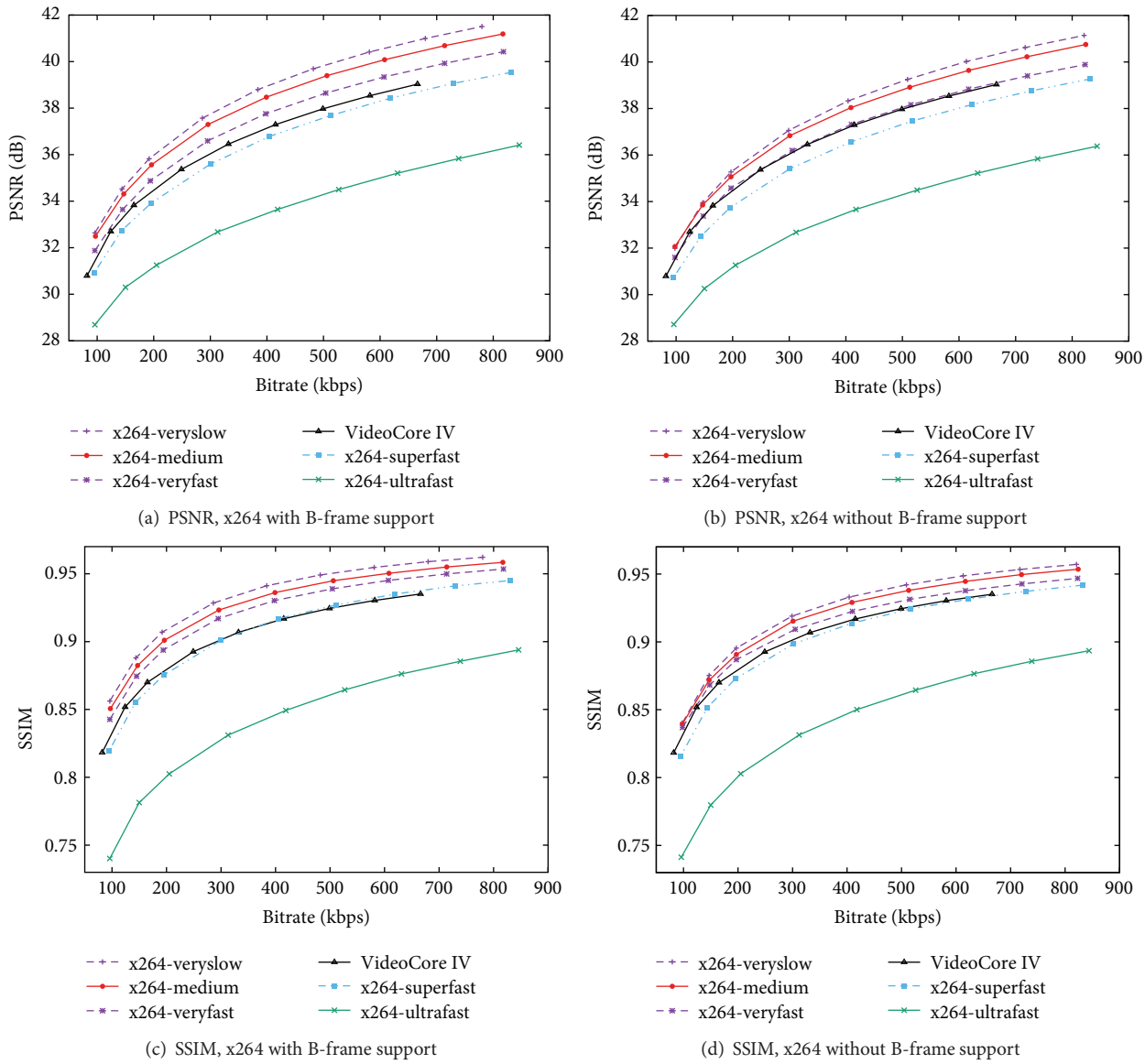
5.1.3. Test-Set. Two sets of video sequences are used to evaluate the performance of the encoder. First, we use the YUV sequence [21] commonly used in video coding studies; thus, the results are reproducible and easy to compare with other works. Second, we use customized video sequence obtained from our deployment because it reflects the encoder performance in practice. We have obtained the second test-set from live broadcast streaming such as CNN, NBC, and ESPN. They are good test cases for evaluating real-time transcoding system because live streaming service requires low delay (latency). For x264 implementation, we have included several libraries, drivers, and firmware for Raspberry Pi [22, 23] based on the publicly available x264 [24]. H.264 encoder in VideoCore IV does not support B frames because B frames cause high encoding and decoding latency. For fair comparison, we have tested both cases (x264 with B frames and x264 without Bframes) to quantify the effect of B frames on video quality while varying the video bitrate.

