On-Demand Video Dispatch Networks:
A Scalable End-to-End Learning Approach

Damao Yang
Bilibili Inc.
yangdamao@bilibili.com
adrian.y.dm@gmail.com

Sihan Peng
NYU Shanghai
sp4059@nyu.edu

He Huang
Purdue University
hhuang1206@gmail.com

Hongliang Xue
xuehongliang518@gmail.com

Abstract

We design a dispatch system to improve the peak service quality of video on demand (**VOD**). Our system predicts the hot videos during the peak hours of the next day based on the historical requests, and dispatches to the content delivery networks (**CDN**s) at the previous off-peak time. In order to scale to billions of videos, we build the system with two neural networks, one for video clustering and the other for dispatch policy developing. The clustering network employs autoencoder layers and reduces the video number to a fixed value. The policy network employs fully connected layers and ranks the clustered videos with dispatch probabilities. The two networks are coupled with weight-sharing temporal layers, which analyze the video request sequences with convolutional and recurrent modules. Therefore, the clustering and dispatch tasks are trained in an end-to-end mechanism. The real-world results show that our approach achieves an average prediction accuracy of 17%, compared with 3% from the present baseline method, for the same amount of dispatches.

I. Introduction and Related Work

A typical architecture of Internet VOD is depicted in Figure 1(a). Given the knowledge of the relation between users and videos as well as of the relation between users and CDNs, one key issue as Figure 1(b) is to dispatch videos to CDNs. Improper dispatches cause poor video service qualities. It is noticed that five main challenges occur with the emerging popularity of VOD: (1) large video amount (currently 50 million) compared with limited storage capacity (800,000 videos per CDN on average) and dispatch bandwidth (30,000 videos per day on average); (2) ever-increasing uploaded videos (60,000 videos per day on average) every day; (3) notable differences between peak (1,667,000 qps) and off-peak (235,000 qps) requests and video preferences; (4) erratic variants of user requests; (5) implicit and intrinsic request relations among users due to different populations, economies and cultures. Bearing the above challenges in mind, we aim at a scalable video dispatch system working at off-peak time to enhance the peak service qualities on the next day.
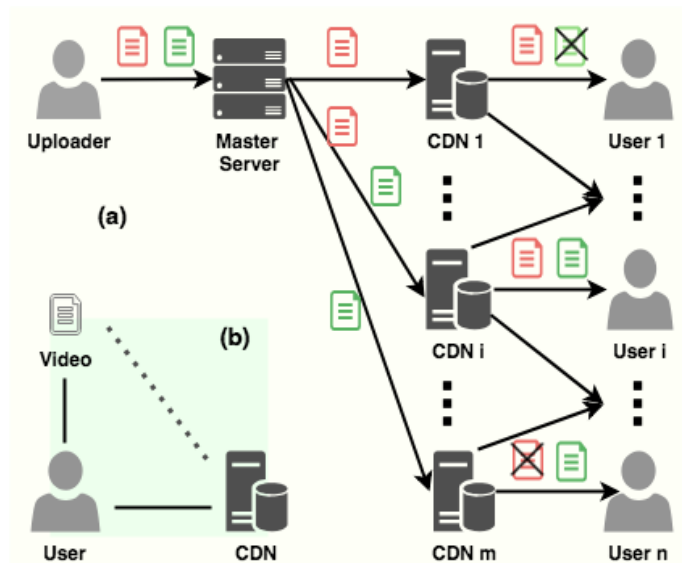

Fig. 1. Video Dispatch System and Problems

Our present baseline method triggered the dispatch of video v to CDN i, when the requests of v from the users served by i exceed a threshold h for a period p. This implies that one successful service corresponds to h CDN misses. Besides, dispatch at

peak time occupies network bandwidth and resources, which affects the peak service quality. We apply two ratios as evaluation metric: r_{whole} as the number of requested videos after dispatched within a whole day over the total dispatch number, and r_{peak} as the number of peak requested dispatched from several hours ago till recently over the total dispatch number. While r_{whole} can be optimized by fine tuning h and p, r_{peak} stays low, meaning that the baseline method performs poor for peak service quality. This is due to difficulty (3) and (4), and the baseline method fails to explore the relation between peak and off-peak requests, only to exploit a very short memory of requests without knowing the longer changing pattern.

Recurrent neural networks (**RNN**s) memorize state transition efficiently thanks to their recursive structures with hidden state storage. RNN has been applied to predict noisy and non-stationary time series, where the series are unsupervised learned before sent to RNN in case that RNN prefers short-term dependencies than long-term dependencies caused by vanishing gradients. Long short-term memories (**LSTM**s) and gated recurrent units (**GRU**s) are proposed with learnable gates to control information flowing in order to alleviate this problem. Both LSTM and GRU have achieved the state-of-the-art performances in sequential applications. A common point in the above sequential models is that every single point in the series weighs equally importantly, and therefore they are vulnerable to noise and prone to overfitting to specific series patterns.

