Wireless Content Distribution Network

Ziqi Zheng
SEIEE, Shanghai Jiao Tong University

Abstract:
We consider a basic content distribution scenario consisting of a single origin server connected through a shared bottleneck link to a number of users each equipped with a cache of finite memory. Every improved model including the basic one has the limits of caching.
In order to alleviate the congestion during peak-traffic time, we proposed a caching scheme by encoding, decentralization and dynamic allocating. Here we also introduce and compare several different methods of updating scenario. The goal of proposed scheme is to satisfy users' request with the minimum number of bits sent over the shared link by planning the caches in a variety of situations.

Key words: caching, coded, decentralized, popularity, dynamic

I. Introduction

The traditional Internet is design with the philosophy of link-centric, where the end hosts need to figure out the IP addresses of the destination hosts and then communicate. After decades of rapid development, the current Internet has become a platform providing various services instead of a simple communication tunnel. Among these services, the content distribution is the most popular one, which generates a huge number of traffics over the network. Today, the Internet has a notable new trend, where the access to the Internet through wireless links has become the dominant way. However, this new trend also imposes new challenges to the network infrastructure. The fundamental reason is because the wireless links have less bandwidth resources compared to the wired ones. This research is to study how to design the network architecture and corresponding mechanisms to facilitate content distribution over wireless networks.

Traffic in content delivery networks exhibits strong temporal variability, resulting in congestion during peak hours and resource underutilization during off-peak hours. It is therefore desirable to try to "shift" some of the traffic from peak to off-peak hours.

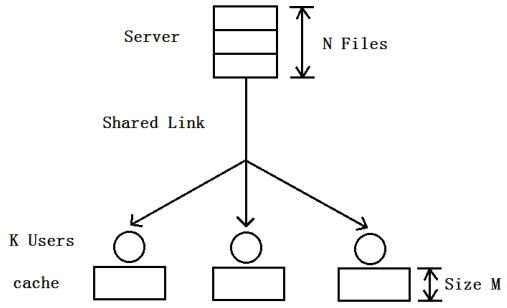

Figure 1. Caching system considered in this paper. A server containing N files of size F bits each is connected through a shared link to K users each with an isolated cache of size MF bits. The goal is to design the placement phase and the delivery phase such that the peak rate (i.e. the load normalized by the file size) of the shared bottleneck link is minimized. In the figure, N = K = 3 and M = 1.

Our discussion and results are all builds on the basic model sketched in Figure1. K users are connected to a server through a shared, error-free link. The server has a database of N files of equal size. Each of the users has access to a cache memory big enough to store M of the files. We will adopt these notations throughout this paper. Just this model makes the "shift" mentioned above possible.

For example, there are lots of movies in server. If you have downloaded movie A to your local caches during off-peak hours, then you can watch it immediately whenever you want. Nevertheless you can't watch movie B right now obviously. And our objective is to design a caching allocation strategy in off-peak such that the load(L) of the shared link during peak hours is minimized.

Example 1 *(Uncoded Caching)*

For a memory size of MF bits (normalized to M bits), one possible strategy is for each user to cache the same M/N fraction of each file during off-peak hours. In the traffic peak, the server simply transmits the remaining (1−M/N) fraction of any requested file over the shared link. Clearly, each user can recover its requested file from the content of its local cache and the signal sent over the shared link. In the worst case the users request different files, results in the delivery load for this caching scheme is thus

$$L_w = \min\{K, N\} \cdot (1 - {}^{M}/_{N})$$

We refer to this caching allocation strategy as *uncoded caching*.

II. Improvement Result

The server can never know which file the users want in advance. Like in section I, filling local cache with single file for one user may lead to extreme results (L=0 or L=M). So example 1 is a usual substitute scenario. Here we introduce an approach that achieves a significant reduction in network load.

Example 2 *(Coded Caching)*

For simplicity, Consider the case N = K = 2, so that there are two files, say F_1 = A, F_2 = B, and two users each with cache memory of size M.

