Load Balancing Performance in Distributed Storage with Regular Balanced Redundancy

Mehmet Fatih Aktaş, Amir Behrouzi-Far and Emina Soljanin

Department of Electrical and Computer Engineering, Rutgers University
Email: {mehmet.aktas, amir.behrouzifar, emina.soljanin}@rutgers.edu

***Abstract*—Contention at the storage nodes is the main cause of long and variable data access times in distributed storage systems. Offered load on the system must be balanced across the storage nodes in order to minimize contention, and load balancing should be robust against the skews and fluctuations in content popularities. Data objects are replicated across multiple nodes in practice to allow for load balancing. However redundancy increases the storage requirement and should be used efficiently. We evaluate load balancing performance of natural storage schemes in which each data object is stored at d different nodes and each node stores the same number of objects. We find that load balance in a system of n nodes improves multiplicatively with d as long as $d = o(\log(n))$, and improves exponentially as soon as $d = \Theta(\log(n))$. We show that load balance improves the same way with d when the service choices are created with XORs of r objects rather than object replicas, which also reduces the storage overhead multiplicatively by r. However, unlike accessing an object replica, access through a recovery set composed by an XOR'ed copy requires downloading content from r nodes, which increases load imbalance additively by r.**

I. Introduction

Every distributed computing system is built on a storage layer that provides data write/read service for the executing workloads. Thus the overall system performance is strongly tied to the data access (I/O) performance of its underlying storage system [1]. Data access times in modern large scale systems greatly suffer from the performance variability at the storage nodes [2], which is caused by many factors, but primarily by multiple-workload resource sharing and the resulting resource contention [3]. Performance variability exists at any level of load, but it is aggravated by the most overloaded storage nodes [4]. It is therefore paramount that the system be able to well balance the offered data access load across the storage nodes.

Data replication is used in modern storage systems (e.g., HDFS [5], Cassandra [6], Redis [7]) to offer *multiple nodes (choices)* for splitting the offered load for each stored object. The best support for load balancing is provided when each object is stored at each node, but that is mostly not feasible at large scale, where only limited redundancy can be used. If the offered load for each object is known and fixed, each object could be stored with the adequate level of redundancy. However, in practice, object popularities, and in turn the load offered on them, are not only unknown but also fluctuate. Thus the ability of load balancing should be robust against the skews and changes in object popularities [1], [8]. On the other hand, the cumulative offered load for all objects stored in a system is known to vary at a much slower pace and is naturally easier to estimate (see, e.g., Fig. 7 in [9]).

We here evaluate certain load balancing metrics for several storage schemes in which each object is replicated d times and each node stores the same number of objects. We assume that all object offered loads are possible and equally likely as long as the cumulative offered load for all objects remains constant. Each node can serve at most a load of 1, and the offered load for each object can be split across its choices (nodes hosting the object) as long as the cumulative load at each node remains below 1. We address the following questions:

- **Q1** What fraction of the object offered load vectors satisfying the constant cumulative condition can the system support?
- **Q2** If an offered load vector can be served, what is the best achievable load balance in the system? How does it change with the design parameters: number of objects stored per node, number of choices d provided for each object and the layout of the stored objects across the nodes?
- **Q3** Can we store XORs of multiple objects instead of object replicas (cf. code) to improve the load balance in the system with less storage overhead?

The storage schemes we consider are common. Optimizing storage schemes for various purposes are studied elsewhere (e.g. for improving data access in [10]).

Prior and Related Work: Load balancing in storage systems has been studied in two main settings which we refer to as the *dynamic* and *static*. In dynamic settings, requests for data objects arrive sequentially. Each request gets dynamically assigned to one of nodes storing the requested object according to some strategy based on the nodes' load information, with the objective to minimizes the maximum load on any node (see e.g., [11], [12]). In static settings, the goal is to either 1) design storage-efficient schemes (e.g., batch codes) that allows good load balance across the nodes [13] for some set of expected offered loads, or 2) understand which object offered loads a given storage scheme (based on, e.g., some known erasure code) can support [14].

We are here concerned with a static setting, and the questions we ask are related to those asked in the "service rates of codes" problem [14], [15]. Unlike the previous work, 1) we consider schemes that store more than a single object per node, 2) we find which fraction of a targeted offered load region can the considered schemes achieve without explicitly finding

978-1-7281-1944-1/19/$31.00 ©2019 IEEE

the complete achievable region. We were able to do that by establishing a connection of our problem with uniform spacings (described in Sec. II-B), 3) we address the question of load imbalance which is (as mentioned above) important in practice.

