A Complexity Aware Scheduler With Dynamic Slot Allocation for Cloud Video Transcoding

JIANN-JONE CHEN1, (Member, IEEE), HAN-YEN YU2, CHING-CHENG HWUANG3, CHENG-JEI SUNG4, AND YOU-SHIN WU1

1Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan
2Digital Education Institute, Institute for Information Industry, Taipei 10622, Taiwan
3Asus Company, Hsinchu 300046, Taiwan
4Taiwan Branch, Teradyne Company, Hsinchu 30288, Taiwan

Corresponding author: Jiann-Jone Chen (jjchen@mail.ntust.edu.tw)

This work is partially supported by the Taiwan Ministry of Science and Technology (MOST) with grant No. MOST 111-2221-E011-073.

ABSTRACT For a multimedia cloud computing platform, it needs to perform video transcoding to provide bandwidth-compatible bit-streams for users. In configuring a MapReduce system, it allocates slots to workers with the assumption of homogeneous worker power and performs task scheduling by assuming equal task time complexity. However, the computing power of a practical cluster of workers and the time complexity of tasks is time-varying in any case. The task-scheduling algorithm has to well manipulate these heterogeneous cloud resources and task complexity. We first find a good partition size for a video transcoding job for efficient processing. We proposed a Complexity-Aware Scheduling (CAS) algorithm that reorders task assignment priorities according to task complexity to maintain load-balancing operations. By utilizing a Neural Network model to refine the task complexity estimation for the CAS(CASNN), the scheduling and transcoding speedup performances can further be improved. Based on the CASNN, we proposed a Dynamically Adjusting Slot number allocation (DAS) method, DASCASNN, to adjust the slots according to resource utilization status to improve the processing performance. Experimental results show the proposed DASCASNN can help reduce 30% of the transcoding time on average as compared to available schedulers and increase the resource utilization rates from 82.7% to 98%.

INDEX TERMS Cloud video transcoding, resource management, complexity-aware scheduling, dynamic slot allocation, load balance operation, heterogeneous parallel system.

I. INTRODUCTION

For universal video access, a cloud computing-assisted video processing and streaming platform has to provide variable bitrate video bitstreams for users [1]. It has to transcode the original video format to another for the target user device. For example, an original HEVC format video has to be transcoded to H.264/MPEG4 one. Other feasible scalable dimensions comprise spatial, quality, and temporal scalability. A Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [2] service framework proposed by MPEG can dynamically adapt media streaming over HTTP to provide bandwidth-compatible video consumption. For one MPEG-DASH client, after requesting for connection, it downloads a Media Presentation Description (MPD) file from the server which helps the client to select the next video segment to download and playback according to its available bandwidth. The job of video transcoding is computationally expensive and has to be carried out through a cloud computation platform to accommodate the computation loading and support the MPEG-DASH service.

The cloud video streaming server is developed based on distributed resources and grid computation frameworks. One transcoding job of a cloud platform has to be partitioned into smaller segments for concurrent processing through task scheduling and assignment operations. The Hadoop MapReduce framework is widely adopted for developing a cloud platform to process parallelizable problems across different datasets through a computing cluster with $N_w$ workers, $\mathbb{W} = \{w_j\}_{j=1,2,\ldots,N_w}$. One big job, $J$, is partitioned into smaller
tasks, say \( N_t \) tasks \( \{ t_i \}_{i=1,2,\ldots,N_t} \), which are distributed by a mapper to the Hadoop File System (HDFS) and to activated workers [3]. After all \( t_i \)s have been processed by respective workers, they are transmitted to a reducer to integrate small tasks back to a finished job [4], [5]. Cloud computation is capable of handling tremendous distributed computing and storage resources. The task scheduling algorithm for a \( J \) dominates the efficiency of any distributed systems, such as Grid, Cloud, and P2P networks [6]. However, how to minimize the makespan, \( T_{MS} \), through task scheduling, is NP-Complete [7]. There is no universal best task scheduling algorithm for all kinds of applications. Scheduling algorithms are usually optimized for specific applications.

A cloud video transcoding framework [8] takes bitrates and encoding speed as costs to perform optimization control of a task scheduler. Another [9] sets the highly load-balanced system operation as the control target to design the parallel video encoding system. A divisible load theory is utilized to distribute video frames among workers, in considering load partition granularity and the associated overhead. A random distribution [10] and a round-robin scheduling [11], [12] approaches were proposed to improve processing efficiency. The round-robin approach is easy to implement but less efficient when processing tasks with different time-complexity under a heterogeneous \( \mathbb{W} \). A weighted round-robin method [13] considers the individual worker’s processing capability when assigning tasks. An optimized scheduler dataflow [14] was proposed to minimize the job completion time and execution cost, and the best-compromised operation parameters were also determined. To avoid overflow of real-time transcoding applications, a dynamic resource allocation method based on prediction for cloud video processing [15] was proposed to allocate virtual machine resources. Some schemes [16], [17], [19], [20] are designed to dynamically adjust the number of slots to increase system resource utilization rates. Different scheduling algorithms [21], [22], [23] were proposed to deal with task assignments under heterogeneous computer clusters.

When a cloud platform is managing a homogeneous \( \mathbb{W} \), it can predict the task time-complexity or processing time to perform task scheduling for load-balanced operations [24]. In addition to scheduling algorithms, the input file partition size, \( z \), also dominates the processing performance. It can be determined by mathematically derived models to yield the highest transcoding performance [10]. Although this \( z \) would be kept unchanged after system pre-configuration, performing scheduling by assuming homogeneous task processing time would suffer from degraded performance, as the practical execution time for one video segment would depend on its complexity. A multi-layer input file decomposition method [25] was proposed to partition an input file \( F_{in} \) with different sizes to yield large and small segments. The small ones can be assigned to workers after all the larger ones to avoid a convoy effect [27] that would lead to load-unbalance and longer job processing time (i.e., makespan \( T_{MS} \)). To serve multi-jobs, the scheduler maintains an opportunistic environment and uses node availability rates and pairwise available correlations to improve fairness [28]. They proposed to predict task time complexity and monitor the storage location based on the fair scheduling algorithm to decide whether to reserve an available slot or not for other local storage-supported tasks to reduce data access time. A dynamic predictive execution algorithm [29], [30] was proposed, in which it monitors delayed tasks and workers for task re-assignment to avoid possible longer makespan. A Hybrid SMRP algorithm that utilizes dynamic priority and localization techniques was proposed to increase data locality rate and reduce completion time [31]. It is 2.19% faster than the First-In-First-Out (FIFO) scheduler and 0.78% faster than the Fair scheduler (FS). Although available scheduling methods can be utilized, how to perform cloud-based video transcoding has not been well addressed, especially when the clients request to access segments of not yet transcoded tasks. In general, maintaining highly load-balanced operations when performing resource allocation and task assignment would yield high processing speed. It has to monitor the worker’s processing status and avoid excessive use of a particular worker during processing jobs.

In this research, we study how to utilize the resources of a cloud to process a video transcoding job, \( J \), which requires high time-complexity video processing and larger data segment access. We first investigate how to best partition an input video file, \( F_{in} \), to minimize the \( T_{MS} \) of a \( J \). We proposed to formulate the \( T_{MS} \), including packet upload time, concurrent processing time, and video packet processing time, under a specified cloud computation system configuration, by a mathematical model and derive an optimal partition size, \( z \), that can yield the minimum \( T_{MS} \). Secondly, in addition to determining the best \( z \), we also investigate multi-layer file partition methods [25] to improve cloud video processing efficiency. With the best \( z \) as one system parameter for configuration, we conduct experiments of cloud video transcoding using different task scheduling algorithms under a heterogeneous \( \mathbb{W} \) to design the best scheduler. Thirdly, for practical cloud transcoding operations, the online progress may not meet the task scheduling script that records the task assignment order and which worker to assign to. Under this condition, one task scheduler has to compromise between content complexity, i.e., task time-complexity, and system status, i.e., task processing progress and resource utilization status, both of which can be considered as time-varying variables, to best utilize computing resources. Fourthly, we investigate how to best predict the task processing time, which is found to dominate the performance of the online scheduler script, and perform cloud scheduling for load-balanced operations. Finally, combining all the factors, file partition size, task complexity prediction, scheduler script, and status of computing resources, we proposed a Content-Aware Scheduler with Dynamic Allocation for Slot based on refined task complexity prediction through Neural Network models denoted as DASCASNN.
In what follows, section II introduces the research backgrounds. Section III discusses cloud video transcoding operations, including determining the file partition size, ordering task assignment, and load-balancing control of cloud computation, and their impacts on video transcoding efficiency. The framework and operations of the proposed DASCASNN system are described in section IV. Section V is the experimental study. Section VI concludes this paper.

