

# Evaluating Load Balancing Performance in Distributed Storage With Redundancy

Mehmet Fatih Aktaş<sup>✉</sup>, Amir Behrouzi-Far, Emina Soljanin<sup>✉</sup>, *Fellow, IEEE*, and Philip Whiting<sup>✉</sup>

**Abstract**—To facilitate load balancing, distributed systems store data redundantly. We evaluate the load balancing performance of storage schemes in which each object is stored at  $d$  different nodes, and each node stores the same number of objects. In our model, the load offered for the objects is sampled uniformly at random from all the load vectors with a fixed cumulative value. We find that the load balance in a system of  $n$  nodes improves multiplicatively with  $d$  as long as  $d = o(\log(n))$ , and improves exponentially once  $d = \Theta(\log(n))$ . We show that the load balance improves in the same way with  $d$  when the service choices are created with XOR’s of  $r$  objects rather than object replicas. In such redundancy schemes, storage overhead is reduced multiplicatively by  $r$ . However, recovery of an object requires downloading content from  $r$  nodes. At the same time, the load balance increases additively by  $r$ . We express the system’s load balance in terms of the maximal spacing or maximum of  $d$  consecutive spacings between the ordered statistics of uniform random variables. Using this connection and the limit results on the maximal  $d$ -spacings, we derive our main results.

**Index Terms**—Load balancing, distributed storage, redundant storage, distributed systems.

## I. INTRODUCTION

**D**ISTRIBUTED computing systems are built on a storage layer that provides data write/read service for executing workloads. Thus, the overall performance of a computing system depends on the data access (I/O) performance implemented by the underlying storage system. In production systems, data access times are the main bottleneck to performance [2]. Indeed, access times in modern large-scale systems (e.g., Cloud systems) greatly suffer from storage nodes that exhibit poor or variable performance [3]. Poor performance is caused by many factors, but primarily it comes from multiple-workload resource sharing and the resulting

Manuscript received December 25, 2019; revised November 3, 2020; accepted January 13, 2021. Date of publication January 25, 2021; date of current version May 20, 2021. This work was supported by the National Science Foundation under Grant CIF-1717314. This article was presented in part at the 2019 XVI International Symposium “Problems of Redundancy in Information and Control Systems.” (Corresponding author: Mehmet Fatih Aktaş.)

Mehmet Fatih Aktaş, Amir Behrouzi-Far, and Emina Soljanin are with the Department of Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ 08901 USA (e-mail: mehmet.aktas@rutgers.edu; amir.behrouzifar@rutgers.edu; emina.soljanin@rutgers.edu).

Philip Whiting is with the Department of Engineering, Macquarie University, Sydney, NSW 2109, Australia (e-mail: philip.whiting@mq.edu.au).

Communicated by A. Jiang, Associate Editor for Coding Theory.

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TIT.2021.3054385>.

Digital Object Identifier 10.1109/TIT.2021.3054385

contention at the system resources [4]. Poor or variable performance is possible at any level of load but it is certainly aggravated at overloaded storage nodes [5]. It is, therefore, paramount for distributed systems to be able to balance the offered data access load across the storage nodes.

It follows that to achieve good data access performance, we must balance the offered load across the storage nodes as evenly as possible. In modern storage systems (e.g., HDFS [6], Cassandra [7], Redis [8]), data objects are replicated and made available across multiple nodes so that the offered load for each object, which we refer to as the *demand* for the object, can be split across *multiple nodes (service choices)*. The best support for load balancing is achieved when each object is stored at each node, but that is feasible only in exceptional cases at large scale. If the demand for each object is known and fixed, each object could be stored with an adequate level of redundancy. However, in practice, object popularities, and in turn their demands, are not only unknown but also fluctuate over time. Thus, load balancing should be robust against skews and changes in object popularities [2], [9].

Load balancing has been considered in two important settings. In the first, we call the *dynamic setting*, load balancing is addressed from the point of view of scheduling tasks for processing. Here the nodes correspond to single queues which are processed independently and in parallel. Load balancing amounts to interrogating some subset of all the queues and offering new tasks to those nodes which are least loaded. The simplest of these models is the one in which each task is sent to the node with the least number of tasks. Clearly this achieves the ideal load balance. This scheme is only practical for a relatively small number of nodes but becomes unworkable for large-scale systems with tens of thousands of nodes or more. For this reason, a great deal of attention has been placed on developing schemes which offer tasks to a restricted number of nodes. These schemes include those based on the well-known *power of  $d$  choices* paradigm. A range of asymptotic results have been obtained following this direction, often using analysis based on balls into bins models [10]–[12].

All of the above literature addresses the load balancing question from the point of view of spreading tasks evenly across the nodes. Its main weakness however is that the well-understood power of  $d$  choices is only applicable for systems where the arrivals can be placed at any one of the nodes. For instance, for scheduling compute tasks across nodes within the same data center. However, the flexibility of querying any  $d$  bins at random does not exist in storage

systems. This is because each object is typically stored only at a limited number of nodes and an arriving request can only be served at one of the nodes that host the requested object.

A more appropriate model for storage is to suppose that each request is offered to one of a subset of nodes, each of which hosts the requested object. Kenthapadi and Panigrahy propose a model in [13] along these lines where subsets of nodes are represented as edges in a graph. They studied this restricted model for the power of two choices with  $n$  balls and  $n$  bins. Edges are selected according to incoming object requests and then the arrival is assigned to the least loaded node among the vertices of the edge. Godfrey then extended this in [14] to general power of  $d$  choices. Applying this model to a system with  $n$  storage nodes with each object being replicated  $d = \Omega(\log(n))$  times, Godfrey's results lead to the conclusion that effective load balancing can be achieved in the sense of power of  $d$  choices. Godfrey then goes on to show that if  $d$  grows more slowly, then the above conclusion for the power of  $d$  schemes is no longer valid. Storage schemes we consider in this paper are a natural special case of the balanced allocations on hyper graphs that was considered by Godfrey. Beyond the above conclusions, Godfrey's results provide little insight into practical storage schemes. For example, how to distribute different objects across various storage nodes or what gain can be made from using coded schemes based on object XOR's. Finally Godfrey's results are shown only for the lightly loaded case, which is when the cumulative load offered on the system scales as the order of the number of nodes. This paper extends Godfrey's results for the case with concrete storage schemes and without restricting ourselves to the lightly loaded case. Furthermore we examine i) the number of different objects stored per node, ii) object overlaps between the storage nodes, iii) using coded objects rather than plain replicas, and address their impact on storage efficiency and load balancing performance.

Under the dynamic setting, load offered for the objects is not known a priori. Requests arrive sequentially and each is assigned to a node based on the current load at each node. We now turn to the second setting, namely the *static setting*. In this setting, a different question is asked: is it feasible to carry the load, if the load offered for each object is known from the start. Any assignment strategy realized under the dynamic setting is also achievable under the static setting. This is because knowing the offered loads for the stored objects in advance makes it only easier to balance the load. Load balancing performance in the static setting therefore represents the best-case performance of the system.

The question asked in the static setting gives rise to two distinct approaches. The first approach leads to the design of redundancy schemes, namely *batch codes*. They balance the load as long as any  $m$  objects are chosen with replacement out of all objects and then requested simultaneously [15]. The storage schemes we consider fall into the class of combinatorial batch codes [16]. We should note that the static model adopted for batch codes has been extended to a more dynamic setting, which led to the design of *asynchronous* batch codes [17]. Batch codes were originally designed to balance only a single batch of requests. An asynchronous

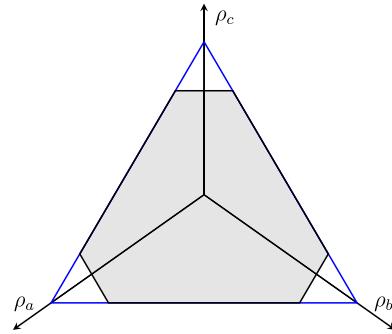


Fig. 1. An illustration of the demand offered for three objects  $a, b, c$  and the fraction of those that are supported by the system. The simplex region corresponds to all demand vectors that can be offered on the system. It is given by  $\{(x, y, z) \mid x + y + z = \Sigma\}$  for cumulative load  $\Sigma$ . The shaded region shows the demand vectors that are supported by the system.

variant is designed to balance the present batch together with the upcoming batch or batches. This approach asks the question the other way around and seeks to find the set of all object demand vectors that can be supported by a system with a given storage scheme, namely the system's *service capacity region* [18], [19]. Our treatment of load balancing falls into this second approach. References [18] and [19] only address the case where each node stores a single object, their approach being to find the system's complete service capacity region. However determining this region with multiple objects at each node appears to be a largely intractable problem. In our approach, we rely on a new stochastic formulation which allows us to analyze the load balancing in this scenario and to draw a range of conclusions on the design and structure of storage schemes. Furthermore, the primary goal in the service rate approach is to evaluate system stability under different object popularities. In this paper however we address the related problem of *feasibility* of load balancing, i.e., the degree to which storage resources can be adapted according to the changes of the object popularities.

It is helpful at this point to add a few words by way of explaining the model setup that we use in this paper. First, we consider only *regular balanced* storage schemes in which each object is replicated  $d$  times (hence regular) and each node stores the same number of objects (hence balanced). Storage schemes specify where each object copy is stored, and therefore determine the set of all possible ways that one can split and assign individual object demands across the nodes. Second, as far as object demands are concerned, we suppose that the cumulative load is fixed and that all object popularities are equally likely. This is motivated by the fact that the cumulative demand for all the objects stored in the system is known to vary slowly over time and therefore is easy to estimate (see, e.g., Fig. 7 in [20]). Individual demands for objects fluctuate much more rapidly. Additionally, our assumption of fixed cumulative load on the system is the continuous generalization of the offered load model used in the batch code problem.

We now turn to the metrics which we will be using to analyze load balancing performance. These metrics can be understood by considering Fig. 1. Fig. 1 shows a simplex

region that corresponds to the set of all possible demand vectors for three objects. If load balancing is ideal, then for all these vectors stability can be achieved. However, this is too onerous in practice as it would require an unacceptably large storage overhead. A compromise therefore is to minimize the fraction of demand vectors which cannot be supported. Under our formulation, this corresponds to our assumption that the object demands are uniformly distributed on the simplex region. This is indeed the standard model used in the study of load balancing in the dynamic setting. Overall, we measure the *robustness* of load balancing as the probability  $P_\Sigma$  that the system will be stable when the demand vector is sampled uniformly at random from the simplex region defined by cumulative load  $\Sigma$ . We also use another metric that is closely connected to  $P_\Sigma$  to measure the load imbalance. Precisely for a system of  $n$  nodes under a cumulative load of  $\Sigma$ , the load imbalance,  $\mathcal{I}$ , is given by minimizing the maximum load and dividing it by its minimum possible value  $\Sigma/n$ .

With the metrics now established, we can state the main contributions of this paper. These can be understood by considering the following questions with respect to load balancing and storage:

- Q1** How does  $\mathcal{I}$  scale with the number of objects and nodes in the system where there is no storage redundancy?
- Q2** Does the degree of overlap between the service choices of different objects play a critical role in terms of achieving better load balance? How does  $\mathcal{I}$  depend on the number of service choices  $d$  provided for each object?
- Q3** XOR’ing reduces storage requirements, however can effective load balancing still be achieved using XOR’s, rather than object replicas?

We address these questions for common storage schemes with regular balanced redundancy. Optimizing storage schemes for various purposes are studied elsewhere (e.g., for improving data access in [21]).

**Our contribution:** From the results of the paper, we are able to conclude the following answers **A1**, **A2** and **A3** for the questions **Q1**, **Q2** and **Q3** above.

**A1:** For the storage schemes with no redundancy, we find  $\mathcal{I} = \Theta(\log(n)/m)$  where  $m$  is the number of different objects stored on each node. This implies that in the limit as  $n \rightarrow \infty$ , load imbalance grows as  $\log(n)$ . This implies that if we want to maintain a load of  $\Sigma/n$  in the maximally loaded node, we need  $n \log(n)$  nodes in the system.

**A2:** To answer **Q2**, we consider  $d$ -replication storage schemes. Different storage schemes under a regular balanced requirement lead to differences in the way objects overlap at the nodes. By examining three different storage designs, we find that the scheme with consistently small overlaps outperforms schemes with fewer overlaps which are necessarily larger in size. For one class of schemes with limited overlap, which we call the  $r$ -gap schemes, we have obtained the following asymptotic results  $\mathcal{I} = \Theta(\log(n)/d)$  when  $d = o(\log(n))$  and  $\mathcal{I} = \Theta(\log \log(n)/\log(n))$  when  $d = \Theta(\log(n))$ . These results imply that i) creating  $d$  service choices for each object initially reduces the load imbalance in the system multiplicatively by  $d$ , ii) there is an exponential reduction in

load imbalance as soon as  $d$  reaches of order  $\log(n)$ . This quantifies the tradeoff between storage and service capacity for  $r$ -gap schemes.

**A3:** Using XOR’s reduces the amount of storage needed but at the same time increases the capacity needed to access various objects. This is because to obtain a single object, we have to access several object codes. An  $r$ -XOR object code is one which is constructed from  $r$  objects. Our asymptotic results show that storage with  $d$ -fold redundancy implemented with  $r$ -XOR’s have the advantage that it has the same scaling of  $\mathcal{I}$  in  $d$  as if the service choices were created with replicas. Thus in large-scale systems, there is no loss of significant benefit over replication. XOR’ing can be used to trade off between storage and the access capacity.

The paper is organized as follows: Sec. I gives an overview of the literature on load balancing in storage context. We also discuss the connections between our approach and the prior work. Sec. II presents our storage and offered load model and its connection with uniform spacings. We also precisely define the metrics that we use to evaluate load balancing performance. In Sec. III we consider storage schemes with no redundancy and answer **Q1**. In Sec. IV we consider storage schemes with object replication and answer **Q2**. In Sec. VI we consider creating storage redundancy with XOR’s rather than object replicas and answer **Q3**.

## II. SYSTEM MODEL AND PERFORMANCE METRICS

In this section, we introduce our system model and define the metrics we use to evaluate load balancing performance. We study load balancing in the static setting with a continuous service and offered load model. Our model reveals an interesting connection of the load balancing problem to convex polytopes and the spacings between ordered random uniform samples, the so-called *uniform spacings* [22]. We elaborate on this in Sec. II-B. The latter connection enabled us to apply prior results on uniform spacings in answering the questions posed in Sec. I.

### A. Storage and Access Model

We consider a system of  $n$  storage nodes  $s_1, \dots, s_n$  hosting  $k$  data objects  $o_1, \dots, o_k$ , possibly with redundancy. Each node provides the same *capacity* for content access, which is defined as the maximum number of bytes that can be streamed from a node per unit time. An *object* denotes the smallest unit of content, and mathematically, it is a fixed-length string of bits. XOR’ing multiple objects is carried out bitwise.

We refer to the *offered load* for object  $o_i$  as its *demand*  $\rho_i$ . Demand for an object represents the average number of bytes streamed from the system per unit time to access the object, divided by a single node’s content access capacity. We refer to a node that hosts an object as a *service choice* for the object. Multiple service choices for an object can be created by replicating it over several nodes. We consider  $d$ -choice storage schemes with replicas in Sec. IV. Alternatively, XOR’ed object copies can be used to create multiple service choices. We consider  $d$ -choice storage schemes with XOR’s in Sec. VI. When XOR’ing is used, a service choice for an object refers

to a *recovery set*, that is, a set of nodes that can jointly recover the object. Accessing an object through one choice should not interfere with accessing the same object through another choice. Different service choices for the same object are therefore disjoint.

Demand for an object can be arbitrarily split across its service choices. When a load of  $\rho$  is exerted by an object on a recovery set, each node within the set will be offered a load of  $\rho$ . The *load on a node* is given by the sum of the offered load portions exerted on it by the objects for which the node can serve as a choice. A node is said to be *stable* if the load on it is less than 1. A system is said to be stable if every node within the system is stable. We assume that each of the object demands  $\rho_i$  is split across its service choices so that the load on the maximally loaded node is minimized. As we describe further in Sec. II-C, this can be obtained by solving a norm minimization problem given the storage scheme and the value of  $\rho_i$ .

A *storage allocation* defines how each object is assigned, possibly with redundancy, to storage nodes. This paper focuses on *regular balanced d-choice* storage allocations.

*Definition 1:* A regular balanced  $d$ -choice allocation stores each object with  $d$  service choices and distributes object copies across the nodes so that each node stores the same number of different objects (either as an exact or XOR'ed copy).

There are many ways to design a  $d$ -choice allocation. We detail some of them in Sec. IV and VI. In the rest of the paper, unless otherwise noted, the allocation itself will refer to a regular balanced allocation.

**Connection with batch codes:** A  $(k, N, m, n, t)$  *batch code* encodes  $k$  objects into  $N$  copies with redundancy and distributes them across the  $n$  nodes in such a way that any  $m$  of these objects can be accessed by reading at most  $t$  objects from any node [15]. The goal while designing batch codes is to minimize the total storage requirement. Redundancy can be either in the form of replicating individual objects or encoding (e.g., XOR'ing) multiple objects together. Multiset batch codes are concerned with a more general case in which the selection of  $m$  objects for access is done with replacement. It should be noted that each of the objects needs to be accessed separately, that is, the content that is read for accessing an object cannot be used to access another object. We should note that the demand model assumed for batch codes have been extended to cases with additional constraints, such as balancing the access frequencies over the nodes [23]. This area of research has investigated the use of certain class of redundancy schemes to balance access when the popularity ranks of the objects are known in advance.

In Sec. IV we will consider storage allocations that are constructed by object replication. Such allocations implement batch codes as follows.

*Lemma 1:* Any  $d$ -choice regular balance storage allocation with object replication represents a  $(k, kd, n, n, 1)$  batch code and a  $(k, kd, d, n, 1)$  multiset batch code.

*Proof:* See Appendix A.  $\square$

Batch codes with replication are known as *combinatorial* batch codes and their construction has been well

studied [16], [24]. In particular, a combinatorial batch code is named as *d-uniform* if it stores each object in exactly  $d$  nodes, which is exactly the  $d$ -choice requirement we consider here. An approach that is based on block design has been given in [25] to construct optimal  $d$ -uniform batch codes.

