Service Rate Region: A New Aspect of Coded Distributed System Design

Mehmet Aktaş, Gauri Joshi, *Member, IEEE*, Swanand Kadhe, *Member, IEEE*, Fatemeh Kazemi, *Student Member, IEEE*, and Emina Soljanin, *Fellow, IEEE*

***Abstract*— Erasure coding has been recognized as a powerful method to mitigate delays due to slow or straggling nodes in distributed systems. This work shows that erasure coding of data objects can flexibly handle skews in the request rates. Coding can help boost the *service rate region*, that is, increase the overall volume of data access requests that the system can handle. This paper aims to postulate the service rate region as an important consideration in the design of erasure-coded distributed systems. We highlight several open problems that can be grouped into two broad threads: 1) characterizing the service rate region of a given code and finding the optimal request allocation, and 2) designing the underlying erasure code for a given service rate region. As contributions along the first thread, we find the rate regions of maximum-distance-separable, locally repairable, and simplex codes. We show the effectiveness of hybrid codes that combine replication and erasure coding in terms of code design. We also discover fundamental connections between multi-set batch codes and the problem of maximizing the service rate region.**

***Index Terms*— Erasure coded storage, coded computing, resource allocation, distributed systems, batch codes.**

I. Introduction

THE emergence of flexible and affordable cloud storage and computing has resulted in an exponential growth in the amount of data stored and processed in cloud data centers. This increase in data is accompanied by a similar rapid increase in the volume of users accessing it, resulting in frequent contention for shared cloud resources. A simple way to handle more requests in a fast and reliable fashion is to replicate data at multiple nodes [2], [3]. However, replication can be expensive in terms of storage, especially when the data is updated frequently. Moreover, the popularity of different data objects can vary drastically across objects and over time. While edge caches can handle skews in popularity by selectively increasing the number of replicas of the 'hot' or popular objects [4]–[9], such quick adaptation may not be possible in the data-center setting, especially for large data objects that are used in data analytics or machine learning applications. Besides, in caching, the backhaul link's limited capacity is considered the main bottleneck of the system, and the goal is usually to minimize the backhaul traffic or maximize the cache hit rate by prefetching the popular contents at the edge nodes of limited storage capacity. However, caching does not aim to handle scenarios such as live streaming where many users want to get the same content simultaneously, given the network's limited service capacity (bandwidth).

In this work, we propose the use of erasure coding to handle data access requests to distributed storage systems. We consider coded distributed systems where k different data objects (rather than k chunks of one object) are erasure coded into n coded objects which are stored on n nodes. We consider *heterogeneous* requests to access these objects at rates $\lambda_1, \lambda_2, \ldots, \lambda_k$, respectively. Each of the n nodes can serve at most μ rate of requests. Thus, the total request rate allocated to each node must not exceed μ. Under these constraints, we aim to characterize the set of achievable vectors $(\lambda_1, \lambda_2, \ldots, \lambda_k)$, which we refer to as the *service rate region* of a coded distributed system. Since the nodes storing coded objects can be used to partially serve requests for any of the objects included in that coded combination, coded distributed systems are more flexible and can have a different (possibly more favorable) service rate region than an uncoded system with the same number of nodes. We illustrate this through the motivating example below.

A. Motivating Example

Consider an example shown in Figure 1, where two objects a and b are redundantly stored on 4 nodes. Figure 1 (left) shows 3 redundant storage schemes: replication, coding, and replication and coding combined. Given that each node can serve $\mu = 1$ request per second, we want to maximize

Manuscript received September 3, 2020; revised June 22, 2021; accepted September 16, 2021. Date of publication October 4, 2021; date of current version November 22, 2021. This work was supported in part by the National Science Foundation (NSF) CAREER Award under Grant CCF-2045694 and in part by NSF under Grant CIF-1717314. An earlier version of this paper was presented in part at the 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton) [DOI: 10.1109/ALLERTON.2017.8262713]. *(Corresponding author: Fatemeh Kazemi.)*

Mehmet Aktaş is with the Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey (e-mail: mfatihaktas@gmail.com).

Gauri Joshi is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: gaurij@cmu.edu).

Swanand Kadhe is with the Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, CA 94720 USA (e-mail: swnanand.kadhe@berkeley.edu).

Fatemeh Kazemi is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: fatemeh.kazemi@tamu.edu).

Emina Soljanin is with the Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08854 USA (e-mail: emina.soljanin@rutgers.edu).

Communicated by C. Hollanti, Associate Editor for Coding Theory.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2021.3117695.

Digital Object Identifier 10.1109/TIT.2021.3117695

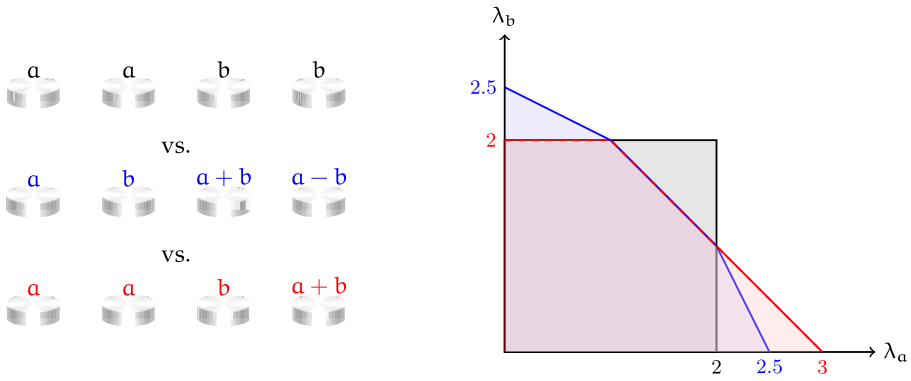

Fig. 1. (left) Replicated, coded, and hybrid systems with $n = 4$ nodes storing $k = 2$ files. (right) Service rate regions of the three systems when the service capacity of each node is $\mu = 1$. The regions have the same areas. Coding can handle the skews in request arrival rates λ_a and λ_b for the two stored objects.

λ_a and λ_b, the rate of requests for a and b that can be supported. Object a can be downloaded from the node storing a, or from two nodes that store coded combinations of a and b.

Figure 1 (right) shows the service rate regions of the 3 storage systems. The replicated system can achieve the square service rate region with $0 \leq \lambda_a, \lambda_b \leq 2$; this is because there are two copies of each object and each node can support $\mu = 1$ rate of requests. The coded system with two nodes storing $a+b$ and $a-b$ respectively instead of uncoded copies of a and b achieves the blue colored shaded service rate region. This system can handle skews in λ_a and λ_b better than the replicated system when one of the two objects a and b are more frequently accessed but both objects are unlikely to be popular simultaneously. The service rate region of a combined replication and coding system (shown in red) can better support asymmetries in the demands λ_a and λ_b and is the best choice when the request rate for a is expected to be larger than that of b.

B. Related Previous Work

Erasure codes are often used in distributed storage systems to improve reliability against disk failures [2]. A class of codes, which are commonly used in distributed storage, are systematic maximum-distance-separable (MDS) codes [10], where an object is divided into k chunks (also called stripes) that are then encoded into n chunks by adding $n-k$ redundant parity-check chunks, thus providing resilience to the failure of up to $n-k$ nodes. Until recently, erasure coding in storage systems was mostly used for 'cold' or less frequently accessed and less latency-sensitive data. This is because, erasure coded systems require access to k nodes (each storing one of the k chunks of an object) in order to download the object, and slowdown of any one of these nodes can become a bottleneck in serving data access requests. Thus, replication is generally preferred over erasure coding for hot and latency-sensitive data access. Recently, the idea of redundant data access requests, that is, sending requests to all n nodes of an erasure-coded system and waiting for any k nodes to respond, has been shown to be effective in overcoming such tail latency due to straggling nodes [11]–[15]. Similar redundancy ideas are also used in the context of distributed computing [16]–[21]. However, most of these works on faster data access from coded distributed systems focus on homogeneous reads, where all k chunks of an object are accessed at the same time.

Heterogeneous data access has been previously considered in the context of hot data download (see e.g., [22]–[25]) and load balancing (see e.g., [26]). In the hot data download context, heterogeneity arises when one of the data objects is highly popular. Works such as [22]–[25] analyze the expected latency experienced by requests that are replicated across the hot object's recovery sets. Previous works on load balancing heterogeneous requests to coded systems that arises when a batch of simultaneous requests consists of different numbers of requests for each of the k objects. Special coding schemes, known as multi-set batch codes [26], have been proposed to allow serving such requests with a balanced amount of downloaded data across the servers (see e.g. [27], [28] and references therein). In this paper, we do not impose any limits on data access heterogeneity, that is, on the arrival rates λ_1, λ_2, …, λ_k. Our main focus is the service rate region, that is, the set of request arrival rates that the system can support; we discuss connections to download latency and load balancing in Sec. VIII.

The main difference between this and, nearly all, recent work on coded distributed storage is that the proposed work primarily addresses the external uncertainty in the storage systems (download requests fluctuations) rather than the internal uncertainty (e.g., straggling) in operations of the system itself. A related line of work by some of the authors (addressing external uncertainty) considers systems with uncertainty in the mode and level of access to the system [29]–[31].

C. Goals and Organization of this Paper

The main goal of this paper is to propose the service rate region as an important paradigm in the design of erasure coded distributed systems. Characterizing service rate region gives us a clear picture of the collective rate of requests that can be supported by the system as well as its robustness to heterogeneous request patterns where some objects are more frequently accessed than others. In this paper we highlight two main threads of ongoing and future research directions that explore different aspects of the service rate region of

coded distributed systems: 1) designing optimal policies to split incoming requests across the nodes in order to maximize the achievable service rate region for a given storage scheme, and 2) designing the underlying code to maximize the service rate region or to cover a given region with minimum storage, as introduced in Section III.

The first problem of optimal request splitting can be formulated as a constrained optimization problem. However, it cannot be trivially solved using linear solvers because the number of optimization variables is large and the problem becomes computationally intractable. As contributions along this thread, we characterize the rate region of some well-known classes of codes such as maximum-distance-separable (MDS), locally recoverable (LRC), Simplex (also called Hadamard) codes, and first-order Reed-Muller (RM) codes. These analyses provide insights into how the service rate region is affected by the length and the rate of the underlying code. We can probably find the best service rate region for certain classes of codes such as MDS codes and Simplex codes, but finding the optimal request splitting scheme for other code classes still has many open questions. We highlight three different techniques to solve the problem of optimal request splitting to maximize the service rate region: 1) Section V uses a waterfilling algorithm to find the rate region of MDS and LRC codes, 2) Section VI uses fractional matching and vertex cover on graph representation of codes (which we introduce in Section VI-B) to find the rate region of Simplex codes, and 3) the geometric approach used in Section VII to find the rate region of first-order RM codes. Along the second thread of designing the underlying code, in Section III we highlight the complementary problems of maximizing the rate region of a given number of servers and covering a desired rate region using a minimum number of nodes. The key insight from this exploration is that hybrid codes that carefully combine replication and erasure coding of data (such as the (a, a, b, $a+b$) system considered in the motivating example above) are best-suited for maximizing the service rate region in many cases.

A crucial goal of this paper is to provide a comprehensive list of open problems in connection with this emerging idea of using the service rate region to guide the design and analysis of erasure coded systems. These problems are of interest to the information and coding theory, (combinatorial) optimization, as well as the queueing/networking communities and can bridge interdisciplinary connections between them. In Section VIII we discuss specific problems such as service rate region considerations in the design of codes, designing codes that cover a given request rate distribution, latency analysis of erasure coded systems, service rate region with redundant requests. In Section VI-E we discover fundamental connections between batch codes and the problem of maximizing the service rate region. In fact, codes that maximize the service rate region are a generalization of primitive multi-set batch codes where the demands λ_a and λ_b of different objects are not constrained to be integers.

Several problems presented in this paper not only require expertise from different areas, but have also already been addressed in those areas in some special forms and under different names. We will explain how some problems associated with the service rate region generalize some previously studied problems. We will also present several problems that belong to but have not been asked yet in certain areas, and thus experts in those areas could potentially provide answers with not too much difficulty.

D. How to Read This Paper

Depending on the reader's main interest and prior knowledge, different sections of paper may or may not be relevant. Sections II to IV should be read by everyone since they provide the system model, problem formulation, some preliminary notions as well as several examples running throughout the paper. Information theorists and queueing theorists may be primarily interested in Section V, theoretical computer scientist in Section VI, and coding theorists in Section VII. Each of the Sections V to VII can be read immediately after the introductory Sections II to IV, and independently of the other two. Similarly, the readers can select open problems from the large list in Section VIII according to their interests and expertise – this section includes performance analysis and networking problems as well as coding theory and data allocation problems.

II. Distributed Service Model

We distinguish between two functional components at each node: one for data storage and the other for service request processing. This is indicated in Figure 2 illustrating a system of $n = 7$ servers.

A. Data Storage Model

Consider that we have k data objects (to be) redundantly stored across $n \geq k$ servers. We assume all data objects are of the same size, and all servers have a storage capacity of one object. Mathematically objects are represented as elements of some finite field $\mathbb{F}_q$. Each server can store a linear combination of data objects, which amounts to a coded object of the same size. We assume that the same erasure code is used for all the objects in the system. Simple replication of objects, that is, storing identical copies, is allowed and, as we will see later, often to a certain extent desirable. It is worth to note here that the assumptions we make in our storage model are common in the prior work, see, e.g., [13], [14], [24], [25]. We denote the coded objects as $c_1, c_2, \ldots, c_n$. Because of redundancy, any data object can be recovered (computed) from multiple sets of encoded objects.

Definition 1: A recovery set for a coded object $c_i \in \mathbb{F}_q$ is a minimal set of coded objects R such that there exists a recovery function $rc : \mathbb{F}_q^{|R|} \rightarrow \mathbb{F}_q$ satisfying $rc(R) = c_i$.

Each object has at least one recovery set (the object itself), and may potentially have multiple recovery sets. We denote by $R_{i,1}, \ldots, R_{i,t_i}$ the t_i recovery sets of object i. Figure 2 shows a system where three data objects a, b and c are encoded by a $[7, 3]$ binary Simplex code. The recovery sets for object a are its systematic copy (a) and the pairs of linear combinations $(b, a+b)$, $(c, a+c)$ and $(b+c, a+b+c)$. For a systematic MDS

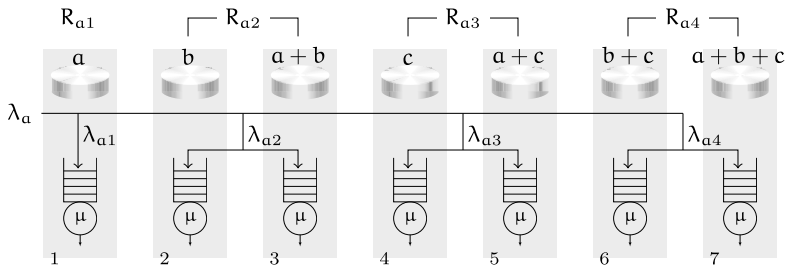

Fig. 2. A distributed system storing $k = 3$ data objects a, b, c over $n = 7$ nodes. Each node stores a symbol of the $[7, 3]$ Simplex code. R_{a_i}'s represent the recovery sets for a, and λ_{a_i} denotes the portion of requests rate for serving a that is assigned to R_{a_i} such that $\lambda_a = \lambda_{a_1} + \lambda_{a_2} + \lambda_{a_3} + \lambda_{a_4}$ holds.

code, the recovery sets for each object are its uncoded copy and any size-k subset of the remaining $n-1$ nodes, and thus the number of recovery sets for each data object is $1 + \binom{n-1}{k}$. For instance for the $[4, 2]$ MDS coded system shown in blue in Figure 1, the recovery sets for object a are (a), $(a+b, a-b)$, $(b, a+b)$ and $(b, a-b)$.

B. Data Access Model

We describe two data access models which are two different ways of implementing resource sharing among the incoming data access requests. We refer to them as the queuing and the bandwidth model. Both of these access models result in similar mathematical formulations of the service rate region, which we define in Section III. In both models, we assume that requests to download object i arrive at rate λ_i, and that the service rate at each server is μ requests per unit time.

1) Queueing Model: Requests sent to each server are placed in a queue at the server (the queue can follow either first-come-first-served or any other scheduling discipline). In order to maintain the stability of the queue at each server, the total request arrival rate at each server should not exceed its service rate μ. Our goal is to characterize the service rate region, that is, the set of arrival rates $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ for the k objects that can be supported by the system.

2) Bandwidth Model: Suppose that storage drives associated with each node can concurrently serve only a limited number of data access requests. This is because each drive has an I/O bus with a finite access bandwidth W bits/second, and a download request requires streaming at a fixed bandwidth of b bits/second. Therefore, a node can serve only $\mu = W/b$ number of requests concurrently. Let λ_i be the number of requests for file i that are simultaneously present in the system. Our goal is to characterize all request combinations for the k data objects $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ that can simultaneously be served by the system. Note that unlike the queuing model, λ_i's are integers in this case. In Section VI-E we define the notion of the integral service rate region for this model and show how it is fundamentally connected to batch codes [26].

C. Extension to Coded Computing

We present the queueing and bandwidth models in the context of data-access, but these models and the resulting formulation of the service rate region can be directly applied to determine the service rate region of coded computing systems as well. Consider, for example, a system of $n = 4$ servers storing large matrices A, B, $A + B$ and $A + 2B$ respectively. Each request in the rate λ_A has a query vector x and its goal is to obtain a matrix-vector product Ax, whereas the requests in λ_B seek to compute Bx. The task of computing Ax can be completed either by sending x to the server storing A, or by sending it to any two out of the three remaining servers. In the queueing model, requests are placed in a queue at each server with service rate μ, whereas in the bandwidth model, μ is the maximum number of tasks that each server can handle simultaneously.