Convolutional neural networks (**CNN**s), which are typically implemented with globally shared nonlinear filters followed by pooling operations, strengthen and extract local features. CNN has formed the crux of computer vision applications. CNN is usually followed with max pool to subtract the most representative features for each local region.

Time series analysis with CNN also emerges in recent years. In this paper, we put a top-k pool ahead of CNN to extract the peak-time pattern, place three max pools behind CNN to extract the inter-day pattern, and concatenate CNN with RNN to learn the dynamics with low complexity and in avoidance of overfitting compared with pure RNN. In Wavenet, causal CNN without pools were proposed to generate predicted 1D signals, forcing the filters always to calculate on the previous points.

It is noticeable that in the above researches, all the input and output elements value equally. However, the peak-time-oriented dispatch problem focuses more on the peak elements than the off-peak ones. Besides, the predicted peak requests are not necessarily precise in an exact time, since the peak time sustains in a period. This motivates us to design our algorithm to differentiate peak and off-peak request patterns.

A. Ranking Problems

At first glance, video dispatch can be solved the same way as a variation of 0/1 knapsack problem. Metaheuristic techniques are guaranteed neither an optimal solution, nor polynomial prediction time complexity. Neural network methods, such as Hopfield Net, and recently proposed Pointer Net, though have been successfully verified in similar combinatorial optimization applications, suffer from either scalability or low-dimension restrictions.

Alternatively, the dispatch problem is to pick top-k videos by the probabilities of the future requests from the users served by certain CDN servers. This process simulates a ranking problem. Learning to rank (**L2R**) has produced a much better performance than traditional methods. The L2R algorithms are categorized as pointwise, and other techniques. The pointwise ranking focuses on total and absolute orders. This works for the dispatch problem because the strategies cannot be made until the total video requests have been seen, while the pairwise ranking aims at partial orders and the listwise ranking costs too high complexity for large scale data set.

B. Clustering and Manifold Learning

In the ranking or time series forecasting problems, the set of ranked items or series are predetermined. The VOD provider, however, is facing new videos all the time, which is nontrivial in dispatch problems. Clustering methods categorize data with distance-based metric without human labelling so that new videos as well as present ones are represented with a nearest centroid. To cope with "curse of dimensionalit", manifold learning projects features into a low dimensional subspace, uncovering the intrinsic distribution of the input data in visible and "compact" way. Autoencoder, as a nonlinear dimensionality reduction method, was proposed to pretrain deep neural networks. It and variations have been applied as preprocessing for clustering.

Distributed clustering algorithms were proposed for large data set. Unfortunately, they do not maintain consistent clustering indices at training and predicting iterations. We propose a supervised-like clustering modification to solve this problem.

C. End-to-end Learning

Recently it has been a fashion to train multi-task neural networks in an end-to-end learning mechanism for better performances rather than to optimize them independently. Within the ideal end-to-end learning trait, all learnable parameters are differentiable

with respect to the ultimate task's losses (in supervised learning) or rewards (in reinforcement learning). One strategy is to carefully design the structure of these networks so that they are connected by shared components. For example, Faster Rcnn combined region proposal with objection recognition by a shared CNN part as feature extractors, and led to globally optimized object detection results; value iteration networks introduced a fully differentiable CNN planning module to approximate the value iteration algorithm, simultaneously approximating the Markov decision process and optimizing the control.

D. Our Contributions

1) An end-to-end learning mechanism which fulfills video clustering and dispatch tasks.
2) A novel structure for time series forecasting which utilizes both peak-time and whole-day temporal features.
3) A supervised-like clustering method fit for the mini-batch gradient descent (**MBGD**) training and the distributed environment, and therefore scalable for big data.
4) State-of-the-art performances under the real-world VOD system.

E. Notations

The symbols in this paper are defined as follows:

D denotes the constant as the number of consecutive days when training or predicting a batch of data.

T denotes the constant as the number of time intervals within a day.

K denotes the constant as the number of peak time intervals within a day.

U denotes the set of the users.

I denotes the set of CDNs, and I_u the set of the CDNs that serve user u.

V denotes the set of the videos, and V_i the set of the videos dispatched to CDN i

R denotes the set of the requested videos within a day, R_u the set of the requested videos from the user u within a day, R^d the set of the requested videos on day d, and R_d^u the set of the requested videos by the user u on day d. PR denotes the set of the requested videos at peak time, and similarly with PR_u, PR_d, and PR_d^u.

C denotes the mapping from the video to its cluster, and $C_{(r)}$ denotes the cluster of the video r. CP denotes the probabilities of the clustered videos dispatched to the CDNs, and $CP_{C_{(r)},i}$ the probability of the cluster of the requested video r dispatched to IDC i. CPT denotes the upper bound of the sum of clustered dispatch probabilities to the CDNs.

