Collaborative Delay-Aware Scheduling in Peer-to-Peer UGC Video Sharing[*]

Xu Cheng
School of Computing Science
Simon Fraser University, BC, Canada
xuc@cs.sfu.ca

Feng Wang
School of Computing Science
Simon Fraser University, BC, Canada
fwa1@cs.sfu.ca

Jiangchuan Liu
School of Computing Science
Simon Fraser University, BC, Canada
jcliu@cs.sfu.ca

Ke Xu
Department of Compter Science and Technology
Tsinghua University, Beijing, China
xuke@csnet1.cs.tsinghua.edu.cn

ABSTRACT

We have recently witnessed an explosion of user-generated content (UGC) sharing, particularly video clips, as the new killer Internet application. Given the sheer amount of resource demands, the peer-to-peer (or peer-assisted) model has been suggested for this new service scenario. There are however a series of unique challenges from the UGC videos to be addressed, in particular, their significantly shorter lengths. As such, any delay, even being minor as compared to those for conventional movie-like videos, will be perceptually amplified.

Given the much more stringent delay requirement, the UGC video sharing thus calls for sophisticated scheduling to provide quality playback. In this paper, we propose a novel collaborative delay-aware scheduling (CODAS) that is customized for the short UGC videos. CODAS improves playback quality and reduces server workload, through adaptive prioritization of data requests and tighter collaboration between peer suppliers and the server.

We present detailed design and optimization of CODAS, particularly the synergy policies in different zones of a shrinking window. We evaluate it through extensive trace-driven simulations and PlanetLab prototype experiments, and the results show the great improvement over the state-of-the-art solutions.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Algorithms, Design, Performance

[*]This research is supported by a Canada NSERC Strategic Project Grant, an NSERC Discovery Grant, an NSERC DAS Grant, and an MITACS Project Grant.

Keywords

User-Generated Content, Video on Demand, Peer-to-Peer, Scheduling

1. INTRODUCTION

In the past five years, we have witnessed an explosion of user-generated content (UGC) sharing, particularly video clips, as a new generation of killer Internet application. The most successful site, YouTube, enjoys more than one billion views every day [10]. The success of similar sites such as Youku (the most popular video sharing site in China) further confirms the mass market interest. However, given the sheer amount of resource demands, the traditional client/server architecture has severely hindered their further expansion (e.g., YouTube has extremely high expense on server bandwidth) [3]. To alleviate the bottleneck, the popular peer-to-peer (or peer-assisted [2, 5]) model has been suggested for this new service scenario, with some commercial prototypes being deployed recently. A typical example is the Peer-to-Peer Video Accelerator (PPVA)[1], which extends the widely-used PPLive by an on-demand streaming engine, providing a universal platform that transparently accelerates a collection of popular UGC video sharing sites. As of December 2008, the PPVA client software has been installed by over 150 million users with 40 million daily accesses.

Despite preliminary successes, PPVA and similar peer-to-peer engines are still facing a series of unique challenges from UGC videos; in particular, most of the UGC videos are relatively short clips [3]. Figure 1 shows the distribution of video lengths we collected from the latest PPVA traces[2]), where over 90% of the videos have length being shorter than 10 minutes, and 25% even shorter than 3 minutes. This is significantly different from the traditional movie-like videos, not to mention live TV channels of unlimited length. As such, any delay, even relatively minor as compared to traditional video services, will be perceptually amplified by the short video length, thus challenging the patience of users. In fact, as Figure 2 shows, many current PPVA peers' con-

[1]http://ppva.pp.tv
[2]The detailed methodology of measurement can be found in [13].

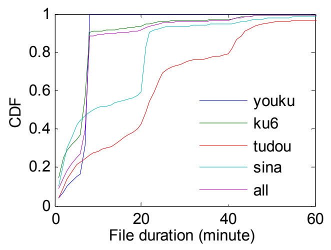

Figure 1: PPVA trace: CDF of video file duration

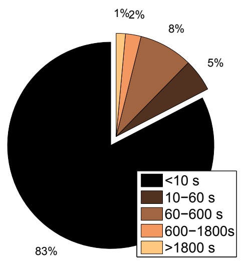

Figure 2: PPVA trace: breakdown of continuous downloads

tinuous downloads last for less than 10 seconds, and a great number of users do not finish downloading.

