Enabling Interactive Video Streaming for Public Safety Monitoring Through Batch Scheduling

Matin Hosseini
Center of Advanced Computer Studies,
School of Computing and Informatics
University of Louisiana at Lafayette
Email: mx0212@louisiana.edu

Mohsen Amini Salehi
HPCC Laboratory,
School of Computing and Informatics
University of Louisiana at Lafayette
Email: amini@louisiana.edu

Raju Gottumukkala
Informatics Research Institute,
College of Engineering
University of Louisiana at Lafayette
Email: raju@louisiana.edu

***Abstract*—Public safety officials want to have maximum situational-awareness though real-time information, such as video content, for natural disaster management. The video content can be generated by surveillance cameras or crowd-sourced (e.g., using smart-phones) and live-streamed to the Incident Commander. Such video contents need to be processed to adapt the characteristics of the specialized multi-view display devices. When a disaster occurs, there is a surge in the number of videos streamed to the Incident Commander that oversubscribes the processing servers and the network load. Incident Commanders, however, need a smooth and uninterrupted viewing experience specifically for the important events of interest that can change over time. The challenge is how to enable the Incident Commander to interactively prioritize important video streams to receive them uninterruptedly while the system is oversubscribed. In such a system, normal video streams (*i.e.,* non-prioritized ones) should not be interrupted at the expense of prioritization. To address this challenge, in this research, we propose a stream-priority aware resource allocation mechanism to enable interactive video prioritization without a major impact on the flow of non-prioritized video streams. The mechanism includes a method to select appropriate tasks from the arriving ones and a method to map the selected task to the appropriate video server. Our simulation results express that the percentage of normal and prioritized video streaming tasks that have completed on-time are improved, when compared with baseline scheduling methods.**

***Index Terms*—Priority-aware scheduling method, intractive video stream prioritization, frame-skipping.**

I. Introduction

At the time of a natural disaster, public safety officials and Incident Commanders should develop an efficient action plan to mitigate the consequences of the disaster. Reddy *et al.,* [15] identify inefficient information acquisition as one of the three major challenges in existing natural disaster management systems.

Live video streaming is extensively utilized by Incident Commanders to obtain maximum situational-awareness from disastrous areas. As shown in Figure 1, the videos are captured and streamed from different sources, such as planted surveillance cameras or crowd-sourced videos (*e.g.,* through smart-phones). The streamed videos are transmitted over a Software Defined Network (SDN) to a command center. The video streams are displayed on specific monitoring devices in the command center that have the ability to show multiple streams simultaneously. As such, incoming video streams have to be converted (also, termed *transcoded*) on dedicated video stream servers in a real-time manner to adapt to the characteristics of the monitoring devices.

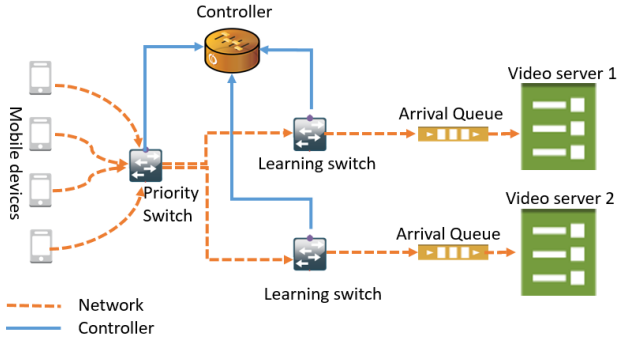


Fig. 1. An overview of video surveillance system for natural disaster management. The system includes network and video processing components.

A problem arises at the time of disaster when several users start to capture the disaster scene and stream the videos to the Incident Commander. This causes oversubscription on the video processing servers. The problem complicats furthur when some of the servers become unavailable, possibly due to the disaster. These problems negatively impact the quality of streaming by creating delay or even interruption in the stream. Public safety officials, however, need a smooth and uninterrupted viewing experience for different events of interest; in particular, those providing more important information from the disaster scene [13]. Therefore, they desire to *interactively* prioritize important video streams (known as *high-priority* streams) to secure a higher level of Quality of Service (QoS) for them . Also, the rest of video streams (known as *normal-priority* streams) should be able to continue streaming.

In the current systems, the same QoS is offered to all video streams [14]. That is, we cannot discriminate video streams and enforce a higher QoS to more important ones. Accordingly, our *goal* in this research is to enable Incident Commanders to interactively prioritize video streams of their interest and secure a higher QoS for them. However, normal-priority video streams should not be interrupted at the expense

of the prioritization.

We define an oversubscribed server as a machine that cannot complete its assigned tasks on time (*i.e.,* before tasks' individual deadlines). Such over-subscription causes missing the deadline of live video stream tasks and delay in viewing them at the command center. Therefore, the problem we investigate in this research is: *how to schedule tasks of high-priority streams along with tasks of normal-priority streams when the processing servers are oversubscribed so that the QoS for high-priority tasks is guaranteed and normal-priority tasks can continue streaming?* Efficient resource allocation methods are required to overcome the oversubscription on video processing servers through efficient mapping of live video streaming tasks to the servers. The mapping should be aware of the high-priority video streams and guarantee their QoS, without any major impact on the QoS (delay) of normal-priority streams.

