uc3m | Universidad Carlos III de Madrid

e-Archivo

This is a postprint version of the following published document:

Luo, T., Aggarwal, V., & Peleato, B. (2019). Coded Caching With Distributed Storage. IEEE Transactions on Information Theory, 65(12), 7742–7755.

DOI: 10.1109/tit.2019.2940979

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Coded caching with distributed storage

Tianqiong Luo, Vaneet Aggarwal, *Senior Member, IEEE*, and Borja Peleato, *Member, IEEE*

***Abstract*—Content delivery networks store information distributed across multiple servers, so as to balance the load and avoid unrecoverable losses in case of node or disk failures. Coded caching has been shown to be a useful technique which can reduce peak traffic rates by pre-fetching popular content at the end users and encoding transmissions so that different users can extract different information from the same packet. On one hand, distributed storage limits the capability of combining content from different servers into a single message, causing performance losses in coded caching schemes. But, on the other hand, the inherent redundancy existing in distributed storage systems can be used to improve the performance of those schemes through parallelism.**

This paper designs coded caching and delivery schemes tailored towards systems where the library is distributed across multiple servers, possibly with some redundancy in the form of maximum distance separable (MDS) erasure codes. Different schemes are proposed based on the capacity of the users' caches, as well as the number of parity servers. The main focus is on scenarios with one (RAID-4) or two (RAID-6) parity servers, but the paper also includes simple extensions for cases with more than two or no parity servers at all. The proposed schemes are shown to reduce the worst case latency, or equivalently the peak transmission rate from any server, below that of state-of-the-art algorithms.

***Index Terms*—Coded caching, storage system, RAID-4, RAID-6, erasure codes**

I. Introduction

For several decades, CPUs have doubled their speed every two years in what is commonly known as Moore's law, but the storage technology has not been able to keep up with this trend: magnetic hard drives have steadily increased their capacity, but not their speed. Current computers and communication networks are not limited by the speed at which information can be processed, but rather by the speed at which it can be read, moved, and written. Furthermore, the recent information explosion is driving an exponential increase in the demand for data, which is not expected to slow down any time soon. Users and applications demand more data at higher speeds, straining the devices and networks to their maximum capabilities.

The IT industry has addressed this problem through parallelism and caching: instead of using a single high capacity storage drive to serve all the requests, networks usually distribute popular files across multiple independent servers that can operate in parallel and cache part of the information at intermediate or final nodes. This paper proposes and analyzes multiple caching mechanisms for multi-server systems with different system parameters. Previous literature has addressed coded caching for single server systems and distributed storage without caching but, to the extent of our knowledge, this is the first work that considers both coded caching at the users and distributed storage at the servers. Furthermore, it provides solutions for systems with and without file striping (*i.e.*, with files split among multiple servers and with whole files stored in each server).

Distributed storage deals with how the information is stored at the servers. Disk failures are very common in large storage systems, so they need to have some amount of redundancy. Erasure codes have recently sparked a renewed interest from the research community for this task. Files are encoded and distributed among a set of nodes (disks, servers, etc.) in such a way that the system can recover from the failure of a certain number of nodes [1]. One widely used distributed storage technique based on erasure codes is RAID (redundant array of independent disks). It combines multiple storage nodes (disks, servers, etc.) into a single logical unit with data redundancy. Two of the most common are RAID-4 and RAID-6, consisting of block-level striping with one and two dedicated parity nodes, respectively [2], [3]. Most large scale systems use some form of RAID with striping across multiple storage drives, but some store or replicate whole files as a single unit in the network nodes (*e.g.*, data centers) [4], [5]. This increases the peak rate, but it also simplifies book-keeping and deduplication, improves security, and makes the network more flexible.

Caching deals with the high temporal variability of network traffic: the peak traffic in the network is reduced by pre-fetching popular content in each receiver's local cache memory during off-peak hours, when resources are abundant. Coded caching has also recently become quite popular among the coding community, starting with the work by Maddah-Ali and Niesen in [6], which focused on how a set of users with local memories can efficiently receive data from a single server through a common link. Their seminal paper proposed a caching and delivery scheme offering a worst case performance within a constant factor of the information-theoretic optimum, as well as upper and lower bounds on that optimum. The lower bounds were later refined in [7] and new schemes were designed to consider non-uniform file sizes and popularity [8], [9], [10]; multiple requests per user [11], [12]; variable number of users [13]; decentralized caching [14]; and multiple servers with access to the whole library of files [15].

Maddah-Ali and Niesen's work in [6] caches the infor-

T. Luo is with Google Inc, Mountain View, CA 94043. V. Aggarwal and B. Peleato are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907. (tianqiongluo@gmail.com, {vaneet,bpeleato}@purdue.edu)

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
DOI: 10.1109/TIT.2019.2940979

mation uncoded and encodes the transmitted packets. This scheme performs well when the cache size is relatively large, but a close inspection shows that there are other cases in which its performance is far from optimal. Tian and Chen's recent work in [16] designs a new algorithm which encodes both the cached and transmitted segments to achieve a better performance than [6] when the cache size is small or the number of users is greater than the number of files. Both of the previous schemes assume that a single server stores all the files and can combine any two segments into a message. Then, they design a list of messages to be broadcast by the server, based on the users' requests. In practice, however, it is often the case that content delivery networks have multiple servers and throughput is limited by the highest load on any one server rather than by the total traffic between servers and users. Shariatpanahi et al. addressed this case in [15], but still assumed that all servers had access to all the files and could therefore compose any message. They proposed a load balancing scheme distributing the same list of messages among all the servers, scaling the peak rate by the number of servers.

If each server only has access to some of the files, the problem is significantly more complicated. The general case, where each segment can be stored by multiple servers and users, is equivalent to the multi-sender index coding problem. The index coding problem is one of the core problems of network information theory but it remains open despite significant efforts from the research community [17], [18], [19]. Its generalization to multiple senders is even more challenging [20], [21]. Instead of addressing the multi-sender index coding problem in its general form, we focus on the case where each data segment is stored in a single server, all caches have the same capacity, and users request a single file.

Figure 1 illustrates the system model that we will consider. Multiple servers with different content are connected to a set of cache-enabled users through broadcast links. All users share the same link to each server, but the servers operate on orthogonal, independent and error-free links, without interfering or sharing resources with each other. Furthermore, servers take an erasure coding approach to redundancy, without file replication. Hence, each data segment is stored in a single data server in plain form, and redundant servers store linear combinations of multiple segments. One relevant area where this model could be of interest is femtocaching [22]. In femtocaching, users reconstruct their requested files from distributed cache-equipped small base stations (SBS) with backhaul links. Files are replicated or coded at multiple SBSs so that users can fulfill their requests by connecting to a subset of the SBSs.

Summarizing, prior work on distributed storage has studied how a single user can efficiently recover data distributed across a set of nodes and prior work on coded caching has studied how a set of users with local memories can efficiently receive data from a single node. However, to the extent of our knowledge, it has not been studied how the cache placement and content delivery should be performed when multiple nodes send data to multiple users through independent channels. In this paper, we aim to design placement and delivery protocols for the multi-server multi-user system. We combine

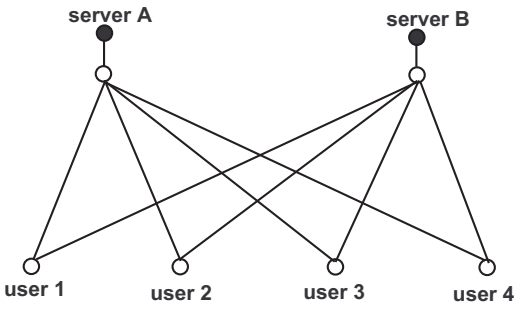

FIGURE 1: Illustration of the system model with 2 servers and 4 users. All users are connected to the same server though a shared link.

distributed storage with coded caching utilizing parallelism and redundancy to reduce the worst case latency for any set of requests, or equivalently the peak traffic rate on a single link. The main contributions of our paper are: (1) a flexible model for multi-server systems where each file can be divided among multiple servers (striping) or kept as a single block in one server; (2) an extension of the coded caching algorithms in [6] and [16] to multi-server systems; (3) new caching and delivery schemes offering significantly lower peak rates by taking advantage of parity servers.

The rest of the paper is structured as follows: Section II introduces the system model and two existing coded caching algorithms for single server systems, namely the one proposed by Maddah-Ali and Niesen in [6] and the interference elimination scheme in [16]. Section III extends both algorithms to a multi-server system with file striping, while Sections IV and V consider the case where servers store whole files. Specifically, Section IV extends Maddah-Ali and Niesen's scheme, suitable for systems with large cache capacity, and Section V extends the interference elimination scheme, which provides better performance when the cache size is small. Section VI bounds the gap to optimality for the proposed schemes. Finally, Section VII provides simulations to support and illustrate our algorithms and Section VIII concludes the paper.

II. Background

This section describes the multi-node multi-server model in II-A and then reviews two existing coded caching schemes that constitute the basis for our algorithms. Subsection II-B summarizes Maddah-Ali and Niesen's coded caching scheme from [6] and subsection II-C summarizes Tian and Chen's interference elimination scheme from [16].

A. System Model

We consider a network with K users[1] and N files distributed across L data servers. Some parts of the paper will also

[1] Servers and users can be anything from a single disk to a computer cluster, depending on the application.

include additional parity servers, denoted parity server P when storing the bitwise XOR of the information in the data servers (RAID-4) and parity server Q when storing a different linear combination of the data (RAID-6). The network is assumed to be flexible, in the sense that there is a path from every server to every user [15], but all users share the link to each server as illustrated in Fig. 1. All servers store the same number of files and each user has a cache with capacity for M files. For the sake of simplicity, this paper assumes that all files have identical length and popularity.