$|A|$ denotes the cardinality of the set A.

$\Re(f)$ denotes the cardinality of the range of the mapping f.

$A1 \times A2$ denotes the Cartesian product of two sets $A1$ and $A2$.

$\otimes$ represents convolution operator.

$top(K|S)$ represents extracting the K largest elements out of S according to the ordered set given by S.

$sort(V)$ represents sorting V in descending order.

$maxpool(M, r, c)$ represents to extract the max element of each submatrix $M'_{r \times c}$ of the matrix M, and reshape the output to 1D.

$ReLU(x)$, $sigmoid(x)$, and $tanh(x)$ are the nonlinear activation functions, referring to rectifier, sigmoid, and hyperbolic tangent functions.

$||v||_2$ denotes the L2-norm of v.

II. PROBLEM FORMULATION

$$\min_{C,CP} \sum_{u \in U} \sum_{r \in PR_u^{D+1}} |r| \prod_{i \in I_u} \left(1 - CP_{C_{(r)},i}\right) \big| R^1 \ ... \ R^D$$

$$s.\ t. \sum_{r \in \bigcup_{u \in U} \bigcup_{d=1}^{D} R_u^d} CP_{C_{(r)},i} \le CPT_i, \forall i \in I$$

$$0.0 \leq CP_{C(r),i} \leq 1.0, \forall r \in \bigcup_{u \in U} \bigcup_{d=1}^{D} R_u^d, \forall i \in I$$

$$C(r) \in C, \forall r \in \bigcup_{u \in U} \bigcup_{d=1}^{D} R_u^d, \forall i \in I$$

III. THE STRUCTURE OF DISPATCH NEURAL NETWORKS

We set up two neural networks, clustering network and policy network, as depicted in Figure 2. The clustering network is composed of temporal layers and clustering layers. The policy network comprises accumulation layers, temporal layers and policy layers. The two networks are coupled with the temporal layers, which have identical structures and shared weights. The outputs of the clustering network and the primitive inputs constitute the inputs of the policy network. The outputs of the policy network are the probabilities of the videos dispatched to users rather than to CDNs. The probabilities of the videos dispatched to CDNs will be calculated in a following "post-processing" procedure.
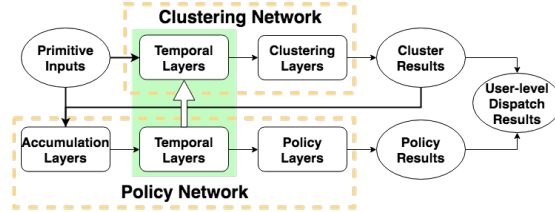

Fig. 2. Structure of the Two Coupled Networks

A. Primitive Inputs

The primitive inputs for one video are its request sequences. Each sequence represents the requests from a user on T intervals of D days. Therefore, the primitive inputs have a user dimension and a time dimension with intraday and interday parts. The inputs of video v is denoted as follows, where $x_u^{t,d} \geq 0$ denotes the requested number of v from the user u on interval t on day d.

$$X_{(v)} = \begin{pmatrix} \left(x_1^{1,1} \ ... \ x_1^{T,1} \right) & ... & \left(\left(x_1^{1,D} \ ... \ x_1^{T,D} \right) \right. \\ \vdots \\ \left(x_{|U|}^{1,1} \ ... \ x_{|U|}^{T,1} \right) & ... & \left(\left(x_{|U|}^{1,D} \ ... \ x_{|U|}^{T,D} \right) \right) \end{pmatrix}_{(v)}$$

We define $X_u^{t,d} = \left(x_u^{1,d} \ ... \ x_u^{T,d} \right)$ for convenience.

The primitive inputs exclude the information other than video requests, such as video metadata, weekends and holidays. There are three reasons: (1) the mapping from video content to user request is not bijection due to redundant uploads; (2) labelling the videos with tags may be subjective and ambiguous; (3) the video requests are highly nonstationary processes, and modeling with other observations is vulnerable to overfitting.

B. Temporal Layers

The temporal layers process the time dimension of the inputs. There are two components, CNN module for the intraday part and RNN module for the interday part, as depicted in Figure 3. The temporal layers are designed to predict the future requests with daily peak-time features and whole-day trajectories.

The CNN module has two parallel groups of convolutional filters with pools, which we call peak convolutional part (**PCP**) and mean convolutional part (**MCP**) respectively. PCP and MCP take X_u^d as a batch.

In PCP, the inputs are processed with a global top-K pool sorted by the total requests at each time. Then 1D convolutions are calculated.

$$XK_u^d = top\left(K \middle| argsort\left(\sum_{x \in V} X_u^d \right) \right)$$

4

$$YP_u^d = ReLU(W_{peak} \otimes XK_u^d + b_{peak})$$

W_{peak} and b_{peak} are learnable parameters.