III. Two Problems of Service Rate Regions

In this section, we describe two classes of problems that arise in the context of using the service rate region as a metric to design erasure-coded distributed systems. These problems are 1) maximizing the service rate region of a given storage scheme via optimal resource allocation and 2) designing a storage-efficient erasure-coded scheme to cover or achieve a desired service rate region. This section aims to formulate these problems and highlight the variety in the mathematical techniques applicable to service rate region problems. These techniques bridge deep connections between fundamental coding theory and resource allocation problems. In subsequent sections, we mainly address the first problem described above: maximizing the service rate region of a given storage allocation.

A. Finding the Service Rate Region of a Storage Scheme

Given distributed system with k data objects stored on n servers, we first present the problem of maximizing the service rate region by optimally splitting incoming request rates $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ across the n servers. The problem of optimally allocating incoming data access requests to one or more servers can be formulated as a linear optimization problem, which we describe below.

Recall that each server has the service rate μ, that is, it can serve μ requests in unit time. Let us use $\lambda_{i,j}$ to denote the portion of requests for object i that is assigned to the recovery set $R_{i,j}$, $j = 1, \ldots, t_i$.[1] Without loss of generality, suppose that the first $k-1$ elements of the demand vector $\boldsymbol{\lambda}$ are given

[1]Note that we do not make any assumptions on the arrival process such as Poisson arrivals.

TABLE I
SUMMARY OF THE CODING SCHEMES AND THE TECHNIQUES USED TO CHARACTERIZE THEIR SERVICE RATE REGIONS

Technique	Codes	Results	Description
Waterfilling	Systematic MDS	Theorems 1 and 2	Characterizes the service rate region for $n-k \geq k$
		Lemmas 1 and 2	Show optimality of waterfilling for $n-k < k$
Combinatorial	Binary Simplex	Theorem 3	Characterizes the service rate region
	Non-Systematic MDS	Proposition 3	Characterizes the service rate region
	Primitive Multiset Batch	Proposition 4	Shows a relation between batch codes and integral service rate region (cf. Definition 7)
Geometric*	Binary Simplex	Section VII-C1	Characterizes the service rate region for binary $[7,3]$ Simplex code
	Binary First-Order Reed-Muller	Section VII-C2	Characterizes the service rate region for binary non-systematic $[8,4]$ Reed-Muller code

*Two illustrative examples are included; the rate regions are characterized in [33].

and we aim to maximize λ_k. Then the optimal request rate split $\lambda_{i,j}$ for all $i = 1, \ldots, k$ and $j = 1, \ldots, t_i$ is the solution to the following linear optimization problem:

$$\max_{\lambda_{i,j}: i\in[1,k], j\in[1,t_i]} \lambda_k \quad \text{s.t.} \tag{1}$$

$$\sum_{j=1}^{t_i} \lambda_{i,j} = \lambda_i \text{ for } 1 \leq i \leq k, \tag{2}$$

$$\sum_{i=1}^{k} \sum_{\substack{1\leq j\leq t_i \\ \ell \in R_{i,j}}} \lambda_{i,j} \leq \mu \text{ for } 1 \leq l \leq n, \tag{3}$$

$$\lambda_{i,j} \geq 0, \text{ for } 1 \leq i \leq k, 1 \leq j \leq t_i. \tag{4}$$

The first set of constraints (2) guarantees that the demands for all objects are served. The second set of constraints (3) ensures that the total demand assigned to each server are within its service capacity limit. Note that these recovery groups $R_{i,j}$ can overlap. The set of constraints (3) ensures the stability of the system, where the request arrival rates must be below the corresponding service rates. The rates at which the system is able to serve requests should not be confused with information rates used to provide the service. For example, if satisfying a request for object a involves downloading objects b and $a+b$, then the user requesting a will also get b. He will receive twice as much information as requested but not more service as he did not request b.

Definition 2: Given a distributed system with k data objects stored on n servers and a vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ of non-negative real numbers, a set $\{\lambda_{i,j} : 1 \leq i \leq k, 1 \leq j \leq t_i\}$ satisfying (2)-(4) is referred to as a valid allocation associated with $\boldsymbol{\lambda}$.

Definition 3: Given a distributed system with k data objects stored on n servers, a demand vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ is said to be achievable if there exists a valid allocation associated with $\boldsymbol{\lambda}$. The set of all achievable demand vectors $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ is referred to as the service rate region of the system.

Remark 1: It is well-known in queueing theory (see e.g., [32]) that any vector λ outside the service rate region would make the system unstable. Note that we do not consider coalescing of requests, where a request of a sent to servers storing b and $a+b$ decodes both a and b and then shares the b file with requests for b. Each request for a or b is handled independently.

Note that the queueing and bandwidth data access models, despite their different practical meanings, give rise to the service rate region which is the solution to the optimization problem described by (1)-(2) above.

There are scenarios wherein each user occupies the entire bandwidth of the server they are accessing. This can happen, for instance, when users are streaming from low-bandwidth edge devices and each user needs to be served at a specific rate. We call a service rate region under this constraint as an *integral service rate region* (formally defined in Section VI-E).

Depending on the number of objects and nodes, and the coding scheme, this problem can be very computationally expensive to solve. For example, for systematic MDS codes the number of recovery sets is $\binom{n-1}{k} + 1$, which grows exponentially as n and k increase. Below we highlight three varied approaches that allow us to solve this problem in closed-form for certain classes of codes: 1) water-filling algorithms similar to those used for proving capacity theorems in information theory, 2) combinatorial optimization on graphs, and 3) a geometric approach. Later in the paper we use these approaches to characterize the service rate regions of MDS and locally recoverable codes (Section V), Simplex codes (Section VI), and Reed-Muller codes (Section VII), respectively. We summarize the coding schemes considered in this paper and the approaches used to characterize their service rate regions in Table I.

1) Water-Filling Algorithm: The water-filling or water-pouring algorithm is a common technique to allocate power across multiple channels in a digital communication system [34], [35]. It treats the channels as vessels with uneven bottom levels, proportional to their noise variance. Power is first allocated to the least noisy channel until its signal plus noise reaches the level of the next lowest noise level. In Section V, we extend the concept of water-filling to allocate

requests to servers by treating each server as a vessel with capacity μ, when the storage scheme is an MDS or a locally recoverable code. Each small volume $\epsilon > 0$ of requests within the total $\sum_{i=1}^{k} \lambda_i$ demand is assigned to a recovery group by adding ϵ volume of water to the corresponding vessels. We show that allocating each request to the least-loaded nodes in the smallest recovery group (first the systematic nodes, then the local parities and then the global parities) maximizes the service rate region. In other words, water-filling resource allocation achieves the optimal system throughput. See Section V for a formal description of this technique and bounds on the resulting service rate region. An ongoing research direction is to explore the use of water-filling for other classes of codes and proving its throughput-optimality.

2) Combinatorial Approach: This approach establishes a significant connection between the service rate problem and the well-known fractional matching problem in (hyper)graphs. A connection between distributed storage allocation problems (see [36], [37] and references therein) and matching problems in hyper-graphs has been observed in computer science literature [38] (see also [39]). In particular, it was noted that the uniform model of distributed storage allocation considered in [36] leads to a question which is asymptotically equivalent to the fractional version of a long-standing conjecture by Erdős [40] on the maximum number of edges in a uniform hypergraph.

Here, we introduce a novel technique for constructing a special graph representation of a linear code. In particular using this approach, the following results are shown: 1) equivalence between the service rate problem and the well-known fractional matching problem and 2) equivalence between the integral service rate problem and the matching problem. These equivalence results allow one to use techniques in the rich literature of the graph theory for solving the service rate problem. Leveraging these equivalence results, it is shown that the maximum sum rates that can be simultaneously served by the system equals the fractional matching number in the graph representation of the code, and thus is lower bounded and upper bounded by the matching number and the vertex cover number, respectively. This is of great interest because if the graph representation of a code is bipartite, then the derived upper bound and lower bound are equal which allows one to establish the maximum sum rates that can be served by the system. Utilizing this result, the service rate region of the binary Simplex codes is characterized whose graph representation is bipartite as shown in Sec.VI-D1.

We also show in Sec. VI-E that the notion of integral service rate region opens up interesting connections with batch codes, a class of codes designed for simultaneous access [26]. Specifically, we show that the service rate problem can be viewed as a generalization of the batch code problem, and the multiset primitive batch codes problem is a special case of the service rate problem when the portion of requests assigned to the recovery sets is limited to be integral.

3) Geometric Approach: Finding the service rate region of a given storage scheme is an optimization problem. One natural way to look at this problem is through the geometric approach, introduced in [33], that provides a set of half-spaces whose intersection surrounds the service rate region of a given linear storage scheme. In other words, the geometric approach provides upper bounds (half-spaces) on the sum of each subset of arrival rates in any demand vector $(\lambda_1, \cdots, \lambda_k)$ in the service rate region of a linear code in a more straightforward manner in comparison to other approaches. This technique is of great significance since it allows one to derive upper bounds on the service rates of linear codes without explicitly knowing the list of all possible recovery sets while waterfilling and combinatorial approaches rely on enumeration of all recovery sets that gets increasingly complex when the number of objects k increases.

Using the geometric technique, upper bounds on the service rates of the binary first order Reed-Muller codes and binary Simplex codes are derived. It is worth mentioning that only the cardinality of the recovery sets matters in deriving upper bounds on the service rate of the first order Reed-Muller codes using the geometric approach. Subsequently, it is shown that how the derived upper bounds can be achieved. Moreover, it is shown that given the service rate region of a code, a lower bound on the minimum distance of the code can be obtained. This approach will be discussed further in Sec. VII. For the original observation and more details, see [33].

B. Designing Storage Schemes to Maximize or Cover the Service Rate Region

Complementary to the problem of finding the service rate region of a given storage scheme, we now discuss the problem of designing the underlying storage scheme to achieve a target service rate region with the minimum number of nodes. The target service rate region represents a known probability distribution of demand vectors $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ that can be supported by the system. A system designer aims to support this demand distribution using the minimum number of servers or to maximize the volume of the service rate region for a given number of servers. Below we discuss these two facets of the storage scheme design problem and provide some initial solution perspectives. These problem is largely open and requires fundamental coding theoretic innovations.

1) Maximize Service Rate Region With a Given Number of Servers: Consider a practical scenario where a fixed number of nodes n is available to store k objects, and a coding scheme is to be designed to maximize the service rate region. This problem was considered for $k = 2$ in [1] for $k = 3$ in [41]. For example, consider four different schemes to store $k = 2$ files on a system of $n = 8$ servers as shown in Figure 3, where α is a primitive element of $\mathbb{F}_9$ or a larger finite field.

Their service rate regions are illustrated on the left side of the figure. It is interesting to note that the service region depends on the encoding rather than on the code itself. In particular, the two codes in the middle (shown in blue and purple) are identical from a coding theory perspective, but their service rate regions are different. Amongst the four codes, we observe a combination of replication and coding (shown in red), where we create 3 replicas each of a and b and 2 coded combinations $a+b$ and $a+\alpha b$ can achieve the largest (by area) service rate region. Recent work [1] described the service rate region when

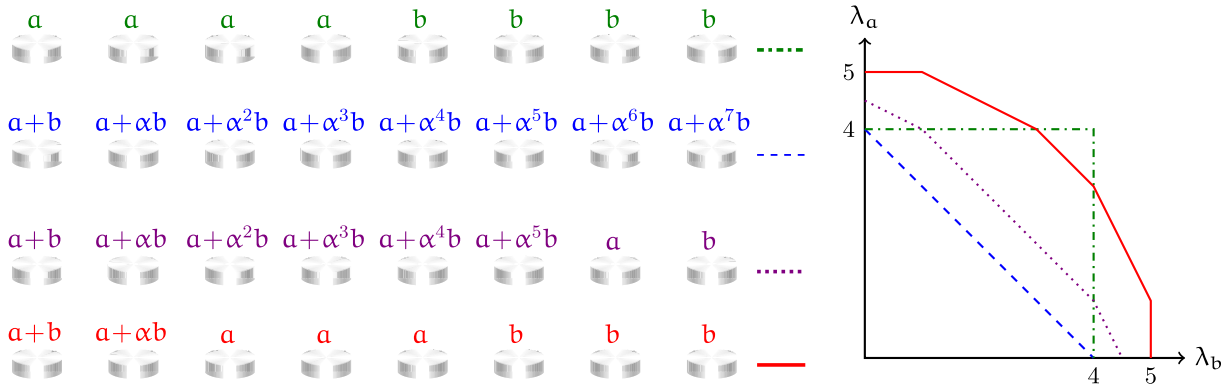

Fig. 3. Four coding schemes and their corresponding service rate regions. The largest region is achieved by combining coding and replication. (α is a primitive element of a sufficiently large finite field.)

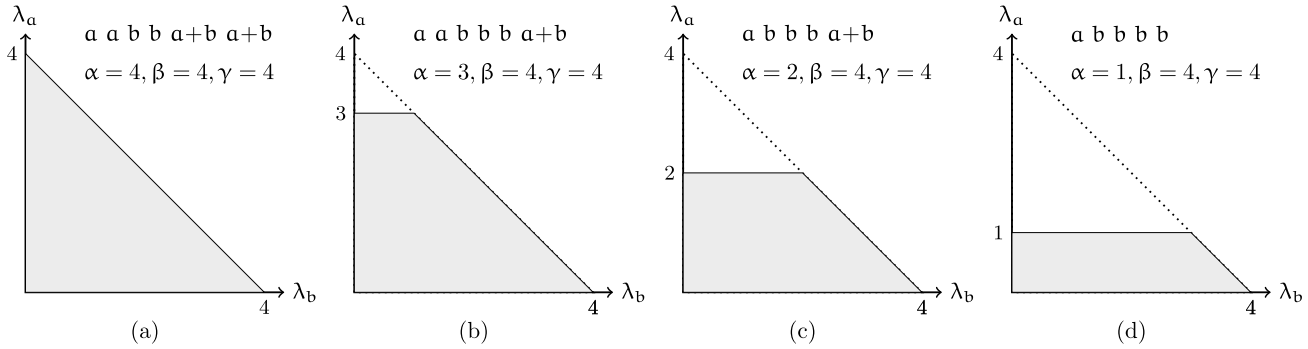

Fig. 4. Four service rate regions defined by the constraints $\lambda_a, \lambda_b \geq 0, \lambda_a \leq \alpha, \lambda_b \leq \beta, \lambda_a + \lambda_b \leq \gamma$, and their corresponding storage schemes that cover them with a minimum number of nodes.

there are A replicas of object a, B replicas of object b, and C coded combinations of a and b. Given the total number of servers $n = A + B + C$, determining the optimal values of A, B and C that maximize the area of the service rate region is an ongoing research direction. More generally, designing the generator matrix of a code to maximize the area/volume of the service rate region is an open problem. Also, since the service rate region is multi-dimensional, designing a fair metric other than the area/volume of the service rate region in order to compare the rate regions of two different classes of codes is an open problem. We propose some alternative metrics in Section VIII.

2) Minimize the Number of Servers to Cover a Given Service Rate Region: Consider a scenario where $k = 2$ data objects, movies "a" and "b", are stored redundantly across multiple nodes in a coded storage system. At each time, each node can serve at most one request and each user can request to download at most one of the two movies a and b. It is known that the number of users who are interested in downloading the movie a and b is less than or equal to α (i.e., $\lambda_a \leq \alpha$) and β (i.e., $\lambda_b \leq \beta$), respectively. Also, it is known that the total number of users in the area is at most γ (i.e., $\lambda_a + \lambda_b \leq \gamma$). This means that the desired service rate region of this storage system is a bounded set $\mathcal{R}$ defined as follows:

$$\mathcal{R} = \{\lambda_a, \lambda_b \geq 0, \lambda_a \leq \alpha, \lambda_b \leq \beta, \lambda_a + \lambda_b \leq \gamma\}. \quad (5)$$

Two natural questions that arise in the design of this distributed storage system are the following: 1) What is the minimum number $n(\mathcal{R})$ of nodes required to serve all request vectors (λ_a, λ_b) in the set $\mathcal{R}$? 2) How should the files a and b be stored redundantly in $n(\mathcal{R})$ storage nodes (i.e., what is the most storage-efficient redundancy scheme)?

Using the example shown in Figure 4, we briefly illustrate how the storage-minimizing scheme varies with the shape of the service rate region that we wish to cover. Let $\beta = 4$ and $\gamma = 4$, and $\alpha \in \{1, 2, 3, 4\}$. The corresponding four storage-minimizing redundancy schemes (one for each α) together with their service rate regions are shown in Figure 4. In Figure 4(a), the rate region is dominated by points (λ_a, λ_b) for which the demands for a and b are complementary to each other, that is, if λ_a is high then λ_b is low, and vice-versa. In this case, adding two coded nodes $a+b$ is the most storage-efficient way for achieving the service rate region. On the other hand, in Figure 4(d), where the demand for movie b dominates the total request rate $\lambda_a + \lambda_b$, the best storage scheme does not have any coded nodes; it simply replicates object b four times, and keeps just one uncoded copy of a.

In general, the problem of minimizing the number of nodes required for covering a desired service rate region can be formulated as an integer linear programming (ILP). This is a challenging problem because in order to list all the constraints, one needs to explicitly know all possible recovery sets which becomes increasingly complex when the number of files k

increases. Recently, this problem of designing storage-efficient schemes to cover a desired service rate region has been studied for the first time in [42], but there are still many open problems. For further details, please see [42].