II. BACKGROUND

A. CLOUD COMPUTING

The system control flow for performing cloud transcoding is described with the aides of Fig. 1. The processing steps after the user submits a job can be carried out in three stages: (1) Initialization; (2) Scheduling; (3) Execution. At the initialization stage, the system partitions the input video file \( F_{in} \) into segments, creates directories, and uploads files to the Hadoop Disk File System (HDFS) for concurrent processing. For live-streaming input, the input is a sequence of segments each comprising a few GOPs. Some initialization procedures have to be carried out before the JobClient sends a job to the JobTracker, such as fetching the Job ID, creating HDFS directories, and transmitting the required JAR (Java Archive) application packets to the distributed file system. It then submits the Job to the \( \psi_J \) through the Remote Procedure Call (RPC) protocol.

At the scheduling stage, when the \( \psi_J \) receives a \( J \) submitted by the JobClient, it creates a JobInProgress (JIP) to check user access right of queues and whether the allocated memory space can afford to execute the job. The TaskScheduler, in addition to assigning tasks to workers according to their resource utilization status, acts as a load balancer. FIFO, FS, and Capacity Scheduler (CS) are widely used schedulers. When the \( \psi_J \) receives a \( J \), it uses three description layers, i.e., JIP, TaskInProgress (TIP) and TaskAttempt (TA), to trace the task execution progress. The \( \psi_J \) creates a JIP to monitor the execution stage.
the overall Job processing progress and creates several TIPs to trace the task execution progress. When one task execution fails, the $\psi_j$ will restart it again until the task is finished or too many restarting times are detected. When all TIPs are finished, the corresponding JIP can be marked as a success. When the assignTasks function gets tasks ready for execution, it creates a task list for the $J$ represented as $T(J) = \{t_1, t_2, \ldots, t_N\}$, for the TaskScheduler to assign tasks through the RPC function.

At the execution stage, the TaskScheduler informs the corresponding TaskTracker, denoted as $\psi^J_T$, through the heartbeat, i.e., HeartBeatResponse. In general, the $\psi^J_T = \psi_T(w_j)$ monitors the task execution progress of a $w_j$. When a $\psi^J_T$ receives the HeartBeatResponse and finds a new task assignment, $t_i \in T$, it invokes the task execution of the corresponding file $f_i \in \mathbb{F}$. For one $t_i$, it creates a local working directory to store the uncompressed data from the JAR file and creates a TaskRunner to invoke a JavaMachine to execute the $t_i$. Each $\psi^J_T$ reports task execution status to the $\psi_j$ such that the $\psi_j$ can monitor the progress of all $t_j$s. When there is a slot released from a $w_k$ that finishes a task execution, the $w_k$ will inform the $\psi_j$ through the heartbeat, by which the $\psi_j$ can assign a new $t_i$ to the $w_k$. The proposed cloud-based streaming platform is developed based on this framework.

**B. HADOOP SYSTEM RESOURCE MANAGEMENT**

Denote an allocated $k$-th slot in a $w_j$ as $s^J_k$ that can be considered as a certification for the $w_j$ to execute tasks. In a heterogeneous $\mathbb{W}$, powerful $w_j$s can promise more slots to accommodate and execute more tasks [32]. How many slots can be allocated for one $w_j$ depends on how many tasks it can execute concurrently. In a Hadoop system, each slot $s^J_k$ is configured to be with Map or Reduce functions. For cloud computing, how to allocate available computing resources to process jobs efficiently is a task assignment problem. It has to handle the resource utilization of a heterogeneous $\mathbb{W}$, monitor CPU utilization rates of workers, $P_U(w_j)$s, and predict task complexity, $T_c(t_i)$, to perform task assignments. However, it's NP-hard to perform resource allocation and task scheduling for multi-target optimizations.

For system configuration, the processing power of the least capable worker, denoted as $p_a$, is used as the slot quantization unit. Denote the processing power of a $w_j$ as $P(w_j)$. The system calculates the quantized worker power, $P(w_j)/p_a$, to allocate a worker-affordable number of slots to each $w_j$, denoted as $n_j(w_j)$.

This slot allocation as a system configuration helps transform one heterogeneous $\mathbb{W}$ to the one with homogeneous slot units, which can make task scheduling easier. After the pre-configuring, the system will not change the $n_j(w_j)$. This static slot allocation approach may occasionally make the $P_U(w_j)$ too high or too low. A Self Adaptive Hadoop Scheduler (SAHS) [17] is designed to maximize the $P_U(w_j)$. We modified the SAHS and designed the control steps as: (1) it executes a resource monitoring procedure to monitor the $P_U(w_j)$; (2) it informs the $\psi_j$ of $P_U(w_j)$ through the heartbeat and determines to increase or decrease the $n_j(w_j)$. The resource monitoring procedure is carried out through the RPC function for the $\psi_j$ to communicate with $\psi^J_T$s. Each $\psi^J_T$ executes the RPC procedure periodically to send task execution progress and $P_U(w_j)$ to the $\psi_j$, which can assign tasks to $w_j$s according to the reported status information. The communication and interaction operations act like the human heartbeat that helps the system: (1) make sure which $\psi^J_T$ is active; (2) enable the $\psi_j$ to update status parameters of all $w_j$s; (3) assign tasks to $w_j$s with reference to the $P_U(w_j)$ status. Each $\psi^J_T$ periodically reports worker processing status to the $\psi_j$, which comprises new task assignments or slot allocation operations. The $\psi_j$s can keep through heartbeat the most updated information from $\psi^J_T(w_j)$s instead of tedious polling operations. The JIP has to record static and dynamic job processing status parameters. The former comprises the $n_j(w_j)$ and memory space needed for the map tasks. The latter are the numbers of working and finished map tasks. The $\psi_j$ is designed to monitor the task processing progress and calculate the $P_U$, and transfer this data to the $\psi_j$. To achieve high $R_U$ and maintain load-balanced system operation, the task scheduler has to dynamically reallocate slots, with reference to $T_c(t_i)$s of different job queues. In this research, we proposed to derive and calculate a task partition size, and justify the result through experiments, based on which a task scheduler and a dynamic slot allocation algorithm can be developed to improve the overall cloud transcoding efficiency.

**C. TASK ASSIGNMENT**

Scheduling algorithms [33] for cloud computing are firstly reviewed for comparisons. The FIFO scheduler processes a job queue sequentially. When utilized in a shared $\mathbb{W}$, the FIFO is inefficient because large jobs will prevent executions of small jobs. Under this condition, adopting the CS or the FS helps yield higher $R_U$. When adopting the FS, no predefined capacity would be reserved for any job queues, and it can maintain balanced resource utilization among jobs. In comparison, the CS can timely process small jobs and can achieve high $R_U$. The CS allows several job queues to share a $\mathbb{W}$ and configures each queue to utilize a predefined capacity of the cluster.

A queue elasticity [34] function is designed for the CS to utilize resources more than its capacity. In the Hadoop system, one task assignment is executed with three steps: (1) When one $\psi^J_T$ detects available map slots in $w_j$, i.e., $n_j^{map}(w_j) > 0$, it informs the $\psi_j$ through the heartbeat; (2) The TaskScheduler selects one job from a queue and selects one task for the $\psi^J_T(w_j)$ to execute required procedures; (3) Assign the task for processing through the FS or others. The concurrently processed tasks from one job may be subject to unpredictable procedure errors, due to either computation or transmission malfunctions, which would reduce the processing speed and extend the job completion time. A speculative execution optimization technique [38] is adopted to eliminate
the artifact of job disorders. It can predict the task processing
time and assign the task to the \( w_j \) with the earliest finish time.