In MCP, the inputs are passed via 1D convolution filters.

$$YMC_u^d = ReLU(W_{mean} \otimes X_u^d + b_{mean})$$

W_{mean} and b_{mean} are learnable parameters.

Then YMC_u^d is reshaped to three matrices $Row_u^d, Column_u^d, Block_u^d$: $R_{row} \times R_{column}$, $C_{row} \times C_{column}$, and $(B_{row1} \times B_{row2}) \times (B_{column1} \times B_{column2})$, where $T = R_{row} \times R_{column} = C_{row} \times C_{column} = (B_{row1} \times B_{row2}) \times (B_{column1} \times B_{column2})$, and $K = R_{row} = C_{column} = (B_{row1} \times B_{column1})$ The rows are closely related to hours, and the columns to minutes.

These matrices are processed with 2D max pools and weightedly summed:

$$YR_u^d = maxpool(Row_u^d, 1, R_{column})$$

$$YC_u^d = maxpool(Column_u^d, C_{row}, 1)$$

$$YB_u^d = maxpool(Block_u^d, B_{row2}, B_{column2})$$

$$YM_u^d = w_{row}YR_u^d + w_{column}YC_u^d + w_{block}YB_u^d$$

w_{row}, w_{column}, and w_{block} are learnable parameters.

The outputs of the CNN module, C_u^d, are the concatenation of YP_u^d and YM_u^d.

The RNN module adopts GRU, and takes C_u^d as the dth input for user u. The dth output R_u^d is computed as :

$$z_u^d = sigmoid(W_z C_u^d + U_z R_u^{d-1} + b_z)$$

$$r_u^d = sigmoid(W_r C_u^d + U_r R_u^{d-1} + b_r)$$

$$R_u^d = (1 - z_u^d)R_u^{d-1} + z_u^d tanh(W_h C_u^d + U_h(r_u^d R_u^{d-1}) + b_h)$$

W_z, U_z, b_z, W_r, U_r, b_r, W_h, U_h, and b_h are learnable parameters.

The outputs of the temporal layers from user u are R_u^D. We denote the outputs for video v from all users as $T_{(v)} = \left(R_1^D \;...\; R_{|U|}^D \right)_{(v)}$.

C. Clustering Layers

The clustering layers process the user dimension of the inputs. There are four components, normalization module, autoencoder module, clustering module and loss module.

The normalization module scales the inputs $T_{(v)}$ according to L2-norm, as $NT_{(v)}$.

The autoencoder module adopts a feedforward structure, which encodes $NT_{(v)}$ to $E_{(v)}$ on the 2D plane $Enc^2 = (-1.0, 1.0) \times (-1.0, 1.0)$, and then decodes back to $D_{(v)}$.

In the clustering module, we divide Enc^2 into $|\Re(C)|$ hierarchical blocks B^2. We construct B^2 as Figure 4 and follows:

1) Let $rintv = (-1.0, 1.0)$, and NDH the number of divisions per right-side hierarchy. Let $RIntv^1 = \phi$.
2) Divide $rintv$ uniformly into NDH open sub-intervals Sub^1. Let $\alpha = \left(0.0, \frac{1.0}{NDH}\right) \in Sub^1$. Insert $Sub^1 \setminus \{\alpha\}$ into $RIntv^1$.
3) Let $rintv = \alpha$. Repeat 2 until $|RIntv^1| \geq \frac{1}{2}|\Re(C)|^{1/2}$.
4) Let $Intv^1 = \{\alpha, (-\alpha_R, -\alpha_L)| \; \alpha \in RIntv^1\}$
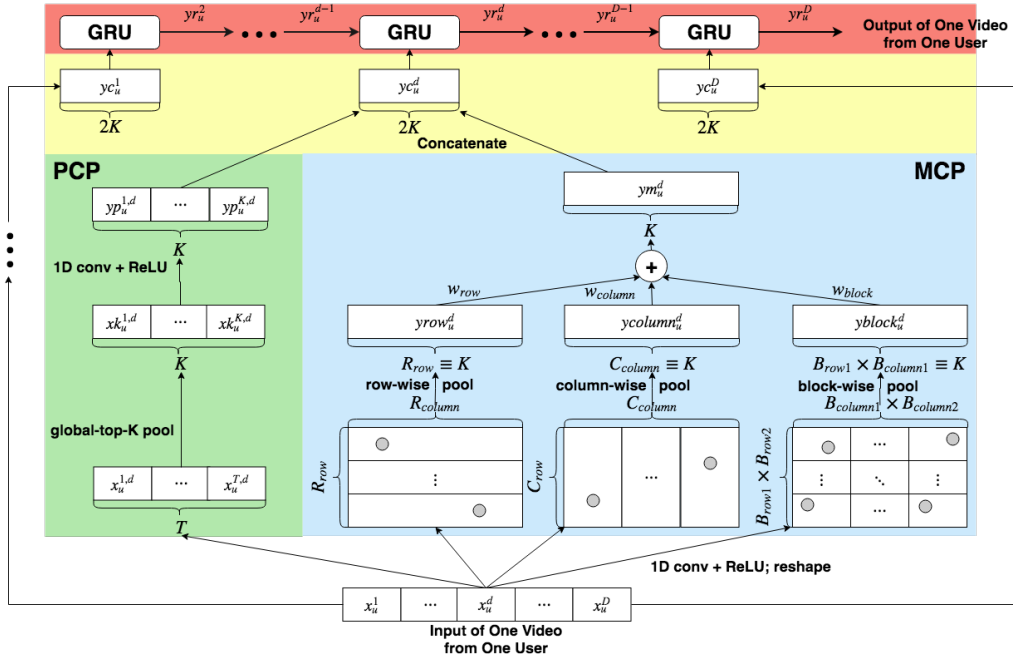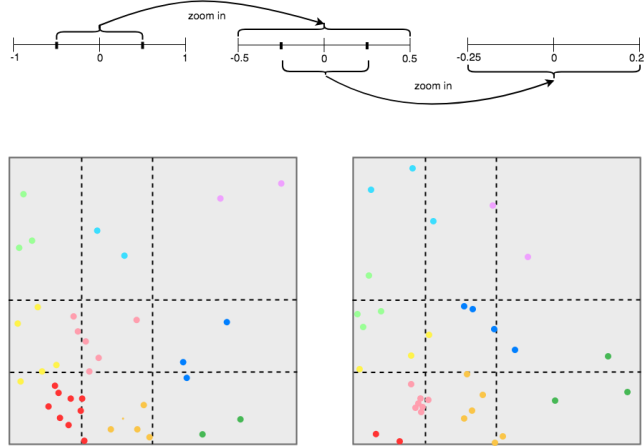5) Let $B^2 = Intv^1 \times Intv^1$.