Firstly, let us consider the two extreme cases M = 0 and M = N. Our fundamental model makes no sense when M = 0. If M = N, no optimizing is needed. So we consider the more interesting case, that is M = 1. We split both files A and B into two subfiles of equal size, i.e., A = (A_1, A_2) and B = (B_1, B_2). During off-peak time, we put Z_1 = (A_1, B_1) and Z_2 = (A_2, B_2) into user 1's and user 2's local caches separately. Assume for example that user 1 requests file A and user 2 requests file B. given that user 1 already has subfile A_1, it only needs to obtain the missing subfile A_2, which is cached in the second user's memory Z_2. Similarly, user two only needs to obtain the missing subfile B_1, which is cached in the first user's memory Z_1. In other words, each user has one part of the file that the other user needs.

The server can in this case simply transmit$A_2 \oplus B_1$, where $\oplus$ denotes bitwise XOR. And we know

Formula 1 *If* $c = a \oplus b$*, then it must be*

$$a = b \odot c$$

where $\odot$ *denotes bitwise XNOR, also*

$$b = a \odot c$$

Since user 1 already has B_1, it can recover A_2 from $A_2 \oplus B_1$. Similarly, since user two already has A_2, it can recover B_1 from $A_2 \oplus B_1$. Thus, the transmitting signal received over the shared link helps both users to effectively exchange the missing subfiles available in the cache of the other user.

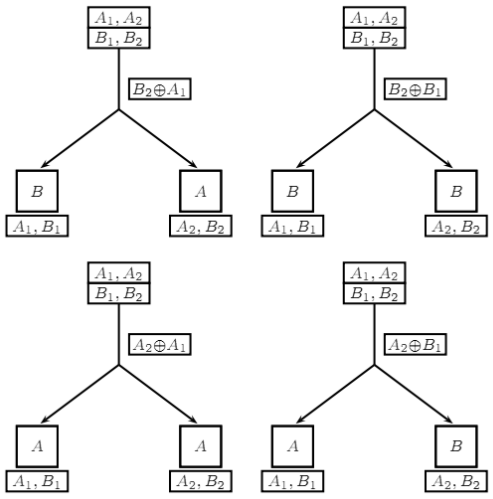

Figure 2. Caching strategy for N = 2 files and K = 2users with cache size M = 1 with all four possible user requests. Each file is split into two subfiles of size 1/2, i.e., A = (A_1, A_2) and B = (B_1, B_2). The scheme achieves rate R = 1/2.

The signals sent over the shared link for all other requests are illustrated in Figure 2. Now we extend it to arbitrary parameters. But for general, $0 \leq M \leq N$, and we focus on the case $N \geq K$, in which

$$L_c = \frac{K}{1 + KM/N} \cdot (1 - M/N)$$

The factor $1/(1 + KM/N)$ is called global caching gain. It is to be interpreted as a multicasting gain available simultaneously for all possible demands.

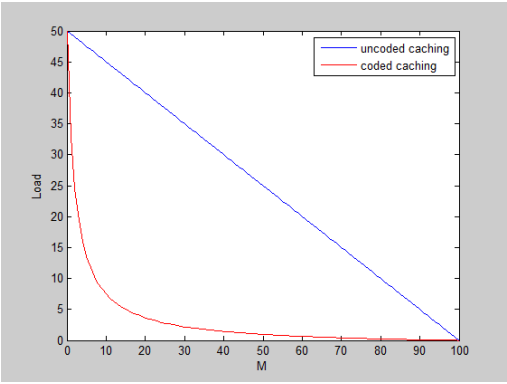

Figure 3. Comparison of uncoded and coded caching load during peak hours.

Let us use MATLAB to compare the delivery load of uncoded scheme versus coded scheme with N = 100 and K = 50 as shown in Figure 3.

III. Influence of Real-Life

Crucially, the coded caching scheme has its limitation like any system. It may be invalid when those ideal assumptions don't work anymore. So we continue discussing some follow-up work.