**D. VIDEO CODING STRUCTURE**

For video transcoding, re-encoding a decoded video is the
simplest approach but the most time-consuming. As the
motion estimation and coding mode selection of H.264 account for 66% of the overall video encoding time [35],
re-using these coding parameters to perform transcoding is
widely adopted. Under this condition, the transcoding time
would be independent of video content complexity. Video
is coded with a Group Of Picture (GOP) structure that
comprises frames with type I (Intra-coded), P (predictive-
coded), and B (bidirectional-predictive coded). It has to part-
tition one video file along the GOP boundary (i.e., closed
GOP) such that partitioned packets can be processed inde-
pendently and concurrently among different \( w_j/s \). If not, it is
called open GOP and requires accessing data across \( w_j/s \) for
video coding/decoding. For the example shown in Fig. 2, the
\((k-1)\)-th GOP, GOP\((k-1)\), is a closed GOP inside which every
image can be processed independently. The GOP\((k)\) is an
open GOP, and it needs to access the frame P\(_{13}\) in the
GOP\((k-1)\) and I\(_{16}\) in the GOP\((k)\) to transcode/process the B\(_{14}\)
and B\(_{15}\) frame images. In partitioning video files, the size of
segmentation may be determined by video length or packet
size. To solve the open-GOP problem mentioned above, the
system can be designed to: (1) transfer the P\(_{13}\) frame to the
worker that processes GOP\((k)\); (2) copy the P\(_{13}\) frame to
GOP\((k)\); (3) copy the GOP\((k-1)\) to the packet that comprises
GOP\((k)\). Although these methods help enable independent
GOP processing, the burden of transmission bandwidth and
processing power increases. In general, it uses a constant
partition size for efficient packet transmission. We proposed
to, before the partition boundary, include the first I frame
of the next GOP in the current partition. This partitioning
method enables independent processing for cloud computing.

We conducted an experiment to perform transcoding on
segments comprising video data with 90–95 secs. of play-
back times, and it showed nearly uniform processing time,
as shown in Fig. 3(a). The other way around, video com-
plexity signifies no direct correlation with transcoding time
in this experiment. We can assume that the transcoding times
for packets with the same video playback time are the same.

Partitioning one video bitstream into 16 MegaBytes (MB)
segments, each one comprises a different number of video
frames, and the transcoding time is observed to be linearly
proportional to the video length in time, as shown in Fig. 3(b).
We can assume that, for videos with the same resolution,
the transcoding time would be proportional to the number
of frames in a video segment/packet. This experiment on
transcoding time vs. video length relation helps justify this
assumption.

**III. CLOUD VIDEO TRANSCODING**

Since the video data amount is much larger than that of other
data types and the time-complexity of video data process-
ing is non-homogeneous, it has to manage load balancing
operation to speed up the processing. How to best partition
a big video file is described in sec. III-A. How to achieve
load-balanced system operation to speed up processing is
discussed in sec. III-B

**A. FILE PARTITION SIZE AND EFFICIENCY**

Assuming that a \( \mathcal{W} \) comprises \( N_w \) workers and a submitted
J with a video file \( F_{in} \) is partitioned into \( N_t \) segments, the
cloud computing efficiency under different \( N_w \) and \( N_t \) had
been investigated [25]. In general, the \( N_w \) is fixed, and the
computation load can be balanced when \( N_t \geq N_w \) and
(\( N_t \mod N_w \) = 0 under the environment of processing tasks with
homogeneous time complexity by a homogeneous \( \mathcal{W} \). How-
ever, this homogeneity assumption does not hold for practical
cloud video processing: (1) When \( N_t = N_w \), the startup time
and packet transmission time would be minimum. However, experiments showed that the entire transcoding time would be much longer, in that same size video segments require different processing times and early finished tasks have to wait for the latest one to finish the entire job; (2) When $N_t > N_w$, the partitioned video segment size is smaller and requires a shorter processing time. Under this condition, task scheduling and assignment can be performed with more flexibility to prevent workers from being idle. However, adopting smaller partition sizes will result in more task initialization times and overhead packet transmission time.

Let $l_p$ denote video length in time for one packet and the transcoding time for one packet, $T_p$, can be expressed as:

$$T_p = s \cdot l_p,$$  \hspace{1cm} (1)

where $s$ denotes the efficiency of the worker in performing transcoding. The RAM size and CPU core number also affect transcoding efficiency. In the Hadoop system, the first batch of video segments has to be transmitted to workers before transcoding. The operation flow of a video transcoding job.

The third one is the packet upload transmission time of all the required overall startup time under the $N$ workers, which is independent of overall transcoding time (cf. eq. 1) divided by the number of workers. This system has to maintain highly load-balanced operations to minimize the total transcoding time. Eq. 2 demonstrates a simple cloud processing time model to help explain the operations. By computing $z$ with a parameter set: $BW = 20$ Mbps, $Z = 1024$ MB, $N_w = 4, T_S = 5$ sec., it yields $z = 80$ MB. Different computer clusters with different processing power, data access rates, and bandwidth will demonstrate different processing efficiency. It needs to conduct experiments on practical cloud video transcoding to evaluate the efficiency under a certain $z$, which will be discussed in Sec. V-A.

B. TASK ASSIGNMENT FORMULATION

As video segments partitioned by $z$ comprise different numbers of video frames, it would lead to load-unbalance and more processing time. As shown in Fig. 5(a), tasks in $w_1$ require longer execution time and tasks in $w_2$ and $w_3$ have to wait for $T_f$ to finish the job. Another example is shown in Fig. 5(b). When the numbers of tasks assigned to different workers are different, workers with fewer tasks have to wait for the one with more tasks to finish the job. It needs to improve the task assignment method to solve this inefficient resource utilization problem.

Eq. 4 can be used to find a proper partition size, $z$, for different system setups. By setting $z < 64$ MB, each video segment can be stored within one HDFS block. However, the video lengths measured in time of the same size segments would be different. The system has to maintain highly load-balanced operations to minimize the total transcoding time. Eq. 2 demonstrates a simple cloud processing time model to help explain the operations. By computing $z$ with a parameter set: $BW = 20$ Mbps, $Z = 1024$ MB, $N_w = 4, T_S = 5$ sec., it yields $z = 80$ MB. Different computer clusters with different processing power, data access rates, and bandwidth will demonstrate different processing efficiency. It needs to conduct experiments on practical cloud video transcoding to evaluate the efficiency under a certain $z$, which will be discussed in Sec. V-A.

\begin{equation}
T_MS = s \cdot (\frac{L}{N_w}) + T_S \cdot (\frac{Z}{z \cdot N_w}) + N_w \cdot (\frac{z}{BW}), \hspace{1cm} (2)
\end{equation}

where $L$ is the video length in time of an input video file $F_{in}$, $N_w$ denotes the number of overall workers in a $W$, $T_S$ denotes the task startup time, $Z$ denotes the input file size, $BW$ denotes the uplink bandwidth and $z$ is the partition size that is a variable to be observed. The first term in eq. 2 is the overall transcoding time (cf. eq. 1) divided by the number of workers, which is independent of $z$. The second one denotes the required overall startup time under the $N_w$ worker setup. The third one is the packet upload transmission time of all tasks.

In eq. 2, different $z$ values will lead to different $T_MS$. To minimize the $T_MS$, we perform partial differential operations on eq. 2 concerning $z$:

$$\frac{\partial T_MS}{\partial z} = \frac{N_w}{BW} - T_S \cdot \frac{Z}{N_w \cdot z^2} = 0.$$

Further derivations yield the following one:

$$z = \sqrt[3]{T_S \cdot Z \cdot BW \cdot N_w}, \hspace{1cm} z <\text{ block size}. \hspace{1cm} (4)$$

FIGURE 4. The operation flow of a video transcoding job.

