2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)

Scalable Video Transcoding in Public Clouds

Qingye Jiang
School of Computer Science
The University of Sydney
Sydney NSW 2006, Australia
Email: qjiang@ieee.org

Young Choon Lee
Department of Computing
Macquarie University
Sydney NSW 2109, Australia
Email: young.lee@mq.edu.au

Albert Y. Zomaya
School of Computer Science
The University of Sydney
Sydney NSW 2006, Australia
Email: albert.zomaya@sydney.edu.au

***Abstract*—In this paper, we present the challenges involved in large-scale video transcoding application in public clouds. We introduce the architecture of an existing video transcoding system which is tightly coupled with an existing video sharing service. We examine the horizontal scalability of the video transcoding system on AWS EC2. With an online transaction processing (OLTP) model, the system achieves linear horizontal scalability up to 1,000 vCPU cores, but starts to experience performance degradation beyond that. We analyze the resource consumption pattern of the existing system, then introduce an improved architecture by adding a message queue layer. This effectively decouples the video transcoding system from the video sharing service and converts the OLTP model into a batch processing model. Large-scale evaluations on AWS EC2 indicate that the improved design maintains linear horizontal scalability at 10,100 vCPU cores. The hybrid design of the system allows it to be easily adapted for other batch processing use cases without the need to modify or recompile the application.**

***Index Terms*—video transcoding, batch processing, distributed computing, horizontal scalability**

I. Overview

Video transcoding is a common use case for websites and mobile applications providing video sharing service. When an end user uploads a video for sharing, the uploaded video comes with a particular file format, resolution, and bitrate. To provide the best user experience for the viewers, the service provider often needs to convert the same video into different file format, resolution, and bitrate so that it can be delivered to different devices with different runtime environments over different network connectivity conditions.

Video transcoding is one of the many features of the video sharing service. In the academic world, scalable video transcoding systems use either MPI or MapReduce as the distributed computing technique. In the industry, scalable video transcoding systems are usually coupled with the video sharing service in some way, including obtaining transcoding jobs from and updating transcoding status back to a core database. In both cases, these systems exhibit linear horizontal scalability at small scale, but starts to experience performance degradation at large scale. Such performance degradation often occurs when the resource consumption on the database server is very low, making it very difficult to debug and improve.

In this paper, we present the challenges involved in large-scale video transcoding systems. We analyze an existing video transcoding system, which is tightly coupled with an existing video sharing service through a shared database. The system achieves linear horizontal scalability up to 1,000 vCPU cores, but start to experience performance degradation beyond 1,000 vCPU cores. By examining the resource consumption pattern of the existing system, we improve the system by introducing a message queue layer. Such slight change converts the tightly coupled situation into a loosely coupled situation, from an online transaction processing (OLTP) model to a batch processing model. This effectively improved the horizontal scalability of the video transcoding system. Futhermore, we introduced a hybrid design that includes a Java application and a bash script. The Java application forms the batch processing framework, while the bash script performs the actual transcoding work. The specific contributions of this paper include:

- We present a scalable video transcoding system with a producer/consumer model. Large-scale evaluations on AWS EC2 indicate that the scalable video transcoding system maintains linear horizontal scalability at 10,100 vCPU cores.[1] This is by far the largest video transcoding system that has been reported.
- The hybrid design of the system allows it to be easily adapted for other batch processing use cases without the need to modify or recompile the application.

The rest of this paper is organized as follows. Section II describes the use case, the test data, and the test environment. Section III describes the performance metrics of the existing system, along with an analysis on the resource consumption pattern and possible bottlenecks. Section IV describes the improved architecture and implementation, following by how large-scale performance evaluations in Section V. Section VI reviews related work. We conclude our paper in Section VII.

II. Application Scenario

Figure 1 illustrates the application scenario of the video transcoding system. When the end user uploads a video to the video sharing service, the video is stored in a bucket on Amazon Simple Storage Service (S3) and a new record is inserted to a table in a relational database, with the transcoding status set to PENDING. When the total number of videos in PENDING status reaches a certain threshold, a group of

[1]To allow other researchers to reproduce our test results, the source code and test data is made available on GitHub https://github.com/qyjohn/ScaleTranscoder.