The problem of resource allocation for live video streaming is different from other resource allocation and scheduling problems in the sense that it has to deal with high-priority versus normal-priority tasks. In addition, tasks of normal-priority video streams can be *approximately processed* to reduce their execution times. That is, normal-priority video streams have the flexibility to skip processing of some frames [6]. The approximate processing of normal-priority video streams (known as *frame-skipping*) impacts their viewing quality; however, it can be tolerated in favor of high-priority ones [6].

Live video streaming tasks have individual deadlines for processing that must be respected to keep up with the live streaming. As there is no value in processing live streaming tasks that miss their deadlines, such tasks are dropped (*i.e.,* discarded) from the system. Accordingly, we define QoS of high-priority and normal-priority video streams based on the number of tasks that meet their deadlines. As such the goal of the resource allocation mechanism is to minimize the percentage of live video streaming tasks that miss their deadlines.

Our proposed resource allocation mechanism includes a method to *select* appropriate tasks from the arriving ones and a method to *map* (*i.e.,* assign) the selected task to the appropriate video servers. The mapping method is aware of the flexibility in the execution time of normal-priority tasks and uses that to minimize the number of both high-priority and normal-priority tasks missing their deadlines.

We evaluate the efficacy of our resource allocation mechanism under various circumstances that can possibly occur during a time of disaster. In particular, we analyze the behavior of our mechanism when the system is under different oversubscription levels and when some of the processing servers are unavailable. In summary, the key contributions of this paper are as follows:

- Providing a resource allocation mechanism that is stream-priority aware and can guarantee QoS of the high priority video streams when the system is oversubscribed.
- Providing a mapping method that assigns high-priority video streaming tasks to servers with the minimum impact on the quality of normal-priority tasks.
- Analyzing the behavior of the proposed resource allocation mechanism under different circumstances that can happen at the time of disaster, namely when the oversubscription level increases, and when some of the video processing servers are unavailable due to failure.

The rest of paper is organized as follows: section II provides some related work on video stream transcoding. In section III we briefly talk about our system model. Our proposed algorithm consisting of task selection algorithm and provisional mapping will be discussed in section IV. We perform some experimental result evaluation in section V and finally discuss conclusion and future work in section VI.

II. Related Work

Several research works have been undertaken in the areas of video stream processing and natural disaster management. In this section, we review the recent research works in these areas and position this research work with respect to them.

A. Video Stream Transcoding

Video stream transcoding generally depends on the type of video streaming. As we can see in Figure 2, video streaming can be carried out either in video on-demand (VOD) or live streaming.

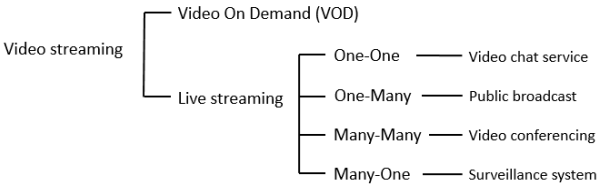


Fig. 2. Various types of video streaming.

Live video streaming can itself be performed in four different fashions, namely *one-to-one, one-to-many, many-to-many* and *many-to-one*. One-to-one streaming is mostly used in video chat services (*e.g.,* Skype and other Video-over-IP applications [2]). In one-to-many streaming, one server broadcasts the live content to many viewers (*e.g.,* LiveStreaming[1] and Periscope[2]). Many-to-many streaming is mostly used in video conferencing (*e.g.,* GoToMeeting[3] and e-learning systems). In these systems, each participant receives live video streams from all other participants. In many-to-one video streaming, several (many) participants stream videos to one centralized system. Video surveillance services, such as those used in traffic management [18], or crowd-sourced video streaming used in natural disaster management, fall in this category.

In all of the video-streaming fashions, video transcoding must be done to adapt the stream based on the characteristics of the viewers' devices. In VOD streaming (*e.g.,* Youtube[4],

[1] https://livestream.com
[2] https://www.periscope.tv/
[3] https://www.gotomeeting.com/
[4] https://www.youtube.com

Netflix[5]), generally, transcoding is achieved in an off-line fashion (known as pre-transcoding). Li *et al.,* [9], [11] provide an architecture for lazy (*i.e.,* on-the-fly) transcoding of video streams that are rarely requested by viewers. In all types of live video streaming, however, transcoding operation has to be carried out on-the-fly.

Video surveillance processing systems [7] consist of three parts. *First*, the live video stream is encoded in the source (by the capturing device). *Second*, the encoded video is sent through the network (*e.g.,* through SDN). *Third*, the live video stream is transcoded based on the characteristics of the monitoring device in the Incident Commander. Transcoding can include operations such as converting the spatial resolution, frame rate, and codec standard of the video streams [11]. In addition, it can include other video processing operations such as manipulating video contents [5] or applying frame-skipping to drop frames from a video stream and shorten its execution time [3].