### B. Offered Load and Uniform Spacing Model

We suppose that the system can be offered any object demand (offered load) vector  $(\rho_1, \dots, \rho_k)$  in the set

$$\mathcal{S}_\Sigma = \left\{ (\rho_1, \dots, \rho_k) \mid \sum_{i=1}^k \rho_i = \Sigma, \rho_i \geq 0 \right\}. \quad (1)$$

That is, the cumulative offered load remains constant but the object popularities can change arbitrarily. The cumulative condition we impose on the offered load is the continuous generalization of the load model assumed in the multiset batch code problem. Recall that a multiset batch code is designed to support a user who can simultaneously access  $m$  objects that are selected with replacement out of all objects stored in the system.

We further assume that the demand vector  $(\rho_1, \dots, \rho_k)$  is sampled uniformly at random from  $\mathcal{S}_\Sigma$ . Uniform distribution across all possible demand vectors models the case where no a priori knowledge is given on the object popularities. In other words, it represents the case with maximum uncertainty about the object popularities. This assumption is the continuous generalization of what has been used in balls-into-bins models. There each ball arrives for one of the stored objects chosen uniformly at random from all stored in the system. In addition as we discuss shortly, sampling demand vectors uniformly at random is able to model the skewed nature of object popularities in real systems [9]. However it should be noted that modeling with a more general distribution would yield additional insight on load balancing under more specific and possibly more realistic offered load models. An example of such a model would be one that puts larger probability mass on the demand vectors representing skewed object popularities.

In what follows, we define uniform spacings. They are mathematical objects connected with the uniform sampling of points from a simplex. Let  $U_{(1)}, \dots, U_{(k-1)}$  be  $k-1$  i.i.d. uniform samples in  $[0, 1]$ , given in non-decreasing order. Then  $S_i = U_{(i)} - U_{(i-1)}$  for  $i = 1, \dots, k$ , where  $U_{(0)} = 0$  and  $U_{(k)} = 1$ , are known as  $k$  uniform spacings on the unit line.

*Lemma 2 (See e.g. [22]):* Uniform spacings  $(S_1, \dots, S_k)$  are uniformly distributed over the simplex

$$\left\{ (s_1, \dots, s_k) \mid \sum_{i=1}^k s_i = 1, s_i \geq 0 \text{ for } i = 1, \dots, k \right\}.$$

Lemma 2 implies that object demands  $\rho_i$  in our model under a cumulative load of  $\Sigma$  can be seen as  $k$  uniform spacings in  $[0, \Sigma]$ . This connection allows us to use the results on uniform spacings to evaluate load balancing performance in systems with  $d$ -choice storage allocation. We do the evaluation in terms of the performance metrics defined in the following subsection.

We next examine the popularity skew characteristics captured by our uniform offered load model (as promised above). Without loss of generality, let us assume that the cumulative demand  $\Sigma$  offered on the system is 1. Let  $N(\alpha, \beta)$  denote the number of objects with a demand of  $\geq \alpha$  and  $\leq \beta$ . Then  $N(\alpha, \beta)$  is given by the number of uniform spacings that are within  $[\alpha, \beta]$ . An asymptotic characterization of  $N(\alpha, \beta)$  has been given in [26] as follows.

*Theorem 1* ([26, Theorem 8.1-2-3]):

*R1* .  $N(\alpha/k, \beta/k)$  is asymptotically normally distributed as  $k \rightarrow \infty$  with an asymptotic mean and variance

$$\begin{aligned}\mu_k &\sim k(e^{-\alpha} - e^{-\beta}), \\ \sigma_k^2 &\sim k(e^{-\alpha} - e^{-\beta} - (\alpha e^{-\alpha} - \beta e^{-\beta})^2).\end{aligned}$$

*R2* .  $N(\alpha/k^2, \beta/k^2)$  has an asymptotic Poisson distribution with parameter  $\beta - \alpha$ .

*R3* .  $N((\log(k) + \alpha)/k, (\log(k) + \beta)/k)$  has an asymptotic Poisson distribution with parameter  $e^{-\alpha} - e^{-\beta}$ .

Results in Theorem 1 tell us a great deal about the object popularities implemented by our demand model. With high probability, only a few of the objects will be highly popular ( $\rho \sim \log(k)/k$ ), only a few will have very low popularity ( $\rho \sim 1/k^2$ ), while most objects will have around-average popularity ( $\rho \sim 1/k$ ). This reflects the skewed object popularities observed in real storage systems (see e.g. Fig. 3 in [27]).

### C. Storage Service Capacity

We now obtain mathematical expressions determining the *service capacity region* for a storage system. In particular, we will express the set of all object demand vectors under which the system with a given storage allocation can operate under stability. Service capacity for systems that store content with erasure coding was first studied in [18] and further studied in [19]. We adopt a formulation similar to the one introduced in [18]. The formulation we present in this section provides a geometric interpretation of the performance metrics  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  introduced in Sec. II-D.

*Definition 2:* Service capacity region for a system with a given storage allocation is the set of all object demand vectors  $\rho = (\rho_1, \dots, \rho_k)$  under which the system can operate under stability.

In what follows, we explain how to express the service capacity region as a solution for a system of linear inequalities. Let us consider a system in which object  $o_i$  is stored on  $d_i$  nodes for  $i = 1, \dots, k$ . Then its demand  $\rho_i$  can be distributed across its  $d_i$  service choices, each handling a fraction of  $\rho_i$ . Let us denote the portion of  $\rho_i$  that is assigned to the  $j$ th choice of  $o_i$  with  $\rho_i^{(j)}$ . Then we have  $\rho_i = \rho_i^{(1)} + \dots + \rho_i^{(d_i)}$ . We represent the stacked collection of all these per-node demand portions with the following vector of length  $d_1 + \dots + d_k$ :

$$\mathbf{x}^T = \left( \rho_1^{(1)}, \dots, \rho_1^{(d_1)}, \dots, \rho_k^{(1)}, \dots, \rho_k^{(d_k)} \right).$$

Converting back to  $\rho$  from  $\mathbf{x}$  is a matter of matrix-vector multiplication as  $\rho = \mathbf{T} \cdot \mathbf{x}$ , where  $\mathbf{T}$  is a binary matrix of size  $k \times (d_1 + \dots + d_k)$ . System stability is ensured if and only if the total demand flowing into each node is less than

its capacity 1. This can be expressed as a linear inequality for each node and a matrix inequality for the whole system of  $n$  nodes as

$$\mathbf{M} \cdot \mathbf{x} \prec \mathbf{1}, \quad \mathbf{x} \succeq \mathbf{0}, \quad (2)$$

where  $\prec$  and  $\succeq$  denote the standard partial orderings in  $R^n$ , and  $\mathbf{0}$  and  $\mathbf{1}$  denote the all-zeros and ones vectors of length  $n$ , respectively. The overall service capacity region of the system is given by

$$\mathcal{C} = \{ \rho \mid \exists \mathbf{x}; \mathbf{M} \cdot \mathbf{x} \prec \mathbf{1}, \mathbf{T} \cdot \mathbf{x} = \rho, \mathbf{x} \succeq \mathbf{0} \}. \quad (3)$$

$\mathbf{M}$  expresses the storage allocation and it is a binary matrix of size  $n \times (d_1 + \dots + d_k)$ . It is constructed by setting  $\mathbf{M}[i, j]$  to 1 if the demand portion  $\mathbf{x}[j]$  flows into node- $i$ , and to 0 otherwise. When storage redundancy is created with only object replicas, each column of  $\mathbf{M}$  becomes a binary representation of a node that stores the corresponding object copy. Precisely, each column of  $\mathbf{M}$  would consist of a single 1, and the position of this 1 within the column is equal to the position of the represented node within the sequence of all nodes. For instance for the system that stores  $a, b, c$  across three nodes by allocating two service choices for each as  $\{(a, c), (b, a), (c, b)\}$ , we have

$$\begin{aligned}\mathbf{x}^T &= \left( \rho_a^{(1)}, \rho_a^{(2)}, \rho_b^{(1)}, \rho_b^{(2)}, \rho_c^{(1)}, \rho_c^{(2)} \right), \\ \mathbf{M} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.\end{aligned}$$

When storage redundancy consists of coded objects, some of the demand portions  $\rho_i^{(j)}$  might be assigned to recovery sets. A recovery set for an object is a set of nodes from which the object can be recovered. When a demand portion of  $\rho$  is assigned to a recovery set, then a fraction of  $\rho$  capacity will be used up at each node within the recovery set. Then the columns of  $\mathbf{M}$  that consist of multiple ones represent recovery sets for the corresponding objects. For instance, for the storage allocation  $\{(a, b + c), (b, a + c), (c, a + b)\}$ , we have

$$\begin{aligned}\mathbf{x}^T &= \left( \rho_a^{(1)}, \rho_a^{(2)}, \rho_b^{(1)}, \rho_b^{(2)}, \rho_c^{(1)}, \rho_c^{(2)} \right), \\ \mathbf{M} &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}.\end{aligned}$$

*Lemma 3:* The service capacity region for any storage system is a convex polytope.

*Proof:* The convex polytope expressed by (2) in  $R_+^{d_1+\dots+d_k}$  consists of all demand portion vectors  $\mathbf{x}$  under which the system is stable. Capacity region  $\mathcal{C}$  is the linear transformation of this polytope by  $\mathbf{T}$ . Hence  $\mathcal{C}$  is another convex polytope in  $R_+^k$ .  $\square$

As noted in Sec. II-A, we consider the case where object demands  $\rho_i$  are split across their choices such that the load on the maximally loaded storage node is minimized. This means that for a given object demand vector  $\rho$ , out of all demand portion vectors  $\mathbf{x}$  that satisfy (2), the system will split the demands across the nodes according to  $\mathbf{x}^*$ . This achieves the

best possible load balance. Thus  $\mathbf{x}^*$  is the optimal solution for the following convex optimization problem:

$$\min_{\mathbf{x}} \|\mathbf{M} \cdot \mathbf{x}\|_{\infty}; \quad \mathbf{T} \cdot \mathbf{x} = \boldsymbol{\rho}, \quad \mathbf{x} \succeq \mathbf{0}, \quad (4)$$

where  $\|\cdot\|_{\infty}$  denotes the infinity norm.

Copying an object to a node that did not previously host it, increments the number of service choices for the object. We next state a simple but useful fact as the first step to understanding the gains of increasing the number of service choices for the objects.

**Lemma 4:** Let the system capacity region be  $\mathcal{C}$  for a given storage allocation. Keeping the number of nodes fixed, let us store an object replica (or a coded copy) on a node that did not previously host the object (or any object present in the coded copy). Let  $\mathcal{C}'$  be the system capacity region for this modified allocation. Then  $\mathcal{C} \subset \mathcal{C}'$ .

*Proof:* See Appendix C.  $\square$

#### D. Performance Metrics

We now give precise definitions for the two metrics that we use to quantify load balancing performance in distributed storage. The first metric measures the system's *robustness* against the presence of skews and changes in object popularities. We quantify robustness as the fraction of demand vectors that are supported by the system in the simplex that consists of all vectors that sum up to  $\Sigma$ .

**Definition 3 (Measure of Robustness):** For a system with a given storage allocation, let the capacity region be the polytope  $\mathcal{C}$  and let  $\mathcal{S}_{\Sigma}$  be defined for a given cumulative load  $\Sigma$  as in (1).  $\mathcal{P}_{\Sigma}$  for the system is given by

$$\mathcal{P}_{\Sigma} = \frac{\text{Volume}(\mathcal{C} \cap \mathcal{S}_{\Sigma})}{\text{Volume}(\mathcal{S}_{\Sigma})}. \quad (5)$$

$\mathcal{P}_{\Sigma}$  is obviously 0 when  $\Sigma > n$ , hence we assume  $\Sigma \leq n$  implicitly throughout. The shaded region in Fig. 1 illustrates the intersection of the simplex  $\mathcal{S}_{\Sigma}$  and the system capacity region. Recall that the demand vector  $(\rho_1, \dots, \rho_k)$  offered on the system is sampled uniformly at random from  $\mathcal{S}_{\Sigma}$ . Therefore another way to define  $\mathcal{P}_{\Sigma}$  is that it is the probability that the system defined by  $\mathcal{S}_{\Sigma}$  will be stable. In other words,  $\mathcal{P}_{\Sigma}$  is the *probability of robustness* for a system that operates under a cumulative demand of  $\Sigma$ .

The expression given for  $\mathcal{P}_{\Sigma}$  in (5) is a useful geometric interpretation. It implies that once the capacity region of a system is determined, evaluating  $\mathcal{P}_{\Sigma}$  for it becomes a computational geometry problem. Finding volumes or pairwise intersections of convex polytopes are well studied problems, and numerous efficient algorithms are available to compute both in the literature, e.g., see [28]. Eq.(5) essentially gives a recipe to exactly compute  $\mathcal{P}_{\Sigma}$  for a system with any given storage allocation. This, together with the fact that service capacity region is a convex polytope (Lemma 3), implies  $\mathcal{P}_{\Sigma}$  is non-increasing in  $\Sigma$ .

**Corollary 1:** For any system, if  $\Sigma > \Sigma'$  then  $\mathcal{P}_{\Sigma} \leq \mathcal{P}_{\Sigma'}$ .

*Proof:* See Appendix B.  $\square$

The second metric measures the load imbalance across the storage nodes. In the balls-into-bins model, load imbalance is

quantified by the number of balls in the maximally loaded bin. Our metric is a continuous generalization of this. In addition, we relate the load on the maximally loaded node to its smallest possible value. This makes our metric independent of the cumulative offered load.

**Definition 4 (Measure of Load Imbalance):** Consider the system with  $n$  storage nodes operating under a cumulative load of  $\Sigma$ . Load imbalance factor  $\mathcal{I}$  for the system is defined as minimum of the maximal load on any node over all feasible loads, divided by its minimum possible value, i.e.,  $\Sigma/n$ .

It is given as

$$\mathcal{I} = \frac{\|\mathbf{M} \cdot \mathbf{x}^*\|_{\infty}}{\Sigma/n}. \quad (6)$$

where  $\mathbf{M}$  is the binary matrix representing the system's storage allocation (as described in Sec. II-C), and  $\mathbf{x}^*$  is the solution for the minimization program given in (4).

Notice that  $\mathcal{I}$  is always  $\geq 1$ . We abstract away the resource sharing dynamics and other system related aspects and evaluate performance through the metrics of load imbalance. It should be noted that this is the same approach taken in previous studies with balls-into-bins models. Even though much complexity is abstracted away from the system model, load balance is a good proxy to get an understanding of system's performance in terms of response time. This is because the more evenly the load is balanced, the smaller the system's response time is. We argue this for illustrative purposes as follows. Let us suppose that resource sharing at each storage node is implemented with a first-come first-served (FCFS) or a processor sharing (PS) queue. Response time at a node will then get increasingly larger and more variable the greater the load is to the node. Let us denote the average response time of node  $i$  under an average offered load of  $\rho_i \in (0, 1)$  with  $T(\rho_i)$ . We know that under either FCFS or PS queue,  $T(\rho)$  will scale as  $\rho/(1 - \rho)$ , which is a convex monotonically increasing function of  $\rho$ . Roughly,  $\rho_i/\sum_{i=1}^n$  fraction of the request arrivals will experience an average response time of  $T(\rho_i)$ . By Jensen's inequality, we find for all non-negative  $\rho_i$  that

$$\sum_{i=1}^n \frac{\rho_i}{\sum_{i=1}^n \rho_i} T(\rho_i) \geq T\left(\frac{1}{n} \sum_{i=1}^n \rho_i\right).$$

Notice that the right hand side of the above inequality is the average system response time when the load is perfectly balanced across the nodes. This tells us that perfect load balance minimizes the average response time under any cumulative offered load. The same argument given above can be used to observe that load balance is desirable to optimize other convex quantities such as the second moment of response time.

Lemma 4 given in the previous sub-section says that storing an additional redundant object copy in the system expands the service capacity region that is supported by the system. Keep in mind that the total capacity in the system does not change but, by creating additional storage redundancy, we are able to use the available capacity more efficiently for content access. This helps to show how increasing storage redundancy improves load balancing performance in terms of  $\mathcal{P}_{\Sigma}$  and  $\mathcal{I}$ .

**Corollary 2:** Consider a system with load balancing performance of  $\mathcal{P}_{\Sigma}$  and  $\mathcal{I}$ . Suppose a new redundant object copy is

stored in the system as described in Lemma 4, and let  $\mathcal{P}'_\Sigma, \mathcal{I}'$  represent the metrics of load balancing performance for the modified system. Then we have  $\mathcal{P}'_\Sigma \geq \mathcal{P}_\Sigma$  and  $\mathcal{I}' \leq \mathcal{I}$ .

*Proof sketch:* By Lemma 4,  $\mathcal{C} \subset \mathcal{C}'$ . This together with (5) directly implies  $\mathcal{P}'_\Sigma \geq \mathcal{P}_\Sigma$ . In order to see  $\mathcal{I} \geq \mathcal{I}'$ , it is enough to observe the following. For an arbitrary object demand vector  $\rho$ , let  $\mathbf{x}$  be the demand portion vector that minimizes the load on the maximally loaded node in the unmodified system. Given that  $\mathcal{C} \subset \mathcal{C}'$ ,  $\mathbf{x}$  is also achievable by the modified system.  $\square$

### E. Note on the Proofs and the Notation

We place the proofs in the Appendix, in order not to disrupt continuity of the text. Throughout the paper,  $\log$  refers to the natural logarithm, and  $\log_{(i)}$  refers to  $i$  times iterated logarithm, e.g.,  $\log_{(2)}(x)$  stands for  $\log \log(x)$ . We denote convergence of a sequence using the “ $\rightarrow$ ” notation. Suppose that  $\{f_n(x); n \geq 1\}$  is a sequence of functions  $f_n : D \rightarrow \mathbb{R}$  and  $f : D \rightarrow \mathbb{R}$ . If  $\lim_{n \rightarrow \infty} f_n(x) = f(x)$  at every  $x \in D$ , we will denote this as  $f_n(x) \rightarrow f(x)$ . Throughout this paper, we will rely on almost sure convergence in probability. Suppose that  $X_1, X_2, \dots$  is a sequence of random variables in a sample space  $\Omega$  and suppose  $X$  is another random variable in  $\Omega$ . Then  $\{X_n; n \geq 1\}$  converges to  $X$  almost surely if

$$\Pr\left\{\omega \in \Omega : \lim_{n \rightarrow \infty} X_n(\omega) = X(\omega)\right\} = 1.$$

We denote almost sure convergence of  $X_n$  to  $X$  as  $X_n \rightarrow X$  a.s.