5.1.4. Results. We present the video quality result of *foreman_cif* in Figure 10 (we have obtained similar results for all other video sequences and omitted them for the sake of brevity). We confirm that VideoCore IV provides a better performance in terms of video quality (PSNR and SSIM) comparing to the x264 with "superfast" preset (superfast is sufficient for handling live streaming). In addition, we can see that the impact of B frames on video quality is almost negligible; PSNR difference is less than 0.5 dB and SSIM difference is less than 0.02. Regardless of the presence of B-frame, the higher video bitrates yield the higher PSNR and SSIM. Note that the tendencies in PSNR and SSIM results are very similar; however, their meaning is different. A high PSNR means fewer errors introduced to the video, whereas high SSIM indicates the similarity between original and encoded video. Our transcoding solution does not incur errors nor distortion; therefore, both PSNR and SSIM are relatively high.

Note that the second test-set is intended to evaluate the performance of the video encoders in a real deployment; therefore, we only use several video bitrates (0.4, 0.6, and 0.8 Mbps for SD and 0.8, 1.6 and 2.4 Mbps for HD) for evaluation. Figures 11(a) and 11(c) show that the VideoCoreIV's performance on low video bitrate (0.4 Mbps) is lower than that of x264. We have found that VideoCore's video encoder itself has a bug implementation, and hence this leads to low video quality if the target bitrate is low. However, we believe this is not an issue for our deployment because the client's video player switches to such low bitrate only when the network condition is seriously bad. Given the high capacity of our network configuration, we do not expect that would be a common case in our deployment. As shown in Figures 11(b) and 11(d), the video quality of VideoCore is high enough in case of high bitrate settings (bitrates > 0.6 Mbps). It provides a high quality that is very close to x264 with "medium" preset. We have also confirmed that users are satisfied with VideoCore's quality in a subjective manner.

5.2. Transcoding Speed

5.2.1. Test-Set. Transcoding speed is critical for handling live streams. To record the time taken for transcoding procedure, we first captured two video streams, one SD and one HD channels, in an offline manner. Then, we measured the



(a) PSNR, x264 with B-frame support

(b) PSNR, x264 without B-frame support

(c) SSIM, x264 with B-frame support

(d) SSIM, x264 without B-frame support

Figure 10: Foreman YUV (352 × 288, 25 fps) is used to evaluate the performance of the encoder (IDR is to 2 seconds). VideoCore IV yields a better (or similar) performance comparing to the x264 with superfast.

transcoder's speed performance under a series of stress tests. There is some overhead for both demuxing and muxing in the process; however such overhead is negligible. This is because of the fact that video transcoding (both the encoding and decoding) requires high complexity; therefore, the bottleneck of the whole process is in video transcoding itself. The detailed specifications of the test streams are (i) SD channel with 720 × 480, 30 fps, H.264 high profile, level 4.0, bitrate 1.2 Mbps, and (ii) HD channel with 1280 × 720, 30 fps, H.264 high profile, level 4.0, bitrate 4 Mbps. We transcoded both the SD and HD videos to 0.8 Mbps and 2.4 Mbps, respectively, and kept other parameters the same. For other SD channels with lower resolutions in our deployment (e.g., lower than 720 × 480), the transcoding speed is faster than the results shown here.

5.2.2. Results. We measured the speed of the software transcoder as the number of frames generated in a second (i.e., fps). The FFmpeg framework with built-in H.264 video decoder is used in software video transcoder implementations. Specifically, FFmpeg (linked with libx264) executes H.264 video encoding procedure on a Linux desktop (Intel Core i5-4570 CPU@3.20 GHz and 16 GB RAM). We have created FFmpeg and libx264 with all the CPU capabilities to accelerate the video transcoding process. "Superfast" and "medium" (default) presets are used for evaluation (note that "superfast" provides video quality similar to that of VideoCore IV as shown in Figure 10). In Figure 12, we confirm that if the output video quality is similar, software transcoder running on Intel i5 CPUs is about 5.5x and 4x faster than Raspberry Pi's video transcoder for SD and