The servers are assumed to operate on independent error-free channels, so that two or more servers can transmit messages simultaneously and without interference to the same or different users. A server can broadcast the same message to multiple users without additional cost in terms of bandwidth, but users cannot share the content of their caches with each other. This assumption makes sense in a practical setting since peer-to-peer content sharing is generally illegal. Also, users typically have an asymmetric channel, with large download capacity but limited upload speed.

Similarly, each server can only access the files that it is storing, not those stored on other servers. A server can read multiple segments from its own files and combine them into a single message, but two files stored on different servers cannot be combined into a single message. However, it will be assumed that servers are aware of the content cached by each user and of the content stored in other servers, so that they can coordinate their messages. This can be achieved by exchanging segment IDs through a separate low-capacity control channel or by maintaining a centralized log.

The problem consists of two phases: placement and delivery. During the placement phase, the content is stored in the user's caches. The decisions on where to locate each file, how to compute the parity, and what data to store in each cache are made based on the statistics for each file's popularity, without knowledge of the actual user requests. The delivery phase starts with each user requesting one of the files. All servers are made aware of these requests and proceed to send the necessary messages.

The goal of this paper is to design placement and delivery schemes to minimize the worst case latency for any set of requests. Such latency will be determined by the maximum amount of information to be transmitted by any one server, so our schemes will attempt to balance the load across all of them. The content of each server is assumed to be pre-fixed, but multiple cases are considered: files can be spread across all data servers (Section III) or stored as single unit in one server (Sections IV and V). Furthermore, we will consider scenarios with and without parity servers storing linear combinations of the files.

Throughout the paper, we use subindices to represent file indices and superindices to represent segment indices, so F_i^j will represent the j-th segment from file F_i. Some parts of the paper will also use different letters to represent files from different servers. For example, A_i represents the i-th file from server A and A_i^j represents the j-th segment from file A_i. The paper focuses on minimizing the peak rate (or delay), implicitly assuming that different users request different files. Therefore, we will indistinctly refer to users or their requests.

B. Maddah-Ali and Niesen's scheme

The coded caching scheme proposed by Maddah-Ali and Niesen in [6] has a single server storing all the files $\{F_1, F_2 \dots, F_N\}$, and users are connected to this server through a shared broadcast link. Their goal is to design caching and delivery schemes so as to minimize the peak load on the link, *i.e.*, the total amount of information transferred from the server to the users. This scheme splits each file F_i into $\binom{K}{t}$ nonoverlapping segments F_i^j of equal size, $j = 1, \dots \binom{K}{t}$, with $t = \frac{KM}{N}$, and caches each segment in a distinct group of t users. In other words, each subset of t users is assigned one segment from each file for all the users to cache[2]. In the delivery phase the server sends one message to each subset of $t+1$ users, for a total of $\binom{K}{t+1}$ messages. This caching scheme ensures that, regardless of which files have been requested, each user in a given subset of $t+1$ nodes is missing a segment that all the others have in their cache. The message sent to that subset of nodes consists of the bitwise XOR of all $t+1$ missing segments: a set of users $\mathbf{S}$ requesting files $F_{i_1}, F_{i_2}, \dots, F_{i_{t+1}}$ would receive the message

$$m^{\mathbf{S}} = F_{i_1}^{j_1} \oplus F_{i_2}^{j_2} \oplus \cdots \oplus F_{i_{t+1}}^{j_{t+1}}, \tag{1}$$

where j_k is the index for the segment cached by all the users in the set except the one requesting F_{i_k}. Each user can then cancel out the segments that it already has in its cache to recover the desired segment. In the worst case, *i.e.*, when all users request different files, this scheme yields a (normalized by file size) peak rate of

$$\begin{aligned} R_C(K,t) &= \binom{K}{t+1} \Big/ \binom{K}{t} \\ &= K(1-M/N)\frac{1}{1+KM/N}. \end{aligned} \tag{2}$$

Under some parameter combinations, broadcasting all the missing segments uncoded could require lower rate than $R_C(K,t)$, so the generalized peak rate is

$$\min\{R_C(K,t), N-M\}$$

but this paper will ignore those pathological cases, assuming that N, M, and K are such that $R_C(K,t) \leq N-M$. It has been shown that this peak rate is the minimum achievable for some parameter combinations and falls within a constant factor of the information-theoretic optimum for all others [6], [23].

This scheme, henceforth refered to as "M-N scheme" will be the basis for multiple others throughout the paper. It is therefore recommended that the reader has a clear understanding of the M-N scheme before proceeding.

[2]Parameter t is assumed to be an integer for the sake of symmetry. Otherwise some segments would be cached more often than others, requiring special treatment during the delivery phase and complicating the analysis unnecessarily.

C. Interference Elimination

A close examination of Maddah's algorithm reveals that it has poor performance when the cache is small and $N \leq K$. Thus, a new coded caching scheme based on interference elimination was proposed by Tian and Chen in [16] for the case where the number of users is greater than the number of files. Instead of caching file segments in plain form, they propose that the users cache linear combinations of multiple segments. After formulating the requests, undesired terms are treated as interference that needs to be eliminated to recover the requested segment. The transmitted messages are designed to achieve this using maximum distance separable (MDS) codes [24][25].

In the placement phase, this scheme also splits each file into $\binom{K}{t}$ non-overlapping segments of equal size and each segment is cached by t users, albeit combined with other segments. Let $F_i^{\mathbf{S}}$, where $\mathbf{S} \subseteq \{1, 2, \ldots, K\}$ and $|\mathbf{S}| = t$, denote the file segment from file F_i chosen to be cached by the users in $\mathbf{S}$. In the placement phase user k collects the file segments

$$\{F_i^{\mathbf{S}} | i \in \{1, 2, \ldots, N\}, k \in \mathbf{S}\}, \tag{3}$$

($P = \binom{K-1}{t-1} N$ in total), encodes them with a MDS code $\mathcal{C}(P_0, P)$ of length $P_0 = 2\binom{K-1}{t-1} N - \binom{K-2}{t-1}(N-1)$, and stores the $P_0 - P$ parity symbols in its cache.

The delivery phase proceeds as if all the files are requested. When only some files are requested, the scheme replaces some users' requests with the "unrequested files" and proceeds as if all files were requested. A total of $\binom{K-1}{t}$ messages are transmitted (either uncoded or coded) for each file F_i, regardless of the requests. Uncoded messages provide the segments that were not cached by the users requesting F_i, while coded messages combining multiple segments from F_i are used to eliminate the interference in their cached segments. Each user gathers $\binom{K-2}{t-1}(N-1)$ useful messages which, together with the $P - P_0$ components stored in its cache, are enough to recover all P components in the $\mathcal{C}(P_0, P)$ MDS code. A more detailed description of the messages can be found in [16].

Therefore, the total number of messages transmitted from the server is $N\binom{K-1}{t}$. In this interference elimination scheme, the following normalized (M, R) pairs are achievable:

$$\left(\frac{t\left[(N-1)t + K - N\right]}{K(K-1)}, \frac{N(K-t)}{K} \right), \; t = 0, 1, \ldots, K. \tag{4}$$

This scheme is shown to improve the inner bound given in [6] for the case $N \leq K$ and has a better performance than the algorithm in subsection II-B when the cache capacity is small.

III. File striping

The simplest way to extend single-server coded caching algorithms to a multi-server system is to spread each file across all servers. That way, each user will request an equal amount of information from each server, balancing the load. This is called data striping [26] and it is common practice in data centers and solid state drives (SSD), where multiple drives or memory blocks can be written or read in parallel. The users then allocate an equal portion of their cache to each server and the delivery is structured as L independent single-server demands. We now proceed to give a detailed description of how striping can reduce the peak rate of M-N scheme, but the same idea can be applied to any other scheme.

Each of the N files $\{F_1, F_2 \ldots, F_N\}$ is split into L blocks to be stored in different servers and each block is divided into $\binom{K}{t}$ segments. These segments are denoted by $F_i^{(j,m)}$, where $i = 1, 2, \ldots, N$ represents the file number; $j = 1, 2, \ldots, \binom{K}{t}$ the segment number; and $m = 1, 2, \ldots, L$ the block number. The m-th server is designed to store the m-th block of each file, that is $F_i^{(j,m)}$ for every i and j.

The placement is the same as in M-N scheme. Each segment is cached by t users, with $\{F_i^{(j,1)}, F_i^{(j,2)}, \ldots, F_i^{(j,L)}\}$ being cached by the same user. We notice that each message transmitted by M-N scheme in Eq. (1) can be split into L components

$$F_{i_1}^{(j_1,m)} \oplus F_{i_2}^{(j_2,m)} \oplus \cdots \oplus F_{i_{t+1}}^{(j_{t+1},m)}, \tag{5}$$

$m = 1, 2, \ldots, L$ to be sent by different servers. Then the problem can be decomposed into L independent single-server subproblems with reduced file sizes of $\frac{F}{L}$ bits. The subproblems have the same number of users, files, and cache capacity (relative to the file size) as the global problem. Since all servers can transmit simultaneously, the peak load is reduced to $\frac{1}{L}$ of that in Eq. (2) (M-N single server scheme).

If one additional parity server P is available (RAID-4), it will store the bitwise XOR of the blocks for each file, *i.e.*, $F_i^{(j,1)} \oplus F_i^{(j,2)} \oplus \cdots \oplus F_i^{(j,L)}$ for all i and j. Then, server P can take over some of the transmissions, reducing the peak load to $\frac{1}{L+1}$ of that with M-N scheme[3]. Specifically, instead of having all data servers transmit their corresponding component in Eq. (5), server P can transmit the XOR of all those components, relieving one data server from transmitting. The users can combine the $L-1$ messages from other data servers with this XOR to obtain the missing one. Similarly, if two additional parity servers P and Q are available (RAID-6), it is possible to choose any L out of the $L+2$ servers to take care of each set of messages in Eq. (5), thereby reducing the peak rate to $\frac{1}{L+2}$ of that with M-N scheme.