## III. LOAD BALANCING WITH NO REDUNDANCY

In this section we consider *single-choice* allocations in which each of the  $k$  objects is stored on only a single node and each of the  $n$  nodes stores  $m = k/n$  different objects. We assume  $n|k$ . Demand for each object in this case has to be completely served by the only node hosting the object, and each node has to serve the total demand for all objects stored on it.

As discussed in Sec. II-B, object demand vector  $(\rho_1, \dots, \rho_k)$  can be described by  $k$  uniform spacings in  $[0, \Sigma]$ . Given that uniform spacings are exchangeable RV's, we can say without loss of generality that if node  $s_i$  stores objects  $o_{(i-1)m+1}, \dots, o_{im}$ , then load  $l_i$  exerted on  $s_i$  is given by  $l_i = \sum_{j=(i-1)m+1}^{im} \rho_j$ . For the system to be stable, all  $l_i$  must be  $< 1$ . Thus  $\max\{l_1, \dots, l_n\} < 1$  is necessary and sufficient for system stability. This implies  $\mathcal{P}_\Sigma$  for the system is given by  $\Pr\{\max\{l_1, \dots, l_n\} < 1\}$ . In the uniform spacing literature, random variables (RV's)  $l_i$  have been studied and are called non-overlapping  $m$ -spacings. Their maximum is referred to as the *maximal non-overlapping  $m$ -spacing*.

*Definition 5:* Maximal non-overlapping  $m$ -spacing for  $k$  uniform spacings on the unit line is defined for  $k = m \cdot n$  as

$$M_{k,m}^{(\text{no})} = \max_{i=1, \dots, n} U_{(im)} - U_{((i-1)m)},$$

or as

$$\max_{i=1, \dots, n} \sum_{j=(i-1)m+1}^{im} S_j.$$

In a system of  $n$  nodes storing  $k$  objects under a cumulative demand of 1, load on the maximally loaded node is given by  $M_{k,m}^{(\text{no})}$ . Using a combination of the ideas presented in [29]–[31], we can derive the following convergence results for  $M_{k,m}^{(\text{no})}$ .

*Lemma 5:* For fixed  $m$ , as  $n \rightarrow \infty$

$$\Pr\left\{M_{k,m}^{(\text{no})} \cdot mn - \log(n) - f_n < x\right\} \xrightarrow{d} G(x), \quad (7)$$

where  $G(x) = \exp(-\exp(-x))$  is the standard Gumbel function and  $f_n = (m-1) \log_{(2)}(n) - \log((m-1)!)$ .

In the limit  $n \rightarrow \infty$ , the following inequality holds

$$M_{k,m}^{(\text{no})} \leq \frac{\log(n)}{mn} + O\left(\frac{\log_{(2)}(n)}{n}\right) \text{ a.s.} \quad (8)$$

Furthermore, as  $n \rightarrow \infty$

$$\frac{M_{k,m}^{(\text{no})} \cdot mn}{\log(n)} \rightarrow 1 \text{ a.s.} \quad (9)$$

*Proof:* See Appendix D.  $\square$

Now we are ready to express the metrics  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  for a system with single-choice allocation.

*Lemma 6:* In a system with a single-choice storage allocation,

$$\mathcal{P}_\Sigma = \Pr\left\{M_{k,m}^{(\text{no})} < 1/\Sigma\right\}, \quad \mathcal{I} = M_{k,m}^{(\text{no})} \cdot n. \quad (10)$$

*Proof:* When system operates under a cumulative offered load of  $\Sigma$ , the load on the maximally loaded node is given by  $M_{k,m}^{(\text{no})} \cdot \Sigma$ . This together with the definition of  $\mathcal{P}_\Sigma$  (Def. 3) and  $\mathcal{I}$  (Def. 4) gives us (10).  $\square$

Using Lemma 6 and Lemma 5, we determine the behavior of  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  for large  $n$  as follows:

*Theorem 2:* Consider a system with single-choice storage allocation. For fixed  $m$ , as  $n \rightarrow \infty$

$$\Pr\left\{\mathcal{I} \cdot m - \log(n) - f_n < x\right\} \rightarrow G(x), \quad (11)$$

$$\frac{\mathcal{I} \cdot m - f_n}{\log(n)} \rightarrow 1 \text{ a.s.} \quad (12)$$

where  $f_n = (m-1) \log_{(2)}(n) - \log((m-1)!)$ .

If  $\Sigma_n = b_n \cdot n / \log(n)$  for some sequence  $b_n > 0$ , then as  $n \rightarrow \infty$

$$\mathcal{P}_{\Sigma_n} \rightarrow \begin{cases} 1 & \limsup b_n < m, \\ 0 & \liminf b_n > m. \end{cases} \quad (13)$$

*Proof:* See Appendix I.  $\square$

*Remark 1:* Theorem 2 implies for a system of  $n$  nodes with no redundancy and fixed  $m$  that load imbalance  $\mathcal{I} = \Theta(\log(n))$ . This is due to the following fact. As  $n$  increases, maximal load on any node decays as  $\Sigma/n$  in the case with perfect load balancing. However due to the skewness in popularity, demands for the popular objects go down as  $\log(n)/n$  (recall Theorem 1). This is why the load imbalance in the system grows with  $n$  as  $\log(n)$ . The scaling of load imbalance with  $\log(n)$  as we find here is aligned with the well-known result derived in the dynamic load balancing setting: if  $n$  balls arrive sequentially and each is placed into one of the  $n$  bins randomly, the maximally loaded bin will end up with  $\Theta(\log(n)/\log_{(2)}(n))$  balls w.h.p. [11].

In addition (12) shows that  $\mathcal{I}$  decays multiplicatively with  $m$  for large scale systems. This implies that in large scale systems, if the system can support a cumulative load of  $\Sigma$  while storing  $k$  objects, it will also be able to support a cumulative load of  $r \times \Sigma$  while storing  $k \times r$  objects.

#### IV. LOAD BALANCING WITH $d$ -FOLD REDUNDANCY

In this section we consider  $d$ -choice allocations in which each of the  $k$  objects is stored on  $d$  different nodes ( $d$ -choices) and each of the  $n$  nodes stores  $kd/n$  different objects.

As discussed in Remark 1, load imbalance  $\mathcal{I}$  in the system decays with the number of objects ( $m$ ) stored per node. We here, and also in Sec. VI, consider the worst case for load balancing, that is  $k = n$ . This makes the problem more tractable to formulate and study the load balancing problem. This also makes it easier to explain and interpret the derived results. Results that we present now can be extended for the general case with a fixed value of  $k/n > 1$  by using arguments that are very similar to those we discuss.

In what follows, we first define and discuss *maximal  $d$ -spacing*. It is a mathematical object defined in terms of uniform spacings and is instrumental while deriving our results. We then present our results on the load balancing performance of systems with  $d$ -choice allocations.

##### A. Uniform Spacings Interlude

*Definition 6 ( $M_{k,d}$ ):* Maximal  $d$ -spacing within  $k$  uniform spacings on the unit line is defined as

$$M_{k,d} = \max_{i=0, \dots, k-d} U_{(i+d)} - U_{(i)} \quad \text{or} \quad \max_{i=1, \dots, k-d+1} \sum_{j=i}^{i+d-1} S_j,$$

where  $d$  is any integer in  $[1, k]$ , and  $U_0 = 0$  and  $U_k = 1$  as given in Sec. II-B.

It is worth to note that maximal  $d$ -spacing  $M_{k,d}$  defined above is the overlapping counterpart of the maximal non-overlapping  $m$ -spacing  $M_{k,m}^{(\text{no})}$  defined in Def. 5. We extensively use the results presented in [30], [32]–[34] on  $M_{k,d}$  while deriving our main results. We state the ones that we use in the remainder.

*Lemma 7* ([34, Theorem 1]): For any integer  $d \geq 1$ , as  $k \rightarrow \infty$

$$\Pr \left\{ M_{k,d} \cdot k - \log(k) - (d-1) \log_2(k) - \log((d-1)!) \leq x \right\} \rightarrow G(x). \quad (14)$$

*Lemma 8* ([32, Theorem 2, 6]): For  $d = o(\log(k))$ , as  $k \rightarrow \infty$

$$\frac{M_{k,d} \cdot k - \log(k)}{(d-1) \left( 1 + \log_2(k) - \log(d) \right)} \rightarrow 1 \quad \text{a.s.} \quad (15)$$

For  $d = c \log(k) + o(\log_2(k))$  with some constant  $c > 0$ ,

$$V_k = \frac{M_{k,d} \cdot k - (1+\alpha)c \cdot \log(k)}{\log_2(k)}$$

satisfies

$$\begin{aligned} \limsup_{k \rightarrow \infty} V_k &= \beta^*(1+\alpha)/\alpha \quad \text{a.s.} \\ \liminf_{k \rightarrow \infty} V_k &= -\beta^\dagger(1+\alpha)/\alpha \quad \text{a.s.} \end{aligned} \quad (16)$$

where  $\alpha$  is the unique positive solution of  $e^{-1/c} = (1+\alpha)e^{-\alpha}$ , and  $\beta^*$  and  $\beta^\dagger$  are constants taking values in  $[-0.5, 1.5]$  and  $[-1.5, -0.5]$  respectively.

The uniform spacings we have considered so far are defined on the unit line segment. However, in order to evaluate load balancing performance in systems with  $d$ -choice storage allocation, we need to consider uniform spacings on the *unit circle*.

*Definition 7 ( $M_{k,d}^{(c)}$ ):* Maximal  $d$ -spacing within  $k$  uniform spacings on the unit circle is defined as

$$M_{k,d}^{(c)} = \max_{i=1, \dots, k} \sum_{j=i}^{i+d-1} S_i, \quad \text{where } S_i = S_{i-k} \text{ for } i > k.$$

We show in Appendix E that results stated in Lemma 7 and 8 for maximal  $d$ -spacing on the unit line carry over to its counterpart defined on the unit circle.

##### B. Evaluating $\mathcal{P}_\Sigma$ and $\mathcal{I}$ for $d$ -Choice Storage Allocation

A  $d$ -choice storage allocation defines a  $d$ -regular balanced bipartite mapping from the set of objects to the set of nodes, which we refer to as the *allocation graph*. Its construction can be described as follows: i) Map primary copies for all objects to nodes with a bijection  $f_0$ , ii) For  $i$  going from 1 to  $d$ , map the  $i$ th redundant object copies to nodes with a bijection  $f_i$  such that  $f_i(o) \neq f_j(o)$  for every  $j < i$  and  $o$ . Thus every node stores a single primary and  $d-1$  redundant object copies, and each copy stored on the same node is for a different object. We refer to a node with the index of the primary object copy stored on it, i.e., object  $o_i$  is hosted primarily on  $s_i$ . The subscript in  $s_i$  or  $o_i$  will implicitly denote  $i \bmod n$  throughout. We denote the set of nodes that host object  $o_i$  with  $C_i$ . In other words  $C_i$  consists of the service choices available for  $o_i$ .

The number of service choices available for the objects is not the only factor that impacts load balancing performance in storage systems. Layout of the content across the storage nodes also plays a role in load balancing. Given that the total number of object copies stored in the system is greater than the number of storage nodes, we need to have  $|C_i \cap C_j| > 0$  for some  $j \neq i$ . Overlaps between  $C_i$  might lead to contention when the content popularity is skewed towards objects with overlapping service choices. Both the number of overlapping  $C_i$  and the size of the overlaps should be minimized in order to improve load balancing performance. However, size and number of overlaps cannot be reduced together given a fixed number of nodes in the system. For every regular balanced  $d$ -choice allocation with object replicas we have

$$\sum_{i=1}^k \sum_{j \neq i} |C_i \cap C_j| = (d-1)d \cdot k, \quad (17)$$

where  $|\cdot|$  denotes the cardinality of a set. This equality follows by observing that each node serves as a service choice for  $d$  different objects and each is counted in exactly  $d - 1$  of the overlaps  $C_i \cap C_j$ . It should be noted that the sum in (17) is equal to *twice* the cumulative cardinality of the overlaps between all pairs of  $(C_i, C_j)$ . We emphasize that cumulative cardinality of the pairwise overlaps is fixed. Hence reducing the size of overlaps between some of the sets  $C_i$  can only come at the cost of enlarging overlaps between some other  $C_j$ .

We first consider the two following simple designs for constructing  $d$ -choice storage allocations.

**Clustering design:** This design is possible only if  $d|n$ . Let us partition the nodes into  $n/d$  sets, each of which we call a *cluster*. Let us then assign each object to a cluster such that each cluster is assigned exactly  $d$  objects. Every object is stored across all the nodes within its assigned cluster. The resulting storage has an allocation graph that is composed by  $n/d$  disjoint  $d$ -regular complete bi-partite graphs. Hence we obtain a storage scheme with  $d$ -choice allocation. For instance, 3-choice allocation for 9 objects  $a, \dots, i$  with cyclic design would look like

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \begin{bmatrix} d \\ e \\ f \end{bmatrix} \quad \begin{bmatrix} d \\ e \\ f \end{bmatrix} \quad \begin{bmatrix} d \\ e \\ f \end{bmatrix} \quad \begin{bmatrix} g \\ h \\ i \end{bmatrix} \quad \begin{bmatrix} g \\ h \\ i \end{bmatrix} \quad \begin{bmatrix} g \\ h \\ i \end{bmatrix}.$$

**Cyclic design:** In this design we follow a *cyclic* construction. We start by assigning the original object copies to the nodes according to an arbitrary bijection  $f_0$ . We assign the  $i$ th replicas of the objects for  $i = 1, \dots, d - 1$  by using bijection  $f_i$ , where  $f_i$  is obtained by applying circular shift on  $f_0$  repeatedly  $i$  times. Note that shifting is applied in the same direction in all the steps. In other words, we pick  $f_i$  such that  $f_{i+1}(o) = f_i(o) + 1 \bmod n$  for  $i = 0, \dots, d - 1$  and every  $o$ . For instance, 3-choice allocation for 7 objects  $a, \dots, g$  with cyclic design would look like

$$\begin{bmatrix} a \\ g \\ f \end{bmatrix} \quad \begin{bmatrix} b \\ a \\ g \end{bmatrix} \quad \begin{bmatrix} c \\ b \\ a \end{bmatrix} \quad \begin{bmatrix} d \\ c \\ b \end{bmatrix} \quad \begin{bmatrix} e \\ d \\ c \end{bmatrix} \quad \begin{bmatrix} f \\ e \\ d \end{bmatrix} \quad \begin{bmatrix} g \\ f \\ e \end{bmatrix}. \quad (18)$$

For a given set of objects  $S$ , the union of their choices  $C_i$  forms the *node expansion* of  $S$ , which we denote by  $N(S)$ . If  $|N(S)| = x$ , then there is at most  $x$  amount of capacity available for the joint use of the objects within  $S$ . It is surely impossible to stabilize the system when the cumulative demand for  $S$  is greater than  $N(S)$ . Thus it is desirable to increase the size of the node expansions in the allocation graph in order to guarantee stability for larger skews in content popularity. Greater expansion for a given  $S$  requires reducing the size of the overlaps between the choices  $C_i$  for the objects in  $S$ . This would imply overlapping the choices  $C_i$  of objects within  $S$  with the choices  $C_j$  of objects outside of  $S$ .

It is not easy to define a knob that regulates both the overlaps between object choices  $C_i$  and the node expansions in the allocation graph. We next define a class of allocations in

which the overlaps and node expansions are loosely controlled by a single parameter.

**Definition 8 (r-Gap Design):** An allocation is an  $r$ -gap design if  $|C_i \cap C_j| = 0$  for  $j > i$  and  $\min\{j-i, n-(j-i)\} > r$ .

**Lemma 9:** In a  $d$ -choice allocation with  $r$ -gap design,  $r \geq d - 1$  and  $x \leq |N(S)| \leq x + 2r$  for any  $S = \{o_i, o_{i+1}, \dots, o_{i+x-1}\}$  for any  $i$ .

*Proof:* See Appendix F.  $\square$

We can use the properties of  $r$ -gap design to find necessary and sufficient conditions for the stability of a storage system.

**Lemma 10:** Consider a system with  $d$ -choice storage allocation that is constructed with an  $r$ -gap design and operating under a cumulative demand of  $\Sigma$ . Then for system stability, a necessary condition is given as

$$M_{n,i}^{(c)} \leq (i + 2r)/\Sigma, \quad \text{for any } i = 1, \dots, n - 2r, \quad (19)$$

and a sufficient condition is given as

$$M_{n,r+1}^{(c)} \leq d/\Sigma. \quad (20)$$

In other words, we have for  $i = 1, \dots, n - 2r$

$$\Pr\left\{M_{n,r+1}^{(c)} \leq d/\Sigma\right\} \leq \mathcal{P}_\Sigma \leq \Pr\left\{M_{n,i}^{(c)} \leq (i + 2r)/\Sigma\right\}.$$

*Proof:* See Appendix G.  $\square$

Notice that clustering or cyclic design is an  $r$ -gap design. Hence the necessary and sufficient conditions given in Lemma 10 for system stability are valid for storage allocation with either design. In addition, the well-defined structure of these two designs allows us to refine the results given in Lemma 10 as follows.

**Lemma 11:** In a  $d$ -choice allocation constructed with clustering or cyclic design

$$\Pr\left\{M_{n,d}^{(c)} \leq d/\Sigma\right\} \leq \mathcal{P}_\Sigma \leq \Pr\left\{M_{n,d+1}^{(c)} \leq 2d/\Sigma\right\}.$$

*Proof:* See Appendix H.  $\square$

Using the bounds given in Lemma 11, we can find an asymptotic characterization for  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  as follows.

