Animation Rendering on Multimedia Fog Computing Platforms

Hua-Jun Hong, Jo-Chi Chuang, and Cheng-Hsin Hsu
Department of Computer Science, National Tsing Hua University, Hsin-Chu, Taiwan

***Abstract*—Modern distributed multimedia applications are resource-hungry, and they often leverage on-demand cloud services to reduce their expenses. Existing cloud services deploy many servers in a few data centers, which consume a lot of electricity to power up and cold down, and thus are expensive and environmentally unfriendly. In this paper, we present a multimedia fog computing platform that utilizes resources from public crowds, edge networks, and data centers to serve distributed multimedia applications at lower costs. We use animation rendering as a case study, and identify several challenges for optimizing it on our multimedia fog computing platform. Among these challenges, we focus on the problem of predicting the completion time of each rendering job. We propose an efficient algorithm based on state-of-the-art machine learning algorithms. We also fine-tune the algorithm using multi-fold cross-validation for higher prediction accuracy. With real datasets, we conduct trace-driven simulations to quantify the performance of our prediction algorithm and that of the whole platform. The simulation results show that our proposed algorithm outperforms a state-of-the-art statistical model in several aspects: completed job ratio by 20%, makespan by 2 times, and normalized deviation by 30 times, on average. Moreover, the overall performance of the platform with our proposed algorithm is fairly close to that with an Oracle of the actual job completion time: a small factor of 1.48 in terms of makespan is observed.**

***Index Terms*—Edge computing, volunteer computing, fog computing, prediction, animation rendering**

I. Introduction

We propose to adopt the concept of *fog* computing to build a platform for multimedia applications at reduced expenses compared to cloud computing. Fog computing, which extends cloud computing from data centers to edge networks, was initially proposed for Internet-of-Things applications [12]. Fog computing was then generalized [27] to also leverage resources of end devices owned by public crowds. That is, fog computing collectively utilizes resources from data centers, edge networks, and end devices, and thus may better support multimedia applications, which are delay-sensitive, resource-hungry, and heterogeneous.

As illustrated in Fig. 1, we envision a multimedia fog computing platform, or *fog provider*, that offers computation, communication, storage, and sensing resources to *fog users*, and harvests these resources from various *fog devices*. Other than the servers and networking equipments in data centers and edge networks, fog devices also include desktops, laptops, and smartphones. The fog devices have dynamic workload, and are not always fully loaded, e.g., when a student is taking notes in lectures using his/her laptop, or when a smartphone is being charged in the evenings. Therefore, fog providers may purchase the otherwise wasted resources from fog device owners, called *fog workers*, and resell these resources to fog users. Doing so provides a *cost-effective* fog computing platform: compared to cloud platforms, fog platforms utilize the wasted resources without (i) buying additional computers and spaces to build data centers, (ii) building expensive cooling systems, (iii) renting high-speed, reliable, and costly networks, and (iv) hiring engineers to manage data centers. These unique advantages not only enable fog providers to potentially offer the resources at lower costs, but also reduce the amount of electricity to power up and cool down data centers, leading to smaller carbon footprints.

We consider animation rendering as a sample multimedia fog computing application, which dictates increasingly more resources in the past few years. In 2011, rendering Cars 2 required a 12500-core rendering farm [8], while 4.4 times of resources were required in 2014 to render Big Hero 6 [2]. It is not affordable for animation studios and startups to build and maintain such computationally powerful rendering farms. Therefore, cloud rendering services, such as Shaderlight [5], RenderStorm [3], and ZYNC [7] get popular among these studios and startups. The cloud rendering services: (i) offer on-demand resources, (ii) relieve animation studios from managing large rendering farms, and (iii) provide up-to-date rendering software. However, even using cloud rendering services, animation rendering is still expensive, e.g., using Shaderlight [5] to render 300 frames (about 10 seconds) in 1080p costs 700 dollars. Considering multiple iterations of refinements, rendering even a short animation clip in the cloud would be too costly. Therefore, animation rendering is a good candidate application for multimedia fog computing platforms.

Supporting animation rendering in multimedia fog platforms is no easy task. Animation rendering requires: (i) high computational power for rendering, (ii) large storage space for rendering inputs and results, (iii) well-connected communication channels for distributing jobs and collecting results, and (iv) on-device sensors to monitor resource utilization. While multimedia fog computing platforms offer all these resources, fog devices are not fully controlled/owned by fog

This study is conducted under the "4G+ experimental network buildup and emerging mobile communication system technology development project" of the Institute for Information Industry which is subsidized by the Ministry of Economic Affairs of the Republic of China. This work is also partially supported by Ministry of Science and Technology (MOST) of Taiwan under grant number MOST 105-2221-E-007-088.