It is known that the peer-to-peer streaming performance is highly sensitive to the scheduling policy for data exchanges [7]. Given the much more stringent delay requirement, the UGC video sharing thus calls for sophisticated scheduling to provide quality playback with minimized cost. Moreover, such an optimization is desirable for both subscription and free video sharing services.

In this paper, we propose a novel *collaborative delay-aware scheduling* (CODAS) that is customized for the short UGC videos. CODAS minimizes startup delay and playback discontinuity, as well as server load, through adaptive prioritization of data requests, and tighter collaboration between peer suppliers and server. We present detailed design and optimization of CODAS, in particular, the synergy policies in different zones of a shrinking window. We have also performed extensive trace-driven simulations and PlanetLab prototype experiments, and the results show great improvement over the state-of-the-art solutions.

2. RELATED WORK

It is known that the scheduling is important to peer-to-peer streaming performance [7], and there have been numerous studies and implementations of peer-to-peer live video streaming [8]. Massoulie *et al.* designed a decentralized algorithm, random useful packet forwarding (RUPF) [9], in which peers identify the neighboring peer that needs data the most and serve the data to them. Guo *et al.* proposed an adaptive queue-based chunk scheduling (AQCS) method [4], in which the amount of workload assigned to a peer is proportional to its available upload capacity. Liang *et al.* investigated the scheduling sensitivity of representative designs [7]. Aggarwal *et al.* investigate intelligent scheduling in multicast VoD, showing Earlist Deadline First (EDF) is optimal approach [1].

Motivated by the successes of live streaming, on-demand streaming through peer-to-peer overlays has also attracted significant attentions. The protocols can be broadly classified into two categories according to their overlay structures, namely, tree-based (e.g., ChunkySpread [11]) and mesh-based (e.g., CoolStreaming [14] and PPLive); there have also been several practical applications, such as PPLive P2P-VoD [6], etc. The asynchronous issue in VoD is solved by cache-and-relay (e.g., peer-assisted VoD [5]), in which a peer buffers the data by a sliding window starting from the playback position, and serves others whose playback position is within the window. BitTorrent Streaming (BiToS), is aware of the sequence of the data piece [12], as it defines a high priority set containing the missing data pieces that are close to the playback position, and downloads them with higher probability and others with lower probability.

UGC VoD is a new generation of video service, and different from traditional VoD systems, its distinct characteristic of short video length brings more stringent delay requirements. Yet in most of the existing works, the peers do not differentiate the priority of the requests, and there lacks a collaboration between the receivers and the senders. Our approach takes the buffer level into account and explores the collaboration in the system.

3. SYSTEM OVERVIEW

3.1 System Model

We consider a peer-assisted on-demand streaming system for UGC videos: the system consists of a video repository server and its web portal, and the set of clients interested in the videos. The videos are downloaded by the clients in a complementary manner from both the server and other clients through a peer-to-peer overlay.

The current PPVA adopts a mesh-based overlay organization, where the peers pull expected data pieces with a set of partners (other peers or the server) through a sliding-window-based scheduling algorithm inherited from PPLive. Similar algorithms have been widely used in other peer-assisted live or on-demand streaming systems [14]. Yet, as mentioned earlier, the UGC videos have a series of unique features, most notably, the much shorter length, which brings new challenges to the scheduler design:

- *More stringent delay requirements*: Given the shorter length of the UGC videos, the startup and playback delay would be effectively amplified from the perception of clients;
- *Higher peer dynamics* (*churns*): Given the clients are less patient in waiting for short videos, more trials of joining/leaving would occur, leading to high churn rates, evidently from the PPVA trace that many do not finish.