B. QoS-Based Video Stream Processing

At the time of a natural disaster, the Incident Commander monitoring system is oversubscribed by many live videos streamed to it. Under these circumstances, meeting the QoS demands of the live video streams depends on efficient allocation of video streaming task to video processing servers.

Ma *et al.,* [12] propose a resource allocation method based on the intensity of the video streaming workload. They dedicate separate queues based on the priority of the video streams to schedule them on the servers. They allocate faster processing servers to video streams that are in the higher priority queue. In this system, the priority of video streams are determined based on the viewers' requests. In contrast, we present a system that supports dynamic prioritization of video streams based on the Incident Commander preference. In addition, our proposed system is designed to be robust against oversubscription of video streams at the time of natural disaster.

Efficient scheduling of video transcoding tasks is dependent on accurate execution time prediction of the tasks. Li *et al.,* [10] propose an architecture for one-to-many live video streaming using cloud services. Their goal is to minimize the incurred cost of using cloud services and meet the QoS demands of stream viewers. They define the QoS demands of viewers in terms of minimizing the streaming startup delay and drop rate in the live streams. In the scheduler, the execution time of each transcoding task is predicted based on the execution time of previous transcoding tasks of the same video stream. Our work is different from [10] in the sense that we consider a many-to-one live streaming and the viewer (*i.e.,* Incident Commander) can interactively change the priority of the video streams. In addition, we consider the ability of frame-skipping to shorten the execution time of low priority transcoding tasks. However, we utilize the method presented in [10] to predict the execution time of transcoding tasks.

[5]https://www.netflix.com

III. System Model

We consider a disastrous situation where different users live stream videos to the Incident Commander using different capturing devices and formats [1]. The video content should be transcoded based on the characteristics of the monitoring devices in the command center. The transcoding process can include changing bit-rate, codec standard, and spatial and temporal resolution [1]. We assume that, at each moment, there are N videos are being streamed and transcoded on the video processing servers.

As depicted in Figure 3, each video stream is composed of several segments which is further divided into *Group Of Pictures* (GOP) [11]. Each GOP can be transcoded independently, as such, we consider it as an independent task. To keep up with live video streaming, each GOP is assigned an individual hard deadline. That is, the GOP is dropped (*i.e.,* discarded), if it misses its deadline.

As we can see in Figure 3, each GoP contains multiple frames of different types. It starts with an Independent frame (called `I`-frame) and followed by Predictive frames (`P`-frame) and Bi-directional frames (`B`-frame) [1]. `I`-frame contains a complete image of a scene but `P`-frames and `B`-frames keep only the residues of the frames.

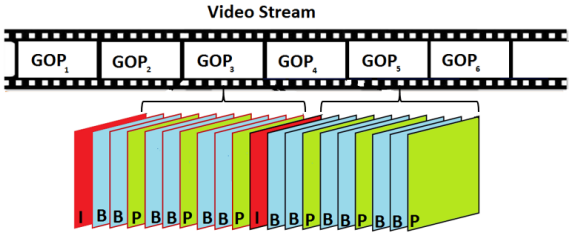


Fig. 3. A video stream is composed of several segments and each segment includes several GoPs. Within each GOP, there are `I`, `P`, and `B` frame types.

We assume that, at the time of a disaster, our video processing servers are oversubscribed due to a massive number of live video streams or due to network or server outage. We define an oversubscribed server as a congested server that, in general, cannot complete all the GOP tasks assigned to it on time [17]. In this situation, the Incident Commander desires to interactively prioritize some video streams over others. Accordingly, arriving GOP tasks are divided into two categories, namely *high-priority* and *normal-priority* GOP tasks.

The structure of resource allocation system is depicted in Figure 4. Arriving GOP tasks are queued before being mapped to a particular machine. To discriminate normal-priority and high-priority tasks, we dedicate separate queues for the arriving normal-priority and high-priority tasks. Our video processing servers are in form of a cluster with k dedicated and homogeneous machines. Each machine has a local queue with a limited size to fetch GOP tasks before their executions start. Once a GOP task is mapped to a local queue, it cannot be rescheduled on other machines because it

involves overhead of moving the GOP to another machine. As we consider a homogeneous cluster, all local queues have the same limit (*i.e.,* same local queue size).

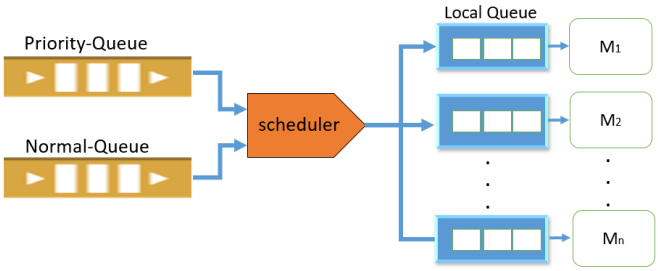


Fig. 4. Structure of the resource allocation system.