978-1-5090-1445-3/16$31.00

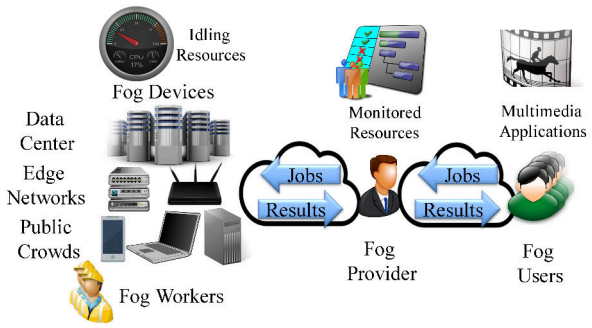

Fig. 1: Overview of a multimedia fog computing platform.

providers, which leads to high *uncertainty* and *heterogeneity*. For example, a laptop may be turned off, run out of batteries, or get into wireless dead-zone at any time, which leads to uncertainty. Moreover, there are many kinds of fog devices, such as desktops, laptops, smartphones owned by different fog workers, resulting in high heterogeneity. The uncertainty and heterogeneity may turn the decisions made by our multimedia fog computing platform *suboptimal* and/or *outdated*.

To cope with the uncertainty and heterogeneity, our multimedia fog platform works as follows. Fog providers keep track of the resource utilization of fog devices, such as CPU, GPU, and RAM, to quantify the amounts of idling resources. Fog providers then estimate the available resources of each fog device in future time periods. Moreover, fog providers measure the capabilities of fog devices, such as GFLOPS (Giga Floating Point Operations Per Second) and clock rate. The capabilities and characteristics of rendering jobs, such as the number of frames are used to predict the completion time. The predicted completion time and available resources are then used for scheduling jobs from fog users to fog devices/workers. Upon the jobs are completed, the results are sent back to fog users. To optimize the overall performance of our multimedia fog platform, there are many research problems that need to be solved. In this paper, we focus on *predicting the completion time of each rendering job*, due to the space limitations.

TABLE I: Diverse Completion Time of 5 Jobs on Different Devices

Device	Specifications			Job Rendering Time (sec)				
	GFLOPS	Clock	Cores	1	2	3	4	5
Server	133	2.2 GHz	16	19	18	42	16	19
Desktop	72	2.5 GHz	8	24	15	31	14	15
Laptop	50	2.0 GHz	8	27	19	50	18	21

A. Motivative Examples

Predicting rendering jobs' completion times is very challenging, because rendering jobs have different characteristics and fog devices have diverse specifications. We conduct a set of experiments that render five jobs using a real fog rendering service[1] on three devices ranging from a laptop to server. The device specifications and job completion times are given in Table I. We make two observations from the table. First,

[1]The service/company name is withheld due to a non-disclosure agreement.

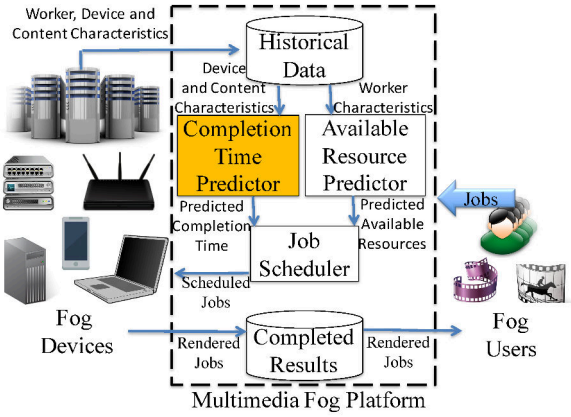

Fig. 2: The architecture of our multimedia fog platform.

while GFLOPS seems to be the most critical factor affecting the completion time at first glance, the completion time can *not* be predicted using GFLOPS alone. Particularly, although the desktop has a lower GFLOPS value than the server, the majority of jobs have shorter completion times on a desktop than on a server. This observation reveals that other device specifications also affect the completion time. Second, we find that the server has a shortest completion time when rendering job 1. A closer look indicates that job 1 consists of scenes that can better be parallelized; hence, job 1 can better utilize the 16 cores of the server. In contrast, jobs 2–4 are more sequential and dictate higher clocks (rather than more cores). These two observations on real data demonstrate the necessity of an intelligent prediction algorithm for the job completion time, based on the job characteristics and device specifications.

B. Paper Organization and Main Results

We present an overview on the three main challenges in Sec. II. We focus on the problem of predicting job completion time in Sec. III, where we design and fine-tune a prediction algorithm based on machine learning approaches. Significant improvements on completion time prediction accuracy are observed from the numerical analysis based on real-life datasets. We next evaluate our proposed algorithm using trace-driven simulations in Sec. IV. The trace-driven simulations reveal several advantages of our proposed algorithm: (i) the completed job ratio is improved by 20%, which enables the fog provider to earn 1.3 million dollars [5] more every week, (ii) the makespan is significantly reduced by 2 times, and (iii) the normalized deviation of the completion time prediction is improved by 30 times. We believe that the proposed completion time prediction algorithm serves as a good starting point of a more general fog computing platform for animation rendering and other distributed multimedia applications.

II. Overview

Our multimedia fog platform consists of three main components: (i) available resource predictor, (ii) completion time predictor, and (iii) job scheduler. Fig. 2 shows the architecture.

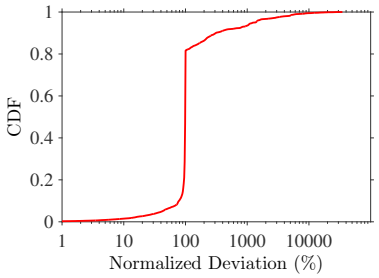

Fig. 3: Linear regression is insufficient for completion time prediction.