A. Decentralized Caching

The coded caching scheme described in section II has both already known the number and the identity of the users in delivery during off-peak hours. This is clearly not a realistic assumption since we usually do not know in the morning which users will request content in the following evening. Moreover, if instead of the synchronized user requests here we have more realistic asynchronous requests, then users join and leave the system over a period of several hours during the peak hours, resulting in a time-varying number of users. Finally, users may be in different networks during the two periods of time.

In this part, we solve this problem in the positive by developing a caching algorithm that creates simultaneous coded-multicasting opportunities without coordination in the off-peak hours.

Example 3 *(Decentralized coded caching)* Consider the caching problem with N = 2 files A and B, and K = 2 users each with a cache of size M. During off-peak hours, each user caches a subset of MF/2 bits of each file independently at random. As a result, each bit of a file is cached by a specific user with probability M/2. Let us focus on file A. The actions of the placement procedure effectively partition file A into 4 subfiles,

$$A = (A_0, A_1, A_2, A_{12})$$

Where A_0 denotes the bits of file A that are not stored in anyone's caches. A_1 and A_2 denote the bit of file A that are stored in the cache memories of user 1 and 2 separately. A_{12} belongs to both user 1 and 2.

Theorem 1 (Law of large numbers) *Let* $X_1, X_2, \ldots, X_n$ *be a sequence of independent and identically distributed random variables, each having a mean* μ*. Define a new variable*

$$X = \frac{1}{n} \cdot (X_1 + X_2 + \cdots + X_n)$$

Then, as $n \to \infty$*, the sample mean equals the population mean of each variable.*

$$\lim_{n\to\infty} P(|X - \mu| < \varepsilon) = 1$$

for all $\varepsilon > 0$.

If file size F is large enough, actually even a 10MB file contains 81920 bits which is large enough, each bit can be interpreted as a random variable in the sequence. So the probability of subfiles is approximately

$$\mathrm{P}(A_0) = \left(1 - {}^{M}/_{2}\right)^2$$

$$\mathrm{P}(A_1) = \mathrm{P}(A_2) = \frac{M}{2} \cdot \left(1 - {}^{M}/_{2}\right)$$

$$\mathrm{P}(A_{12}) = \left(\frac{M}{2}\right)^2$$

Assume that user 1 requests file A and user 2 requests file B. Since the cache of user 1 has A_1 and the cache of user 2 contains B_2. Hence the server needs to transmit $\mathrm{T} = (A_2 \oplus B_1) + A_0 + B_0$.

The size of T (surely normalized by F) is $(M/2) \cdot (1 - M/2) + 2(1 - M/2)^2$

This can be rewritten as

$$(2/M)(1 - M/2)[1 - (1 - M/2)^2].$$

For general, if $\mathrm{M} \geq 1$ and $\mathrm{N} \geq \mathrm{K}$, the minimum load is

$$L_d = \frac{N}{M} \cdot \left(1 - {}^{M}/_{N}\right) \cdot [1 - \left(1 - {}^{M}/_{N}\right)^{\mathrm{K}}]$$

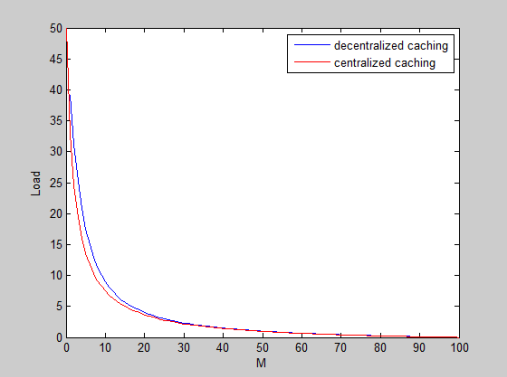

Figure 4. Comparison of centralized and decentralized caching load during peak hours.

From Figure 4, we can find the decentralized caching load if a little more than centralized one when affected by reality reasons. But its performance is good enough.

