OPTIMIZING SEGMENT CACHING FOR PEER-TO-PEER ON-DEMAND STREAMING

Ho-Shing Tang *S.-H. Gary Chan* *Haochao Li*

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{wilsont, gchan, cs_lhcaa}@cse.ust.hk

ABSTRACT

In peer-to-peer (P2P) on-demand streaming applications, multimedia content is divided into segments and peers can seek any segments for viewing at anytime. Since different segments may be of different popularity, random segment caching would lead to a segment popularity-supply mismatch, and hence an uneven workload distribution among peers. Some popular segments may be far from peers, leading to inefficient search and streaming. In this paper, we study optimal segment caching for P2P on-demand streaming.

We first state the segment caching optimization problem, and propose a centralized heuristic to solve it, which serves as a benchmark for other algorithms. We then propose a distributed caching algorithm termed POPCA (POPularity-based Caching Algorithm), in which each peer adaptively and independently replaces segments to minimize the popularity-supply discrepancy and the segment distance from peers. Through simulations, we show that POPCA achieves near-optimal performance, and lower peer workload and segment distance as compared with other schemes.

1. INTRODUCTION

In a P2P on-demand streaming network, there is a server (or a cluster of servers) with all the segments of user interest, and a pool of peers who may randomly seek any segments at anytime.[1] Each peer locally caches a number of segments depending on their disk or memory caching capacity. If cached properly, segments can be found directly in other peers most of the time; the server is contacted only when the search is unsuccessful. This is how the server load can be significantly reduced.

The segment caching optimization problem is that given limited local storage and heterogeneous segment popularity, we need to decide which segments to cache in each peer to ensure that the popularity and the supply of segments are matched, and the peer can reach close segment suppliers. Since we focus on segment caching, other issues such as buffering, scheduling, and stream optimization through, for example, network coding are beyond the scope of this work. We consider segment caching for a single streaming session (i.e. a movie or an interactive window of a time-shift TV channel), though, multi-sessions can easily be supported by replicating multiple instances of our algorithm.

Many P2P on-demand streaming systems adopt the sliding window caching strategy, in which each peer caches segments within a sliding window of its play-point [1]. The advantage of this strategy is that peers watching the same portion of content can share. The supply implicitly matches the popularity (or demand) of segments. However, as a peer caches segments according to its play-point, the original cached segments will no longer be available if the peer seeks out some other positions of the media content. Performing a seek operation will lead to severe disruption of the multimedia stream to all of the streaming children. Therefore, sliding window caching does not work well under frequent seeks.

To address the above, peers can store segments statically (independent of the play-points) in their local storage depending on their caching capacity. A straight-forward approach in static caching is to randomly cache segments in storage [2]. However, as mentioned before, random caching can lead to uneven workload among peers due to heterogeneous segment popularity. Our caching algorithm, POPCA, is a popularity-based static caching approach, which is robust to frequent seeks and achieves more uniform workload among peers.

Some static caching schemes are based on the segment popularity result in lower demand-supply discrepancy of segments [3]. However, without minimizing the segment distance, the above caching schemes may not lead to efficient P2P search and streaming.

This paper is organized as follows. In Section 2 we present a centralized algorithm. We discuss the distributed segment caching altorithm POPCA in Section 3. In Section 4 we evaluate the performance of the proposed algorithm with illustrative simulation results. We conclude in Section 5.

This work was supported, in part, by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611107), and the Hong Kong Innovation Technology Fund (ITS/013/08).

[1] In this paper, we use "user" and "peer" interchangeably. Moreover, we use "seek" and "search" interchangeably.

2. A CENTRALIZED CACHING SOLUTION

In this section, we present a centralized solution based on global knowledge. The goal of segment caching optimization is to determine, for each peer, which segments to cache such that the segment distance is minimized under the constraints of demand-supply match.

Let $\mathcal{V}$, $\mathcal{U}$ and $\mathcal{S}$ be the set of peers, the set of servers and the set of segments, respectively, where $\mathcal{U} \subseteq \mathcal{V}$, $|\mathcal{V}| = N$ and $|\mathcal{S}| = S$. The servers cached all the segments. Denote d_{ij} the distance between peers i and j, where $i, j \in \mathcal{V}$. Let p_s be the popularity or demand of segment s (the probability of a peer accessing segment s), where $\sum_{s\in\mathcal{S}} p_s = 1$. Further let q_s be the supply of segment s (the fraction of peers caching segment s in the network), where $\sum_{s\in\mathcal{S}} q_s = 1$. The 0-1 variable y_{ijs} indicates if peer i has some segment supplier j caching segment s, where $i \in \mathcal{V}$, $j \in \mathcal{V}$, $s \in \mathcal{S}$.