Moreover, there are two databases storing historical data and completed results, respectively. The first two predictors utilize historical data and state-of-the-art machine learning approaches to estimate the available resources and completion time. The job scheduler assigns rendering jobs to individual fog devices.

We classify the historical data into three categories: (i) device, (ii) worker, and (iii) content characteristics. Device characteristics are the hardware specifications of fog devices, e.g., GFLOPS and clock. Worker characteristics are the behaviors of workers, e.g., a graduate student may have 80% idling CPU cycles in the morning when writing papers and only have 10% in the evening when playing computer games. Content characteristics are the features of the rendering jobs, e.g., the number of frames, the number of polygons, and the size of rendered images.

The available resource predictor leverages the worker characteristics to estimate the amount of idling resources of fog workers in a future time period. The completion time predictor estimates the completion time of rendering jobs using device and content characteristics. The job scheduler then takes the estimated available resources and completion time as inputs to efficiently decide which rendering jobs should be rendered by which fog worker. The fog workers render the jobs and send the completed results back to the multimedia fog platform. Fog providers store these jobs in the completed results database and validate the integrity/correctness of the results. Last, fog providers send the completed results to the animation studios.

In this paper, we focus on predicting job completion time, which is challenging because both device and content characteristics must be considered, as we demonstrate in Sec. I-A. Indeed, a linear-regression based solution adopted by the fog rendering company produces inaccurate completion time estimations. In particular, we analyze 14,198 real rendering jobs, and compute the *normalized deviation* as the prediction error normalized to the actual job completion time. We give the Cumulative Distribution Function (CDF) of the normalized deviation in Fig. 3. This figure reveals that more than 80% of the predictions with linear regression deviate from the actual job completion time by more than 100%, while some extreme predictions deviate by 100 times. To address the limitation of inaccurate completion time estimations, we rigorously develop the completion time predictor in Sec. III.

III. Completion Time Predictor

A. Algorithm Overview

The completion time prediction problem is a *regression* problem with historical data on rendered jobs. The core idea is to adopt and fine-tune multiple algorithms based on machine learning approaches, such as Random Forest (RF), Gradient Boosting Tree (GBT), Support Vector Machine (SVM), Deep Learning [15, 22, 23], and even the new approaches developed in the future. More specifically, we *train* and *use* several machine learning approaches to predict the job completion time. Next, based on the recent R-square scores, we choose the best performing machine learning approach to predict the job completion time.

Algorithm 1 Completion Time Prediction Algorithm

```
1:  function HYPER_PARAM_SELECTION
2:      for i ∈ I do
3:          Use cross-validation to find the optimal values of H_i for the
    highest R-square score
4:      return H_i  ∀i ∈ I
5:  function MODEL_PARAM_SELECTION
6:      for i ∈ I do
7:          Train the model parameters P_i with historical data for the highest
    R-square score
8:      return P_i  ∀i ∈ I
9:  function VALIDATE_ACCURACY
10:     for i ∈ I do
11:         Calculate the R-square score r_i of the last W historical records
12:     return i*, where r_i* ≥ r_i  ∀i ∈ I
13: function TIME_EVAL
14:     return the completion time predicted by approach i*
```

Let $\mathbf{I}$ be the set of all machine learning approaches chosen by fog providers. For each approach $i \in \mathbf{I}$, we have to determine its *hyperparameters* $\mathbf{H}_i$ and train its *parameters* $\mathbf{P}_i$. Hyperparameters refer to the parameters that are chosen by human being offline using large historical data; while parameters are trained and updated more often. For example, fog providers may run extensive cross-validation to choose optimal hyperparameters ($\mathbf{H}_i$). However, the model parameters ($\mathbf{P}_i$) can be updated daily (say, at midnights) to cope with dynamics, e.g., new versions of fog applications (rendering engines) or the different distribution of fog devices; online updates of model parameters are also possible. Upon selecting $\mathbf{H}_i$ and $\mathbf{P}_i$, we can predict the job completion time using machine learning approach i ($i \in \mathbf{I}$). To pick the best prediction from all approaches, we compare the last W predicted values of approach i against the actual job completion time to calculate R-square score r_i. W is a sliding window, chosen by fog providers. We call the highest R-square score r_{i^*}, and when receiving the incoming job, we use machine learning approach i^* to predict the job completion time.

Algorithm 1 summarizes our Completion Time Prediction (CTP) algorithm, with four functions:

- *HYPER_PARAM_SELECTION*: to determine $\mathbf{H}_i$
- *MODEL_PARAM_SELECTION*: to determine $\mathbf{P}_i$
- *VALIDATE_ACCURACY*: to select r^*
- *TIME_EVAL*: to predict the completion time

We emphasize that the proposed CTP algorithm is general, and fog providers may choose machine learning approaches among $\mathbf{I}$ to meet their needs. In the rest of this section, we use animation rendering as the fog application, and demonstrate the dataset, decisions on $\mathbf{I}$ and optimal $\mathbf{H}_i$.