Our objective is to minimize the percentage of high-priority GOP tasks that are dropped because of missed deadlines without inducing major task dropping on normal GOPs. Normal-priority GOP tasks are not in the focus of Incident Commander. Therefore, although our goal is not to cause any task dropping on normal-priority GOP tasks, we have the flexibility to perform frame-skipping [3] on them to decrease their transcoding time in favor of meeting deadlines of other high-priority or normal-priority GOP tasks.

IV. Proposed Resource Allocation Mechanism

A. Overview

Because there are two types of tasks (*i.e.,* normal-priority and high-priority) in the system, as shown in Figure 4, we consider a dedicated *arrival queue* for each priority level. Upon arrival of a GOP task, it is assigned to one of these arrival queues based on its priority. In the event that a GOP task is completed, the scheduler method is executed to assign one or more tasks to the local queue of machines that have a free slot. The scheduler has two components that work together to achieve the goal of scheduling. The components of the scheduler are as follows:

- Task Selection: The job of this component is to determine which tasks needs to be selected for mapping. For that purpose, this component determines an appropriate way to order GOP tasks within each arrival queue.
- Provisional Mapping: For a selected task, this component determines the appropriate machine queue and provisionally maps the task to it. The component asks Task Selection for the next appropriate task and provisionally maps it, until either all the free slots are provisionally mapped or there is no task left in the arrival queues for mapping.

Once all the free slots are provisionally mapped, the mapping decisions can be finalized. In the following subsections, we elaborate on the details of these two components.

B. Task selection

Task Selection component, in each iteration, selects a task and feeds it to the Provisional Mapping component with the goal of maximizing the likelihood of meeting the tasks' deadline while considering their priorities. To achieve this goal, we define the concept of *Latest Start Time* (LST) as the latest time that a task can start its execution in a machine without missing its deadline.

We sort the arrival queues based on the LST of tasks in those queues. Then, the Task Selection component selects a task for mapping from the arrival queues based on the following criteria:

- Higher Priority
- Lower LST

Let t_{h1} be the first task from the high-priority queue and t_{n1} the first task in the normal-priority queue. Normally, t_{h1} is selected by the Task Selection component, unless, the LST of t_{n1} is lower than the LST of t_{h1} (*i.e.,* $LST(t_{n1}) < LST(t_{h1})$). In this case, t_{n1} can be selected, but only if it does not cause the tasks in the high-priority queue to miss their deadlines. In fact, if the deadline of t_{n1} is lower than or equal to the LST of t_{h1}, then we can assure that the execution of t_{n1} does not violate the deadline of t_{h1}. Thus, t_{n1} can be selected for mapping earlier than t_{h1}. It is worth noting that, if the LST of t_{h1} can be satisfied, then the LST of all the remaining tasks in the high-priority queue will be satisfied too. That is, selecting t_{n1} does not violate the deadline of any task in the high-priority queue.

Algorithm 1 Pseudo-code for Task Selection

Input:
normal-priority queue (N-Queue)
high-priority queue (P-Queue)

```
1:  Sort P-Queue and N-Queue based on LST
2:  P1=P-Queue[1]
3:  N1=N-Queue[1]
4:  Selected_Gop = Compare(P1,N1)
5:  if Selected_Gop = P1 then
6:      Provisional_Mapping(P1)
7:  else
8:
9:      if deadline(Selected_Gop) < LST(P1) then
10:         Provisional_Mapping(N1)
11:     else
12:         Provisional_Mapping(P1)
13:     end if
14:     end if
```

Algorithm 1 explains the method for the Task Selection component. As we can see in the algorithm, first, both queues are sorted based on their LST values. Ties are broken in favor of the task with an earlier deadline (Line 1 in Algorithm 1). Then, the LST of the first task in the two queues are compared and the one with the lowest LST is selected (Line 4). If the selected task is from the high-priority queue then we can proceed with the Provisional Mapping component. Otherwise (*i.e.,* if the selected task is from the normal-priority queue), the task is selected to be fed to the Provisional Mapping component only if the deadline of the selected task is lower

than or equal to the LST of the first task in the high-priority queue (see Line 9). It is noteworthy that if the deadline of first task in normal-priority queue is greater than the LST of the first task task in the high-priority queue, then the task from the high-priority queue is selected for provisional mapping.

C. Provisional Mapping

1) Estimating Execution Start Time of the Selected Task: Once a task is selected by the Task Selection component, we need to determine the appropriate mapping for it. The appropriate mapping for a task is the machine that provides the minimum execution start time for it. To obtain the minimum execution start time for the selected task in Provisional Mapping, a temporary virtual queue is formed to estimate the start time of the selected task on each machine. The virtual queue for machine j includes all tasks provisionally mapped to that machine.