To address these challenges, we propose a novel collaborative delay-aware scheduling (CODAS) that is customized for the short UGC videos. CODAS minimizes startup delay and playback discontinuity, as well as server load, through two salient features: adaptive prioritization of data requests, and tighter collaboration between the server and peer suppliers.

It is worth noting that currently we assume all clients play by the rules, that is, following our mechanism. Tamper resistance may be investigated as part of the future work. Furthermore, our mechanism is at the application-level, and thus such network structure issues as NAT can be well addressed by employing the existing NAT traversal techniques (e.g., STUN[3]).

3.2 Shrinking Window Overview

Shrinking window is a core component that facilitates CODAS's collaborative and prioritized scheduling. Different from conventional sliding windows of fixed sizes, a shrinking window initially covers the entire short video, and its head

[3]http://nutss.gforge.cis.cornell.edu/stunt.php

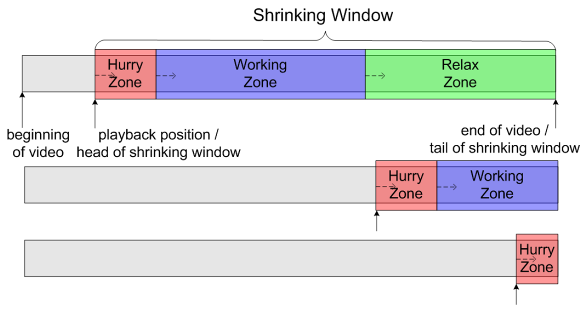

Figure 3: Illustration of a shrinking window

moves forward along with video playback. As the name suggests, it keeps shrinking until playback ends. We divide the shrinking window into three parts from head to tail: *hurry zone* (HZ), *working zone* (WZ) and *relax zone* (RZ). Figure 3 illustrates the concept of the shrinking window.

Our goal to separate different zones is to take the receiver's buffer level into account, so as to enable different degrees of aggressiveness for scheduling, which will be discussed in detail in the next section.

4. ZONE-BASED SCHEDULING

4.1 Hurry Zone (HZ) Scheduling

The Hurry Zone (HZ) is the most important part of the shrinking window, as it is used to guarantee a short startup delay and continuous playback. The HZ being not full indicates that the peer is suffering or about to suffer from delay, as some pieces near the playback position are missing. To avoid delay, such data pieces will be aggressively transmitted, so as to fill up the HZ as soon as possible. Specifically, the peer suppliers and the server will both prioritize the peer receivers whose HZ is not full.

It is intuitive that a large HZ size will increase server workload. Specifically, if the peer suppliers cannot provide enough rate, the server has to serve the peer, supplying a traffic volume of $L_H \cdot r_B \cdot \frac{r_S}{r_S + \sum r}$, where L_H is the size of the hurry zone in second, r_B is video's bitrate, r_S is the rate provided by the source, and $\sum r$ is the total rate provided by the other peer suppliers. This volume is monotonically increasing with the size of HZ, L_H.

Given that each peer supplier has only limited uploading capacity, a large HZ size will also increase peer churns. To understand this, suppose a peer supplier has a list of peer receivers to serve fairly, and each receiver requested a certain amount of data, which is a factor of the HZ size, the time to complete the transmission is thus also a factor of the HZ size. The service to a receiver that is not in HZ will be suspended if a new HZ request comes. Hence the probability that the peer supplier stops its service is proportional to the HZ size. Since the probability of receiving a new HZ request also increases with the HZ size, the chances that the receiver experiences poor service would be greatly affected by a large HZ. Furthermore, if large HZ degrades the service provided by the peer suppliers, the peer might turn to the server, which in turn increases the server workload.