978-1-7281-0912-1/19/$31.00 ©2019 IEEE
DOI 10.1109/CCGRID.2019.00017

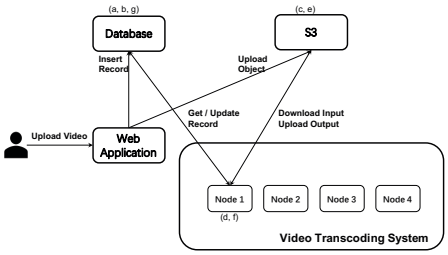


Fig. 1: Architecture of the Existing Video Transcoding Application.

transcoder nodes (Amazon EC2 instances) are launched to transcode the uploaded videos. The video transcoding system employs a stateless design in that (a) a transcoder node does not persist any job state on itself, and (b) the transcoder nodes are not aware of each other and do not communicate with each other in any way. On each transcoder node, multiple threads are launched to do the actual work, where the number of threads equals to the number of vCPU cores available on the transcoder node. Each thread performs the following tasks in sequence: (a) obtains one record with PENDING status from the MySQL database at a time, (b) updates the corresponding record with the status set to PROCESSING to avoid other worker threads from working on the same video clip, (c) downloads the video clip from S3 to local disk, (d) uses FFmpeg [1] to convert the video clip from MP4 format to WMV format, (e) uploads the output file from local disk back to the same S3 bucket, (f) deletes both the MP4 and WMV files from local disk, and (g) updates the corresponding record with the status set to COMPLETED. This process continues until the status of all records in the database become COMPLETED.

We use 1,000,000 video clips to evaluate the performance of the video transcoding system. These video clips are replicated from 20 original video clips with lengths from 6 seconds to 117 seconds. For each original video clip we create 50,000 replicas, representing 5% of the total workload. The records of these video clips are evenly distributed in the table space, simulating a large-scale video sharing service on which a large group of users sharing short videos taken from different mobile devices. Table I provides a detailed description of the video clips used in the evaluation. For each video clip, a UUID is generated and used as the object key, which "hints" S3 to distribute the video clips to multiple partitions for improved performance. The video clips are pre-staged onto S3, and the corresponding records are pre-inserted into the database. The size of the table (with the corresponding index) is 260 MB, which is significantly smaller than the memory available on the database server (60 GB). Before we start the video transcoding system, we pre-load all the data into memory with a "SELECT * FROM jobs" query. As a result, all database operations are performed against in-memory data, apart from persisting updated data to the underlying storage.

[1]https://www.ffmpeg.org

TABLE I: Video Clips for Testing.

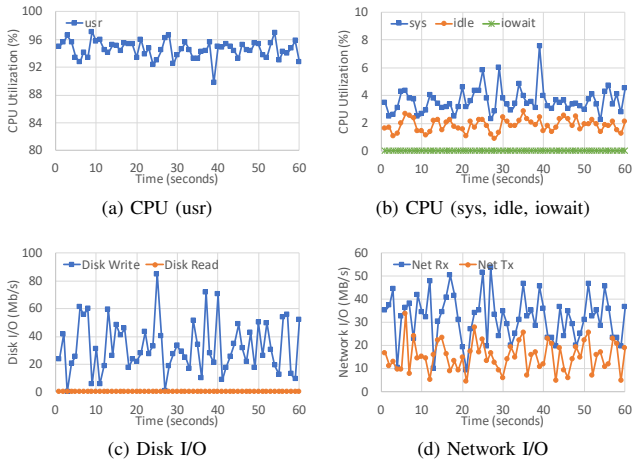
Video Clip	Length (seconds)	MP4 Size (bytes)	WMV Size (bytes)
01	6	558708	369095
02	12	1,141,455	644,331
03	18	1,746,849	916,367
04	24	2,333,556	1,188,403
05	30	2,906,927	1,457,239
06	36	3,487,456	1,726,075
07	42	4,073,664	1,994,911
08	48	4,665,068	2,273,347
09	54	5,250,610	2,542,183
10	60	5,847,580	2,811,019
11	66	6,455,206	3,083,055
12	72	7,048,175	3,355,091
13	78	7,626,107	3,623,927
14	84	8,215,317	3,892,763
15	90	8,824,693	4,164,799
16	96	9,419,633	4,436,835
17	102	9,997,732	4,705,671
18	108	10,585,186	4,977,707
19	114	11,182,329	5,246,543
20	117	11,451,138	5,368,155

