Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 8, August 2019, pp. 2208-2216
DOI: 10.23940/ijpe.19.08.p21.22082216

Cloud-OM Patching: A Novel Video Stream Scheduling Scheme based on Hybrid Cloud-Overlay Architecture

Guangqian Kong*, Xun Duan, and Yun Wu

School of Computer Science and Technology, Guizhou University, Guiyang, 550025, China

Abstract

Patching is an effective multicast video stream scheduling technology that provides "real" VoD services. However, patching streams cannot be shared by other users, resulting in insufficient server scalability. In addition, IP multicast cannot be deployed on the Internet in a large scale; thus, the application of multicast stream scheduling technology is limited. Based on the above problems, this paper first proposes a hybrid cloud-overlay multicast architecture based on overlay network and cloud computing. It consists of three parts: cloud layer, overlay layer, and monitoring layer. Then, based on this architecture, a Cloud-OM patching stream scheduling algorithm is proposed, which combines patching stream sharing and multicast tree level time difference cache sharing technology. The experimental results show that Cloud-OM patching can effectively reduce the server bandwidth requirements and has better performance than standard patching, double patching, and batched patching, especially for popular videos.

Keywords: overlay multicast; cloud computing; video-on-demand; video stream scheduling

(Submitted on June 11, 2019; Revised on July 10, 2019; Accepted on August 6, 2019)

1. Introduction

With the rapid development of the Internet and digital video technology, video-on-demand (VoD) services are becoming more and more popular. For example, many VoD platforms, such as the personal video sharing website YouTube [1] and the educational video service MOOC [2], have gathered a large number of users. VoD is a streaming media application that uses streaming technology to provide users with on-demand video playback services by playing while downloading. The traditional VoD system usually guarantees the service performance by linearly increasing the server to ensure the availability of the video service. This method is costly and has poor scalability. Since then, distributed servers, IP multicast, P2P, and other technologies have been proposed and applied to VoD systems to better solve the problem of poor scalability, but they also have their own shortcomings, such as distributed server implementation costs, the inability to use IP multicast on a large scale in the Internet, and the difficulty of P2P technology to guarantee quality of service (QoS).

Overlay multicast (OM) implements a function like IP multicast by constructing a multicast delivery tree at the application layer. The biggest difference between overlay multicast and end-system multicast is that it is an application layer multicast technology with network infrastructure (proxy). The reliability, stability, and hardware and software performance of professional proxy servers are incomparable to end hosts. In addition, when deploying agents in a multicast overlay network, they can be installed at specific network locations to maximize system performance, which is impossible with end-system multicast. OM can effectively solve the problems of IP multicast limitations and the scalability of video services. On the other hand, in multicast streaming applications, the video stream scheduling scheme plays a key role in achieving network resource scheduling and system performance optimization. The most representative is the batching [3] proposed by Dan et al., which divides client requests into individual batching time windows. Requests in a time window are served by a multicast batching stream. Hua et al. proposed patching [4] on the basis of the batching solution to provide users with a "true" VoD service without an initial waiting delay by adding independent patching streams. Compared with the batching scheme, the patching scheme improves the sharing efficiency of regular multicast streams, reduces user waiting delays, and

* Corresponding author.
E-mail address: gq_kong@163.com

saves system resources. However, all the patching streams are served by a server-dedicated channel to a single user and cannot be shared with other users. Therefore, the scalability of the solution is still difficult to guarantee for popular videos.

In recent years, with the rise of cloud computing, more and more streaming media services have begun to migrate to the cloud. For example, Alibaba Cloud [5] provide streaming acceleration services, and Amazon provides VoD content delivery services [6]. Cloud computing uses a highly centralized data center design approach to centrally store resources on servers in the data center and provide cloud users with a "pay-as-you-go" price model. Cloud-based video services provide reliable, elastic, on-demand, and cost-effective resource delivery with the cloud's powerful computing, storage, and transport capabilities [7-9]. However, there are still many theoretical and practical challenges when migrating video streaming services to the cloud. For example, servers in the cloud have different capacities and lease strategies [10], and video stream servers need to accurately predict future requirements to minimize service costs [11]. Cloud-assisted video streaming services [12-15] are a cost-effective migration way to improve quality of service and quality of experience by renting cloud services based on the effective use of existing service facilities. However, how to take the advantages of cloud computing for video streaming services is still a hot issue that needs to be solved urgently.