On the other hand, a small HZ size also introduces overhead. In particular, a peer would frequently switch between HZ and WZ, which essentially amplifies the HZ requests. We now derive the lower bound of the HZ size, so as to guarantee playback continuity. Note that, when a peer detects some pieces in the last second of the HZ being missing, it needs to recover the missing pieces in $(L_H - 1)$ seconds to avoid any delay. Assume the receiver need t seconds to detect and request to the suppliers, we have $(L_H - 1 - t) \cdot (\sum r + r_S) > 1 \cdot r_B$, thus the minimum size of the HZ is $L_H > \frac{r_B}{\sum r + r_S} + 1 + t$. Here, t depends on the network condition (queueing and transmission delay) and the system capability (e.g., CPU, I/O and memory). Considering the above factors, our experience (on the PlanetLab network) shows that a few seconds for t is a reasonable setting. Assuming the server can provide enough bandwidth so that $r_S + \sum r > r_B$, the size of HZ is set to 5-12 seconds.

In summary, although the HZ size is small, its role is very important for guaranteeing the playback quality, and its size will not affect the playback quality, as long as it is large enough for detecting and recovering the missing pieces. Moreover, as the HZ size increases, the server workload will increase.

4.2 Working Zone (WZ) Scheduling

Next we discuss the scheduling in the working zone. When the hurry zone is full, it indicates that the receiver will not suffer from delay in a short time. Less aggressive scheduling will work in this case. Moreover, unless the server is highly over-provisioned, we do not suggest the server to serve requests from this zone. This is simply because the server is a reliable dedicated source, which should be preserved for much more important hurry zone requests.

The key objective for working zone scheduling thus becomes the accommodation of overlay dynamics. Specifically, with the pull-based scheduling, if a peer supplier leaves the system, the pieces that are requested from this supplier will be missing. Such missing pieces should be detected and re-requested in time. To this end, each piece in the working zone is set with a timer, which will timeout before the playback of this piece. If the piece has been downloaded, the timer of this piece will be no longer in use; if a timer timeouts, the peer will re-request this missing piece, along with the other pieces that are about to be downloaded.

Let the piece index be i and the current playback position be j. With a rate of $\sum r$ from the peer suppliers, we set the timer of the piece as $\frac{i-j}{\sum r} + c$, where c is an aggressiveness factor. The initial timer t_0 is roughly the expected time to receive the piece plus c. When the piece timeouts due to a missing piece, the current playback position has moved to $j_1 = (j + t_0 \cdot r_B)$, and the new timer for this piece should be $t_1 = \frac{i-j_1}{\sum r} + c$. Assume the piece is still missing, the third timer will be set, and so forth. In general, we have $\sum t_k = t_0 \cdot \frac{\sum r}{r_B} \cdot (1 - (1 - \frac{r_B}{\sum r})^{k+1})$.

Suppose the piece is the last one in the working zone, whose size is L_W, and the receiver can tolerate n timeouts before playing to this position, we have $\sum_{k=0}^{n-1} t_k \leq \frac{i-j}{r_B} = L_W$. It is easy to prove that if $c \leq 0$, $\lim_{n\to\infty} \sum_k^n t_k = L_W$, indicating the piece could tolerate infinite timeouts. Therefore, we define $c \geq 1$, indicating that the receiver can wait c seconds to set a new timer for this piece. After simplifying the inequality, we finally get $n \leq \frac{\log(L_W \cdot r_B + c \cdot \sum r) - \log(c \cdot \sum r)}{\log \sum r - \log(\sum r - r_B)}$.

From the equation we can compute the maximum number of timeouts as a function of the total rate and the distance between the piece and the playback position. For example, with $c = 2$, if the WZ size is 60 seconds and the receiver can obtain 192 KByte/s rate, the last piece in the WZ can tolerate 7 timeouts. We do not expect the maximum number of timeouts to be too much, because it makes the process of re-scheduling frequent and increases the control overhead.

When the working zone is also full, the peer enters the relax zone. As the name suggests, we use the least aggressive scheduling approach in this zone, because the peer is not likely to suffer from delay for a while. We do not set timer in the relax zone, either.

5. COLLABORATION OF PEER SUPPLIER AND SERVER

In this section, we further examine the contributions from peer suppliers and from the server.

For peer suppliers, an important control factor is max_U, the maximum number of the peers it can serve. The peer supplier will not accept any non-hurry downloading request if it currently is serving max_U peers. We define BW_U and BW_D as peer's upload and download capacity, respectively.