A similar process with identical file splitting can be followed for the interference cancelling scheme, achieving the same scaling of the peak rate: $\frac{1}{L}$ when there is no parity, $\frac{1}{L+1}$ with a single parity server, and $\frac{1}{L+2}$ with two parity servers. We therefore conclude that, when file striping is used, the system can take full advantage of the parallel channels. The inability to combine segments from different servers in a single message has no impact in the peak rate.

In practice, however, some storage systems prefer to avoid striping and store whole files as a single unit in each server to simplify the book-keeping, ensure security, and make the network more flexible [4], [5]. The rest of the paper will focus on the case where nodes store entire files, and each user requests a file stored in a specific node.

[3]The number of segments must be a multiple of L to achieve this reduction, but it is always possible to divide each segment into multiple chunks to fulfil this condition.

Server A	Server B	$\cdots$	Server L
A_1	B_1	$\cdots$	L_1
A_2	B_2	$\cdots$	L_2
$\vdots$	$\vdots$		$\vdots$
A_r	B_r	$\cdots$	L_r

TABLE I: Files stored in each server

IV. Scheme 1: Large cache

In this section, we extend Maddah-Ali and Niesen's scheme to the multiple server system. Instead of spreading each file across multiple servers as in Section III, each file is stored as a single unit in a data server, as shown in Table I. The parity servers store linear combinations of data segments and offer redundancy in the system.

The performance of M-N scheme in Eq. (2) is highly dependent on the cache capacity M. Compared with the interference elimination scheme in section II-C, the advantage of M-N scheme lies in that file segments are stored in plain form instead of encoded as linear combinations. This saves some segments from being transmitted in the delivery phase, but it requires larger cache capacities to obtain coded caching gains. Hence, M-N scheme is appropriate when the cache capacity is large.

The placement phase of our algorithm is identical to that in the traditional scheme. For example, in a system with $K = 6$ users with cache capacity $M = 4$ and $N = 8$ files, each file is divided into 20 segments and each segment is stored by $t = 3$ users. Table III indicates the indices of the 10 segments that each user stores, assumed to be the same for all files without loss of generality.

In order to simplify later derivations, the notation is clarified here. Since the peak rate for the storage system is considered, we assume that all users request different files, hence each user can be represented by the file that it has requested. Denote $\mathbf{S}$ to be the user set and $m_A^{\mathbf{S}}$ to represent the message sent from server A to all the users in $\mathbf{S}$. Furthermore, if $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_i\}$ represents a vector of file indices and $\boldsymbol{\gamma} = \{\gamma_1, \gamma_2, \ldots, \gamma_i\}$ represents a vector of segment indices, then $\mathbf{A}_{\boldsymbol{\alpha}}$ represents the set of requests (or users)

$$\mathbf{A}_{\boldsymbol{\alpha}} = \{A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_i}\}$$

and $\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}}$ represents the message

$$\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} = A_{\alpha_1}^{\gamma_1} \oplus A_{\alpha_2}^{\gamma_2} \oplus \ldots \oplus A_{\alpha_i}^{\gamma_i},$$

where A_i^j represents the j-th segment from the i-th file in server A. Similarly, $\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}}$ represents the the message:

$$\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} = (A_{\alpha_1}^{\gamma_1} \oplus B_{\alpha_1}^{\gamma_1}) \oplus \ldots \oplus (A_{\alpha_i}^{\gamma_i} \oplus B_{\alpha_i}^{\gamma_i}).$$

We start by exploring an intuitive extension of M-N scheme to the multi-server system, in subsection IV-A. Then, subsection IV-B proposes an idea to further lower the peak rate in a simple system with two data and one parity server. Finally, subsections IV-C and IV-D extend that idea to the cases with one and two parity servers, respectively.

A. Intuitive extension to multiple servers

In a system without redundancy, such as the one shown in Table I, the servers cannot collaborate with each other. During the delivery phase, each user is assigned to the server storing the file that it requested, and then each data server transmits enough messages to fulfil its requests. Specifically, following M-N scheme, a server receiving m requests would need to transmit $\binom{K}{t+1} - \binom{K-m}{t+1}$ messages, *i.e.*, one for each group of t users containing at least one of its requesters. The normalized peak rate for that server would therefore be

$$\left(\binom{K}{t+1} - \binom{K-m}{t+1}\right) \Big/ \binom{K}{t}$$

The worst case occurs when all users request files from the same server, *i.e.*, $m = K$. Then the peak transmission rate is the same as in the single server system, shown in Eq. 2.

With parity servers storing linear combinations of the data, the peak rate can be reduced. In general, distributed storage systems use MDS codes for the parity[4], so any subset of L servers can be used to generate any message. For example, consider the three-server system shown in Table II, if message $m = A_1^1 \oplus B_2^2$ needed to be transmitted, where A_1^1 is stored in server A and B_2^2 is stored in server B, server A would transmit A_1^1 and server B would transmit B_2^2. Alternatively, we can have server A transmit $A_1^1 \oplus A_2^2$ and the parity server transmit $A_2^2 \oplus B_2^2$. Any two out of these three servers can compose any message. Rotating among all subsets of L servers would allow each server to participate in at most a fraction $\frac{L}{L+L'}$ of the messages in M-N scheme, where L' is the number of parity servers. Therefore, a simple balancing of the load would allow us to scale the normalized peak rate to

$$\frac{L}{L+L'} R_C(K,t), \tag{6}$$

where $R_C(K,t)$ is the peak rate of M-N scheme, defined in Eq. 2. However, we intend to design caching and delivery algorithms capable of further reducing the peak rate of any one server as shown in subsections IV-B, IV-C,and IV-D.

B. One parity and two data servers

This section focuses on a very simple storage system with two data servers and a third server storing their bitwise XOR, as shown in Table II. Despite each server can only access its own files, the configuration in Table II allows composing any message by combining messages from any two servers. Intuitively, once all requests for files stored in server A (or B) have been fulfilled, that server can work with the parity server to fulfill requests for files stored in server B (or A). This allows serving two requests for files stored in the same server in parallel, balancing the load and reducing the worst case peak rate below that achieved without the parity server (see Section IV-A).

However, there is a better transmission scheme where messages from all three servers are combined to get more information across to the users. The basic idea is to include

[4]Some systems use repetition or pyramid codes [27], [28], [29] to reduce the recovery bandwidth, but this paper will focus on MDS codes.

Server A	Server B	Server P
A_1	B_1	$A_1 \oplus B_1$
A_2	B_2	$A_2 \oplus B_2$
$\vdots$	$\vdots$	$\vdots$
A_r	B_r	$A_r \oplus B_r$

TABLE II: Files stored in each server

some unrequested segments, as well as the requested ones, in each message from a data server. If the additional segments are well chosen, they can be combined with messages from the parity server to obtain desired file segments. The algorithm developed in this section is based on this idea.

Just like in M-N scheme, data servers will send each message to a set of $t+1$ users and the message will contain the XOR of $t+1$ segments (one for each user). These segments are chosen so that all users except the intended receiver can cancel them out. If the user had requested a file stored by the sender, the message will contain the corresponding segment; otherwise the message will include its complement in terms of the parity in server P, *i.e.*, A_i^j instead of B_i^j and vice versa. Therefore, the contents of each message from server A or B are uniquely determined by the sender and the set of receivers, denoted by S_1 or S_2 respectively. For example, if the caches and requests are as shown in Table III, the message from server A to $S_1 = \{A_1, A_2, A_3, B_4\}$, corresponding to users 1 through 4, will be $m_A^{\mathbf{S_1}} = A_1^{11} \oplus A_2^5 \oplus A_3^2 \oplus A_4^1$.

Lemma 4.1: Let the receivers for servers A and B be

$$S_1 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{A}_*\} \qquad S_2 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{B}_*\},$$

respectively, where α and β denote (possibly empty) sets of indices, the $*$ denote arbitrary sets, and $S_1 \neq S_2$. The corresponding messages are

$$m_A^{\mathbf{S_1}} = \mathbf{A}_{\boldsymbol{\alpha}}^* \oplus \mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{A}_*^* \qquad m_B^{\mathbf{S_2}} = \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^* \oplus \mathbf{B}_*^*,$$

with segment indices chosen so that each user can cancel all but one of the components. This provides users $\mathbf{B}_{\boldsymbol{\beta}}$ and $\mathbf{A}_{\boldsymbol{\alpha}}$ with some unrequested segments $\mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}$ and $\mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}}$, respectively. Then server P can send the message

$$m_P^{\mathbf{S_1} \cap \mathbf{S_2}} = (\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}}) \oplus (\mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}),$$

to $S_1 \cap S_2$, so that each user in S_1 and S_2 obtains a requested segment and those in the intersection obtain two. These three transmissions are equivalent to messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ as defined in Eq. (1) for M-N single server scheme. They both provide the same requested segments to their destinations.

Proof: All the users in S_1 and S_2 get at least one desired segment, from the server storing their requested file. Those in $S_1 \cap S_2$ also receive an unrequested segment from server A or B. It only remains to prove that users in $S_1 \cap S_2$ can use this unrequested segment to obtain its complement from $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$.