**Theorem 3:** Consider a system with  $d$ -choice storage allocation constructed with clustering or cyclic design.

When  $d = o(\log(n))$ , the following inequality hold in the limit as  $n \rightarrow \infty$

$$\frac{1}{2} \leq \frac{\mathcal{I} \cdot d}{\log(n) + (d-1)(1 + \log_2(n) - \log(d))} \leq 1 \quad \text{a.s.} \quad (21)$$

If  $\Sigma_n = b_n \cdot n/\log(n)$  for some sequence  $b_n > 0$ , then as  $n \rightarrow \infty$

$$\mathcal{P}_{\Sigma_n} \rightarrow \begin{cases} 1 & \limsup b_n/d < 1, \\ 0 & \liminf b_n/2d > 1. \end{cases} \quad (22)$$

When  $d = c \log(n)$  for some constant  $c > 0$ , the following inequality holds in the limit  $n \rightarrow \infty$

$$\frac{1}{6} \leq \frac{2c\alpha}{3(\alpha+1)} \cdot \frac{\mathcal{I} \cdot \log(n)}{\log_2(n)} \leq 1 \quad \text{a.s.}, \quad (23)$$

where  $\alpha$  is the unique positive solution of  $e^{-1/c} = (1+\alpha)e^{-\alpha}$ . If  $\Sigma_n = b_n \cdot n / \log(n)$  for some sequence  $b_n > 0$ , then as  $n \rightarrow \infty$

$$\mathcal{P}_{\Sigma_n} \rightarrow \begin{cases} 1 & \limsup b_n \cdot 1.5\tau/d < 1, \\ 0 & \liminf b_n \cdot 0.25\tau/d > 1, \end{cases} \quad (24)$$

where  $\tau = c(1 + \alpha)^2 / \alpha$ .

*Proof:* See Appendix J  $\square$

*Remark 2:* Theorem 3 implies that load imbalance  $\mathcal{I} = \Theta(\log(n)/d)$  when  $d = o(\log(n))$ , and  $\mathcal{I} = \Theta(\log \log(n)/\log(n))$  when  $d = \Theta(\log(n))$ . These imply that i)  $d$  choices for each object initially reduces load imbalance multiplicatively by  $d$ , ii) there is an exponential reduction in load imbalance as soon as  $d$  reaches of order  $\log(n)$ , that is, what we started with  $\mathcal{I} = \Theta(\log(n))$ , for  $d = 1$ , is exponentially larger than what we end up with  $\mathcal{I} = \Theta(\log \log(n)/\log(n))$  for  $d = \Theta(\log(n))$ . The second implication extends Godfrey's first observation in [14] to the static setting under general offered load. Godfrey derived his results with the dynamic balls-into-bins model under light offered load, i.e., when  $O(n)$  balls are sequentially placed into  $n$  bins. Godfrey's second observation implies in the dynamic setting that for any  $1 < d < \Theta(\log(n)/\log \log(n))$ , there exists a  $d$ -choice storage allocation such that incrementing  $d$  from 1 to 2 will not yield exponential reduction in load imbalance. The first implication given above, shows this observation of Godfrey for a concrete storage design and extends it to the static setting under general offered load.

In what follows, we discuss some of our arguments using simulation results on  $\mathcal{I}$ . We compute  $\mathcal{I}$  by taking an average of its values obtained from  $10^5$  simulation runs. Within each simulation run, the object demand vectors that are offered to the system are sampled uniformly at random from the simplex  $\mathcal{S}_\Sigma$ , which is defined by a fixed  $\Sigma$  as in (1).

Fig. 2 plots  $\mathcal{I}$  for  $n = 100$ . Notice that  $\mathcal{I}$  is close to  $\log(n)$  when  $d = 1$  as suggested by Theorem 2. As  $d$  is incremented,  $\mathcal{I}$  decays as  $1/d$  as suggested by Theorem 3. This illustrates that our asymptotic results are close estimates for the finite case.

Constructions with  $r$ -gap designs decouple object choices  $C_i$  that are  $r$ -apart at the cost of enlarging the overlaps between those that are close to each other, as in the clustering or cyclic designs. The Balanced Incomplete Block Designs (BIBD) allow control of the overlaps between every pair of  $C_i$ .

*Definition 9 (BIBD, [24]):* A  $(d, \lambda)$  block design is a class of equal-size subsets of  $\mathcal{X}$  (the set of stored objects), called blocks (storage nodes), such that every point in  $\mathcal{X}$  appears in exactly  $d$  blocks (service choices), and every pair of distinct points is contained in exactly  $\lambda$  blocks.

Since we assume  $k = n$ , the block designs we consider are symmetric. A symmetric BIBD with  $\lambda = 1$  guarantees that  $|C_i \cap C_j| = 1$  for every  $j \neq i$ . Since this case represents the minimal overlap between sets  $C_i$  we focus on this case. The block design we consider refers to  $(d, 1)$  symmetric BIBD, that is every object appears in  $d$  nodes and every pair of distinct objects is contained in exactly one node. Since every pair of

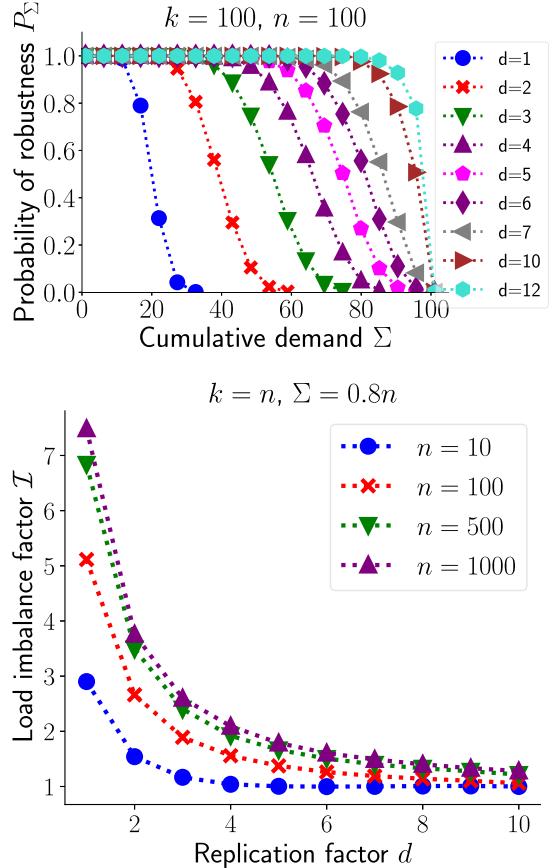


Fig. 2. Simulated average values for  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  for a system that implements  $d$ -choice allocation with cyclic design. Cumulative offered load on the system is set to  $\Sigma = 0.8n$ . System stores  $k = n$  objects across  $n$  nodes.

$C_i$  overlaps at one node, we have

$$\sum_{i=1}^k \sum_{j \neq i} |C_i \cap C_j| = (k-1)k.$$

Then by (17), such block designs are possible only if  $k = d^2 - d + 1$ . For instance a 3-choice allocation with a block design is given as

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} a \\ f \\ g \end{bmatrix}, \begin{bmatrix} a \\ d \\ e \end{bmatrix}, \begin{bmatrix} b \\ d \\ f \end{bmatrix}, \begin{bmatrix} b \\ e \\ g \end{bmatrix}, \begin{bmatrix} c \\ d \\ g \end{bmatrix}, \begin{bmatrix} c \\ e \\ f \end{bmatrix} \quad (25)$$

The sufficient and necessary conditions presented in Lemma 10 cannot be used on a storage allocation with a block design, since they are not  $r$ -gap designs. However using ideas that are similar to those used to derive Lemma 10, we can find the following conditions for system stability.

*Lemma 12:* Consider a system with  $d$ -choice allocation constructed with a block design and operating under a cumulative offered load of  $\Sigma$ . For stability of the system, a necessary condition is given as  $M_{n,d}^{(c)} \leq (d^2 - 2d + 3)/\Sigma$  and a sufficient condition is given as  $M_{n,d}^{(c)} \leq d/2\Sigma$ .

*Proof:* See Appendix K.  $\square$

The stability conditions given in Lemma 12 allow us to find bounds on  $\mathcal{P}_\Sigma$  and  $\mathcal{I}$  for storage allocations with block

designs, similar to those that were stated in Theorem 3. We do not state them here since they are obtained by simply modifying the multiplicative factors in the bounds given in Theorem 3. The upper bound on  $\mathcal{I}$  in this case decays as  $1/d$  with increasing  $d$ , which says that providing  $d$  service choices for each object initially reduces load imbalance at least multiplicatively by  $d$ . However, the lower bound on  $\mathcal{I}$  decays in this case as  $1/d^2$ , that is, block designs can possibly implement better scaling of  $\mathcal{I}$  in  $d$  compared to clustering or cyclic designs.

Our asymptotic analysis does not allow ordering different designs of  $d$ -choice storage allocations in terms of their load balancing performance. As discussed previously, all  $d$ -choice allocations yield the same cumulative overlap between object choices  $C_i$  (recall (17)) and each design gives a different way of distributing the overlaps across object choices  $C_i$ . With simulations we find that it is better to evenly spread the overlaps between choices  $C_i$  using a block design, that is, *many but consistently small overlaps are better than fewer but occasionally large overlaps*. Fig. 3 shows the average  $\mathcal{I}$  for systems with 3- and 5-choice allocations that are constructed using clustering, cyclic or block designs. We see here, and in other simulation results we omit, that the largest gain in load balancing is achieved by moving from clustering to cyclic, while moving further to block design yields a smaller gain in  $\mathcal{I}$ . Furthermore cyclic designs exist for any value of  $k$ ,  $n$  and  $d < n$ , while block designs exist only for a restricted set of  $k$ ,  $n$  and  $d$ . Cyclic design therefore appears to be favorable for constructing multiple-choice storage allocations in real systems.

Currently we don't have a rigorous way to understand how designs with different overlaps compare with each other in terms of  $\mathcal{P}_\Sigma$  or  $\mathcal{I}$ . In the following subsection, we present our intuitive reasoning on why consistently small overlaps is better in terms of load balancing than shrinking overlaps between some objects and making them larger for others.

### C. On the Impact of Overlaps Between the Service Choices

Storage redundancy allows the system to split the demand for the popular objects across multiple nodes, hence enabling the system to achieve better load balance across the nodes in the presence of skews in object popularities. In order to minimize the risk of overburdening a storage node, a natural strategy would be to decouple the overlaps between the service choices ( $C_i$ ) for the objects that are expected to be more popular than others. In this paper we assume no a priori knowledge on the object popularities; in particular we assume cumulative demand remains constant at  $\Sigma$  while all possible object popularity vectors are equally likely, which implies that the object demands are distributed as the uniform spacings within  $[0, \Sigma]$  (Sec. II-A). Our model then seeks to answer how one should design the overlaps between the service choices of objects when no a priori knowledge is available on the object popularities.

Recall from (17) that all  $d$ -choice allocations yield the same cumulative overlap between object choices  $C_i$  and each design gives a different way of distributing the overlaps across  $C_i$ .

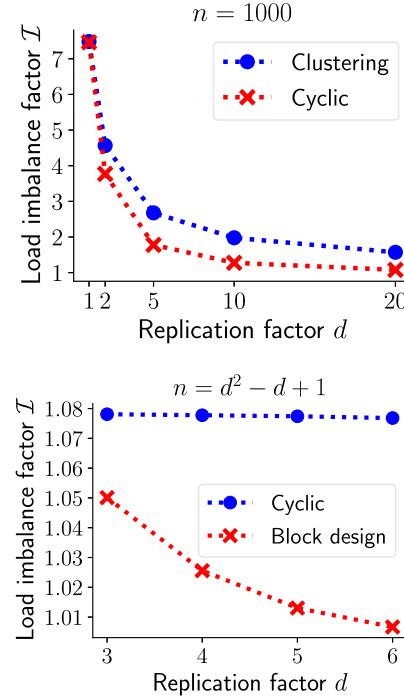


Fig. 3. Simulated average value of  $\mathcal{I}$  for  $d$ -choice allocation with different designs. Note that clustering and block designs do not co-exist for the same  $d$  and  $n$ .

With simulations (as presented in Fig. 3) we found that in order to achieve higher load balancing performance, it is better to spread the overlaps evenly across all pairs of objects (using block design) than distributing them in an unbalanced manner by implementing smaller overlaps between some objects while implementing larger number of overlaps between others (such as using clustering or cyclic design). Reducing the overlaps between the service choices for a given set of objects  $S$  enlarges the node expansion of  $S$  (as explained in detail in Sec. IV-B), hence increasing the capacity available for jointly serving the objects within  $S$ . However this leads to a reduction in the node expansion for other sets of objects, hence reducing the capacity available for the joint use of those objects.

Overall reducing the service choice overlaps between the objects that are known to be more popular than others will allow the system to balance the offered load, which is expected to be skewed towards the popular objects, more effectively. However reducing the overlaps for a particular set of objects is risky when we don't know which objects are going to be more popular, because this would increase the overlaps for other sets of objects, one of which might end up being the true set that is more popular than others. This is exactly the case implemented in our offered load model; few objects will be highly popular while most of them will have average popularity (as implied by *R3* and *R1*) and we don't know a priori which of those that are highly popular. When no information is available on which objects will be more popular, it is not possible to select the popular objects and reduce the overlaps between their service choices. Then the natural strategy would be to avoid the risk of large overlaps. Indeed the simulations show for our case with no a priori knowledge on object popularities that

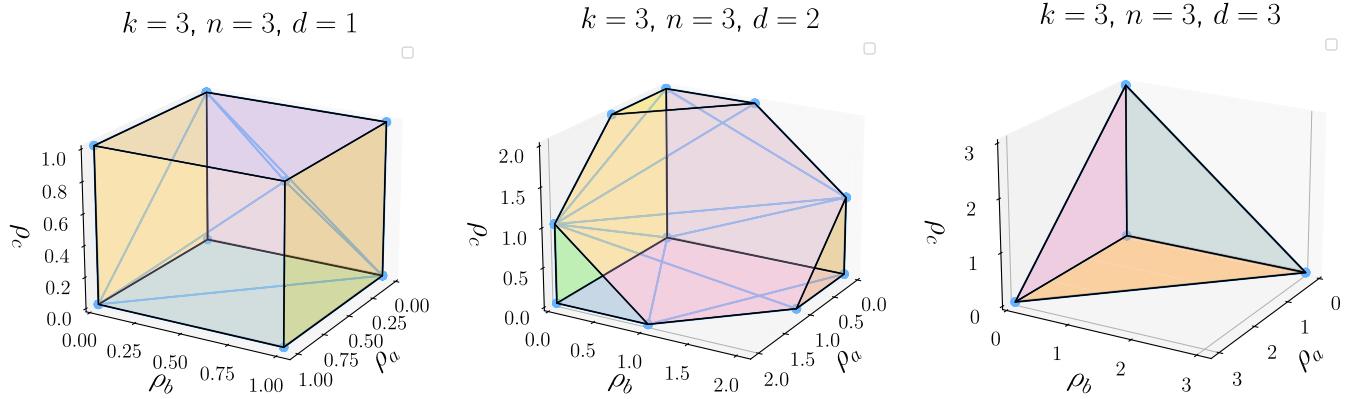


Fig. 4. Service capacity region of regular balanced  $d$ -choice allocation for  $d = 1, 2, 3$ .

allocating the service choices for a group of sets of objects with smaller service choice overlap (as in clustering or cyclic design) performs on average worse than treating all objects the same and minimizing the overlaps across the service choices of all pairs of objects (as in block designs).

The rationale of favoring many but consistently small overlaps over fewer but occasionally larger overlaps has very recently been observed to perform well also in the context of scheduling compute jobs with bi-modal job size distribution. The authors in [35] consider replicating every arriving job (ball) across  $r$  nodes (bins), in which the overlaps between the sets of nodes assigned to subsequent jobs impact queueing times at the nodes. The authors observed that the most effective way to control the overlaps across the subsequent node-assignment rounds is to use a block design, which balances the large jobs across the nodes more effectively than cyclic or random job-to-node assignment strategies.

## V. INTERPRETING LOAD BALANCING PERFORMANCE WITH THE SHAPE OF SERVICE CAPACITY REGION

Fig. 4 plots the capacity region  $\mathcal{C}$  for a system of three servers and three objects with  $d$ -choice allocation constructed with cyclic design for  $d = 1, 2, 3$  (the cyclic design was introduced in Sec. IV-B). When  $d = 1$ ,  $\mathcal{C}$  is given by the standard unit cube. Setting  $d = 2$  extends  $\mathcal{C}$  by a unit length at the *skew* corners that lie on coordinate axes. We call them *skew* corners because the object demand vectors that are close to the corners represent the load scenarios with skewed object popularities. Setting  $d = 3$  extends the *skew* corners by an additional unit of length and yields a simplex capacity region. This implies that the total capacity that is available in the system (which is 3 in this example) can be arbitrarily used for serving any stored object when  $d = 3$ , i.e., when each object is available at every server.

Previously, we observed that incrementing  $d$  from one to two yields the greatest increase in the system's load balancing performance and further increments yield diminishing gains (cf. Fig. 2). We now investigate this through the geometric interpretation of  $\mathcal{P}_\Sigma$  that was given in (5). As a Corollary of Lemma 4, the capacity region for  $d$ -choice allocation is contained by that for  $(d + 1)$ -choice allocation. This can be

seen for the example given in Fig. 4. Recall that  $\mathcal{S}_\Sigma$  is the  $k - 1$  dimensional standard simplex of side length  $\Sigma$  as defined in (1) and  $\mathcal{C}$  is the  $k$  dimensional polytope representing the system capacity region.  $\mathcal{P}_\Sigma$  is proportional to  $\text{Vol}(\mathcal{A})$  where  $\mathcal{A} := \mathcal{S}_\Sigma \cap \mathcal{C}$  (by (5)), which increases with  $d$ , hence  $\mathcal{P}_\Sigma$  increases with  $d$ .  $\mathcal{A}$  is a  $k - 1$  dimensional polytope such that  $\mathcal{S}_\Sigma$  and  $\mathcal{A}$  share the same (Chebyshev<sup>1</sup>) center. Again examples in Fig. 4 help seeing this.