This paper is organized as follows: Sec. II presents the storage model, offered load model and its connection with uniform spacings, and finally introduces the metrics we use to evaluate a system's load balancing performance. In Sec. III and IV we discuss storage schemes in which each object is stored in single or multiple nodes, and evaluate their load balancing performance. In Sec. V we discuss creating multiple service choices for objects as recovery sets using XORs rather than object replicas, and evaluate the impact of this change on the storage overhead and load balancing performance.

Note on the proofs, plots, and notation: We omit the proofs here because of the space constraint. For the proofs, we refer the reader to the long version of this paper [16]. Each point in Fig. 1 and 2 are computed by taking the average of 10^5 simulation runs. Within each simulation run, offered loads for the objects are sampled as described in Sec. II-B and the load imbalance factor $\mathcal{I}$ is computed as described in Sec. II-C. Throughout, $\log$ refers to the natural logarithm, and $\log_i$ refers to i times iterated logarithm, e.g., $\log_2(x)$ stands for $\log\log(x)$.

II. System Model and Performance Metrics

A. Storage and Access Model

We consider a system of n storage nodes $s_1, \ldots, s_n$ (redundantly) hosting k data objects $o_1, \ldots, o_k$ where $n|k$. 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.

Offered load for object o_i is denoted by ρ_i and refers to the number of bytes streamed from the system per unit time to access o_i, divided by the capacity of a single node. We use *choice* for an object to refer to a node that hosts the object. Multiple choices for an object can be created either by replicating it over several nodes (see Sec. IV), or by XOR'ing it together with other objects and storing the result on a node that did not previously host any of the XOR'ed objects (Sec. V). When XOR'ing is used, a choice for an object refers to a *recovery set*, i.e., a set of nodes that can jointly recover the object. Accessing an object through one choice shall not interfere with accessing the same object through another choice, thus, different choices for the same object are disjoint.

Offered load for an object can be arbitrarily split across its choices. When a load of ρ is exerted by an object on a recovery set, each node within the set that composes the choice will be exerted a load of ρ. *Load on a node* is given by 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 its load is less than 1. A system is said to be stable if each node within the system is stable. We assume that offered load ρ_i's on the objects are split across their choices so that the load on the maximally loaded node is minimized.

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 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 different ways to design a d-choice allocation. We detail some of them in Sec. IV and V. For the rest of the paper, unless otherwise noted, allocation itself will mean a regular balanced allocation. The reader familiar with batch codes should be able to see that any regular balanced d-choice allocation represents a $(k, kd, n, n, 1)$ batch code and a $(k, kd, d, n, 1)$ multiset batch code [13].

B. Offered Load and Uniform Spacing Model

We assume that the system is expected to operate under any object offered load vector $(\rho_1, \ldots, \rho_k)$ in the set

$$\mathcal{S}_\Sigma = \Big\{(\rho_1, \ldots, \rho_k) \;\Big|\; \sum_{i=1}^{k} \rho_i = \Sigma,\ \rho_i \geq 0\Big\}. \tag{1}$$

That is, cumulative offered load remains constant at Σ but the object popularities can change arbitrarily.

We further assume that $(\rho_1, \ldots, \rho_k)$ is sampled uniformly at random from $\mathcal{S}_\Sigma$, and establish a connection with *uniform spacings*; a mathematical object connected with the uniform sampling of points from a simplex geometry (as explained below). The results available in the literature on the properties of uniform spacings (see e.g. [17]) and those derived here by us have been instrumental in obtaining our main results.

Let $U_{(1)}, \ldots, U_{(k-1)}$ be $k-1$ i.i.d. uniform samples in $[0, 1]$, given in non-decreasing order. Then $S_i = U_{(i)} - U_{(i-1)}$'s for $i = 1, \ldots, k$, where $U_{(0)} = 0$ and $U_{(k)} = 1$, are known as k *uniform spacings* on the unit line.

Lemma 1 (see e.g. [17])**.** *Uniform spacings $(S_1, \ldots, S_k)$ is uniformly distributed over the simplex*

$$\Big\{(x_1, \ldots, x_k) \;\Big|\; \sum_{i=1}^{k} x_i = 1,\ x_i \geq 0 \text{ for } i = 1, \ldots, k\Big\}.$$

Lemma 1 implies that object offered loads ρ_i's for a cumulative load of Σ can be seen as the k uniform spacings in $[0, \Sigma]$. This connection allows us to use the results on uniform spacings in order to evaluate the metrics we define next to evaluate load balancing performance in storage systems.

C. Performance Metrics

We use two metrics to answer the questions posed in Sec. I.