The goal is to minimize the average segment distance, defined by $\min \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{V}} p_s d_{ij} y_{ijs}$, such that the demand-supply discrepancy of segment s match within the scope of access-balancing, defined by $\left|\frac{p_s - q_s}{q_s}\right| < \epsilon$.

To illustrate, let us start with a simple case where a single segment needs to be cached in k peers in order to minimize the segment distance. This problem is the same as the k-median problem, which selects k facilities from the set of facilities $\mathcal{F}$ to open such that the sum of distances between nodes and their closest opened facilities is minimized. The k-median problem is well-studied and can be solved by heuristics [4].

Now we generalize the algorithm for all segments, by running the k-median heuristic S times, where S is the number of segments. Let r_i be the residual caching capacity of peer i, C be the sum of caching capacity of all peers, and $\mathcal{F}$ be the set of peers whose residual caching capacity is greater than zero. The centralized caching algorithm (as presented in Algorithm 1) iterates through each segment following a descending order of segment popularity. By starting with more popular segments (i.e. more weighted segments in the optimization statement), the algorithm is more likely to yield a better optimization value. In each iteration, the k-median heuristic is run to determine which k peers cache the segment, where the k peers are selected from the set of peers with positive residual capacity, and k is proportional to the segment popularity and the number of segment suppliers so that more popular segments are cached by a larger number of peers.

As the centralized caching algorithm requires global knowledge and needs to be run for every change of the segment popularity, it is not scalable to a large group and dynamic segment popularity. Hence, we will use it as a performance benchmark for our distributed algorithm POPCA.

Algorithm 1 CENTRALIZEDCACHING

```
1: S ← sortSegmentByPopularity(S, p)
2: for all s ∈ S do
3:     F ← ∀i ∩ (r_i > 0)
4:     k ← min(p_s hC, N)
5:     /* Input: V, F, k, d; Output: x, r */
       kMedianHeuristic(V, F, k, d, x, r)
6: end for
```

3. POPCA: DISTRIBUTED POPULARITY-BASED SEGMENT CACHING

In this section, we present the details of our distributed popularity-based segment caching algorithm, POPCA, which is run adaptively and independently by peers. Each peer runs the segment replacement algorithm adpatively and independently, so as to (i) match the segment supply to the popularity, and (ii) minimize the segment distance, elaborated as follows:

Matching supply and popularity: Let $\mathcal{S}_{surplus}$ be the set of over-supplied segments, i.e. $s \in \mathcal{S}_{surplus}$ iff $\frac{\hat{q_s} - \hat{p_s}}{\hat{q_s}} > \epsilon$. Similarly, let $\mathcal{S}_{deficit}$ be the set of under-supplied segments, i.e. $s \in \mathcal{S}_{deficit}$ iff $\frac{\hat{p_s} - \hat{q_s}}{\hat{q_s}} > \epsilon$. The popularity and the supply of each segment can be estimated by means of distributed averaging [5]. For each peer, segments can either be cached or not cached, as indicated by the sets $\mathcal{S}_{cached}$ and $\mathcal{S}_{not-cached}$. In order to match the supply with the popularity, each peer replaces segment s with a replacement probability P_s^R and cache segment s' with a cache probability $P_{s'}^C$.

The rationale of the replacement probability P_s^R and the cache probability $P_{s'}^C$ is as follows. For an over-supplied segment s, each of the $N\hat{q_s}$ peers caching segment s independently replaces the segment with a probability P_s^R, so that $N(\hat{q_s} - \hat{p_s})$ copies are replaced and eventually only $N\hat{p_s}$ copies are left in the network. To match the demand and supply, the following equation needs to be satisfied: $N\hat{q_s} \cdot P_s^R = N(\hat{q_s} - \hat{p_s})$. Thus, the replacement probability is $P_s^R = \frac{\hat{q_s} - \hat{p_s}}{\hat{q_s}}, \forall s \in \mathcal{S}_{surplus} \cap \mathcal{S}_{cached}$.

A peer with residual capacity will cache an under-supplied segment s'. If the popularity-supply discrepancy of a segment is large, the segment will be cached with a higher probability. The cache probability should be proportional to the popularity-supply discrepancy, i.e. we take $P_{s'}^C = \frac{\hat{p_{s'}} - \hat{q_{s'}}}{\sum_{i\in\mathcal{S}_{deficit}\cap\mathcal{S}_{not-cached}} (\hat{p_i} - \hat{q_i})}, \forall s' \in \mathcal{S}_{deficit} \cap \mathcal{S}_{not-cached}$.

Minimizing segment distance: Let $\mathcal{S}_{match}$ be the set of popularity-supply matched segments, i.e. $s \in \mathcal{S}_{match}$ iff $\left|\frac{\hat{p_s} - \hat{q_s}}{\hat{q_s}}\right| < \epsilon$, $\mathcal{W}$ be a subset of all peers where $\mathcal{W} \subseteq \mathcal{V}$, $D_{ws}^{(1)}$ be the distance between peer w and its closest segment supplier with segment s, and $D_{ws}^{(2)}$ be the distance between peer w and its second closest segment supplier with segment s, where $w \in \mathcal{W}$.

