120th ASEE Annual Conference & Exposition
FRANKLY, WE DO GIVE A D*MN
June 23-26, 2013

Paper ID #6792

Development of a Multi-Platform High Performance Computing Streaming Video Distribution Cluster

Prof. Carlos R Morales, Purdue University, West Lafayette
Mr. Perry Lucas Cox
Mr. Matthew John Farrenkopf
Mr. Robert Eric Knorr, Purdue University

I studied at Purdue University to receive a Bachelor's of Science in Computer Graphic Technology. I specialized in production and practiced in production management.

Erick Morales
Mr. Christopher Gaeta
Mr. Martin Jerome Durchholz

Development of a Multi-Platform High Performance Computing Streaming Video Distribution Cluster

A. introduction

The Purdue University College of Technology Distance Learning Center (CoT DLC) developed a multiplatform (PC, iPhone, Android) streaming video platform to distribute distance-learning content using a variety of HPC techniques. The developed system is capable of dynamically scaling and transcoding content based on server demand, available bandwidth, format of source material, and client format restrictions.

The CoT DLC needed a mechanism for distributing distance-learning course content to students distributed through out the globe. The variety of connection speeds, latency, and Internet access restriction imposed by some governments increased the technical challenges that would need to be met by our solution.

Prior to creating a custom solution, the group evaluated a wide range of commercial and open-source streaming platforms using a robust set of objective criteria. The most important criteria were identified as: (1) ability to stream content to any common client (PCs/Macs, Tablets, mobile phones), (2) ability to encode / transcode content into any necessary format in real-time or faster, (3) ability to adequately handle adverse conditions such as network congestion, and (4) ability to handle a large number of simultaneous client requests.

B. system overview

To accomplish encoding, the team erected a computing farm with a cluster of nodes dedicated to real-time video encoding. At the core of the encoding process were dedicated hardware h.264 encoders that accepted our videos as 1080p via HDMI and/or SDI and output h.264 files ranging in data-rate from 2 to 20 Mbps. Lower bitrate files were created with off-the-shelf encoding software capable of utilizing NVidia Quadro and Tesla GPUs on the servers.

The system used a hardware based uncompressed 1080p video-switcher to manipulate the instructor's video. The output video signal was then compressed using a series of hardware based H.264 encoders and servers equipped with Nvidia Tesla GPUs to create the wide variety of video streams required for the student's clients. The streams were then delivered via HTTP and RTMP from multiple servers. See figure 1 for an overview.

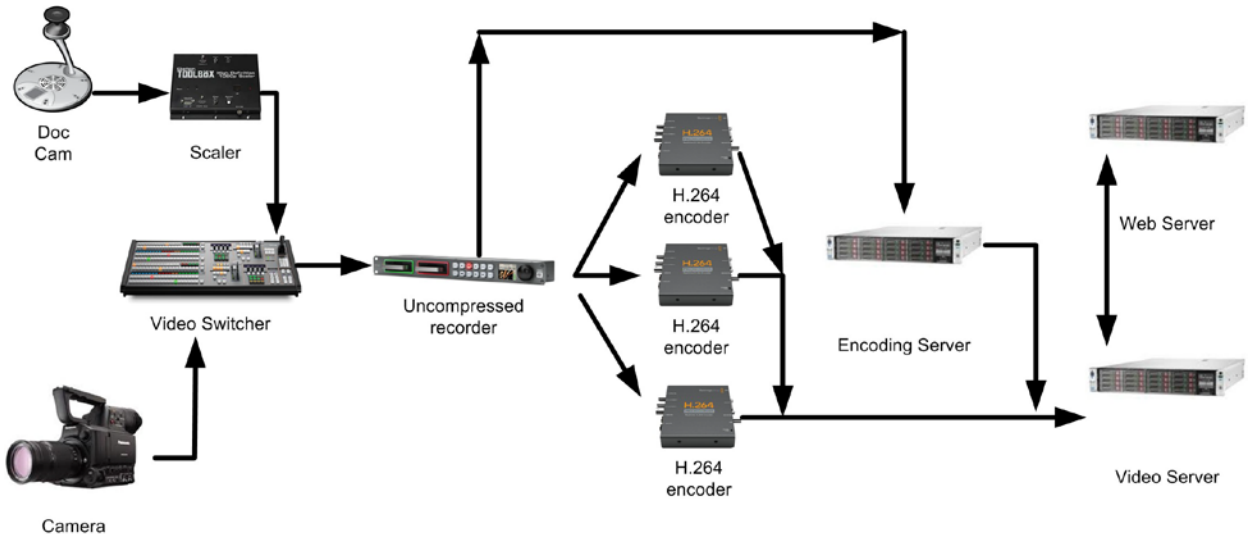

Figure 1. System overview

On the production end of the pipeline, the team used a Blackmagic ATEM/2 switcher connected to a document camera and a Panasonic AG-AF100 via HD-SDI at 1080p. This set up made a tremendous impact on both the visual quality of the videos and increased the efficiency during the encoding and streaming phase of the project. The resolution of the videos (1920 x 1080) provided ample resolution for showing details.

