Highlight-aware Content Placement in Crowdsourced Livecast Services

Cong Zhang*, Jiangchuan Liu♯, Haitian Pang†, Fangxin Wang*
*School of Computing Science, Simon Fraser University, Canada
♯College of Natural Resources and Environment, South China Agricultural University, China
†Department of Computer Science and Technology, Tsinghua University, China
Email: congz@cs.sfu.ca, csljc@ieee.org, pht14@mails.tsinghua.edu.cn, fangxinw@cs.sfu.ca

***Abstract*—Recent years have witnessed an explosion of crowdsourced livecast (i.e., live broadcast) services, in which any Internet users can act as broadcasters to publish livecasts to fellow viewers. To help grow broadcasters' channels, crowdsourced livecast services provide a past-broadcast saving service, allowing viewers to watch the replays they may have missed. Our real-trace measurement and questionnaire survey show that (1) the duration of most of livecasts is extremely long; (2) a much longer duration largely affects the viewers' Quality-of-Experiences (QoE) when watching the replays. To address this issue and improve viewers' QoE, we propose a crowdsourced framework HighCast based on the interactive messages contributed by the viewers in crowdsourced livecast services. According to a highlight-aware detection module, HighCast can exploit the detection results to schedule the content placement by considering the importance of the predicted streaming highlights. The trace-based evaluations illustrate that the proposed framework improves the prediction accuracy and reduces the viewing latency.**

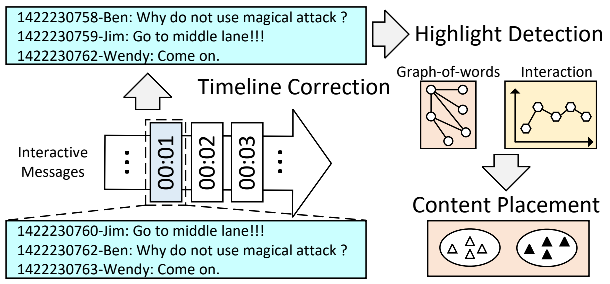
I. Introduction

Recently, crowdsourced livecast, such as Twitch.tv (or Twitch for short) and YouTube Gaming has become one of the most fashionable Internet applications. Different from professional content providers, e.g., American Broadcasting Company, these broadcasting "freshmen" can enable their personal devices (e.g., laptop and mobile phone) to perform various shows, such as game playthrough, costume design, and music-making, to fellow viewers conveniently. Taking Twitch as an example, more than 2 million unique monthly broadcasters perform shows and attract more than 15 million unique daily viewers in 2017. Besides, over 1 million concurrent viewers watched a game event in last January.

To continually promote channel's growth and attraction, crowdsourced livecast services provide *past-broadcasts saving service* allowing the viewers to watch the replays they may have missed[1]. To better investigate the challenges and opportunities therein, we have conducted a viewing questionnaire survey on viewers' personal preference and watching experience. Our data analysis reveals that (1) 50% of viewers tend to browser the video *highlights*[2] rather than watching a whole video; (2) most of the viewers (about 67%) prefer watching the replays within 60 minutes. Our Twitch-based measurement, however, shows that the duration of about 60% of replays is more than 100 minutes, which is longer than the VoD contents (10 minutes) in traditional streaming systems [1]. Therefore, how to improve the viewers' QoE and optimize the content placement of the replays becomes a challenging problem.

Thanks to the live chatting service in crowdsourced livecast services [2], the users' interactive messages create a unique opportunity to detect the highlights in the past-broadcasts. In this paper, we present a crowdsourced framework **HighCast** to capture the highlights during the replays and optimize the corresponding content placement. The trace-driven evaluation shows that HighCast achieves a higher detection accuracy and optimization performance than other approaches. To the best of our knowledge, this is the first work to optimize the replay content placement by exploiting the interactive messages to detect livecast highlights in crowdsourced livecast services.

II. Highlight-aware Content Placement



1422230758-Ben: Why do not use magical attack ?
1422230759-Jim: Go to middle lane!!!
1422230762-Wendy: Come on.

1422230760-Jim: Go to middle lane!!!
1422230762-Ben: Why do not use magical attack ?
1422230763-Wendy: Come on.

Fig. 1: Illustration of the framework HighCast

In this section, we introduce the HighCast framework, as shown in Figure 1. It contains three components: (1) Timeline correction: The input of our design is interactive message dataset, which also includes the sender ID and the timestamp of each message, but the timestamp only reflects the sending time; therefore, we design this component to adjust each message to the correct time of live broadcasting according to the length of a message. (2) Highlight detection: Based on the adjusted input messages, we employ the graph-of-words model to detect the highlights. (3) Content placement: Using the results of highlight detection, we optimize the content

[1]https://help.twitch.tv/customer/portal/articles/1575302-videos-on-demand

[2]In this paper, we define a video segment recording a key event, such as teamfight, as a video highlight.

978-1-5386-2542-2/18/$31.00 © 2018 IEEE

placement. Due to the space limitation, we only introduce the details of the highlight detection and the content placement.

