Multiple-Cache Pairing for Fine-Grained Scalable Video Caching and Networking

Qiushi Gong
Southeast University
Nanjing, Jiangsu, China
qsgong@njnet.edu.cn

John W. Woods
Rensselaer Polytechnic Institute
Troy, New York
johnwoods@ieee.org

Koushik Kar
Rensselaer Polytechnic Institute
Troy, New York
koushik@ecse.rpi.edu

Jacob Chakareski
The University of Alabama
Tuscaloosa, Alabama
jacob@ua.edu

ABSTRACT

In this paper, we consider fine-grained scalable video caching to provide video-on-demand (VoD) service, where exclusive-or (XOR) network coding is adopted to reduce data traffic. To further reduce backhaul data traffic, caches are grouped into pairs, which can be modeled as a maximum-weighted matching (MWM) problem. A heuristic harmony search algorithm is employed to solve the problem. Numerical results show that cache cooperating in pairs can help limiting backhaul data traffic. Further, the algorithm is shown to achieve greater backhaul traffic savings than other approximate MWM solutions.

CCS CONCEPTS

• **Information systems** → **Multimedia information systems**; • **Networks** → *Network services*; *Network performance evaluation*.

KEYWORDS

fine-grained SVC, network coding, video caching, harmony search

ACM Reference Format:
Qiushi Gong, John W. Woods, Koushik Kar, and Jacob Chakareski. 2019. Multiple-Cache Pairing for Fine-Grained Scalable Video Caching and Networking. In *The 14th International Conference on Future Internet Technologies (CFI'19), August 7–9, 2019, Phuket, Thailand*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341188.3341196

CFI'19, August 7–9, 2019, Phuket, Thailand

ACM ISBN 978-1-4503-7238-1/19/08…$15.00
https://doi.org/10.1145/3341188.3341196

1 INTRODUCTION

Video streaming and VoD services are becoming one of the most demanding applications on the Internet. As predicted by Cisco, IP video could represent 82% of all user traffic by 2021 [9]. Effective delivery of video is however quite challenging due to heterogeneity among users, who may have distinct preferences and constraints on resolution, frame-rate, and bit-depth. The network environment also varies among the users (e.g. wired vs. mobile networks) resulting in different available bandwidths [19].

To reduce source overload and improve video delivery performance, VoD service offers Content Delivery Networks (CDN) [10] that provide caching for geographically dispersed users [14]. Clients then acquire available video content from local caches instead of directly fetching data from a central server, which can significantly reduce total bandwidth consumption. This raises the natural question: Considering that different users (with different end-system capabilities) may request video clips/movies at different bitrates, is it necessary to store video content at the caches at different source rates? With scalable video caching, fortunately, the answer to this equestion is a 'no'.

Several previous works have considered the problems of scalable video caching. Layered scalable video coding (SVC) video caching [20] stores base layers of each movie at the caching node and adjusts the enhancement layers according to user requests. Sanchez et al. [4], [17] studied layered SVC caching over HTTP-streaming and concluded that by using SVC instead of non-scalable coders, the caching efficiency was significantly improved since storing same video clip at several source bitrates is not necessary. Tong et al. [23] considered video streaming in a small cellular network and adopted caching at each base station to reduce backhaul bandwidth consumption while guaranteeing good video quality. Zhou et al. [24] analyzed different working strategies of scalable video caching devices and designed an adaptive algorithm for caching mode selection when dealing with varying video requests. Most previous scalable video caching works focused on layered video caching: unfortunately resulting in a variant of the NP-hard knapsack problem, since the source bitrates provided by each layer are discrete while the cache size is constrained. To circumvent this problem, here we apply the

fine-grained scalable video coder interframe EZBC [22], [11] that has been shown to encode video clips at almost a continuous range source bitrates.

Besides, bandwidth efficiency can be further increased through cooperation among the caches using network coding [1]. The intermediate nodes can linearly combine the incoming data and then forward them to destinations using network coding. Network coding allows the multicast capacity of a network to be achieved and can provide a throughput advantage as compared to routing. A simple network coding approach is to set the field size to 2 (i.e. to perform exclusive-or (XOR) coding) which can be very helpful on boosting network throughput especially for wireless networks. Some works regarding XOR network coding are briefly summarized here. Rout et al. [16] proposed a XOR network coding based algorithm which increased the lifetime of wireless sensor network and also revealed lower latency than solutions without network coding. Hay et al. [8] proposed XOR network coding based protocols which have better packet delivery ratio (PDR) performance than traditional ad hoc routing protocols.

