A Containerized Media Cloud for Video Transcoding Service

Yu Dong, Xusheng Zhang, Yanan Zhao, Li Song
Institute of Image Communication and Network Engineering
Shanghai Jiao Tong University
Shanghai, China
{thesmallfish, zhangxusheng, yanzhao, songli}@sjtu.edu.cn

Abstract—Videos transmitted on the Internet need multiple versions due to the variety of needs from users. Video providers need to transcode an original video into different formats, with different bitrates and resolutions to satisfy those needs. To meet such demands, we design and implement a novel lightweight containerized media cloud for video transcoding service in this paper. We show the high efficiency, robustness, scalability and portability of our system.

Index Terms—media cloud, containerized

I. INTRODUCTION

Numerous videos are produced and consumed on the Internet every day. However, an original video stream might not fit for all cases due to users’ different network states and device capabilities. A video transcoding system that can transcode a video stream into different formats with variable parameters is in need for video providers.

Large video providers like YouTube and Netflix have their own distributed systems for all kinds of video services on large public or private clouds. But for smaller providers, such systems are costly and unnecessary. A system on small private cloud with several powerful servers can satisfy their demands.

There have been many architectures on hand for distributed video transcoding systems, mostly using open source platforms like Apache Hadoop. In such systems, there are always multiple functional modules coupling together. System deploying and upgrading can also be inconvenient.

Martin Fowler and James Lewis [6] proposed the concept Microservices(MS) for system architecture. MS means designing an application as independent services, and reduce system coupling. Docker container [7], which can quarantine a single process with resource allocation, fits well for MS.

Aiming at the needs from small video providers, combining docker container and MS, we design and implement a novel containerized multimedia processing system. Our system is deployed over a private cloud and provides video transcoding service. It is lightweight and as efficient and robust as those heavy systems, as well as scalable. Moreover, its portability is better and resource consumption is less.

The rest parts of this paper are organized as follows. Section II introduces some related work on distributed video processing systems. In section III we illustrate the system we design in detail. In section IV we analysis our system and compare it with previous work to show its advantages. Section V comes to the conclusion.

II. RELATED WORK

There have already been several different work on video processing methods and systems.

Ryu et al. [2] implemented a framework on Apache Hadoop with FFmpeg, OpenCV employed for video and image processing and use the MapReduce architecture. Their system can achieve human face tracking tasks with good scalability.

Chen et al. [3] proposed a MapReduce-based parallel video transcoding method and implement a demo system on Apache Hadoop, which can split a video into sub-clips. Zhao et al. [4] proposed a prediction-based task scheduling method for video transcoding over MapReduce cluster, which is locality-aware and parallelized. Their methods reduce video transcoding time.

Chang et al. [5] proposed a real-time distributed video transcoding system, which support streaming video transcoding. Their system turn out to reduce CPU load in real time and can process more video streams at the same time.

In our previous work [1], we have already proposed a lightweight distributed media processing system for UHD service and it shows remarkable performance in both efficiency and network load.

Previous video transcoding systems are either too heavy to deploy, or do not have enough functions. So in this paper, we design and implement a novel containerized multimedia processing system for video transcoding service over private cloud. Our system shows good performance compared with previous work.

III. SYSTEM DESIGN

In this section, we will illustrate the multimedia processing system design in detail, including its architecture, components and applications.
A. System Architecture

The system we design is based on the server-worker system model, where a task is distributed by server and done by worker. In our system, there are three layers, which are front layer, service layer and control layer. There are also three flows named data flow, control flow and state flow, based on our previous work [1].

Layers are the abstracts of different module groups.

- **Front layer** is the abstract of user interface(UI). It interact with users as the gate of the system. Data and information flow into the system through the front layer. It is the outside layer of our system.

- **Service layer** is the abstract of the service pool, which contains most of the functional modules of the system, each module named a service according to MS. System functions are realized by getting different groups of services working together. Service layer is the inside layer of our system.

- **Control layer** is the abstract of the interface between front layer and service layer. It connects the other two layers together and passes data and information between them. Control layer contains services that take control of or monitor the whole system to get tasks done properly.

Flows represent the data and information transmission in the system.

- **Data flow** represents the transmission of input and output multimedia data in our system. In general, input data are the data to be processed provided by users, and output data are the result data. Data are transmitted from front layer to service layer directly, but under supervising of control layer.

- **Control flow** represents the transmission of control messages in our system. Usually, control messages are generated by user and transmitted from front layer to service layer through control layer. Control messages are always in JSON format, transmitted via HTTP protocol.