IV. Storage Schemes Considered in This Paper

Each of the n nodes in the system stores a linear combination of k data objects which are mathematically represented as elements of the finite field $\mathbb{F}_q$. We refer to a linear combination of data objects as coded object. If the linear combinations involves only one object, we call it systematic. We use the same terminology for the corresponding storage nodes. We refer to a linear code over a finite field $\mathbb{F}_q$ with block-length n and dimension k as an $[n,k]$ code. The generator matrix G of a linear code is an $k \times n$ size matrix whose rows are a basis of the code, and their linear combinations form the codewords. We focus our attention to three classes of codes that are well-known in coding theory, see, e.g., [43].

A. Maximum-Distance-Separable (MDS) Codes

MDS codes achieve the well-known Singleton upper bound on the minimum distance, $d_{\min} \leq n-k+1$, hence the name. For an $[n,k]$ linear MDS code, any k columns of the generator matrix are linearly independent, and thus all the k data objects can be recovered from any k encoded objects. Therefore, for a systematic MDS code, the minimal recovery sets of a systematic column are the column itself and any k of the remaining $n-1$ columns. An example of MDS codes commonly used in distributed storage systems is Reed Solomon codes.

B. Simplex Codes

A binary *Simplex code* (aka Hadamard code in CS literature) is a $[2^k-1,k]$ code with a generator matrix consisting of all distinct nonzero vectors of $\mathbb{F}_2^k$.[2] Note that any generator matrix of a Simplex code has this form. Simplex codes are useful in distributed storage systems, since each symbol of a Simplex code has $t = 2^{k-1}-1$ disjoint recovery sets of size two each [44]. They are known to be optimal in several ways: i) they meet the upper bound on the distance of codes having recovery sets of size at most two [44]; ii) they achieve the maximum storage efficiency among the binary linear codes with a given number of disjoint recovery sets of size two [45]; iii) they meet the Griesmer bound and are therefore linear codes with the lowest possible length given the code distance [46]. Simplex codes play an important role in Computer Science as well, where they are known as Hadamard codes.

C. First Order Reed-Muller (RM) Codes

A k-dimensional binary first-order Reed-Muller code $\mathrm{RM}_2(1,k-1)$ with parameter $k \geq 2$, is a linear $[2^{k-1},k]$ code [47]–[50]. RM codes are important in both theory and practice. For a given k, the generator matrix of the $\mathrm{RM}_2(1,k-1)$ can be constructed as follows.

Denote the set of all $(k-1)$-dimensional binary vectors by $\mathbb{F}_2^{k-1} = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ where $n = 2^{k-1}$ and for $i \in \{1,\ldots,n\}$, $\mathbf{x}_i = (x_{i,k-1},\ldots,x_{i,1})$ with $x_{i,j} \in \mathbb{F}_2$, $j \in \{1,\ldots,k-1\}$. For any $\mathcal{A} \subseteq \mathbb{F}_2^{k-1}$, define the indicator vector $\mathbb{I}_{\mathcal{A}} \in \mathbb{F}_2^{k-1}$ as follows:

$$(\mathbb{I}_{\mathcal{A}})_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in \mathcal{A}; \\ 0 & \text{otherwise.} \end{cases}$$

For the k rows of the generator matrix of $\mathrm{RM}_2(1,k-1)$, define k row vectors of length 2^{k-1} as follows, $\mathbf{r}_0 = (1,\ldots,1)$ and $\mathbf{r}_j = \mathbb{I}_{\mathcal{H}_j}$, where $j \in \{1,\ldots,k-1\}$ and $\mathcal{H}_j = \{\mathbf{x}_i \in \mathbb{F}_2^{k-1} \mid x_{i,j} = 0\}$. The set $\{\mathbf{r}_{k-1},\ldots,\mathbf{r}_1,\mathbf{r}_0\}$ defines the rows of a non-systematic generator matrix of the $\mathrm{RM}_2(1,k-1)$. For a systematic generator matrix of $\mathrm{RM}_2(1,k-1)$, the set of rows $\{\mathbf{r}_{k-1},\ldots,\mathbf{r}_1,\sum_{i=0}^{k-1}\mathbf{r}_i\}$ can be considered.

V. Service Rate Region Using Waterfilling

In this section we find the service rate region of a system of n servers that store k data objects $u_1,\ldots,u_k$ using maximum-distance-separable codes or locally recoverable codes. Suppose that we use an $[n,k]$ systematic MDS code to generate the data stored on each of the n servers. Each object u_i can be downloaded from the server storing it, which we refer to as the systematic server, or by accessing any k of the remaining $n-1$ servers. Let the arrival rate of requests for object u_i be λ_i. We want to determine the set of arrival rate vectors $(\lambda_1,\ldots\lambda_k)$ that can be supported by the system. In other words, without loss of generality, we want to maximize λ_k for any given a feasible set of rates $(\lambda_1,\ldots,\lambda_{k-1})$.

We propose the following *water-filling* algorithm to split the request rate among the n servers of an (n,k) coded system. The high-level idea behind this algorithm is that requests are first routed to the respective uncoded or systematic server. Once the systematic servers are saturated, the requests are sent to the k least-loaded servers that have not been yet saturated by μ rate of requests. Below, we present the water-filling algorithm for MDS and locally recoverable codes and show that its resulting request allocation achieves the optimal service rate region for MDS codes. However, the core idea of water-filling-based request splitting is broadly applicable beyond these two classes of codes.

Definition 4 (Waterfilling Algorithm for MDS Coded Systems): Assume that the request arrival rates λ_1, λ_2, ...λ_k for the k objects are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ without loss of generality. Let γ_i denote the assigned load, or the request rate assigned to server i. The water-filling algorithm assigns them to the n servers as follows.

1) **Assign requests to systematic (uncoded) nodes.** We first assign the arrival rate λ_i for object i to the respective systematic (uncoded) server until that server is saturated. Thus, the i^{th} systematic server gets assigned the load $\gamma_i = \min(\lambda_i,\mu)$ for $i = 1,\ldots k$. The remaining

[2]Although Simplex codes can be defined over any finite field, in this paper we restrict our attention to the binary Simplex codes.

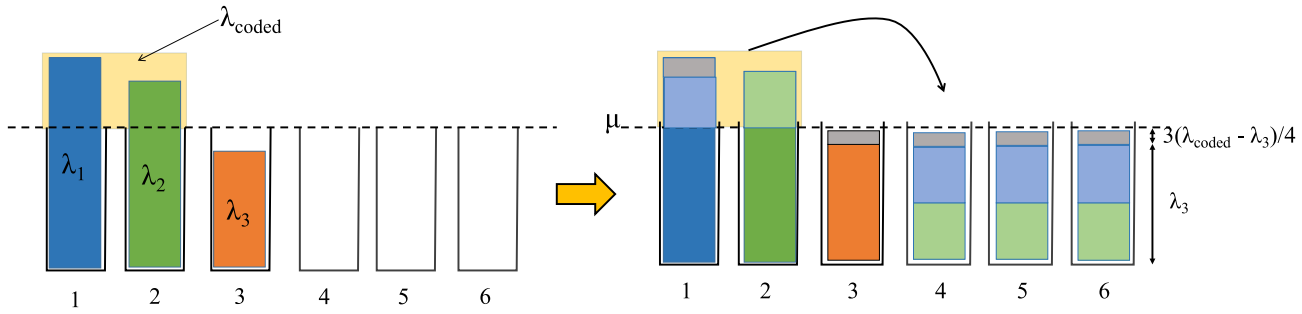

Fig. 5. Water-filling strategy to serve the requests using coded nodes for a $(6, 3)$-MDS code.

$\sum_{i=1}^{k}(\lambda_i - \min(\lambda_i, \mu)) = \sum_{i=1}^{k}(\lambda_i - \mu)^+$ arrival rate needs to be served using the $(n-k)$ coded nodes and the unsaturated systematic nodes.

2) **Assign each request to the k least-loaded nodes.** The remaining $\lambda_{coded} = \sum_{i=1}^{k}(\lambda_i - \mu)^+$ load is split across the nodes in the following manner. While $\lambda_{coded} > 0$ and $\min_i \gamma_i < \mu$ do the following:
   - Find the set $\mathcal{S}$ of the k least-loaded servers (with minimum γ_i) in the system. If there are more than k servers with the same minimum γ_i, choose k servers uniformly at random.
   - Assign a small rate $\epsilon > 0$ of requests to these k least-loaded servers
   - Decrement λ_{coded} by ϵ, that is, $\lambda_{coded} \leftarrow \lambda_{coded} - \epsilon$
   - Increment the corresponding k server loads by ϵ, that is, $\gamma_i \leftarrow \gamma_i + \epsilon$ for all $i \in \mathcal{S}$.

The algorithm is illustrated in Figure 5 for a $(6, 3)$ MDS code. After sending the requests to their respective systematic nodes, nodes 1 and 2 are saturated but since $\lambda_3 < \mu$, the third systematic node has some remaining service capacity, which can be used to serve the overflow of requests for objects 1 and 2. The total overflowing request rate for data objects 1 and 2 is $\lambda_{\text{coded}} = (\lambda_1 - \mu)^+ + (\lambda_2 - \mu)^+$ left to be served. These requests can be served by accessing any $k = 3$ of the unsaturated nodes in the system, decoding all $k = 3$ data objects, and obtaining the object of interest. Since all $k = 3$ objects end up being decoded, we do not need to consider the overflowing requests for objects 1 and 2 separately, but consider them together as $\lambda_{\text{coded}} = (\lambda_1 - \mu)^+ + (\lambda_2 - \mu)^+$.

We first send λ_3 out of the λ_{coded} rate (shown by the light green and blue shaded regions in Figure 5) to the MDS coded nodes 4, 5, 6, since these are the k least loaded nodes in the system. After this allocation, servers 3, 4, and 5, 6 all have $\mu - \lambda_3$ capacity left and are the least-loaded nodes in the system. Any 3 of these 4 servers can be used to serve the remaining $\lambda_{\text{coded}} - \lambda_3$ rate of requests (shown in grey color). Each coded request will be served by accessing 3 coded data objects and then decoding the object of interest. There are $\binom{4}{3} = 4$ sets of 3 servers each, and each server participates in 3 such sets. Thus, the additional load allocated to servers 3, 4, 5, 6 (shown in grey) is $3(\lambda_{\text{coded}} - \lambda_3)/4$ rate each. Here we need the additional load $3(\lambda_{\text{coded}} - \lambda_3)/4$ to be less than $\mu - \lambda_3$, the remaining capacity of each of the nodes 3, 4, 5 and 6.

A. Service Rate Region for MDS Codes

Below we first find a converse or upper bound on the achievable service rate region of MDS codes.

Theorem 1: The set of all achievable request vectors $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ of an (n, k) systematic-MDS coded system lies inside the region described by

$$\sum_{i=1}^{k} \left(\min(\lambda_i, \mu) + k(\lambda_i - \mu)^+\right) \leq n\mu, \tag{6}$$

Proof: To prove this outer bound on the achievable rate region, observe that each server in the system can support μ requests/time, and thus the total capacity is $n\mu$. Downloading each data object from its own systematic (uncoded) node uses only 1 unit of capacity. However, downloading an object from k coded servers requires k units of capacity per unit request rate. Thus, if λ_i is the rate of request arrivals for object i, the minimum system capacity utilized by these requests is $\min(\lambda_i, \mu) + k(\lambda_i - \mu)^+$, where $\min(\lambda_i, \mu)$ requests are served by the systematic node storing object i. Since the total system capacity is $n\mu$, the sum of the capacity utilized by all requests must be less than $n\mu$. Thus we have (6). □

Next, we show that this water-filling algorithm is optimal, that is, it can serve any achievable set of request rates $(\lambda_1, \ldots \lambda_k)$. To prove the optimality we separately consider two cases below: 1) $n - k \geq k$ (the code rate $\leq 1/2$), and 2) $n - k < k$ (the code rate $> 1/2$).

Theorem 2: The water-filling algorithm proposed in Definition 4 is optimal, that is, it achieves the outer bound given by (6), for any MDS code when $n - k \geq k$.

Proof: For $n - k \geq k$ we now evaluate the set of arrival rates that can be achieved by the waterfilling algorithm and show that it matches the outer bound in (6). Without loss of generality, sort the arrival rates in descending order such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$. After sending requests to systematic servers until they are saturated, the total residual arrival rate is $\lambda_{coded} = \sum_{i=1}^{k}(\lambda_i - \mu)^+$, as illustrated in Figure 5 for the $(6, 3)$ MDS coded system. Assume that $\lambda_1 \geq \mu$. If this is not true, then $\lambda_{coded} = 0$ and all requests can be served by systematic servers.

The waterfilling algorithm, we first uniformly split requests over $n - k$ coded nodes, $k + 1, \ldots n$, until the load at all these nodes becomes equal to λ_k (the load of least-loaded systematic node). Since each request needs be sent to k out of the $n - k$ servers, up to $\min(\lambda_k, \mu)(n - k)/k$ requests

can be served in this manner. After this assignment, there are $n-k+1$ nodes from node $i=k,\dots n$ with the same load $\gamma_i=\min(\lambda_k,\mu)$. The waterfilling algorithm now assigns each request to the least-loaded k out of these $n-k+1$ servers until their load reaches $\min(\lambda_{k-1},\mu)$, the load of the $(k+1)^{th}$ least-loaded server. The request rate assigned this way is $(\min(\lambda_{k-1},\mu)-\min(\lambda_k,\mu))(n-k+1)/k$. Recursively repeating this process for every $r=k,\dots,2$, we uniformly split $\min((\gamma_{r-1}-\gamma_r)(n-r+1)/k,\lambda_{coded})$ requests over $n-r+1$ servers, $r,r+1,\dots n$. Thus the maximum rate λ_{coded}^{max} of requests that can be supported using coded servers is

$$\begin{aligned}\lambda_{coded}^{max}&=\min(\lambda_k,\mu)\frac{n-k}{k}\\&\quad+(\min(\lambda_{k-1},\mu)-\min(\lambda_k,\mu))\frac{n-k+1}{k}\\&\quad+\cdots+(\min(\lambda_1,\mu)-\min(\lambda_2,\mu))\frac{n-1}{k}\qquad(7)\\&=\min(\lambda_1,\mu)\frac{n}{k}-\sum_{i=1}^{k}\min(\lambda_i,\mu)\frac{1}{k}\qquad(8)\\&=\mu\frac{n}{k}-\sum_{i=1}^{k}\min(\lambda_i,\mu)\frac{1}{k}\qquad(9)\end{aligned}$$

In Figure 5, the height of each lightly-shaded portion, starting from the bottom upwards, corresponds to each term in the above summation.

After saturating the systematic nodes, the residual rate $\lambda_{coded}=\sum_{i=1}^{k}(\lambda_i-\mu)^+$ supported by the coded servers can be at most λ_{coded}^{max}. That is,

$$\lambda_{coded}\le\lambda_{coded}^{max}\qquad(10)$$

$$\sum_{i=1}^{k}(\lambda_i-\mu)^+\le\mu\frac{n}{k}-\sum_{i=1}^{k}\min(\lambda_i,\mu)\frac{1}{k}\qquad(11)$$

Rearranging, this is equivalent to (6). Thus, for $n-k\ge k$, waterfilling can achieve the region given by the outer bound in Equation 6. Hence, the proposed waterfilling algorithm is optimal for $n-k\ge k$. □

Next let us consider the second case $n-k<k$. For this case, we cannot always achieve the same rate region as given by the outer bound in (6). However, we can show that the waterfilling algorithm is optimal, and no other rate splitting scheme can yield a strictly larger rate region.

Lemma 1: It is optimal to first send requests to their systematic node. Only when the systematic node is saturated, requests should be served using coded servers.

Proof: Suppose $\lambda_i<\mu$ for some i, that is all requests for object i can be served by the systematic node. Instead, suppose we serve $\lambda_i-\epsilon$ rate using the systematic node i, and send the remaining ϵ portion to k other servers, and decode file f_i from the coded versions. As a result we are reducing the load on the systematic node by ϵ, and instead adding ϵ load to K other servers. If $n-k<k$, at least one of these k servers is also a systematic node, which stores file f_j. Thus, the maximum rate of requests for file f_j that can be served by its systematic node reduces by ϵ. For $n-k>k$, we showed in Theroem 2 that the water-filling algorithm, which first sends requests to the systematic node is optimal. Thus, there is no loss of optimality in sending requests to the systematic node until it is saturated. □

Lemma 2: After the systematic node is saturated, it is optimal to always send each request to the k least-loaded servers that can serve it.

Proof: Each $\epsilon>0$ portion of the request rate λ_{coded}, needs to be allocated to k servers. Using any algorithm for picking the k servers, we could reach one of the two possible states:

1) $r\ge k$ unsaturated servers with the same load $\gamma<\mu$. Then we can split a maximum of $(\mu-\gamma)r/k$ request rate uniformly over these servers. As a result all servers will be saturated, and the outer bound on the service rate region can be achieved.
2) There are exactly k unsaturated servers in the system with loads $\gamma_1\ge\gamma_2\ge\gamma_3\ge\cdots\ge\gamma_k$, where at least one of these inequalities is strict. The waterfilling can serve an additional $\mu-\gamma_1$ rate of requests. This would leave a non-zero amount of capacity unused.

Since water-filling algorithm always sends requests to the k least-loaded nodes in the system, it achieves the first state whenever it is feasible. And if the system ends up in the second state, water-filling minimizes the total unused system capacity $n\mu-\sum_{i=1}^{n}\gamma_n$ by always allocating requests to the least-loaded servers. □

B. Service Rate Region for Locally Recoverable Codes

Locality of a code captures the number of symbols participating in recovering a lost symbol. In particular, an $[n,k]$ code is said to have locality r if every symbol is recoverable from a set of at most r symbols. For linear codes with locality, a *local* parity check code of length at most $r+1$ is associated with every symbol. The notion of locality can be generalized to accommodate *local codes* of larger distance as follows (see [51]). For an $[n,k]$ code $\mathcal{C}$ and a subset $S\subset[n]$, we use $\mathcal{C}_S$ to denote $\mathcal{C}$ restricted to symbols in S.