Additionally, several prior works have considered coding with caching. Maddah-Ali et al. [12] developed a decentralized style algorithm for the caching policy by creating simultaneous coded-multicasting opportunities with no centralized coordinations and showed that the proposed method can reduce traffic load on the bottleneck link. Coded-caching is preferred in this work especially when the user quantity is sufficiently large. Wu et al. [21] designed a caching strategy called coding cache that incorporates random linear network coding for a multi-path network, and showed that their strategy can outperform other schemes in terms of cache hit rate.

In this work, we bring the highly scalable video coder into a VoD system with multiple caches. Caches are grouped into pairs, hence XOR network coding and the scalability of SVC coder can be exploited to cut down the backhaul data traffic. To find the (near) optimal cache cooperation pairs, we adopt a heuristic algorithm called harmony search. The performance of different ways of pairing are compared in terms of backhaul data traffic.

The remaining part of the paper is organized as follows. In section II, we describe the system model and the problem formulation. The optimal and heuristic solutions are proposed in section III, and their performance are presented and compared in section IV. Section V concludes the paper.

2 SYSTEM MODEL

2.1 Fine-grained SVC Caching and Network Coding

The system consists of a VoD server S and N_{Cache} caches connecting to the server via a network cloud as shown in Fig. 1. Each cache periodically updates its contents and the period during which the cache content remains unchanged is called a *time-slot*. Let $\mathcal{S}_t$ denote the set of the indices of video clips to be cached for time slot t and let p_i represent the probability of video i. For a given time-slot, the number of requests of video clip i is recorded as n_i. The cache node will select K clips that have the largest requests number n_i to store (i.e. put index i into set $\mathcal{S}_t$) and the *probability* value p_i is then calculated by normalizing n_i: $p_i = n_i / \sum_{i \in \mathcal{S}_t} n_i$. During each update, the cache storage space is reallocated in order to optimize the performance measured by average distortion of the received videos. For convenience, the notation $V_i \triangleq R_i t_i$ is introduced to represent the size of each video clip, where R_i and t_i are the source rate and length of video clip i, respectively. In this work we assume that users tend to view the whole video, and the cache therefore will store the entire video clip at any calculated rate, i.e. the length of each video clip to be cached remains unchanged. The variable V_i is thus a direct one-to-one mapping of source rate R_i. The traditionally used distortion-rate function $D_i(R_i)$ is adopted here to measure quality (e.g. in terms of mean-square error) of video clip i at rate R_i. Given $p_i, i \in \mathcal{S}_t$, the V_i of all video clips are chosen so that the weighted distortion measure $\sum_{i \in \mathcal{S}_t} p_i D_i(R_i)$ is minimized, subject to the cache size constraint. The optimization problem that minimize the average distortion is described as[1]:

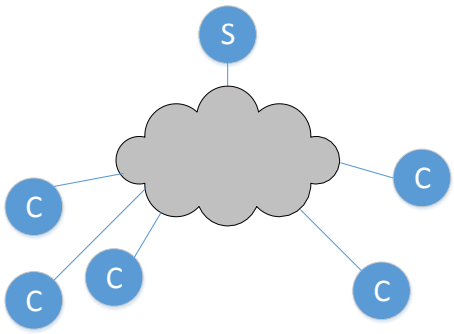

Figure 1: System model of the multiple-cache pairing problem. Each cache may participate into the pairing with another cache to save backhaul data traffic via exclusive-or network coding.

$$\min_{R_i, i \in \mathbf{S}_t} \quad \sum_{i \in \mathcal{S}_t} p_i D_i(R_i) \tag{1}$$

$$s.t. \quad \sum_{i \in \mathcal{S}_t} V_i = C. \tag{2}$$

Note that to fully utilize the cache space, the constraint (2) is an equality constraint.

Considering that the videos requested at one cache may have already been stored in other caches (although perhaps at a lower quality), during an update one cache may refer to other caches before fetching from the server. This fact, along with the use of network coding, can be utilized to reduce backhaul traffic. This is discussed next.

[1] In our experiment, $D_i(R_i)$ uses log MSE instead of MSE.