In order to better understand the effect of incrementing  $d$  on the load balancing performance, let us go through the examples given in Fig. 4. Suppose  $\Sigma = 3$ . Then we have  $\text{Vol}(\mathcal{S}_\Sigma) = 9\sqrt{3}/2$ . When  $d = 1$ ,  $\mathcal{A} = \{(1, 1, 1)\}$  and  $\text{Vol}(\mathcal{A}) = 0$ , hence  $\mathcal{P}_\Sigma = 0$ . When  $d = 2$ ,  $\mathcal{A}$  is a polygon with the set of vertices

$$\{(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)\}$$

and  $\text{Vol}(\mathcal{A}) = 3\sqrt{3}$ , hence  $\mathcal{P}_\Sigma = 2/3$ . When  $d = 3$ ,  $\mathcal{A} = \mathcal{S}_\Sigma$ , hence  $\mathcal{P}_\Sigma = 1$ . This geometric view can be extended to larger dimensions. Incrementing  $d$  extends  $\mathcal{C}$  by a unit length in the skew corners, which also expands  $\mathcal{A}$ . This expansion in  $\mathcal{A}$  happens outward from its center equally in every direction when  $d$  is small. However, the boundary of  $\mathcal{S}_\Sigma$  does not allow expansion in every direction beyond a value of  $d$ . Furthermore, the shape of  $\mathcal{S}_\Sigma$  causes the expansion per increment in  $d$  to diminish in volume as  $d$  gets larger. Thus, as  $\text{Vol}(\mathcal{A})$  increases, the increase in  $\mathcal{P}_\Sigma$  per increment in  $d$  diminishes as  $d$  gets larger.

## VI. $d$ -FOLD REDUNDANCY WITH XOR'S

In this section, we will answer **Q3** posed in the Introduction. Thus far, we have only considered  $d$ -choice storage allocations with object replicas. A replicated copy adds a new service choice for only a single object, while a *coded* copy can add a new choice simultaneously for multiple objects. When an XOR of  $r$  objects (i.e.,  $r$ -XOR) is stored on a node that did not previously host any of the XOR'ed objects, each of the  $r$  objects will gain a recovery set, i.e., a set of  $r$  nodes that can jointly serve the object of interest.

<sup>1</sup>Chebyshev center  $c$  of a set  $S$  is computed by solving  $\min_{c,r} \{r \mid \|x - c\| \leq r, \forall x \in S\}$

We here consider the  $d$ -choice storage allocation with  $r$ -XOR's, which is implemented by distributing the  $k$  exact object copies and  $k(d-1)/r$  of their  $r$ -XOR'ed copies evenly across the storage nodes while complying with Def. 1. This makes sure that each object can be directly accessed through its exact copy and through  $d-1$  recovery sets. Note that we consider recovery sets that contain a single XOR'ed object, which potentially is less storage efficient than schemes that have been previously proposed based on batch codes [15]. For instance, the 3-choice allocation given in (18) with object replicas is implemented with 2-XOR's as

$$\begin{bmatrix} a \\ d+c \end{bmatrix} \begin{bmatrix} b \\ f+g \end{bmatrix} \begin{bmatrix} c \\ a+b \end{bmatrix} \begin{bmatrix} d \\ b+e \end{bmatrix} \begin{bmatrix} e \\ a+d \end{bmatrix} \begin{bmatrix} f \\ g+c \end{bmatrix} \begin{bmatrix} g \\ e+f \end{bmatrix}. \quad (26)$$

Allocation with  $r$ -XOR's reduces the storage overhead multiplicatively by  $r$ . However, object access from a recovery set requires downloading an object copy from each of the  $r$  nodes that jointly implement the choice, hence download overhead of object recovery grows multiplicatively with  $r$ . As a direct consequence of this, the load imbalance factor grows additively with  $r$  as stated in the following.

*Theorem 4:* Consider a system with  $d$ -choice storage allocation that is created with  $r$ -XOR's, where  $r \geq 2$  is an integer.

When  $d = o(\log(n))$ , the following inequality holds in the limit  $n \rightarrow \infty$

$$\frac{1}{2} \leq \frac{\mathcal{I} \cdot d}{\log(n) + \beta_{n,d}} \leq 1 \quad \text{a.s.}, \quad (27)$$

where  $\beta_{n,d} = r(d-1) \left( 1 + \log_2(n) - \log(1+r(d-1)) \right)$ , and if  $\Sigma_n = b_n \cdot n / \log(n)$  for some sequence  $b_n > 0$ , then as  $n \rightarrow \infty$

$$\mathcal{P}_{\Sigma_n} \rightarrow \begin{cases} 1 & \limsup b_n/d < 1, \\ 0 & \liminf b_n/2d > 1. \end{cases} \quad (28)$$

When  $d = c \log(n)$  for some constant  $c > 0$ , the following inequality holds in the limit  $n \rightarrow \infty$

$$\frac{1}{2} \leq \frac{\mathcal{I}}{(\alpha+1) \left( \frac{3}{2c\alpha} \cdot \frac{\log_2(n)}{\log(n)} + r \right)} \leq 1 \quad \text{a.s.}, \quad (29)$$

where  $\alpha$  is the unique positive solution of  $e^{-1/c} = (1+\alpha)e^{-\alpha}$ , and if  $\Sigma_n = b_n \cdot n / \log(n)$  for some sequence  $b_n > 0$ , then as  $n \rightarrow \infty$

$$\mathcal{P}_{\Sigma_n} \rightarrow \begin{cases} 1 & \limsup 1.5\tau \cdot b_n/d < 1, \\ 0 & \liminf 0.25\tau \cdot b_n/d > 1, \end{cases} \quad (30)$$

where  $\tau = c(1+\alpha)^2/\alpha$ .

*Proof:* See Appendix L.  $\square$

*Remark 3:* Theorem 4 implies that  $d$ -choice allocation with  $r$ -XOR's achieves the same scaling of the load imbalance factor  $\mathcal{I}$  in  $d$  as if the service choices were created with replicas (as stated in Remark 2), while also reducing the storage requirement multiplicatively by  $r$ . However, accessing an object from a recovery set requires downloading  $r$  object copies to recover one, thus, increasing the object access overhead multiplicatively by  $r$ . As a consequence of this,  $\mathcal{I}$  in this case increases additively in  $r$ , which can be seen from its limiting value range given in (27) and (29).

#### A. Note on Constructing $d$ -Choice Allocations With $r$ -XOR's

A  $d$ -choice storage allocation with  $r$ -XOR's consists of  $k$  exact and  $k(d-1)/r$  of  $r$ -XOR'ed object copies, and distributes them across the nodes in a way that complies with the balanced and regular allocation requirements given in Def. 1. This means each object has  $d-1$  XOR'ed choices, thus each object should be a part of  $d-1$  different XOR'ed copies. In addition, sets of objects that are XOR'ed together should not intersect pairwise at more than one object since this would violate the requirement that service choices must be disjoint for each object.

Clearly,  $d$ -choice allocation with  $r$ -XOR's does not exist for all values of  $k$ ,  $n$ , and  $d$ . First of all, as described previously in this Section,  $k(d-1)/r$  of  $r$ -XOR'ed object copies are required, which means we need to have  $r|k(d-1)$ . Second, the requirement that XOR'ed sets should intersect pairwise at most at one object is similar to a block design. Indeed the 3-choice allocation with 2-XOR's given in (26) is constructed based on a symmetric BIBD with  $\lambda = 1$  (see Def. 9 and the following paragraph). We do not address the construction of  $d$ -choice allocations with  $r$ -XOR's, but only study their load balancing performance by assuming their existence.

## VII. CONCLUSION

Storage systems need to have the ability to balance the offered load across the storage nodes in order to provide fast and predictable content access performance. Data objects are replicated across multiple nodes in distributed storage systems to implement robust load balancing in the presence of skews and changes in object popularities. In this paper, we developed a quantitative answer for two natural questions on implementing resource efficient distributed storage with robust load balancing ability: 1) How does the ability of load balancing improve per added level of storage redundancy for each data object? 2) Can storage efficient alternatives be used instead of replication to improve load balancing?

As an answer for the first question, we found that system's load balancing performance initially improves multiplicatively with the level of added storage redundancy  $d$ . Somewhat interestingly, once  $d$  reaches within a linear range of  $\log(\text{total # of storage nodes})$ , system's load balancing performance improves exponentially. As an answer for the second question, we found that implementing storage redundancy with XOR's of  $r$  objects rather than object replicas yield the same improvement in load balancing performance, while also reducing the storage overhead multiplicatively by  $r$ . However, accessing data storage by decoding from XOR'ed content requires jointly accessing  $r$  storage nodes (in contrast to a replica being available at a single node), which reduces the load balancing performance additively by  $r$ .

## APPENDIX

The following expression for  $M_{k,d=1}$  (maximal spacing within  $k$  uniform spacings in the unit line) is well known

$$M_{k,d=1} = \frac{\max \{E_1, \dots, E_k\}}{E_1 + \dots + E_k} \text{ in distribution,}$$

where  $E_i$  denote i.i.d. unit-mean Exponential random variables (RV's).

Joint distribution of the  $k$  uniform spacings on the unit line  $(S_1, \dots, S_k)$  is known to be the same as the joint distribution of  $(E_1/\Sigma, \dots, E_k/\Sigma)$  where  $\Sigma = E_1 + \dots + E_k$ . Using this representation, maximal non-overlapping  $m$ -spacing within  $k$  uniform spacings on the unit line can be expressed as follows.

*Lemma 13:* For  $k = mn$ , we have

$$M_{k,m}^{(\text{no})} = \frac{\max\{\Gamma_1, \dots, \Gamma_n\}}{\Gamma_1 + \dots + \Gamma_n} \text{ in distribution,}$$

where  $\Gamma_i$  are i.i.d. as Gamma with a shape parameter of  $m$  and a rate of 1, i.e.,  $\Gamma_i = \sum_{j=1}^m E_j$ .

#### A. Proof of Lemma 1

*Proof:* As discussed in Sec. IV-B, a regular balanced  $d$ -choice allocation with object replicas defines a balanced  $d$  regular bipartite mapping from the set of objects and the set of nodes, which we refer to it as its allocation graph. First, by Kőnig's theorem, every regular bipartite graph has a perfect matching, hence the allocation graph has a perfect matching.

Let  $S$  be a set of objects and  $N(S)$  denote its neighborhood, i.e., the set of all nodes that host at least one of the objects in  $S$ . Since the allocation graph has a perfect matching, by Hall's theorem, we have  $|N(S)| \geq |S|$  for every  $S$ . This shows that the storage allocation defines a  $(k, kd, n, n, 1)$  batch code. Given that the graph is  $d$  regular, it can only qualify for a  $(k, kd, d, n, 1)$  multiset batch code.  $\square$

#### B. Proof of Corollary 1

*Proof:* Recall that the system stores  $k$  objects. Let us denote its capacity region with  $\mathcal{C}$  and denote its intersection with  $\mathcal{S}_\Sigma$  as  $\mathcal{T}_\Sigma$ . Notice that  $\mathcal{S}_{\Sigma'}$  is obtained by scaling  $\mathcal{S}_\Sigma$  down with  $\Sigma'/\Sigma$ , hence we have

$$\text{Volume}(\mathcal{S}_{\Sigma'})/\text{Volume}(\mathcal{S}_\Sigma) = (\Sigma'/\Sigma)^k.$$

For any  $\mathbf{x} \in \mathcal{T}_\Sigma \subset \mathcal{C}$ , its scaled version  $\Sigma'/\Sigma \cdot \mathbf{x}$  also lies in  $\mathcal{C}$ . This comes from the convexity of  $\mathcal{C}$ . (Note that the origin  $\mathbf{0} \in \mathcal{C}$ .) Let us then scale down  $\mathcal{T}_\Sigma$  with  $\Sigma'/\Sigma$ , and denote it with  $\mathcal{T}'_\Sigma$ . Then  $\mathcal{T}'_\Sigma$  will also lie in  $\mathcal{C}$ . We also know that

$$\text{Volume}(\mathcal{T}'_\Sigma)/\text{Volume}(\mathcal{T}_\Sigma) = (\Sigma'/\Sigma)^k.$$

We have  $\mathcal{T}'_\Sigma \subseteq \mathcal{C} \cap \mathcal{S}_{\Sigma'}$ , then

$$\begin{aligned} \mathcal{P}_{\Sigma'} &= \frac{\text{Volume}(\mathcal{C} \cap \mathcal{S}_{\Sigma'})}{\text{Volume}(\mathcal{S}_{\Sigma'})} \geq \frac{\text{Volume}(\mathcal{T}'_\Sigma)}{\text{Volume}(\mathcal{S}_{\Sigma'})} \\ &= \frac{\text{Volume}(\mathcal{T}_\Sigma) \cdot (\Sigma'/\Sigma)^k}{\text{Volume}(\mathcal{S}_\Sigma) \cdot (\Sigma'/\Sigma)^k} = \mathcal{P}_\Sigma. \end{aligned}$$

$\square$

#### C. Proof of Lemma 4

*Proof:* Let the given storage allocation, that yields the capacity region  $\mathcal{C}$ , be described by the matrices  $\mathbf{T}$  and  $\mathbf{M}$  (in the sense of (2)). The described modification on the allocation says that a new service choice is added for one of the stored objects by either creating an additional service choice for the

object, via replicating it on a node that did not previously host the object or by adding a new choice simultaneously for multiple objects via encoding the objects together, and storing the coded copy on a node that did not previously host any of the encoded objects. We consider these two cases separately in the following.

#### When the new service choice is created with replication:

Let a tagged object be copied to a node that did not previously host the tagged object. The newly added choice can be captured by adding a new column to both allocation matrices  $\mathbf{T}$  and  $\mathbf{M}$ . Without loss of generality, suppose this new column is appended to both matrices at the end. Let us denote the modified versions of these matrices as  $\mathbf{T}'$  and  $\mathbf{M}'$  respectively.

First, we show that any point in  $\mathcal{C}$  also lies in  $\mathcal{C}'$ . Let us define  $D = \{\mathbf{x} \mid \mathbf{M} \cdot \mathbf{x} \leq \mathbf{1}, \mathbf{x} \geq \mathbf{0}\}$ , and  $D'$  similarly with  $\mathbf{M}'$ . Let  $\mathbf{p} \in \mathcal{C}$ , then there is an  $\mathbf{x}$  in  $D$  such that  $\mathbf{p} = \mathbf{T} \cdot \mathbf{x}$ . Let us generate  $\mathbf{x}'$  by appending a 0 at the end of  $\mathbf{x}$ . Then,  $\mathbf{x}' \in D'$  since  $\mathbf{M}' \cdot \mathbf{x}' = \mathbf{M} \cdot \mathbf{x} \leq \mathbf{1}$ , and  $\mathbf{T}' \cdot \mathbf{x}' = \mathbf{p}$ . Thus,  $\mathbf{p}$  also lies in  $\mathcal{C}'$ , which implies  $\mathcal{C} \subseteq \mathcal{C}'$ .

Next, we show that there is at least one point that lies in  $\mathcal{C}'$  but not in  $\mathcal{C}$ . Suppose that the tagged object is stored in  $d+1$  nodes after its number of choice is incremented (the modification). Then, the system can supply  $(d+1)\mathcal{C}$  units of demand for the tagged object and zero demand for all other objects, while it could not supply this before the modification was implemented on the storage allocation. This together with the fact that  $\mathcal{C} \subseteq \mathcal{C}'$  implies  $\mathcal{C} \subset \mathcal{C}'$ .  $\square$

#### When the new service choice is created with coding:

Let a new coded object copy be stored on a node that did not previously host any of the objects that constitute the coded copy. This adds a new choice for multiple objects simultaneously, which can be captured (as in the case above with replication) by adding new columns to allocation matrices  $\mathbf{T}$  and  $\mathbf{M}$ . The same arguments used above for the case with replication can be easily repeated here showing  $\mathcal{C} \subset \mathcal{C}'$  holds for this case as well.  $\square$

#### D. Proof of Lemma 5

*Proof: Proof of (7):* By Lemma 13, we have

$$M_{k,m}^{(\text{no})} = \frac{\max\{\Gamma_1, \dots, \Gamma_n\}}{\Gamma_1 + \dots + \Gamma_n} \text{ in distribution,} \quad (31)$$

where  $k = m \cdot n$  and  $\Gamma_i$ 's are i.i.d. as Gamma with a shape parameter of  $m$  and a rate of 1.

From Darling [29, Sec. 3], we know for fixed  $m$  as  $n \rightarrow \infty$

$$\begin{aligned} \Pr \left\{ 1/M_{k,m}^{(\text{no})} > m \frac{n}{\log(n)} - (m-1)m \frac{n \log_2(n)}{\log^2(n)} \right. \\ \left. + m \log(\Gamma(m)) \frac{n}{\log^2(n)} + m \frac{n}{\log^2(n)} x \right\} \rightarrow G(x). \end{aligned} \quad (32)$$

From this we get

$$\Pr \left\{ M_{k,m}^{(\text{no})} < \frac{\log(n)}{mn} \left( 1 - (m-1) \frac{\log_2(n)}{\log(n)} \right. \right. \\ \left. \left. + \frac{\log(\Gamma(m))}{\log(n)} - \frac{x}{\log(n)} \right)^{-1} \right\} \rightarrow G(x).$$

Defining

$$\alpha_n = \frac{x}{\log(n)} + (m-1) \frac{\log_2(n)}{\log(n)} - \frac{\log(\Gamma(m))}{\log(n)},$$

we can write

$$\Pr \left\{ M_{k,m}^{(\text{no})} < \frac{\log(n)}{mn} (1 - \alpha_n)^{-1} \right\} \rightarrow G(x).$$

Using the Taylor expansion on  $1/(1 - \alpha_n)$ , we can write

$$\Pr \left\{ M_{k,m}^{(\text{no})} < \frac{1}{mn} \left( x + \log(n) \right. \right. \\ \left. \left. + (m-1) \log_2(n) - \log(\Gamma(m)) \right) \right\} \rightarrow G(x),$$

which gives us (7).