Without loss of generality, consider user $B_{\beta_i} \in S_1 \cap S_2$. The set of segment indices γ in $m_A^{\mathbf{S_1}}$ were chosen so that user B_{β_i} is caching all the segments except the γ_i-th. Similarly, the set of indices η in $m_B^{\mathbf{S_2}}$ was chosen so that B_{β_i} is caching all of them (for all files). Therefore, B_{β_i} can obtain $A_{\beta_i}^{\gamma_i}$ from $m_A^{\mathbf{S_1}}$ and should be able to cancel all terms from $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$ except $A_{\beta_i}^{\gamma_i} \oplus B_{\beta_i}^{\gamma_i}$. Combining both of these yields the desired segment $B_{\beta_i}^{\gamma_i}$. As long as $S_1 \neq S_2$, this segment will be different from the one that B_{β_i} obtains from $m_B^{\mathbf{S_2}}$ because there is a one-to-one relationship between segment indices and user subsets. ■

Segment\ User	1	2	3	4	5	6
1	X	X	X			
2	X	X		X		
3	X	X			X	
4	X	X				X
5	X		X	X		
6	X		X		X	
7	X		X			X
8	X			X	X	
9	X			X		X
10	X				X	X
11		X	X	X		
12		X	X		X	
13		X	X			X
14		X		X	X	
15		X		X		X
16		X			X	X
17			X	X	X	
18			X	X		X
19			X		X	X
20				X	X	X
Request	A_1	A_2	A_3	B_4	B_1	B_2

TABLE III: Mapping of file segments to user caches. Each cache stores the same 10 segments for every file, marked with X in the table.

Take the case in Table III as an example. Lemma. 4.1 states that if $S_1 = \{A_1, A_2, A_3, B_4\}$ and $S_2 = \{A_1, A_2, B_1, B_4\}$, we construct $m_A^{\mathbf{S_1}}$, $m_B^{\mathbf{S_2}}$, $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$ as:

$$\begin{aligned} m_A^{\mathbf{S_1}} &= A_1^{11} \oplus A_2^5 \oplus A_3^2 \oplus A_4^1, \\ m_B^{\mathbf{S_2}} &= B_1^{14} \oplus B_2^8 \oplus B_1^2 \oplus B_4^3, \\ m_P^{\mathbf{S_1} \cap \mathbf{S_2}} &= (A_1^{14} \oplus B_1^{14}) \oplus (A_2^8 \oplus B_2^8) \oplus (A_4^1 \oplus B_4^1). \end{aligned}$$

It can be verified that these messages are equivalent to two transmissions in M-N scheme, specifically those intended for users $\{A_1, A_2, A_3, B_4\}$ and $\{A_1, A_2, B_1, B_4\}$.

Corollary 4.1.1: Assume $S_1 = \{\mathbf{A}_*, \mathbf{B}_{\boldsymbol{\beta}}\}$ and $S_2 = \{\mathbf{B}_*\}$, *i.e.*, it only contains requests for server B. Then server P sends $m_P^{\mathbf{B}_{\boldsymbol{\beta}}} = \mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}$ to all the users in $\mathbf{B}_{\boldsymbol{\beta}}$ in Lemma 4.1, so that all the users in S_1 and S_2 get the same segments as in M-N scheme. The same holds switching the roles of A and B.

Proof: This is a particular case of Lemma 4.1 when $\boldsymbol{\alpha}$ is empty ($\boldsymbol{\beta}$ can be empty or non-empty). ■

Definition 4.1: If user subsets S_1 and S_2 fulfill the conditions in Lemma 4.1, we call (S_1, S_2) an **effective pair**.

Our goal is to design a scheme equivalent to M-N scheme while minimizing the maximum number of messages sent by any one server. If two user subsets form an effective pair, the corresponding messages in M-N scheme (see Eq. (1)) can be replaced by a single transmission from each of the three servers. Hence, we wish to make as many effective pairs as possible.

Lemma 4.2: The peak rate is $\left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t)$ for the server system in Table II, where Δ represents the ratio of unpaired messages, $t = \frac{KM}{N}$, and $R_C(K,t)$ was defined in (2).

Proof: For each effective pair, we can use a single transmission from each server to deliver the same information as two transmissions in M-N single server scheme. This contributes $\frac{1}{2}(1-\Delta)R_C(K,t)$ to the total rate. Unpaired messages are transmitted as described in subsection IV-A, that is combining messages from any two out of the three servers. Assuming that this load is balanced among all three servers, the contribution to the total rate is $\frac{2}{3}\Delta R_C(K,t)$. Adding both contributions yields the rate above. ∎

The following lemma characterizes the ratio of unpaired user subsets Δ in the case with symmetric requests (both servers receive the same number of requests).

Lemma 4.3: If the requests are symmetric, then $\Delta = 0$ when t is even and $\Delta \leq \frac{1}{3}$ when t is odd. That is, the following peak rate is achievable in the case with symmetric requests:

$$R_T(K,t) = \begin{cases} \frac{1}{2}R_C(K,t) & \text{if } t \text{ is even} \\ \left(\frac{1}{2}+\frac{1}{6}\Delta\right)R_C(K,t) & \text{if } t \text{ is odd,} \end{cases} \quad (7)$$

where $R_C(K,t)$ is defined in Eq. (2).

Proof: A pairing algorithm with these characteristics is presented in the Appendix. ∎

Although Δ can reach $\frac{1}{3}$, in most cases the pairing algorithm in the Appendix performs much better. As an example, Table III has each segment cached by $t = \frac{KM}{N} = 3$ users and the normalized peak rate with the pairing algorithm is $\frac{2}{5}$, significantly lower than the $\frac{3}{4}$ with M-N single server scheme.

Finally, we are ready to derive an achievable peak rate for a general set of requests, based on the following lemma.

Lemma 4.4: If (S_1, S_2) form an effective pair, then $S_1' = \{S_1, \mathbf{A_\alpha}\}$ and $S_2' = \{S_2, \mathbf{A_\alpha}\}$ also form an effective pair of a larger dimension. The same holds when an all-B file set is appended instead of the all-A file set $\mathbf{A_\alpha}$.

Proof: The proof is straightforward by observing that (S_1', S_2') still fulfills the conditions in Lemma 4.1. ∎

The extension of Lemma 4.3 to the asymmetric case is as follows. Let K_A and K_B respectively denote the number of requests for servers A and B, and assume $K_A > K_B$ without loss of generality. Divide the $K = K_A + K_B$ requests (or users) into two groups: the first with K_B requests for each server (symmetric demands) and the second with the remaining $K_A - K_B$ requests for server A. We construct effective pairs of length $t+1$ by appending requests from the second group to effective pairs from the first.

Theorem 4.5: If the requests are asymmetric, the ratio of unpaired messages is also bounded by $\Delta \leq \frac{1}{3}$. Specifically, if K_A and K_B respectively denote the number of requests for servers A and B, assuming $K_A > K_B$ without loss of generality, the following normalized peak rate is achievable:

$$R(K_A, K_B, t) = \sum_{l=0}^{t+1} \binom{K_A - K_B}{l} R_T(2K_B, t-l), \quad (8)$$

where R_T is defined in Eq. (7) and $K = K_A + K_B$.

Proof: From Lemma 4.3, $R_T(2K_B, t-l)$ represents the peak rate after pairing all subsets of $t+1-l$ requests from the symmetric group. For each $l = 0, 1, \ldots, t+1$, we multiply $R_T(2K_B, t-l)$ by the number of possible completions with l requests from the second group, to obtain the peak rate corresponding to subsets with $t+1-l$ requests from the first group and l from the second. Adding them for all l gives Eq. (8).

Since $R_T(i,j) \leq \left(\frac{1}{2}+\frac{1}{6}\Delta\right)R_C(i,j)$ with $\Delta \leq \frac{1}{3}$ by Lemma 4.3, and $\sum_{l=0}^{t+1}\binom{K_A-K_B}{l}R_C(2K_B, t-l) = R_C(K,t)$ by combinatorial equations, Eq. (8) implies that $R(K_A, K_B, t) \leq \left(\frac{1}{2}+\frac{1}{6}\Delta\right)R_C(K,t)$ with $\Delta \leq \frac{1}{3}$ as defined in Lemma 4.2. ∎

Corollary 4.5.1: A peak rate of $\frac{5}{9}R_C(K,t)$ is achievable for a system with two data servers and a parity server.

C. One parity and L data servers

The previous subsection has discussed the case with two data servers and one parity server, but the same algorithm can be extended to systems with more than two data servers. Intuitively, if there are L data servers and one parity server, any message can be built by combining messages from any L servers. A first approach could be distributing the $\binom{K}{t+1}$ messages in M-N scheme across the $L+1$ possible groups of L servers, as proposed in subsection IV-A. Each server would then need to send a maximum of $\binom{K}{t+1}\cdot\frac{L}{L+1}$ messages. However, there is a more efficient way of fulfilling the requests based on the algorithms in subsections IV-A and IV-B.

Lemma 4.6: Let $S_1 = \{\mathbf{A_\alpha}, \mathbf{B_\beta}, \mathbf{A}_*, \mathbf{Y}\}$ and $S_2 = \{\mathbf{A_\alpha}, \mathbf{B_\beta}, \mathbf{B}_*, \mathbf{Y}'\}$ be two user subsets, where $\mathbf{Y}$ and $\mathbf{Y}'$ are arbitrary lists of requests for servers C through L and the $*$ represent arbitrary (possibly empty) index sets. Then, S_1 and S_2 can be paired so that servers A, B and P require a single transmission to provide the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in M-N single server scheme. The other data servers, C through L, require a maximum of two transmissions, as shown in Fig. 2.

Proof: The transmissions would proceed as follows:

1) Servers C through L each send two messages, to S_1 and S_2. For example, server C would send $m_C^{S_1}$ and $m_C^{S_2}$, providing a desired segment to users requesting files from C and the corresponding C-segments to those requesting other files.
2) Server A sends[5] $m_A^{S_1}$, providing a desired segment to users requesting $\{\mathbf{A}_*, \mathbf{A_\alpha}\}$ and the corresponding undesired A-segments to those requesting $\mathbf{B_\beta}$.
3) Server B sends $m_B^{S_2}$, providing a desired segment to users requesting $\{\mathbf{B_\beta}, \mathbf{B}_*\}$ and the corresponding undesired B-segments to those requesting $\mathbf{A_\alpha}$.
4) Server P sends $m_P^{\{\mathbf{A_\alpha}, \mathbf{B_\beta}\}}$ to users requesting $\{\mathbf{A_\alpha}, \mathbf{B_\beta}\}$. Using the undesired segments previously received, the users in $\{\mathbf{A_\alpha}, \mathbf{B_\beta}\}$ can solve for the desired A and B segments.