Definition 5 (Locality): An $[n,k]$ code $\mathcal{C}$ is said to have (ℓ,r) information locality $(\ell>r)$, if for every data object i, there exists a set of indices Γ_i such that (i) $i\in\Gamma_i$, (ii) $|\Gamma_i|\le\ell$, and (iii) $d_{\min}(\mathcal{C}_{\Gamma_i})\ge\ell-r+1$. The code $\mathcal{C}_{\Gamma_i}$ is said to be the local code associated with the i-th data object.

Properties 2 and 3 imply that for any codeword in $\mathcal{C}$, the values in Γ_i are uniquely determined by any r of those values. Therefore, the (ℓ,r) locality allows one to *locally* repair any $\ell-r$ erasures in $\mathcal{C}_{\Gamma_i}$, $\forall i\in[n]$, by accessing r other objects. When $\ell=r+1$, the above definition reduces to the classical definition of locality proposed by Gopalan et al. [52], wherein any one erasure can be repaired by accessing at most r objects.

Throughout the rest of this section, we focus on LRCs that have the same structure as the Pyramid code from [53]. In particular, the k data objects are partitioned into k/r groups, and each group has $\ell-r$ local parities satisfying the properties of Definition 5. Each such group is called a local group. Further, the code has p global parities.

Example 1: Consider an $(12,4)$ LRC with $(4,2)$ locality and $p=4$ global parities which encodes $[a,b,c,d]$ into

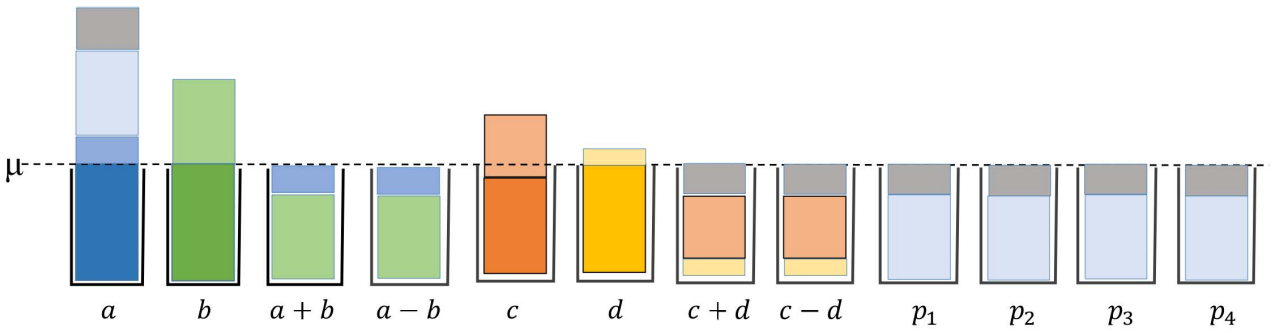

Fig. 6. Water-filling strategy to serve the requests using coded nodes for a $(12, 4)$-LRC code.

$[a, b, c, d, a + b, c + d, a - b, c - d, p_1, p_2, p_3, p_4]$ where p_i, $1 \leq i \leq 4$ denote global parity symbols. Observe that $(a, b, a + b, a - b)$ and $(c, d, c + d, c - d)$ are local groups.

Next, we generalize the waterfilling algorithm to LRCs. One key difference than MDS codes is that in LRCs it is not possible to recover all the k data objects from any k coded objects. In particular, each local group has r linearly independent symbols, and sending a request to more than r servers in a local group is redundant. Further, some set of k servers cannot recover all the data symbols.[3] For instance, for Example 1, any one parity server from each of the two local groups together with any two global parity servers cannot be used to recover the four data objects. On the other hand, it is not difficult to see that in parity-splitting LRCs like Pyramid codes, one can recover the k data objects from any r parity symbols for ℓ local groups, where $1 \leq \ell \leq \lceil k/r \rceil$, and any $k - r\ell$ global parity symbols. We restrict to such sets of servers in the final step of waterfilling.

Definition 6 (Waterfilling Algorithm for LRC Coded Systems): Assume that the request arrival rates $\lambda_1, \lambda_2, \ldots \lambda_k$ for the k objects are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ without loss of generality. Let γ_i denote the assigned load, or the request rate assigned to server i.

1) **Assign requests to systematic (uncoded) nodes and coded nodes in local groups using waterfilling as in Definition 4.**
2) **Assign each request to k least loaded coded nodes.** Denote the remaining load as λ_{coded}.
   While $\lambda_{coded} > 0$ and $\min_i \gamma_i < \mu$ do the following:
   - Find a set $\mathcal{S}$ of k least loaded servers such that if the set contains a parity server from any local group, then it should contain r servers from the same group. If there are multiple such sets of k least loaded servers, choose a set uniformly at random.
   - Assign a small rate $\epsilon > 0$ of requests to every server in $\mathcal{S}$, increment the corresponding server loads by ϵ, and decrement λ_{coded} by ϵ.

The algorithm is illustrated in Figure 6 for the $(12, 4)$ code in Example 1. Notice that $\lambda_1 > \lambda_2 > \lambda_3 > \lambda_4$, where we denote objects $\{1, 2, 3, 4\}$ as $\{a, b, c, d\}$, respectively. First, after sending requests for c and d to the respective systematic nodes, the remaining requests are sent to the local group $\{c+d, c-d\}$. Similarly, after saturating node b, the remaining requests for b are sent to the local group $\{a+b, a-b\}$. Since the local group $\{a + b, a - b\}$ still has some leftover capacity, some of the a requests are sent to this group (after saturating the systematic node a). Afterwards the remaining a requests are sent to the four least loaded nodes, which turn out to be the global parity nodes $\{p_1, p_2, p_3, p_4\}$. At one point, the global parity nodes $\{p_1, p_2, p_3, p_4\}$ share the same load as the local group nodes $\{c + d, c - d\}$. From this point onward, a set of four least-loaded servers out of $\{c + d, c - d, p_1, p_2, p_3, p_4\}$ are selected, and a small rate ϵ is assigned to them. This is continued until all the servers are saturated.

We emphasize that, unlike MDS codes for which the waterfilling algorithm is optimal, it is open whether the waterfilling algorithm is optimal for LRC codes. The techniques used for proving the optimality of the waterfilling algorithm for MDS codes are not sufficient for analyzing LRC codes. This is because, after the systematic nodes are saturated, a request can be satisfied using three ways: (i) only the local parity nodes, (ii) a mix of local and global parity nodes, and (ii) only the global parity nodes (when $n - k \geq k$). Thus, the recovery sets for LRCs are more complex (as opposed to k-node subsets for MDS codes), which calls for novel techniques to analyze the waterfilling algorithm for LRCs.

C. Summary

In this section, we introduced the waterfilling strategy to determine how to split requests across different nodes in a coded distributed system. We analyzed it for MDS and locally recoverable codes, and showed that it is optimal for MDS codes. Proving its optimality for LRCs remains open for the reasons that we described above. Simplex codes can also be considered as LRCs with $(3, 2)$ locality [44], [45]. Therefore, one can use the waterfilling algorithm to allocate requests in a Simplex coded system as well. However, it is not clear how to use waterfilling arguments to characterize the service rate region for Simplex codes in a closed form. As we will see next, the other two approaches, combinatorial optimization and geometric, are well suited to characterize the service rate region for the Simplex codes. In general, not surprisingly, each approach is well suited for specific types of codes.

More broadly, the waterfilling strategy encompasses two key ideas. Firstly, it takes into account the fact that each request is associated with a ranked preference list of subsets of servers that it wants to be assigned to. For example,

[3]LRCs which have the information-theoretically optimal recovery guarantees are referred to as maximally recoverable codes. See [54] and references therein.

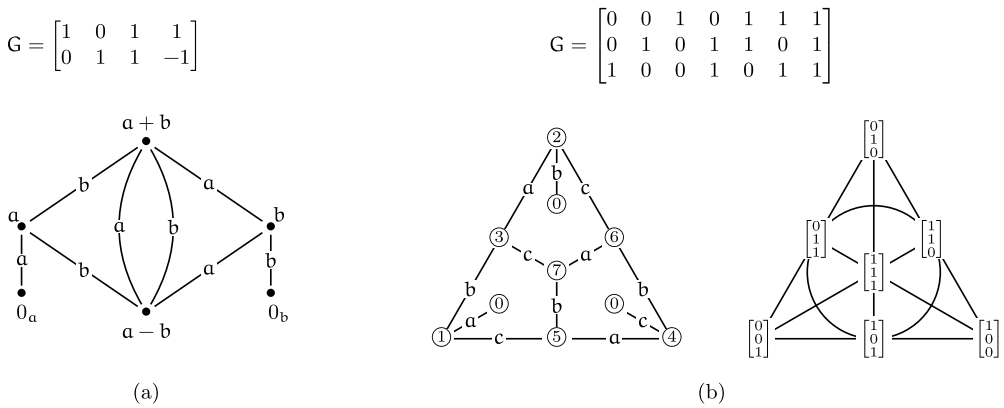

Fig. 7. (a) Generator matrix and its recovery graph for a systematic $[4, 2]$ MDS code. (b) Generator matrix and its recovery graph for the $[7, 3]$ Simplex code, and the Fano plane.

in MDS coded systems sending a request to a systematic node preferred over sending it to k coded nodes. Secondly, at each preference level, waterfilling assigns the request to the least loaded server(s) in order to maximize the achievable service rate region. Although we propose it in the context of coded storage systems, these central ideas of the waterfilling strategy can be utilized more broadly for resource allocation in distributed systems. Exploring other applications of this strategy is an interesting and open future direction.

VI. Service Rate Region Using Combinatorial Optimization on Graphs

In this section, we introduce a graph representation of a coding scheme, and look at the service rate region problem through the lens of combinatorial optimization on graphs. We begin with briefly reviewing the notions of matching and vertex cover in graphs. For details, we refer the reader to standard texts on graph theory, e.g., [55].

A. Matching and Vertex Cover on Graphs

A *matching* in graph Γ is a set of pairwise non-adjacent edges. A maximum matching in Γ is a matching that contains the largest number of edges. The size of a maximum matching is known as the *matching number*, and it is denoted as $\nu(\Gamma)$. Note that a matching can be considered as assigning to each edge a weight from the set $\{0, 1\}$ such that the sum of the weights on the edges incident on any vertex is at most one.

A *fractional matching* allows one to assign any fraction in the interval $[0, 1]$ as a weight to each edge such that the sum of the weights on the edges incident on any vertex is at most one. A maximum fractional matching of Γ has the maximum sum of weights among all the fractional matchings of Γ. The sum of weights of a maximum fractional matching is known as the *fractional matching number*, and it is denoted as $\nu_f(\Gamma)$.

A *vertex cover* of a graph Γ is a set of its vertices such that each edge in Γ is incident to at least one vertex in the set. A minimum vertex cover is a vertex cover of smallest possible size. The size of a minimum vertex cover is known as the *vertex cover number*, and it is denoted as $\tau(\Gamma)$. For any graph Γ, it holds that $\nu(\Gamma) \leq \nu_f(\Gamma) \leq \tau(\Gamma)$, and, in particular for a bipartite Γ, we have $\nu(\Gamma) = \nu_f(\Gamma) = \tau(\Gamma)$.

B. Graph Representation of Storage Schemes

Here, we introduce a graph representation of storage schemes described in Section IV. For simplicity, we consider linear codes, however, it is straightforward to generalize the notion for non-linear codes. For the clarity of exposition, we focus on recovery sets of size one and two. In other words, a recovery set for each object is either a systematic symbol or a group of two symbols, as is the case when $k = 2$ and for Simplex codes of any dimension. As we discuss in Remark 2 later, the notions described next can be easily extended to the general case of arbitrary sized recovery sets by considering hypergraphs. We consider hypergraphs associated with MDS codes in Sec. VI-D2.

Consider an $[n, k]$ code with a $k \times n$ generator matrix G. We define a graph Γ_G associated with the G as follows. Γ_G has n vertices corresponding to n columns of G. For every recovery set (of size two) of data symbol x, the corresponding vertices in Γ_G are connected by an edge with label x. We refer to such an edge as x-recovery edge. If G is systematic, an additional vertex is added for each systematic column, and it is connected by an edge to the vertex corresponding to the systematic column and labeled accordingly. This avoids self-loops corresponding to recovery sets of size one formed by the systematic columns. We refer to Γ_G as a *recovery graph* for the coding scheme G.

Figure 7 shows generator matrices with their recovery graphs for a systematic $(4, 2)$ MDS code and the $[7, 3]$ Simplex code. In Sec. VI, we show how the service rate problem associated with matrix G is related to matching and vertex cover problems of its recovery graph Γ_G.

C. Service Allocation as a Fractional Matching in the Recovery Graph

Associating a recovery graph with a coding scheme allows us to relate the service allocation problem to the problem of finding a fractional matching in the recovery graph. (For the original observation and more details, see [56].) Let us consider a coding scheme G and its recovery graph Γ_G. We demonstrate that a valid allocation for the coding scheme G for a given demand vector is equivalent to a fractional matching with specific constraints on Γ_G. In the rest of this section, we assume without loss of generality that $\mu = 1$.

Proposition 1: Consider a system using an $[n, k]$ code with a generator matrix G where every recovery set is of size at most two and $\mu = 1$. The system can serve a demand vector $(\lambda_1, \cdots, \lambda_k)$ if and only if there exists a fractional matching in the recovery graph Γ_G such that the weights on the edges with label i sum to λ_i.

Proof: Suppose there exists a fractional matching in Γ_G such that the weights on the edges with label i sum to λ_i. Let $w_{i,j}$ denote the weight on the edge corresponding to recovery set $R_{i,j}$. Then, for every object i, assign $w_{i,j}$ fraction of its load λ_i to recovery set $R_{i,j}$ for $j \in [t_i]$. Since the sum of the edge weights at any vertex does not exceed one, no server is assigned requests in excess of its service rate. Further, since the weights on the edges with label i sum to λ_i, all demands are served.

On the other hand, suppose that there is a valid allocation for a demand vector $(\lambda_1, \lambda_2, \cdots, \lambda_k)$. Let $\lambda_{i,j}$ denote the load assigned to recovery set $R_{i,j}$. Then, for every $i \in [k]$, assign the weight $\lambda_{i,j}$ for the edge labeled i that is corresponding to recovery set $R_{i,j}$. Since the allocation $\{\lambda_{i,j} : 1 \leq i \leq k, 1 \leq j \leq t_i\}$ satisfies (2)-(4), it is immediate to see that the weights assigned form a fractional matching such that the sum of the weights on the edges with label i sum to λ_i. □

Remark 2: In defining recovery graphs and in Proposition 1, we restricted our attention to linear coding schemes having recovery sets of size at most two. Extension to the general case of a code having recovery sets of arbitrary size is straightforward: we associate a hypergraph with the code's generator matrix. (A hypergraph is a generalization of a graph in which any subset of vertices may be joined by an edge, called a hyperedge, see, e.g., [57, Chapter 7].) We form a hypergraph Γ_G associated with G such that its vertices correspond to columns of G and hyperedges correspond to recovery sets. It is straightforward to generalize the hypergraph representation for non-linear codes. See Sec. VI-D2 for hypergraphs associated with MDS codes.

The relation to fractional matching enables us to obtain bounds on (and, in some cases, completely characterize) the service rate region. First, we present a bound on the sum of the request rates that can be served by the system using vertex covers in Γ_G. Recall that a vertex cover of a graph Γ is a set of vertices of Γ such that each edge in Γ is incident to at least one vertex in the set. From Proposition 1 and the well-known combinatorial optimization result that the fractional matching number is upper bounded by the vertex cover number of any graph, we get the following upper bound on the sum of request rates that can be served by a system.

Proposition 2: Consider a system using an $[n, k]$ code with a generator matrix G, and let Γ_G be the recovery graph of G. The sum of rates in any demand vector $(\lambda_1, \cdots, \lambda_k)$ that can be served by the system cannot exceed the number of vertices in a cover of Γ_G.

D. Using Graph Representations to Characterize Service Rate Regions

1) Simplex Codes: Here, we characterize the service rate region of binary Simplex codes. For clarity of exposition, we focus our attention to non-overlapping recovery sets of size two. Later, we show that considering all the recovery sets does not increase the service rate region. Note that, since any generator matrix of a Simplex code consists of all non-zero length-k binary vectors, any generator matrix is a column permutation of the other. Thus, recovery graphs associated with all generator matrices of a Simplex code are isomorphic, and consequently, we refer to a recovery graph associated with arbitrary generator matrix of a Simplex code as the recovery graph of the Simplex code. The first step is to show that the recovery graph of a Simplex code is bipartite. (See Figure 7 for an example of the $[7, 3]$ Simplex code.) We note that has been shown in [1], [56], [58]. We present a brief proof for completeness.

Lemma 3 (Structure of the Recovery Graph for Simplex Codes): For a $[2^k-1, k]$ Simplex code with recovery graph Γ_k, the following holds:

1) Γ_k is bipartite.
2) Each vertex of Γ_k has degree k where each edge corresponds to a recovery set of a different object.
3) The 2^{k-1} vertices of Γ_k that correspond to the odd weight columns of G_k form a minimal vertex cover.