We carry out the evaluations on AWS in the ap-southeast-2 (Sydney) region. The c3.8xlarge EC2 instance type is used exclusively in the evaluation of the original system. The EC2 instance has 32 vCPU cores, 100-GB General-Purpose SSD root EBS volume, 60 GB memory, with the advertised network bandwidth being 10 Gbps. The operating system is 64-bit Ubuntu 18.04.1 LTS, and the file system is EXT4. The database is MySQL release version 5.7.24 running on a dedicated EC2 instance. The S3 bucket used to store the video clips is created in the same AWS region. In the evaluations we deploy the video transcoding system in an auto-scaling group, with the transcoder nodes being launched in multiple availability zones. This allows us to easily change the number of transcoder nodes when needed.

III. Performance Metrics of the Original System

Figure 2 presents a 1-minute sampling of the resource consumption pattern on the transcoder node when there is only one transcoder node in the system, with the sampling period being 1 second. The transcoding application utilizes over 90% of the CPU resource (Figure 2a) most of the time. The small system utilization (sys) reflects the context switch between multiple threads. There is also some slight idle CPU time, but there is absolutely no iowait observed (Figure 2b). Disk writes occur in a burst pattern, with no disk reads observed (Figure 2c). The amount of network receive (net rx) is slightly higher than network transmit (net tx), because the size of the source video clip is bigger than the size of the transcoded video clip (Figure 2d). In summary, the application is compute-intensive, with no significant pressure on either disk I/O or network I/O. This makes it an ideal use case for the horizontal scaling technique – that is, increasing the combined processing capacity of a system by adding more nodes into the system.

The performance of the video transcoding system is measured by its transcoding speed, which is the number of video



(a) CPU (usr) (b) CPU (sys, idle, iowait)

(c) Disk I/O (d) Network I/O

Fig. 2: Resource Consumption Pattern on the Transcoder Node.

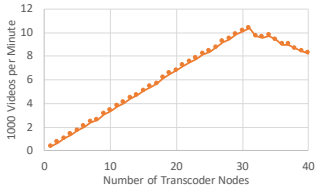
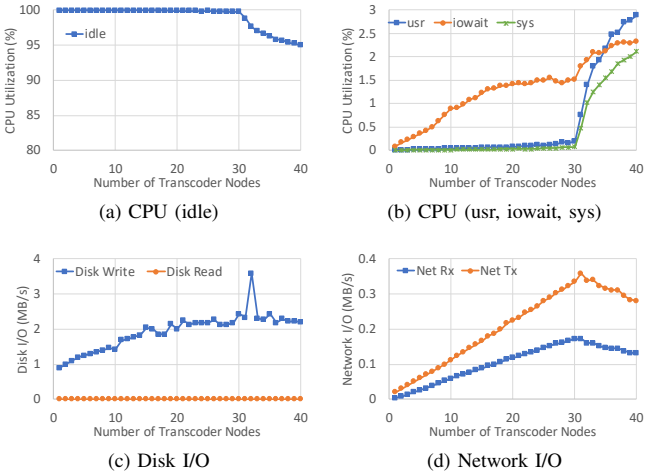


Fig. 3: Performance Growth of the Original Video Transcoding System.

clips processed in a minute. Figure 3 presents the relationship between performance and the number of nodes in the transcoder fleet. At a first glance the system exhibits perfect linear horizontal scaling behavior. However, performance degradation occurs when there are more than 32 transcoder nodes (1,032 vCPU cores) in the fleet. Starting from this point, adding more transcoder nodes into the fleet results in performance decreases instead of performance increases.

With a stateless design, the transcoder nodes do not have any direct performance impact on each other. As such, the observed performance degradation suggests insufficient computing capacity on the coordinating node, which is the database. Figure 4 presents the resource consumption pattern observed on the database node, where each data point represents the average level of resource consumption observed over a 1-minute sampling period with a 1-second sampling interval. Surprisingly, CPU resource is idling most of the time (Figure 4a). In the worse case, the database server only consumes less than 3% CPU time (usr), with approximately the same level of iowait (Figure 4b). There is up to 4 MB/s in disk writes and no disk reads (Figure 4c), which is very small comparing to the level of disk I/O observed on the transcoder node (Figure 2c). Both network transmit and network receive throughput are very low, and they correlate well with the transcoding speed as shown in Figure 3.