Fig. 3. Temporal layers

Fig. 4. Example of Hierarchical Clustering Blocks

Comparison with K-means results in Figure 4 shows that our supervised-like clustering method maintains the consistency of cluster indices in MBGD training and distributed environment, so that the video cluster can be determined without knowledge of the other ones.

The loss module adds the loss from the enc-dec module and the loss from the clustering module weighted by a hyperparameter ω.

$$loss_c = \frac{1}{2} \sum_{v \in V} \left\| NT_{(v)} - D_{(v)} \right\|_2^2 + \omega \min_{b \in B^2} \left\| E_{(v)} - O_b \right\|_2^2$$

O_b denotes the coordinate of the center of b.

We apply unsupervised learning to explore the user dimension because there are few labeled tags or accurate prior knowledge. Besides, we separate the temporal layers from the autoencoder module for two reasons: (1) fully-connected structure is not an optimal choice for time series analysis; (2) the decoders with RNN and CNN suffer from lack of a finite-sized dictionary.

D. Accumulation Layers

The accumulation layers group the primitive inputs by their clusters.

$$XA^c = \sum_{v \in V \bigcap \{C_{(v)}=c\}} X_{(v)}$$

Then follows normalization, and gets $NXA_{(c)}$ After the temporal layers, the outputs are $T_{(c)}$.

E. Policy Layers

The policy layers estimate the probabilities for each video cluster to be dispatched to all users on day $(D+1)$. There are two components, full connection module and loss module. The full connection module transforms $T_{(c)}$ to $UP_{(c)}$ on $(0.0, 0.1)^{|U|}$ with learnable parameters.

The loss module compares $UP_{(c)}$ with the real request data on day *D+1*.

$$R_{(c)} = \left(\frac{XA_1^{D+1}}{\sum\limits_{u \in U} XA_u^{D+1}} \quad \cdots \quad \frac{XA_{|U|}^{D+1}}{\sum\limits_{u \in U} XA_u^{D+1}} \right)_{(c)}$$

$$loss_p = \frac{1}{2} \sum_{c \in \Re(C)} \left\| UP_{(c)} - R_{(c)} \right\|_2^2$$

IV. IMPLEMENTATION AND RESULTS

A. Data Set

We prepare the data set with the real-world video requests from each user accumulated every five minutes. Historical requests are for training, and 15 consecutive days of requests until present are for predicting. There are 177 users, and the dimension of the feature space of one video has 12 minutes × 24 hours × 15 days × 177 users = 764,640.

We analyze the video requests from two aspects. Firstly, we consider each video request sequence from one user as an independent stochastic process and observe the nonstationarities. We sample 5,000 sequences, calculate the difference of the means within a one-day sliding window. After taking the first-order difference of the sequences, the difference of means decreases, as in Table 1. That means nonstationarities are reduced. Next we run the Kwiatkowski-Phillips-Schmidt-Shin (**KPSS**) test to the sampled sequences with one-day lag (288). They also show the nonstationarities.