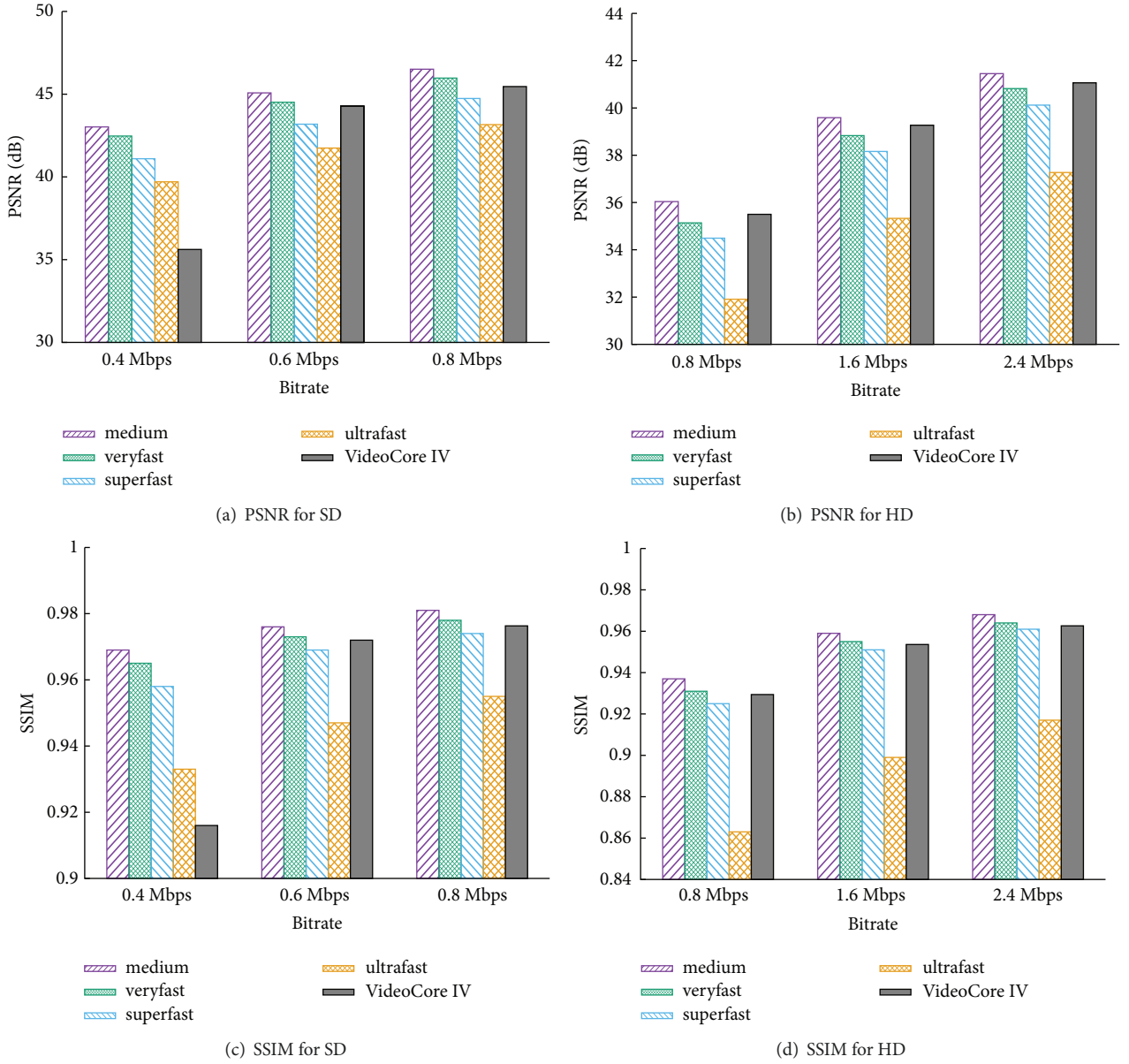


Figure 11: Video quality results of VideoCore IV and x264 with various presets.

HD video, respectively. Transcoding procedure requires high computational resources; thus, the superiority of transcoder on a powerful desktop is naturally expected. In other words, we can run 5.5x (for SD video) or 4x (for HD video) transcoding tasks by replacing Raspberry Pi to Intel i5 desktop in our transcoding system. Note that 30 fps is sufficient for transcoding live streams in real-time. In this sense, transcoding speed of VideoCore IV (e.g., 120 fps and 50 fps for SD and HD video, respectively) is well above the requirement (30 fps) for live stream. This experiment results validate that our VideoCore IV implementation on Raspberry Pi is powerful and cost-effective comparing with a desktop-based transcoding.