Suppose a peer supplier receives λ downloading requests per second, and it has no peer to serve initially. At time 0, the first peer comes and gets BW_U rate; at time $\frac{max_U - 1}{\lambda}$, the peer supplier is serving the maximum number of receivers with each obtaining the source rate of $\frac{BW_U}{max_U}$; and at time $\frac{max_U}{\lambda}$, the $(max_U + 1)$th receiver comes and the sender needs to stop serving the first receiver if the new one is in hurry. Assume that, at time $\frac{max_U}{\lambda}$, the sender has already sent S' data to the first receiver, and the actual required data amount is S, where $S > S'$. We have the probability of finishing serving the first peer before it is interrupted as $\frac{S'}{S}$. It follows that

$$S' = BW_U \cdot \frac{1}{\lambda} + \frac{BW_U}{2} \cdot \frac{1}{\lambda} + \ldots + \frac{BW_U}{max_U} \cdot \frac{1}{\lambda} = \frac{BW_U}{\lambda} \cdot \sum_{i=1}^{max_U} \frac{1}{i}.$$

It is clear that if the value of max_U is larger, the churn rate is lower. Yet it is easy to find that S' grows logarithmically with respect to max_U. Hence, S' will not be dramatically more as max_U increases. For the purpose of implementation, we will set max_U to 8.

On the other hand, the existence of the server is of magnificent importance in our system, and it plays a key role in the collaboration between senders and receivers. As mentioned, to alleviate the workload of the server, it will in general serve HZ requests only. To avoid the flipping between working and hurry zone, the receiver's requested amount of data is general larger than the HZ size, thus it is possible that the receiver changes to the working zone during receiving video data from the server. As such, the server will reduce the rate when it finds out that the receiver is not in the hurry zone. Specifically, the server will reduce δ rate for this receiver every second, and the rate will be reduced to the video's bitrate. Suppose the receiver requests r_S rate from the server and the requested data size is S_S. We have

$$\begin{cases} r_S - (k-1)\delta = r_B \\ \sum_{i=1}^{k} (r_S - (i-1) \cdot \delta) = S_S, \end{cases}$$

where k is the time to finish the transmission. The function can be easily solved as $\delta = \frac{r_S^2 - r_B^2}{2 \cdot S_S - r_S - r_B}$.

Moreover, we have the average server rate as $\frac{r_S + r_B}{2}$. Since the receiver requests the amount of data based on the rates provided by the suppliers, the workload for the server will be reduced given such a reduced rate. With a requested data volume of S, if the server sending rate is not reduced, its workload will be $\frac{r_S}{r_S + \sum r} \cdot S$, whereas the server workload after the reduction becomes $\frac{r_S + r_B}{r_S + r_B + 2 \cdot \sum r} \cdot S$. Suppose $\sum r + r_S = BW_U$, this yields a saving of

$$\Delta_{Workload} = 1 - \frac{BW_D \cdot (BW_D - \sum r + r_B)}{(BW_D - \sum r) \cdot (BW_D + \sum r + r_B)}.$$

Let $\frac{d(\Delta_{Workload})}{d(\sum r)} = 0$, we have the maximum save achieved at $\sum r = BW_U + B_r - \sqrt{2 \cdot BW_U \cdot B_r + 2 \cdot B_r^2}$.

6. PERFORMANCE EVALUATION

6.1 Configuration

The video dataset in the simulation is based on part of the PPVA trace, which includes 7000 videos. To emulate the real Internet environment, the peers are heterogenous in terms of bandwidth, roughly following the statistics in [5] as shown in Table 1. The video bitrate is set to 48 KByte/s (384 kbps), typical for video of PPVA-support sites.