Proof: Recall that a generator matrix G_k of the k dimensional Simplex code consists of all non-zero length-k binary vectors. Let us label each of the $2^k - 1$ vertices of its recovery graph Γ_k with a length-k non-zero binary vector. In addition, let us label each of the additional vertices for systematic columns by the length-k zero vector. Edges in Γ_k correspond to recovery sets of size two. Therefore, two vertices of Γ_k are connected iff the Hamming distance between their labels is one. The lemma immediately follows from this observation. (The bipartite structure of Γ_3 can be observed in Figure 7(b) and Figure 13.) □

We use the above lemma to characterize the rate region of Simplex codes in the following theorem.

Theorem 3: The service rate region of the $[2^k - 1, k]$ Simplex coded system with $\mu = 1$ consists of all demand vectors $(\lambda_1, \cdots, \lambda_k)$ such that $\lambda_1 + \lambda_2 + \cdots + \lambda_k \leq 2^{k-1}$.

Proof: Consider first the non-overlapping recovery sets of size at most two. Let Γ_k be the corresponding recovery graph. Consider an arbitrary demand vector $(\lambda_1, \cdots, \lambda_k)$ such that $\lambda_1 + \cdots + \lambda_k \leq 2^{k-1}$. Assign weight $\lambda_i/2^{k-1}$ to each edge in Γ_k that corresponds to a recovery group of object i. Note that this assignment forms a valid fractional matching (cf. Lemma 3). It follows from Lemma 3 that the vertex cover number of Γ_k is 2^{k-1}, and thus, by Proposition 2, no demand vector $(\lambda_1, \cdots, \lambda_k)$ can be served so that $\lambda_1 + \lambda_2 + \cdots + \lambda_k > 2^{k-1}$.

Next, we show that larger (overlapping) recovery sets of size three do not increase the service rate region. To show this, let us add hyperedges to Γ_k corresponding to recovery sets of size greater than two. Note that 2^{k-1} vertices with odd number of 1's also cover all the hyperedges. Indeed, if this is not the case, there must be a recovery set consisting of servers corresponding to vertices each having an even Hamming weight. Clearly, this is not possible, since the labels of vertices in a recovery set must add to a unit vector. □

It is worth noting two interesting observations. First, it well-known that the codewords of a $[2^k-1,k]$ Simplex code form a simplex in the Hamming space of binary length-(2^k-1) vectors. Interestingly, the service rate region of a $[n=2^k-1,k]$ Simplex code is a $(k-1)$-dimensional simplex in $\mathbb{R}^n$ defined as $\sum \lambda_i \leq 2^{k-1}$, $\lambda_i \geq 0$, $i \in [k]$. As we will see in the next section, looking at codes through the lens of finite geometry enables us to characterize the service rate region of first-order Reed-Muller codes.

Second, from the achievability proof, one can observe that when a server completes a request, it simply starts serving the next request in the queue (if any). A natural question is how many users can be *simultaneously* served by the Simplex-coded system in parallel? This is especially important for scenarios when each user occupies the entire bandwidth of the server. As we discuss in Sec. VIII-B4, this question motivates us to introduce the notion of asynchronous service rate region.

2) MDS Codes: We show how a graph representation of MDS codes allows one to obtain bounds on the service rate region. In a system using an $[n,k]$ systematic MDS code, a data object can be recovered from its systematic copy or from any k of the remaining servers. In other words, the recovery sets of an MDS code are of size either one or k (see Figure 7(a) for the recovery graph of a systematic $[4,2]$ MDS code).

We represent an MDS code using a hypergraph. Note that a is a hypergraph any subset of vertices may be joined by an edged, referred to as a hyperedge, rather than a pair of vertices as in graphs (see, e.g., [57, Chapter 7]). Specifically, given an $[n,k]$ MDS code with a generator matrix G, we form the recovery hypergraph Γ_G such that it contains a vertex for each column of G and a hyperedge for every recovery set. We label every hyperedge of Γ_G with the data symbol whose recovery set it is associated with. For each systematic column of G, we add $k-1$ additional vertices to Γ_G, and connect them with a hyperedge labeled with the corresponding symbol. As we see next, the recovery graph of an MDS code has a specific structure.

Lemma 4 (Structure of the Recovery Graph for MDS Codes): For an $[n,k]$ MDS code with generator matrix G, the following holds.

1) If G has no systematic columns, then Γ_G is a complete hypergraph on n vertices with k parallel hyperedges connecting every k-subset of vertices.
2) If G is systematic, then Γ_G has $n+k(k-1)$ vertices with hyperedges of size k.

The above lemma allows us to obtain bounds on the service rate region of MDS codes as follows.[4]

Proposition 3: For a system using an $[n,k]$ MDS code with no systematic nodes and $\mu=1$, the service rate region is the set of all request vectors $(\lambda_1,\cdots,\lambda_k)$ satisfying $\sum_{i=1}^{k}\lambda_i \leq n/k$. For a system using a systematic $[n,k]$ MDS code and $\mu=1$, the service rate region lies inside the region described by $\sum_{\mathcal{I}\subseteq\{1,\ldots,k\}, \quad i\in\mathcal{I}} \lambda_i \leq k+\frac{|\mathcal{I}|}{k}(n-k)$.

Proof: Consider an $[n,k]$ MDS code with no systematic symbols. For any $(\lambda_1,\cdots,\lambda_k)$ in its service rate region, there must exist a fractional matching in hypergraph Γ_G such that the sum of the weights of hyperedges of label i is λ_i by Proposition 1. Obtain a hypergraph Γ'_G from Γ_G by collapsing parallel hyperedges connecting every k-subset of vertices into one hyperedge and assign its weight to be the sum of the weights of the parallel hyperedges. From Lemma 4, Γ'_G is a complete hypergraph on n vertices with each hyperedge of cardinality k (which is known as a k-uniform hypergraph). Note that any fractional matching in Γ_G induces a valid fractional matching in Γ'_G since the sum of the weights on hyperedges incident to any vertex does not change. Then, the converse and the achievability follow by noting that, for a complete k-uniform hypergraph on n vertices, the fractional matching number is n/k (see, e.g., [59]).

The proof of the upper bound for a systematic code essentially follows the same steps as above. Consider the case when, for some $\mathcal{I}\subseteq\{1,2,\ldots,k\}$, $\lambda_i>0$ for $i\in\mathcal{I}$ and $\lambda_i=0$ otherwise. Then, by Prop. 1, for $i\in\{1,2,\ldots,k\}\setminus\mathcal{I}$, every hyperedge labeled with λ_i must have zero weight in any fractional matching corresponding to a valid service allocation. Let Γ'_G be the graph obtained from Γ_G by removing all hyperedges labeled λ_i for each $i\in\{1,2,\ldots,k\}\setminus\mathcal{I}$, and deleting the dummy vertices corresponding to the systematic columns $i\in\{1,2,\ldots,k\}\setminus\mathcal{I}$. In other words, after removing the hyperedges labeled λ_i, we delete the resulting independent vertices. Note that any fractional matching in Γ_G corresponding to a valid service allocation is also a fractional matching in Γ'_G. Further, from Lemma 4, it is straightforward to see that the pruned graph Γ'_G contains $N=n+|\mathcal{I}|(k-1)$ vertices. Therefore, the bound follows from Prop. 1 by using the fact that the fractional matching number for a k-uniform hypergraph on N vertices is at most N/k (see, e.g., [59]). □

As an example, consider the non-systematic $[8,2]$ MDS code shown in the second row (in blue) in Figure 3. The corresponding recovery graph is the complete graph on 8 vertices. As shown in the proposition, the service rate region is the simplex $\lambda_1,\lambda_2\geq 0$, $\lambda_1+\lambda_2\leq 4$, depicted in blue in Figure 3.

For an example of a systematic MDS code, consider Figure 7 (a). It shows a $[4,2]$ code along with the corresponding recovery graph having 6 vertices. As per Proposition 3, the sum of arrival rates $\lambda_1+\lambda_2\leq 3$. Now, consider the case that only a is being requested, i.e., $\lambda_2=0$. Then, we prune the recovery graph to remove the edges labeled with b and deleting dummy vertices corresponding vertices. The pruned graph consists of the vertices a and 0_a connected by an edge, and the triangle formed by b, $a+b$, and $a-b$. It is easy to see that the fractional matching number for this graph is at most $5/2$, and thus, $\lambda_1\leq 5/2$.

E. Integral Service Rate Region and Batch Codes

Recall that in the bandwidth model (In Sec. II-B2), each server can concurrently serve only a limited number of data

[4]The bounds obtained here for systematic MDS codes are loose as compared to those obtained in Theorem 1 using the waterfilling algorithm in Section V. Note that the waterfilling algorithm is defined only for systematic codes, whereas recovery (hyper)graphs can be used to analyze non-systematic codes as well.

access requests. In this model, λ_i represents the number of requests for object i that are simultaneously present in the system. Even though this model results in the same allocation problem as in the queuing model (Sec. II-B1), it opens up interesting questions. In particular, consider scenarios wherein each user occupies the entire bandwidth of the server they are accessing. For example, this can happen when users are streaming from low-bandwidth edge devices and each user needs to be served at a specific rate. Motivated by these scenarios, we introduce the notion of integral service rate region defined as follows.

Definition 7: The integral service rate region of an $[n, k]$ coding scheme is a set of demand vectors $(\lambda_1, \cdots, \lambda_k)$ for which there exists a valid allocation $\{\lambda_{i,j} : 1 \leq i \leq k, 1 \leq j \leq t_i\}$ satisfying (2), (3) and (4) such that each $\lambda_{i,j}$ is an integer.

From the definition of the integral service rate region and Proposition 1 it follows that a system using an $[n, k]$ code with a generator matrix G contains a demand vector $(\lambda_1, \lambda_2, \cdots, \lambda_k)$ in its integral service rate region if and only if there exists an integral matching in the recovery graph Γ_G such that the weights on the edges with label i sum to λ_i. An open problem associated with this observation is discussed in Section VIII-B3.

Next, we show that, if the integral service rate region of a coding scheme contains a specific region (in particular, all demand vectors $(\lambda_1, \cdots, \lambda_k)$ such that $\sum_{i=1}^{k} \leq t$ for a positive integer t), then the coding scheme must be a *batch code* [26] – a well-known class of codes in computer science.

Batch codes (in particular, multiset batch codes) are designed to simultaneously serve a certain number of requests (each asking for one object)

such that the worst-case maximal load on the system as well as the total amount of used storage are minimized [26]. In the simplest form (called *primitive* batch codes), k data objects are encoded into n objects, which are distributed among n servers (one object per server). The encoding should be such that an arbitrary subset (or batch) of t objects can be decoded by simultaneous reads from a (sub)set of the servers. The formal definition of primitive multiset batch codes is as follows (see, e.g., [26], [60], [61]).

Definition 8: An (n, k, t) (primitive multiset) batch code over $\mathbb{F}_q$ encodes k objects $x_1, \cdots, x_k \in \mathbb{F}_q$ into n objects $c_1, \cdots, c_n \in \mathbb{F}_q$ in such a way that for any multiset $i_1, \cdots, i_t \in [k]$, there is a partition of the servers into subsets $S_1, \cdots, S_t \subseteq [n]$ such that each object x_{i_j}, $j \in [t]$, can be recovered by downloading (at most) one object from each server in S_j.

Next, we show that the service rate region problem can be seen as a generalization of the (primitive multiset) batch code problem.

Proposition 4: The integral service rate region of a storage system using an $[n, k]$ code with $\mu = 1$ includes all demand vectors $(\lambda_1, \cdots, \lambda_k)$ such that $\sum_{i=1}^{k} \lambda_i \leq t$ if and only if the code is an (n, k, t) batch code.

Proof: Consider a demand vector $(\lambda_1, \cdots, \lambda_k)$ such that each λ_i is a non-negative integer and $\sum_{i=1}^{k} \lambda_i \leq t$. It should be noted that any such demand vector can be considered as a multiset of size $\sum_{i=1}^{k} \lambda_i$ (which is at most t), where element i is repeated λ_i times. The proposition 4 then follows directly from the definitions of the integral service rate region and the multiset primitive batch codes. □

Batch codes are related to a class of codes designed for private information retrieval (PIR) [62]. The key property of PIR codes [61], [63] is that they have a number of disjoint recovery sets. Specifically, a binary $[n, k]$ code is called a t-server PIR code if for every $i \in [k]$, there exist a partition of the servers into subsets $S_1, \cdots, S_t \subseteq [n]$ such that the object x_i can be recovered by downloading (at most) one object from each server in S_j, for all $j \in [t]$. The integral service rate region for PIR codes immediately follows from Proposition 4 as follows. The integral service rate region of storage system using an $[n, k]$ code with $\mu = 1$ includes all demand vectors $(t \cdot e_1, \ldots, t \cdot e_k)$, $t \in \mathbb{N}$, if and only if the code is a t-server PIR code.

F. Summary

In this section, we proposed a graph representation to capture recovery sets of a linear code, and showed that the service rate allocation problem for a given linear code is equivalent to the fractional matching problem on the recovery graph associated with the code. This enabled us to characterize the service rate region for binary Simplex codes. A natural future direction is to analyze the service rate region for non-binary Simplex codes using the graph-based techniques. We also introduced the notion of integral service rate region, where allocations are constrained to be integers. We proved that the problem of characterizing an integral service rate region can be viewed as a generalization of the problem of designing primitive multiset batch codes. Exploring connections between the general batch codes and the problem of (integral and general) service rate region is an interesting future direction.

VII. Service Rate Region Using Geometry

Here, we look at the service rate problem through a geometric point of view. The problem of characterizing the service rate region of a code is a linear constrained optimization problem. The geometric approach is a powerful technique for addressing this problem that provides upper bounds on the sum of each subset of arrival rates in any demand vector $(\lambda_1, \cdots, \lambda_k)$ that can be served by a linear code. That is, using this approach, one can obtain a finite set of half-spaces (upper bounds) whose intersection encompasses the service rate region of a given linear storage scheme. In this section, we first discuss a geometric view of a storage scheme. We then give a brief description of the approach, and finally, using two examples, we explain how the service rate regions of the binary first order Reed-Muller codes and binary Simplex codes are obtained by the geometric technique. For a formal description and more details, see [33], where this approach is introduced.

A. Geometric Description of Storage Schemes

The projective space of dimension $k-1$ over $\mathbb{F}_q$, denoted as $\mathrm{PG}(k-1, q)$ is the set of k-tuples of elements of $\mathbb{F}_q$, not all

zero, under the equivalence relation given by $(x_1, \ldots, x_k) \sim (\alpha x_1, \ldots, \alpha x_k)$, $\alpha \neq 0$, $\alpha \in \mathbb{F}_q$.

Consider a generator matrix G of a linear $[n,k]_q$ code. Columns of G are vectors in $\mathbb{F}_q^k$. Since in the service rate region problem, we are only concerned with linear dependence of the columns of G, we consider G as a geometric object in the projective space $\mathrm{PG}(k-1,q)$. Each column $\mathbf{g}_i$ of G determines a point, and all columns that are scalar multiples of $\mathbf{g}_i$ are the same point in $\mathrm{PG}(k-1,q)$. Therefore, the columns of G form a multiset of points in $\mathrm{PG}(k-1,q)$. We denote this n-multiset by $\mathcal{G}$ and say that it is induced by G [64].

Any 2-dimensional subspace of $\mathbb{F}_q^k$ determines a line in $\mathrm{PG}(k-1,q)$ and any $k-1$ dimensional subspace of $\mathbb{F}_q^k$ determines a hyperplane in $\mathrm{PG}(k-1,q)$. For details, see [65], [66]. Note that since we are working over finite fields, hyperplanes consist of a finite number of points.

For example, consider the binary $[7,3]$ Simplex code. The columns of its generator matrix are the seven non-zero vectors of $\mathbb{F}_2^3$, and the seven points in the projective space $\mathrm{PG}(2,2)$. Figure 7-b shows the corresponding 7-multiset, known as the Fano plane. Since $k=3$, the 7 lines of the $\mathrm{PG}(2,2)$ are also the hyperplanes of this 2-dimensional projective space.

Consider next the first storage scheme in Figure 3 with four replicas of a and four replicas of b. In the 8-multiset induced by this scheme, there are only two different points $(1,0)$ and $(0,1)$ and the multiplicity of each is four. Finally, consider the last storage scheme in Figure 3 with three replicas of a, three replicas of b and two independent linear combinations $a+b$ and $a+\alpha b$. In the 8-multiset induced by this scheme, the points are $(1,0)$ and $(0,1)$, each with multiplicity three and $(1,1)$ and $(1,\alpha)$ each with multiplicity one.

B. Geometric Interpretation of the Service Rate Region Problem

The following proposition plays a key role in deriving upper bounds on cumulative rates that can be simultaneously served by linear codes. In particular, it provides an upper bound on the sum of each subset of rates in any demand vector $(\lambda_1, \cdots, \lambda_k)$ in the service rate region of the system. In other words, it shows that the service rate region lies inside a region defined as the intersection of a finite number of half spaces (derived upper bounds).

Proposition 5: For a system using an $[n,k]$ code with n-multiset $\mathcal{G}$ in $\mathrm{PG}(k-1,q)$, consider any arbitrary demand vector $(\lambda_1, \cdots, \lambda_k)$ in its service rate region. Then, it holds that

$$\sum_{\substack{i \in \mathcal{I} \\ \mathcal{I} \subseteq \{1,\ldots,k\}}} \lambda_i \leq \mu \cdot |\mathcal{G} \setminus \mathcal{H}|$$

where $\mathcal{H}$ is a hyperplane of $\mathrm{PG}(k-1,q)$ not containing $\mathbf{e}_i$ for all $i \in \mathcal{I}$.