2.2 Multiple Cache Pairing

The problem of the caching policy design and inter-cache cooperation via network coding was proposed and solved in our previous work [7], i.e. for given $p_i, i \in \mathcal{S}_t$ and C, the calculation of R_i (or V_i if the length of video clip is considered) is provided. However, in [7], cooperation between only two caches is considered. In this work, multiple (more than 2) caches are considered, where the caches involved are separated into groups of two and the video streams are jointly coded using exclusive-or network coding.

We define $\mathcal{C}$ as the set that contains indices of all caches, i.e. cache index $i \in \mathcal{C} = \{1, 2, \ldots, N_{\text{Cache}}\}$. Without loss of generality, we assume that cache i and j will cooperate at time-slot t; hence we define the set $\{i,j\}_t$ as the cooperating cache pair of i and j at time-slot t. Also, we define the set $\mathcal{Q}(t)$ that contains all $\{i,j\}_t$ at time-slot t. Let $V_{i,k}(t)$ and $\Delta_{V_{i,k}}(t)$ denote the size and the corresponding increase of video clip k of cache i at time-slot t, i.e.

$$\Delta_{V_{i,k}}(t) \triangleq \max\{V_{i,k}(t) - V_{i,k}(t-1), 0\}, \tag{3}$$

and let $B_{C^k_{i,j}}(t)$ denote the size of the potentially codable part of video clip k at cache i if cooperating with cache j:

$$B_{C^k_{i,j}}(t) \triangleq \max\{\min\{V_{i,k}(t), V_{j,k}(t-1)\} - V_{i,k}(t-1), 0\}. \tag{4}$$

The total amount of potentially codable rate of cache i if cooperated with j is given by the summation of all potential codable part:

$$B_{C_{i,j}}(t) \triangleq \sum_{k \in \mathcal{S}_t^{C_i}} B_{C^k_{i,j}}(t), \tag{5}$$

where $\mathcal{S}_t^{C_i}$ represents the set of video indices (VID) to be cached in cache i at time-slot t. Also, when two caches requests for the update of the same video clip, the server will send only the overlapping part due to the high scalability of the coder. We use $V_k(t)$ here to represent the size of video clip k required by both cache i, j:

$$\begin{aligned} V_k(t) \triangleq\ & \max\{\min\{V_{i,k}(t), V_{j,k}(t)\} \\ & - \max\{V_{i,k}(t-1), V_{j,k}(t-1)\}, 0\}, \end{aligned} \tag{6}$$

and let $B_{R_{i,j}}(t)$ represent the total amount of $V_k(t)$:

$$B_{R_{i,j}}(t) \triangleq \sum_{k \in \mathcal{S}_t^{C_i} \cap \mathcal{S}_t^{C_j}} V_k(t). \tag{7}$$

Therefore, the aggregate data volume that can be saved by cooperating between cache i and j is

$$B_{i,j}(t) = B_{C_{i,j}}(t) + B_{C_{j,i}}(t) + B_{R_{i,j}}(t). \tag{8}$$

This saving consists of two parts: 1) the saving achieved by network-coding: $B_{C_{i,j}}(t) + B_{C_{j,i}}(t)$; 2) the saving achieved by combining data required by both caches i and j due to video scalability: $B_{R_{i,j}}(t)$. Obviously, we can have $B_{i,j}(t) = B_{j,i}(t)$. Also, according to equation (4), $B_{C^k_{i,j}} = 0$ if $i = j$ and hence $B_{i,i}(t) = 0$. Consequently the resulting backhaul data traffic for cache update is:

$$B_{\text{update}}(t) = \sum_{i=1}^{N_{\text{Cache}}} \sum_{k \in \mathcal{S}_t^{C_i}} \Delta_{V_{i,k}}(t) - B_{\mathcal{Q}}(t), \tag{9}$$

where $B_{\mathcal{Q}}(t)$ stands for the data volume that can be saved via cache cooperation:

$$B_{\mathcal{Q}}(t) = \sum_{\{i,j\}_t \in \mathcal{Q}(t)} B_{i,j}(t). \tag{10}$$

Equation (9) says that the backhaul data traffic for cache update is calculated by summing up the increment of all clips required by every cache, and then subtracting the part saved by network coding.