In this paper, we propose a hybrid cloud-overlay architecture for video stream scheduling on VoD services. In our architecture, the cloud layer, overlay layer, and monitoring layer are separated. The cloud layer is responsible for the management of video resources such as video segments and videos metadata, the overlay layer is responsible for the management of multicast tree and the distribution of video segments, and the monitoring layer is responsible for user behavior monitoring, cloud service capacity prediction, and coordination between the cloud layer and the overlay multicast layer. Based on this architecture, a Cloud-OM patching video stream scheduling scheme is proposed. Our solution fully utilizes the overlay node cache and the video stream delivery time difference between the multicast tree level nodes to maximize the sharing of the regular multicast stream and the patching stream, thus effectively saving the bandwidth of the video servers in cloud.

This paper is organized as follows. First, in Section 2, we show the architecture, composed by a cloud layer, overlay layer, and monitoring layer. We describe the details of the Cloud-OM patching algorithm in Section 3. Next, our experimental results are presented in Section 4. Finally, in Section 5, we give our concluding remarks.

2. Hybrid Cloud-Overlay Architecture

In this section, we propose a hybrid cloud-overlay architecture that can effectively guarantee the QoS for VoD. The proposed architecture, composed of a cloud layer, overlay layer, and monitoring layer, is shown in Figure 1. The cloud layer is mainly responsible for storing and querying video metadata, encoding and decoding video files, segmenting videos, and issuing multicast streams and patching multicast streams. The overlay layer is mainly responsible for constructing overlay topology and multicast delivery trees and managing overlay nodes. The monitoring layer is mainly responsible for monitoring user behavior, predicting service capacity, and coordinating between the cloud layer and the overlay multicast layer.

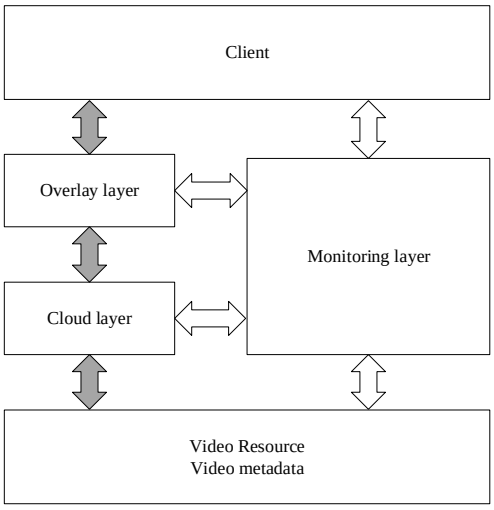

Figure 1. Hybrid Cloud-Overlay architecture

The cloud layer, shown in Figure 2, consists of interconnected video schedule nodes, video segment nodes, and metadata nodes. The video segment node is responsible for storing the encoded and segmented video data, and it is connected by a load balancer to implement load balancing between nodes. The metadata node is responsible for storing video metadata, including segment number, multicast stream ID, patching multicast stream ID, and other information. The video schedule node is responsible for receiving the information of the monitoring layer and storing it in the metadata node. At the same time, the video schedule node will issue the regular multicast stream and the patching multicast stream and deliver these streams to the overlay layer.

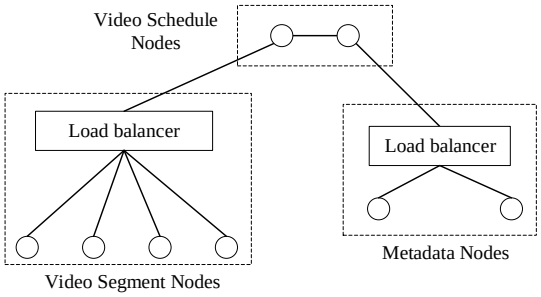

Figure 2. Cloud layer

The overlay layer, shown in Figure 3, is composed of overlay nodes and client nodes. First, the overlay mesh is constructed by the neighbor aggregation [16], and then the multicast tree is generated by the source tree method combined with the k-hop tree index [17].