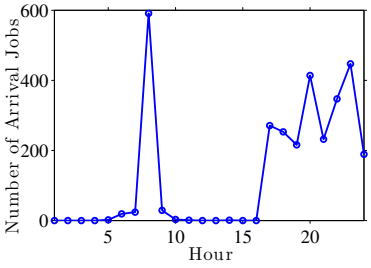

Fig. 4: The number of arrival jobs on a typical weekday.

TABLE II: Statistics of the Animation Rendering Dataset

Feature	Mean	Std.
CPU Usage (%)	21	12
RAM Usage (KB)	348	162
No. Frames	394	416
No. Polygons	45148	230607
Image Size (Pixels)	129307	19390
Completion Time (s)	101	9063

B. Datasets

We employ an *animation rendering dataset* from the collaborated fog rendering company for evaluations. There are 33,078 records collected between December 2015 and March 2016. Fig. 4 shows the number of arrival jobs every hour from a sample day, which contains 4072 jobs and almost all the jobs are submitted between 7–8 (in the morning) and 16–24 (in the evening). Every record is a rendering job from an animation studio, which uses one of the eight rendering engines supported by the fog rendering company. Some records are filtered out as they are corrupted during the data collection process, resulting in 32,704 records left. For each record, the resource usage, such as the CPU usage, the characteristics of rendering jobs, such as the number of frames, the network conditions, such as the time of sending a job, and the completion time are recorded. We adopt number of cameras, frames, lights, materials, polygons, textures, vertices, height, and width as the features to drive $\mathbf{M}$ prediction algorithms. In Table II, we show some sample statistics of the animation rendering dataset.

Furthermore, we employ an *available resource dataset* to drive the simulator in Sec. IV. The available resource dataset [24] is collected between August and November 2013. It contains real CPU, RAM, disk, and network availability of 1,750 virtual machines on Bitbrains [1]. On average, the recorded duration of each VM is 43 minutes and the resource usages are 0.6 GB, 1.4 GHz, 0.1 MB/s, and 0.1 MB/s of RAM, CPU, disk, and network, respectively.

C. Selection of Machine Learning Approaches

We consider three state-of-the-art machine learning approaches, SVM, RF, and GBT, which have been used to solve regression problems, such as Web search ranking and human pose estimation. Based on these three approaches, we have developed three algorithms: M_{SVM}, M_{RF}, and M_{GBT} using open source libraries [4, 6]. Some preliminary tests using the real-life dataset indicate that M_{SVM} has prohibitively long training time, as long as hours. Hence, we no longer discuss M_{SVM} in the rest of this paper. That is, we let $\mathbf{I} = \{\text{RF}, \text{GBT}\}$.

Both M_{RF} and M_{GBT} are tree-based algorithms, but there are major differences between them. M_{RF} constructs a multitude of independent decision trees, where each tree is trained with random samples of data. That is, when M_{RF} adds a new decision tree, it randomly selects several features and then randomly specifies a threshold value for these features. It splits the dataset using previously selected features and thresholds, and selects the new feature and the threshold that reduce the highest data entropy. M_{RF} then makes predictions by averaging all trees' prediction results. In contrast, M_{GBT} consists of a sequence of trees, where each successive tree is built to predict the residuals of the preceding one. M_{GBT}'s trees are trained and combined using a more sophisticated weighting scheme. Readers are referred to Friedman et al. [15] for more details on RF and GBT.

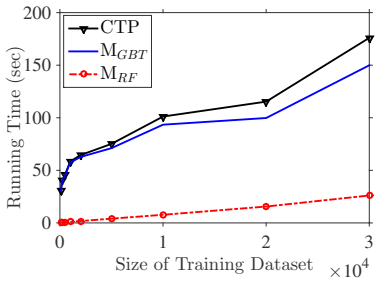

Fig. 5: The average training time of different machine learning approaches under different dataset sizes.

Although larger $\mathbf{I}$ leads to no-worse prediction accuracy, larger $\mathbf{I}$ also results in longer training time. To check the training time of $\mathbf{I} = \{\text{RF}, \text{GBT}\}$, we measure the average training time with training dataset sizes between 100 to 30,000 on an AMD 2.6 GHz workstation. We report the average training time of 10 runs in Fig. 5, which reveals that the training time with dataset of 30,000 is merely 176 seconds. This is negligible to typical fog providers, who retrain the model parameters daily.

D. Optimal Hyperparameters

We conduct 10-fold cross-validation to choose the optimal hyperparameters $\mathbf{H}_{\text{RF}}$ and $\mathbf{H}_{\text{GBT}}$. k-fold cross-validations first split our dataset into k equal-sized folds, and train a model using $k-1$ folds as training data. After that, we validate our model on the remaining fold to evaluate the performance of the chosen hyperparameters. This procedure repeats k times, each with different training folds. Last, the performance is the average value of k validations.