- **State flow** represents the transmission of state information in our system, including task tracking information, system monitoring information and other information showing states. State information can transmit between different layers.

B. System components

We pack each service of our system into a docker image. From a docker image, we can easily start docker containers that run the service packed in the image. The services are independent from each other, and different services do not contain any same function. A single service may contain different functions, but those functions are closely coupled and there is no need to tell them apart. In our system, services include HTTP service, video transcoding service, message queue service, cache service, storage service, database service, control service and monitor service.

Fig. 1 shows the system architecture including component services and the flows among them. In the system. Each services are realized by using open source tools and platforms.

Front layers contains web client.

- **Web Client** lets users upload new videos, set up parameters for video transcoding tasks and submit tasks. They can also view the detailed task states of existing tasks on web client. We use **Nginx**, an open source HTTP server to set up the web client. It provides HTTP service.

Service layer are composed of multiple services.

- **Worker** provides the services of executing the tasks given by users from web client. A worker container in our system is always based on the worker of **Celery**, an open source distributed task queue. A basic **Celery** worker can consume messages from the message queue it connects to. Our worker also contains **FFMPEG**, an open source multimedia tools package, X265, an open source codec for the latest video coding standard HEVC, and **Vapoursynth** [8], an open source nonlinear video manipulation application, including video enhancement, super resolution and up frame rates. Worker can transcode video into the format and parameters users require. Our system can have multiple workers for different or the same kind of tasks.

- **Message Queue** is a service to store task messages coming from front layer to be executed by workers. In our system, we use **Rabbit MQ**, an open source message broker that can be used by **Celery** as the message queue.

- **Storage** is a service to store input, output and temporary data generated by workers when processing multimedia data. In our system, we use **HDFS**, the distributed file system of **Apache Hadoop** as storage because of its data redundancy.

- **Cache** is a service to store task states for frequently read and write before a task is finished. It requires fast I/O rate when accessed with high frequency. In our system, we use **Redis**, an open source in-memory data storage as cache.

- **Database** is a service for permanently storing task states when a task is finished to save memory space used by cache. It also update the states of an unfinished task from cache with a certain frequency. In our system, **MySQL**, a popular open source database is used to store states and can be queried by other services.

Control layer includes controller and monitor.

- **Controller** is service that takes control of the whole system. It handles the control flow coming from web client, and pass the messages to appropriate services in service layer. It also analyzes state flow collected from database and return required information to web client for user. It is a logical **Celery** server and connect to **Rabbit MQ** message queue to send task messages. It also contains **FFMPEG** for slicing a video stream into parts for parallel processing by multiple workers and combing result parts into one result file.

- **Monitor** is a service that collect all state information of the system and display part of the information that users are interested in on a dashboard. The dashboard is
also accessible on web client. Monitor in our system is composed of three open source tools. Grafana [9] is an open source platform for graphic time series analytics and monitoring. It is used as the dashboard of the monitor. Influxdb [10] is an open source time series database, and Telegraf is the agent for collecting system state information working together with Influxdb. Telegraf collect hardware information and state of each docker container standing for each service, and save the data into Influxdb. At the same time, Grafana reads time series state data from Influxdb as well as those task states provided by controller, and displays them on web client.

C. System Applications

The basic application of our system is video transcoding, which takes an video stream as input and outputs video streams with preset parameters and formats. Parameters include bitrate, bit depth, resolution, frame rate, color format and several video enhancement parameters. Formats include video encoding format, audio encoding format and video encapsulation format.

For example, a common application of our system is transcoding a video file of MPEG-4 format, video codec AVC, resolution HD(1920x1080), bitrate 30Mbps, frame rate 24 FPS, bit depth 8 with audio codec AC3 into a video stream of MPEG-2 TS format, video codec HEVC, resolution 4K(3840x2160), bitrate 20Mbps, frame rate 50 FPS, bit depth 10 with HDR color format and audio codec E-AC3. Such application is called HD to Full-4K in our system.

Our system can also produce video streams with different encoding parameters simultaneously from one input stream.

In additional, the system is suitable for video modifying applications like video cutting, montage, edge resection and audio extracting.

IV. SYSTEM ANALYSIS

In this section, we will first illustrate the deployment environment of our multimedia processing system. Next we are going to use some case tests to show its high efficiency, robustness, scalability and portability. And finally, we will compare it with some previous work to show its advantages.