A simple comparison of the requested and received segments shows that these transmissions deliver the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in M-N single server scheme. ∎

As an example, Table IV shows the segments that each user gets in transmissions (1)-(4) when $S_1 = \{A_1, A_2, B_1, C_1\}$

[5]It would be enough for A to send $m_A^{\{\mathbf{A}_*, \mathbf{A_\alpha}, \mathbf{B_\beta}\}}$ instead of $m_A^{S_1}$, but we use the latter for the sake of simplicity. The same applies to the message from server B.

Trans.\Req.	A_1	A_2	B_1	B_2	C_1	C_2
(1)	C_1^5		C_1^3		C_1^4	C_2^8
(2)	A_1^1	A_2^2	A_1^3			
(3)	B_1^5		B_1^6	B_2^7		
(4)	P_1^5		P_1^3			
in total	A_1^1, A_1^5	A_2^2	B_1^3, B_1^6	B_2^7	C_1^4	C_2^8

TABLE IV: Segments received by each user in transmissions (1)-(4) from Lemma 4.6, where $P_i^j = A_i^j \oplus B_i^j \oplus C_i^j$.

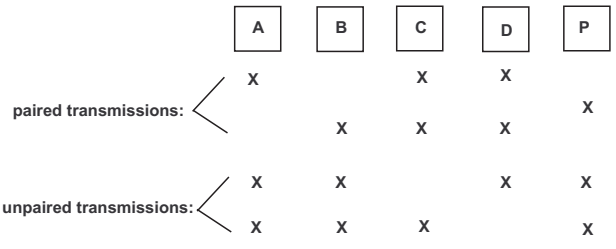

FIGURE 2: Pairing for 4 data servers and one parity server system. A, B, C, D are data servers and P represents the parity server. X means there is a message transmitted from the corresponding server.

and $S_2 = \{A_1, B_1, B_2, C_2\}$, respectively corresponding to segments $\{A_1^1, A_2^2, B_1^3, C_1^4\}$ and $\{A_1^5, B_1^6, B_2^7, C_2^8\}$.

Theorem 4.7: The following normalized peak rate is achievable for a system with $L \geq 3$ data servers and one parity server:

$$R_P(K,t) = \frac{L-1}{L} R_C(K,t), \tag{9}$$

where R_C is defined in Eq. (2).

Proof: First we show that we can deliver $\frac{2}{L}\binom{K}{t+1}$ of the messages in M-N scheme using at most $\frac{1}{L}\binom{K}{t+1}$ transmissions from servers A, B and P; and at most $\frac{2}{L}\binom{K}{t+1}$ transmissions from each of the other servers. This can be done by pairing the messages as shown in Lemma 4.6, if they include requests for A or B, and by using the scheme in subsection IV-A, if they do not.

Selecting these $\frac{2}{L}\binom{K}{t+1}$ messages can be done as follows: group messages by the number of segments that they have from servers A or B. Within each group, we pair the messages as shown in Lemma 4.6. This is equivalent to pairing the A and B requests into effective pairs according to Theorem 4.5 and considering all possible completions for each pair using requests for other servers. Theorem 4.5 showed that at least $\frac{2}{3} \geq \frac{2}{L}$ of the messages in each group can be paired. Messages which have no A or B segments can be transmitted as described in section IV-A, without requiring any transmissions from servers A, B or P.

The remaining $\frac{L-2}{L}\binom{K}{t+1}$ messages can be transmitted as described in subsection IV-A (composing any message by L out of $L+1$ servers), distributing the savings evenly among servers C through L. This requires $\frac{L-2}{L}\binom{K}{t+1}$ transmissions from servers A, B and P, and $\frac{L-3}{L}\binom{K}{t+1}$ from each of the rest.

Each server then transmits a total of $\frac{L-1}{L}\binom{K}{t+1}$ segment-sized messages, hence the peak rate in Eq. (9). ■

Server P	Server Q
$A_1 + B_1 + \ldots + L_1$	$A_1 + \kappa_B B_1 + \ldots + \kappa_L L_1$
$A_2 + B_2 + \ldots + L_2$	$A_2 + \kappa_B B_2 + \ldots + \kappa_L L_2$
$\vdots$	$\vdots$
$A_r + B_r + \ldots + L_r$	$A_r + \kappa_B B_r + \ldots + \kappa_L L_r$

TABLE V: Files stored in parity servers in RAID-6

Theorem 4.7 provides a very loose bound for the peak rate in a system with one parity and L data servers. In practice, there often exist alternative delivery schemes with significantly lower rates. For example, if all the users request files from the same server, that server should send half of the messages while all the other servers collaborate to deliver the other half. The rate would then be reduced to half of that in M-N scheme. Similarly, if $L > t+1$ and all the servers receive similar numbers of requests, the scheme in IV-A can provide significantly lower rates than Eq. (9).

D. Two parity and L data servers

This subsection extends our algorithm to a system with L data and two linear parity servers operating in a higher order field instead of GF(2). The parity server P is assumed to store the horizontal sum of all the files and the parity server Q is assumed to store a different linear combination of the files BY ROW, as shown in Table V. It will be assumed that the servers form an MDS code. We will show that with a careful design of the delivery strategy, the peak rate can be reduced to almost half of that with M-N single server scheme.

Lemma 4.8: Let $S_1 = \{\mathbf{A}_*, \mathbf{Y}\}$ and $S_2 = \{\mathbf{B}_*, \mathbf{Y}\}$, where $\mathbf{Y}$ represents a common set of requests from any server. Then S_1 and S_2 can be paired so that a single transmission from each server fills the same requests as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in Eq. (1).

Proof: The transmission scheme shares the same pairing idea as the algorithm in subsection IV-B. The transmissions are as follows:

1) Server A sends $m_A^{\mathbf{S_1}}$, providing a desired segment to users requesting its files and the corresponding undesired A-segments to others.
2) Server B sends $m_B^{\mathbf{S_2}}$, providing a desired segment to users requesting its files and the corresponding undesired B-segments to others.
3) Servers $C, D, \ldots, L$ each send a single message to $S_1 \bigcap S_2 = \{\mathbf{Y}\}$ with the following content for each user:
   - Users requesting files from server B received some undesired segments from server A. Servers $C, D, \ldots, L$ send them the matching ones so that the desired segments can be decoded using the parity in server P later.
   - The remaining users in $\mathbf{Y}$ will get the desired segment corresponding to S_1 when possible, otherwise they will get the undesired segment corresponding to S_2.

   In other words, each server $C, \ldots, L$ will send segments corresponding to S_1 to users requesting its files or those

Trans.\Req.	A_1	A_2	B_1	B_2	C_1	C_2
(1)	A_1^1	A_2^2	A_1^3		A_1^4	A_2^5
(2)	B_1^6		B_1^7	B_2^8		
(3)	C_1^6		C_1^3		C_1^9	C_2^{10}
(4)	P_1^6		P_1^3		P_1^4, Q_1^4	P_2^5, Q_2^5
in total	A_1^1, A_1^6	A_2^2	B_1^3, B_1^7	B_2^8	C_1^4, C_1^9	C_2^5, C_2^{10}

TABLE VI: Segments users get in (1)-(4) transmissions (In order to simplify notation, denote $P_i^j = A_i^j + B_i^j + C_i^j$ and $Q_i^j = A_i^j + \kappa_B B_i^j + \kappa_C C_i^j$).

from server B, and segments corresponding to S_2 to the rest. At this point, all the users have satisfied their requests related to S_1, except those requesting files from server B, who satisfied their requests related to S_2 instead. Each user has also received $L-2$ undesired "matched" segments[6], corresponding to S_1 for those requesting files from server B and corresponding to S_2 for the rest.

4) Finally, parity servers P and Q each transmit a message to $S_1 \bigcap S_2 = \{\mathbf{Y}\}$ with a combination of segments for each user (see Table V). Those requesting files from server B will get two combinations of the segments corresponding to S_1, while the rest will get two combinations of the segments corresponding to S_2. Since each user now has $L-2$ individual segments and two independent linear combinations of all L segments, it can isolate the requested segment (as well all the "matching" segments in other servers).

A simple comparison of the requested and received segments shows that these transmissions deliver the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in M-N single server scheme. ■

As an example, Table VI shows the delivered segments in transmissions (1)-(4) if $m^{\mathbf{S_1}} = \{A_1^1, A_2^2, B_1^3, C_1^4, C_2^5\}$ and $m^{\mathbf{S_2}} = \{A_1^6, B_1^7, B_2^8, C_1^9, C_2^{10}\}$.

Theorem 4.9: For the L data server and two parity server system, the following normalized peak rate is achievable:

$$R_Q(K,t) = \left(\frac{1}{2} + \frac{L-2}{2L+4}\Delta\right) R_C(K,t), \qquad (10)$$

where $\Delta \leq \frac{1}{3}$ is the pairing loss and R_C is the rate of the single server M-N scheme in Eq. (2).

Proof: Group messages by the number of segments that they have from servers A or B. Within each group, we pair the messages as shown in Lemma 4.8. If the number of requests from A or B is not zero, this is equivalent to pairing the A and B requests into effective pairs according to Theorem 4.5 and considering all possible completions for each pair using requests for other servers. Theorem 4.5 showed that at most $\frac{1}{3}$ of the messages in each group remains unpaired. For the messages which do not contain segments from A or B we repeat the same process with two other servers, with identical results: at most $\frac{1}{3}$ of them remain unpaired.