We explain how to compute the cost of replacing segment

(a) replace segment s at peer i (b) cache segment s' at peer i

Fig. 1. An illustration of computing cost of replacing segment s at peer i, and benefit of caching segment s' at peer i.

s, and the benefit of caching segment s' at peer i. Let x be the original second closest segment supplier of w for segment s, and y be the original closest segment supplier of w for segment s. To compute the cost of replacing segment s (Figure 1a), where $s \in \mathcal{S}_{match} \cap \mathcal{S}_{cached}$, a subset of peers $\mathcal{W}$ is examined. If peer i is the closest segment supplier of peer w for segment s, replacing s at the peer i will result in peer w switching to x. The segment distance for this is increased by $D_{ws}^{(2)} - d_{iw}$. Thus, the cost of replacing segment s is $cost(s) = \hat{p_s} \cdot \sum_{w \in \mathcal{W}} \max\left(D_{ws}^{(2)} - d_{iw}, 0\right)$.

Similarly, to compute the benefit of caching segment s' (Figure 1b), $s' \in \mathcal{S}_{match} \cap \mathcal{S}_{not-cached}$, a subset of peers $\mathcal{W}$ is examined. If the distance between i and w is less than the distance between y and w, caching s' at peer i will result in peer w switching to peer i for segment s'. The segment distance for this is decreased by $D_{ws'}^{(1)} - d_{iw}$. Thus, the benefit of caching segment s' is $benefit(s') = \hat{p_{s'}} \cdot \sum_{w \in \mathcal{W}} \max\left(D_{ws'}^{(1)} - d_{iw}, 0\right)$.

The score of replacing segment s by s' is given as $score_{ss'} = benefit(s') - cost(s)$. If the score of a segment replacement is greater than zero, it is beneficial to do that as the segment distance is likely to reduce. In the score computation, a peer only needs to examine a subset of peers due to scalability.

4. PERFORMANCE EVALUATION

In this section, we evaluate POPCA through simulations. In our simualtion, BRITE is used to generate an Internet-like topology (with more than $3,000$ nodes and $10,000$ links). Users arrive according to a Poisson process with rate λ peers/seconds. Their lifetimes are exponentially distributed with mean τ seconds, independent of each other. To simulate peer churn, peers rejoin the network after leaving the network. We consider that segment popularity follows a Zipf distribution.

We define the following performance metrics to evaluate segment caching schemes:

(1) *Segment distance:* It is the average distance between a peer and the closest segment supplier, defined by $\sum_i \sum_s \hat{p_s} D_{is}^{(1)} / N$.

(2) *Search latency:* It represents the average time elapsed between executing the search and returning a set of peers by the search. We assume that the requestor will contact the server if the search is not successful.

(3) *Hit rate:* It is the probability that a search is successful. Since our system is dynamic with joins and leaves, segments may not always be found.

We compare POPCA with traditional schemes given by sliding window caching and random caching using static storage, and centralized popularity-based caching as presented in Section 2. They are denoted as Random, Sliding, and Centralized, respectively. The centralized popularity-based caching is served as an optimum for performance comparison.

Besides the client-server approach, we compare POPCA with two other reactive-based segment search approaches, DHT-PNS and biased random walk, explained as follows:

(1) *DHT-PNS:* DHT-PNS is a locality-aware Chord-based DHT implementation (by using Proximity Neighbor Selection). It applies more flexiable routing table construction in which any close nodes whose identifier is in $[s+2^{i-1}, s+2^i)$ can be filled in the entry of i-th finger. In our simulation, a $(\log S)$-bit key is used in the DHT network (instead of a fixed-length key, e.g. 160-bit key) to give a more optimistic result for its search latency.

(2) *Biased random walk:* It is an unstructured P2P search. In its overlay construction, high-capacity peers (in our simulations, capacity refers to caching capacity) connects to more neighbors. In search operation, biased random walk is performed towards high-capacity peers. The maximum number of neighbors is set to 30 and search scope is set to 10.

Unless otherwise stated, the following default settings are used: the number of segments (S) is 128, the parameter for caching capacity (p) is 0.4, the parameter for segment popularity (θ) is 1, ϵ is 0.1, $|\mathcal{W}|$ is 10, the maximum number of entries in a segment table row is 5, the segment table advertising period is 60 seconds, the number of overlay neighbors is 4, τ is $7,200$, and the average number of users (N or $\lambda\tau$) is $1,024$ (i.e. λ is 0.1422). Each peer runs the segment replacement algorithm every 5 minutes.