Proof: If a vector $(\lambda_1, \ldots, \lambda_k)$ is in the service rate region, then it holds that simultaneously the cumulative request rate of $\sum_{\substack{i \in \mathcal{I} \\ \mathcal{I} \subseteq \{1,\ldots,k\}}} \lambda_i$ for all objects $i \in \mathcal{I}$ is served by the system. On the other hand, since the hyperplane $\mathcal{H}$ does not contain any unit vector $\mathbf{e}_i$ for all $i \in \mathcal{I}$, the points contained in $\mathcal{H} \cap \mathcal{G}$, on their own, are not able to generate $\mathbf{e}_i$ for all $i \in \mathcal{I}$. Thus, for each $i \in \mathcal{I}$, whatever the used recovery sets for object i are, in order to serve the request rate λ_i for object i, some points outside of $\mathcal{H}$ with the cumulative service rate of (at least) λ_i must be used. Thus, in order to satisfy the cumulative request rate of $\sum_{\substack{i \in \mathcal{I} \\ \mathcal{I} \subseteq \{1,\ldots,k\}}} \lambda_i$ for all objects $i \in \mathcal{I}$, the cumulative service rate of the points in $\mathcal{G}$ that are outside of $\mathcal{H}$ must be at least $\sum_{\substack{i \in \mathcal{I} \\ \mathcal{I} \subseteq \{1,\ldots,k\}}} \lambda_i$. □

C. Using Geometric Approach to Characterize Service Rate Regions

The geometric approach equipped with Proposition 5 can be used to obtain the service rate region of Simplex and Reed-Muller codes. The service rate region of the binary $[2^k-1,k]$ Simplex code and binary $[2^{k-1},k]$ first order Reed-Muller code are given in [33]. To illustrate the method, we here provide two examples: binary $[7,3]$ Simplex and $[8,4]$ first order Reed-Muller codes.

1) Simplex Codes: In this section, as an example, we show how the service rate region of the $[7,3]$ Simplex code is characterized using the geometric approach. Consider a storage system using the $[7,3]$ Simplex code. Without loss of generality, assume that $\mu = 1$. Let (x_1, x_2, x_3) denote a generic non-zero vector in $\mathbb{F}_2^3$. Observe that the hyperplane $\mathcal{H}$ given by $\sum_{i=1}^3 x_i = 0$ (namely, the hyperplane containing the points $(0,1,1)$, $(1,0,1)$ and $(1,1,0)$ in the Fano plane depicted in Figure 7-b) does not contain any unit vector $\mathbf{e}_i$, $i \in \{1,2,3\}$. Thus, for any demand vector $\boldsymbol{\lambda} = (\lambda_a, \lambda_b, \lambda_c)$ in the service rate region, applying Proposition 5 results in $\lambda_a + \lambda_b + \lambda_c \leq 4$. The reason is that the hyperplane $\mathcal{H}$ does not contain the points $(0,0,1)$, $(0,1,0)$, $(1,0,0)$ and $(1,1,1)$. Thus, so far we have shown that the service rate region is contained in the polytope $\mathcal{P} = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^3_{\geq 0} : \sum_{i=1}^3 \lambda_i \leq 4 \right\}$. It is interesting to note that, in the recovery graph, that the vertices corresponding to the points outside of $\mathcal{H}$ form a minimal vertex cover (see Figure 7-b).

For the achievability proof, since the service rate region is a convex subset of $\mathbb{R}^3_{\geq 0}$, we only need to show that the vertices of the polytope $\mathcal{P}$, i.e., $(0,0,0)$, $(4,0,0)$, $(0,4,0)$ and $(0,0,4)$, are in the service rate region of this storage system. To see that, we observe that there are four disjoint recovery sets for each data object (see Figure 2 for object a), and thus the request rate of 1 can be assigned to each of these recovery sets without violating the node capacity constraints.

2) First-Order Reed-Muller Codes: To show a sketch of the proof in characterizing the service rate region of the first order Reed-Muller code, we consider a non-systematic $[8,4]$ first order Reed-Muller code. Consider a system where four objects a, b, c, and d are stored across 8 servers using the first order Reed-Muller code $\mathrm{RM}_2(1,3)$ with a non-systematic generator matrix as:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

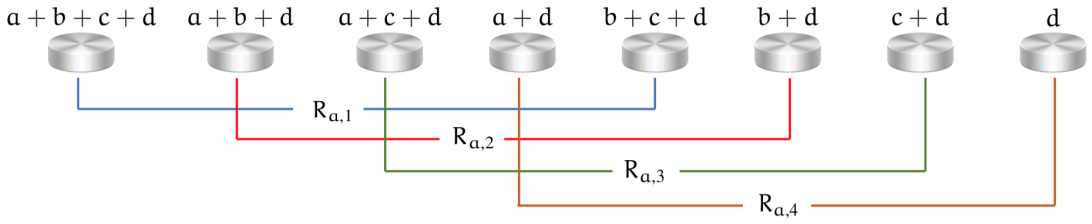

Fig. 8. Recovery sets for data object a in the $[8, 4]$ Reed-Muller code.

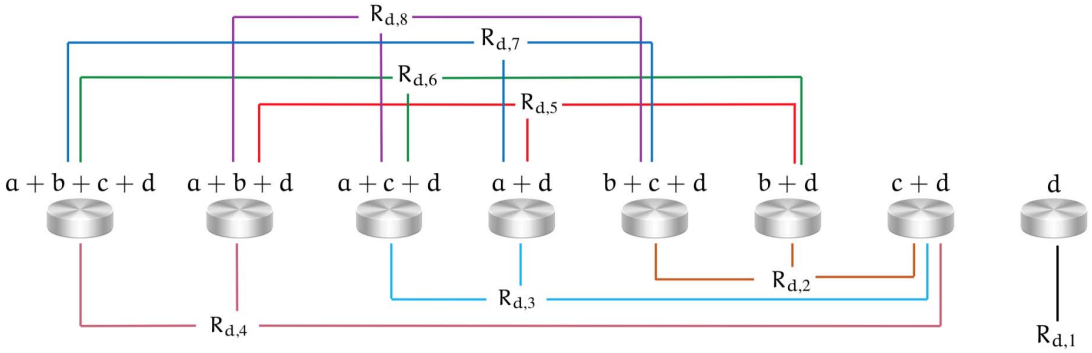

Fig. 9. Recovery sets for data object d in the $[8, 4]$ Reed-Muller code.

which encodes $[a, b, c, d]$ into $[a+b+c+d, a+b+d, a+c+d, a+d, b+c+d, b+d, c+d, d]$. The recovery sets for object a are shown in Figure 8. The recovery sets for objects b and c can be obtained similarly to those for a. The recovery sets for object d are shown in Figure 9.

Let $(x_1, \ldots, x_4)$ be a non-zero vector in $\mathbb{F}_2^4$. Observe that the hyperplane $\mathcal{H}$ given by $\sum_{i=1}^{4} x_i = 0$ does not contain any unit vector $\mathbf{e}_i$, $i \in [4]$. The hyperplane $\mathcal{H}$ does not contain the 4 column vectors $(1, 1, 0, 1)$, $(1, 0, 1, 1)$, $(0, 1, 1, 1)$ and $(0, 0, 0, 1)$ of the generator matrix. Thus, for any demand vector $\boldsymbol{\lambda} = (\lambda_a, \lambda_b, \lambda_c, \lambda_d)$ in the service rate region, applying the Proposition 5 results in the constraint below

$$\lambda_a + \lambda_b + \lambda_c + \lambda_d \leq 4. \tag{12}$$

On the other hand, we know that the unit vector $\mathbf{e}_i$ for all $i \in \{1, 2, 3\}$ is not a column of the generator matrix which means that files a, b, and c do not have any systematic recovery sets. Thus, for files a, b, and c, the cardinality of all recovery sets is at least two, and the minimum system capacity utilized by λ_i for $i \in \{a, b, c\}$ is $2\lambda_i$. For file d, since all columns of the generator matrix have one in the last row, the cardinality of every recovery set is odd. Hence, for file d, the unit vector $\mathbf{e}_4$, which is a column of G, forms a recovery set of cardinality one, while all other recovery sets have cardinality at least three. Thus, the minimum system capacity utilized by λ_d for $\lambda_d \leq 1$ is λ_d and for $\lambda_d \geq 1$ is $1 + 3(\lambda_d - 1) = 3\lambda_d - 2$. Since the system has 8 servers, each of service capacity 1, based on the capacity constraints, the total capacity utilized by the requests for download must be at most 8. Thus, any vector $(\lambda_a, \lambda_b, \lambda_c, \lambda_d)$ in the service region must satisfy:

$$\begin{cases} 2(\lambda_a + \lambda_b + \lambda_c) + \lambda_d \leq 8 & \text{for } \lambda_d \leq 1 \\ 2(\lambda_a + \lambda_b + \lambda_c) + 3\lambda_d - 2 \leq 8 & \text{for } \lambda_d \geq 1 \end{cases} \tag{13}$$

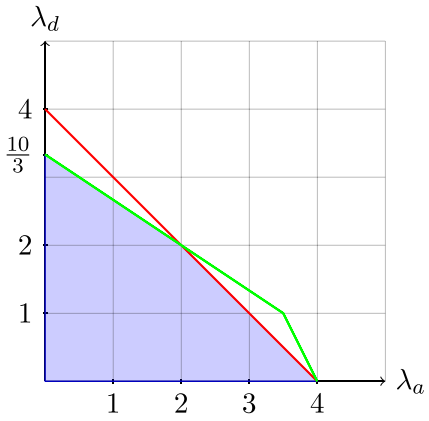

Fig. 10. Service rate region of the $[8, 4]_2$ first order Reed-Muller code in $\lambda_a - \lambda_d$ plane with $\lambda_b = \lambda_c = 0$ where the constraints (12) and (13) are respectively shown with the red line and the green line.

We showed that the service rate region lies inside the polytope $\mathcal{T} = \{\boldsymbol{\lambda} \in \mathbb{R}^4_{\geq 0} : \boldsymbol{\lambda} \text{ satisfies } (12), (13)\}$. Suppose $\lambda_b = \lambda_c = 0$. Figure 10 depicts the service rate region of this storage scheme in the $\lambda_a - \lambda_d$ plane wherein (12) and (13) are respectively shown with the red line and the green line.

For the achievability proof, one needs to provide constructions only for the vertices of polytope $\mathcal{T}$ in $\lambda_a - \lambda_d$ plane. The demand vector $(\lambda_a, \lambda_b, \lambda_c, \lambda_d) = (4, 0, 0, 0)$ can be achieved by assigning the request rate of 1 to each of the 4 disjoint recovery sets of file a shown in Figure 8. For the $(\lambda_a, \lambda_b, \lambda_c, \lambda_d) = (2, 0, 0, 2)$, the $\lambda_a = 2$ can be served by assigning the request rate of 1 to each of the recovery sets $(b+d, a+b+d)$ and $(b+c+d, a+b+c+d)$, and $\lambda_d = 2$ can be satisfied by assigning the request rate of 1 to the systematic recovery set (d), and the request rate 1 to the recovery set

$(a+d, c+d, a+c+d)$ of file d. For the demand vector $(\lambda_a, \lambda_b, \lambda_c, \lambda_d) = (0, 0, 0, \frac{10}{3})$, the $\lambda_d = \frac{10}{3}$ can be served without violating the node capacity constraints by assigning the request rate of 1 to the systematic recovery set (d), and the request rate of $\frac{1}{3}$ to each of the 7 recovery sets of size 3 for file d, depicted in Figure 9.

D. Summary

In this section, we proposed a geometric technique for addressing the problem of characterizing the service rate region of a given linear storage scheme without explicitly listing the set of all possible recovery sets. By leveraging the proposed geometric technique, initial steps were taken towards deriving upper bounds on the service rate regions of some parametric classes of linear codes. In particular, upper bounds on the service rate regions of the binary first order Reed-Muller codes and binary simplex codes, as two classes of codes which are important in both theory and practice, were derived. Then, it has been shown that how the derived upper bounds can be achieved. Utilizing the geometric technique to investigate the service rate regions of other common coding schemes such as MDS codes, second order Reed-Muller codes, non-binary Reed-Muller codes, and non-binary simplex codes are amongst the most natural future directions.

VIII. Ongoing and Open Problems

This paper presented several initial fundamental results and techniques concerning the service rate region of a distributed coded storage system. We summarised the findings at the end of each section. Table I outlines the results concerning particular code classes. The table also indicates the limitations of this early work. Thus, many direct extensions and continuations within the described thrusts are apparent. Some compelling problems of varying degrees of difficulty include, e.g., extending the results to other classes of codes.

There are many related problems just outside of the main scope of the paper. This section presents a summary of connected ongoing and open problems along two threads: 1) performance analysis of storage schemes, which requires queueing and combinatorial optimization expertise, and 2) designing the storage schemes to maximize the service rate region, which requires information and coding theory expertise. Since each of these problems would greatly benefit from jointly solving both the performance analysis and code design problems, we believe that these directions will bridge deeper connections between these communities.

A. Performance Analysis and Networking Problems

1) The Coverage of a Rate Region: The service rate region of a given storage scheme *covers* the set of achievable demand vectors $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$. Since this is a multi-dimensional region, we need a way to map it to a single scalar metric that can be used to objectively compare two service rate regions. One candidate is the volume of the k-dimensional region. Instead, we propose a more natural candidate metric – we consider a probability distribution $f_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda})$ of demand vectors and measure the fraction of requests that are *covered* by the service rate region. The coverage or the covered mass of a rate region $\mathcal{S}$ is defined as

$$M(\mathcal{S}, f_{\boldsymbol{\Lambda}}) = \int_{\boldsymbol{\lambda} \in \mathcal{S}} f_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda}) d\boldsymbol{\lambda}. \tag{14}$$

For example, in Figure 11 we compare the coverage of the replication and MDS coding storage schemes for $k = 2$ objects stored on $n = 4$ servers. The heatmap shows the probability distribution $f_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda})$ of the demand vectors, where one of a or b is likely to be in high demand, but both objects are not in high demand simultaneously. The MDS coded system has a coverage of $M_{\text{MDS}} = 0.91$, which is larger than the coverage $M_{\text{Rep}} = 0.82$ of the replicated system.

Analyzing the service rate regions of well-known classes of codes for typical demand or content popularity distributions such as the Zipf distribution is an open performance analysis problem. It will provide valuable insights that can be used in designing codes that provide maximum coverage with the minimum number of nodes.

2) Analyzing the Cost of Serving Requests: Serving a download request collaboratively by two or more servers (some of which store encoded objects) occupies more system resources than serving it at a single server. For instance, accessing a from b and $a+b$ requires downloading two objects to access one object. An interesting research direction is to study this cost quantitatively. In the following we formally propose the cost associated with a given storage scheme.

Let us define the *service cost* of a single request as the number of objects that are downloaded in order to satisfy the request, which is the size of the corresponding recovery group R_j. We define the normalized service cost $C(\boldsymbol{\lambda})$ of a demand vector $\boldsymbol{\lambda}$ as the cumulative transfer rate required by the servers to serve this demand, divided by the sum of the elements of $\boldsymbol{\lambda}$, that is,

$$C(\boldsymbol{\lambda}) = \frac{\sum_{i=1}^{k} \sum_{j=1}^{t_i} |R_j| \cdot \lambda_{i,j}}{\sum_{i=1}^{k} \sum_{j=1}^{t_i} \lambda_{i,j}}, \tag{15}$$

where $\lambda_{i,j}$ is the portion of the request rate λ_i allocated to the j^{th} repair group R_j, whose size is $|R_j|$. The service cost $C(\boldsymbol{\lambda})$ represents the amount of data that is downloaded per request, and it depends on the underlying coding scheme and the request allocation scheme used to split requests across recovery groups. For example, if we use replication coding, then the size of each recovery group $|R_j| = 1$, and thus, $C(\boldsymbol{\lambda}) = 1$. With erasure coding, a request may need to download two or more coded objects to recover one data object, and thus we will incur a higher cost. Consider again the example of serving the demand with $\lambda_a = 1.5\mu$ and $\lambda_b = 0.5\mu$ would cost 2μ in the replicated system (a, a, b, b) and 4μ in the MDS coded system $(a, b, a+b, a+2b)$. Normalizing these by the demands, we get the cost $C(\boldsymbol{\lambda}) = 1$ for the replicated system and $C(\boldsymbol{\lambda}) = 1.6$ for the MDS-coded system. Figure 12 shows a heat map of the *normalized service cost* for all demand vectors within the service rate region of two systems.

An open question for future research is to compare the expected costs of different service rate regions. We can again

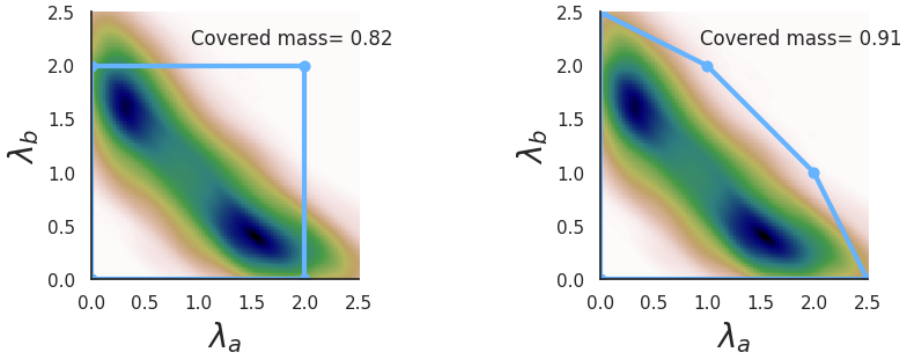

Fig. 11. The demand rate distribution shown as a heatmap for (left) replication system (a, a, b, b) and (right) MDS coded system $(a, b, a+b, a+2b)$. Blue lines show the boundary of system's service rate region. Covered mass is the cumulative probability mass for the demand vectors that lie within the system's service capacity region.