Difference of Means		
Original		0.00616
After first-order differences		0.00158
KPSS Test		
KPSS statistic		-5.38
P-value		0.000003
Critical values	1%	-3.43
	5%	2.86
	10%	2.57

TABLE I
NONSTATIONARITY ANALYSIS FOR VIDEO REQUESTS

Secondly, we analyze the peak request frequencies. The total request number is 82,700,000 for two hours, with 8,930,000 videos, so the average request frequency per video is 9. We sort the videos in descending order of request frequencies, and show the request distribution after excluding current videoss. Figure 5 shows a typical long-tail shape.

B. Environment

We run on multi-machines with 48-core CPU (Intel Xeon 2.20GHz) and 128G memory, based on TensorFlow 1.3 and coordinated with Apache Zookeeper 3.4.
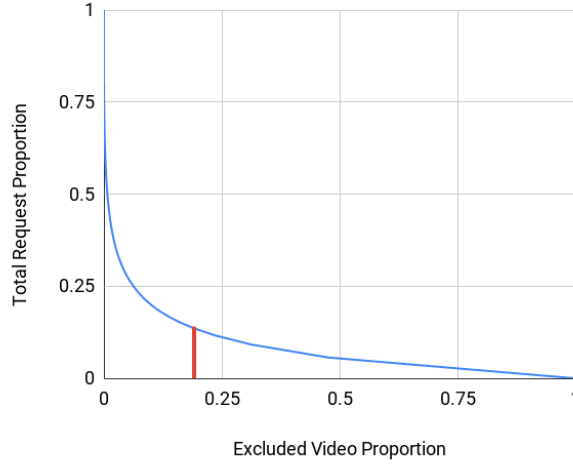
7

Fig. 5. Peak-time Video Request Distribution

C. Trainers and Predictors

Training the clustering network requires a clustering trainer, and training the policy network requires a clustering predictor and a policy trainer. Predicting involves clustering and policy predictors.

The trainers comprises all of the modules in the respective networks. The clustering predictor is composed of the all layers but the decoder and loss modules in the clustering network, and the policy predictor is composed of all layers but the loss module in the policy network.

D. Asynchronous Training Mechanism

Three models are generated through training the temporal, clustering, and policy layers. The clustering model is updated in the clustering network. The temporal and policy models are updated in the policy network.

We train the two networks asynchronously as in Figure 6. We start the policy trainer first with randomized clustering indices. Then start the clustering predictor and trainer with the identical temporal model trained for several iterations in the policy trainer. The clustering trainer fetches the temporal model from the policy trainer every 200 iterations. The clustering predictor fetches the clustering model from the clustering trainer at each iteration.

The video dispatch scenario differs from the regular L2R in that the items are assumed as independent in the latter case, while the popular videos often enjoy continual requests. Therefore, the dataset for the policy network needs shuffling to break the chronological order so as to remove autocorrelation between the neighboring samples. This procedure is inspired by experience replay in reinforcement learning.

All of the learnable parameters are initialized with Xavier initialization . The MBGD optimizer is utilized, and the learning rates are layer-wisely tuned with decay settings.
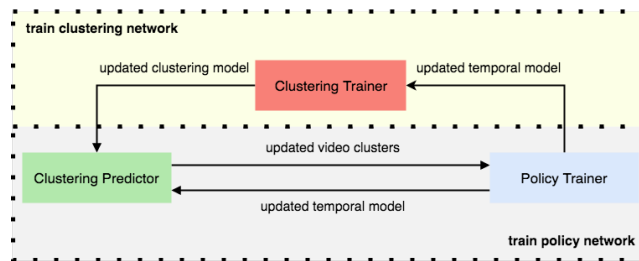

Fig. 6. Asynchronous Training Mechanism

E. Prediction Mechanism with Post-processing

1) Run the clustering predictor for $C(V)$.
2) Run the policy predictor for $UP_{|\Re(C)|,|U|}$.
3) Calculate $CP_{C,I} = UP_{|\Re(C)|,|U|} UC_{|U|,|I|}$.
4) For $\forall i \in I$, sort V in the descending order of $CP_{C(V),i}$. The videos in the same cluster are sorted in the descending order of uploaded time. Dispatch the first 10,000 videos to i.