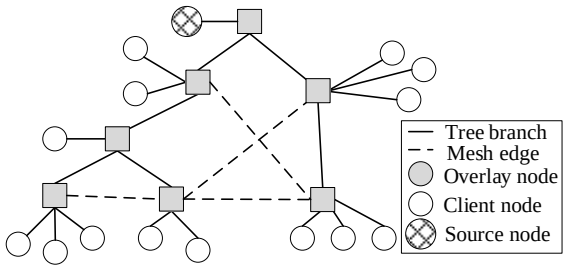

Figure 3. Overlay layer

The monitoring layer, shown in Figure 4, consists of collectors, a monitor, a recorder, an indicator, a query module, and a log module, which is implemented by dedicated servers. The collector includes three types: client request collector, cloud status collector, and overlay node status collector. It is mainly responsible for collecting information such as client video requests, cloud node status, and overlay node status. The collector contains traffic measurement system that identifies the type of traffic received [18]. The monitor calls sub-monitors to process the data obtained by collectors and the query module and then stores the relevant information into the recorder and the log. If it is necessary to instruct the cloud layer and the overlay layer to perform related actions, the relevant instructions are sent to the indicator.

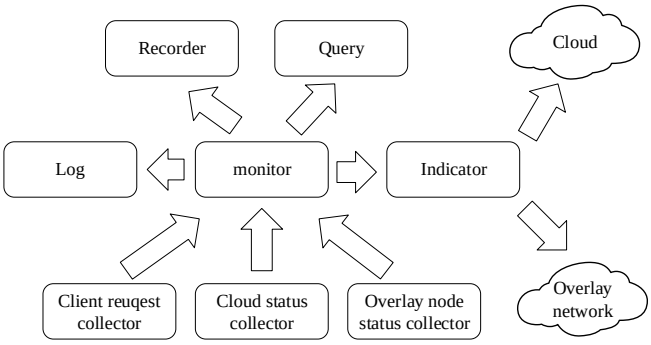

Figure 4. Monitoring layer

3. Video Stream Scheduling Scheme

For the sake of clarity and convenience, several terms and definitions of calculations have been defined as follows:

- V: video;
- B: client cache size (minutes);
- $|V|$: video length (minutes);
- N: segment number of video;
- d: segment length (minutes);
- b: video playback rate(bps);
- L: overlay multicast tree height;
- λ: request rate.

The overlay node cache is divided into a regular multicast stream cache and a patching multicast stream cache. These caches are all further divided into previous segment cache and current segment cache. The current segment cache is used to receive the stream data from the parent node. When a video segment is fully received in the current segment cache, the video segment is moved to the previous segment cache to provide multicast stream downloads to its child nodes. The overlay node cache structure is shown in Figure 5.

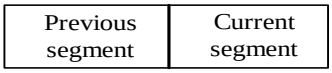

Figure 5. Overlay node cache structure

The main problem with the standard patching scheme is that one patching stream is only provided to one user, and other users cannot share the same patching stream. The double patching solution reduces the bandwidth consumption of the server by providing a long patching stream and a short patching stream, so that more user requests can share a long patching stream. However, this performance improvement is at the expense of adding additional video data, so it still increases the overhead of the server.

We propose a Cloud-OM (cloud-overlay multicast) patching algorithm based on hybrid cloud-overlay architecture. The algorithm constructs a multicast delivery tree in the overlay layer and issues regular multicast streams and patching multicast streams through the cloud layer. The time difference between the multicast levels and the overlay node caches enable efficient multicast video stream scheduling while reducing server bandwidth consumption.

The steps of the Cloud-OM Patching algorithm are as follows:

Step 1 The monitoring layer listens for the user request. If the request is the first request to the video V, the monitoring layer queries video metadata in the cloud layer; if not, proceed to step 5.

Step 2 The cloud layer returns video metadata to the monitoring layer, and the monitoring layer instructs the overlay layer to construct a multicast tree with the cloud server as a source point through the source tree method.

Step 3 The monitoring layer instructs the cloud layer to initiate a regular multicast stream RS_n and instructs the overlay layer to join the user to the level 1 overlay node.

Step 4 The overlay nodes at each level in the overlay layer receive multicast video streams in turn and store them in the regular stream RS_n cache according to the multicast tree structure.

Step 5 The monitoring layer checks the current status of the regular multicast stream, and if the video segment length it has sent $\geq$ the client default buffer upper limit B, then return to step 3.