5.3. Energy Consumption. We have confirmed that our hardware-assisted transcoder on Raspberry Pi is good enough for video transcoding live streams in real-time in terms of video quality and transcoding speed. To monitor and evaluate the energy efficiency, we have measured the power consumption of both the Raspberry Pi and a dedicated transcoding server (Intel i5 CPU with 16 GB RAM). We have executed the same video transcoding task (HD 1280 × 720) on both platforms while logging the instantaneous power measurements. We present one typical power measurement result in Figure 13 (we have conducted multiple runs and obtained the same results). We can see that the desktop implementation of transcoding server consumes more than 35 times power comparing to that of Raspberry Pi while running transcoding task.

From Figure 13, we notice that Intel-based transcoding server finishes the same transcoding task much faster (6.1x) than the Raspberry Pi which corroborates our finding from

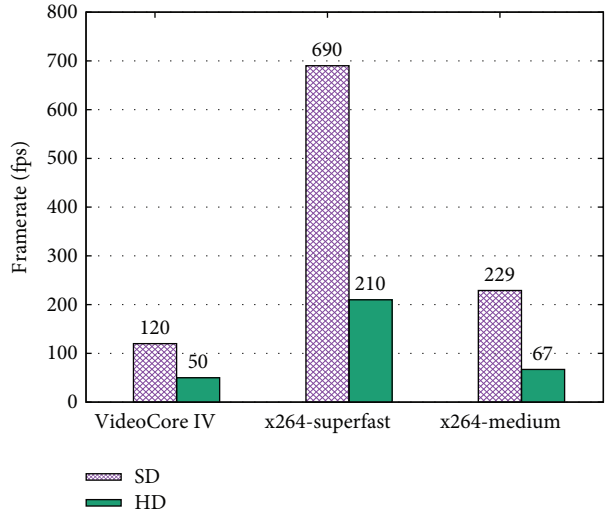
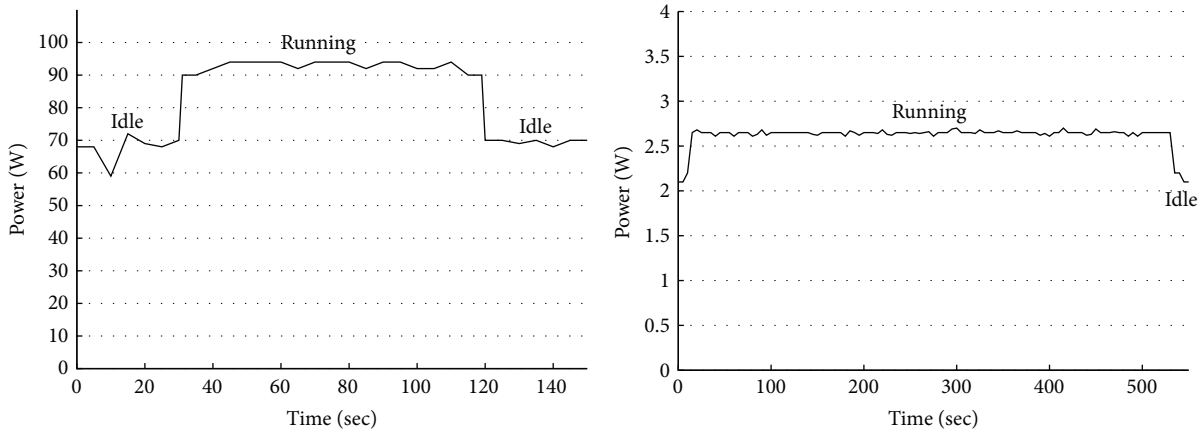


Figure 12: The performance of transcoding on a dedicated, powerful desktop outperforms that of hardware-assisted VideoCore IV; however, VideoCore IV is still suitable for transcoding in real-time.



(a) Desktop transcoding server (Intel i5 CPU)

(b) Raspberry Pi Model B (VideoCore IV)

Figure 13: VideoCoreCluster significantly reduces power consumption.

the transcoding speed evaluation. The areas in Figure 13 represent the total power consumption of each transcoder implementation. Table 2 summarizes the power measurements for both platforms. We can see the VideoCoreCluster is more energy efficient (i.e., 5.76x less energy) than a transcoder made with a general-purpose processor when they provide the same transcoding capacity. Because the same network switches and configuration apply to both cases, we have omitted their power consumption.