Since $\Delta_{V_{i,k}}(t)$ is fixed for cache i of video clip k at time-slot t, to minimize the backhaul traffic consumed by cache updates, we need to maximize the backhaul traffic saving $B_{\mathcal{Q}}(t)$ by properly choosing the cooperating cache pairs $\mathcal{Q}(t)$. We can generalize this as the following optimization problem:

$$\max_{\mathcal{Q}(t)} \quad B_{\mathcal{Q}}(t), \tag{11}$$

$$s.t. \quad i, j \in \mathcal{Q}(t). \tag{12}$$

The $B_{i,j}(t)$ consists of all possible pairs as illustrated in Table 1. The optimization problem can be viewed as a *Maximum Weighted Matching* (MWM) problem: the optimal matching is sought such that the total weight of the links in the matching set $\mathcal{Q}(t)$ is maximized. In the following section, we will propose a heuristic optimization approach for this problem.

Table 1: Aggregate data volume between cache i and j.

	Cache 1	Cache 2	Cache 3	...
Cache 1	-	$B_{1,2}(t)$	$B_{1,3}(t)$	...
Cache 2	$B_{2,1}(t)$	-	$B_{2,3}(t)$	...
Cache 3	$B_{3,1}(t)$	$B_{3,2}(t)$	-	...
...	...	...	...	...

3 HARMONY SEARCH ALGORITHM

Optimal MWM problem solutions, although polynomial time, are time-consuming. Gabow et al. in [5] showed that weighted matching problem has a running complexity of $O(|V||E| + |V|^2 \log|V|)$, where $|V|$ and $|E|$ are the number of vertices and edges in a graph respectively. Specifically in this problem, we have $|V| = N_{\text{cache}}$ and $|E| = \binom{N_{\text{Cache}}}{2}$, the complexity of optimal MWM is $O(N^3)$ for a fixed $N = N_{\text{Cache}}$. Hence in this section, a heuristic algorithm called *Harmony Search* (HS) is proposed. Inspired by the improvisation of musicians, the algorithm of the harmony search [25] selects input iteratively from a harmony memory (HM) for each optimization variable ("chord") and then evaluates the output. If the outcome is better than the existing one, the combination of the input will be recorded and the HM is updated accordingly.

The iteration terminates when certain criteria are met e.g. a certain number of iterations have been executed. However, for a fixed HM described above, harmony search can only provide a local optimal solution. A simple modification is also provided in [25] by introducing a harmony memory considering rate (HMCR) and a pitch adjusting rate (PAR). Briefly speaking, HMCR allows the musician to select a pitch from the universal set instead of solely from HM with a certain probability, while PAR allows the musician, after having selected a pitch, to substitute the selected pitch with a neighboring one. In other words, HMCR and PAR enable the algorithm to reach solutions outside the HM with certain probability. Also in [13], harmony search is enhanced by letting PAR varying within a calculated range during iterations instead of being set fixed throughout the test and here we will adopt this strategy.

To apply the improved harmony search to our problem, some modifications are necessary. The structure of the HM is shown in Table 2, where N_{HMS} denotes the harmony memory

Table 2: Harmony memory structure.

Cache 1	2	$\cdots$	N_{Cache}	(0)	Evaluation
C_1^1	C_1^2	$\cdots$	$C_1^{N_{\text{Cache}}}$	C_1^0	$B_{Q_1}(t)$
C_2^1	C_2^2	$\cdots$	$C_2^{N_{\text{Cache}}}$	C_2^0	$B_{Q_2}(t)$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$C_{N_{\text{HMS}}}^1$	$C_{N_{\text{HMS}}}^2$	$\cdots$	$C_{N_{\text{HMS}}}^{N_{\text{Cache}}}$	$C_{N_{\text{HMS}}}^0$	$B_{Q_{N_{\text{HMS}}}}(t)$

size (HMS), i.e. the number of partners for each cache to select to cooperate in HM and $N_{\text{HMS}} \leq N_{\text{Cache}}$. Also, the dummy cache 0 is considered in this structure if N_{Cache} is odd. C_j^i represents the rank j choice of cache in the HM to be cooperating with cache i and $C_j^i \neq i$. Moreover, $B_{\mathcal{Q}_j}(t)$ in the *Evaluation* column stands for the $B_{\mathcal{Q}}(t)$ value of rank j in the HM and $B_{\mathcal{Q}_j}(t)$ follows a decreasing order, i.e. $B_{\mathcal{Q}_1}(t) \geq B_{\mathcal{Q}_2}(t) \geq \cdots \geq B_{\mathcal{Q}_{N_{\text{HMS}}}}(t)$. Each row in Table. 2 corresponds to one possible set of pairings and the higher the row is, the more traffic can be saved via network coding. And after termination of the harmony search, the result of row 1 will be selected as the output.