To maximize the quality of the encoding, the team insured the video produced at this stage would be uncompressed and sampled at 4:2:2. This would result in better encoding than if the video was sampled at 4:1:1 or if some level of compression was applied by the camera before getting incorporated into the video switcher. To accomplish this, the team elected to use the AF100 HD-SDI output to feed the switcher rather than capturing to the built in SD Card, which would have resulted in the camera applying AVCHD compression. While AVCHD compression is very good, it does not compress as well as

Additional steps incorporated by the team to improve quality at this stage, included using an external scaler for the document camera and using 10-bit color space within the swithcher. By using an external scaler to process the video from the document camera into 1080p from the document camera's native resolution of 1080i, the team was able to get a better quality signal from the document camera. The choice to use 10-bit colorspace increased the quality of the video signal.

The video switcher accepted the video from the AF100 and document camera. It allows the team to use traditional television production techniques, such as adding lower thirds, chroma keying, or adding transitions. Out of this process, the team was able to create a very high quality video (see figure 2)

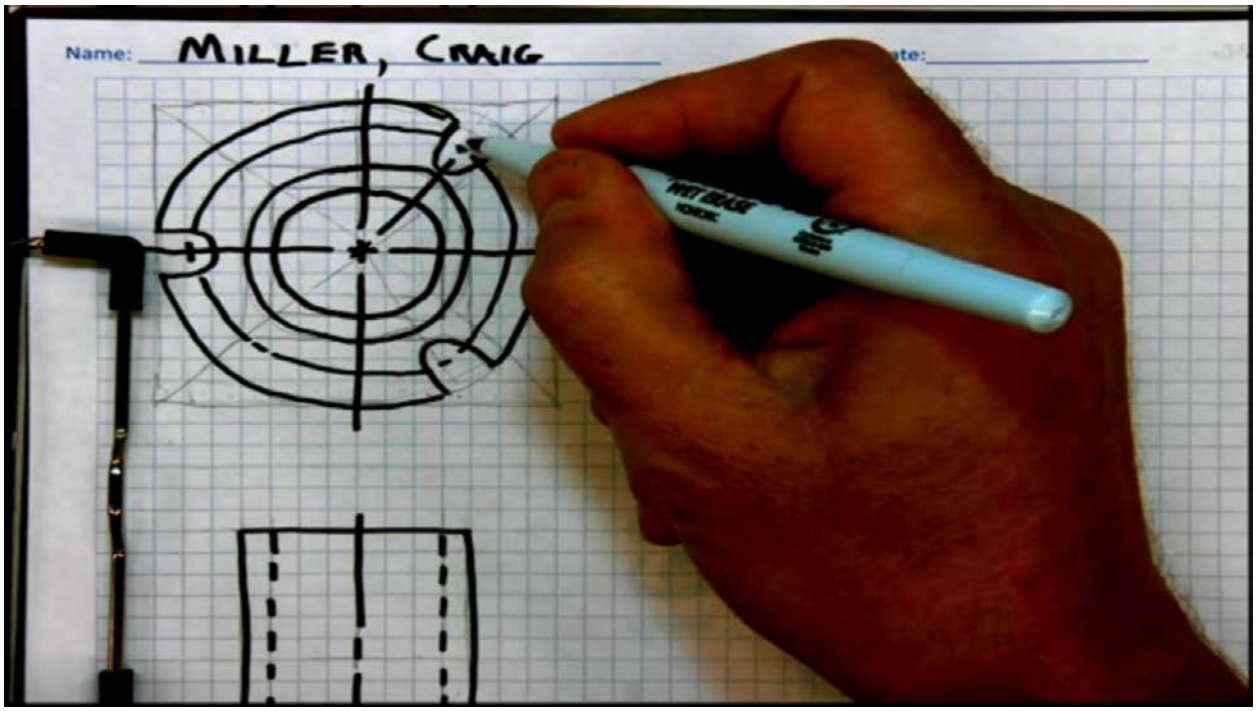

Figure 2. sample lecture content

In technical terms, the video switcher was set to output an uncompressed 1080p 10-bit stream at 29.97 fps over HD-SDI, which generated approximately 781 GB per hour of video. This was stored on a BlackMagic HyperDeck Studio. This device accepts a pair of SSDs and records uncompressed video from HDMI or HD-SDI. The device alternates writing data between the two SSDs, which can be hot swapped to enable for unlimited capture. The team used 400 GB Vertex 3 SSDs, which captured approximately 25 minutes per SSD.