The estimated execution start time of the selected task on M_j is the estimated completion time of the last task provisionally mapped to M_j. As shown in Equation 1, the estimated completion time of the last task i on machine M_j, denoted C_{ij}, is the sum of three parts. The first part is the remaining time for the currently executing task on machine j, denoted R_j.The second part is the sum of the estimated execution times of N tasks waiting in the local queue of machine j, denoted e_q. The third part is the sum of the estimated execution times of the i tasks provisionally mapped to machine j, denoted e_v (*i.e.,* tasks in the virtual queue of machine j).

$$C_{ij} = R_j + \sum_{q=1}^{N} e_q + \sum_{v=0}^{i} e_v \tag{1}$$

In live stream transcoding, because the GOP tasks are processed (*i.e.,* transcoded) upon generation, the execution time of the tasks are unknown beforehand. Besides, there is no historic execution information to use for predicting the tasks execution times[6]. However, it has been shown that generated GOPs in a video stream have approximately the same number of frames, thus, similar execution times [10]. In fact, execution time of GOP transcoding tasks in a live stream follow a Normal distribution $N(\mu, \sigma)$ where μ and σ are the average and standard deviation of execution time of the previous GOP tasks in the video stream. Accordingly, the execution time of GOP tasks waiting in the local queue (e_q) and the tasks in the virtual queue (e_v), in Equation 1, can be estimated by sampling from the Normal distribution. In addition, to calculate the remaining execution time of the current task on machine j (*i.e.,* R_j in Equation 1), the elapsed execution time of the task is deducted from the estimated execution time of the task.

2) Rescheduling Using Conservative Backfilling: For a selected task, if the provisional mapping method cannot allocate it before its deadline, then it has to be dropped. However, if we reschedule tasks within a virtual queue, then we may be able to allocate the selected task before its deadline. Therefore, we need a method that can reschedule tasks within a virtual queue to minimize the number of tasks that miss their deadlines.

Conservative backfilling [16] is a method that can schedule a task ahead of already scheduled tasks in a virtual queue, as long as it does not violate their deadlines. Accordingly, selected task s can backfill an already allocated task i in the virtual queue of machine j, if it does not violate the deadline of task i. Let D_i be the deadline of task i and C_{ij} the completion time of task i on machine j. Then, backfilling of task s before task i is feasible, if and only if $D_i - C_{ij} \geq e_s$.

To apply backfilling for selected task s in a virtual queue of machine j, we start performing the feasibility check from the last task in the virtual queue. The feasibility check stops as soon as a feasible slot is found for mapping task s so that it can meet its deadline.

3) Frame-Skipping in Video Streaming: Execution time of GOP transcoding tasks can be diminished by skipping processing a number of frames in the GOPs. Burza *et al.,* [3] propose a method to skip processing of `B` frames in a GOP. Because no other frame in a stream is dependent on the `B` frames, skipping them has no influence on the other frames. The side-effect of skipping `B` frames is that the video stream seems to be frozen for a short time. In this way, transcoding execution time of a GOP task can be reduced by up to 20% without having interruption in viewing the video stream [6]. Frame-skipping technique in a video stream can be helpful in meeting the deadlines of normal-priority and high-priority GOP tasks.

When we cannot allocate a normal-priority task before its deadline, the task has to be dropped. However, we can apply the frame-skipping technique to increase the likelihood of allocating the task before its deadline. Also, if we cannot find an allocation for a high-priority task, we can apply frame-skipping on the normal-priority tasks already allocated in the virtual queues. This will increase the chance of meeting deadlines for high-priority tasks.

4) Provisional Mapping Method: Based upon the methods and techniques discussed in Subsections IV-C1 to IV-C3, in this part, we explain how the Provisional Mapping maps a selected task to a video processing server. For the sake of clarity, the flowchart of the Provisional Mapping method is presented in Figure 5.

As we can see in the flowchart, the selected task is mapped to the machine with the minimum start time. For that purpose, we estimate the start time of the selected task i in each virtual queue based on the method presented in Subsection IV-C1.

If we cannot find any mapping to allocate task i prior to its deadline, then we try to allocate the task on the machine that provides the minimum start time using conservative backfilling, as explained in Subsection IV-C2. If the task cannot be allocated even by using conservative backfilling, the last step is to find a time-slot for the task using frame-skipping, as explained in Subsection IV-C3.

If selected task i is normal-priority , then we apply frame-skipping task i and then we try to backfill it on the machine with minimum start time again. If the selected normal-priority

[6]We should note that the historic execution time information is available in the case of VOD [11].

task i cannot be allocated before its deadline, the task is dropped. Alternatively, if the selected task i is high-priority and it cannot be allocated before its deadline, then our strategy is to create space for the task in the virtual queues using frame-skipping. For that purpose, we apply frame-skipping on normal-priority tasks in the virtual queues until an allocation is found. It is worth noting that, in this case, virtual queues are considered based on the start time they provide for task i.

Finally, if we cannot find any allocation for a high-priority task even after frame-skipping, then in the virtual queue that provides the shortest start time for the task, we drop the normal-priority tasks until the high-priority task can be allocated before its deadline. In the event that there is no normal-priority task to drop in favor of the high-priority task i, it is dropped.