Harmony memory is initialized by randomly assigning C_j^i a cache index such that $C_j^i \neq i$ and all $B_{\mathcal{Q}_j}(t)$ are set zero. Different from the harmony search in [25] in which all optimizing variables can be selected independently, in this problem caches select partners in sequence so that conflicting pairing can be avoided. Without loss of generality we start from cache 1. Let cache 1 select its partner from $\{C_1^1, C_2^1, \ldots, C_{N_{\text{HMS}}}^1\}$ with probability P_{HMCR} or from cache $\{2, \ldots, N_{\text{Cache}}\}$ with probability $1 - P_{\text{HMCR}}$. After selecting a partner, cache 1 may substitute the index of its partner with $C_j^i + 1$ or $C_j^i - 1$ with probability P_{PAR}. In [13], the harmony search algorithm is improved by letting PAR vary within a provided range instead of being fixed; here we will adopt this strategy. Now the partner of cache 1 is determined and the algorithm can proceed to the next cache in the HM. For any following cache i, if it has been assigned to pair with another cache, the algorithm will jump to the next one. Otherwise it will select a partner via the same procedure of cache 1. After all caches are paired, calculate the corresponding $B_{\mathcal{Q}}(t)$ and update the HM in accordance to the value of $B_{\mathcal{Q}}(t)$.

The procedure of harmony search for this problem is described in **Algorithm 1**, where $\text{PAR}_{\min}$ and $\text{PAR}_{\max}$ represent the lower/upper bound of PAR value, $\text{bw}_{\min}$ and $\text{bw}_{\max}$ stand for the lower/upper bound of pitch adjusting range, N_{Itr} denotes the total number of iterations and function *rand()* returns a uniformly distributed value between $(0, 1)$. During the procedure of **Algorithm 1**, when half of the caches are processed, the other half are automatically paired. For a fixed number of iterations, the running time of **Algorithm 1** is linear to N_{Cache}.

Algorithm 1 Harmony search algorithm for cache pairing

Input: C_j^i, $\text{PAR}_{\min}$,$\text{PAR}_{\max}$,$\text{bw}_{\min}$,$\text{bw}_{\max}$,$B_{\mathcal{Q}_j}(t)$, N_{Itr},P_{HMCR};

for $Itr = 1$ to N_{Itr}
  Calculate $P_{\text{PAR}} = \text{PAR}_{\min} + \frac{\text{PAR}_{\max} - \text{PAR}_{\min}}{N_{\text{Itr}}} Itr$, and $\text{bw} = \text{bw}_{\max} \exp\left(\frac{\ln\left(\frac{\text{bw}_{\min}}{\text{bw}_{\max}}\right)}{N_{\text{Itr}}} Itr\right)$ [13];
  for $i = 1$ to N_{Cache}
    if cache i is not *paired*
      $r_1 = rand()$;
      if $r_1 \leq P_{\text{HMCR}}$
        randomly pick a partner from available C_j^i;
      else
        randomly pick a partner from all available caches;
      end
      $r_2 = rand()$;
      if $r_2 \leq P_{\text{PAR}}$
        if $r_2 \leq 0.5$
          shift the picked partner cache index by $\lceil \text{bw} \cdot r_2 \rceil$
        else
          shift the picked partner cache index by $-\lceil \text{bw} \cdot r_2 \rceil$
        end
      end
      Label the picked cache as *paired*;
    end
  end
  Calculate corresponding $B_{\mathcal{Q}}(t)$
  if $B_{\mathcal{Q}}(t) > B_{\mathcal{Q}_1}(t)$
    Let all $B_{\mathcal{Q}_j}(t) = B_{\mathcal{Q}_{j-1}}(t), j \in \{2, \ldots, N_{\text{HMS}}\}$, and $B_{\mathcal{Q}_1} = B_{\mathcal{Q}}(t)$;
    Let all $C_j^i = C_{j-1}^i, j \in \{2, \ldots, N_{\text{HMS}}\}, i \in \{1, \ldots, N_{\text{Cache}}\}$, and let C_1^i be the picked corresponding partners during this iteration;
  end
end