For M_{RF} has two hyperparameters $\mathbf{H}_{\text{RF}} = \{t, f\}$: (i) the number of trees t and (ii) the number of considered features

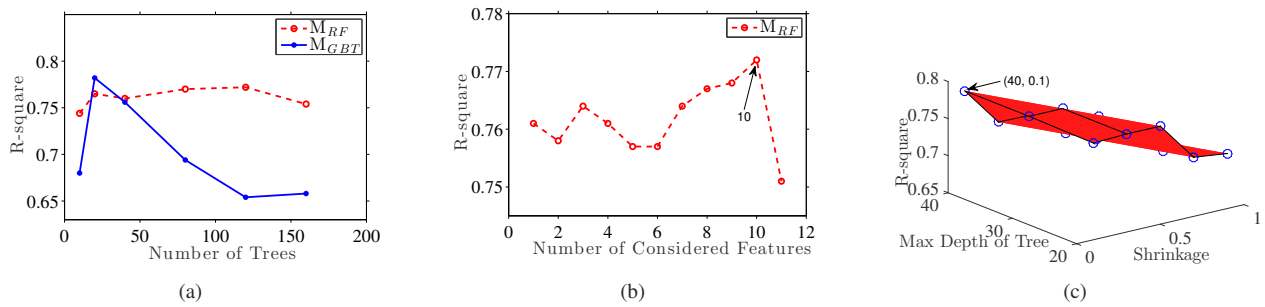

Fig. 6: Hyperparameters selections: (a) the number of trees (both M_{RF} and M_{GBT}), (b) the number of considered features (M_{RF}), and (c) maximal depth of trees and shrinkages (M_{GBT}).

f. We plot the R-square scores with different numbers of trees when $f = 10$ in Fig. 6(a). This figure shows that the number of trees $t = 120$ results in the best R-square score. The same trend remains with different numbers of considered features f. Therefore, we stick with 120 trees and vary the numbers of considered features f on each split. We plot the R-square scores of different number of considered features f in Fig. 6(b). This figure shows that 10 considered features lead to the best prediction performance.

M_{GBT} has three hyperparameters $\mathbf{H}_{\text{GBT}} = \{n, d, s\}$: (i) the number of trees n, (ii) the maximal depth of each tree d, and (iii) the shrinkage s (i.e., the learning rate). The number of trees $n = 20$ results in the best R-square score (see Fig. 6(a)) with $d = 40$ and $s = 0.1$. Such trend remains under other values of d and s. We then fix the number of trees n as 20, and plot the R-square scores of different maximal tree depth d and shrinkage s in Fig. 6(c). This figure reveals that maximal tree depth $d = 40$ and shrinkage $s = 0.1$ lead to the optimal R-square scores.

In the rest of the paper, we adopt the derived hyperparameters if not otherwise specified. We acknowledge that these values are *data-driven*, and thus may need to be re-derived when the characteristics of the historical data dramatically change. Last, if other machine learning approaches are employed, their hyperparameters also need to be derived using 10-fold cross validation.

IV. Trace-Driven Simulations

A. Setup

We implement a detailed simulator of the multimedia fog computing platform using Java. Since the focus of this paper is on the completion time predictor, it is thoroughly evaluated in our trace-driven simulations. We implement our CTP and the state-of-the-art GMM (Gaussian Mixture Model) [14] algorithms in the simulator[2]. Moreover, we realize a perfect, but unrealistic, Oracle algorithm that looks up the animation rendering dataset for actual completion time. Oracle gives a performance upper bound of the prediction algorithms. We assume that the fog workers provide us with their daily schedules, which are essentially the available resource dataset. Last, we realize an efficient *Earliest Start Schedule* (ESS) [13, Chapter 3.2] algorithm to minimize the makespan. The job scheduler batches all the received rendering jobs every day and executes daily at 23:59 at the midnight.

We consider several performance metrics:

- **Completed jobs ratio**: The ratio between the number of completed rendering jobs achieved by a completion time prediction algorithm and the total number of jobs.
- **Normalized deviation**: The deviation between the predicted completion time and the actual completion time, normalized to the actual completion time.
- **Makespan**: The total time to complete a set of rendering jobs received from the animation rendering studios.
- **Rescheduled jobs**: The number of rendering jobs that are not completed before the assigned fog devices leave. These rendering jobs will be rescheduled by the job scheduler.
- **Normalized CPU consumption**: The average CPU consumption normalized to that of the Oracle.
- **Normalized RAM consumption**: The average RAM consumption normalized to that of the Oracle.

We use real datasets to drive the simulator. In particular, the following inputs are from our datasets (see Sec. III-B): (i) the time of submitting each rendering job, (ii) the consumed resources of each rendering job, and (iii) the completion time of each rendered job. We use Poisson process with a mean arrival rate $\lambda = 30$ minutes for fog devices. The capabilities of each fog device are randomly chosen from the available resource dataset. The CTP and GMM algorithms use the rendering jobs completed on day D to train their models and predict the completion time of the rendering jobs received on day D+1. We run the evaluations 10 times using the CTP, GMM, and Oracle algorithms. We calculate the average performance results every day, and also report the 95% confidence intervals whenever applicable. To stress our platform, we report sample results from the 7 most active days[3] of the dataset in the rest of this section.

[2]The details of the literature about completion time prediction, including GMM are given in Sec. V.

[3]17,416 rendering jobs, about 53% of all jobs, are received during these 7 days.