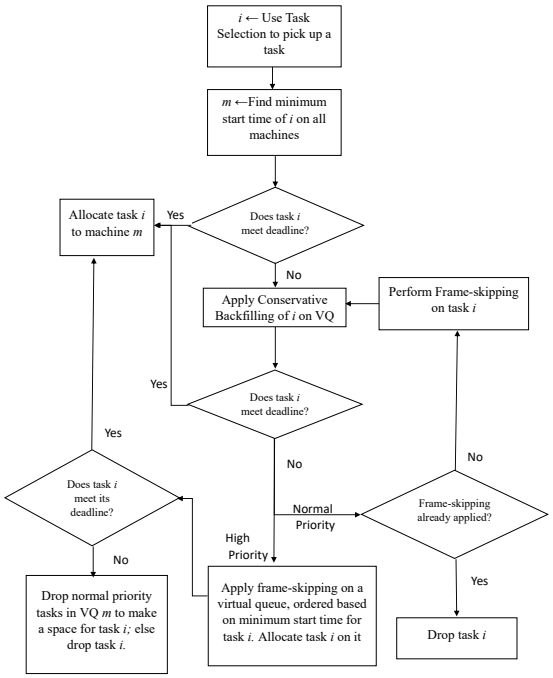


Fig. 5. Provisional mapping algorithm, allocate the selected task to the machine.

V. Performance Evaluation

A. Experimental setup

We used CloudSim [4], a discrete event simulator, to model and evaluate our proposed methods. To simulate live video streaming, we used video streams in a benchmark[7] used in previous research works [9], [11].

To simulate the live video streams arriving to the system, we consider Poisson distribution (with $\lambda = 0.01$) for start times of the video streams.We consider different number of GOP for simulating disastrous time. GOPs within a video stream are generated and sent on a periodic basis, denoted δ. However, each GOP (task) has a different network delay before arriving to the video processing servers. As such, arrival time of task i, denoted r_i, is calculated as: $r_i = r_{i-1} + \delta + \varepsilon_i$. We consider GOP tasks generated every 100 ms [10] and the network delay for task i (that is, ε_i) is generated by sampling from a Normal distribution $N(0, 10)$.

The performance metric in the experiments is minimizing the number of GOP tasks that missed their deadlines (*i.e.,* dropped). In particular, we evaluate how different scheduling methods perform in terms of number of high-priority and normal-priority tasks dropped.

For the sake of accuracy and to remove any randomness from the results, each experiment has been conducted 10 times and the mean and 95% of confidence interval of the results are reported.

B. The Impact of High Priority Tasks

The main goal of this research is to enable prioritizing some video streams interactively, particularly when the system is oversubscribed. Accordingly, we are interested in seeing how the system performs when different portion of the incoming streams are marked as high-priority .

For that purpose, in this experiment, we consider a scenario in which the Incident Commander requests different percentages of video streams as high-priority while the processing servers are oversubscribed. To simulate an oversubscribed system, we consider 400 GOP tasks from videos captured by different users arrive to the system with 5 processing servers. The portion of high-priority GOP tasks range from 10% to 50% of the whole workload. It is worth noting that, as we increase the percentage of high-priority tasks, the total number of tasks remain constant. That is, the percentage of normal-priority tasks reduces as we increase the percentage of high-priority tasks.

Figure 6 illustrates the percentage of GOP tasks dropped from high-priority and normal-priority categories when our proposed method is applied. As we can see in the figure, by increasing the percentage of high-priority tasks, the percentage of normal-priority tasks that missed their deadlines increases to keep the precedence of high-priority tasks. We can observe that the percentage of high-priority tasks missing their deadlines does not vary significantly. In particular, percentage of high-priority tasks missing their deadlines remain constant when they construct 10% to 20% or when between 30% to 50% of the tasks. The reason for increments in the percentage of high-priority tasks missing their deadlines is that by decreasing the number of normal-priority tasks, there are less opportunities for frame-skipping in favor of allocating high-priority tasks. Thus, the high-priority tasks cannot be allocated and miss their deadlines.

C. The Impact of Frame-Skipping

The goal of this experiment is to investigate the impact of frames-skipping method on the performance of the whole system in terms of number of tasks that can meet their deadlines. For that purpose, we study the behavior of the

[7]The benchmark is available publicly at https://goo.gl/TE5iJ5

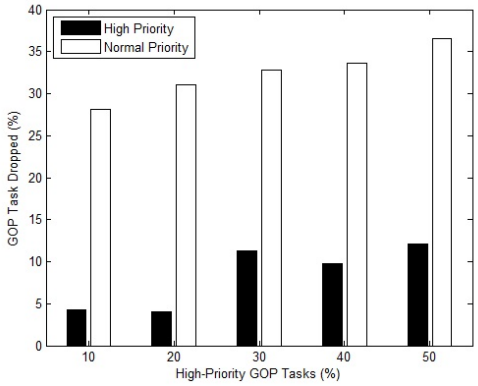


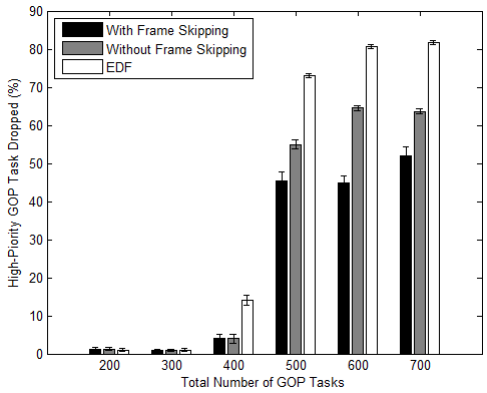
Fig. 6. The impact of percentage of high-priority tasks on the normal-priority and high-priority tasks missing their deadlines. Horizontal axis shows the percentage of high-priority GOP tasks and the vertical axis show the percentage of high-priority and normal-priority GOP tasks missing their deadlines.