(a) CPU (idle) (b) CPU (usr, iowait, sys)

(c) Disk I/O (d) Network I/O

Fig. 4: Resource Consumption Pattern of the Database Node.

The fact that iowait occurs when the disk I/O throughput is low suggests that the database server is making a large number of small I/O requests. The General-Purpose SSD EBS volume can consistently deliver 3 IOPS per GB [1]. When the root EBS volume is 100 GB, it is capable of performing 300 I/O operations per second. As shown in Figure 3, the video transcoding system is capable of processing approximately 10,000 video clips per minute, which is 166 video clips per second. Assuming two disk I/O requests on the database node are needed to process each video clip, it demands approximately 330 IOPS from the root EBS volume. These two disk I/O requests correspond to two database updates in the transcoding job, one to change the status of a record to PROCESSING, and the other to change the status of the same record to COMPLETED. When the video transcoding system creates more disk I/O requests (by adding transcoder nodes to the fleet) than the disk I/O capacity available on the database node, the disk I/O requests queues up on the operating system level, resulting in the iowait observed. As a consequence, the performance of the video transcoding system starts to deteriorate.

The above-mentioned mechanism suggests that the video transcoding system is tightly coupled with the video sharing service through the database. By using the database as the job coordination layer, the video transcoding system has the same workload characteristics as an OLTP system. At a first glance this seems to be an easy problem to solve. A larger EBS volume can provide higher disk I/O capacity, hence performance gain can be achieve by using a larger EBS volume on the database server. However, there are some well-known issues with such approach. Firstly, there exists a 16,000 IOPS hard limit for a single EBS volume. Secondly, currently we are only utilizing less than 1% out of the 100-GB storage capacity,

[1]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html

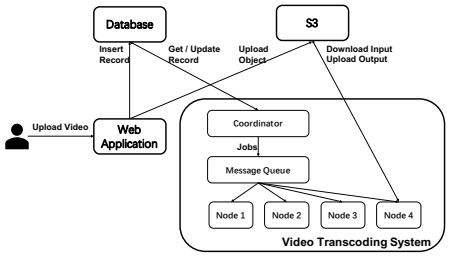


Fig. 5: Architecture of the Improved Video Transcoding Application.

provisioning more storage capacity to provide greater disk I/O capacity is not economically feasible. As such, an improved architecture is needed to solve the performance issue with the video transcoding system.

IV. Improved Architecture

Figure 5 presents the architecture of the improved video transcoding system. By adding a message queue layer between the database and the transcoder nodes, we decouple the video transcoding system from the database. This also converts the architecture from an OLTP system into a batch processing system. Upon start up, a job coordination program performs a SELECT query to retrieve all records in PENDING state and publishes them into the message queue as jobs pending processing. With a producer/consumer design, the transcoder nodes consume jobs from the message queue, process the corresponding video clips, and acknowledge to the message queue when jobs are completed. The job coordination program checks the message queue for completed jobs with a 1-second interval, then updates the corresponding records in the database with the status set to COMPLETED. To reduce the volume of disk I/O requests, the database updates are make in batches.

In our implementation, the message queue is RabbitMQ 3.6.9 and it runs on the same server as the database. In general, message queue is a memory-intensive application. However, the size of the data set (260 MB) is significantly smaller than the memory available on the database server (60 GB). Considering the resource abundance on the database server, the job coordination program also runs on the database server. Apart from the video transcoding fleet, the improved architecture does not require additional nodes in the system.