The uncompressed files generated by this process were great in terms of quality, but contained too much data to be streamed to the students. A cluster of H.264 hardware encoders was attached to the uncompressed HD-SDI video coming from the HyperDeck Studio. Each encoder was set to generate an h.264 file at a variety of different bit rates ranging from 2 megabits per second to 20 megabits per second. This set up enabled us to create distance-learning content at a variety of data-rates in real-time.

While the h.264 hardware encoders provide us the ability to create high-bandwidth content very quickly, this solution was not sufficient because the encoders did not produce video below 2 mbps. Thus, even the lowest quality video stream produced by the encoders was too high to be distributed to everyone in the course. This solution also had the limitation of only producing content in H.264, which is widely used but does not work with 100% of the devices that team wanted to target.

C. encoding and compression

To expand the reach of the videos files, the team erected a set of encoding servers. Each server contained 2 XEON E5 processors, 96 GB of RAM, and a CUDA enabled graphics card. A combination of NVIDIA Tesla, Quadro 4000, and Quadro 6000 cards were used.

On the software end, the team used a combination of Adobe Media Encoder CS6 and Microsoft Expression Encoder due to their support for GPU accelerated video encoding and also their ability to produce i-frame aligned video files. Other encoding solutions evaluated by the team, such as ffmpeg, did not support both GPU encoding and also the production of i-frame aligned video streams.

In our scenario, GPU acceleration was absolutely critical due to the large number of videos produced on a daily basis by the team. Without GPU acceleration it typically took 15 hours to encode 1 hour of video in all of different bandwidths and formats required by our server. With our GPU accelerated pipeline, the same task took approximately 1.5 hours.

The i-frame aligned files enabled us to dynamically switch among different quality enocodes of the same video files seamlessly. As bandwidth increased or decreased for a student during an on-line session, the server is able to select the best quality stream that will play over the student's connection and dynamically switch the stream of video. The end result is a viewing experience that upgrades and/or downgrades video for the student based on the connection conditions.

By using the H.264 hardware encoders and the encoding servers, the team was able to create versions of the video at a variety of bitrates ranging from 500 Kb/sec to 20 Mb/sec in a variety of formats including h.264, flv/f4v, Theora, and WebM. This created the files necessary for the video server to deliver the content to the students.

D. deployment

The encoded files were then deployed in a variety of different formats including YouTube, Vimeo, and our self-hosted streaming servers. The team iterated through multiple strategies for deploying the self-hosted video streaming server. In the end, Wowza was used to stream h.264 via RTMP and HTTP pseudostreaming. JWplayer was used to determine the capabilities of the student's devices and then request the appropriate stream from the Wowza server.

After the student requested a lecture via the class's web-page, the JWPlayer would determine his device [PC, MAC, Tablet, iPhone, Android, etc.], connection speed, set-up multiple boundary points for selecting streams at different connection speeds, and then start to play the appropriate stream for the user.

The team set-up the server stream H.264 over RTMP with a fallback to HTTP pseudo streaming and a final fallback to file download. RTMP was selected as the primary streaming mechanism due to its grater number of features related to dynamically handling changes in network connection speeds.

To adequately deliver video to the greatest number of students, an HTTP pseudo streaming fallback was required. RTMP works great with high-speed connections. However, in scenarios where the video is received at a slower data rate than it is consumed, the player will stop and not buffer gracefully like HTTP pseudo streaming. It also requires the user to accept connections via port 1935.

If RTMP failed, our set-up (JPlayer and Wowza) would seamlessly switch to an HTTP pseudo stream, which would work over port 80 and progressively download without creating an unpleasant experience for the user. A detriment to this methods however was that it did not offer all of the bandwidth dynamic features offered by RTMP.

As a last resort, the server also had a fallback from HTTP pseudo streaming, which would enable to user to simply download the files directly from the web-server. This accommodated users with relatively low connection speeds who preferred to download the files over a long period of time rather an enduring a long progressive HTTP pseudo stream or an RTMP stream that would not play.

In the end, the team elected to modify the JWPlayer to serve the video in the following sequence with graceful fall back: HTTP, RTMP, file download. With these modifications to the playback strategy, the team gave up some of the dynamic bandwidth mechanisms of RTMP, but gained the ability to deliver video to any of the students' devices reliably over a wide range of connection.

E. conclusion

In the end, the created solution meets all of the project's goals of delivering any of the videos to any of the common platforms in a very robust and scalable manner. The combination of dedicated video hardware and a GPU enabled compression cluster enabled the team to efficiently compress video and target a wide variety of platforms. Improvements to the system could be made by explicitly using Teslas for the compression phase. As the added CUDA cores would decrease compression times. However, at its current scale, the system was able to compress all of the 8-hours of video per day produced by the team in real-time. .