Step 6 The monitoring layer checks the current list of the patching multicast stream. If no patching multicast stream that satisfies the user request is found, the monitoring layer instructs the cloud layer to initiate a patching multicast stream numbered PS_m and instructs the overlay layer to join the user to level 1 overlay node; otherwise, proceed to step 8.

Step 7 The overlay nodes at each level receive the patching multicast video stream in turn and store it in the patching multicast video stream PS_m cache.

Step 8 The monitoring layer instructs the overlay layer to join the user to the overlay node of the specified level and receives the regular multicast stream RS_n and the patching multicast stream PS_m.

The Cloud-OM patching scheme combines batched patching and multicast tree level time difference share technology.

Like batched patching, we also divide V into N segments of equal size, S_1, S_2, ⋯, S_N, so each segment size is $|V|/N$.

For example, the multicast tree height L = 3. When the server initiates a regular multicast stream RS_n, it will go from level 1 to level 2, and then to level 3 along the hierarchy of the multicast tree. The overlay nodes at each level put the previous segment full video into the "previous segment" cache for the child nodes to download. Therefore, the multicast stream between each level node has a time difference of one video segment. We illustrate the Cloud-OM patching algorithm by taking a new request in the t-segment of the regular video stream as an example. If there is no patching multicast stream available, the cloud server will initiate a new patching multicast stream PS_m with a length of t. Requests arriving at time $t – t + 3$ will share this patching multicast stream. For example, a request arriving at time $t + 1$ will join an overlay node on L2 level, sharing a regular multicast stream RS_n and a patching multicast stream PS_m. The client only needs two channels to receive the regular multicast stream and the patching multicast stream. When the request arrives after $t + L$, the server will initiate a new patching multicast stream PS_{m+1}.

Figure 6 shows the comparison of batched patching, double patching, and our scheme. Batched patching uses the method of sending the patched stream in batches, so that requests in one video segment time share the same patching stream, thus saving the bandwidth requirement of the server. Double patching uses a long patching stream to transmit more than just the regular patching data and uses a short patching stream to transmit the missing patching data. In this way, it can exchange a small amount of server resource "waste" for more efficient patching stream sharing. Our scheme combines batched patching's video segmentation technology and multicast tree inter-level time difference sharing technology to share the patching multicast stream more efficiently. For a client request arriving within $t – t + L$, the server sends only one patching multicast stream. For comparative purposes, we calculate the server mean bandwidth requirement of the patching multicast stream of each solution in the $t – t + L$ time period. The total server patching multicast stream bandwidth required for the batching patching scheme is: $\sum_{k=0}^{L-1}(t+k)\cdot|V|/N$. The total server patching multicast stream bandwidth requirement of double patching (for comparison, we assume $\omega = L$) scheme is: $(t + L + (1 + L)L/2)\cdot|V|/N = (t + L((3/2) + (L/2)))\cdot|V|/N$. Our scheme's server patching multicast stream bandwidth requirement is: $t\cdot$|V|/N. Therefore, compared with batched patching, the server patching multicast stream bandwidth ratio is: $t/(Lt + (L - 1)L/2)$ When $t >> L$, it can be approximated to $1/L$, so our scheme can achieve great server patching multicast stream bandwidth savings. When the multicast tree height L is large, such as L = 5, the server patching multicast stream bandwidth savings can reach 80%. Compared with double patching, the required server patching multicast stream bandwidth ratio is: $t/(t + 1.5L + L^2/2)$, which can achieve higher server patching multicast stream bandwidth savings when L is larger and t is smaller. For example, when t = 20 and L = 5, the server patching multicast stream bandwidth ratio is 1/2, which can save 50% of the server patching multicast stream bandwidth. Meanwhile, when t = 10 and L = 5, the server patching multicast stream bandwidth ratio is 1/3, which can save the server patching multicast stream bandwidth by 66.7%.