**Proof of (9):**

For the maximal spacing  $M_{k,d=1}$  in  $k$  uniform spacings on the unit line, results in [30, Theorem 2.1] show that

$$\lim_{k \rightarrow \infty} M_{k,d=1} \cdot k / \log(k) = 1 \text{ a.s.} \quad (33)$$

The same theorem actually shows that the error in the above convergence is  $O(\log_2(k) / \log(k))$  a.s. as  $k \rightarrow \infty$ . The presented proof is established from the following

$$\Pr \left\{ |M_{k,d=1} \cdot k / \log(k) - 1| > \delta_k \right\} = O(k^{-\delta_k}) \quad (34)$$

for any sequence such that  $\delta_k \log(k) \rightarrow \infty$  and  $\delta_k \rightarrow 0$  as  $k \rightarrow \infty$ .

Recall that  $M_{k,m}^{(\text{no})}$  refers to the maximal non-overlapping  $m$ -spacing in  $k = n \cdot m$  uniform spacings on the unit line. By applying the argument that is used to prove [30, Theorem 2.1], we here show that a modified version of (33) holds also for  $M_{k,m}^{(\text{no})}$ . In the statement of the Lemma, this is expressed in (9).

The following bound, which is similar to that given in [32, Lemma 7], will allow us to obtain a result similar to (34). Let  $u_k$  be a fixed sequence to be defined later. Using the representation of  $M_{k,m}^{(\text{no})}$  that is given in (31)

$$\Pr \left\{ M_{k,m}^{(\text{no})} > u_k \right\} \\ = \Pr \left\{ M_{k,m}^{(\text{no})} > u_k; \sum_{i=1}^n \Gamma_i \leq k - k^{3/4} \right\} \\ + \Pr \left\{ M_{k,m}^{(\text{no})} > u_k; \sum_{i=1}^n \Gamma_i > k - k^{3/4} \right\} \\ \leq \Pr \left\{ \sum_{i=1}^n \Gamma_i \leq k - k^{3/4} \right\} \\ + \Pr \left\{ \max_{1 \leq i \leq n} \Gamma_i > u_k (k - k^{3/4}) \right\} \\ \stackrel{(a)}{\leq} e^{-\sqrt{k}/2} + n \cdot \gamma_m [u_k (k - k^{3/4})] \quad (35)$$

where i)  $\Gamma_i$ 's are i.i.d. as Gamma with a shape parameter of  $m$  and a rate of 1, ii) (a) follows by a large deviation argument on the left side of the expression and a union bound on the right side, iii)  $\gamma_m$  denotes the tail distribution of  $\Gamma_i$  as

$$\gamma_m(x) \sim \int_x^\infty \frac{u^{m-1}}{\Gamma(m)} e^{-u} du.$$

For some sequence  $\delta_k > 0$ , let us set

$$u_k = \frac{(1 + \delta_k) \log(k) + (m-1) \log_2(k)}{k}.$$

We know by [32, Lemma 5] that as  $x \rightarrow \infty$

$$\gamma_m(x) \sim \frac{x^{m-1}}{\Gamma(m)} e^{-x}.$$

Now define  $\epsilon_k = k^{-1/4}$  and further suppose that  $\delta_k = O(\log_2(k) / \log(k))$ . Then we get

$$\gamma_m[u_k(k - k^{3/4})] \\ \sim \frac{(1 - \epsilon_k)^{m-1}}{\Gamma(m)} \times \\ \frac{((1 + \delta_k) \log(k) + (m-1) \log_2(k))^{m-1}}{k^{(1+\delta_k)(1-\epsilon_k)} \log(k)^{(m-1)(1-\epsilon_k)}} \\ \leq O(k^{-(1+3\delta_k/4)}).$$

Substituting this estimate into (35) we obtain

$$\Pr \left\{ M_{k,m}^{(\text{no})} > u_k \right\} \leq O(n^{-\delta_k/2}). \quad (36)$$

Arguing as in the proof of [30, Theorem 2.1], let us define the subsequence  $k_t$ ,  $t = 1, 2, \dots$  to be  $k_t = \lfloor e^{\sqrt{2t}} \rfloor$  where rounding is to the largest multiple of  $m$ . Further choose the subsequence  $\delta_{k_t}$  to be

$$\delta_{k_t} = \frac{\log(t)}{\alpha \sqrt{t}} = O \left( \frac{\log_2(k_t)}{\log(k_t)} \right).$$

Notice that  $\delta_{k_t}$  as given above satisfies our previous assumptions:  $\delta_{k_t} \cdot \log(t) \rightarrow \infty$  and  $\delta_{k_t} \rightarrow 0$  as  $t \rightarrow \infty$ .

For  $0 < \alpha < 1/\sqrt{2}$  we have

$$\sum_t k_t^{-\delta_{k_t}/2} = \sum_t e^{-\log(t)/(\alpha \sqrt{2})} < \infty.$$

Then by the first Borel-Cantelli lemma, it follows that the inequality

$$M_{k_t,m}^{(\text{no})} > \frac{(1 + \delta_{k_t}) \log(k_t) + (m-1) \log_2(k_t)}{k_t}$$

occurs finitely often. Thus we have

$$M_{k_t,m}^{(\text{no})} \leq \log(k_t)/k_t + O(\log_2(k_t)/k_t) \text{ a.s.} \quad (37)$$

Given that  $(k_{t+1} - k_t) \sim k_t / \log(k_t)$ , again from Darling [29, Sec. 3] we have the following bound for  $0 \leq \ell \leq k_{t+1} - k_t$

$$\left| \frac{\log(k_t + \ell)}{k_t + \ell} - \frac{\log(k_t)}{k_t} \right| = O \left( \frac{\ell \cdot \log(k_t)}{k_t \cdot k_{t+1}} \right) \leq O(1/k_t) \quad (38)$$

Clearly  $M_{k_t,m}^{(\text{no})} \geq M_{k_t+\ell,m}^{(\text{no})}$ . Therefore it follows from (37) and (38) that

$$M_{k,m}^{(\text{no})} \leq \log(k)/k + O(\log_2(k)/k) \text{ a.s.}$$

This shows (8).

We can therefore conclude that

$$\limsup_{k \rightarrow \infty} M_{k,m}^{(\text{no})} \cdot k / \log(k) \leq 1. \quad (39)$$

Given that  $M_{k,m}^{(\text{no})} \geq M_{k,d=1}$ , from (33) it immediately follows

$$\liminf_{k \rightarrow \infty} M_{k,m}^{(\text{no})} \cdot k / \log(k) \geq 1.$$

This together with (39) implies that as  $k \rightarrow \infty$

$$M_{k,m}^{(\text{no})} \cdot k / \log(k) \rightarrow 1 \text{ a.s.}$$

This gives us (9).  $\square$

### E. Maximal d-Spacing on the Unit Circle

We here show that the maximal  $d$ -spacing  $M_{k,d}^{(c)}$  defined for  $k$  ordered uniform samples on the unit circle (see Def. 7) converge to its counterpart  $M_{k,d}$  defined on the unit line. In the following, we show convergence first in distribution, then in probability, and finally almost surely. Note that showing almost sure convergence implies convergence in probability, which then implies convergence in distribution. Convergence in this order is presented so as to make the arguments transparent.

*Lemma 14:* For  $d < k$ ,

$$\Pr\{M_{k,d} > x\} \leq \Pr\{M_{k,d}^{(c)} > x\} \leq \frac{k}{k-d} \Pr\{M_{k,d} > x\}.$$

*Proof:* Let us denote the events  $\{M_{k,d} > x\}$  and  $\{M_{k,d}^{(c)} > x\}$  respectively with  $L$  and  $C$ .

The first inequality is immediate; if a sequence of spacings  $s = (s_1, s_2, \dots, s_k) \in L$  then  $s \in C$ , while the opposite direction may not hold. Thus,  $L \subseteq C$ , hence  $\Pr\{L\} \leq \Pr\{C\}$ .

Next we show the second inequality. Let  $s \in L$ . Then, at least  $k-d$  different permutations of  $s$  lie in  $L$ . In order to see this, let the maximal  $d$ -spacing within  $s$  be  $\mathbf{m} = (s_i, \dots, s_{i+d-1})$ . Shifting (by feeding what is shifted out back in the sequence at the opposite end)  $s$  to the left by at most  $i-1$  times will preserve  $\mathbf{m}$ , hence each of the  $i-1$  shifted versions will also lie in  $L$ . Similarly, shifting  $s$  to the right by at most  $k-(i+d-1)$  times will also preserve  $\mathbf{m}$ . We call such permutations, which are obtained by shifting with wrapping around, a *cyclic permutation*.

Let us introduce a set  $L' \subset L$  such that for any  $s \in L'$ , no cyclic permutation of  $s$  lies in  $L'$ .  $L$  contains at least  $k-d$  cyclic permutations of every  $s \in L'$ . This together with the fact that all sequences of spacings are equally likely (Lemma 2) gives us  $(k-d) \Pr\{L'\} \leq \Pr\{L\}$ .

Now let  $s' \in C$ . All  $k-1$  cyclic permutations of  $s'$  will also lie in  $C$  (recall that we are now working on the unit circle). This together with the fact  $L' \subset L \subseteq C$  and Lemma 2 gives us  $k \cdot \Pr\{L'\} = \Pr\{C\}$ . Putting it all together, we have  $\Pr\{C\}/k = \Pr\{L'\} \leq \Pr\{L\}/(k-d)$ , which yields the second inequality.

A simpler way to find the second inequality in Lemma 14 is given as follows. Recall that the uniform samples, together with the 0 point, are ordered on the unit circle as  $0, U_{(1)}, \dots, U_{(k-1)}$ . Let us denote the index of the sample at which the maximal  $d$ -spacings starts with  $I$ , e.g.,  $I = i$  means

that the maximal  $d$ -spacing starts at the  $i$ th minimum uniform sample,  $I = 0$  means it starts at the point of 0. We have

$$\Pr\{M_{k,d} > x\} \geq \Pr\{M_{k,d}^{(c)} > x; I \leq k-d+1\}$$

since the event on the right implies the event on the left. The right hand side of this inequality can be written as

$$\Pr\{M_{k,d}^{(c)} > x\} \cdot (k-d)/k,$$

using the independence of the events and the fact that  $I$  is uniform on  $1, \dots, k$ .  $\square$

*Lemma 15:* For  $d = o(k)$ ,  $M_{k,d}^{(c)}/M_{k,d} \rightarrow 1$  in probability as  $k \rightarrow \infty$ .

*Proof:* It is easy to see  $M_{k,d}^{(c)} \geq M_{k,d}$ . Let  $D = M_{k,d}^{(c)} - M_{k,d}$  and  $S$  be the set of all sequence of spacings for which  $D > 0$ . For every  $s \in S$ ,  $d-1$  of its cyclic permutations (see the Proof of Lemma 14 for the definition of a cyclic permutation) also lie in  $S$  while the remaining  $k-d$  of them lie in  $S^c$  (complement of  $S$ ). Thus, for every  $d$  points in  $S$ , there are at least  $k-d$  points in  $S^c$ , and all the points in  $S$  or  $S^c$  (i.e., all spacings) have the same probability measure (by Lemma 2). This gives us the following upper bound  $\Pr\{D > 0\} = \Pr\{S\} \leq d/k$ , which  $\rightarrow 0$  as  $k \rightarrow \infty$ . This implies  $M_{k,d}^{(c)}/M_{k,d} \rightarrow 1$  in probability.  $\square$

In order to use the results known for the convergence of  $M_{k,d}$  in probability or a.s. in addressing  $M_{k,d}^{(c)}$ , we need the following Lemma.

*Lemma 16:* For  $d = o(k)$ ,  $M_{k,d}^{(c)}/M_{k,d} \rightarrow 1$  a.s. as  $k \rightarrow \infty$ .

Before we move on with the proof of Lemma 16, we next express the maximal  $d$ -spacing  $M_{k,d}^{(c)}$  on the unit circle in terms of the two different instances of its counterpart defined on the unit line.

Let  $2m+1 \geq 1$  be arbitrary and place  $2m+1$  i.i.d. uniform random variables on the unit circle with  $0 \leq U_{(1)} \leq \dots \leq U_{(2m+1)} \leq 1$  where the points 0 and 1 are identified. Let  $O$  denote the linear sequence starting at 0 and  $P$  be the linear sequence starting at  $U_{(m+1)}$  the median, without loss of generality. We know  $U_{(m+1)} = a \in (0, 1)$  almost surely. By adding further i.i.d. uniform variates we get two sequences of uniform spacings with parameter  $k \geq 2m+1$ , where the first starts at  $O$  and the second at  $P$ .

Let  $d_k = o(k)$  be a sequence of  $d_k$ -spacings for the  $k$ th realisation. Let  $M_{k,d_k}^{(c)}$  be the maximal *circular*  $d_k$ -spacing on the previously constructed unit circle (as defined in Def. 7), and let  $M_{k,d_k}^{(O)}$ ,  $M_{k,d_k}^{(P)}$  be the maximal  $d_k$ -spacing for the line segments that stretch along the sequences  $O$  and  $P$  respectively. We say that the circular spacings are *covered* by  $O$  and  $P$  if any circular  $d_k$ -spacing on the circle is either a  $d_k$ -spacing for  $O$  or for  $P$  (or both). This will always be the case if the number of intervening points  $N_k$  going from the beginning of  $O$  to the beginning of  $P$  clockwise is such that  $N_k \geq d_k$  and also for the number of points  $M_k$  going from the end of  $P$  to the end of  $O$  clockwise. Clearly if the circle spacings are covered by  $O$  and  $P$ ,

$$M_{k,d_k}^{(c)} \leq \max\{M_{k,d_k}^{(O)}, M_{k,d_k}^{(P)}\}. \quad (40)$$

We now show that a.s. for any sequence there is a  $K_d$  sufficiently large so that  $\forall k \geq K_d$  it holds that  $N_k, M_k \geq d_k$ .

It is enough to show this for  $N_k$  as the same argument will apply to  $M_k$ .

The interval from the beginning of  $O$  to the beginning of  $P$  has length  $a$  (recall  $U_{(m+1)} = a$ ) and therefore

$$N_k/k \rightarrow a \text{ a.s.}$$

by the strong law of large numbers. Therefore  $\exists K$  such that  $\forall k \geq K$ ,  $N_k \geq \frac{a \times k}{2}$  and  $a > 0$  a.s. Since  $d_k = o(k)$  it follows that  $\exists K_N \geq K$  such that  $N_k \geq d_k$ ,  $k \geq K_N$ . By the same argument  $\exists K_M$  such that  $M_k \geq d_k$ ,  $k \geq K_M$ . Now  $K_d = \max\{K_N, K_M\}$  is the required number and it follows that inequality (40) holds  $\forall k \geq K_d$  a.s.

Now we are ready to prove Lemma 16, that is to show  $M_{k,d}^{(c)}/M_{k,d} \rightarrow 1$  a.s.

*Proof:* First, given that  $M_{k,d}^{(c)} \geq M_{k,d}$ , we have

$$\liminf_{k \rightarrow \infty} M_{k,d}^{(c)}/M_{k,d} \geq 1. \quad (41)$$

Next using (40), we have

$$M_{k,d}^{(c)} \leq \max \left\{ M_{k,d_k}^{(O)}, M_{k,d_k}^{(P)} \right\}.$$

This allows us to find

$$\begin{aligned} \limsup_{k \rightarrow \infty} M_{k,d}^{(c)}/M_{k,d} \\ \leq \max \left\{ \limsup_{k \rightarrow \infty} M_{k,d}^{(O)}/M_{k,d}, \limsup_{k \rightarrow \infty} M_{k,d}^{(P)}/M_{k,d} \right\}. \end{aligned}$$

This, together with the fact that  $M_{k,d}^{(O)}$  and  $M_{k,d}^{(P)}$  converge to  $M_{k,d}$  a.s., gives us

$$\limsup_{k \rightarrow \infty} M_{k,d}^{(c)}/M_{k,d} \leq 1. \quad (42)$$

Putting (41) and (42) together completes the proof.  $\square$

In the following, we show that the results given in Lemma 7 and Lemma 8 for  $M_{k,d}$  carry over to  $M_{k,d}^{(c)}$ .

*Lemma 17:* For any integer  $d \geq 1$ , as  $k \rightarrow \infty$

$$\begin{aligned} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - (d-1) \log_2(k) \right. \\ \left. + \log((d-1)!) \leq x \right\} \rightarrow G(x). \quad (43) \end{aligned}$$

That is the distribution of  $M_{k,d}^{(c)} \cdot k - \log(k) - (d-1) \log_2(k) + \log((d-1)!)$  converges to standard Gumbel distribution as  $k \rightarrow \infty$ .