The application running on the transcoder nodes adopts a hybrid approach. The application receives job description from the message queue in the form of strings, then calls a bash script *execute.sh* with the string as the runtime parameter (Figure 6). The bash script then performs the actual work (Figure 7), including (a) downloading the MP4 video clip from S3, (b) use FFMPEG to perform the transcoding, and (c) uploading the WMV video clip to S3. After the bash script finishes execution, the application acknowledges back to the message queue that this particular job is completed. This architecture can be easily extended to a variety of other use cases by simply modifying the bash script, without the need to modify or recompile the application itself.

```
public void execute(String jobInfo){
    try{
        // execute a particular job
        String cmd = "~/bin/execute.sh " + jobInfo;
        Process proc = Runtime.getRuntime().exec(cmd);
        proc.waitFor();
    } catch (Exception e){}
}
```

Fig. 6: Jobs obtained from the message queue is processed by calling a external bash script.

```
#!/bin/bash
s3Bucket=my_s3_bucket
src=/tmp/$1.mp4
dst=/tmp/$1.wmv
aws s3 cp s3://$s3Bucket/$1.mp4 $src
ffmpeg −i $src $dst
aws s3 cp $dst s3://$s3Bucket/$1.wmv
rm $src $dst
```

Fig. 7: The external bash script performs the actual transcoding job. This script can be easily modified to handle jobs in other use cases.

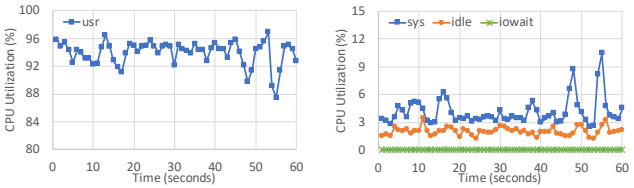
V. Performance Metrics of the Improved System

Due to EC2 instance quota limits, we use a combination of c3.8xlarge, c4.8xlarge and c5.9xlarge instance types in this evaluation. Table II describes the configuration of the different instance types in details. It should be noted that these instance types have different CPU's with different performance. We use the publicly available PassMark scores to represent the performance of the CPU. The c4.8xlarge instance type has a more powerful CPU than c3.8xlarge, while the c5.9xlarge instance type has a more powerful CPU than c4.8xlarge. Also, each instance type has different number of vCPU cores and memory configuration.

Due to cost considerations, the large-scale evaluation is carried in multiple stages with different step sizes. In the first stage, we use the c3.8xlarge instance type and increase the number of transcoder nodes from 0 to 100 (3,200 vCPU cores), with a step size of 5 nodes. In the second stage, we use the c3.8xlarge instance type and increase the number of transcoder nodes from 100 to 200 (6,400 vCPU cores), with a step size of 10 nodes. In the third stage, we use a mixture of all three instance types to scale the number of transcoder nodes from 200 to 300 nodes (10,100 vCPU cores), with a step size of 50 nodes. The last three data points in the subsequent analysis represent results obtained from transcoder fleets with (100, 50, 50), (125, 50, 75) and (175, 45, 80) instances for the (c3.8xlarge, c4.8xlarge, c5.9xlarge) instance types. To account for the difference in the number of vCPU

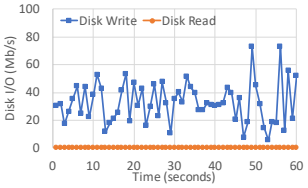
TABLE II: Configuration of EC2 Instances. The PassMark scores are obtained from https://www.cpubenchmark.net/.

Instance Type	CPU Type	PassMark Score	vCPU Cores	Memory (GB)	SSD Storage (GB)
c3.8xlarge	Intel Xeon E5-2680 v2 @ 2.8 GHz	15877	32	60	100
c4.8xlarge	Intel Xeon E5-2666 v3 @ 2.9 GHz	24877	36	60	100
c5.8xlarge	Intel Xeon Platinum 8124M @ 3.0 GHz	26652	36	72	100

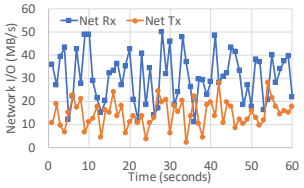


(a) CPU (usr)

(b) CPU (sys, idle, iowait)



(c) Disk I/O



(d) Network I/O

Fig. 8: Resource Consumption Pattern on the Transcoder Node.

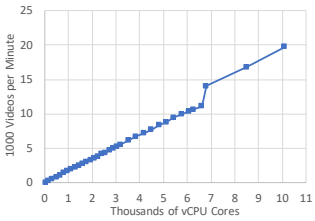
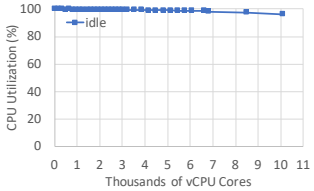


Fig. 9: Performance Growth of the Improved Video Transcoding Application.