Each pair of messages can be delivered using a single transmission from each server, as shown in Lemma 4.8, hence paired messages contribute $\frac{1}{2}(1-\Delta)R_C(K,t)$ to the total rate, where Δ denotes the ratio of unpaired messages. Unpaired messages are transmitted as described in subsection IV-A, that is using L out of the $L+2$ servers. Balancing this load among all the servers, they contribute $\frac{L}{L+2}\Delta R_C(K,t)$ to the total rate. Adding both contributions yields the rate above. ■

[6]Users in $\mathbf{Y}$ requesting files from servers A or B received $L-1$ "matched" segments instead of $L-2$, but we can ignore the extra one.

V. Scheme 2: Small cache

This section extends the interference elimination scheme in section II-C to a multi-server system. The interference elimination scheme is specially designed to reduce the peak rate when the cache size is small [16]. Unlike M-N scheme, which caches plain segments, the interference elimination scheme proposes caching linear combinations of them. That way each segment can be cached by more users, albeit with interference. This section will start by addressing the system without parity in Table I, showing that the transmission rate decreases as $\frac{1}{L}$ with the number of servers. Then it performs a similar analysis for the case with parity servers, which can be interpreted as an extension of the user's caches.

Theorem 5.1: In a system with L data servers and parallel channels, the peak rate of the interference cancelling scheme can be reduced to $\frac{1}{L}$ of that in a single server system, *i.e.*, the following (M,R) pair is achievable:

$$\left(\frac{t\left[(N-1)t+K-N\right]}{K(K-1)}, \frac{N(K-t)}{LK}\right), \; t = 0,1,\ldots,K. \qquad (11)$$

This holds regardless of whether each file is spread across servers (striping) or stored as a single block in one server.

Proof: Section III showed that striping the files across L servers reduces the peak rate of the interference cancelling scheme by $\frac{1}{L}$ compared with a single server system.

In contrast to M-N scheme, the interference cancelling scheme sends the same number of segments from each file, regardless of the users' requests. Moreover, each message consists of a combination of segments from a single file [16]. Therefore, the same messages can be transmitted even if different files are stored in different servers. Each server will need to transmit a fraction $\frac{1}{L}$ of the messages, since it will be storing that same fraction of the files. The peak load can then be reduced to $\frac{1}{L}$ of that in Eq. (4). ■

If there are parity servers, we can further reduce the transmission rate by regarding them as an extension of the users' cache. Section II-C explained that in the interference elimination algorithm [16], each user caches the parity symbols resulting from encoding a set of segments with a systematic MDS code $\mathcal{C}(P_0, P)$. It is possible to pick the code in such a way that some of these parity symbols can be found as combinations of the information stored in servers P and Q. Then, instead of storing them in the user's cache, they are discarded. Those that are needed in the delivery phase will be transmitted by the parity servers.

For example, parity server P stores the horizonal sum of the files, so it can transmit messages of the form:

$$\sum_{i=1}^{N/L} \sum_{j=1}^{\binom{K-1}{t-1}} \lambda_{ij} \left(A_i^{\mathbf{s}_j} + B_i^{\mathbf{s}_j} \ldots + L_i^{\mathbf{s}_j}\right),$$

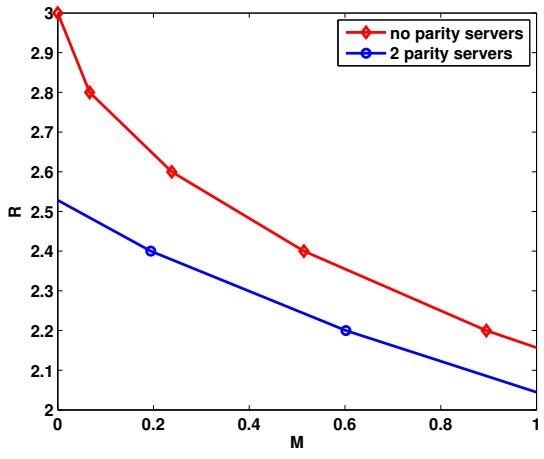

FIGURE 3: Comparison of the performance between multi-server system without parity servers and the system with two parity servers.

with arbitrary coefficients λ_{ij} for any user set $\mathbf{s}_j$. This corresponds to a linear combination of all the segments in Eq. (3). Similarly, parity server Q can transmit some other linear combinations of the segments which can also work as components of an MDS code. This effectively increases the size of the cache memories by M' file units, corresponding to the amount of information that the parity servers can afford to send each user during the delivery phase.

Theorem 5.2: If there are η parity servers and $K \geq N$, the following (M, R) pairs are achievable for $t = 0, 1, \ldots, K$

$$\left(\frac{t\left[(N-1)t + K - N\right]}{K(K-1)} - \eta \frac{N(K-t)}{LK^2}, \frac{N(K-t)}{LK} \right).$$

Proof: The information sent by the parity server is bounded by the peak rate of the data servers, *i.e.*, $\frac{N(K-t)}{LK}$ according to Eq. (11). Assuming a worst case scenario, each transmission from a parity server will benefit a single user. Therefore, each parity server can effectively increase the cache of each user by $M' = \frac{N(K-t)}{LK^2}$. ■

This memory sharing strategy provides significant improvement when the cache capacity is small. Fig. 3 shows the performance for $K = 15$ users and $N = 12$ files stored in $L = 4$ data servers. When the cache size is small, the peak rate of the system with two parity servers is much lower than that without parity servers. As the cache grows the advantage of the system with parity servers becomes less clear.

The interference elimination scheme is specially designed for the case with less files than users ($N \leq K$) in the single server system. However, since the peak load is reduced by $\frac{1}{L}$ in a multi-server system, the interference elimination scheme might also have good performance when $N > K$ if L is large. In order to apply the algorithm, we can just add $N-K$ dummy users with arbitrary requests. Then, we have the following corollary from Theorem 5.2:

Corollary 5.2.1: If there are η parity servers and $K \leq N$, the following (M, R) pairs are achievable:

$$\left(\frac{t^2}{N} - \eta \frac{(N-t)}{LN}, \frac{(N-t)}{L} \right), \qquad t = 0, 1, \ldots, N.$$

VI. Gap to Optimality

This section bounds the gap to optimality for the algorithms proposed in Sections III and IV, whose performance is summarized in Table VII. Let $R_C^\star(M)$ denote the minimum peak rate in a single server system with cache sizes of size M. This paper considers a system with L data and L' parity servers that can transmit in parallel; let $R_{L,L'}^\star(M)$ denote the corresponding minimum peak rate. This section will first show that the stripping scheme proposed in Section III provides a peak rate within a factor of 12 from $R_{L,L'}^\star(M)$. Then it will show that, when files must be stored as a single unit in a server, the algorithm proposed in Section IV is within a factor of $12L$ from $R_{L,L'}^\star(M)$.

Maddah-Ali and Niesen used a cut-set argument to derive a lower bound on the peak rate of a system with a single server, and proved that the peak rate with their scheme was within a constant factor of optimal [6]. Tighter bounds have been found in some cases [7], [16] but, to the extent of our knowledge, the general expression for the minimum peak rate is still unknown.

Theorem 6.1: The peak load with M-N scheme in Eq. (2) is within a factor of 12 from optimal.

Proof: The proof was shown in [6]. ■

An intuitive relationship between $R_{L,L'}^\star(M)$ and $R_C^\star(M)$ is shown in the following lemma.

Lemma 6.2: In a distributed storage system with L data servers storing different files, L' parity servers, and a number of users smaller than the number of files per server ($K < N/L$)

$$R_{L,L'}^\star(M) \geq \frac{1}{L+L'} R_C^\star(M) \tag{12}$$

Proof: With $L + L'$ parallel channels, the peak load per link can be reduced to $\frac{1}{L+L'} R_C^\star(M)$ in the best case, *i.e.*, when load is perfectly balanced among all channels without redundancy. ■

Theorem 6.3: In a distributed storage system with L data servers storing different content and L' parity servers, the peak load derived by the stripping scheme as shown in Section III is within a factor of 12 from the optimal $R_{L,L'}^\star(M)$.

Proof: Section III showed that the stripping scheme balances the load by transmitting messages in parallel and reduces the peak load to $\frac{1}{L+L'}$ of that with M-N scheme ((Eq. 2)). Therefore, the stripping scheme is also within a constant factor of optimal and Theorem 6.3 follows from Theorem 6.1 and Lemma 6.2. ■

When files are stored whole in one server, we are limited in terms of the segments that can be combined into a transmitted message.

Theorem 6.4: In a distributed storage system with L data servers storing different files and L' parity servers, the peak load provided by Scheme 1 in Section IV is within a factor of $12L$ from optimal.

Server system	Normalized peak rate	Comments
single server	$R_C(K,t) = \binom{K}{t+1}/\binom{K}{t}$	$t = \frac{KM}{N}$ $\Delta \leq \frac{1}{3}$
L data 1 parity	$\frac{L-1}{L} R_C(K,t)$	
L data 2 parity	$(\frac{1}{2} + \frac{L-2}{2L+4}\Delta) R_C(K,t)$	
L data L' parity	$\frac{L}{L+L'} R_C(K,t)$	Load balancing

TABLE VII: Normalized peak rate of Scheme 1.

Proof: The peak loads for the L data servers and L' parity servers are shown in Table VII. Since $\frac{L-1}{L} \leq \frac{L}{L+L'}$ for $L' = 1$ and $\frac{1}{2} + \frac{L-2}{3(2L+4)} \leq \frac{L}{L+L'}$ for $L' = 2$, we can conclude that Scheme 1 always provides a peak rate below $\frac{L}{L+L'}$ of that of M-N scheme. It is therefore less than $12\frac{L}{L+L'} R^{\star}_C(M)$ according to Theorem 6.1 and finally less than $(12L) R^{\star}_{L,L'}(M)$ according to Lemma 6.2. ■

The gap to optimality for Scheme 2, proposed in Section V, can be unbounded. For example, the optimal peak rate must always be lower than the number of users K, since we can always ignore the cached data and transmit all the requested files. However, Table VIII shows that when the number of files N is greater than the number of users K, the rate of Scheme 2 approaches N as the number of files grows. The gap between K and N can be arbitrarily large.