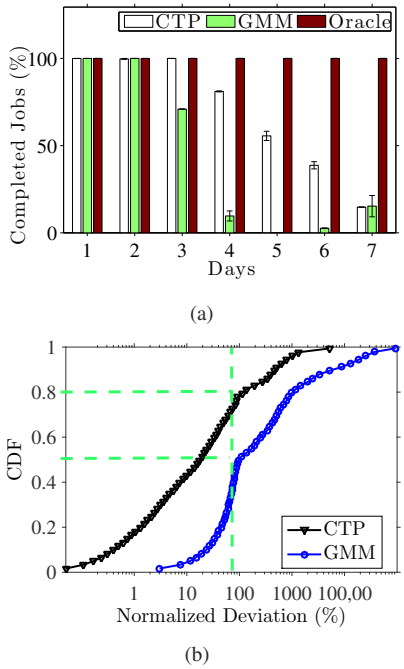

Fig. 7: The CTP algorithm completes more jobs with higher prediction accuracy: (a) completed job ratio and (b) normalized deviation.

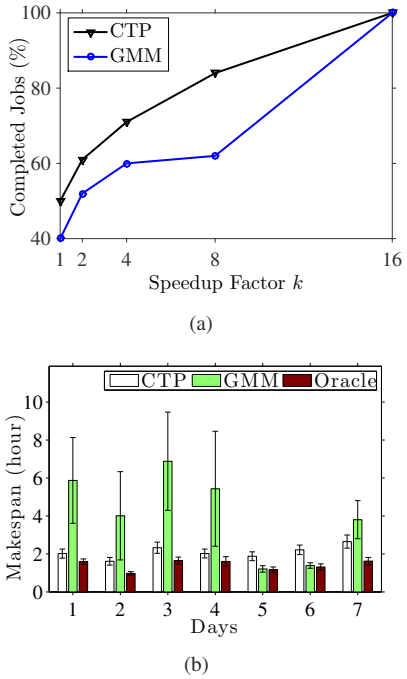

Fig. 8: The performance of our CTP algorithm: (a) implications of speedup factor on completed job ratio and (b) makespan.

B. Results

Our CTP algorithm leads to higher completed job ratio and smaller normalized deviation. Fig. 7(a) reports the completed jobs ratio. Our CTP algorithm achieves 86% of completed jobs, while the GMM algorithm only achieves 66% on average. More specifically, with the GMM algorithm, there are 2,455 more uncompleted jobs, which contain 579,573 frames, compared to our CTP algorithm. Based on some back-of-envelope calculations using the price of cloud rendering service [5], this is equivalent to 1,352,337 dollars. That is, *a fog provider may increase its weekly profit by* 1.3 *million by using our CTP algorithm*. A closer look into Fig. 7(a) reveals that the CTP algorithm completes 1.42, 8, and 14 times more rendering jobs, compared to the GMM algorithm on days 3, 4, and 6. Moreover, on day 5, the GMM algorithm completes almost no rendering job, while the CTP algorithm completes half of the rendering jobs. This can be explained by Fig. 7(b), which shows that the GMM algorithm results in much higher normalized deviation than the CTP algorithm. More specifically, 80% of the predictions with the CTP algorithm deviate from the actual job completion time by less than 100%, while only 50% of the predictions with the GMM algorithm do the same. On average, the CTP algorithm outperforms the GMM algorithm by 30 times in terms of normalized deviation.

TABLE III: Numbers of Over-/Under-Estimated Completion Times in a Sample Run

Algorithm	GMM	CTP	Oracle
Over Estimated	13442	6165	0
Under Estimated	219	7019	0
Perfectly Estimated	0	477	13661

Our CTP algorithm results in shorter makespan. Although the CTP and GMM algorithms cannot complete all the rendering jobs on the busiest days, the capability of the fog devices will increase over time following the Moore's Law. We let k be a speedup factor, and vary it from 1 to 16. We then plot the average completed job ratio under different k values in Fig. 8(a). We observe that our CTP algorithm will continue outperforming the GMM algorithm in terms of completed job ratio in the future. With $k = 16$, the rendering jobs are completed by all algorithms, and we plot the makespan with $k = 16$ in Fig. 8(b). On days 1, 2, 3, 4, and 7, the GMM algorithm results in 2.9, 2.5, 2.9, 2.7, and 1.4 times longer makespan compared to the CTP algorithm. This can be explained by Fig. 7(b), which shows that the CTP algorithm is more accurate than the GMM algorithm. On other two days, the CTP algorithm only spends 36% more time compared to the GMM algorithm to complete the rendering jobs. Across the 7 days, the CTP algorithm outperforms the GMM algorithm by 2 times in terms of average makespan. Furthermore, we observe that the CTP algorithm results in a small makespan factor of 1.48 compared to the Oracle. We also notice that over-estimated completion time may incur negative impacts on the makespan. This is because over-estimated job completion time prevents the job scheduler to assign the job to some fog devices that *in fact* have enough resources to render it. The GMM algorithm indeed results in 50 times of normalized deviation on average of the over-estimated completion time, while CTP only leads to 52% normalized deviation. This partially explains why the resulting makespan of the GMM algorithm is much higher than that of the CTP algorithm.