system under different oversubscription levels. In particular, we considered 200 to 800 GOP tasks arriving to the system within the same time period. To evaluate the impact of frame-skipping method, we compared the percentage of tasks that miss their deadlines when our proposed scheduling is equipped with the frame-skipping method against the situation in which it does not use it. In addition, to study the performance of our proposed scheduling method, we compare its performance against Earliest Deadline First (EDF) scheduling method. EDF is a method widely used in scheduling of soft real-time and multimedia systems [8].In this experiment, we consider 20% of GOP tasks as high-priority .

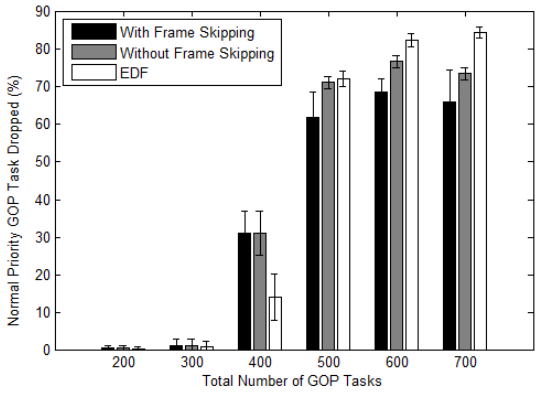
Figure 7 shows the percentage of high-priority tasks (in Sub-figure V-C) and normal-priority tasks (in Sub-figure V-C) that miss their deadlines when the degree of oversubscription changes. In both of the sub-figures, we can observe that when the system is oversubscribed, our scheduling method with frame-skipping significantly outperforms other methods both for normal-priority and high-priority tasks. However, when the system is not oversubscribed (*i.e.,* number of tasks less than or equal 300) there is not any significant difference between different methods. The reason is that when the system is oversubscribed the number of tasks that miss their deadlines increases. In this situation, applying the frame-skipping method can allocate more tasks before their deadlines. We can conclude that frame-skipping method should be applied only when the system is oversubscribed and when the system is not oversubscribed there is no point to incur its overhead.

D. The Impact of Failure Rate

Machine failure or unavailability commonly occurs during a natural disaster. As such, we are interested to see how our proposed scheduling method treats high-priority and normal-priority tasks in the presence of failure. For that purpose, we evaluate the performance of the system in terms of percentage of high-priority and normal-priority tasks that miss their deadlines when the number of available video processing servers varies for the same incoming workload.



(a) high-priority tasks



(b) normal-priority tasks

Fig. 7. The impact of frame-skipping on our proposed scheduling method. Horizontal axis shows different number of tasks and vertical axis shows the percentage of tasks missing their deadlines.

In this experiment, we vary the percentage of video processing servers failure from 0% to 50% and evaluate the percentage of high-priority and normal-priority tasks that miss their deadlines in each case. The number of arriving GOP tasks in this experiment is 600 and the 20% of tasks are high-priority . Also, we consider 10 video processing servers when there is no failure in the system.

Figure 8 shows the result of this experiment. In this figure, the horizontal axis shows the percentage of machine failure and the vertical axis shows the percentage of high-priority and normal-priority GOP tasks missed their deadlines (dropped) in each case using our proposed scheduling method.

As We can see in Figure 8, by increasing the percentage of machine failure, the percentage of GOP tasks that miss their deadlines (dropped) has increased. However, we observe that the percentage of high-priority and normal-priority GOP tasks that miss their deadlines enters into a steady state,

particularly, when the failure percentage is greater that 30%. The reason for the insignificant fluctuations is the increasing use of the frame-skipping method as the failure rate increases. Specifically, frame-skipping rate increases from approximately 0.9% to 15.5%, when the failure increases from 30% to 50%. Additionally, in Figure 8, we can observe that the scheduling method prioritizes high-priority tasks and keep the percentage of dropped tasks constantly below normal-priority tasks.

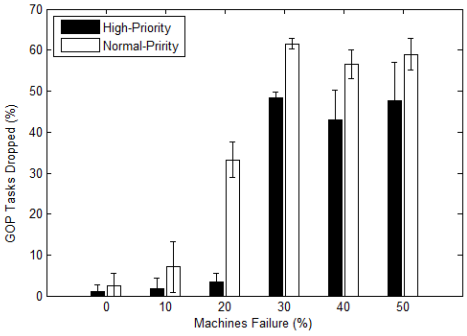


Fig. 8. The impact of video processing servers' failure on the percentage of high-priority and normal-priority tasks miss their deadlines. The horizontal axis shows the percentage of processing servers failed and the vertical axis shows the percentage of high-priority and normal-priority GOP tasks dropped.