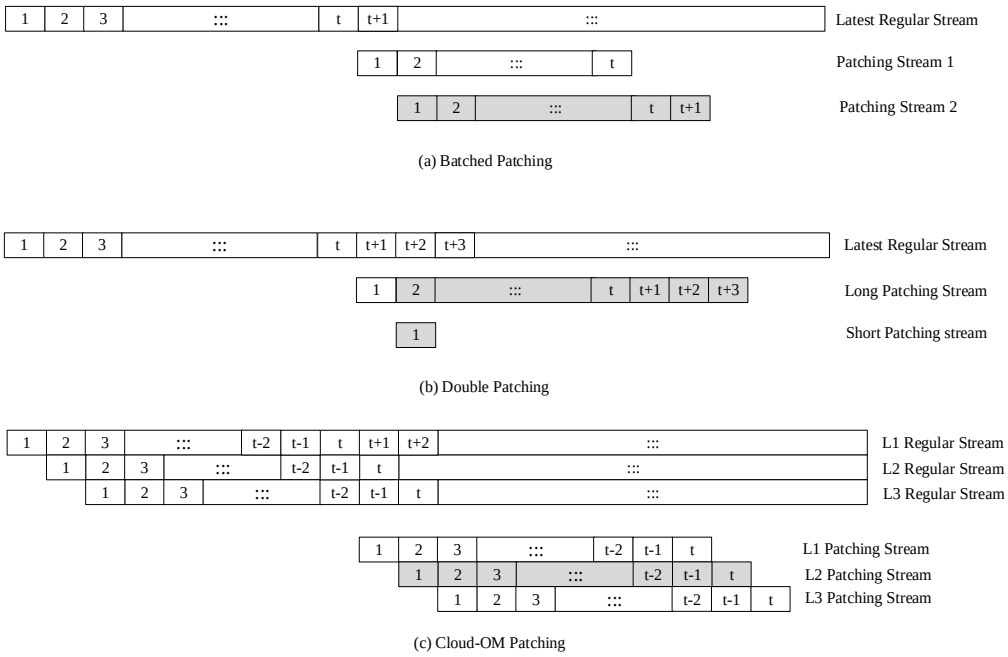

Figure 6. A comparison of batched patching, double patching, and Cloud-OM patching

4. Performance Study

In this section, we evaluate the performance of the proposed scheme through simulations. We focus on analyzing and comparing our scheme with double patching [19] and batched patching [20]. The server's mean bandwidth requirement is used as the performance metric.

4.1. Simulation Environment

Firstly, the performance parameters of the simulation test environment are shown in Table 1. For the sake of convenience of analysis, we assume that the video is encoded with MPEG-1 coding with an average playback rate of 1.5 Mbps and a length of 90 minutes. Moreover, the video is partitioned into 30 segments in default, and this parameter varies from 10 to 50. The default mean multicast tree height is 3, and its variation is 2 to 6. The client buffer size is 18 in default, and the range is 2 to 30. The arrival of the service requests follows a Poisson distribution with a mean arrival rate λ, and correspondingly the mean request interval is $1/\lambda$, which varies from 5 seconds to 95 seconds.

Table 1. Performance parameters

Parameter	Default	Variation
Normal playback rate b (Mbps)	1.5	N/A
Mean multicast tree height L	3	2 - 6
Video length $\|v\|$ (minutes)	90	N/A
Video segment number	30	10 - 50
Client buffer size B (minutes)	18	2 - 30
Mean request interval $1/\lambda$ (seconds)	50	5 - 95

4.2. Effect of Mean Request Interval

In this subsection, we compare the changes in the mean bandwidth requirement of the server among our solution, double patching, and batched patching at different mean request arrival intervals, as shown in Figure 7. We observe that double patching requires a higher mean server bandwidth when the mean request arrival interval is smaller, and the mean server bandwidth requirement gradually decreases as the request interval increases. Batched patching has no change in mean server bandwidth requirements from 5 to 95 of mean inter-request arrival time, and it remains at around 8Mbps. This is mainly because batched patching uses segmentation technology to send only one patching stream for multiple requests in a video segment; therefore, it can maintain a fixed mean server bandwidth requirement. In our solution, we combine video segmentation technology and multicast tree inter-level sharing technology. It can obtain a fixed mean server bandwidth requirement just like batched patching, and it can save more than 40% of mean server bandwidth requirements in our simulation scenario. This result shows that our solution is very suitable for popular video programs with high mean request arrival rates. By sharing the patching multicast streams and using the overlay node cache, the sharing of the regular multicast stream is more effectively realized, thus obtaining more server bandwidth savings and higher scalability.