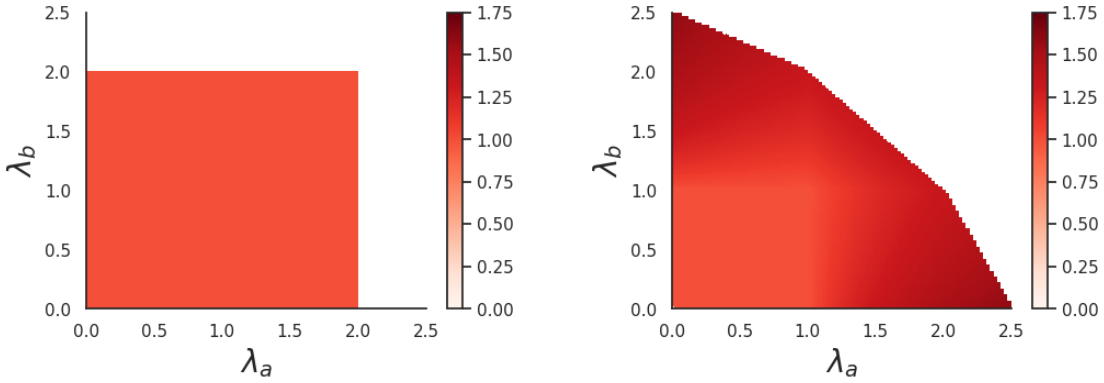

Fig. 12. The normalized service cost $C(\boldsymbol{\lambda})$ shown as a heat map for (left) replicated system (a, a, b, b) and (right) MDS system $(a, b, a+b, a+2b)$. The normalized service cost at each point in the service rate region is defined as the number of bits that need to be downloaded per data bit. For the replicated system, the cost is $C(\boldsymbol{\lambda}) = 1$, whereas for the MDS system that cost varies from 1 to 1.6 depending on the skewness of the object demands (λ_a, λ_b).

consider a probability distribution $f_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda})$ of demand vectors and measure the fraction of requests that are *covered* by the service rate region. Then the expected service cost of a rate region $\mathcal{S}$ is $C(\mathcal{S}, f_{\boldsymbol{\Lambda}}) = \int_{\boldsymbol{\lambda}\in\mathcal{S}} C(\boldsymbol{\lambda}) f_{\boldsymbol{\Lambda}}(\boldsymbol{\lambda}) d\boldsymbol{\lambda}$. We conjecture that for imbalanced demand distributions, where several objects are not in high demand at the same time, the coded systems will incur little additional cost, but will have higher coverage.

Instead of measuring the service cost in terms of the amount of data accessed to serve one request as captured by $C(\boldsymbol{\lambda})$, an alternate metric is to consider the total computing time spent serving each request, as considered in [12], [25], [67]–[69]. These papers identify regimes where coded data access or computing incurs a lower total computing time than replicated systems.

3) Latency Analysis of Coded Systems: In this paper, we focused on the service rate region, that is, the demand vectors $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ that can supported by the system while ensuring that the total request rate assigned to each server does not exceed its capacity μ. We did not consider the delay experienced by each request. The request splitting policies proposed in this paper such as the water-filling algorithm are throughput-optimal, and not necessarily delay-optimal. Open problems for future research include 1) analyzing the latency experienced by a user accessing data from coded distributed system and 2) designing delay-optimal policies for splitting requests across recovery groups, which is much harder than designing throughput-optimal policies.

The first problem of analyzing the latency of accessing content from coded distributed systems, specifically MDS and availability coded systems, has been previously considered in [12]–[14], [22], [70], [71]. However, these works consider redundant requests, that is, each request is sent to multiple recovery groups and the request is considered served when the data is successfully accessed from any one of the groups. They show that the resulting system is a generalized fork-join queueing system, whose analysis is a famously hard problem in queueing theory [72]–[74], and find bounds on the expected latency. Besides the fork-join queue model, several approaches like block-t and probabilistic scheduling have been developed to remove some of the key assumptions in latency analysis [75], [76]. Analyzing the latency of hybrid systems that use a combination of replication and erasure coding, both with and without sending redundant requests to multiple recovery groups, as proposed in this paper is a pertinent open problem.

The second problem of designing delay-optimal policies, a highly challenging problem, requires striking the perfect balance between using several recovery groups in parallel to serve a set of incoming requests and queueing more requests for sequential processing at smaller recovery groups. Our

throughput-optimal allocation policies such as water-filling are biased towards the latter strategy as they give higher priority to sending requests to smaller recovery groups. An alternative approach is to design policies that perform the best possible *load balancing* of requests across the nodes, that is, minimize the cumulative request rate assigned to the maximally loaded node, as recently considered in [77]. Such load-balancing strategies can give better latency performance than the throughput-optimal strategies considered in this paper.

4) Expanding the Service Rate Region via Redundant Requests: In this paper we assume that each data access request is sent to exactly one of its recovery groups. Instead, assigning requests to more than one recovery groups, waiting for any one copy to be downloaded and canceling the outstanding requests can reduce the latency experienced by that request. But too much redundancy can increase the waiting time in queue for subsequent requests. Several recent works study queueing systems with redundancy [12], [17], [18], [67], [70], [71], [78]–[82]. Some of these works [12], [14], [78] observe that when the distribution of the service time of each request is heavier than an exponential distribution (in particular, new-shorter-than-used distributions such as the hyper-exponential distribution) then redundancy expands the achievable rate region beyond the sum capacity of individual servers. That is, a system of n servers with capacity μ requests per time each can support a demand higher than $n\mu$. This non-intuitive phenomenon has been recently studied in the context of replication of jobs in computing systems in [68], [83]–[85]. However, understanding the expansion of the service rate region due to heavy-tailed and new-shorter-than-used service times for erasure coded distributed systems such as those considered in this paper is an open question. The effect of heavy-tailed distributions on the latency of jobs with many parallel tasks, where straggling tasks are replicated is also previously studied in [69], [86]–[88], albeit without considering queueing of tasks.

B. Coding Theory and Data Allocation Problems

Next we discuss some open problems from the perspective of code design or data allocation to achieve the best possible service rate region with minimum number of storage nodes.

1) Designing Codes to Achieve a Rate Region With Desired Properties: In Section III-B, we introduced the problems of designing the coded storage schemes to maximize the volume of the service rate region with a given number of servers or cover a given service rate region with the minimum number of servers. Note that the service rate regions of two generator matrices G and G' of the same linear code might not be the same. Depending on the application, one may be interested in using a particular code with some desired properties. Then, these problems can be interpreted as finding the best generator matrix of a code with respect to the service rate region. Also, depending on the application, a metric different from the volume of the region might be of interest to a system designer. An alternative metric of practical interest is to optimize for a given number of servers n and demand distribution $f_{\mathbf{\Lambda}}$ is the fraction $M(\mathcal{S}, f_{\mathbf{\Lambda}})$ of the demand distribution $f_{\mathbf{\Lambda}}$ covered by the region, as defined in (14). Using this metric can present some interesting challenges in designing the coding schemes. For example, consider two objects a and b whose demands are negatively correlated, that is, when a is popular, b is not popular, and vice-versa. Then encoding a and b together to form the coded object $a + \alpha b$ will give better coverage than encoding $a+\alpha c$, where c's demand is positively correlated with a, that is, c and a becomes popular at the same time.

Another design objective can be to achieve the tail latency $\Pr(T > \tau) \leq \delta$ for a given demand distribution $f_{\mathbf{\Lambda}}$, deadline τ and tail probability δ with the minimum number of servers. For commonly observed demand distributions in which only a few objects are in high demand simultaneously, we expect that the erasure coded storage schemes to significantly outperform the replication based storage schemes.

2) Data Striping Across Multiple Nodes and Multiple Objects Per Node: For simplicity in introducing the new concept service rate region, in this paper, we treat each data object as an atomic unit such that the k data objects are used as k information symbols when creating encoded versions of the objects. We also assume that each of the n servers has the capacity to store exactly one data object. Distributed storage systems often employ data striping or sub-packetization [2], that is, dividing object a is divided into stripes a_1, a_2, ..., a_s, which are used as source symbols, and encoded and stored across different nodes. Spreading an object across more nodes can allow faster parallel reads of large objects.

However, there are two possible drawbacks of parallelism. First, its impact on service rate region is not straightforward. Request service times might go down super linearly with the reduced data size. That is, if downloading a takes t seconds, downloading each a_i where $i = 1, \ldots, s$ might take longer than t/s. In this case, parallelism might lead to a smaller service rate region. Second, parallel download requires accessing multiple nodes simultaneously. Failure or slowdown at any one of these nodes can bottleneck the data access and increase latency. Removing the assumption of atomicity of each data object and generalizing the concept of service rate region to characterize the rate region of distributed storage with data striping is an open problem for future research.

Even without data striping, each node may store multiple data objects, unlike our assumption that each server has the capacity to store exactly one data object and the entire object is accessed by each request. In this setting, designing optimal storage of objects so as to maximize the service rate region is an interesting open problem. In particular, as observed in [77], minimizing the overlap between recovery groups of different objects could lead to a storage allocation that maximizes the service rate region. One possibility is to use expander graphs to design such storage allocation schemes.

Sub-packetization can also possibly make the system's service rate region larger. Whether it actually gets larger is not obvious. The answer depends on the scaling of the request service times with the object sizes. We elaborate on this connection below with an example. It is worth to note here that, in real storage systems (e.g., Google file system), object sizes are experimentally tuned to a value that is not too small

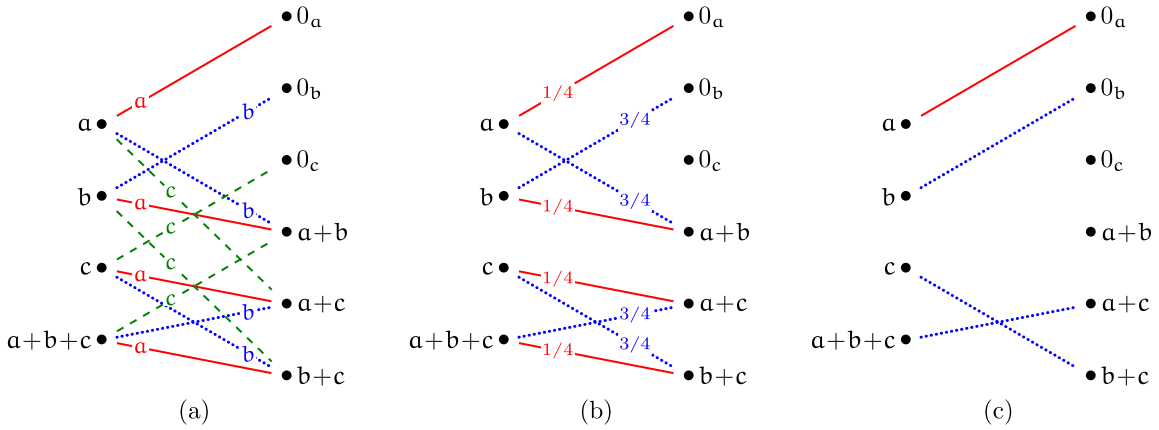

Fig. 13. (a) The recovery graph of the $[7, 3]$ Simplex code. (b) A fractional matching for demand vector $(1, 3, 0)$. (c) An integral matching for demand vector$(1, 3, 0)$.

in order to keep the system level overheads small compared to the actual time spent while fetching data from the nodes.

Recall that we define the node capacity as the maximum number of requests that can be served by a node per second. Suppose now we divide each object into two equal chunks and store them across separate servers. An object request will now be served by splitting it into two chunk requests and then assigning them to the respective servers. Obviously, the time to download a single chunk from a server, t_1, will be less than the time to download an object (two chunks), t_2. Note that, due to the system level overheads, download time is not a linear function of the data size. That is why, we cannot conclude that $t_1 = t_2/2$. Note also that, to download an object, the system now needs to serve two chunk requests instead of one object request. This overall means that downloading objects by fetching chunks from multiple servers can lead to consuming more system capacity than downloading the whole object from a single server. It is therefore not clear whether dividing objects into chunks can lead to larger service rate region.

3) Integral Service Rate Region: We have defined the integral service rate region as a set of demand vectors $(\lambda_1, \cdots, \lambda_k)$ for which there exists a valid allocation[5] $\{\lambda_{i,j} : 1 \leq i \leq k, 1 \leq j \leq t_i\}$ such that each $\lambda_{i,j}$ is an integer (see Sec. VI-E). Recall that $\lambda_i = \sum_{1 \leq j \leq t_i} \lambda_{ij}$ for $1 \leq i \leq k$, and thus, all points $(\lambda_1, \cdots, \lambda_k)$ in the integral service rate region have all integer components. An important open problem asks whether an integer component point $(\lambda_1, \cdots, \lambda_k)$ in the service rate region is always in the integral service rate region, i.e., whether for an integer component point $(\lambda_1, \cdots, \lambda_k)$ in the service rate region, there always exists a valid allocation $\{\lambda_{i,j} : 1 \leq i \leq k, 1 \leq j \leq t_i\}$ such that each $\lambda_{i,j}$ is a non-negative integer. Confirming that this is true tells us (cf. Proposition 4) that an $[n, k]$ code whose service rate region includes the points satisfying $\sum_{i=1}^{k} \lambda_i \leq t$ for some positive integer t is an (n, k, t) batch code. Thus the service rate region problem can be seen as a generalization of the batch code problem. The potentially more general service rate region problem may be easier to solve than the corresponding batch code problem. For example, proving that the binary $[2^k - 1, k]$ Simplex code is a $(2^k - 1, k, 2^{k-1})$ batch code is fairly involved [58], while deriving its service region is straightforward by using either the combinatorial or the geometric techniques, as we did above. Another unexplored direction is using the techniques proposed in this paper to derive the batch properties of binary Hamming and Reed-Muller codes, previously considered in [89]).

Next, we consider an example to illustrate the problem of integral service rate region that we just defined. Also, we want to point out an interesting question concerning the matching in graphs that, to the best of our knowledge, has not been asked before.

Consider the binary $[7, 3]$ Simplex code and its recovery graph as shown in Figure 13-(a). We are interested in serving the demand vector $(1, 3, 0)$. An easy way to see that this vector is in the service rate region is to consider the fractional matching on the recovery graph that assigns weight $1/4$ to all a recovery edges, weight $3/4$ to all b recovery edges, and 0 to all c recovery edges, as shown in Figure 13-(b) and instructed by Theroem 3. Alternatively, we can satisfy this demand with an integral service where file a is downloaded solely from the node storing a, while file b is downloaded from the node storing b and the repair groups c & $b+c$ and $a+c$ & $a+b+c$, as shown in Figure 13-(c). The matching problem asks the following: Given an $[n, k]$ code recovery graph and a matching such that the for object i, the sum of weights on its recovery edges is an integer λ_i, is there an integral matching with λ_i recovery edges for object i, for all $1 \leq i \leq k$?

The answer to this question is yes for the binary Simplex codes. Since these codes are batch codes [58], the claim follows easily from the results in Section VI. Moreover, an algorithm is presented in [56] that takes an integer demand vector $(\lambda_a, \lambda_b, \lambda_c)$ in the service rate region of the $[7, 3]$ Simplex code and produces an integral matching with λ_a a-recovery edges, λ_b b-recovery edges, and λ_c c-recovery edges (see [56, Algorithm 1]). This algorithm can be easily extended to the binary $[15, 4]$ Simplex code, but a generalization to an arbitrary $[2^k - 1, k]$ Simplex code is an open problem.

4) Asynchronous Service Rate Region: In systems serving multiple users, it is natural to ask the following: If a user

[5]An allocation is valid if it satisfies (2), (3) and (4).

leaves the system, can another user interested in downloading different objects take the freed place? This question is of interest, e.g., whenever different queries take different times to process. We refer to storage schemes that support such dynamics as *asynchronous*, following [90] which has introduced this question and the terminology in connection with batch codes.

Consider the $[7, 3]$ Simplex code, and observe that point $(\lambda_a, \lambda_b, \lambda_c) = (1, 3, 0)$ belongs to its service rate region, and can be achieved by assigning the entire $\lambda_a = 1$ to the node storing a, and splitting $\lambda_b = 3$ evenly between the nodes storing b, c & $b + c$, and $a + c$ & $a + b + c$. Suppose that all users downloading object a leave the system. Can then $\lambda_c = 1$ users take their place? Although point $(\lambda_a, \lambda_b, \lambda_c) = (0, 3, 1)$ belongs to the service region, the answer is no because the only available servers in the system are those storing a and $a + b$. Consider again the $[7, 3]$ Simplex code and point $(\lambda_a, \lambda_b, \lambda_c) = (1, 3, 0)$, but this time the demands are split shown in Figure 13-(b), and described in the proof of Theroem 3 for the general case. Note that now the departure of all users downloading object a frees the system to serve any new demand vector as long as it belongs to the service rate region.

There are two natural questions: 1) Are there allocation schemes that are scalable to user departures/arrivals? 2) What is the service rate region of a storage scheme if we require that each point be not only achievable but also achievable by scalable allocations? The latter question was addressed for batched codes in [90], where it was found out that, e.g., Simplex codes can serve any multiset of size 2 in an asynchronous way, as opposed to any multiset of size four without this requirement.

IX. Concluding Remarks

We introduced the service rate region as a new aspect in the design of distributed storage and computing systems. The service rate region of a storage system storing k files is the set of request demand rates $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)$ that can be supported by a set of n servers, each of which has a limited service capacity μ.