	I	**II**	**III**
Downloading (KByte/s)	96	192	384
Uploading (KByte/s)	16	32	96
Distribution	0.2	0.2	0.6

Table 1: Bandwidth Capacity and Distribution

The user arrival rate follows a Poisson distribution. The duration a user stays in the system follows a Weibull distribution modeled from the PPVA trace. We assume the user will watch the video one after another, and when the duration is greater than a randomly assigned lifetime following the above distribution, the user will leave the system. As a result, the number of peers keeps increasing, so that we can study the scalability of the system. For other important settings, we set the size of WZ as 60 second, and a peer can upload to at most $max_U = 8$ peers.

We investigate several metrics, which together reflect the playback quality and the system scalability.

- *Startup delay.* We define that the peer can only start to play the video when the playback buffer contains the first two-second part of the video. Therefore, the startup delay is the time to search the sources and to fill up the two-second playback buffer.

- *Playback continuity.* We record the summation of all the watched videos' lengths for each peer D_{watch} and the delay it experienced during the playback D_{delay} (excluding the startup delay), thus we calculate the playback continuity as $\frac{D_{watch}}{D_{watch} + D_{delay}}$. The playback continuity being 1 indicates that the peer did not encounter any delay.

- *Normalized server bandwidth.* We record the traffic downloaded from the server T_{server}, and the total traffic T_{all}. We then calculate the normalized server bandwidth as $\frac{T_{server}}{T_{all}}$. The value of 1 means all the traffic is downloaded from the server.

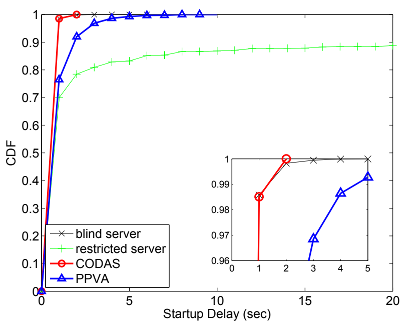

Figure 4: Cumulative distribution of startup delay (simulation result)

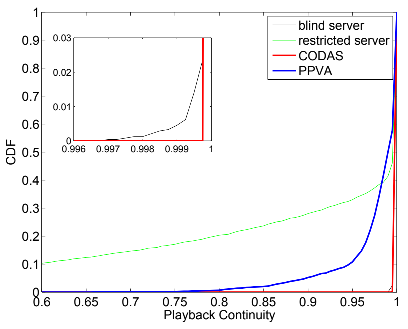

Figure 5: Cumulative distribution of playback continuity (simulation result)

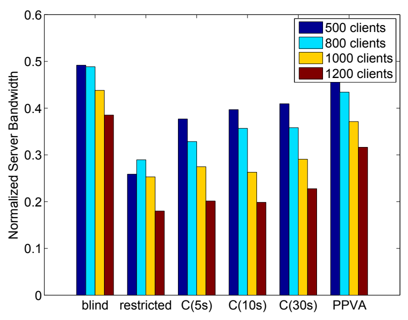

Figure 6: Comparison of normalized server bandwidth (simulation result)

For comparison, we have implemented two baseline schemes. The first one is a peer-to-peer system with a *blind server*, in which the server can provide any bandwidth at any time, and it is not aware of receiver's status. The second one is a system with a *restricted server*, in which the server can only provide bandwidth if there is no other peer supplier in the system. Obviously, the system with the restricted server needs to maintain much less server bandwidth, but it will significantly degrade the playback quality, causing long startup delays and discontinuous playbacks. On the other hand, the system with the blind server will achieve good playback quality but greatly increase the server workload.

We have also implemented a *PPVA-like scheme*, which utilizes a sequential-based scheduling with a sliding window. The key difference of PPVA-like scheme from CODAS is that, the receiver does not have a specific strategy for the pieces that are about to miss the deadline, as the strategy for different pieces in the sliding window is identical, although the pieces are requested sequentially. Moreover, the suppliers do not differentiate the status of the receivers.