*Proof:* Let us first denote the event that the maximal  $d$ -spacing on the unit circle lies between the 1st and  $k$ th uniform sample with  $E$ , meaning that  $(M_{k,d}^{(c)} \mid E) = M_{k,d}$ . Since the maximal spacing is equally likely to start at any one of the uniform samples  $U_0, \dots, U_{k-1}$ , we have  $\Pr \{E\} = (k-d)/k$ . Let us also define

$$f_k = \log(k) + (d-1) \log_2(k) - \log((d-1)!).$$

By the law of total probability

$$\begin{aligned} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \\ = \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E \right\} \\ + \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E^c \right\}. \quad (44) \end{aligned}$$

Left hand side of the sum above can be bounded as

$$\begin{aligned} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E \right\} \\ \stackrel{(a)}{=} \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x; E \right\} \\ \stackrel{(b)}{\geq} \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\} - \Pr \left\{ E^c \right\}, \end{aligned}$$

where (a) follows from  $(M_{k,d}^{(c)}; E) = (M_{k,d}; E)$ , and (b) comes from the inequality for events  $A$  and  $B$

$$\Pr \{A; B\} = \Pr \{B\} - \Pr \{B; A^c\} \geq \Pr \{B\} - \Pr \{A^c\}.$$

Putting this in (44) gives us

$$\begin{aligned} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \\ \geq \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\} \\ - \Pr \left\{ E^c \right\} \\ + \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E^c \right\}, \end{aligned}$$

since  $\Pr \{E^c\} = d/k \rightarrow 0$  as  $k \rightarrow \infty$ . Overall this gives us

$$\begin{aligned} \liminf_{k \rightarrow \infty} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \\ \geq \liminf_{k \rightarrow \infty} \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\}. \end{aligned}$$

Given that  $M_{k,d}^{(c)} \geq M_{k,d}$ , we have the lower bound

$$\begin{aligned} \limsup_{k \rightarrow \infty} \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \\ \leq \limsup_{k \rightarrow \infty} \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\}. \end{aligned}$$

Both the lower and upper bounds given above are equal to  $G(x)$  by Lemma 7, hence showing (43).  $\square$

*Lemma 18:* For  $d = o(\log(k))$ , as  $k \rightarrow \infty$

$$\frac{M_{k,d}^{(c)} \cdot k - \log(k)}{(d-1)(1 + \log_2(k) - \log(d))} \rightarrow 1 \text{ a.s.}$$

*Proof:* For brevity, let us define the function

$$f(x) = \frac{x \cdot k - \log(k)}{(d-1)(1 + \log_2(k) - \log(d))}.$$

The fact that  $M_{k,d}^{(c)} \geq M_{k,d}$  gives us

$$f(M_{k,d}^{(c)}) \geq f(M_{k,d})$$

for  $k$  sufficiently large a.s. By (15) in Lemma 8, the right hand side of the above inequality  $\rightarrow 1$  as  $k \rightarrow \infty$  a.s. Then we have

$$\liminf_{k \rightarrow \infty} f(M_{k,d}^{(c)}) \geq 1. \quad (45)$$

Inequality (40) gives us

$$f(M_{k,d}^{(c)}) \leq f(\max \{M_{k,d_k}^{(O)}, M_{k,d_k}^{(P)}\})$$

for  $k$  sufficiently large a.s. This implies that

$$\begin{aligned} \limsup_{k \rightarrow \infty} f(M_{k,d}^{(c)}) \\ \leq \max \left\{ \limsup_{k \rightarrow \infty} f(M_{k,d}^{(O)}), \limsup_{k \rightarrow \infty} f(M_{k,d}^{(P)}) \right\}. \end{aligned}$$

By (15) in Lemma 8,

$$\limsup_{k \rightarrow \infty} f(M_{k,d}^{(O)}) = 1 \quad \text{and} \quad \limsup_{k \rightarrow \infty} f(M_{k,d}^{(P)}) = 1 \quad \text{a.s.}$$

Hence we have

$$\limsup_{k \rightarrow \infty} f(M_{k,d}^{(c)}) \leq 1.$$

This together with (45) concludes the proof.  $\square$

*Lemma 19:* For  $d = c \log(k) + o(\log_2(k))$  with some constant  $c > 0$ ,

$$V_k = \frac{M_{k,d}^{(c)} \cdot k - (1 + \alpha)c \cdot \log(k)}{\log_2(k)}$$

satisfies

$$\begin{aligned} \limsup_{k \rightarrow \infty} V_k &= c^*(1 + \alpha)/\alpha \quad \text{a.s.} \\ \liminf_{k \rightarrow \infty} V_k &= -c^\dagger(1 + \alpha)/\alpha \quad \text{a.s.} \end{aligned} \quad (46)$$

where  $\alpha$  is the unique positive solution of  $e^{-1/c} = (1 + \alpha)e^{-\alpha}$ , and  $c^*$  and  $c^\dagger$  are constants taking values in  $[-0.5, 1.5]$  and  $[-1.5, -0.5]$  respectively.

*Proof:* Shown applying the same ideas used in the proof of Lemma 18 given above.  $\square$

#### F. Proof of Lemma 9

*Proof:* Recall that nodes are indexed by the index of the primary object copies they store, i.e., node  $s_i$  stores the primary copy for object  $o_i$  for  $i = 1, \dots, k$ . We denote the set of choices for  $o_i$  with  $C_i$ .

**Proof of  $r \geq d - 1$ :** We first prove that  $r \geq d - 1$  by contradiction. Suppose  $r < d - 1$ . Pick an arbitrary object  $o_i$  with the set of choices  $C_i$ . Then  $o_{i-r}$  is co-located together with  $o_i$  on one of the nodes in  $C_i$ , which we refer to as  $s^*$ . Given that  $s^*$  is a choice for  $o_i$ , any object stored on it must be in the set  $\{o_{i-r}, o_{i-r+1}, \dots, o_{i+r}\}$ . By now  $s^*$  stores  $o_i$  and  $o_{i-r}$  by design, and we now look at the remaining  $d - 2$  storage slots of  $s^*$ . Given that  $s^* \in C_i$ ,  $r$ -gap allocation dictates that  $s^*$  can store only objects within  $\{o_{i-r}, \dots, o_{i+r}\}$ . We assumed that  $s^*$  stores  $o_{i-r}$  which means  $s^* \in C_{i-r}$  as well. So if  $s^*$  stores any object in  $\{o_{i+1}, \dots, o_{i+r}\}$  then  $C_{i-r} \cap C_j \neq \emptyset$  for some  $j > i$ , which would violate the definition of  $r$ -gap design. Therefore all the remaining  $d - 2$  storage slots of  $s^*$  must be occupied by the objects in the set  $\mathcal{O} = \{o_{i-r+1}, \dots, o_{i-1}\}$ , which means there needs to be at least  $d - 2$  different objects within  $\mathcal{O}$  (the same object cannot be stored multiple times on the same node), implying  $r \geq d - 1$ .

Perhaps, an easier way to show this is given as follows. The objects which may share a node with an object  $i$  are those within a set

$$S_i = \{r_{\min} \leq i \leq r_{\max}\}$$

where  $r_{\max} - r_{\min} \leq r$  and we apply arithmetic  $\bmod n$ . Such sets always contain at most  $r + 1$  elements. For example if  $r = 2$  and  $i = 3$ , we may take the set  $\{2, 3, 4\}$ , if  $n \geq 5$ , which has  $3 = r + 1$  elements. Or if  $n = 6$ ,  $i = 5$  and  $r = 4$

we may take the set  $\{5, 6, 1, 2, 3\}$  and note that  $3 - 5 + 6 = 4$  using arithmetic  $\bmod 6$ . The set contains 5 elements.

Now consider an object  $i$  and the corresponding node set  $C_i$ . It has  $d^2$  slots which have to be occupied. Since all sets containing  $i$  have at most  $r + 1$  elements under an  $r$ -gap design, it must be the case that  $(r + 1)d \geq d^2$  which implies  $r \geq d - 1$ .

**Proof of the lower bound for the node expansion of sets of objects:** We next show that  $x \leq N(S) \leq x + 2r$  for any  $S = \{o_i, \dots, o_{i+2r-1}\}$  for  $i = 1, \dots, n$ . Storage allocation defines a regular bipartite graph, then by Hall's theorem we have  $|N(S)| \geq x$ . The copies of  $o_i$  can expand across at most  $n_{i-r}, \dots, n_{i+r}$ , and the copies of  $o_{i+2r-1}$  can expand at most across  $n_{i+2r-1-r}, \dots, n_{i+2r-1+r}$ . Then  $S$  can expand at most across  $n_{i-r}, \dots, n_{i+2r-1+r}$ , meaning  $|N(S)| \leq x + 2r$ .  $\square$

#### G. Proof of Lemma 10

*Proof: Necessary condition:* System is surely unstable if a set of objects  $S$  has a cumulative offered load larger than  $|N(S)|$ . Lemma 9 states that every consecutive  $i$  objects expands across at most  $i + 2r$  nodes, meaning that the system can possibly be made stable only if the cumulative offered load for any  $i$  consecutive objects is less than  $i + 2r$ , which is exactly what is expressed in (19).

**Sufficient condition:** Suppose that the maximum offered load on any  $r$  consecutive objects is  $d$ , which can be described with the maximal  $r$ -spacing as  $M_{n,r}^{(c)} \cdot \Sigma \leq d$  (recall  $\Sigma$  is the cumulative offered load on the system).

Let  $x$  be an integer in  $[1, n]$ . Consider the following spiky load scenario starting at  $o_x$ ; offered load  $\rho_i$  for  $o_i$  is  $d$  when  $i = x + (r + 1)j$ ,  $j = 0, 1, \dots, \lfloor n/(r + 1) \rfloor$  and 0 otherwise. In this case, an offered load of magnitude  $d$  for each spiky object  $o_i$  can be supplied by using up the capacity in all the nodes available in its set of  $d$  choices  $C_i$  since all other objects that overlap with  $o_i$  in their service choices have 0 offered load (by the  $r$ -gap design property). The system can supply the spiky load regardless of the value for  $x$ . Given that the system's service capacity region is convex (Lemma 3), any convex combination of any set of spiky load scenarios can also be supplied by the system. This can be expressed as follows: system can operate under stability as long as the offered load on every  $r + 1$  consecutive objects is at most  $d$ , which implies (20).  $\square$

#### H. Proof of Lemma 11

*Proof:* Lower bounds come from substituting  $r = d - 1$  in those given in Lemma 10. Upper bounds come from observing that every  $d + 1$  consecutive objects expand to at most  $2d$  nodes in the design with clustering, and every  $d$  consecutive objects expand to  $2d - 1$  nodes in the design with cyclic construction.  $\square$

#### I. Proof of Theorem 2

*Proof:* Recall that the load at the maximally loaded node  $l_n^{\max}$  is given by  $M_{k,m}^{(n)} \cdot \Sigma_n$ . Almost sure convergence given in (9) implies for  $\Sigma_n = b_n \cdot n / \log(n)$  that

$$l_n^{\max} \cdot m / b_n \rightarrow 1 \quad \text{a.s.}$$

This implies in the limit  $n \rightarrow \infty$  for any  $\delta > 0$

$$\begin{aligned} \Pr\left\{|l_n^{\max} \cdot m/b_n - 1| > \delta\right\} \\ = \Pr\left\{l_n^{\max} > b_n/m \cdot (1 + \delta)\right\} \\ + \Pr\left\{l_n^{\max} < b_n/m \cdot (1 - \delta)\right\} \rightarrow 0. \end{aligned}$$

Given that both terms in the sum above is non-negative, we have

$$\begin{aligned} \Pr\left\{l_n^{\max} > b_n/m \cdot (1 + \delta)\right\} \rightarrow 0, \\ \Pr\left\{l_n^{\max} < b_n/m \cdot (1 - \delta)\right\} \rightarrow 0. \end{aligned}$$

Recall from (10) that  $\mathcal{P}_{\Sigma_n}$  is given by  $\Pr\left\{l_n^{\max} < 1\right\}$ . Then the convergence of probabilities given above implies (13).

(11) and (12) come from substituting  $\mathcal{I} = M_{k,m}^{(n)} \cdot n$  (by (10)) in the convergence results given in Lemma 5.  $\square$

#### J. Proof of Theorem 3

*Proof:* We first need to recall Lemma 11; under a cumulative demand of  $\Sigma$ ,  $M_{n,d}^{(c)} \cdot \Sigma \leq d$  is sufficient and  $M_{n,d}^{(c)} \cdot \Sigma \leq 2d$  is necessary for system stability. Here we will refer to  $d$  as  $d_n$  to make it explicit that it is a sequence in  $n$ .

**Proof of (22):** In this case  $d = o(\log(n))$ . Almost sure convergence given in (15) together with Lemma 16 implies for  $\Sigma_n = b_n \cdot n / \log(n)$  that

$$M_{n,d}^{(c)} \cdot \Sigma_n / b_n \rightarrow 1 \text{ a.s.}$$

Recall that  $M_{n,d}^{(c)} \cdot \Sigma_n / d_n \leq 1$  is sufficient and  $M_{n,d}^{(c)} \cdot \Sigma_n / 2d_n \leq 1$  is necessary for system stability, which respectively implies that  $\mathcal{P}_{\Sigma_n} \rightarrow 1$  if  $\limsup_{n \rightarrow \infty} b_n/d_n < 1$ , and  $\mathcal{P}_{\Sigma_n} \rightarrow 0$  if  $\liminf_{n \rightarrow \infty} b_n/2d_n > 1$ , hence (22).

**Proof of (24):** In this case  $d = o(\log(n))$ . Almost sure convergence given in (16) together with Lemma 16 implies for  $\Sigma_n = b_n \cdot n / \log(n)$  that in the limit  $n \rightarrow \infty$  we have almost surely

$$0.5\tau \leq M_{n,d}^{(c)} \cdot \Sigma_n / b_n \leq 1.5\tau.$$

Then the necessary and sufficient conditions (as used in the previous step while showing (22)) for system stability imply (24).

**Proof of (21) and (23):** In order to prove (21), let us now suppose that content access capacity at each node is  $C$ , in which case the sufficient and necessary conditions for stability are respectively written as  $M_{n,d}^{(c)} \cdot \Sigma \leq dC$  and  $M_{n,d}^{(c)} \cdot \Sigma \leq 2dC$ . Using these we find that  $C \geq M_{n,d}^{(c)} \cdot \Sigma / d$  is sufficient and  $C \geq M_{n,d}^{(c)} \cdot \Sigma / 2d$  is necessary for system stability. This means that the maximum load on any node in the system will lie in  $[M_{n,d}^{(c)} \cdot \Sigma / 2d, M_{n,d}^{(c)} \cdot \Sigma / d]$ , which implies that the load imbalance factor  $\mathcal{I}$  for the system lies in  $[M_{n,d}^{(c)} \cdot n / 2d, M_{n,d}^{(c)} \cdot n / d]$ .

Finally using the results of almost sure convergence given for  $M_{n,d}$  in Lemma 8 (hence given for  $M_{n,d}^{(c)}$  as well due to Lemma (16)), we find (21) and (23).  $\square$

#### K. Proof of Lemma 12

*Proof:* We use the following fact, which we refer to as **F** here: in a storage allocation with block design, every pair of objects overlaps at *exactly* one node in their choices.

**Necessary condition:** Expansion of a set  $S$  of  $d$  objects is maximized (of size  $d^2$ ) when the choices for each object are pairwise disjoint. This is not possible due to **F**. Let us start forming  $S$  by picking an arbitrary object  $o_i$  with the set of choices  $C_i$ . In order to maximize the expansion of  $S$ , let us form the rest of  $S$  by selecting one object from each node in  $C_i$ . Given **F**, no pair in  $S \setminus o_i$  is hosted on the same node. However, this does not prevent all objects within  $S \setminus o_i$  to be hosted on some other node (since a node hosts  $d$  different objects). In this case the expansion of  $S$  will consist of  $d + (d-1) + (d-2)^2 = d^2 - 2d + 3$  nodes, which gives us the necessary condition for stability.

**Sufficient condition:** We here consider the spiky load scenario discussed in the proof of Lemma 10; let  $x$  be an integer in  $[0, n]$ , and the offered load for  $o_i$  is  $\rho$  if  $i = x + dj$  for some  $j = 0, 1, \dots, \lfloor n/d \rfloor$  and 0 otherwise. Let us refer to objects with spiky load as “a spiky object”. Each spiky object shares its  $d$  choices with every other spiky object, and the worst case sharing is when the object has to share  $d-1$  of its choices with others. In the worst case, the system is stable only if  $\rho \leq 1 + (d-1)/2$ , which gives us the sufficient condition for stability.  $\square$

#### L. Proof of Theorem 4

*Proof:* This proof is very similar to that of Theorem 3, except for the complication that there is no cyclic equivalence of regular balanced  $d$ -choice allocation with XOR’s unlike the case in allocations with object replicas. That is why we first find auxiliary cyclic allocations that serve as lower or upper bound on the load balancing ability of the  $d$ -choice allocation with  $r$ -XOR’s (this is what makes the proof more difficult), then we derive our results by studying these auxiliary cyclic allocations.

We start by showing sufficient and necessary conditions for system stability.

**(i) Sufficient condition for system stability:** This part consists of three intermediate steps.

**Step 1: Cyclic allocation with  $r$ -XOR’s.** Consider a *cyclic*  $d$ -choice allocation in which for each object  $o_i$  that is primarily stored on node  $s_i$ ,  $d-1$  choices (recovery sets) are formed by the  $d-1$  consecutive disjoint  $r$ -sets of nodes that come right after  $s_i$  (in the order of node indices, by wrapping around the sequence of nodes if necessary). For instance, in 3-choice cyclic allocation over nodes  $[1, \dots, 6]$  with  $r=2$ , pairs of nodes  $(s_5, s_6)$  and  $(s_1, s_2)$  can jointly serve the object  $o_4$  that is primarily stored on  $s_4$  (recall that we assume the total number of stored objects  $k$  is equal to the total number of storage nodes  $n$ ).

Notice that a cyclic allocation cannot be implemented with XOR’s. This is because an additional  $r$ -XOR’ed copy adds a new choice simultaneously for  $r$  objects over a set of  $r+1$  nodes, and it is not possible for all these added choices to be

a proper cyclic choice. For instance, let objects  $a$ ,  $b$  and  $c$  be stored on nodes  $s_1$ ,  $s_2$  and  $s_3$  respectively, and let us store  $a + b$  on  $s_3$ . Then,  $s_2$  and  $s_3$  form a choice for  $a$ , which is a proper cyclic choice, while  $s_1$  and  $s_3$  form a choice also for  $b$  and this is improper for a cyclic allocation, which we simply refer to as a non-cyclic choice. However, it is still possible to create an allocation that implements both cyclic and non-cyclic choices with XOR's, then restrict it to behave as a cyclic allocation as follows. Firstly, each of the  $d - 1$  cyclic recovery choices can be created for each object via a separate XOR'ed copy. These XOR'ed copies will incur non-cyclic choices as discussed, but we will ignore and never use them for object access. For instance in the previous example, the incurred non-cyclic choice implemented by  $a$ ,  $a + b$  stored on  $(s_1, s_3)$  can be ignored and never used to access  $b$ , while a new proper cyclic choice can be added for  $b$  by storing  $b + c$  on  $s_4$ . In the following we use cyclic allocation, which is created with the restriction described here, merely as a tool to derive our results.