In Figure 2, we show the segment distance versus $\lambda\tau$ given different segment caching strategies. When the number of peers increases, the segment distance decreases. This is because there is a higher number of copies for each segment, so it is more likely that segments are cached in close peers. POPCA achieves substantially lower segment distance than random and sliding window caching, and is close to centralized segment caching. This shows that POPCA provides efficient search and streaming for on-demand streaming service. Due to heavier weights on popular segments and minimization of segment distance, POPCA achieves close to optimal performance.

We compare in Figure 3 the search latency of POPCA and other schemes versus $\lambda\tau$. As the number of peers increases, the search latency remains flat. This shows that the search latency is independent of the number of peers. POPCA has near-instant search latency because searches are only

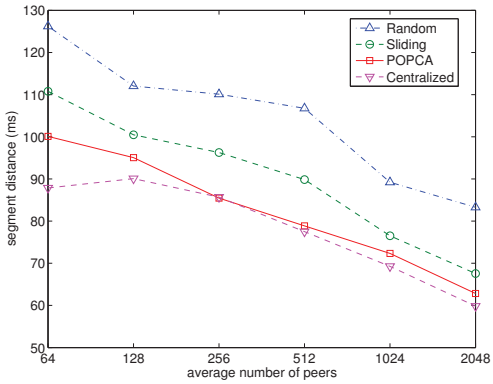

Fig. 2. Segment distance against number of peers using different caching algorithms.

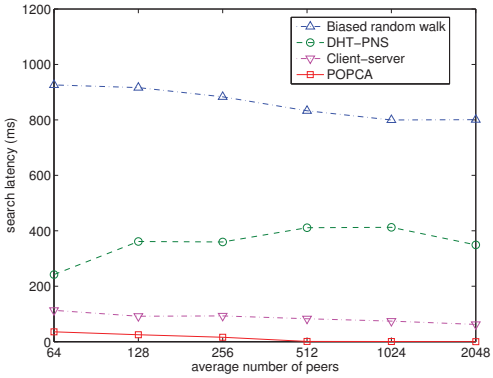

Fig. 3. Search latency against number of peers using different search algorithms.

achieved by a local segment table lookup. In client-server, peers always contact the server to locate segments, accounting for a single round-trip search latency. The search latencies of DHT-PNS and biased random walk are much higher as the searches have to visit multiple intermediate peers.

Figure 4 shows the hit rate achieved by different schemes versus $\lambda\tau$. When the number of segments increases, the number of copies for each segment decreases. Thus, the hit rate drops. DHT-PNS has the highest hit rate since DHT is more efficient in locating rare objects, while biased random walk experiences substantially lower hit rate. POPCA achieves a high hit rate, comparable to DHT-PNS. With high hit rate, POPCA can significantly reduce the workload of the server.

5. CONCLUSION

In providing P2P on-demand streaming, various segments of different popularity are continuously cached and accessed in peers. A critical problem to address is which segments to cache in order to achieve low segment distance. We study the segment caching optimization problem and propose the distributed algorithm, POPCA, to solve the problem. Through simulations, we show that POPCA has near-optimal performance, and achieves, as compared with other caching schemes, substantially lower segment distance and search latency, while maintaining a high hit rate.

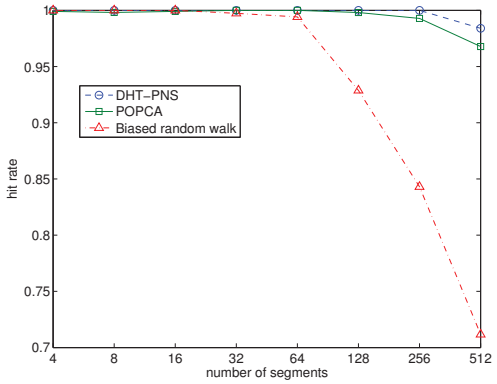

Fig. 4. Hit rate against number of segments using different search algorithms.

6. REFERENCES

[1] T. T. Do, K. A. Hua, and M. A. Tantaoui, "P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment," in *Proceedings of IEEE International Conference on Communications (ICC)*, Paris, France, June 2004, pp. 1467–1472.

[2] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, The Sagamore, Bolton Landing (Lake George), New York, Oct. 2003, pp. 282–297.

[3] Wai-Pun Ken Yiu, Xing Jin, and Shueng-Han Gary Chan, "VMesh: Distributed segment storage for peer-to-peer interactive video streaming," *IEEE Journal on Selected Areas in Communications (JSAC) special issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, pp. 1717–31, Dec. 2007.

[4] Vijay Arya, Naveen Garg, Rohit Khandekar, Kamesh Munagala, and Vinayaka Pandit, "Local search heuristic for k-median and facility location problems," in *Proceedings of ACM Symposium on Theory of Computing*, 2001, pp. 21–29.

[5] Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, and Richard M. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 512–520, June 2007.