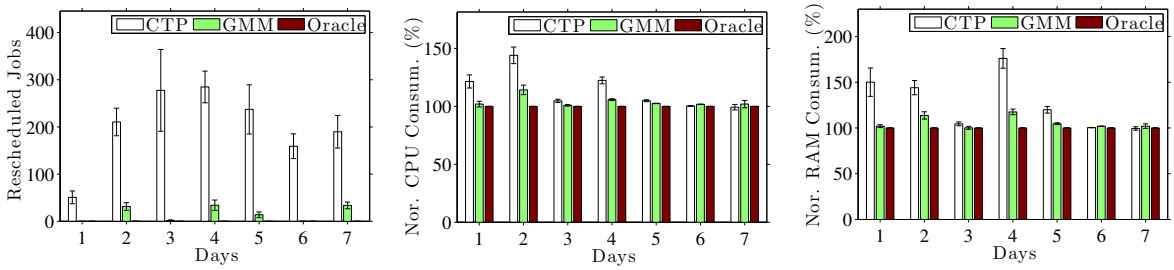

Fig. 9: Our CTP algorithm adapts to departed fog devices: (a) rescheduled jobs, (b) normalized CPU consumption, and (c) normalized RAM consumption.

Our CTP algorithm adapts to departed fog devices at low overhead. Table III shows that 51.4% of the job completion times are under-estimated using our CTP algorithm. Under-estimated completion time leads to uncompleted rendering jobs due to premature departures of fog devices. Fortunately, our CTP algorithm adapts to uncompleted rendering jobs by rescheduling them to other fog devices. Fig. 9(a) shows that the CTP algorithm reschedules 1431 more jobs compared to the GMM algorithm. More rescheduled jobs lead to higher resource consumptions, as illustrated in Figs. 9(b) and 9(c). Compared to the GMM algorithm, our CTP algorithm only consumes 8.5% and 17% more CPU and RAM, respectively. This shows that the additional overhead caused by the under-estimated completion times of the CTP algorithm is manageable.

V. Related Work

Several studies in the literature can be seen as special cases of our multimedia fog computing platform. They can be classified into the following three groups.

P2P Cloud. Traditional Peer-to-Peer (P2P) systems are designed for video streaming and file sharing. For example, Liu et al. [18] survey the P2P video streaming systems and Pouwelse et al. [19] conduct a measurement study on the BitTorrent P2P file sharing system. Researchers also propose fully-distributed P2P cloud in the literature. For example, Xu et al. [30] propose P2P cloud storage. Babaoglu et al. [11] design and implement a P2P cloud system, and propose an algorithm to partition the resources for better performance. Graffi et al. [16] propose a protocol for resource reservations, such as storage space and network bandwidth. Different from our multimedia fog platform, P2P clouds are fully-distributed systems without any central entity and thus have a difficult time: (i) managing end devices owned by crowds and (ii) allocating the resources to provide Quality of Service (QoS) guarantees.

Mobile Offloading. The mobile offloading concept is proposed to solve the problem of running complex jobs on resource-limited mobile devices. Researchers propose to offload these jobs to powerful cloud servers. Lin et al. [17] propose a context-aware mobile cloud offloading decision engine, which considers several contexts, such as signal strength and GPS locations to make the offloading decisions. The decisions may reduce energy consumption or response time. Researchers also offload the resource-hungry jobs to edge networks. Satyanarayanan et al. [21] propose Cloudlet, which is a server placed next to a WiFi access point. Mobile devices connect to Cloudlet servers and offload their jobs through one-hop networks. Willis et al. [29] propose ParaDrop, which allows mobile users to run their jobs on WiFi access points. Researchers also try to offload the jobs among multiple mobile devices. For example, Verbelen et al. [28] extend the Cloudlet by aggregating nearby mobile devices for computationally demanding jobs. In contrast to mobile offloading, our multimedia fog platform uses heterogeneous devices and supports diverse applications, including computing-, communication-, storage-, and sensing-intensive applications.

Volunteer Computing. The volunteer computing concept is proposed by Sarmenta [20], which leverages the resources from public crowds, such as desktops for computationally-demanding jobs. SETI@Home [10] is a volunteer computing application analyzing radio signals from the space for extraterrestrial intelligence. BOINC [9] is a generalized platform of volunteer computing, which utilizes computers connected to the Internet for computational jobs. Different from our multimedia fog platform, the volunteer computing workers are attracted by some high profile projects, such as finding aliens, analyzing protein structures, and discovering antimalarial drugs for free. Hence, volunteer computing does not suffer from some challenges of the multimedia fog platforms. For example, fog providers need to carefully manage resources of each job to remain profitable. In addition, fog providers must guarantee that a job can be completed on time in order to retain fog users.

Estimation Algorithms of Job Completion Time. The job completion time prediction has also been studied [14, 25, 26]. Tao et al. [26] and Sonmez et al. [25] estimate the completion time using Sliding Median (SM) and Exponential Smoothing (ES) algorithms on grid computing platforms. They are both sliding-window-based algorithms. Estrada et al. [14] model the job completion time on volunteer computing platforms. They adopt GMM model on networking, job processing, and validation delays, which is used as the baseline algorithm

in Sec. IV. These algorithms [14, 25, 26] are all statistical models without taking the characteristics of rendering jobs into considerations.