FIGURE 5. Examples of load-unbalanced processing due to: (a) different task execution time; (b) different number of assigned tasks.
The finish time of a $t_i$ processed by a $w_j$ is computed as:

$$T_F(t_i, w_j) = T_F(t_x, w_j) + T_E(t_i, w_j),$$

where $t_x$ denotes the task being processed by the $w_j$ right before assigning the $t_i$. Let $U$ denote the set of tasks not yet assigned, and the minimum completion time (MCT) of a $t_i$ can be represented as:

$$T_{\text{min}}(t_i) = T_F(t_i, w_k), \quad \text{where } k = \arg\min_{j \in \{1, 2, \ldots, N_u\}} T_F(t_i, w_j).$$

In this MCT algorithm, it sequentially selects one task $t_i \in U$ and finds a $w_j$ that yields the minimum $T_{\text{min}}(t_i)$ to process the $t_i$. This MCT algorithm is designed to finish a job as early as possible. However, it cannot well process a video transcoding job with non-uniform task complexity. To improve the MCT efficiency in processing a video transcoding job, a Max-MCT [36] algorithm is proposed in which the next task to be assigned is $t_k$, where

$$k = \arg\max_{t_i \in U} T_{\text{min}}(t_i) = \max_{t_i \in U} \min_{w_j \in W} T_F(t_i, w_j).$$

In eq. 8, the Max-MCT algorithm is designed to estimate finish times of all tasks $t_i \in U$, i.e., $T_{\text{min}}(t_i)$s, among which the maximum one is selected for execution first. In most cases, high complexity tasks will be assigned first with this Max-MCT algorithm. The Min-MCT algorithm is carried out similar to eq. 8 except that it will select a $t_k$, where

$$k = \arg\min_{t_i \in U} T_{\text{min}}(t_i).$$

The makespan to finish the job $J$, denoted as $T_{\text{MS}}(J)$, can be obtained by

$$T_{\text{MS}}(J) = \max_{i=1, 2, \ldots, N_c, j=1, 2, \ldots, N_u} \{T_F(t_i, w_j)\}.$$  

The resource utilization rate of a $w_j$, $R_U(J, w_j)$, can be calculated by

$$R_U(J, w_j) = \frac{\max \{T_F(t_i, w_j)\}}{T_{\text{MS}}(J)},$$

and the system resource utilization rate in processing a $J$ is calculated by

$$R_U(J) = \frac{\sum_{j=1}^{N_u} R_U(J, w_j)}{N_w}.$$  

The task scheduling for cloud computing can be considered as the Knapsack Problem [37], in which the computer cluster resource is fixed and it needs efficient scheduling methods to increase the $R_U$. This sort of Knapsack problem that solves combinational optimization is NP-hard, which can be solved by a dynamic programming algorithm. However, the estimation error of task complexity makes it difficult to be utilized in practical applications.

In short, it does not take the entire set of task complexity into consideration such that several high complexity tasks are assigned first, in which the complexity is for this MDLBA experiment, abbreviated as MDMCT. High complexity tasks are assigned first, in which the complexity is determined by video length in that segment/packet. The size ratio between large and small segments is around 1/4 ~ 1/5.

Practical MCT and MDMCT task assignment examples are demonstrated below. Assume $N_w = 3$ and $N_t = 12$ and individual task execution times are shown in Table 2, in which there are 9 high and 3 low complexity tasks. For the MCT, it sequentially assigns tasks according to eq. 7 and the process is shown in Fig. 7. As shown, after assigning all high complex tasks, $w_2$ and $w_3$ have to wait for $t_9$ in $w_1$ to finish the job (Fig. 7(e)), and it suffered severe convoy effect even after low complexity tasks, $t_{10}$, $t_{11}$, and $t_{12}$, are assigned to $w_1$, $w_2$, and $w_3$, respectively, as shown in Figs. 7(f)(g)(h). It shows that the low complexity tasks cannot well eliminate the convoy effect due to uneven execution times of high complexity ones. In short, it does not take the entire set of task complexity into consideration such that several high complexity tasks are assigned to the same worker.

The 1st operation of the MDMCT is the same as that of the MCT. Its 2nd operation for $t_6$ is shown in Fig. 8(b). Fig. 8(f)
FIGURE 7. The MCT task assignment: status after assigning: (a) $t_1$; (b) $t_2$; (c) $t_5$; (d) $t_4$; (e) $t_5 \sim t_9$; (f) $t_{10}$; (g) $t_{11}$; (h) $t_{12}$.

FIGURE 8. The high complexity first MDMCT scheduler: status after assigning: (a) $t_1$; (b) $t_6$; (c) $t_4$; (d) $t_7$; (e) $t_9$; (f) $t_5$, $t_3$, $t_8$ and $t_2$; (g) $t_{10}$; (h) $t_{11}$ and $t_{12}$. 
shows the assigned tasks with high complexity. As the task complexity has been considered during scheduling, it can avoid assigning high-complexity tasks to the same worker for load balance. Low complexity tasks, $t_1$, $t_2$, and $t_3$, were then assigned to $w_1$, $w_3$, and $w_2$, respectively, as shown in Fig. 8(h). The $T_{MS}$ and $R_U$ before and after assigning low complexity tasks denoted as method$^1$ and method$^2$, respectively, are shown in Table 3. In comparison, the MDMCT demonstrates better load-balanced operations and a shorter $T_{MS}$ that is 11% smaller than that of the MCT. In terms of $R_U$, the MDMCT demonstrated higher $R_U$ after assigning low complexity tasks. The MCT demonstrated higher $R_U$ after assigning shorter segment tasks. The MDMCT yields less convoy effect and better load-balanced operation. In general, adopting a smaller $z$ yields longer $T_{MS}$ in that it needs to initialize more tasks. Also, adopting a smaller $z$ can effectively reduce the convoy effect and yield higher $R_U$ before processing the 2nd partition segments [26]. When the 2nd partition segments are involved, it can increase $R_U$ to more than 98% for all $z$s.

### D. Load Balance Performance

We take practical examples of the Min-MCT and the Max-MCT algorithms to demonstrate and analyze their capability in load balancing control. In the Min-MCT, it estimates MCTs of all $t_i$, $t_i \in U$, and record them in a table. After selecting the task whose MCT is minimum by eq. 9, it updates the MCT table for the next task assignment. One Min-MCT example is shown in Fig. 9(a), in which the Min-MCT first selects $w_1$ to process task $t_1$ in that $T_F(t_1, w_1) = 3$ is minimum. After updating the MCT table, since $T_F(t_2, w_1) = 11$ is minimum, $t_2$ is assigned to $w_1$. It then selects $t_3$ for $w_1$ through the same operation steps. The resultant task execution time on different workers is shown at the right of Fig. 9(a), which demonstrates the task whose MCT is minimum by eq. 9, it updates the MCT table for the next task assignment. One Min-MCT example is shown in Fig. 9(b), in which it assigns $t_1$, $t_2$, and $t_3$ to $w_1$, $w_2$, and $w_3$, respectively, and demonstrates better load-balanced operations as compared to the Min-MCT example. Their $T_{MS}$s are 23 secs. and 16 secs, respectively.

The three scheduling methods, MCT, Min-MCT, and Max-MCT, are carried out for comparisons under the same system setup shown in Table 5 but are reconfigured to one master and four workers with one slot each comprising two cores. It adopts two-layer packet sizes, which are 60MB and 15MB, respectively. The $T_{MS}$ and $R_U$ performance evaluation is provided in Table 4. As shown, the $R_U$ of MCT$^1$ and MCT$^2$ are 85.5% and 92.5%, respectively. The proposed MDMCT$^1$ can help eliminate the convoy effect and achieve a higher $R_U = 96.7%$. Both MCT$^2$ and MCT$^1$ did not shorten the $T_{MS}$ from the two-layer task assignment, but they increase the $R_U$, i.e., 85.5% to 90.3% and 92.5% to 97.7%, respectively. The MDMCT$^2$ increases the $R_U$ to be as high as 98.6%.

As the MCT considers only the current task complexity and selects a $w_j$ that can earliest finish the task, it will lead to load-unbalanced operations, especially for the second half part of the job. The convoy effect cannot be eliminated by adopting the two-layer task assignment procedure. The Min-MCT and the MDMCT select from all unused ones, $t_j \in U$, the lowest and the highest complexity tasks, respectively, for assignment and can yield load balancing operations and less convoy effect. The Min-MCT selects the lowest complexity task first for assignment, such that higher complexity tasks assigned later may lead to severe convoy effect, which can be well eliminated by the second layer task assignment. The MDMCT selects higher complexity tasks first for assignment, and the later lower complexity task assignment would yield less convoy effect. By utilizing the two-layer task assignment method, in addition to reducing the convoy effect, it can increase the $R_U$ and shorten the $T_{MS}$. The MDMCT$^2$ is selected as the baseline for performing cloud computing in the following sections.