Previously considered design considerations for distributed storage include reliability against node failures, repair efficiency, data locality, and latency. Codes that optimize these aspects may not support a large volume of access requests, especially when different objects have different demands. The service rate region can capture this aspect and enable the design of storage schemes that maximize the volume of heterogeneous data access requests that can be satisfied with a minimum number of resources. We highlighted two problems of interest: 1) optimal splitting of the requests for each object across its recovery groups to maximize the service rate region and 2) design of the underlying coding scheme to achieve a service rate region with desired properties.

Through preliminary work on the problem of optimal request splitting, we show how the notion of the service rate region employs diverse mathematical techniques such as water-filling, geometric representations of codes, and combinatorial optimization over graphs. In particular, we characterize the rate regions of maximum distance separable (MDS) codes, Reed-Muller codes, and Simplex codes using three different techniques: water-filling, combinatorial and geometric representations.

Our initial work on the thread of designing coding schemes to maximize the service rate region with a given number of servers provide the novel insight that codes that are a hybrid of replication and coding can achieve the best service rate region. Further exploration of code design for service rate region maximization can help discover fundamental connections with existing classes of codes such as batch codes and availability codes. We hope that the open problems presented in this paper will result in interdisciplinary interactions between the networking and coding theory communities and result in practical insights to boost the service capacity of distributed storage and computing systems.

Acknowledgment

The authors thank Sarah Anderson, Ann Johnston, Gretchen Matthews, Esmaeil Karimi, Carolyn Mayer, and Gala Yadgar for helpful discussions.

References

[1] M. Aktas *et al.*, "On the service capacity region of accessing erasure coded content," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 17–24.

[2] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *ACM SIGMOD Records*, vol. 17, no. 3, pp. 109–116, Jun. 1988.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10, doi: 10.1109/MSST.2010.5496972.

[4] G. Yadgar, O. Kolosov, M. F. Aktas, and E. Soljanin, "Modeling the edge: Peer-to-peer reincarnated," in *Proc. 2nd USENIX Workshop Hot Topics Edge Comput., (HotEdge)*, I. Ahmad and S. Sundararaman, Eds. Renton, WA, USA: USENIX Association, Jul. 2019, pp. 1–11.

[5] M. A. Maddah-Ali and U. Niesen, "Coding for caching: Fundamental limits and practical challenges," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 23–29, Aug. 2016.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 1, Mar. 1999, pp. 126–134.

[7] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Boston, MA, USA: Addison-Wesley, 2002.

[8] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[9] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many matching games for proactive social-caching in wireless small cell networks," in *Proc. 12th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, May 2014, pp. 569–574.

[10] E. Berlekamp, *Algebraic Coding Theory*. New York, NY, USA: McGraw-Hill, 1968.

[11] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 107–144.

[12] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Trans. Modeling Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 1–30, Apr. 2017.

[13] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Proc. Allerton Conf. Commun., Control, Comput.*, Oct. 2012, pp. 326–333.

[14] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, Feb. 2016.

[15] K. V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran, "EC-cache: Load-balanced, low-latency cluster caching with online erasure coding," in *Proc. USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Savannah, GA, USA, 2016, pp. 401–417.

[16] Y. Raaijmakers and S. Borst, "Achievable stability in redundancy systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 3, pp. 1–21, Nov. 2020.

[17] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, Jun. 2015, pp. 347–360.

[18] K. Gardner, S. Zbarsky, M. Harchol-Balter, and A. Scheller-Wolf, "The power of d choices for redundancy," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Sci.*, Jun. 2016, pp. 409–410, doi: 10.1145/2896377.2901497.

[19] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[20] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.

[21] A. Mallick, U. Sheth, G. Palanikumar, M. Chaudhari, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," in *Proc. ACM Sigmetrics*, May 2020, pp. 95–96. [Online]. Available: https://arxiv.org/abs/1804.10331

[22] S. Kadhe, E. Soljanin, and A. Sprintson, "Analyzing the download time of availability codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 1467–1471.

[23] S. Kadhe, E. Soljanin, and A. Sprintson, "When do the availability codes make the stored data more available?" in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 956–963.

[24] M. F. Aktas, E. Najm, and E. Soljanin, "Simplex queues for hot-data download," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 1, pp. 35–36, Sep. 2017.

[25] M. F. Aktas, S. Kadhe, E. Soljanin, and A. Sprintson, "Download time analysis for distributed storage codes with locality and availability," *IEEE Trans. Commun.*, vol. 69, no. 6, pp. 3898–3910, Jun. 2021.

[26] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proc. Thirty-Sixth Annu. ACM Symp. Theory Comput. (STOC)*, 2004, pp. 262–271.

[27] N. Silberstein and A. Gál, "Optimal combinatorial batch codes based on block designs," *Des. Codes Cryptogr.*, vol. 78, pp. 409–424, Feb. 2014.

[28] A. S. Rawat, Z. Song, A. G. Dimakis, and A. Gál, "Batch codes through dense graphs without short cycles," *IEEE Trans. Inf. Theory*, vol. 62, no. 4, pp. 1592–1604, Apr. 2016.

[29] M. Noori, E. Soljanin, and M. Ardakani, "On storage allocation for maximum service rate in distributed storage systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2016, pp. 240–244.

[30] P. Peng and E. Soljanin, "On distributed storage allocations of large files for maximum service rate," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 784–791.

[31] P. Peng, M. Noori, and E. Soljanin, "Distributed storage allocations for optimal service rates," *IEEE Trans. Commun.*, early access, Jul. 9, 2021, doi: 10.1109/TCOMM.2021.3095968.

[32] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control, and Stochastic Networks Perspective*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[33] F. Kazemi, S. Kurz, and E. Soljanin, "A geometric view of the service rates of codes problem and its application to the service rate of the first order Reed–Müller codes," in *Proc. IEEE Internat. Symp. Inf. Theory (ISIT)*, Jan. 2020, pp. 66–71. [Online]. Available: arXiv:2001.09121.

[34] R. G. Gallager, *Information Theory and Reliable Communication*. Hoboken, NJ, USA: Wiley, 1968.

[35] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Hoboken, NJ, USA: Wiley, 2006.

[36] M. Sardari, R. Restrepo, F. Fekri, and E. Soljanin, "Memory allocation in distributed storage networks," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2010, pp. 1958–1962.

[37] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocations," *IEEE Trans. Inf. Theory*, vol. 58, no. 7, pp. 4733–4752, Jul. 2012.

[38] N. Alon, P. Frankl, H. Huang, V. Rödl, A. Rucinski, and B. Sudakov, "Large matchings in uniform hypergraphs and the conjectures of Erdős and Samuels," *J. Combinat. Theory A*, vol. 119, no. 6, pp. 1200–1215, Aug. 2012.

[39] Y.-H. Kao, A. G. Dimakis, D. Leong, and T. Ho, "Distributed storage allocations and a hypergraph conjecture of Erdős," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 902–906.

[40] P. Erdős, "A problem on independent R-tuples," in *ARTICLE IN PRESS B. Bollobás et al./ Journal of Combinatorial Theory, Series A*. Princeton, NJ, USA: Citeseer, 1965.

[41] S. E. Anderson, A. Johnston, G. Joshi, G. L. Matthews, C. Mayer, and E. Soljanin, "Service rate region of content access from erasure coded storage," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

[42] F. Kazemi, S. Kurz, E. Soljanin, and A. Sprintson, "Efficient storage schemes for desired service rate regions," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Apr. 2021, pp. 1–5. [Online]. Available: arXiv:2010.12614.

[43] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, 2nd ed. Amsterdam, The Netherlands: North-Holland Publishing Company, 1978.

[44] V. R. Cadambe and A. Mazumdar, "Bounds on the size of locally recoverable codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 11, pp. 5787–5794, Nov. 2015.

[45] S. Kadhe and R. Calderbank, "Rate optimal binary linear locally repairable codes with small availability," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 166–170.

[46] A. Klein, "On codes meeting the Griesmer bound," *Discrete Math.*, vol. 274, nos. 1–3, pp. 289–297, Jan. 2004.

[47] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection," *Trans. I.R.E. Prof. Group Electron. Comput.*, vol. EC-3, no. 3, pp. 6–12, Sep. 1954.

[48] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," Massachusetts Inst. Tech. Lexington Lincoln Lab., Lexington, MA, USA, Tech. Rep. 44, 1953.

[49] E. F. Assmus and J. D. Key, *Designs and Their Codes*, no. 103. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[50] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[51] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, "Optimal linear codes with a local-error-correction property," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 2776–2780.

[52] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Aug. 2012.

[53] N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with locality for two erasures," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2014, pp. 1962–1966.

[54] P. Gopalan, C. Huang, B. Jenkins, and S. Yekhanin, "Explicit maximally recoverable codes with locality," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5245–5256, Sep. 2014.

[55] D. West, "Introduction to graph theory," in *Featured Titles for Graph Theory Series*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[56] F. Kazemi, E. Karimi, E. Soljanin, and A. Sprintson, "A combinatorial view of the service rates of codes problem, its equivalence to fractional matching and its connection with batch codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jan. 2020, pp. 646–651. [Online]. Available: arXiv:2001.09146

[57] V. Voloshin, *Introduction to Graph and Hypergraph Theory*. Hauppauge, NY, USA: Nova Science Publishers, 2009.

[58] Z. Wang, H. M. Kiah, Y. Cassuto, and J. Bruck, "Switch codes: Codes for fully parallel reconstruction," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2061–2075, Apr. 2017.

[59] E. R. Scheinerman and D. H. Ullman, *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. New York, NY, USA: Dover, 2013.

[60] M. B. Paterson, D. R. Stinson, and R. Wei, "Combinatorial batch codes," *Adv. Math. Commun.*, vol. 3, 2009.

[61] V. Skachek, "Batch and PIR codes and their connections to locally repairable codes," in *Network Coding and Subspace Designs* (Signals and Communication Technology). Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-70293-3_16.

[62] S. Yekhanin, "Private information retrieval," *Commun. ACM*, vol. 53, no. 4, pp. 68–73, Apr. 2010.

[63] A. Fazeli, A. Vardy, and E. Yaakobi, "Codes for distributed PIR with low storage overhead," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 2852–2856.

[64] S. Dodunekov and J. Simonis, "Codes and projective multisets," *Electron. J. Combinatorics*, vol. 5, no. 1, p. 37, 1998.

[65] M. A. Tsfasman and S. G. Vladut, "Geometric approach to higher weights," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1564–1588, Nov. 1995.

[66] A. Beutelspacher, B. Albrecht, and U. Rosenbaum, *Projective Geometry: From Foundations to Applications*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[67] G. Joshi, E. Soljanin, and G. Wornell, "Queues with redundancy: Latency-cost analysis," in *Proc. ACM SIGMETRICS Workshop Math. Modeling Anal.*, Jun. 2015, pp. 1–3.
[68] G. Joshi, "Synergy via redundancy: Boosting service capacity with adaptive replication," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, pp. 21–28, Mar. 2018.
[69] M. F. Aktas and E. Soljanin, "Straggler mitigation at scale," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2266–2279, Dec. 2019.
[70] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.
[71] M. F. Aktas and E. Soljanin, "Heuristics for analyzing download time in MDS coded storage systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1929–1933.
[72] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands I," *SIAM J. Appl. Math.*, vol. 44, no. 5, pp. 1041–1053, Oct. 1984.
[73] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Trans. Comput.*, vol. 37, no. 6, pp. 739–743, Jun. 1988.
[74] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," *Unpublished, Available Online*, to be published. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.3062&rep=rep1&type=pdf
[75] S. Chen *et al.*, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 1042–1050.
[76] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2443–2457, Aug. 2016.
[77] M. F. Aktas, A. Behrouzi-Far, E. Soljanin, and P. Whiting, "Evaluating load balancing performance in distributed storage with redundancy," 2019, *arXiv:1910.05791*. [Online]. Available: http://arxiv.org/abs/1910.05791
[78] Y. Kim, R. Righter, and R. Wolff, "Job replication on multiserver systems," *Adv. Appl. Probab.*, vol. 41, no. 2, pp. 546–575, Jun. 2009.
[79] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf, "A better model for job redundancy: Decoupling server slowdown and job size," in *Proc. IEEE 24th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2016, pp. 1–10.
[80] K. Gardner, E. Hyytiä, and R. Righter, "A little redundancy goes a long way: Convexity in redundancy systems," *Perform. Eval.*, vol. 131, pp. 22–42, Jun. 2019.
[81] Y. Raaijmakers, S. C. Borst, and O. J. Boxma, "Delta probing policies for redundancy," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, pp. 72–73, Jan. 2019.
[82] Y. Raaijmakers, S. Borst, and O. Boxma, "Redundancy scheduling with scaled Bernoulli service requirements," *Queueing Syst.*, vol. 93, nos. 1–2, pp. 67–82, Oct. 2019, doi: 10.1007/s11134-019-09621-2.
[83] G. Joshi and D. Kaushal, "Synergy via redundancy: Adaptive replication strategies and fundamental limits," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 737–749, Apr. 2021.
[84] E. Anton, U. Ayesta, M. Jonckheere, and I. M. Verloop, "On the stability of redundancy models," *Oper. Res.*, vol. 69, no. 5, pp. 1540–1565, 2021, doi: 10.1287/opre.2020.2030.
[85] E. Anton, U. Ayesta, M. Jonckheere, and I. M. Verloop, "Improving the performance of heterogeneous data centers through redundancy," in *Proc. Abstract Proc. ACM SIGMETRICS/Int. Conf. Meas. Modeling Comput. Syst.*, May 2021, pp. 55–56.
[86] D. Wang, G. Joshi, and G. W. Wornell, "Efficient straggler replication in large-scale parallel computing," *ACM Trans. Modeling Perform. Eval. Comput. Syst.*, vol. 4, no. 2, pp. 1–23, Jun. 2019.
[87] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," in *Proc. ACM SIGMETRICS Distrib. Cloud Comput. Workshop*, Jun. 2015, pp. 1–4.
[88] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proc. ACM Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, 2014, pp. 599–600.
[89] T. Baumbaugh, Y. Diaz, S. Friesenhahn, F. Manganiello, and A. Vetter, "Batch codes from Hamming and Reed–Müller codes," *J. Algebra Combinatorics Discrete Struct. Appl.*, vol. 5, no. 3, pp. 153–165, Oct. 2018.
[90] A.-E. Riet, V. Skachek, and E. K. Thomas, "Asynchronous batch and PIR codes from hypergraphs," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

Mehmet Aktaş received the B.S. degree in electrical and electronics engineering from Bilkent University, Turkey, and the M.S. and Ph.D. degrees in computer engineering from Rutgers University. He is currently an Assistant Professor at Bilkent University. Previously, he was a Senior Software Engineer at MathWorks, MA, USA. His research interest is broadly to make distributed computer systems faster and more robust. Along this line, he has been doing research in both systems development and theoretical analysis. While solving problems, he mostly relies on probabilistic modeling and tools from applied probability, such as queueing theory, order statistics, and reinforcement learning.

Gauri Joshi (Member, IEEE) received the B.Tech. and M.Tech. degrees in electrical engineering from the Indian Institute of Technology (IIT) Bombay in 2010 and the Ph.D. degree from MIT EECS in June 2016, advised by Prof. Gregory Wornell. She has been an Assistant Professor with the ECE Department, Carnegie Mellon University, since September 2017. Her research interests include distributed machine learning, coding theory, and algorithms for parallel computing. Previously, she worked as a Research Staff Member at IBM T. J. Watson Research Center. Her awards and honors include NSF CAREER Award (2021), the ACM Sigmetrics Best Paper Award (2020), NSF CRII Award (2018), the IBM Faculty Research Award (2017), the Best Thesis Prize in Computer Science at MIT (2012), and the Institute Gold Medal of IIT Bombay (2010).

Swanand Kadhe (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Texas A&M University in 2017. He has been a Visiting Researcher at Nokia Bell Labs, Duke University, and The Chinese University of Hong Kong. From 2009 to 2012, he was a Research and Development Engineer at TCS Innovation Labs, Bengaluru. He is currently a Post-Doctoral Researcher with the EECS Department, University of California at Berkeley. His research interests lie broadly in federated and distributed machine learning, coding and information theory, privacy and security, and blockchains. He was a recipient of the 2016 Graduate Teaching Fellowship from the College of Engineering, Texas A&M University.

Fatemeh Kazemi (Student Member, IEEE) received the M.S. degree in electrical engineering from the University of Tehran in 2016. She is currently pursuing the Ph.D. degree with the ECE Department, Texas A&M University. Her research interests lie in the areas of distributed storage and computing systems, with a focus on privacy, efficiency, and reliability of these systems. In particular, she has been working on private information retrieval, service rates of distributed systems, coded caching, group testing, and machine learning. She was a recipient of the 2021 Graduate Teaching Fellowship awarded by the College of Engineering, Texas A&M University.

Emina Soljanin (Fellow, IEEE) is currently a Professor at Rutgers University. Before moving to Rutgers in 2016, she was a Distinguished Member of Technical Staff for 21 years with the Mathematical Sciences Research of Bell Labs. Her interests and expertise are broad. Over the past quarter of the century, she has participated in numerous research and business projects, as diverse as power system optimization, magnetic recording, color space quantization, hybrid ARQ, network coding, data and network security, distributed systems performance analysis, and quantum information theory. She is an Outstanding Alumnus of the Texas A&M School of Engineering, the 2011 Padovani Lecturer, the 2016/2017 Distinguished Lecturer, and the 2019 President of the IEEE Information Theory Society. She served as an Associate Editor of Coding Techniques for IEEE TRANSACTIONS ON INFORMATION THEORY, the Information Theory Society Board of Governors, and various roles on other journal editorial boards and conference program committees (see https://www.ece.rutgers.edu/emina-soljanin).