6.2 Simulation Results

We first study the startup delay. Figure 4 shows the cumulative distribution function (CDF) of the startup delay. Nearly 10% of peers failed to start playing in the restricted server system, because they cannot get served by either server or other peers. Excluding the failed peers, most peers can start playing in 5 seconds. In the blind server system, most peers start playing in 2 seconds, yet CODAS performs even better, because the churns in the hurry zone, especially among peer suppliers, are well handled. We have compared different HZ sizes and found that the size does not affect the startup delay, as we discussed, the HZ sizes mainly affect the server workload (confirmed later), while not the playback quality. As a comparison, the PPVA-like scheme performs worse than CODAS, because although the PPVA-like scheme utilizes the sequential-based scheduling, the senders are not aware of the receiver's jeopardy of the delay due to the lack of collaboration, and the peer churn makes it worse.

Next we study the playback continuity. As shown in Figure 5, the restricted server system performs worst again, where some peers even have a continuity of less than 0.6 (the failed peers have already been filtered). When a receiver cannot get enough rate, it has to wait until a new peer supplier accepts its request, since the server will not accept it. In the blind server system, most peers have a smooth playback experience, yet CODAS is even better again, as it guarantees a 100% playback continuity. It is because in CODAS, the senders devote the rate to the receivers that are about to encounter delay, showing the character of collaboration and delay-awareness. We find that different HZ sizes also perform almost identically. PPVA-like scheme has worse performance than CODAS, as half of the peers experience some degrees of delays, yet very few peers suffer from long delay.

Last we investigate the server workload, as Figure 6 shows the normalized server bandwidth for different client populations. "C(5s)" indicates the system utilizing CODAS with HZ being 5 seconds, and similar to others. Not surprisingly, the blind server system and the restricted server system spend most and least of the server bandwidth, respectively. As a comparison, CODAS performs better than the PPVA-like scheme. This is because in CODAS, the server gives priority to the receivers in the hurry status. As discussed earlier, the HZ size affects the server workload, and the simulation results confirm this. Also from the figure we find that all of the approaches have shown the scalability over the client populations, and both the PPVA-like scheme and CODAS are more scalable than the naive blind server and restricted server system.

7. PROTOTYPE EXPERIMENT

To further investigate the performance of the CODAS design, we have implemented a prototype and conducted experiments over the PlanetLab network[4]. We have conducted a series of experiments with the number of participated nodes ranging from 50 to 380. We set the HZ size as 5 seconds and 10 seconds to conduct two series of experiments. Other configurations are similar to that in the simulations.

Figure 7 shows the startup delay. We find that the client population does not affect the startup delay of both approaches (thus we only plot one PPVA-like scheme result), and the HZ size does not affect it much, either. With CODAS, most videos are started within 5 seconds and 3.1 seconds on average, which is much better than that with PPVA-like scheme (7.6 seconds on average). The reason is well explained in the simulation. In addition, the server capacity is a factor in the experiment, since in the real world, the server is not as powerful as it is in the simulation.

[4]http://www.planet-lab.org

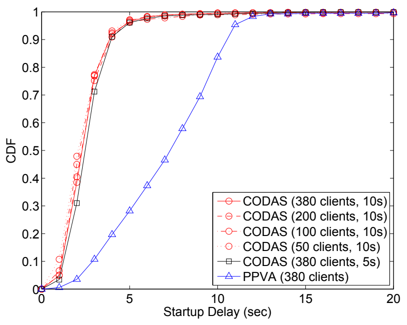

Figure 7: Cumulative distribution of startup delay (PlanetLab result)

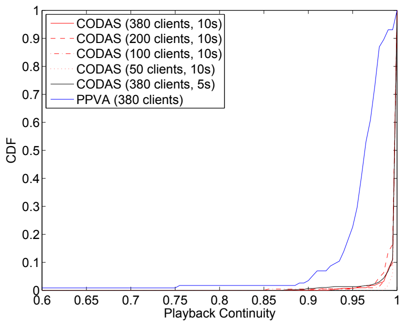

Figure 8: Cumulative distribution of playback continuity (PlanetLab result)