### IV. Content-Aware Scheduling with Dynamic Slot Allocation

The control target of a cloud task scheduler is to shorten the entire job processing time. As the $P_U(w_j)$ is time-varying, the processing efficiency of this static slot allocation approach
time, i.e., and decreases the processing speed. When the processing occasionally require longer execution times, such that the operation, and uneven resource allocation, some tasks may be further improved. By monitoring the instant $P_U(w_j)$s and task complexity through the heartbeat, the system can dynamically adjust the $n_i(w_j)$s to maintain highly load-balanced operations.

A. COMPLEXITY AWARE SCHEDULER (CAS)

As the processing time of tasks from the same job would be different, due to procedure error, load-unbalanced operation, and uneven resource allocation, some tasks may occasionally require longer execution times, such that the scheduler frequently invokes speculative execution functions and decreases the processing speed. When the processing progress of one task is delayed 20% more than its counterparts, it is considered as falling behind. Under this condition, the worker, say $w_k$, that is found to yield the earliest finish time, i.e., $T_F(t_i, w_k)$ through eq. 7, would be selected to process the task. It’s better to predict the task complexity for a $t_i$, $T_0(t_i)$, before the speculative execution, such that the system can perform the dynamic slot adjustment procedure.

Both Max-MCT and MDMCT algorithms were developed by assuming the computing powers of $w_j$s $\in W$ are homogeneous. They are efficient since all task processing time, $T_F(t_i, w_j)$, can meet the task assignment schedule. To process transcoding jobs in a practical computer cluster, we proposed a Complexity-Aware Scheduler (CAS) which was developed and improved from the MDMCT, the FS, and a LATES [38]. For the LATES, the new speculative execution function is designed to predict the overall job completion time, by which it reorders task assignment priorities for speedup. The CAS is similar to the Max-MCT, and it’s a dynamic counterpart of the static Max-MCT one. In other words, the CAS is designed to practically assign tasks according to the Max-MCT schedule. As shown in Fig. 10, the CAS assigns high complexity tasks to different $w_j$s for load-balanced system operation, under which low complexity tasks will automatically be assigned at last to eliminate the convoy effect as the MDMCT does. The Max-MCT can be considered as the benchmark of the CAS, and the MDMCT acts as an improved Max-MCT. However, the CAS can be easily implemented in a practical heterogeneous $W$. Experiments showed that the MDMCT outperforms CAS a little in $T_{MS}$ and $R_U$ performances, which will be presented in section V-C.

In section III-D, it shows that the MDMCT procedure can effectively eliminate the convoy effect and shorten the $T_{MS}$ of processing a $J$. From the viewpoint of processing a specific $J$ or a batch of $J$s, it is an efficient scheduling method of cloud computation for processing off-line files. For live-streaming services that demand low delay playback, the task set for scheduling should be kept small, i.e., the buffering window size for task assignment. Also, there’s no specific file boundary for live-streaming services. Under this condition, the MDMCT procedure becomes trivial, while the CAS can still efficiently process the task scheduling across the boundary of the buffering window. The CAS can be considered as a fine-granular version of the multi-layer task assignment method, MDMCT. In summary, considering video content complexity, both MDMCT and CAS are efficient in speeding up the transcoding process, but the latter is more suitable for practical live-streaming applications.

B. COMPLEXITY-AWARE SELF-ADAPTIVE SCHEDULER

In configuring the Hadoop, each $w_j$ is allocated a certain number of certificated slots, $n_i(w_j)$, for executing tasks, in which all $n_i(w_j)$s will not be changed during execution. As both task complexity and worker processing power are not homogeneous, the scheduling algorithm has to dynamically adjust the $n_i(w_j)$ to accommodate this mutual-way variation. A Self Adaptive Hadoop Scheduler (SAHS) [17] was proposed to increase/reduce the $n_i(w_j)$ of each worker depending on its $P_U$ over the run time. For dynamical slot allocation, it has to set up an initial maximum and minimum number of slots for each $w_j$, $n_i^{Max}(w_j)$ and $n_i^{Min}(w_j)$, and a threshold value, $Th_{PS}$, for the $P_U(w_j)$. In general, the system sets the initial $n_i(w_j)$ based on the number of cores that a $w_j$ has.

When setting a large $Th_{PS}$, one $w_j$ is likely to be allocated more slots while $P_U(w_j) < Th_{PS}$ which results in a heavy workload. On the contrary, a smaller $Th_{PS}$ will prohibit the system from utilizing available slots and the overall $P_U$ will also be lower. Experiments verified that setting $Th_{PS} = 80\%$ yields good cloud computing performances [18]. When the $\psi_f$ receives the status information from a $\psi_T(w_j)$, if $P_U(w_j) < Th_{PS}$, it checks whether: (1) there are available map slots left in $w_j$, i.e., $n'^{free}_i(w_j) \neq 0$; (2) current map slots are fewer than its upper limit, $n_i(w_j) < n_i^{Max}(w_j)$; (3) there are unassigned map tasks, i.e., $|U| \neq 0$. If all these conditions are satisfied, it allocates one map slot and assigns one map task to this $\psi_T(w_j)$ to improve the $P_U(w_j)$ and shorten the $T_{MS}$, as demonstrated in the lower right block of Fig. 11. When $P_U(w_j) > Th_{PS}$, it checks whether $n'^{free}_i(w_j) > 0$ and $n_i(w_j) > n_i^{Min}(w_j)$. If both conditions are satisfied, it will delete one map slot from the $w_j$ to avoid load unbalance. Fig. 11 shows the flowchart of the modified SAHS control steps.

For both MDMCT and CAS, the system measures the complexity of a task, $T_0(t_i)$, by the number of coded frames within the segment and creates a scheduling script for all tasks of the submitted $J$. According to this scheduling script, it assigns all tasks to the $W$, which is supposed to help
efficiently process the $J$ if all tasks are processed and finished on time. As the $P_U(w_j)$ is time-varying, the task processing time, $T_E(t_i, w_j)$, may not meet the schedule such that the job processing will exceed the scheduled $T_{MS}$. In addition to analyzing video content complexity, the SAHS monitors the $P_U(w_j)$s through heartbeats to adjust the $n_s(w_j)$ for better load-balanced operations. We call the method that combines SAHS with CAS as Dynamically Adjusting Slot number with Complexity-Aware task Scheduler, abbreviated as DASCAS. The diagram of the proposed DASCAS is shown in Fig. 12.

C. PREDICTION COMPLEXITY BY ARTIFICIAL NEURAL NETWORK

In the DASCAS experiment, we used the number of video frames in a segment to represent the task complexity $T_c(t_i)$. Experiments for multi-job schedules revealed that the task processing time depends not only on the number of video frames. Beyond the playback time of one segment ($T_p$), other video coding parameters, such as resolution, encoded bitrates, numbers of I, B, and P frames in a GOP, denoted as $n_I$, $n_B$, $n_P$, respectively, may also affect the task processing time. In general, the actual task processing time correlates with each parameter, but video coding configuration parameters also affect each other. To precisely predict the task processing time from different video coding parameters, these correlations have to be considered as a whole, which is quite complicated work. The artificial neural network model (ANN) can implement massively parallel computation for classification and prediction, which can effectively solve this kind of problem. By setting proper numbers of layers, neurons, and learning iteration steps, it can effectively combine different correlations as a whole to perform prediction or classification precisely [39]. We proposed to take these six parameters, $P_{ANN} = (\text{resolution, bitrates, } n_I, n_B, n_P, T_p)$, as the input for an ANN to output the predicted task processing time. The parameter set $P_{ANN}$ can be extracted from video codestreams by the FFmpeg video processing tool, and the first $P_{ANN}$ parameter, resolution ($H \times W$), is the number of overall pixels in a frame.

V. EXPERIMENTAL STUDY

A. VIDEO FILE PARTITION

An experiment of cloud video transcoding with different $z$s was conducted to find the best operation parameter. One server computer with 2 CPUs, 96G Ram, and 24 cores, whose model is Intel Xeon E5-2620/2.0GHz, is used as our system platform. We set up 1 master, 3 clients, and 4 slave workers that comprise 2, 3, 4, and 6 CPU cores, i.e., $\mathbb{W} = \{w_1(2), w_2(3), w_3(4), w_4(6)\}$, where $w_j(s)$ means the $w_j$ affords to provide $s$ slot certifications, or another simple notation $|\mathbb{W}| = \{2, 3, 4, 6\}$. The VMware is installed on the server to provide the required computer cluster. To perform video transcoding, an FFmpeg video toolbox is adopted. Table 5 shows the experimental setup. The master acts as the central management unit which handles the HDFS NameNode and JobTracker function. Besides, the master monitors the worker’s progress and allocates resources. The client issues requests to the cloud system and uploads the
required video file to the HDFS. It also handles the final output file which can be DASH compatible for streaming applications.