4 NUMERICAL RESULTS

We first describe the simulation setup before presenting the numerical results. The Netflix Prize data set [2] is adopted as the source of video requests. The data cover a period of 731 days. For the harmony search algorithm, we set the value of HMCR to be 0.85 as [25] and [13] suggest. And for simplicity, we assume that the size of each cache is the same and equal to 12 TB with $K = 12000$. For comparison, some other solutions are also considered:

- *The optimal solution.* For the optimal solution, we directly apply the MWM tool [18] which is developed from the algorithm in [6].
- *Path growing algorithm.* The path growing algorithm provided in [3] starts from an arbitrary point and searches for the path with largest weight value, and put the path link (selected pairs of points) into matched set. The procedure of the path growing algorithm is described in **Algorithm 2**. The path growing algorithm has a running time of $O(|E|)$ where E stands for the set of edges among all vertices in the graph. Since $|E| = \binom{N_{\text{Cache}}}{2}$ the running time of this algorithm is then $O\left(N^2\right)$ for a fixed $N = N_{\text{Cache}}$ [15].
- *Random pairing.* As the name suggests, it groups caches into pairs uniformly at random.
- *No inter-cache cooperation.* Each cache updates independently from the server without any coordination with the other cache.

Algorithm 2 Path Growing Algorithm

Input: $M_1 = \emptyset, M_2 = \emptyset$,idx $= 1$;

```
while elements in Table 1 are not all zeros
  arbitrarily choose a cache i such that at least one B_{i,k}(t) ≠ 0, k ∈ {1,...,N_Cache};
  while at least one B_{i,k}(t) ≠ 0, k ∈ {1,...,N_Cache};
    choose cache j such that {i,j} = arg max B_{i,j}(t);
    put {i,j} into M_idx;
    idx = 3 − idx;
    set all B_{i,k}(t) = B_{k,i}(t) = 0, k ∈ {1,...,N_Cache}
    let i = j;
  end
end
calculate B_Q(t) of M_1, M_2 correspondingly and output the one with higher B_Q(t) value.
```

The results of backhaul data traffic plotted versus the number of caches are provided in Fig. 2. The backhaul traffic of all methods increases as more caches are added to the system, since the source server has to serve more requests. Also, it can clearly be observed that when caches are grouped into pairs for cooperation, backhaul traffic can be saved. This is observed by comparing the 'no cooperation' case with the other curves. Note that the gap between the 'no cooperation' case and other curves becomes more significant as the number of caches increases. As stated previously under (8), this can be attributed into two factors: network coding, and the savings of fine-grained SVC. Additionally, for any given number of caches, the average PSNR of all methods is identical while it ranges from 49.30 dB to 49.36 dB as the number of caches increases from 5 to 18.

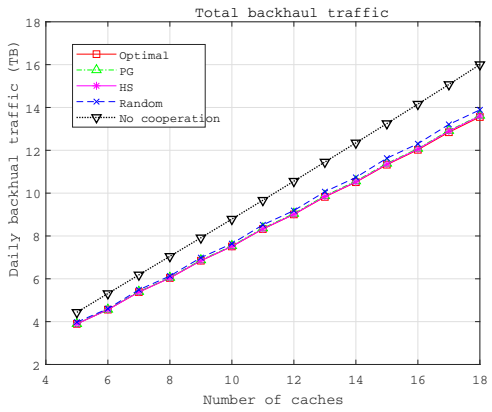

Figure 2: Backhaul traffic versus N_{Cache} for different approaches.

To further compare the performance in terms of backhaul traffic savings, the $B_{\mathcal{Q}}(t)$ value of all approaches is provided in Fig. 3. This figure represents the traffic saved on the backhaul network. Therefore, a higher curve suggests that its corresponding approach results in more bandwidth savings. As can be seen in the figure, the performance of both the harmony search and the path growing algorithm are close to that of the optimal solution, and the the savings achieved by the harmony search is slightly higher than that of the path growing algorithm. The performance difference between the harmony search and the path growing may seem small, but the harmony search is still more preferable considering that the computing complexity of the growing path algorithm is proportional to the square of the number of caches, compared to harmony search's linear complexity. In the following part we will show that the harmony search can actually achieve more backhaul traffic savings than path growing. Random pairing simply groups any two caches into pairs with equal probability and does not make any effort to seek pairing with better performance, hence it only provides the worst result among all approaches.