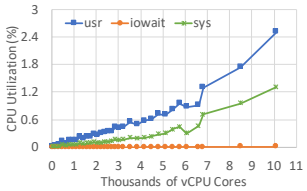
cores on the transcoder nodes, we use the number of vCPU cores to represent the size of the transcoder fleet.

Figure 8 presents a 1-minute sampling of the resource consumption pattern on the transcoder node when there is only one transcoder node (c3.8xlarge) in the system. The video transcoding application utilizes over 90% of the CPU resource (Figure 8a), with some small system utilization (sys) and idle time and no iowait (Figure 8b). Disk writes are very low, with no disk reads observed (Figure 8c). Network transmit (net tx) and receive (net rx) throughputs are on the same level as disk write throughput (Figure 8d). In summary, the resource consumption pattern on the improved transcoder node is very similar to that on the original transcoder node as described in Figure 2.

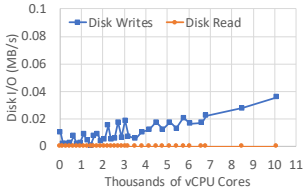
Figure 9 presents the relationship between performance and the number of vCPU cores in the transcoder fleet. There are up to 10,100 vCPU cores in the transcoder fleet, with up to 18,960 GB memory. The improved video transcoding system



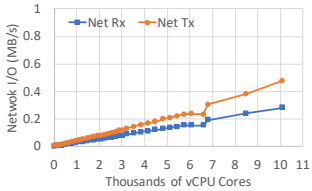
(a) CPU Utilization



(b) CPU Utilization



(c) Disk Writes



(d) Network I/O

Fig. 10: Resource Consumption Pattern on the Database Node.

exhibits a good linear horizontal scaling behavior throughout the evaluation. The sudden performance jump observed at vCPU=6800 is due to the introduction of more powerful CPU's on the c4.8xlarge and c5.9xlarge instance types. Throughout the evaluation, no performance degradation is observed. This proves that the improved architecture is effective in achieving higher performance by (a) decoupling the video transcoding system from the database, and (b) replacing the OLTP model with the batch processing model.

Figure 10 presents the resource consumption pattern observed on the database node, where each data point represents the average level of resource consumption observed over a 1-minute sampling period with the sampling interval being 1 second. CPU resource is idling most of the time (Figure 10a). In the worse case, the database server only consumes less than 3% CPU time (usr), with no iowait observed (Figure 10b). There is less than 0.04 MB/s in disk writes and no disk reads (Figure 10c). Both network transmit and network receive throughput are very low (Figure 10d), and they correlate well with the transcoding speed as shown in Figure 9.

With the improved architecture described in Section IV, this system can be easily extended to a variety of other batch processing use cases by simply modifying the bash script, without the need to modify or recompile the application itself. On the worker node level, the bash script implementation is stateless in that it does not store job states on the node or communicate job states with other worker nodes. On the system level, the producer/consumer architecture facilitated by a message queue allows the system to be horizontally scalable. As demonstrated by the video transcoding use case, the system

does not experience any performance degradation at 10,100 vCPU cores.

VI. Related Work

Video transcoding has been a thoroughly researched topic in multimedia and signal processing. Vetro et. al. [1] review the various techniques in the transcoding of block-based video coding schemes, with a focus on bitrate reduction, spatial and temporal resolution reduction, as well as error resilience in video transcoding. Xin et. al. [2] outline technical issues in video transcoding, with a focus on reducing transcoding complexity and improving video quality by exploiting information obtained from the source video bit stream. Ahmad et. al. [3] review various video transcoding architectures and discuss the trade-off between the computational complexity and the quality of the reconstructed video.