B. Non-uniform File Popularities

Hitherto in this paper, we have adopted a worst-case definition of rate with respect to user requests. However, different pieces of content have usually different popularities (i.e., probabilities of being requested by the users). In order to capture this effect, the definition of rate needs to be changed from worst case to expected value.

Considering the impact of specific file popularities $p_1, p_2, \ldots, p_n$ on the performance of caching, we use an optimal strategy, least-frequently used (LFU), in uncoded caching scheme with only a single user (K=1).To be surprised, perhaps our intuition tell us the strategy does not carry over to multiple users (K>1). In fact, we will argue that LFU can be arbitrarily suboptimal in the multi-user setting.

We now describe the proposed scheme in detail. Let us partition the N files into s groups. In the statement of algorithm, we use the notation $p_i, i \in \mathbb{N}$ and $i < \mathrm{N}$ to denote the popularity of file I except zero. p_0 is the least popularity among all files in current group.

Algorithm 1 Geometric Partitioning

procedure MERGE SORT
  if (low==high) small = large
  else
    mid = (low+high) / 2
    MergeSort(small,large,low,mid)
    MergeSort(small,large,mid+1,high)
    Merge(small,large,low,mid,high)
  end if
end procedure
procedure PARTITIONING
  $p_0 = p_1$
  head $= 1$
  for i=1,2,…,N do
    if (p_i>2p_0)
      $p_0 = p_i$
      Group $(p_{head}, \dots, p_{i-1})$
      head = i
    end if
  end for
end procedure

In order of most popular, we get these groups $G_1, G_2, \dots, G_s$ from smallest to largest popularity. And in the same group, the popular file won't be more than double the out of fashion file.

The server uses the same delivery procedure as in section II s times, once for each group of users K_s. Then each subfile times a weight factor w_s differed by group and aggregate. The more popular, the larger w_s is. Of course the sum of each weighted subfile size must be M.

$$L_n \leq \min \sum_{i=1}^{s} \mathbb{E}[L_d(M_i, N_i, K_i)]$$

Where $\mathbb{E}$ denotes expectation, $L_d(M, N, K)$ is defined in section II. It's worth noting that we are interested in expected load instead of peak load here because of the characteristics of model.

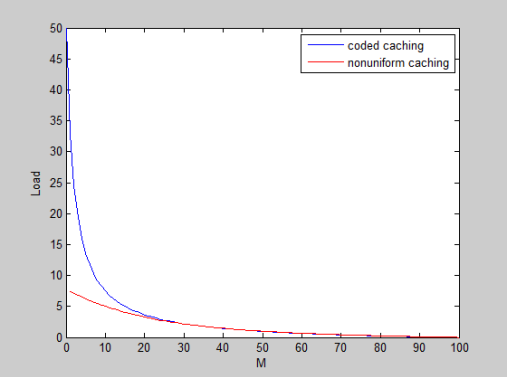

Figure 5. Comparison of general coded caching and divide and conquer caching scheme performance in expected load.

The popularity distribution is varied and conditions become complicated. We constrain those specific parameters as in Figure caption above. The expected load is apparently decrease when the memory of caches is not quite a few.

C. Online Caching

Maybe you have noticed that caches is updated only during the off-peak hours, but not during the peak time. Many caching systems used in practice use online cache updates, in which a decision to update the cache is made in the traffic peak. One popular update rule is least-recently used (better known by its abbreviation LRU), in which the least-recently requested file is evicted from the cache.

Example 4 *(LRU)*

A popular online caching scheme is LRU. In this scheme, the content of a user's cache at the beginning of time slot t is a function of the cache content at time t-1, the output of the shared link at time t-1, and the past requests.

Consider a toy system with N = 2 popular files. A possible evolution of the caches is as follows.

t = 1: Assume the initial set of popular files is {B, C}.

t=2: There is an arrival. The file C is randomly chosen and replaced with file D, so that the set is {B, D}.

t=3: There is no arrival, and the set still is {B, D}.