The first test video is *Batman* with resolution $1920 \times 800$, $1598$ Kbps, $1h28m11s$ long, and $1024$MB size in H.264 format, and the output format is H.264 $480 \times 360@172$Kbps. The MDMCT scheduler was adopted for this experiment, which was executed six times to yield the average practical transcoding time. The estimated processing time by eq. 2 is used for reference. By setting $s = 0.72$, $T_{MS} = 5$ secs. and $BW = 20$ Mbps, and using 4 workers, the practical (solid square) and estimated (dashed line) $T_{MS}$ are plotted in Fig. 13(a). As shown, adopting a smaller $z$ resulted in more task initialization times and a longer $T_{MS}$. The $T_{MS}$ decreases when the system sets a larger $z$. In addition to reducing the task initialization time, setting a larger $z$ requires a longer transmission time such that the probability of transmission interruption will also increase. The two factors cancel out each other and eventually, the $T_{MS}$ reaches stable. For even larger segments, the transmission time increases and the $T_{MS}$ increases slightly, as shown in Fig. 13(a).

The second test video is *MoneyBall* whose experimental result is shown in Fig. 13(b). As shown, the time-size plot demonstrates a similar relational curve to that of the first one. Since the file size, $1610$ MB, of the second video is larger than that of the first one, setting a smaller $z$ will increase both task initialization times and the $T_{MS}$. The $T_{MS}$ increases when the system sets a larger $z$. In addition to reducing the task initialization time, setting a larger $z$ requires a longer transmission time such that the probability of transmission interruption will also increase. The two factors cancel out each other and eventually, the $T_{MS}$ reaches stable. For even larger segments, the transmission time increases and the $T_{MS}$ increases slightly, as shown in Fig. 13(a). The second test video is *MoneyBall* whose experimental result is shown in Fig. 13(b). As shown, the time-size plot demonstrates a similar relational curve to that of the first one. Since the file size, $1610$ MB, of the second video is larger than that of the first one, setting a smaller $z$ will increase both task initialization times and the $T_{MS}$. Under this condition, the best $z$ is around $100$MB. Another experimental result, shown in Fig. 13(c), also demonstrated similar results in which the system adopted the FS scheduler and was configured to $|W| = \{1, 2, 3, 4, 4, 4\}$. These experiments suggest that the proper segment size for efficient transcoding is around $64\sim128$ MB.

Experiments revealed that when adopting a small $z$ for one $J$, the system has to perform more task assignment operations and spend more task initialization times [26], i.e., a smaller $z$ leads to a larger $N_t$. Although it can achieve a higher $R_U$, it will increase the $T_{MS}$, i.e., $T_{MS}(14$ MB$) > T_{MS}(30$ MB$) > T_{MS}(60$ MB$)$, where $T_{MS}(z)$ denotes the makespan when the file partition size is set to $z$. By utilizing the two-layer task assignment method, the system not only can yield smaller $T_{MS}$s but also improve the $R_U$ performance when adopting a larger $z$.

### TABLE 5. Experimental setup.

| Computer specs. | Intel Xeon E5-2620, 2.0 GHz, 24 Cores, 96 GB Ram |
|-----------------|---------------------------------------------|
| VMWare          | OS Ubuntu 12.04, Memory 8 GB                |
| Number of VM workers | 1 master, 4 slave workers, 3 clients         |
| Transcoding Software | FFmpeg                                      |
| Test Video A    | Batman                                      |
| Input Video Format | $1920 \times 800$, 1598 Kbps, 1h28m11s, 1024 MB, 264 |
| Test Video B    | MoneyBall                                   |
| Input Video Format | $720 \times 384$, 1491 Kbps, 2h23min15s, 1610 MB, 264 |
| Output Video Format | $480 \times 360$, 172 Kbps, H.264, 324×240, 76 Kbps |

#### FIGURE 13. The estimated (eq. 2) and practical $T_{MS}$s for different $z$s.

### B. PREDICT TASK PROCESSING TIME BY ANN

The target video format of transcoding is set to be the one with resolution $480 \times 360$ and bitrates $176$ Kbps. Nine videos partitioned into 184 segments are used as training samples to extract the $P_{ANN}$ to train the ANN model that comprises six input nodes, one hidden layer with 100 neurons, and one output neuron. Given the $P_{ANN}$ of a partitioned segment, the ANN output neuron will provide the estimated processing time of this segment. Video for training and testing are shown in Table 6. The Bayesian regularized back-propagation model is adopted to train for the ANN parameters of our case. The log-sigmoid function is adopted for input, and the linear transfer function, *purelin*, is used to enable the output to be arbitrary values. The metric for the ANN convergence is Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{\sum_{i=1}^{N} |P_i - \hat{P}_i|}{N} \times 100,$$

where $P_i$ denotes the real output, i.e., the task processing time, $\hat{P}_i$ denotes the predicted output and $N$ is the number of total samples. The MAPE converges to $2.6\%$ after ten
TABLE 6. Nine training videos for the neural network model and four videos for test.

| Train video | Resolution | Rate [Kbps] | File Size [MB] | Duration | Segments |
|-------------|------------|-------------|---------------|----------|----------|
| Ghost       | 1280 x 544 | 1274        | 914.0        | 1h32m    | 13       |
| Harry       | 1920 x 706 | 1920        | 2.9          | 2h15m    | 28       |
| Lawless     | 720 x 400  | 363         | 696.5        | 1h53m    | 10       |
| Basketball  | 720 x 384  | 1599        | 1.7          | 2h15m    | 24       |
| Perl        | 1920 x 1040| 1380        | 1.8          | 1h26m    | 25       |
| Racing      | 1280 x 720 | 1372        | 1.7          | 1h30m    | 21       |
| Stocks      | 1920 x 1080| 1381        | 3.5          | 2h26m    | 41       |
| Gamblers    | 1280 x 720 | 1661        | 875.0        | 1h13m    | 13       |
| Tokyo       | 856 x 480  | 904         | 692.0        | 1h34m    | 10       |

FIGURE 14. Comparisons of task processing time prediction by different methods.

FIGURE 15. The average transcoding time, $T^{MS}_j$, of all scheduling methods.

time reduction rate

$$\hat{\Delta}_T(i, j) = \frac{T^{i, MS}_j - T^{j, MS}_j}{T^{j, MS}_j},$$

where $j$ and $k$ denote the $i$-th and $j$-th scheduling methods to be compared and evaluated, respectively.

Both CS and the FS allocate system resources statically and consider no content complexity, so their average $T^{MS}_j$ are larger. The FS outperforms the CS in the $\Delta_T$ performance in that it manages load balancing control. Let FS(2) denote the configuration that two slot certificates are allocated statically to each worker. Experiments showed that it will lead to load-unbalanced operations. In the FS($p$), it allocates $n_s(w_j)$ according to the power of $w_j$ and the operational $R_{U_j}$ increases and the $\Delta_T = 14\%$. The CAS manages to improve task scheduling performance according to video content complexity, in which high complex tasks will be assigned for processing before low complex ones. Its $R_U$ and $\Delta_T$ are 90.3% and 16.5%, respectively. The MDMCT acts as a benchmark of the CAS, which performs task scheduling based on both complexity-dependent task assignment and worker processing efficiency. Besides, it adopted a two-layer segment size policy to eliminate the convey effect. Its $\Delta_T = 16.9\%$ is larger than that of the CAS (14.3%). The SAHS is designed to improve the processing performance by managing resource management, which can dynamically adjust the number of certificated slots for a worker to increase the worker $R_U$ and the $\Delta_T$, which are 94.8% and 20.6%, respectively. The DASCAS, a combination of CAS and SAHS, yields higher $\Delta_T = 23.8\%$, as it effectively manipulates and utilizes the variation of video content complexity and the $P_{U(j)}$ status to perform task scheduling and assignment. Experiments of the CAS, the SAHS, and the DASCAS showed that precisely handling the variation of both content complexity and system $P_{U(j)}$ status helps much in improving system performance. Experiments also revealed that the task complexity estimation error of the CAS can be partly compensated by the SAHS. However, the performance of the DASCAS algorithm still depends on accurate task complexity estimation to shorten the $T^{MS}_j$. In the DASCASNN, the content complexity is predicted in a more sophisticated way, and it increases the $\Delta_T$ up to 30.4%, which outperforms all other methods, as shown in Fig. 15 and Table 7.
TABLE 7. Comparisons of execution time among different scheduling methods with the CS as the baseline one in computing $\Delta_T$.