Table 2: Energy consumption comparison.

	Intel server	Raspberry Pi
Average power	92.94 W	2.65 W
Duration	85 sec	519 sec
Energy consumption	7.90 kJ	1.37 kJ

6. Related Work

Adaptive Bitrate Streaming (ABS). Many video service providers use ABS technology to provide video streaming services with high quality. Various video players, such as Smooth streaming and Adobe HTTP Dynamic Streaming, are widely used for ABS; however, their performance problems are reported by several measurement studies [25, 26]. They show that none of these players is good enough; they are either too conservative or too aggressive, and hence they do not efficiently adapt to the network variations. Even worse, adaptation cannot address unfairness in bandwidth sharing, and hence clients suffer low data rate and stall (e.g., <1 Mbps, video starts 60 sec later). Designing both robust and reliable bitrate adaptation algorithm in dynamic networks is difficult. Various bitrate adaptation algorithms have been proposed in several works, with an emphasis on achieving better video quality. For instance, Festive [27] outperforms existing solutions in terms of fairness by >40%, stability by >50%, and efficiency by >10%. BBA in [28] reduces the

rebuffer rate by 10–20% compared to Netflix's ABR algorithm. While the above-mentioned works focus on the client-side implementation, our design focuses on the server side.

Video Transcoding. Video transcoding plays an important role in video streaming services; thus, various transcoding approaches and architectures have been proposed. For instance, several techniques have been proposed using information extracted from video bitstreams to reduce complexity and improve video quality [29] (improve PSNR by ~2 dB). Vetro et al. [30] discussed the block-based video coding scheme using hybrid discrete cosine transform (DCT) and motion compensation (MC). Unlike their claims, cascaded decoder and encoder are much easier and more flexible. A cascaded pixel-domain approach is more appropriate for practical deployment, while hardware video encoder improves video quality and reduces cost. In a real deployment, [31] showed that a cascaded video transcoder is more efficient for point-to-multipoint transcoding because some part of video transcoder can be reused. To optimize video streaming services for mobile devices, cloud-based transcoding has been introduced [32]. Video transcoding on a cloud platform is a good solution for transcoding large amounts of video data because of its high-throughput performance. Several commercial companies, e.g., Microsoft, Amazon, and Telestream Cloud, provide cloud-based transcoding services to users [33].

Specialized Hardware for Video Applications. Research efforts have been made to address the efficiency problems of general-purpose processors (including video decoding and encoding) in multimedia application. Various approaches have been explored to improve hardware efficiency, including specialized instructions [34], customized architectures [35], GPU offloading [36], application specific integrated circuit (ASIC) [37], and FPGA-based accelerators [38]. Computer clusters are a well-known configuration of distributed systems used to provide high-throughput computing. For instance, Condor is a distributed system for scientific applications [39]. Our system is unique in several ways that the target application is real-time computing, and the computing nodes consist of specialized, low-cost ($35), and energy-efficiency (<3 W) hardware. FAWN [40] is a cluster built with low-power embedded CPUs (10–20 W) but is used only for data-intensive computing purposes. In contrast, our transcoding system is suitable for both data-intensive and computationally intensive tasks.

7. Conclusion

Video streaming is one of the killer applications in recent years. The performance of video transcoding is critical to ensuring high quality video streaming services. Toward this, we designed and implemented VideoCoreCluster, a low-cost, high-efficiency video transcoder system that supports live streaming. A commodity, cheap ($35) but powerful and energy efficient (consumes <3 W), Raspberry Pi is used to build real-time video transcoding system. We implemented the cluster utilizing both IoT and RTMP protocols for control and data paths, respectively. Such separation design guarantees low latency in video transcoding and data deliveries to multiple components. VideoCoreCluster built with Raspberry Pi is more energy efficient than a general-purpose processor without sacrificing video quality, reliability, and scalability.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the research fund of Hanyang University (HY-2016-N).

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2014-2019," White Paper, FLGD 12352 06/16, May 2015.

[2] T. Stockhammer, "Dynamic adaptive streaming over HTTP: standards and design principles," in *Proceedings of the Proceeding of the 2nd Annual ACM Multimedia Systems Conference (MMSys '11)*, pp. 133–144, New York, NY, USA, February 2011.

[3] "Eighth MPEG-4 AVC/H.264 Video Codecs Comparison - Standard Version," http://www.compression.ru/video/codec_comparison/h264_2012/.

[4] "x264," http://www.videolan.org/developers/x264.html/.x264.