The results on percentage of performance loss of each method as compared to the optimal result are provided in Fig. 4 and the maxima of each method are listed in Tab. 3. Path growing is an approximation of maximum weighted matching with reduced computing complexity. It can also perform close to the optimal (only having a 1% to 3% performance loss compared to the optimum). Moreover, the harmony search algorithm provides better performance, especially when the number of caches (N_{Cache}) is low. The maximum gap between the performance of the harmony search and the optimum is around 2%. As stated previously, the running time complexity of the harmony search is linear to N_{Cache} for fixed

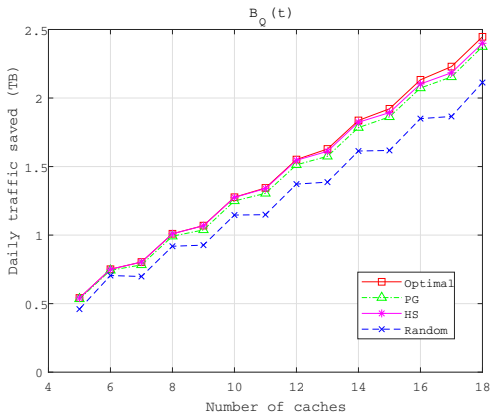

Figure 3: Data traffic on the interlink versus N_{Cache} for different approaches.

iteration numbers while the growing path's is proportional to the square of N_{Cache}. Therefore, harmony search is more preferable than MWM approximation algorithms like path growing, since harmony search can provide similar or better result with less computing time when N_{Cache} is large.

Figure 4: Percentage of performance loss comparing to optimum solution.

Table 3: Maximum performance loss comparing to optimal solution in Fig. 3.

Pairing method	Loss max(%)
Path growing	3.3750
Harmony search	2.0104
Random pairing	16.3236

Note that shapes of all curves in Fig. 3 follow a zigzag pattern, i.e. when N_{Cache} steps from an even number to an odd number the increase of backhaul traffic savings is smaller than stepping from odd to even. This can be attributed to the fact that when N_{Cache} is odd, there will always exist a non-cooperating cache which reduces the amount of backhaul traffic savings. And when N_{Cache} is even, all caches can be paired and hence more traffic can be transferred from the backhaul to the interlinks between the caches.

5 CONCLUSION

In this paper, a maximum weighted matching problem of cache pairing is considered. Within a VoD system, caches cooperate in pairs in order to reduce the backhaul data traffic via fine-grained SVC and XOR network coding, and we concentrate on finding the (near) optimal pairing solution. To solve this problem, the harmony search algorithm is adopted. Besides the optimal MWM solution, an approximate path growing, a random pairing approach and the scenario of not applying cache pairing are also considered for comparison. Numerical results show that by applying cache pairing, backhaul data traffic can be significantly saved. By optimally selecting cooperating pairs, one can further reduce this backhaul traffic. Both the path growing and harmony search algorithms can achieve no more than 4 percent of the performance loss compared to the optimal solution. Since the harmony search can yield better performance savings at lower computing complexity, it may be preferable to use it in practice.

ACKNOWLEDGMENTS

The authors at Rensselaer Polytechnic Institute would like to acknowledge the U.S. National Science Foundation (Award No. 1018398) and Cisco Systems for partial support of this research.