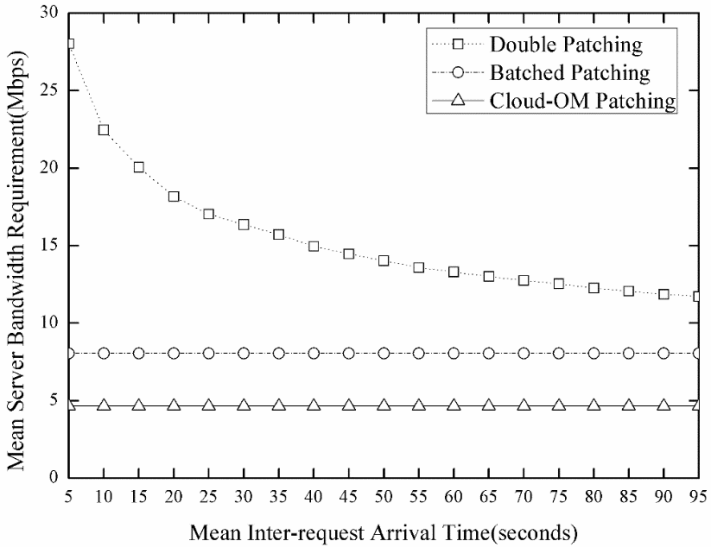

Figure 7. Effect of mean request interval

4.3. Effect of Client Buffer Size

In this subsection, we compare our solution with double patching and batched patching to observe the relationship between client buffer size and mean server bandwidth requirements, as shown in Figure 8. When the client buffer size is small, such as $B = 2$ minutes, all three solutions need a large server bandwidth requirement, especially double patching. This is because when the client buffer is small, the regular multicast stream data that the client can cache is limited, causing the server to initiate a large number of new regular multicast streams in response to the user's request. When the client buffer is gradually increased, the scheme and batched patching can obtain larger server bandwidth savings, mainly because the number of regular multicast streams and patching multicast streams issued by the server is reduced by sharing one patching multicast stream through a batch of requests. At the same time, because our solution further uses the multicast tree node cache to share the regular multicast stream and the patching multicast stream within the inter-level time difference, it can obtain higher mean server bandwidth savings than double patching and batched patching.

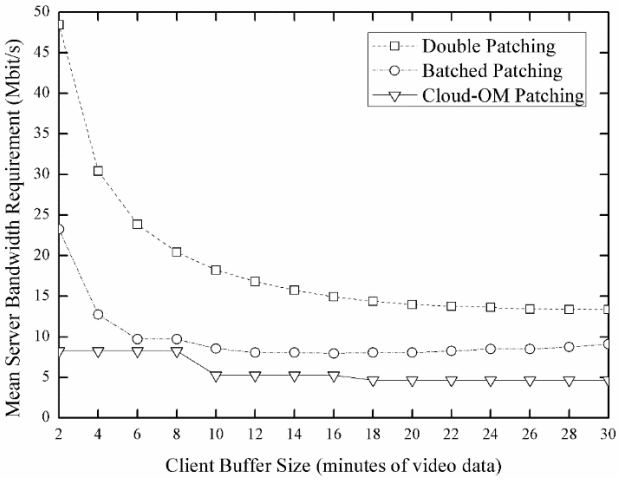

Figure 8. Effect of client buffer size

4.4. Effect of Video Segment Number

In this subsection, we compare our solution with batched patching scheme that also uses video segmentation technology to observe the effect of the number of video segments on the mean bandwidth requirement of the server, as shown in Figure 9. We assume that the mean height of the multicast tree is $L = 3$ and the client cache is $B = 18$ minutes. When the number of segments $N = 10$, the mean server bandwidth requirement of batched patching is about 4.7 Mbps, and our scheme is about 3.3Mbps. As N increases, the mean server bandwidth requirement increases. However, the mean server bandwidth requirement of batched patching increases much faster than our solution. For example, when $N = 50$, the server bandwidth requirement of batched patching is about 11Mbps, while our solution is about 5.9Mbps, which saves about 46% of the server bandwidth compared to batched patching. This result shows that our scheme makes full use of the multicast tree node cache at all levels, which effectively reduces the patching multicast stream number of server issues.