[5] "HTTP Live Streaming (HLS), "IETF Internet-Drafts 2014"," http://tools.ietf.org/html/.

[6] ISO/IEC23009-1., MPEG Dynamic Adaptive Streaming over HTTP (DASH), http://dashif.org/mpeg-dash/.

[7] *Real-Time Messaging Protocol (RTMP)*, http://www.adobe.com/devnet/rtmp.html.specification.

[8] "OpenMAX Integration Layer Application Programming Interface Specification," https://www.khronos.org/registry/omxil/specs/OpenMAX_IL_1_1_2_Specification.pdf.

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

[10] "GStreamer OpenMAX IL wrapper plugin," https://github.com/pliu6/gst-omx.

[11] "Nginx," https://www.nginx.com/.

[12] "Node.js," https://nodejs.org/en/.

[13] "MQTT.js," https://github.com/mqttjs.

[14] "Mosquitto - An Open Source MQTT v3.1/v3.1.1 Broker," http://mosquitto.org/.

[15] "nginx-rtmp-module," https://github.com/pliu6/nginx-rtmp-module.

[16] "Buildroot - Making Embedded Linux Easy," https://buildroot.org/.

[17] "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-IPDF-E.pdf.

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

[19] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *IEEE Electronics Letters*, vol. 44, no. 13, pp. 800-801, 2008.

[20] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[21] "YUV Video Sequences," http://trace.eas.asu.edu/yuv/.

[22] "Raspberry Pi Firmware," https://github.com/raspberrypi/firmware.

[23] Source code for ARM side libraries for interfacing to Raspberry Pi GPU, https://github.com/raspberrypi/userland.

[24] "x264-snapshot-20150917-2245," http://mirror.yandex.ru/mirrors/ftp.videolan.org/x264/snapshots/.

[25] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proceedings of the 2nd Annual ACM Multimedia Systems Conference (MMSys '11)*, pp. 157–168, February 2011.

[26] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proceedings of the 4th Workshop on Mobile Video (MoVid '12)*, pp. 37–42, February 2012.

[27] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proceedings of the 8th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '12)*, pp. 97–108, ACM Press, New York, NY, USA, December 2012, http://dl.acm.org/citation.cfm?doid=2413176.2413189.

[28] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication, SIGCOMM '14*, pp. 187–198, USA, August 2014.

[29] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video transcoding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 84–97, 2005.

[30] A. Vetro, C. Christopoulos, and H. Sun, "Video transcoding architectures and techniques: An overview," *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 18–29, 2003.

[31] J. Youn, K. N. Ngan, T. Sikora et al., "Video transcoding for multiple clients," in *Proceedings of the Visual Communications and Image Processing 2000*, Perth, Australia.

[32] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai, "Cloud transcoder: Bridging the format and resolution gap between Internet videos and mobile devices," in *Proceedings of the 22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '12*, pp. 33–38, Canada, June 2012.

[33] *High Quality Video Transcoding in the Cloud*, https://cloud.telestream.net/.

[34] M.-A. Daigneault, J. M. P. Langlois, and J. P. David, "Application specific instruction set processor specialized for block motion estimation," in *Proceedings of the 26th IEEE International Conference on Computer Design, ICCD '08*, pp. 266–271, USA, October 2008.

[35] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime anywhere anyway signal processing," in *Proceedings of the ISCA 36th Annual International Symposium on Computer Architecture*, pp. 128–139, USA, June 2009.

[36] W.-N. Chen and H.-M. Hang, "H.264/AVC motion estimation implmentation on compute unified device architecture (CUDA)," in *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME '08*, pp. 697–700, Hanover, Germany, June 2008.

[37] Y. Lin, D. Li, C. Lin et al., "A 242mW 10mm^2 1080p H.264/AVC High-Profile Encoder Chip," in *Proceedings of the 2008 International Solid-State Circuits Conference - (ISSCC)*, pp. 314–615, San Francisco, CA, USA, Feburary 2008.

[38] O. Lehtoranta, E. Salminen, A. Kulmala, M. Hännikäinen, and T. D. Hämäläinen, "A parallel MPEG-4 encoder for FPGA based multiprocessor SOC," in *Proceedings of the International Conference on Field Programmable Logic and Applications, FPL*, pp. 380–385, Tampere, Finland, August 2005.

[39] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2–4, pp. 323–356, 2005.

[40] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A fast array of wimpy nodes," in *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles, SOSP '09*, pp. 1–14, USA, October 2009.