VII. Simulations

This section compares all the schemes studied in this paper, for a system with $N = 20$ files stored in $L = 4$ data servers with 5 files each. We show that striping has better performance than the schemes in sections IV and V (Scheme 1 and Scheme 2, respectively) at the cost of network flexibility. If each file is stored as a single block in one server, Scheme 2 has better performance when the cache capacity is small while Scheme 1 is more suitable for the case where the cache capacity is large. The performances of Scheme 1 and Scheme 2 are summarized in Table VII and Table VIII, respectively.

Fig. 4 and Fig. 5 focus on the case with one and two parity servers, respectively. We assume that there are $K = 15$ users, thus there are more files than users, with varying cache capacity. We observe that striping provides lower peak rates than storing whole files, as expected. Additionally, since $N > K$, the interference elimination scheme always has worse performance than M-N scheme when striping is used. Without striping, Scheme 2 provides lower peak rate than Scheme 1 when the cache capacity is small, and it is the other way around when the capacity is large.

Then Fig. 6 and Fig. 7 compare the performance between Scheme 1 and Scheme 2 when there are more users $(K = 60)$ than files for the one or two parity case, respectively. Striping still provides lower peak rate than storing whole files but the two striping curves cross. Interference Elimination is now better than M-N scheme when the cache capacity is small. Similarly, Scheme 2 provides lower peak rate than Scheme 1 for small caches, and the positions are reversed when the cache capacity increases. Moreover, we notice that the curves intersect at a point with larger M than they did in Fig. 4 and Fig. 5, which means that we are more prone to utilize Scheme 2 when there are more users than files.

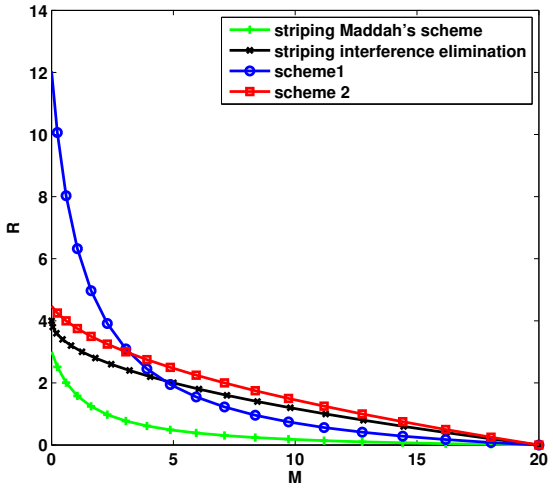

FIGURE 4: Comparison between the performance between Scheme 1 and Scheme 2 in one parity server system when $N = 20$ and $K = 15$.

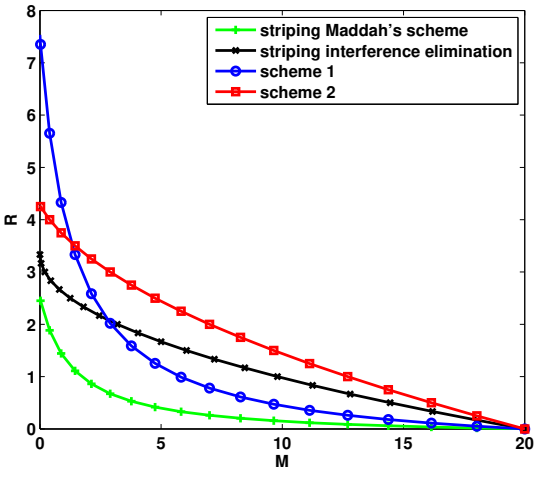

FIGURE 5: Comparison between the performance between Scheme 1 and Scheme 2 in two parity server system when $N = 20$ and $K = 15$.

VIII. Conclusion

This paper proposes coded caching algorithms for reducing the peak data rate in multi-server systems with distributed storage and different levels of redundancy. It shows that, by striping each file across multiple servers, the peak rate can be reduced proportionally to the number of servers. Then it addresses the case where each file is required to be stored as a single block in one server and proposes different caching and delivery schemes depending on the size of the cache memories.

Distributed storage systems generally use MDS codes across the servers to protect the information against node failures.

Server system	Normalized (M,R)	Comments
single server	$\left(\frac{t[(N-1)t+K-N]}{K(K-1)}, \frac{N(K-t)}{K}\right)$	$0 \leq t \leq N$
L data η parity ($K \geq N$)	$\left(\frac{t[(N-1)t+K-N]}{K(K-1)} - \eta\frac{N(K-t)}{LK^2}, \frac{N(K-t)}{LK}\right)$	
L data η parity ($K \leq N$)	$\left(\frac{t^2}{N} - \eta\frac{(N-t)}{LN}, \frac{(N-t)}{L}\right)$	

TABLE VIII: Normalized (M,R) pair of Scheme 2.

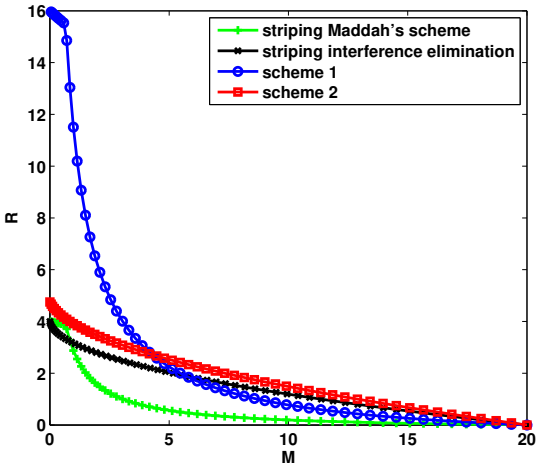

FIGURE 6: Comparison between the performance between scheme 1 and scheme 2 in one parity server system when $N = 20$ and $K = 60$.

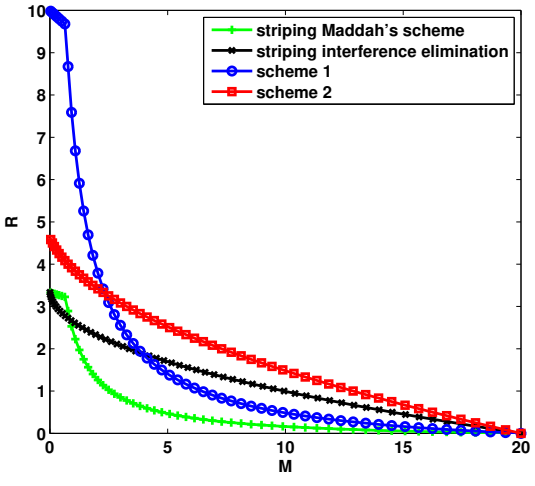

FIGURE 7: Comparison between the performance between scheme 1 and scheme 2 in two parity server system when $N = 20$ and $K = 60$.

The coded caching schemes proposed in this paper are able to leverage that redundancy in creative ways to reduce the achievable traffic peak rate. The results for the proposed schemes are shown in Table VII and Table VIII.

In the future, we will study how this process can be generalized to other erasure codes, such as fractional repetition codes [27][28] or other RAID-6 [30] structures. We also plan to generalize our schemes to the case where files have different popularity, which will require designing erasure codes with different levels of protection for different files. Other interesting research directions that have appeared after the first submission of this manuscript are multi-server systems with random topology [31], refining our pairing algorithm [32], and expanding the performance metric to include the input/output (I/O) load at the servers [33].

APPENDIX

In this appendix, we will elaborate on the pairing scheme in Lemma 4.3 from Section IV-B , specially for the case with even K and symmetric requests.

Definition A.1: Let χ_A denote a set of messages (or, equivalently, subsets of $t+1$ users) to be sent by server A and χ_B denote a set of messages to be sent by server B. If there is an injective function providing each element in χ_A with an effective pair in χ_B, we say that there is a **saturating matching** for χ_A.

In order to reduce the peak rate we want to separate all the messages to be transmitted (equivalently, subsets of $t+1$ users) into two groups χ_A and χ_B such that there are as many effective pairs as possible, as we shall see.

To better illustrate the allocation scheme, the problem of finding effective pairs is mapped to a graph problem. Let G be a finite bipartite graph with bipartite sets χ_A and χ_B, where each message (or user subset) is represented as a vertex in the graph and edges connect effective pairs from χ_A and χ_B. The idea of our design is to allocate as many messages as possible to χ_A, while guaranteeing the existence of a saturating matching for χ_A based on Hall's marriage Theorem [34].

Theorem A.1: (Hall's Marriage Theorem [34]) Let G be a finite bipartite graph with bipartite sets χ_A and χ_B. For a set $\mathbf{u}$ of vertices in χ_A, let $N_G(\mathbf{u})$ denote its neighbourhood in G, *i.e.*, the set of all vertices in χ_B adjacent to some element of $\mathbf{u}$. There is a matching that entirely covers χ_A if and only if

$$|\mathbf{u}| \leq |N_G(\mathbf{u})|$$

for every subset $\mathbf{u}$ of χ_A.

Corollary A.1.1: If all vertices in χ_A have the same degree d_A and all the vertices in χ_B have the same degree d_B ($d_A \geq d_B$), then there is a saturating matching for χ_A.