VI. Conclusion and Future Work

The goal of this research was enabling public safety officials to interactively prioritize live video streams of their interest over other video streams in disastrous circumstances. We presented a resource allocation mechanism to schedule high-priority GOP tasks along with normal-priority GOP tasks in the presence of oversubscription and machine failure. The proposed method functions based on frame-skipping for normal-priority tasks to preserve the precedence of high-priority tasks. Experiment results under different workload and failure rates demonstrate that our system is able to prioritize high-priority tasks without interrupting other normal-priority video streams. In particular, our method can keep the percentage of high-priority tasks that missed their deadlines constantly (at least 12%) lower than normal-priority tasks. In the future, we plan to integrate our proposed method with SDN to build a robust public safety awareness system for natural disaster management. We also plan to study accurate methods for predicting execution time of GOPs in live video streams.

Acknowledgments

This research was supported by NSF under grant number 1451916 and the Louisiana Board of Regents under grant number LEQSF(2016-19)-RD-A-25.

References

[1] Ishfaq Ahmad, Xiaohui Wei, Yu Sun, and Ya-Qin Zhang. Video transcoding: an overview of various techniques and research issues. *IEEE Transactions on multimedia*, 7(5):793–804, 2005.

[2] Salman A Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *Proceedings of 25th IEEE International Conference on Computer Communications*, INFOCOM '06, pages 1–11, 2006.

[3] Marek Burza, Jeffrey Kang, and Peter Van Der Stok. Adaptive streaming of MPEG-based audio/video content over wireless networks. *Journal of Multimedia*, 2(2):17–27, 2007.

[4] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[5] Shih-Fu Chang and Anthony Vetro. Video adaptation: concepts, technologies, and open issues. *Proceedings of the IEEE*, 93(1):148–158, 2005.

[6] Kai-Tat Fung, Yui-Lam Chan, and Wan-Chi Siu. New architecture for dynamic frame-skipping transcoder. *IEEE transactions on Image Processing*, 11(8):886–900, 2002.

[7] Giovanni Gualdi, Andrea Prati, and Rita Cucchiara. Video streaming for mobile video surveillance. *IEEE Transactions on Multimedia*, 10(6):1142–1154, 2008.

[8] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K Sitaraman. Optimizing the video transcoding workflow in content delivery networks. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 37–48, 2015.

[9] Xiangbo Li, Mohsen Amini Salehi, and Magdy Bayoumi. High performance on-demand video transcoding using cloud services. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '16, pages 600–603, 2016.

[10] Xiangbo Li, Mohsen Amini Salehi, and Magdy Bayoumi. Vlsc: Video live streaming using cloud services. In *Proceedings of 5th IEEE International Conferences on Big Data and Cloud Computing*, BDCloud '16, pages 595–600, 2016.

[11] Xiangbo Li, Mohsen Amini Salehi, Magdy Bayoumi, and Rajkumar Buyya. Cvss: A cost-efficient and qos-aware video streaming using cloud services. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '16, pages 106–115, 2016.

[12] He Ma, Beomjoo Seo, and Roger Zimmermann. Dynamic scheduling on video transcoding for mpeg dash in the cloud environment. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 283–294, 2014.

[13] Neil J McCurdy, William G Griswold, and Leslie A Lenert. Realityflythrough: enhancing situational awareness for medical response to disasters using ubiquitous video. In *AMIA*, 2005.

[14] Kyoomars Alizadeh Noghani and M Oguz Sunay. Streaming multicast video over software-defined networks. In *Proceedings of 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 551–556, 2014.

[15] Madhu C Reddy, Sharoda A Paul, Joanna Abraham, Michael McNeese, Christopher DeFlitch, and John Yen. Challenges to effective crisis management: using information and communication technologies to coordinate emergency medical services and emergency department teams. *International journal of medical informatics*, 78(4):259–269, 2009.

[16] Mohsen Amini Salehi, Bahman Javadi, and Rajkumar Buyya. Resource provisioning based on preempting virtual machines in distributed systems. *Concurrency and Computation: Practice and Experience (CCPE)*, 26(2):412–433, 2014.

[17] Mohsen Amini Salehi, Jay Smith, Anthony A. Maciejewski, Howard Jay Siegel, Edwin K.P. Chong, Jonathan Apodaca, Luis D. Briceño, Timothy Renner, Vladimir Shestak, Joshua Ladd, Andrew Sutton, David Janovy, Sudha Govindasamy, Amin Alqudah, Rinku Dewri, and Puneet Prakash. Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system. *Journal of Parallel and Distributed Computing (JPDC)*, 97:96 – 111, 2016.

[18] Suman Srinivasan, Haniph Latchman, John Shea, Tan Wong, and Janice McNair. Airborne traffic surveillance systems: video surveillance of highway traffic. In *Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks*, pages 131–135, 2004.