*Step 2: Cyclic achieves smaller capacity region than non-cyclic.* The capacity region of non-cyclic (our regular balanced) allocation with XOR's contains that of its cyclic counterpart. To see this is true, we give the following explanation. In non-cyclic  $d$ -choice allocation with  $r$ -XOR's, each node participates in *at most*  $k \cdot d/r$  different choices, while in its cyclic counterpart, each node participates in *exactly*  $k \cdot d/r$  different choices. In other words, non-cyclic allocation is using the capacity at the nodes more efficiently than its cyclic counter part, while implementing the same number of choices for each object. This expands the capacity region everywhere, or keeps it the same at worst. To better understand this, consider the following example of a 2-choice allocation with 2-XOR's and its corresponding allocation matrix

$$\begin{bmatrix} a \\ e+f \end{bmatrix}, [b], \begin{bmatrix} c \\ a+b \end{bmatrix}, [d], \begin{bmatrix} e \\ c+d \end{bmatrix}, [f]$$

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}. \quad (47)$$

We next briefly explain what  $M$  represents. The system achieves stability by splitting (balancing) the demand for each object across its  $d$  choices in such a way that no node is over burdened (i.e., each node is offered a load of  $< 1$ ). Each service choice for an object is either implemented by a single (primary) node or jointly by  $r$  nodes (an XOR'ed choice). The portion of an object's demand that is forwarded to and supplied by one of its XOR'ed choices flows simultaneously into the  $r$  nodes that jointly implement the choice. Each 1 within the  $i$ th row of  $M$  represents the assignment of an object's demand portion to node  $s_i$ . For instance,  $s_1$  implements the first (primary) choice for  $a$  (hence the first 1 in the 1st row), and participates in the second choice for objects  $e$  and  $f$  (hence the second and third 1 in the 1st row).

The cyclic counterpart of the allocation given above in (47) would be

$$M^c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \underline{1} & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \underline{1} & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} & 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad (48)$$

where the incurred non-cyclic choices are ignored in  $M^c$ , e.g.,  $a$  is never accessed from  $f+a$  and  $f$ . Each row sums to 3 in  $M^c$ , while half the rows sum to less than 3 in  $M$  (notice the additional  $\underline{1}$ 's in  $M^c$ ). In other words, non-cyclic allocation implements the same number of choices for each object with smaller overlap between different choices compared to its cyclic counterpart. A node's capacity is shared by all the service choices in which the node participates. Thus, it is better to have less overlap between choices in order to achieve greater capacity region. This is the “inefficiency” of cyclic allocation that causes it to achieve a smaller capacity region than its non-cyclic counterpart.

Let  $D = \{\mathbf{x} \mid M \cdot \mathbf{x} \leq \mathbf{1}, \mathbf{x} \geq \mathbf{0}\}$ , and  $D^c$  be defined similarly with  $M^c$ . It is easy to see that any  $\mathbf{x}$  in  $D^c$  will also lie in  $D$  (recall the additional  $\underline{1}$ 's in  $M^c$ ). In addition, non-cyclic  $d$ -choice allocation and its cyclic counterpart share the same  $\mathbf{T}$  (i.e., the other allocation matrix that yields the capacity region  $\mathcal{C}$  (or  $\mathcal{C}^c$ ) by transforming  $D$  (or  $D^c$ ); see Sec. II-C). Thus, we have  $\mathcal{C} \supseteq \mathcal{C}^c$ . This together with Def. 3 implies that probability  $\mathcal{P}_\Sigma$  for non-cyclic  $d$ -choice allocation is at least as large as that for its cyclic counterpart.

*Step 3: A sufficient condition for the stability of cyclic allocation.* Recall from Lemma 10 how we found a sufficient condition for stability when the allocation is constructed with the clustering or cyclic ( $r$ -gap in general) designs. Using the same arguments, a sufficient condition for the stability of the cyclic  $d$ -choice allocation with  $r$ -XOR's is found as  $M_{n,1+r(d-1)}^{(c)} \cdot \Sigma \leq d$ , where  $\Sigma$  is the cumulative offered load on the system and  $M_{n,1+r(d-1)}^{(c)}$  is the maximal  $(1+r(d-1))$ -spacing for  $n$  uniform spacings on the unit circle. The reason for caring about  $(1+r(d-1))$ -spacing's in this case (rather than  $d$ -spacing's as was the case for allocations with object replicas) is because an object's first choice is implemented by the (primary) node that stores the object, and its XOR'ed choices are implemented by the  $d - 1$  disjoint  $r$ -sets of nodes that follow the primary node in (cyclic) order. The reason for keeping the right hand side of the sufficient condition unchanged at  $d$  (as for the allocations with replication) is that object access from an  $r$ -XOR'ed choice requires accessing all  $r$  nodes that jointly implement the choice, so  $r(d - 1)$  nodes that form the  $d - 1$  XOR'ed choices for an object can at most provide a capacity of  $d - 1$ , which together with the capacity of the primary node adds up to  $d$ .

*Final Step: Putting it all together.* As discussed above,  $\mathcal{P}_\Sigma$  for cyclic  $d$ -choice allocations is a lower bound for that

of its non-cyclic counterpart. Thus, the sufficient condition  $M_{n,1+r(d-1)}^{(c)} \cdot \Sigma \leq d$  for the stability of cyclic allocations will also be sufficient for the stability of its non-cyclic counterpart (i.e., our regular balanced allocation).

**(ii) Necessary condition for system stability:** We again here relate the cyclic allocation with  $r$ -XOR's as introduced in part (i) to its non-cyclic counterpart (our regular balanced allocation). We do this again in three intermediate steps that are in the same spirit as those given in part (i).

**Step 1: Cyclic-plus allocation.** Recall in part (i) that we created a cyclic  $d$ -choice allocation by adding all  $k(d-1)$  XOR'ed copies that are necessary to implement the  $d-1$  cyclic choices for each object; and then ignoring the incurred non-cyclic choices by never considering them for object access (e.g., recall the non-cyclic allocation in (47) and its cyclic counterpart in (48)). Let us also consider and use the incurred non-cyclic choices for object access here, and refer to this form of allocation as *cyclic-plus*.

**Cyclic-plus achieves greater capacity region than non-cyclic.** The capacity region of cyclic-plus allocations will contain that of its non-cyclic counterpart, which together with Def. 3 implies that  $\mathcal{P}_\Sigma$  for cyclic-plus allocations will be at least as large as that of its non-cyclic counterpart. This is because cyclic-plus allocations implement all the choices that their non-cyclic counterparts implement plus some additional choices (e.g., compare (47) with (48)), which will yield at least as large a capacity region everywhere as the one without the additional choices.

**Step 2: A necessary condition for the stability of cyclic-plus allocation.** In cyclic-plus  $d$ -choice allocations, there are  $d$  cyclic and  $d$  non-cyclic choices for each object. Notice that each non-cyclic choice for an object is due to a cyclic choice of another object. Consider an object primarily stored on  $s_i$ , then  $s_{i+1} \bmod n$  participates in this object's first cyclic choice and all of its non-cyclic choices. This is a direct consequence of how cyclic choice with XOR's are constructed. For instance, consider object  $a$  in (48), its first cyclic choice is  $(b, a+b)$  and its only non-cyclic choice is  $(f+a, a)$ , where both  $b$  and  $f+a$  are stored on the node that comes right after  $a$ 's primary node. Thus, all of the  $d$  additional non-cyclic choices and the first cyclic choice for an object depend on a single node, which will be a bottleneck when these choices must be used simultaneously to access the object. In other words, all of these  $d+1$  choices (one cyclic and  $d$  non-cyclic) can simultaneously yield at most as much capacity as of a single node.

Due to the bottleneck node described above, even the additional non-cyclic choices are not sufficient to achieve stability in a cyclic-plus  $d$ -choice allocation with  $r$ -XOR's when any  $1+r(d-1)$  consecutive nodes have a cumulative offered load  $> 2d-1$ , that is when  $M_{n,1+r(d-1)}^{(c)} \cdot \Sigma > 2d-1$ , hence a necessary condition for stability is that as  $M_{n,1+r(d-1)}^{(c)} \cdot \Sigma \leq 2d$ . This is easy to see using the exact same arguments we used to show the corresponding necessary stability condition in Lemma 10 for  $d$ -choice allocation with object replicas.

**Final Step: Putting it all together.** We showed that the probability  $\bar{P}_\Sigma$  for cyclic-plus allocation is an upper bound on that of its non-cyclic counterpart, thus  $M_{n,1+r(d-1)}^{(c)} \cdot \Sigma \leq 2d$  is also a necessary stability condition for non-cyclic (our regular balanced)  $d$ -choice allocation.

From now on we will refer to  $1+r(d-1)$  as  $D$ .

**Proof of (28) and (30).** Follows from the exact same arguments used in the proof of respectively (22) and (24) (Theorem 3).

**Proof of (27) and (29).** Using the same arguments used in the proof of (21) and (23) (Theorem 3), we can conclude here that the load imbalance factor  $\mathcal{I}$  for the system lies in  $[M_{n,D}^{(c)} \cdot n/2d, M_{n,D}^{(c)} \cdot n/d]$ . The results of almost sure convergence given for  $M_{n,D}$  in Lemma 8 hold also for  $M_{n,D}^{(c)}$  by Lemma 16. Using the same arguments given in the proof of Lemma 8, we derive (27) and (29).  $\square$

## ACKNOWLEDGMENT

The authors would like to thank Gala Yadgar for the fruitful discussions and recommendations in the early stages of this work.

## REFERENCES

- [1] M. F. Aktaş, A. Behrouzi-Far, and E. Soljanin, "Load balancing performance in distributed storage with regular balanced redundancy," in *Proc. 16th Int. Symp. Problems Redundancy Inf. Control Syst. (REDUNDANCY)*, Oct. 2019, pp. 75–80.
- [2] J. Dean, "Challenges in building large-scale information retrieval systems: Invited talk," in *Proc. 2nd Int. Conf. Web Search Web Data Mining (WSDM)*, R. Baeza-Yates, P. Boldi, B. A. Ribeiro-Neto, and B. B. Cambazoglu, Eds., Barcelona, Spain, Feb. 2009, p. 1.
- [3] S. Melnik *et al.*, "Dremel: Interactive analysis of Web-scale datasets," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 330–339, Sep. 2010.
- [4] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
- [5] X. Ouyang, P. Garraghan, R. Yang, P. Townend, and J. Xu, "Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2016, pp. 1–3.
- [6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, M. G. Khatib, X. He, and M. Factor, Eds., Lake Tahoe, NV, USA, May 2010, pp. 1–10.
- [7] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.
- [8] T. Macedo and F. Oliveira, *Redis Cookbook: Practical Techniques for Fast Data Manipulation*. Newton, MA, USA: O'Reilly Media, 2011.
- [9] G. Ananthanarayanan *et al.*, "Scarlett: Coping with skewed content popularity in mapreduce clusters," in *Proc. 6th Eur. Conf. Comput. Syst. (EuroSys)*, C. M. Kirsch and G. Heiser, Eds., Salzburg, Austria, Apr. 2011, pp. 287–300.
- [10] M. Raab and A. Steger, "'Balls into bins'—A simple and tight analysis," in *Proc. 2nd Int. Workshop Randomization Approximation Techn. Comput. Sci. (RANDOM)*, in Lecture Notes in Computer Science, vol. 1518, M. Luby, J. D. P. Rolim, and M. J. Serna, Eds. Barcelona, Spain: Springer, Oct. 1998, pp. 159–170.
- [11] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM J. Comput.*, vol. 29, no. 1, pp. 180–200, 1999.
- [12] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced allocations: The heavily loaded case," in *Proc. 32nd Annu. ACM Symp. Theory Comput.*, F. Frances, Y. Eugene, and M. Luks, Eds., Portland, OR, USA, May 2000, pp. 745–754.
- [13] K. Kenthapadi and R. Panigrahy, "Balanced allocation on graphs," in *Proc. 17th Annu. ACM-SIAM Symp. Discrete Algorithm (SODA)*, Miami, FL, USA, 2006, pp. 434–443.

- [14] B. Godfrey, "Balls and bins with structure: Balanced allocations on hypergraphs," in *Proc. 19th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, S.-H. Teng, Ed., San Francisco, CA, USA, Jan. 2008, pp. 511–517.
- [15] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proc. 36th Annu. ACM Symp. Theory Comput.*, L. Babai, Ed., Chicago, IL, USA, Jun. 2004, pp. 262–271.
- [16] M. B. Paterson, D. R. Stinson, and R. Wei, "Combinatorial batch codes," *Adv. Math. Commun.*, vol. 3, no. 1, pp. 13–27, 2009.
- [17] A.-E. Riet, V. Skachek, and E. K. Thomas, "Batch codes for asynchronous recovery of data," Jun. 2018, *arXiv:1806.00592*. [Online]. Available: <http://arxiv.org/abs/1806.00592>
- [18] M. Aktas *et al.*, "On the service capacity region of accessing erasure coded content," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Oct. 2017, pp. 17–24.
- [19] 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)*, Guangzhou, China, Nov. 2018, pp. 1–5.
- [20] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads," *VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.
- [21] M. Sardari, R. Restrepo, F. Fekri, and E. Soljanin, "Memory allocation in distributed storage networks," in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX, USA, Jun. 2010, pp. 1958–1962.
- [22] R. Pyke, "Spacings," *J. Roy. Stat. Soc., B, Methodol.*, vol. 27, no. 3, pp. 395–436, 1965.
- [23] H. Dau and O. Milenkovic, "MaxMinSum Steiner systems for access balancing in distributed storage," *SIAM J. Discrete Math.*, vol. 32, no. 3, pp. 1644–1671, Jan. 2018.
- [24] D. R. Stinson, *Combinatorial Designs: Constructions and Analysis*. New York, NY, USA: Springer-Verlag, 2004.
- [25] N. Silberstein and A. Gál, "Optimal combinatorial batch codes based on block designs," *Des., Codes Cryptogr.*, vol. 78, no. 2, pp. 409–424, Feb. 2016.
- [26] D. Darling, "On a class of problems related to the random division of an interval," *Ann. Math. Statist.*, vol. 24, pp. 239–253, 1953.
- [27] Q. Huang, K. Birman, R. V. Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proc. ACM SIGOPS 24th Symp. Oper. Syst. Princ. (SOSP)*, M. Kaminsky and M. Dahlin, Eds., Farmington, PA, USA, Nov. 2013, pp. 167–181.
- [28] B. Büeler, A. Enge, and K. Fukuda, "Exact volume computation for polytopes: A practical study," in *Polytopes—Combinatorics and Computation*. Basel, Switzerland: Birkhäuser, 2000.
- [29] D. A. Darling, "On a test for homogeneity and extreme values," *Ann. Math. Statist.*, vol. 23, pp. 450–456, 1952.
- [30] E. Slud, "Entropy and maximal spacings for random partitions," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 41, no. 4, pp. 341–352, 1978.
- [31] L. Devroye, "Laws of the iterated logarithm for order statistics of uniform spacings," *Ann. Probab.*, vol. 9, no. 5, pp. 860–867, Oct. 1981.
- [32] P. Deheuvels and L. Devroye, "Strong laws for the maximal k-spacing when  $k \propto \log n$ ," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 66, no. 3, pp. 315–334, 1984.
- [33] L. Devroye, "Uniform and exponential spacings," in *Non-Uniform Random Variate Generation*. New York, NY, USA: Springer, 1986, pp. 206–245.
- [34] A. Mijatović and V. Vysotsky, "On the weak limit law of the maximal uniform k-spacing," *Adv. Appl. Probab.*, vol. 48, pp. 235–238, Jul. 2016.
- [35] A. Behrouzi-Far and E. Soljanin, "Redundancy scheduling in systems with bi-modal job service time distributions," in *Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Sep. 2019, pp. 9–16.

**Mehmet Fatih Aktas** received the B.S. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, and the M.S. and Ph.D. degrees in electrical and computer engineering from Rutgers University, New Brunswick, NJ, USA. He is currently a Senior Engineer at MathWorks. His main research interest is to make distributed computer systems faster and more robust to runtime changes. He has been doing research in both systems development and theoretical analysis. In his research, he mostly relies on probabilistic modeling and on tools from applied probability such as queueing theory, order statistics, and reinforcement learning.

**Amir Behrouzi-Far** received the B.S. degree in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2013, and the M.S. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Rutgers University, New Brunswick, NJ, USA. His research interests include the performance evaluation in distributed systems, timeliness in real-time systems, scheduling in computing/storage systems, wireless communications, and reinforcement learning.

**Emina Soljanin** (Fellow, IEEE) is currently a Professor at Rutgers University. Before moving to Rutgers in January 2016, she was a (Distinguished) Member of Technical Staff for 21 years in the Mathematical Sciences Research of Bell Labs. Her research interests and expertise are wide. 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, a 2016/17 Distinguished Lecturer, and 2019 President of the IEEE Information Theory Society. She has served as an Associate Editor for Coding Techniques for the IEEE TRANSACTIONS ON INFORMATION THEORY, and has also served as an Associate Editor for the Information Theory Society Board of Governors and in various roles on other journal editorial boards and conference program committees.

**Philip Whiting** received the B.A. degree from the University of Oxford, the M.Sc. degree from the University of London, and the Ph.D. degree in queueing theory from the University of Strathclyde.

After holding a post-doctoral position with the University of Cambridge, his research interests centered on wireless. In 1993, he participated in the Telstra trial of Qualcomm CDMA in South Eastern Australia. He then joined the Mobile Research Centre, University of South Australia, Adelaide. He was a Researcher at Bell Labs from January 1997 to June 2013. He is currently a Research Professor at Macquarie University, Sydney, Australia, and also a Consultant to Telstra for the past two years. He has over 25 patents in applications for DSL vectoring, wireless networks, and location and tracking. He has received several awards for his work in DSL vectoring and in wireless scheduling. He has held visiting positions including ones at Brown University (Maths), University of Korea (Engineering), and Vrij University (Maths). For the past three years, he has been a STAR Visiting Scholar with the Maths Department, Technical University of Eindhoven, where his collaborative work includes investigations of both CSMA networks and load balancing in queueing networks. Apart from papers in various aspects of telecommunications, he has been an author on various aspects of probability theory, including random Vandermonde matrices, large deviations theory for occupancy models and more recently random CSMA networks and load balancing in queueing networks. His current research includes storage systems and wireless mmwave networks.