Cooperative Caching in Peer-to-Peer Video-on-Demand

Philip Y. Yang and S.H. Gary Chan

cs_yyx@ust.hk, gchan@cse.ust.hk

Contribution: Formulate *and* Solve the Caching Problem

In Peer-to-Peer Video-on-Demand (P2PVoD) network, each peer would contribute its disk storage to store some video segment in order to serve the need of others. We formalized the segment caching problem in P2PVoD network and proposed an effective distributed cooperative caching algorithm.

Delay Function

The *expected delay* for a peer i is defined as

$$E[delay(i)] = epd(i) + eqd(i)$$

It consists of two parts, the *expected propagation delay* and *expected queuing delay*. The *expected propagation delay* is the computed using the

$$epd(i) = \sum_{j \in \mathcal{V}} \sum_{s \in \mathcal{S}} p_s y_{jis} \delta_{ij}$$

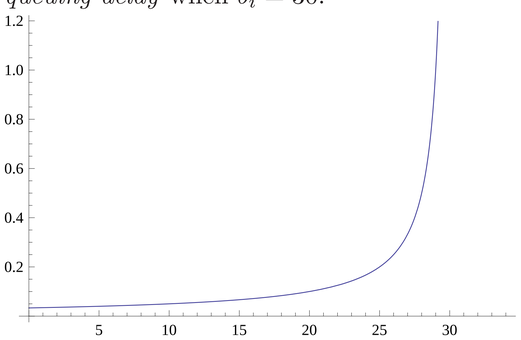
Let $eqd(i)$ be the expected queueing delay estimation and let

$$\tau_i = \sum_{j \neq i \in \mathcal{V}} \sum_{s \in \mathcal{S}} p_s y_{jis} d_s,$$

where d_s is the streaming bitrate of segment s. Then we have,

$$eqd(i) = \begin{cases} 1/(b_i - \tau_i) & \tau_i \leq \alpha, \\ eqd(\alpha) + \beta(\tau_i - \alpha) & \text{otherwise}, \end{cases}$$

where b_i denotes the upload bandwidth of peer i. Here the choice of α and β could be based on some empirical analysis. We take $\alpha = 0.99 b_i$ and $\beta = 10^4 / b_i^2$. Below is an illustration of the *expected queuing delay* when $b_i = 30$.



References

[1] Garey, Michael R. and Johnson, David S. Computers and Intractability: A Guide to the Theory of NP-Completeness 1979, ISBN 0716710455, W. H. Freeman & Co. New York.

Formulation

The Cooperative Caching problem is formulated as a 0-1 integer programming problem over the two set of optimization variables X and Y. We let x_{is} be whether peer i cached segment j and y_{ijs} be whether peer i chooses peer j as its streaming parent for segment s. The objective is to minimize the expected *expected delay*

$$\sum_i E[delay(i)]$$

subject to the constraints that

- each peer has to designate a streaming parents for each video segment
- each designated streaming parent must has already cached the segment
- each peer can cache up to its storage capacity
- each optimization variables are either 0 or 1

The problem is NP-hard.

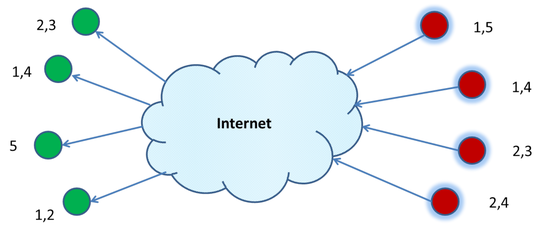
Distributed Algorithm

for all segment s **do**
  $\vec{u}_s \leftarrow \frac{p_s x_{is}}{\vec{p} \cdot \vec{x_i}} b_i$
  $\vec{v}_s \leftarrow \frac{p_{s'}(1 - x_{is})}{\vec{p} \cdot \vec{x_i} + p_{s'}}$
end for
{Estimate cache update probability globally}
$(\hat{u}, \hat{v}) \leftarrow glFindAverage(pid, \vec{u}, \vec{v})$
for all segment s **do**
  $\vec{qo}_s \leftarrow \frac{p_s d_s - \hat{u}_s}{\hat{v}_s}$
  $\vec{qu}_s \leftarrow 1 - \frac{p_s d_s}{\hat{u}_s}$
end for
if cache is not full **then**
  cache segments with $\vec{qu}$ in non-increasing order.
else
  $updateCache(pid, \vec{qu}, \vec{qo})$
end if

Distributed Cooperative Caching

Our distributed takes a few heuristics to solve the segment caching problem. The first one is that in order to make segments available, it is favorable to balance the global segment supply versus demand. The second one is that each peer make cache updating algorithm by catering the need locally, with respect to a subset of peers called *Support Set* in which each member might be affected once a cache replacement decision is made by the peer.

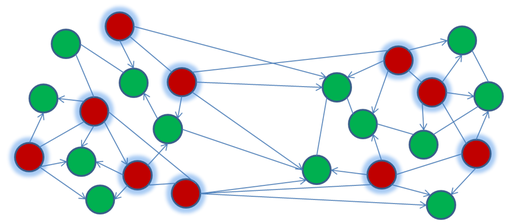
Global Heuristics



Streaming Parent: red ones denote parents

Minimize demand-supply discrepancy We first estimate the current *supply* of each segment over the network. For each *under-supplied* segment, we could compute a probability such that if all the peers not having cached it will cache it with such probability, the *supply* and *demand* of that segment is balanced. Similar probability could be computed for *over-supplied* segments as well.

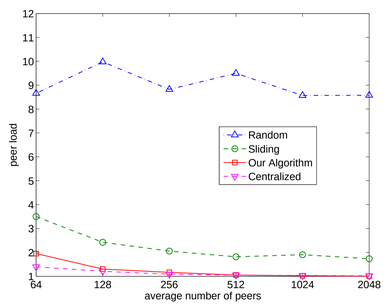
Local Heuristics

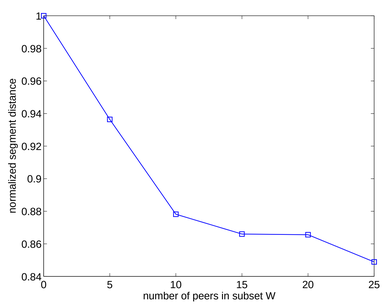


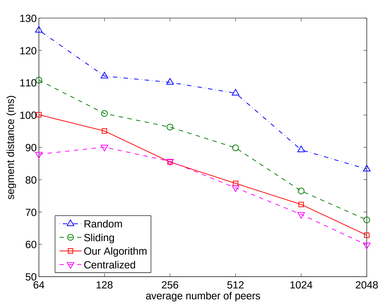
Locality: red ones denote streaming parents

Make local decision With the estimated update probability given by the *global heuristics*, we could compute the *expected change of delay* with respect to the *Support Set* for the peer after caching or replacing a segment. The *score* of a replacement pair is the difference in the *expected change of delay*. We proceed further in a greedy manner until the score is less than some threshold.

Simulation Result







Our distributed algorithm performs well in the simulation.
We compared our algorithm with random caching, sliding window scheme and centralized algorithm. It appears that our distributed algorithm outperforms the random caching in a large margin. Compared with the sliding window algorithm which is used by many, our algorithm showed significant performance gain. In many cases, it also approaches the performance for the centralized algorithm. The simulation shows that as the number of peers in the network grows larger, the end-to-end distance becomes smaller. As for the support set, the larger its size, the better the performance. The experiment also shows that our algorithm does lower the supply-demand discrepancy, which serves as a good indicator that this heuristic might be a good one.