We now introduce an online version of the caching algorithm, which we term coded least-recently sent (LRS).

Algorithm 2 Coded LRS caching for time t.

1: **procedure** PEAK HOURS
2: **for** s=K, K-1, …, 1 **do**
3: **for** $S \subseteq [K]: |S| = s$ **do**
4: server sends $\bigoplus_{k \in S} V_{S\{k\}}(k)$
5: **end for**
6: **end for**
7: **end procedure**
8: **procedure** CACHE UPDATE
9: **end procedure**
10: **procedure** PARTITIONING
11: **for** $k, k' \in [K]$ **do**
12: **if** $d_t(k')$ is not cached at user k
13: **then** k replaces the least recently sent file in cache with a random subset of $\frac{MF}{N'}$ bits of file $d_t(k')$
14: **end if**
15: **end for**
16: **end procedure**

Consider next the cache update procedure. In each time slot t, the users maintain a list of $N \triangleq \alpha N$, $\alpha \geq 1$.

Example 5 *(Coded LRS)*

We consider again a system with N = 2 popular files and assume the same popular-file dynamics as in example 4. Assume there are K = 2 users with a cache memory of M = 1. Let $\alpha = 1.5$ so that each user caches 1/3 of $N' = \alpha N = 3$ files. We presume that initially each user partially caches the files {A, B, C}.

t = 1: The set of popular files is {B, C}. Assume the users request $d_1 = (B, C)$. Both of the requested files are partially cached at the users. In the peak hours procedure, the server send B_0, C_0 and $B_2 \oplus C_1$. This results in a load of

$$L_d(M, N', 2) = 10/9$$

And each user still partially caches {A, B, C}.

t=2: The set changes to {B, D}. Assume the users request $d_2 = (B, D)$. This results in a load of

$$L_d(M, N', 1) + 1 = 15/9$$

The least-recently sent file A is evicted from each cache and replaced by a random third of the file D. The new set is {B, C, D}.

t=3: The set stays {B, D}. Assume the users request $d_3 = (D, B)$, both of which are now partially cached at the users. Unlike before, D_1 is now no longer empty, and the resulting load is

$$L_d(M, N', 2) = 10/9$$

As calculated before. The set stays the same, namely {B, C, D}.

There are three key differences between LRU and LRS. First, coded LRS uses a coded peak hours procedure whereas the transmissions in LRU are uncoded. Second,

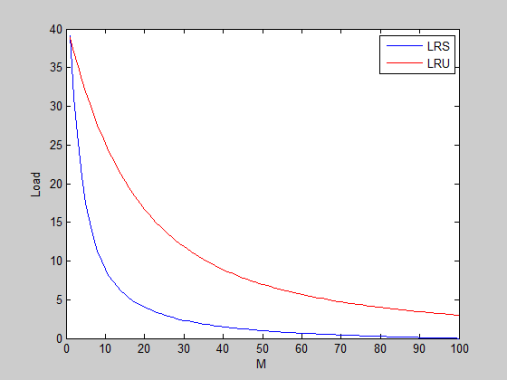

Figure 6. Comparison of least-recently used and least-recently sent strategy.

coded LRS caches many partial files whereas LRU caches fewer whole files. Third, coded LRS uses a LRS eviction rule, taking into account the files requested by all users jointly, compared to the LRU eviction rule, taking into account only the files requested by every user individually.

REFERENCES

[1] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," arXiv:1209.5807 [cs. IT], Sept. 2012. Submitted to IEEE Trans. Inf. Theory.

[2] M. A. Maddah-Ali and U. Niesen, "Decentralized caching attains order-optimal memory-rate tradeoff," arXiv:1301.5848 [cs. IT], Jan. 2013

[3] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," arXiv:1308.0178 [cs. IT], Aug. 2013.

[4] R. Pedarsani, M. A. Maddah-Ali and U. Niesen, "Online coded caching," arXiv:1311.3646 [cs. IT], Nov. 2013.