| Algorithm | CS | FS(2) | FS(p) | CAS | MDMCT | SAHS | DASCAS(NN) |
|-----------|----|-------|-------|-----|-------|------|------------|
| $T_{MS}$ | 1411 | 1375 | 1223 | 1178 | 1153 | 1119 | 1075(98.1) |
| $\Delta_T$(%) | 0 | 6.5 | 8.9 | 6.6 | 8.2 | 8.1 | 11.1 |
| $RU$(%) | 82.7 | 84.9 | 87.4 | 90.3 | 92.8 | 94.8 | 96.2(99.1) |

Under static configuration, the Hadoop cannot re-allocate slots under load-unbalance conditions, which results in smaller $RU$s. Although the FS(p) is better than the FS(2), all slots may be busy but the average $P_U(w_j)$ is still low. Besides, static configuration allows not to adjust the $n_s(w_j)$s to improve the $P_U$ performance. Under this condition, the system will demonstrate quite different $P_U(w_j)$s, and heavy-loaded $w_j$s require longer task processing time. As compared to the FS(p), the proposed DASCAS/DASCASNN can dynamically adjust the $n_s(w_j)$ of a $w_j$ according to the $P_U(w_j)$ and reduce the $T_{MS}$ to 11.4%(19.1%) smaller, as shown in Table 7.

D. MULTI-USER TASK PROCESSING

We also conduct experiments on scheduling when handling more jobs with heterogeneous content complexity. As shown in Fig. 16(a), the FS(2) demonstrates the longest $T_{MS}$ in all cases. The FS(p) helps to shorten the $T_{MS}$ about 20%(4 jobs) and 6%(8 jobs). The MDMCT loosely reordered the task assignment priority and yielded 24%(4 jobs) and 9%(8 jobs) of $\Delta_T$. Under this condition, the proposed DASCAS/DASCASNN can save about 29%/35%(4 jobs) and 12%/21%(8 jobs) of $T_{MS}$ as compared to those of FS(2). As demonstrated, the $\Delta_T$ becomes smaller when more jobs are submitted for processing. It’s reasonable because the system computing resources become insufficient under this condition. When processing 4 jobs, the total active slots, which are counted to be 20 in this case, have to process the overall 70 tasks. The proposed DASCASNN can better utilize available resources, as demonstrated in Fig. 17(a). For 8 jobs, the processing load of the limited computing resources is doubled, and the operational resource utilization capability of the DASCASNN was limited. Also, the CAS capability in reducing the convoy effect becomes inefficient when more job queues were involved for task assignment. Queuing all job tasks together can help to improve the $R_U$ and $P_U$ performance. The system design policy for multi-job queuing would depend on different application requirements and is beyond the scope discussed in this research.

Another experiment has been conducted to investigate the $P_U(w_j)$s of different algorithms during processing. The four test videos are the same as those listed in Table 6, and the pre-configured CPU cores for workers is $|W| = \{1, 1, 2, 2, 4, 4, 4\}$. The resultant $P_U(w_j)$s along processing time are shown in Fig. 17(a). As compared to the FS(2), the CAS helps to increase the average $P_U$ and reduce the $T_{MS}$ from 1,521 secs. to 1,143 secs. The proposed DASCAS utilized advantages of both CAS and SAHS, and shortened the $T_{MS}$ to 1075 secs, i.e., $\Delta_T = 29.7\%$. By adopting the NN model to precisely estimate the heterogeneous task complexity from different jobs, re-arranging task assignment order, and dynamically adjusting the number of slots for workers, the DASCASNN increased the average $P_U$ to near 100% and yielded the shortest $T_{MS} = 980$ secs. The number of total active slots during the process is shown in Fig. 17(b).

Under static configuration, the Hadoop cannot re-allocate slots under load-unbalance conditions, which results in smaller $RU$s. Although the FS(p) is better than the FS(2), all slots may be busy but the average $P_U(w_j)$ is still low. Besides, static configuration allows not to adjust the $n_s(w_j)$s to improve the $P_U$ performance. Under this condition, the system will demonstrate quite different $P_U(w_j)$s, and heavy-loaded $w_j$s require longer task processing time. As compared to the FS(p), the proposed DASCAS/DASCASNN can dynamically adjust the $n_s(w_j)$ of a $w_j$ according to the $P_U(w_j)$ and reduce the $T_{MS}$ to 11.4%(19.1%) smaller, as shown in Table 7.
In comparison, both FS and FS+CAS prohibit the scheduler to use available CPU resources and yield lower $P_U/s$. The DASCAS can increase/decrease $n_i(w_j)$s of specific $w_j$s to maintain high $P_U$ and highly load-balanced operations. The $n_i(w_j)$s with different $n_i^{max}(w_j)$s during the DASCASNN process are shown in Fig. 17(c), which shows that the DASCASNN can detect and adjust the $n_i(w_j)$ of capable workers, i.e., $w_j(s = 4)$ and $w_j(s = 2)$, to better utilize available CPU resources. The design of dynamic configuration, together with the NN model refined CAS, enable the DASCASNN to better utilize computing resources and effectively eliminate the convoy effect. It yielded the highest $P_U$ and $R_U$ among all the mentioned schedulers.

VI. CONCLUSION

In this research, we proposed a DASCASNN method that can dynamically adjust the number of slots in a worker according to CPU utilization rates and task time complexity to improve cloud computing efficiency. We integrate the cloud concurrent processing framework, video transcoding techniques, and task scheduling algorithms based on neural network models to provide a MPEG-DASH compatible cloud video transcoding/streaming platform. We investigated the impact of video content complexity and file partition size on cloud video transcoding efficiency, based on which a complexity-aware task scheduler was proposed to cooperate with a dynamic worker slot number adjustment algorithm to enhance resource utilization and increase cloud video processing efficiency. Contributions comprise: (1) A mathematical model that relates cloud video concurrent processing, bandwidth, input file size, and partition size was designed to help the system find a task partition size that yields the shortest processing time. (2) We exploited the diversity of video content complexity and time-varying resource utilization status to develop a complexity-aware task scheduler to cooperate with a dynamic worker slot number adjustment method. (3) We proposed to estimate the video transcoding time of tasks with the neural network model to improve the accuracy of the complexity-aware scheduling script, which helped to shorten the overall transcoding time. (4) The proposed DASCASNN improves the load balancing operation efficiency of cloud computing and increases the system resource utilization rate from 84.9% to 98.1%, which can save up to 30.4% of processing time as compared to state-of-the-art Hadoop schedulers. How to utilize deep learning methods to improve cloud computing efficiency is considered as our future research.