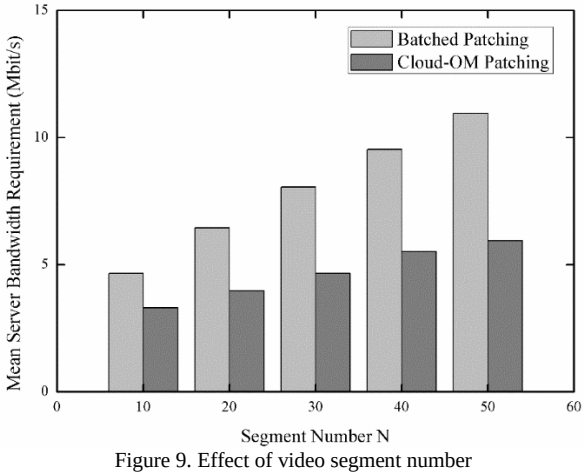

Figure 9. Effect of video segment number

4.5. Effect of Multicast Tree Height

The effect of the multicast tree height on the mean server bandwidth requirement is shown in Figure 10. When the multicast tree height L = 2, the mean server bandwidth requirement is about 5.6Mbps. As L increases, mean server bandwidth requirements gradually decrease linearly. When L = 6, the mean server bandwidth requirement drops to approximately 3.3 Mbps. It can be seen from the above results that the mean server bandwidth requirement is greatly reduced by using the overlay node cache and the time difference sharing between the multicast tree levels. However, as the height of the multicast tree increases, the complexity of the overlay network increases, which also causes problems such as high delay and poor system reliability. Therefore, there is a trade-off between system scalability and performance.

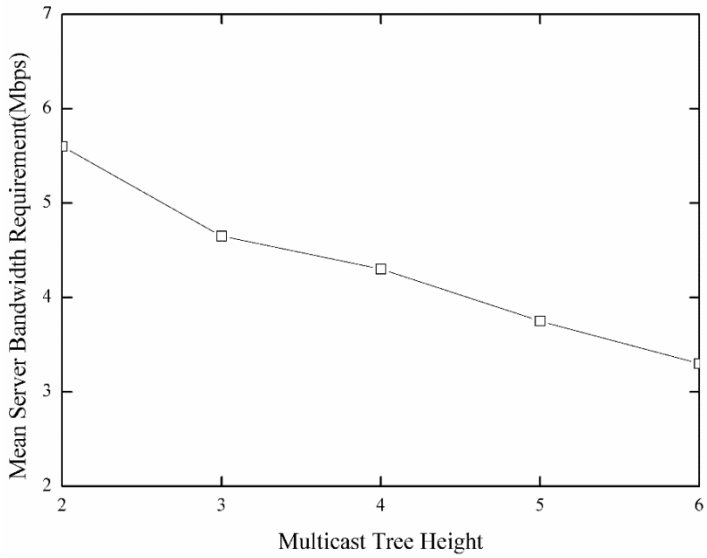

Figure 10. Effect of multicast tree height

5. Conclusions

In this paper, we propose a hybrid cloud-overlay architecture consisting of a cloud layer, an overlay layer, and a monitoring layer. By constructing a multicast tree in the overlay layer, the segmented video and video metadata are stored in the cloud layer; the client request monitor, overlay network monitor, and cloud status monitor are implemented through the monitoring layer; and video segmentation, video metadata maintenance, and multicast stream scheduling are implemented through the cloud layer. On this basis, the Cloud-OM patching video stream scheduling scheme is proposed. This scheme can effectively utilize the patching multicast stream to enable multiple users to share the same patching multicast stream. At the same time, the multicast tree structure can be effectively utilized, and the patching multicast stream can be shared more by using the multicast tree inter-level time difference sharing technology, thereby greatly reducing the server bandwidth requirement. The experimental results show that our solution saves more than 42% of the mean server bandwidth consumption than batched patching, and it saves more than 60% of the mean server bandwidth consumption than double patching when N = 30 and L = 3. The results indicate that our solution is an effective and scalable multicast video streaming scheduling scheme.

Acknowledgments

This research is supported by the Natural Science Foundation of China (No. 61741124) and the Science and Technology Plan Project of Guizhou Province (Qiankehe Platform Talent) (No. 5781).