VI. Conclusion

In this paper, we studied the problem of utilizing idling resources, e.g., from public crowds, to serve distributed multimedia applications for reducing the cost and carbon footprints. We present a multimedia fog computing platform, which carefully manages the idling resources, predicts the completion time of jobs, and schedules the jobs to suitable fog devices. In this paper, we focused on the crucial completion time predictor and proposed the CTP algorithm for it. We conducted trace-driven simulations using real datasets to evaluate the CTP algorithm and the whole platform. The simulation results indicate that our CTP algorithm: (i) completes 86% of the jobs (outperforms GMM by 20% on average), (ii) spends less time to complete rendering jobs (outperforms GMM by 2 times), and (iii) accurately estimates the completion time (outperforms GMM by 30 times).

This paper is merely a starting point of a whole new research direction on *vertically* aggregating the resources from end devices, edge networks, and data centers. Even if we limit our scope to the sample animation rendering application, there are a wide spectrum of open challenges, such as predicting the available resources of fog devices, design a better job scheduling algorithm, verifying the correctness of rendering results, and optimizing the degree of duplicated job assignments. There are even more challenges when considering a unified multimedia fog computing platform. We believe the lessons learned in this paper will stimulate future research in this emerging research direction.

References

[1] Bitbrains. http://www.bitbrains.nl/solvinity-en.

[2] Disney rendered its new animated film on a 55,000-core supercomputer. http://www.engadget.com/2014/10/18/disney-big-hero-6/.

[3] Renderstorm. http://www.render-storm.com/index.html.

[4] Scikit-learn. http://scikit-learn.org.

[5] Shaderlight. http://limitlesscomputing.com/Shaderlight.

[6] XGBoost. https://github.com/dmlc/xgboost.

[7] Zync. https://www.zyncrender.com/.

[8] New technology revs up Pixar's Cars 2. http://www.cnet.com/news/new-technology-revs-up-pixars-cars-2/, June 2011.

[9] D. Anderson. Boinc: A system for public-resource computing and storage. In *Proc. of IEEE/ACM Grid Computing (GC)*, pages 4 – 10, Pittsburgh, PA, November 2004.

[10] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *ACM Transactions on Communications*, 45(11):56–61, November 2002.

[11] O. Babaoglu, M. Marzolla, and M. Tamburini. Design and implementation of a P2P cloud system. In *Proc. of ACM Symposium on Applied Computing (SAC)*, pages 412–417, Trento, Italy, March 2012.

[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proc. of ACM Workshop on Mobile Cloud Computing (MCC)*, pages 13–16, Helsinki, Finland, August 2012.

[13] P. Brucker and S. Knust. *Complex Scheduling*. Springer, 2012.

[14] T. Estrada, M. Taufer, and K. Reed. Modeling job lifespan delays in volunteer computing projects. In *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 31–338, Shanghai, China, May 2009.

[15] J. Friedman, R. Tibshirani, and T. Hastie. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[16] K. Graffi, D. Stingl, C. Gross, H. Nguyen, A. Kovacevic, and R. Steinmetz. Towards a P2P cloud: Reliable resource reservations in unreliable P2P systems. In *Proc. of IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 27 – 34, Shanghai, China, April 2010.

[17] T. Lin, T. Lin, C. Hsu, and C. King. Context-aware decision engine for mobile cloud offloading. In *Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 111 – 116, Shanghai, China, April 2013.

[18] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-peer Networking and Applications*, 1(1):18–29, March 2008.

[19] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent P2P file-sharing system: Measurement and analysis. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 205–216, Ithaca, NY, February 2005.

[20] F. Sarmenta. Volunteer computing. Technical report, Massachusetts Institute of Technology, 2001.

[21] M. Satyanarayanan, P. Bahl, R. Cceres, and N. Davies. The case for VM-based Cloudlets in mobile computing. *IEEE Transactions on Pervasive Computing*, 8(4):14–24, December 2009.

[22] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[23] B. Scholkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[24] S. Shen, V. Beek, and A. Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 465 – 474, Shenzhen, China, May 2015.

[25] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proc. of ACM International Symposium on High Performance Distributed Computing (HPDC)*, pages 111–120, Munich, Germany, June 2009.

[26] M. Tao, S. Dong, and L. Zhang. A multi-strategy collaborative prediction model for the runtime of online tasks in computing Cluster/Grid. *Cluster Computing*, 14(2):199–210, June 2011.

[27] L. Vaquero and L. Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, October 2014.

[28] T. Verbelen, S. Pieter, T. Filip, and D. Bart. Cloudlets: Bringing the cloud to the mobile user. In *Proc. of ACM Workshop on Mobile Cloud Computing and Services (MCS)*, pages 29–36, Low Wood Bay, UK, June 2012.

[29] D. Willis, A. Dasgupta, and S. Banerjee. Paradrop: a multi-tenant platform for dynamically installed third party services on home gateways. In *Proc. of ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC)*, pages 43–44, Chicago, IL, Augest 2014.

[30] K. Xu, M. Song, X. Zhang, and J. Song. A cloud computing platform based on P2P. In *Proc. of IEEE International Symposium on IT in Medicine and Education (ITIME)*, pages 427 – 432, Jinan, China, Augest 2009.