REFERENCES

[1] J. Xin, C.-W. Lin, and M.-T. Sun, “Digital video transcoding,” Proc. IEEE, vol. 93, no. 1, pp. 84–97, Jan. 2005.
[2] I. Sodagar, “The MPEG-DASH standard for multimedia streaming over the Internet,” IEEE Multimedia, vol. 18, no. 4, pp. 62–67, Apr. 2011.
[3] A. Verma, X. Llorà, D. E. Goldberg, and R. H. Campbell, “Scaling genetic algorithms using MapReduce,” in Proc. 9th Int. Conf. Intel. Syst. Design Appl., 2009, pp. 13–18.
[4] W. Xu, L. Huang, A. Fox, D. Patterson, and M. J. Jordan, “Detecting large-scale system problems by mining console logs,” in Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Prim., 2009, pp. 117–132.
[5] D. Kesavaraja and A. Shenbagavalli, “Framework for fast and efficient cloud video transcoding system using intelligent splitter and Hadoop MapReduce,” Wireless Pers. Commun., vol. 102, no. 3, pp. 2117–2132, Oct. 2018.
[6] N. J. Navimipour and F. S. Milani, “A comprehensive study of the resource discovery techniques in peer-to-peer networks,” Peer-to-Peer Netw. Appl., vol. 8, no. 3, pp. 474–492, 2015.
[7] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, “Cost-effective task scheduling for executing large programs in the cloud,” Parallel Comput., vol. 39, nos. 4–5, pp. 177–188, 2013.
[8] X. Li, Y. Cui, and X. Xue, “Towards an automatic parameter-tuning framework for cost optimization on video encoding cloud,” Int. J. Digit. Multimedia Broadcast., vol. 2012, pp. 1–11, Jan. 2012.
[9] P. Li, B. Verravalli, and A. A. Kassim, “Design and implementation of parallel video encoding strategies using divisible load analysis,” IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 9, pp. 1098–1112, Sep. 2005.
[10] Y. Sambe, S. Watanabe, D. Yu, T. Nakamura, and N. Wakamiya, “High-speed distributed video transcoding for multiple rates and formats,” IEICE Trans. Inf. Syst., vols. E88-D, no. 8, pp. 1923–1931, Aug. 2005.
[11] Y. Wang, R. Yao, and X. Wang, “A round Robin with multiple feedback job scheduler in Hadoop,” in Proc. IEEE Int. Conf. Prog. Informat. Comput., May 2014, pp. 471–475.
[12] T. Balharith and F. Alhaidari, “Round Robin scheduling algorithm in CPU and cloud computing: A review,” in Proc. 2nd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS), May 2019, pp. 1–7.
[13] D. C. Devi and V. R. Utharariar, “Load balancing in cloud computing environment using improved weighted round Robin algorithm for nonpreemptive dependent tasks,” World J., vol. 2016, pp. 1–14, Feb. 2016.
[14] H. Kl api, E. Saritari, M. M. Tsangaris, and Y. Ioannidis, “Schedule optimization for data processing flows on the cloud,” in Proc. Int. Conf. Manage. Data (SIGMOD), 2011, pp. 289–300.
[15] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, “Prediction-based dynamic resource allocation for video transcoding in cloud computing,” in Proc. 21st Euromicro Int. Conf. Parallel, Distribut., Newt.-Based Process., Feb. 2013, pp. 254–261.
[16] S. Tang, B.-S. Lee, and B. He, “Dynamic slot allocation technique for MapReduce clusters,” in Proc. IEEE Int. Conf. Comput. Cluster. (CLUSTER), Sep. 2013, pp. 1–8.
[17] A. M. Elkholy and E. A. H. Sallam, “Self adaptive Hadoop scheduler for heterogeneous resources,” in Proc. 9th Int. Conf. Comput. Eng. Syst. (ICCES), Dec. 2014, pp. 427–432.
[18] C.-C. Huang, J.-J. Chen, and Y.-H. Tsai, “A dynamic and complexity aware cloud scheduling algorithm for parallel video transcoding,” in Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICM EW), Jul. 2016, pp. 1–6.
[19] Y. Yao, J. Wang, B. Sheng, C. C. Tan, and N. Mi, “Self-adjusting slot configurations for homogeneous and heterogeneous Hadoop clusters,” IEEE Trans. Cloud Comput., vol. 5, no. 2, pp. 344–357, Apr./Jun. 2017.
[20] E. Mohamed and Z. Hong, “Hadoop-MapReduce job scheduling algorithms survey,” in Proc. 7th Int. Conf. Cloud Comput. Big Data (CCBD), Nov. 2016, pp. 237–242.
[21] N. S. Naik, A. Negi, and V. N. Sastry, “A review of adaptive approaches to MapReduce scheduling in heterogeneous environments,” in Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI), Sep. 2014, pp. 677–683.
[22] A. Rasooli and D. G. Down, “A hybrid scheduling approach for scalable heterogeneous Hadoop systems,” in Proc. SC Companion, High Perform. Comput., Netw. Storage Anal., Nov. 2012, pp. 1284–1291.
[23] T. Matsumo, B. C. Chutter, G. K. N. Krishnan, E. Oppel, M. Veeraraghavan, S. Okamoto, and N. Yamanaka, “Designing a Hadoop system based on computational resources and network delay for wide area networks,” Telecommun. Syst., vol. 70, no. 1, pp. 13–25, Jan. 2019.
[24] D. Seo, J. Kim, and I. Jung, “Load distribution algorithm based on transcoding time estimation for distributed transcoding servers,” in Proc. Int. Conf. Inf. Sci. Appl., 2010, pp. 1–6.
[25] S. Tian, J. Xue, W. Liu, T. Xu, and N. Zheng, “High performance cluster-based transcoder,” in Proc. Int. Conf. Comput. Appl. Syst. Modeling (ICASSM), vol. 2, Oct. 2010, pp. V2–48–V2–52.
[26] C.-Y. Wu, H.-Y. Yu, J.-C. Huang, and J.-J. Chen, “A hierarchical reliability-driven scheduling for cloud video transcoding,” in Proc. Int. Conf. Mach. Learn. Cybern. (ICMLC), Jul. 2015, pp. 456–461.
[27] K. S. Chatrapati, J. U. Rekha, and A. V. Babu, “Competitive equilibrium approach for load balancing a computational grid with communication delays,” J. Theor. Appl. Inf. Technol., vol. 19, no. 2, pp. 126–133, 2010.
JIAN-JONE CHEN (Member, IEEE) received the B.S.E.E. and M.S.E.E. degrees from National Cheng Kung University, Tainan, in 1989 and 1991, respectively, and the Ph.D. degree in electronic engineering from National Chiao Tung University, Hsinchu, in 1997. He was a Researcher with the Information and Communications Research Laboratories, Advanced Technology Center, Industrial Technology Research Institute (ITRI), Hsinchu, from 1999 to 2002. He was a Visiting Scientist with the MIT AI-Laboratories, from September 2000 to November 2000. Since 2002, he has been with the Department of Electrical Engineering, National Taiwan University of Science and Technology (NTUST), where he is currently an Associate Professor. His research interests include image/video processing, deep image/video compression, image retrieval, and several topics in multimedia communications. His research team won the First Prize in the National College Microcomputer Design, in 2011 and 2012, with a live video streaming system and a real-time cloud video transcoding system. In 2021, he received the IEEE ICCE Excellent Paper Award. His research team won the First Prize in EE-Section in the 2021 National Technical College Thematic Competition. From 2018 to 2022, he conducted several industrial research projects on developing AI-assisted application systems, such as lightweight AI module design. He designed AI teaching plans for high-school students for the Taiwan Ministry of Education, in 2020 and 2022.

HAN-YEN YU received the B.S.E.E. degree from the National Yunlin University of Science and Technology, Yunlin, Taiwan, in 2006, and the Ph.D. degree in electrical engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2015. He worked on the NSC and ITRI research projects to design smart surveillance systems, face recognition systems, RFID location systems, and the Internet streaming systems. In 2015, he joined the Digital Education Institute, Institute for Information Industry (III), Taipei. He conducted a project about Taiwan Small-School Alliance interactive real-time teaching through E-learning to meet the needs of learning in the remote areas of different provinces for the shortage of teachers had been a frequently seen situation in those places. The plan successfully integrated the resources of the learning industry, charities, and enterprises of real-time broadcast systems, starting the cross-school services, and gathering more than ten schools in three provinces to form this alliance. He is currently a Senior Engineer with the Digital Education Institute, Institute for Information Industry (III). His research interests include digital image processing, education technology, and image/video encoding.

CHING-CHENG HWUANG received the B.S.E.E. degree in electrical engineering from Tatung University, Taipei, Taiwan, in 2011, and the M.S.E.E. degree in electrical engineering from the National Taiwan University of Science and Technology, Taipei, in 2013. He is currently a Software Engineer with ASUS Corporation. His research interests include cloud image/video processing and cloud computation.

CHENG-JEI SUNG received the B.S. degree from the Electrical Engineering and Computer Science Department, Chung Yuan University, Taoyuan, Taiwan, in 2014, and the M.S. degree from the Electrical Engineering Department, National Taiwan University of Science and Technology. He is currently a Field Application Engineer with Teradyne, Inc. His research interests include cloud image/video processing and cloud computation.

YOU-SHIN WU received the B.S. degree from the Electrical Engineering Department, Taipei University of Technology, Taipei, Taiwan, in 2021. He is currently pursuing the master’s degree with the National Taiwan University of Science and Technology. His research interests include cloud computing, deep image/video compression, and web real-time-communication.