References

1. "Youtube Website," (http://www.youtube.com, last accessed on December 15, 2018)
2. G. W. Matkin, "Open Educational Resources in the Post Mooc Era," *eLearn*, Vol. 2013, No. 4, 2013
3. A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an on-Demand Video Server with Batching," in *Proceedings of the 2nd ACM International Conference on Multimedia*, pp. 15-23, 1994
4. K. A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," in *Proceedings of the Sixth ACM International Conference on Multimedia*, pp. 191-200, 1998
5. "Big Video Solution," (https://media.aliyun.com/?utm_medium=text&utm_source=baidu&utm_campaign=hyy&utm_content

=se_119785, last accessed on December 15, 2018)
6. "Amazon Elastic Compute Cloud," (http://aws.amazon.com/ec2/, last accessed on December 15, 2018)
7. Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "Cloudmedia: When Cloud on Demand Meets Video on Demand," in *Proceedings of International Conference on Distributed Computing Systems*, pp. 268-277, 2011
8. W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia Cloud Computing," *IEEE Signal Processing Magazine*, Vol. 28, No. 3, pp. 59-69, 2011
9. Q. L. Han, S. Liang, and H. L. Zhang, "Mobile Cloud Sensing, Big Data, and 5G Networks Make an Intelligent and Smart World," *IEEE Network*, Vol. 29, No. 2, pp. 40-45, 2015
10. A. Li, X. Yang, S. Kandula, and M. Zhang, "Comparing Public-Cloud Providers," *IEEE Internet Computing*, Vol. 15, No. 2, pp. 50-53, 2011
11. D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-Assured Cloud Bandwidth Auto-Scaling for Video-on-Demand Applications," in *Proceedings of the IEEE INFOCOM*, pp. 460-468, 2012
12. J. He, D. Wu, Y. Zeng, X. Hei, and Y. Wen, "Toward Optimal Deployment of Cloud-Assisted Video Distribution Services," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 23, No. 10, pp. 1717-1728, 2013
13. H. Li, L. Zhong, J. Liu, B. Li, and K. Xu, "Cost-Effective Partial Migration of VoD Services to Content Clouds," in *Proceedings of IEEE 4th International Conference on Cloud Computing*, pp. 203-210, 2011
14. X. Cheng and J. Liu. "Load-Balanced Migration of Social Media to Content Clouds," in *Proceedings of the 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 51-56, 2011
15. V. Rocha, F. Con, and R. Cobe, "A Hybrid Cloud-P2P Architecture for Multimedia Information Retrieval on VoD Services," *Computing*, Vol. 98, No. 1-2, pp. 73-92, 2016
16. A. Nakao, L. Peterson, and A. Bavier, "A Routing Underlay for Overlay Networks," *Computer Communication Review*, Vol. 33, No. 4, pp. 11-18, 2003
17. Y. Wang, H. Z. Wang, J. Z. Li, and H. Gao, "Graph Similarity Join with K-Hop Tree Indexing," in *Proceedings of ICYCSEE 2015*, pp. 38-47, 2015
18. G. Sun, F. Lang, and M. Yang, "Traffic Measurement System based on Hybrid Methods," *Electric Machines & Control*, Vol. 15, No. 6, pp. 91-96, 2011
19. Y. Cai, W. Tavanapong, and K. A. Hua, "A Double Patching Technique for Efficient Bandwidth Sharing in Video-on-Demand Systems," *Journal of Multimedia Applications and Tools*, Vol. 32, No. 1, pp. 115-136, 2007
20. Y. Liu, S. Yu, and J. Zhou, "Adaptive Segment-based Patching Scheme for Video Streaming Delivery System," *Computer Communications*, Vol. 29, No. 11, pp. 1889-1895, 2006

Guangqian Kong received his Ph.D. from Guizhou University in 2009. He is currently an associate professor, graduate supervisor, and member of the China Computer Society. His research interests include computer networks, cloud computing, software defined networks, and deep learning and its applications.

Xun Duan received his Ph.D. from Guizhou University in 2007. He is currently an associate professor and graduate supervisor. His research interests include distributed computing and cloud computing and its applications.

Yun Wu received his Ph.D. from Guizhou University in 2009. He is currently an associate professor, graduate supervisor, and member of the China Computer Society. His research interests include distributed computing, game theory, recommender system, and big data and its applications.