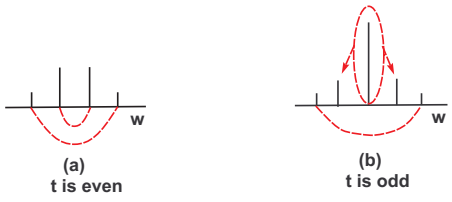

FIGURE 8: Pairing illustration. w is the number of files from server A in a message.

Proof: For any $\mathbf{u} \subseteq \chi_A$, all edges connected to $\mathbf{u}$ are also connected to $N_G(\mathbf{u})$, hence $|N_G(\mathbf{u})| \cdot d_B \geq |\mathbf{u}| \cdot d_A$. Since $d_A \geq d_B$, we know that $|\mathbf{u}| \leq |N_G(\mathbf{u})|$. According to Theorem A.1, there is a saturating matching for χ_A. ■

In order to compute the peak rate in the worst case, we assume that all K users request different files. Since each subset contains $t+1$ files, there are $\binom{K}{t+1}$ messages to allocate between χ_A and χ_B. We classify these subsets according to the number of requests from server A: sets of type w will have w requests from server A and $t+1-w$ from server B. The following proposition states that the messages of the same type are not able to pair with each other.

When t is even and the demands are symmetric, type w sets and type $t+1-w$ sets form a symmetric bipartite graph, so there exists a saturating matching according to Corollary A.1.1. When t is odd, type $(t+1)/2$ sets are paired with the union of type $(t-1)/2$ sets and type $(t+3)/2$ sets. Since the vertices in type $(t-1)/2$ sets and type $(t+3)/2$ sets are connected to the same number of vertices in type $(t+1)/2$ sets, this bipartite graph also fulfills the condition in Corollary A.1.1. Other sets are paired as in the case with t even, that is, type w sets are paired with type $t+1-w$ sets. These pairings are illustrated in Fig.8.

When t is even, there is a matching for every candidate file set, thus the peak rate is cut by half compared with the traditional single server scheme. When t is odd, some vertices of types $(t-1)/2$, $(t+1)/2$, or $(t+3)/2$ could fail to be paired. Denote the ratio of unpaired messages when t is odd by Δ. Any two servers can collaborate to fulfill those requests, so the normalized overall peak rate R_T with symmetric demands is given by:

$$R_T(K,t) = \begin{cases} \frac{1}{2} R_C(K,t) & \text{if } t \text{ is even} \\ \left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t) & \text{if } t \text{ is odd,} \end{cases}$$

The pairing loss Δ is limited. The worst case occurs when there is a big difference between the number of vertices of type $(t+1)/2$ and the number of vertices of types $(t-1)/2$ or $(t+3)/2$. In both cases, the pairing loss Δ is bounded by $\frac{1}{3}$.

REFERENCES

[1] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," IEEE Trans. Inf. Theory, vol. 56, no. 9, pp. 4539–4551, 2010.

[2] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in FAST-2004: 3rd Usenix Conference on File and Storage Technologies, 2004.

[3] J. S. Plank, "The RAID-6 liber8tion code," International Journal of High Performance Computing Applications, 2009.

[4] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster," in Cluster Computing (CLUSTER), 2010 IEEE International Conference on. IEEE, 2010, pp. 188–196.

[5] C.-F. Chou, L. Golubchik, and J. C. Lui, "Striping doesn't scale: How to achieve scalability for continuous media servers with replication," in Distributed Computing Systems, 2000. Proceedings. 20th International Conference on. IEEE, 2000, pp. 64–71.

[6] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," IEEE Trans. Inf. Theory, vol. 60, no. 5, pp. 2856–2867, 2014.

[7] H. Ghasemi and A. Ramamoorthy, "Improved lower bounds for coded caching," IEEE Transactions on Information Theory, vol. 63, no. 7, pp. 4388–4413, 2017.

[8] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," IEEE Trans. Inf. Theory, vol. 63, no. 2, pp. 1146–1158, 2017.

[9] J. Zhang, X. Lin, C.-C. Wang, and X. Wang, "Coded caching for files with distinct file sizes," in IEEE Int. Symp. on Information Theory (ISIT). IEEE, 2015, pp. 1686–1690.

[10] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Order-optimal rate of caching and coded multicasting with random demands," IEEE Trans. Inf. Theory, vol. 63, no. 6, pp. 3923–3949, 2017.

[11] ——, "Caching and coded multicasting: Multiple groupcast index coding," in IEEE Global Conf. on Signal and Information Processing (GlobalSIP). IEEE, 2014, pp. 881–885.

[12] M. Ji, A. Tulino, J. Llorca, and G. Caire, "Caching-aided coded multicasting with multiple random requests," in IEEE Information Theory Workshop (ITW). IEEE, 2015, pp. 1–5.

[13] J. Hachem, N. Karamchandani, and S. Diggavi, "Effect of number of users in multi-level coded caching," in IEEE Int. Symp. on Information Theory (ISIT). IEEE, 2015, pp. 1701–1705.

[14] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," IEEE/ACM Trans. Netw (TON), vol. 23, no. 4, pp. 1029–1040, 2015.

[15] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," arXiv preprint arXiv:1503.00265, 2015.

[16] C. Tian and J. Chen, "Caching and delivery via interference elimination," arXiv preprint arXiv:1604.08600, 2016.

[17] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," IEEE Trans. Inf. Theory, vol. 56, no. 7, pp. 3187–3195, 2010.

[18] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," IEEE Trans. Inf. Theory, vol. 57, no. 3, pp. 1479–1494, 2011.

[19] M. A. R. Chaudhry and A. Sprintson, "Efficient algorithms for index coding," in IEEE INFOCOM Workshops. IEEE, 2008, pp. 1–4.

[20] C. Thapa, L. Ong, S. J. Johnson, and M. Li, "Structural characteristics of two-sender index coding," arXiv preprint arXiv:1711.08150, 2017.

[21] B. S. Rajan et al., "Optimal scalar linear index codes for three classes of two-sender unicast index coding problem," arXiv preprint arXiv:1804.03823, 2018.

[22] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," IEEE Communications Magazine, vol. 51, no. 4, pp. 142–149, 2013.

[23] H. Ghasemi and A. Ramamoorthy, "Improved lower bounds for coded caching," in IEEE Int. Symp. on Information Theory (ISIT). IEEE, 2015, pp. 1696–1700.

[24] R. Blom, "An optimal class of symmetric key generation systems," in Workshop on the Theory and Application of of Cryptographic Techniques. Springer, 1984, pp. 335–338.

[25] C. Suh and K. Ramchandran, "Exact-repair MDS code construction using interference alignment," IEEE Trans. Inf. Theory, vol. 57, no. 3, pp. 1425–1442, 2011.

[26] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, Comparing random data allocation and data striping in multimedia servers. ACM, 2000, vol. 28, no. 1.

[27] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in 48th Annual Allerton Conference on Communication, Control, and Computing. IEEE, 2010, pp. 1510–1517.

[28] Q. Yu, C. W. Sung, and T. H. Chan, "Irregular fractional repetition code optimization for heterogeneous cloud storage," IEEE J. Sel. Areas Commun., vol. 32, no. 5, pp. 1048–1060, 2014.

[29] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," ACM Transactions on Storage (TOS), vol. 9, no. 1, p. 3, 2013.
[30] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," IEEE J. Sel. Areas Commun., vol. 32, no. 5, pp. 1008–1018, 2014.
[31] N. Mital, D. Gunduz, and C. Ling, "Coded caching in a multi-server system with random topology," arXiv preprint arXiv:1712.00649, 2017.
[32] M. Cheng, Q. Zhang, and J. Jiang, "Improved rate for a multi-server coded caching," arXiv preprint arXiv:1802.07410, 2018.
[33] T. Luo and B. Peleato, "The transfer load-i/o trade-off for coded caching," IEEE Communications Letters, to appear.
[34] P. Hall, "On representatives of subsets," J. London Math. Soc, vol. 10, no. 1, pp. 26–30, 1935.

Tianqiong Luo received the B.S degree in Communication Engineering from Fudan University, Shanghai, China in 2013 and the Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, USA in 2018. Dr. Luo is currently working at Google Inc, Mountain View, CA. Her research interests involve signal processing and coding for non-volatile storage, caching and storage system.

Vaneet Aggarwal (S'08 - M'11 - SM'15) received the B.Tech. degree in 2005 from the Indian Institute of Technology, Kanpur, India, and the M.A. and Ph.D. degrees in 2007 and 2010, respectively from Princeton University, Princeton, NJ, USA, all in Electrical Engineering.

He is currently an Associate Professor at Purdue University, West Lafayette, IN, where he has been since Jan 2015. He was a Senior Member of Technical Staff Research at AT&T Labs-Research, NJ (2010-2014), Adjunct Assistant Professor at Columbia University, NY (2013-2014), and VAJRA Adjunct Professor at IISc Bangalore (2018-2019). His current research interests are in communications and networking, cloud computing, and machine learning.

Dr. Aggarwal was the recipient of Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, 2017 Jack Neubauer Memorial Award recognizing the Best Systems Paper published in the IEEE Transactions on Vehicular Technology, and 2018 Infocom Workshop Best Paper Award. He is serving on the editorial board of the IEEE Transactions on Communications and the IEEE Transactions on Green Communications and Networking.

Borja Peleato (S'12–M'13) received the B.S. degrees in Telecommunications and Mathematics from the Universitat Politecnica de Catalunya, Barcelona, Spain, in 2007, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, USA, in 2009 and 2013, respectively. In January 2014, he joined the Electrical and Computer Engineering Department at Purdue University, West Lafayette, IN, USA, where he is currently an Assistant Professor. He was a visiting student at the Massachusetts Institute of Technology in 2006 and a Senior Flash Channel Architect with Proton Digital Systems in 2013. His research interests include wireless communications, signal processing for nonvolatile storage, and convex optimization. He was the recipient of the "La Caixa" Graduate Fellowship in 2006.