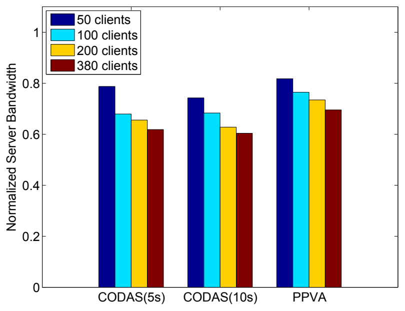

Figure 9: Comparison of normalized server bandwidth (PlanetLab result)

Figure 8 shows the playback continuity. Again, the population does not affect the performance. Different from the simulation results, CODAS cannot guarantee a 100% playback continuity, because in the real world, the latency between nodes is unexpected. Whereas, with CODAS, very few peers experience some playback delay, while with the PPVA-like method, almost all the peers encounter delays. The results confirm that our approach better accommodates peer dynamics by collaboration.

At last, Figure 9 plots the normalized server bandwidth of two CODAS approaches and the PPVA-like scheme with different client populations. The result is consistent with the simulation results, although the difference is not so clear as in the simulation due to the smaller client population. Interestingly, the performance of large HZ size is slightly better than the small HZ size, which is not consistent with the simulation result. This is because the experimental environment of PlanetLab is different from time to time.[5] However, when the client population grows, it behaves more like the results in the simulation.

8. CONCLUSION

The short length of the new generation UGC videos imposes much more stringent delay requirements for real-time streaming, which calls for sophisticated scheduling strategies in a peer-assisted sharing environment. In this paper, we proposed a novel collaborative delay-aware scheduling, CODAS, which explicitly prioritizes the data pieces of different urgencies through a zoned shrinking window. We illustrated detailed operations among peer receivers, suppliers, and the origin server, which collaboratively minimize the startup delay and improve playback continuity. Through trace-driven simulations and prototype experiments, we demonstrated that the CODAS approach greatly improves the playback quality, and noticeably reduces the server workload for peer-assisted UGC video sharing.

9. ACKNOWLEDGMENT

The authors would like to thank administrators and technicians of PPLive/PPVA, for providing PPVA traces.

[5]We have changed the server node each time because each PlanetLab node's traffic is limited. The selected server nodes have similar response time, workload, free CPU percentage, and unlimited bandwidth, and these statistic is monitored by CoMon (`comon.cs.princeton.edu`) in real time.

10. REFERENCES

[1] V. Aggarwal, R. Caldebank, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and F. Yu. The Effectiveness of Intelligent Scheduling for Multicast Video-on-Demand. In *Proc. of ACM Multimedia*, 2009.

[2] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In *Proc. of IEEE INFOCOM*, 2008.

[3] X. Cheng, J. Liu, and C. Dale. Understanding the Characteristics of Internet Short Video Sharing: A YouTube-based Measurement Study. *IEEE Transactions on Multimedia*, 2010.

[4] Y. Guo, C. Liang, and Y. Liu. Adaptive Queue-based Chunk Scheduling for P2P Live Streaming. In *Proc. of IFIP Networking*, 2008.

[5] C. Huang, J. Li, and K. Ross. Can Internet Video-on-Demand be Profitable? In *Proc. of ACM SIGCOMM*, 2007.

[6] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proc. of ACM SIGCOMM*, 2008.

[7] C. Liang, Y. Guo, and Y. Liu. Investigating the Scheduling Sensitivity of P2P Video Streaming: An Experimental Study. *IEEE Transactions on Multimedia*, 11(3):383–360, 2009.

[8] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, 2008.

[9] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized Decentralized Broadcasting Algorithms. In *Proc. of INFOCOM*, 2007.

[10] The Official YouTube Blog. Y,000,000,000uTube. http://youtube-global.blogspot.com/2009/10/y000000000utube.html, 2009.

[11] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *Proc. of IEEE ICNP*, 2006.

[12] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *Proc. of IEEE INFOCOM*, 2006.

[13] K. Xu, H. Li, J. Liu, W. Zhu, and W. Wang. PPVA: A Universal and Transparent Peer-to-Peer Accelerator for Interactive Online Video Sharing. In *Proc. of IEEE IWQoS*, 2010.

[14] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. of IEEE INFOCOM*, 2005.