F. Training Performances

Firstly, Figure 7 shows how the normalization module helps the clustering trainer to converge, where the normalization module is removed for A and B. Loss curve A represents training with a data set of 20 batches, and unchanged temporal models. The loss jumps beyond the initial value immediately after 3 iterations, and then descends slowly. Loss curve B represents training with a data set of only one batch, and periodically (every 500 iteration) changed temporal model. The loss shoots up every time the temporal model is updated to newly randomized values. Loss curve C represents training with a data set of 20 batches, and changed temporal model as in B. It descends 30% at the first 500 iterations, and continues till the 10,000 iterations.
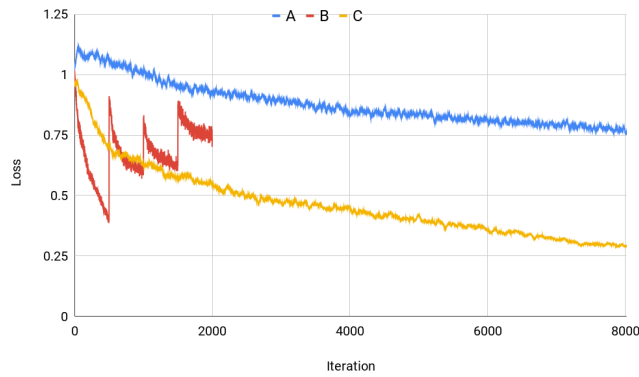

Fig. 7. Clustering Trainer Loss Percentages

Secondly, Figure 8 shows how shuffled data set helps the policy trainer to converge. The data set in A is ordered by ascending date, and divided into two groups in the middle. The policy trainer fetches the two groups of data alternately, training each group for 200 iterations. The data set in B is finely shuffled. Loss A decreases more rapidly than loss B in the beginning, but shoots up after moving to the other half of data, causing it difficult to converge.
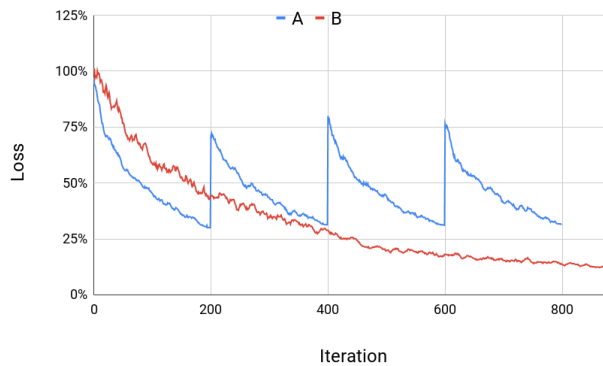

Fig. 8. Policy Trainer Loss Percentages

Thirdly, Figure 9 shows the training results based on the section of "asynchronous training mechanism". They converge without being interfered by each other.
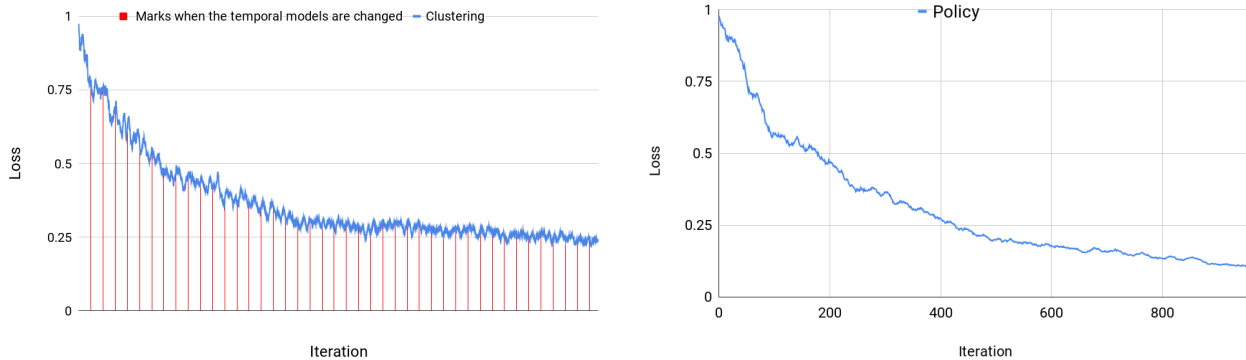
Fig. 9. Asynchronous Training Losses of Both Trainers

G. Assessment for Video Clustering

Firstly, Table 2 compares the inner productions between 10,000 video requests in the same clusters (excluding the inner productions of the video requests to themselves) with those in different clusters. The mean from the same clusters is much larger than that from the different, and the coefficient of variation (**CV**) is smaller. This implies that the video requests from the same cluster are much more similar than from the different.