Sambe et. al. [4] present the design and implementation of a video transcoding cluster with a master/slave architecture. Small-scale evaluations on a cluster with up to 10 nodes indicate a none-linear relationship between performance and the number of nodes. Pereira et. al. [5], Garcia et. al. [6] and Zhao et. al. [7] propose using MapReduce as the distributed computing technique for building scalable distributed video transcoding systems. Fareed [8] proposes a scalable distributed video transcoding system based on MPI, where the cluster includes a master node and multiple worker nodes. However, the scalability of these systems are not reported. Feng et. al. [9] present a parallel video transcoding system with a MapReduce architecture and verify the viability of the proposed system with numerical simulations. Kim et. al. [10] propose a distributed video transcoding system based on the Hadoop MapReduce framework. The authors conduct evaluations on Hadoop clusters with up to 28-node (224 vCPU cores). In this case, linear horizontal scaling behavior is observed for videos larger than 4 GB, but performance degradation is observed for videos smaller than 4 GB. Liu et. al. [11], [12] reported VideoCoreCluster, a low-cost and highly efficient video transcoder cluster based on Raspberry Pi. A common characteristics of existing literature is they are relatively small-scale studies. The largest cluster was reported by Pereira et. al. [5], with up to 225 vCPU cores in the cluster.

With the increasing adoption of public clouds, dynamic resource allocation for video transcoding workload becomes an important issue. Ashraf et. al. [13], [14] present a batch processing system with a producer/consumer design, where the job distribution is facilitated by a message queue. The authors develop a method to predict the video transcoding workload, then use the prediction to dynamically manage the amount of computing resources in the video transcoder fleet. Discrete-event simulation and a small-scale experiment are used to evaluate the effectiveness of the proposed dynamic resource allocation algorithm.

VII. Conclusion

In this paper, we present the challenges involved in large-scale video transcoding use cases. We analyze an existing video transcoding system, which achieves linear horizontal scalability up to 1,000 vCPU cores, but experiences performance degradation beyond 1,000 vCPU cores. By examining the resource consumption pattern of the existing system, we redesign and implement the system with a producer/consumer architecture. Large-scale evaluations indicate that the improved application maintains linear horizontal scalability at 10,100 vCPU cores. This is by far the largest video transcoding system that has been reported. The hybrid design of the system allows it to be easily adapted for other batch processing use cases without the need to modify or recompile the application.

VIII. Acknowledgement

Professor Zomaya‘s work is supported by an Australian Research Council Linkage Grant (LP150101213).

References

[1] 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.

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

[3] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, “Video transcoding: an overview of various techniques and research issues,” *IEEE Transactions on multimedia*, vol. 7, no. 5, pp. 793–804, 2005.

[4] Y. Sambe, S. Watanabe, D. Yu, T. Nakamura, and N. Wakamiya, “High-speed distributed video transcoding for multiple rates and formats,” *IEICE Transactions on Information and Systems*, vol. 88, no. 8, pp. 1923–1931, 2005.

[5] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, “An architecture for distributed high performance video processing in the cloud,” in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. IEEE, 2010, pp. 482–489.

[6] A. Garcia, H. Kalva, and B. Furht, “A study of transcoding on cloud environments for video content delivery,” in *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*. ACM, 2010, pp. 13–18.

[7] H. Zhao, Q. Zheng, W. Zhang, and J. Wang, “Prediction-based and locality-aware task scheduling for parallelizing video transcoding over heterogeneous mapreduce cluster,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 4, pp. 1009–1020, 2018.

[8] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, “Bit rate reduction video transcoding with distributed computing,” in *Proceedings of the 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2012, pp. 206–212.

[9] F. Lao, X. Zhang, and Z. Guo, “Parallelizing video transcoding using map-reduce-based cloud computing,” in *Proceedings of the 2012 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2012, pp. 2905–2908.

[10] M. Kim, Y. Cui, S. Han, and H. Lee, “Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, pp. 213–224, 2013.

[11] P. Liu, J. Yoon, L. Johnson, and S. Banerjee, “Greening the video transcoding service with low-cost hardware transcoders,” in *USENIX Annual Technical Conference*, 2016, pp. 407–419.

[12] P. Liu, J. Yoon, H. R. Kim, and S. Banerjee, “Videocorecluster: Energy-efficient, low-cost, and hardware-assisted video transcoding system,” *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[13] A. Ashraf, F. Jokhio, T. Deneke, S. Lafond, I. Porres, and J. Lilius, “Stream-based admission control and scheduling for video transcoding in cloud computing,” in *Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2013, pp. 482–489.

[14] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, “Prediction-based dynamic resource allocation for video transcoding in cloud computing,” in *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2013, pp. 254–261.