A. Deployment Environment

Our media cloud is built on a private cluster with six physical nodes. One master node is equipped with two Intel Xeon E5-2680 v4 CPU, 128GB memory and an Nvidia GTX 1080 GPU on Ubuntu 16.04 operating system. The other five slave nodes each is equipped with an Intel Xeon E5-2630 v3 CPU, 32GB memory and Ubuntu 14.04 operating system. All the six nodes are connect by a gigabit switch.

Controller, database, message queue, cache, web client, storage and monitor are deployed on master node. Each node has two workers. Particularly, the two workers running on master node can access to additional GPU capability.

All the services are packed up as docker images, and run in docker container. Resources allocated to different services are isolated. By exporting port on docker container, different services can communicate with each other.

Network bandwidth is 1Gbps, and time consumption for transmitting a HD movie(usually about 30 GB) is less than 10 minutes, far less than the time used to process it(usually above 10 hours).

Communication is also secure in private cloud. Devices from outside the network can only get access to our system through the web client, and only authenticated users could get access to their own files.

B. System Features

We analyze the system features by using some test cases including stress test, stopping and starting worker containers randomly and shutting down physical slave nodes.

1) Efficiency: Efficiency of our system can be shown in multimedia data processing speed. Vapoursynth itself is a video processing tool with high efficiency comparing with other tools like Avisynth, and docker container technique take good use of low level hardware resources and has higher efficiency than traditional virtual machines(VMs).

What’s more, our system support source file slicing and GPU acceleration, which can concurrently process a task and use GPU computing resource for video super resolution acceleration.
2) **Robustness**: We test the robustness of our system by stress test, stopping worker containers and shutting down physical slave nodes.

   We used a short video stream (280s long) with small resolution (672x378) as input stream. We created tasks repeatedly using the same input stream and make the transcoding task concurrently. The tasks were easy as it just scan every frames in the video and can be done quite fast. We set the concurrent number of tasks to 20, so that there are at most 20 tasks queuing in the task queue. This is because for our private cloud, saving the data for 20 normal tasks is almost reaching its storage limit. The result of stress test was that the system ran for over 10 hours and successfully finished more than 2000 tasks without any crash. Message queue and controller are robust enough from the stress test, as real video transcoding tasks are time consuming and have less load on message queue and controller.

   Then we repeated the stress test, adding new operations that worker containers might stop randomly or physical slave node might be shut down by accident. In this case, the task doing on the certain worker container or slave node were rescheduled and executed by other well performing workers. This shows the robustness of our system when some software or hardware failure occurs.

   3) **Scalability**: We test the scalability of our system by starting worker containers when the system is on. Logs from message queue show that the new started worker container is connected to the system immediately, and from monitor page on web client, we can see that it start executing tasks. This means that we can add worker containers or even slave nodes to the system when its load is high. The controller and message queue we use also support distributed mode, where multiple processes can be started when load is high. Thus, our system has good scalability.

   We can also add new functional modules to our system by upgrading codes of controller and import new worker containers. There is no need for recompiling the whole system and upgrading just takes very few seconds.

   4) **Portability**: Our system is easily portable. The deployment of the system is simple, just make sure a new physical node is equipped with the docker engine (can be installed with a single command), and copy the docker images to the new device. Automated deployment takes less than 30 seconds, comparing to traditional systems using VMs or even software deployed directly on physical machines that may take tens of minutes or even hours to deploy or upgrade.

   Besides fast deployment, our system can also be upgraded in seconds with the help of full containerization.

   **C. System Comparison**

   We compare our system with some previous works in several features.

   Table I shows comparison of different systems. Our system shows good portability among all systems thanks to containerization. And it also has good robustness, scalability and abundant functions.

   In additional, our video transcoding system has been deployed on Tianhe-2 supercomputer at National Supercomputer Center in Guangzhou, China. The system will be used to do the transcoding and delivering of traffic monitoring video task with thousand parallel.

   **V. Conclusion**

   In this paper, we design and implement a containerized multimedia processing system for video transcoding service over private cloud. The system use multiple open source tools, and is convenient and easy to deploy. It has good scalability and robustness, and is efficient to transcode video and audio with abundant functions. It has also been deployed on Tianhe-2 supercomputer for further use.

   Our future work will focus on better load balancing scheme, as well as distributed server node to make it more robust. We will also optimize the vapoursynth plugins and filters to get higher performance.

   **ACKNOWLEDGMENT**

   This work was supported by NSFC (U1611461, 61671296, 61521062), the 111 Project (B07022 and Sheitc No.150633) and the Shanghai Key Laboratory of Digital Media Processing and Transmissions.

   **REFERENCES**


   [7] https://www.docker.com


   [9] https://grafana.com

   [10] https://www.influxdata.com