	Same	Different
Mean	14.0	0.117
CV	1.41	4.59

TABLE II

INNER PRODUCTIVITIES FROM SAME/DIFFERENT CLUSTER

Secondly, Figure 10 compares the video request densities in different clusters, including non-sparsities (**NS**), L1-norms and L2-norms (whose means are multiplied by 20). NS is defined as the ratio of the number of non-zeros to the length. The means of the three are positively related, and the CVs are close. This implies that the videos with similar request densities are more likely to be in the same cluster.
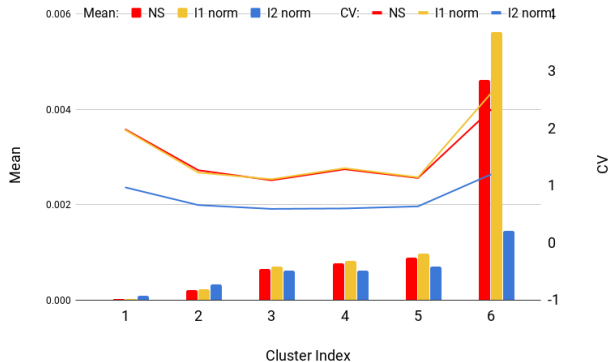

Fig. 10. Video Request Densities

Thirdly, we count the number of videos (**NV**) in each cluster, and calculate the average distances (**AD**) from the encoder outputs to the cluster centroids.Table 3 applies Pearson correlation coefficient to measure the correlation of NV with AD and the area of the cluster. This implies that a larger cluster area for larger values of the encoder outputs is space efficient with sufficient accuracies.

H. Prediction and Controlled Experiment Results

We dispatch respectively 10,000 videos out of 50 million to 50 CDNs at daily off-peak time for the next peak-time requests. Our video dispatch prediction takes 4 hours and 100 machines to compute. We denote our proposed algorithm as A.

10

NV and area	0.6194
NV and AD	-0.4956

TABLE III
CORRELATION COEFFICIENT

We previously dispatched videos based on the threshold of their request frequencies during a period of time, denoted as B. Since B dispatches videos at peak time, extra costs are not ignorable. The prediction accuracy is represented as the ratio of the peak-time request number (revenues) to the peak-time dispatch number (costs).

For comparison, six other methods are listed. Firstly, we compute dispatch policies via classification **without video clustering**. This restricts us to build one request model for one user in order to lower the input dimensions. The outputs are either "to dispatch" or "not to dispatch", while the ground truth is the average requests over the peak time on the next day. Method C applies logistic regression and Method D applies Gradient Boosting Decision Tree.

Secondly, we **remove the end-to-end learning** mechanism, and let the clustering and policy tasks run independently. This restricts us from using temporal layers in the clustering network. Method E substitutes the clustering network with principal component analysis to reduce the input dimension to 4, followed by K-means clustering. Method F removes the temporal layers from the clustering network.

Thirdly, method G substitutes our supervised-like blocks with **K-means clustering**.

Fourthly, method H **removes the CNN module** from the temporal layers.

The input sequences in C, D, E, F and H are sampled per hour. The prediction data set in E and G is limited to the latest 10,000 videos, and the dispatch number is 100. The prediction results are shown in Figure 11, which summarize A and B for 90 days, and C to H for 30 days. The results include the means and standard deviations of the prediction accuracies, but exclude recalls because the CDN storage capacities are much smaller than the total size of the requested videos.

The overall performances of the end-to-end learning methods, namely A, G ,H, D and C are better than the others. Although K-means in G and E provides more accurate clustering results, it suffers from scaling issues and limits the dispatch candidates. The CNN module in method A filters the sequences, meanwhile retaining the important information, which especially have advantages over the fully connected structure as in F, and also in D and C. Besides, the CNN module with learned parameters performs better than trivial downsampling method in H. The RNN module in A, G and H provides better sequence forecasting accuracies mainly because RNN is able to explore complex nonlinear state transitions. Lastly, our previous method B suffers from the dilemma of request frequencies and request amounts as in Figure 5. Although a higher threshold in B implies higher dispatch accuracies, the number of dispatchable videos decrease.
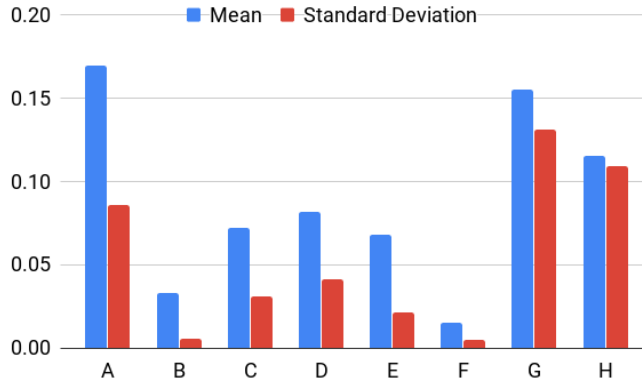

Fig. 11. Prediction Results

V. CONCLUSION

In this paper, we propose a video dispatch algorithm for VOD to enhance peak-time service quality. We cluster the videos with one network by their request patterns before dispatching them from the large candidate set. We develop dispatch policies for the clustered videos with the other network inspired by pointwise L2R algorithms. The two networks are coupled with shared

temporal-feature-extracting layers, which comprise CNN and RNN modules. After training them asynchronously, the average prediction accuracy on real-world video requests is 5 times as high as that of our previous methods.

Future work will be devoted to risk controls to guarantee robustness regardless of rapidly changing situations.