A. Highlight Detection

To detect the highlights, we first parse the viewers' messages based on the co-occurrence text graph, where vertices correspond to terms, and edges correspond to co-occurrence between the terms. Specifically, edges are drawn between vertices if the vertices co-occur within a "window" of maximum N terms. In this paper, the size of "window" depends on the length of a message, that is, N is equal to the number of terms in this message. To examine all of the terms in a short time, we also collect all of the messages in a certain time slot as a document (the default duration of a time slot is set to one minute in this paper). To reflect the impact from the multiple messages sent by a viewer in a duration, we consider the multiple messages as a new message. For each message in a document, we consider each unique term in this message as a vertex and create an edge for any two vertices, which in turn builds a complete sub-graph in the text graph of the current document. After constructing the text graph, we extract highlights based on the viewers' average message number and k-core graph decomposition for the text graph [3] [4].

B. Content Placement

We assume that the replay delivery and storage follow the standard of Dynamic Adaptive Streaming over HTTP, where each replay is transcoded to multiple versions, and each version is divided to multiple segments with the same duration, e.g., 2 seconds. According to the highlight prediction results, we prioritize the segment placement in Content Distribution Network (CDN) such that important segments (i.e., the contents that are in highlights) are distributed more urgently.

We consider a general CDN scenario, where the geo-distributed servers store and deliver replay segments. We use E to denote the set of servers in CDNs, and $D^{(T)}(e)$ denotes the available storage size of server e in time slot T, $e \in E$.We determine the priorities of the segments according to the result of the highlight detection. Let $R_{(e)}^{(T)}$ denote the set of replays in time slot T on server e, $r \in R_{(e)}^{(T)}$, and $S^{(r)}$ denote the set of segments in the replay r. We also denote $H^{(r)}$ as the highlight set of replay r, and $h_i^{(r)}$ as the segment set of highlight i, $h_i^{(r)} \in H^{(r)}$, and $h_i^{(r)} = \{s^{(r)} | s^{(r)} \in S^{(r)}\}$. In this step, we consider all versions with the same segment index as a whole segment. Let $L_{(r,s)}^{(T)}$ be the importance level of segment s in replay r. $L_{(r,s)}^{(T)}$ depends on the following factors: (1) the weight of highlight i, if the segment s belongs to $h_i^{(r)}$; (2) if not, the viewing weight of segment s, which depends on the distance from the nearest highlight; (3) the number of concurrent viewers in the replay r.

Based on the definition of the importance level of segments, we determine which segments to be placed and stored in CDNs by formulating it as a 0-1 knapsack problem and design an algorithm to heuristically solve it in a centralized manner: (1) We collect the information from the highlight detection, the replay servers in the CLAs, and the servers in CDNs, including the set of highlights, the size of segments, and the available storage size of servers in CDNs; (2) Based on this information, we rank the segments in descending order of $L_{(r,s)} / \sum_{v \in V} d(r,s,v)$, where $d(r,s,v)$ is the total size of segment s in multiple versions; (3) According to the ranked list, we iteratively select segments and update the storage size of servers until the idle storage resource is depleted.

III. Preliminary Evaluation

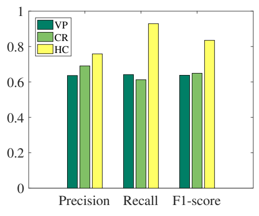


Fig. 2: Prediction result

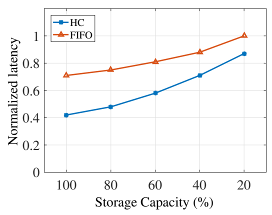


Fig. 3: Viewing latency

To evaluate the performance of the HighCast framework, we have conducted the experiments and simulations based on the stream and message datasets. The experiments focus on the efficiency of the highlight detection. The data-driven simulations evaluate the performance of content placement. We first illustrate the results of the experiments.

We select several hot channels to evaluate the performance of our highlight detection. Based on the stream dataset, we choose five replays from five channels. We implemented the highlight detection using a Python-based program. We first compare the performance of highlight detection with the views-based prediction (VP) and the chat-rate approach (CR). The former depends on the views to detect the highlights, and the latter depends on the chat rate per minute to determine whether a time slot has a highlight. Our method achieves higher average recall, precision, and F1-scores than another two approaches.

We also conducted an extensive simulation to evaluate the performance of content placement in the framework HighCast. According to the viewers' survey and the distribution of viewers of Twitch, we simulate the watching requests from 100 viewers. We assign the viewing latency based on the distance from a viewer to the nearest server. Figure 3 illustrates the efficiency of highlight-aware content placement in the framework HighCast. We normalize the viewing latency according to the maximum latency. Our optimization reduce viewing latency under the different storage capacity.

References

[1] X. Che, B. Ip, and L. Lin, "A survey of current youtube video characteristics," *IEEE MultiMedia*, vol. 22, no. 2, pp. 56–63, Apr 2015.

[2] C. Zhang, J. Liu, M. Ma, L. Sun, and B. Li, "Seeker: Topic-aware viewing pattern prediction in crowdsourced interactive live streaming," in *ACM NOSSDAV, 2017.*

[3] R. Blanco and C. Lioma, "Graph-based term weighting for information retrieval," *Information Retrieval*, vol. 15, no. 1, pp. 54–92, 2012.

[4] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis, "Degeneracy-based real-time sub-event detection in twitter stream," in *AAAI, 2015.*