REFERENCES

[1] Rudolf Ahlswede, Ning Cai, Shuo Yen Robert Li, and Raymond W Yeung. 2000. Network information flow. *IEEE Transactions on Information Theory* 46, 4 (2000), 1204–1216.
[2] Lanning S. Bennett J. 2007. The Netflix Prize. In *Proceedings of KDD cup and workshop*.
[3] Doratha E Drake and Stefan Hougardy. 2003. A simple approximation algorithm for the weighted matching problem. *Inform. Process. Lett.* 85, 4 (2003), 211–213.
[4] Yago Sanchez De La Fuente, Thomas Schierl, Cornelius Hellge, Thomas Wiegand, Dohy Hong, Danny De Vleeschauwer, Werner Van Leekwijck, and Yannick Le Loudec. 2011. iDASH: Improved dynamic adaptive streaming over HTTP using scalable video coding. In *Acm Sigmm Conference on Multimedia Systems*.
[5] Harold N Gabow. 1990. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 434–443.
[6] Zvi Galil. 1986. Efficient algorithms for finding maximum matching in graphs. *Acm Computing Surveys* 18, 1 (1986), 23–38.
[7] Qiushi Gong, John W. Woods, Koushik Kar, and Jacob Chakareski. 2015. Fine-Grained Scalable Video Caching. In *IEEE International Symposium on Multimedia*.
[8] Michael Hay, Basil Saeed, Chung Horng Lung, Thomas Kunz, and Anand Srinivasan. 2013. Network Coding and Quality of Service for Mobile Ad Hoc Networks. In *Wireless Communications & Mobile Computing Conference*. 409–422.
[9] Cisco Index. 2017. Cisco visual networking index: Forecast and methodology. *CISCO White paper* (2017).
[10] Wenjie Jiang, Stratis Ioannidis, Laurent Massouli, and Fabio Picconi. 2012. Orchestrating massively distributed CDNs. In *International Conference on Emerging Networking Experiments & Technologies*.

[11] Yuan Liu and John W. Woods. 2017. New and efficient interframe extensions of EZBC and JPEG 2000. 1–6. https://doi.org/10.1109/MMSP.2017.8122289

[12] Mohammad Ali Maddah-Ali and Urs Niesen. 2015. Decentralized Coded Caching Attains Order-Optimal Memory-Rate Tradeoff. *IEEE/ACM Transactions on Networking* 23, 4 (2015), 1029–1040.

[13] M. Mahdavi, M. Fesanghary, and E. Damangir. 2007. An improved harmony search algorithm for solving optimization problems. *Applied Mathematics & Computation* 188, 2 (2007), 1567–1579.

[14] Valentina Martina, Michele Garetto, and Emilio Leonardi. 2013. A unified approach to the performance analysis of caching systems. (2013).

[15] Robert Preis. 1999. Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *Conference on Theoretical Aspects of Computer Science*.

[16] Rashmi Ranjan Rout and Soumya K. Ghosh. 2013. Enhancement of Lifetime using Duty Cycle and Network Coding in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications* 12, 2 (2013), 656–667.

[17] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Loudec. 2012. Efficient HTTP-based streaming using Scalable Video Coding. *Signal Processing Image Communication* 27, 4 (2012), 329–342.

[18] Daniel Saunders. [n.d.]. Weighted maximum matching in general graphs. *Graph* ([n. d.]).

[19] Yufeng Shan, I. V. Bajic, J. W. Woods, and S. Kalyanaraman. 2009. Scalable Video Streaming With Fine-Grain Adaptive Forward Error Correction. *IEEE Transactions on Circuits & Systems for Video Technology* 19, 9 (2009), 1302–1314.

[20] Liang Wu and Wenyi Zhang. [n.d.]. Caching-based Scalable Video Transmission over Cellular Networks. *IEEE Communications Letters* ([n. d.]).

[21] Qinghua Wu, Zhenyu Li, and Gaogang Xie. 2013. CodingCache: Multipath-aware CCN cache with network coding. In *Acm Sigcomm Workshop on Information-centric Networking*.

[22] Yongjun Wu, Konstantin Hanke, Thomas Rusert, and John W. Woods. 2008. Enhanced MC-EZBC Scalable Video Coder. *IEEE Transactions on Circuits & Systems for Video Technology* 18, 10 (2008), 1432–1436.

[23] Tong Zhen, Yuedong Xu, Yang Tao, and Hu Bo. 2017. Quality-Driven Proactive Caching of Scalable Videos over Small Cell Networks. In *International Conference on Mobile Ad-hoc & Sensor Networks*.

[24] Yipeng Zhou, T. Z. J. Fu, Dah Ming Chiu, and Huang Yan. 2013. An Adaptive Cloud Downloading Service. *IEEE Transactions on Multimedia* 15, 4 (2013), 802–810.

[25] Loganathan G.V. Zong Woo Geem, Joong Hoon Kim. 2001. A new heuristic optimization algorithm: harmony search. *Simulation* 76, 2 (2001), 60–68.