Definition 2. $\mathcal{P}_\Sigma$ for a system denotes the fraction of points in $\mathcal{S}_\Sigma$ (1) at which the system can operate under stability. In other words, since offered load vector $(\rho_1, \ldots, \rho_k)$ is sampled uniformly at random from $\mathcal{S}_\Sigma$, $\mathcal{P}_\Sigma$ denotes the probability that the system defined by $\mathcal{S}_\Sigma$ will operate under stability.

Definition 3 ($\mathcal{I}$)**.** In a system of n storage nodes operating under a cumulative load of Σ, load imbalance factor $\mathcal{I}$ for the system is defined as the load on the maximally loaded node divided by its minimum possible value, i.e., Σ/n.

$\mathcal{P}_\Sigma$ is obviously 0 when $\Sigma > n$, hence we assume $\Sigma \leq n$ implicitly throughout. Notice also that I is always ≥ 1.

III. Load Balancing with no Redundancy

Here, each of the k objects is stored on a single node and each of the n nodes stores $m = k/n$ different objects. Offered load for an object in this case has to be completely served by the only node hosting the object, and each node has to serve the total load for all objects stored on it.

As discussed in Sec. II-B, object offered load vector $(\rho_1, \ldots, \rho_k)$ can be described by k uniform spacings in $[0, \Sigma]$. Given that uniform spacings are exchangeable RV's, we can say that node s_i stores $o_{(i-1)m+1}, \ldots, o_{im}$. Then the load l_i exerted on s_i is given by $l_i = \sum_{(i-1)m+1}^{im} \rho_j$. For the system to be stable, all l_i's must be ≤ 1, and thus $\max\{l_1, \ldots, l_n\} \leq 1$ is necessary and sufficient for stability. Therefore, $\mathcal{P}_\Sigma$ is given by $\Pr\{\max\{l_1, \ldots, l_n\} \leq 1\}$. In the uniform spacing literature, l_i's are called non-overlapping m-spacings, and their maximum is called *maximal non-overlapping m-spacing*.

Definition 4 ($M_{k,m}^{(n)}$)**.** Maximal non-overlapping m-spacing for k uniform spacings on the unit line is defined for $k = m \cdot n$ as

$$M_{k,m}^{(n)} = \max_{i=1,\ldots,n} U_{(im)} - U_{((i-1)m)} \quad \text{or} \quad \max_{i=1,\ldots,n} \sum_{(i-1)m+1}^{im} S_j$$

Notice that $M_{k,m}^{(n)}$ denotes the load on the maximally loaded node in the system when the cumulative offered load is 1. Using a combination of the ideas presented in [18]–[20], we can derive the following convergence results for $M_{k,m}^{(n)}$.

Lemma 2. *For fixed m, as $n \to \infty$*

$$\Pr\left\{ M_{k,m}^{(n)} \cdot mn - \log(n) - f_n < x \right\} \to G(x).$$

where $G(x) = \exp(-\exp(-x))$ is the Gumbel function, and

$$\left(M_{k,m}^{(n)} \cdot mn - f_n \right) / \log(n) \to 1 \text{ a.s.}$$

where $f_n = (m-1)\log_2(n) - \log((m-1)!)$.

Now we are ready to express $\mathcal{P}_\Sigma$ and $\mathcal{I}$ in terms of $M_{k,m}^{(n)}$.

Lemma 3. *In a system with single-choice storage allocation,*

$$\mathcal{P}_\Sigma = \Pr\left\{ M_{k,m}^{(n)} < 1/\Sigma \right\}, \qquad \mathcal{I} = n \cdot M_{k,m}^{(n)}.$$

Using Lemma 3 and Lemma 2, we determine the behavior of $\mathcal{P}_\Sigma$ and $\mathcal{I}$ for large n as follows:

Theorem 1. *Consider a system with single-choice storage allocation. For fixed m, we have as $n \to \infty$*

$$\Pr\{I \cdot m - \log(n) - f_n < x\} \to G(x),$$

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

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 we have in the limit $n \to \infty$

$$\mathcal{P}_{\Sigma_n} = \begin{cases} 1 & \limsup b_n < m, \\ 0 & \liminf b_n > m. \end{cases} \tag{2}$$

Remark 1. Theorem 1 implies $\mathcal{I} = \Theta(\log(n))$ for a system with single-choice storage allocation and fixed m. It also shows that the limiting value of $\mathcal{I}$ decays multiplicatively with m. The scaling of $\mathcal{I}$ with $\log(n)$ is aligned with the well-known result for dynamic load balancing setting with the balls-into-bins model: 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 with high probability [11].

IV. Load Balancing with d-fold Redundancy

In d-choice allocation, each of the k objects is stored on d different nodes (choices) and each of the n nodes stores $k \cdot d/n$ different objects.

As discussed at the end of Sec. III, load imbalance $\mathcal{I}$ in the system decays with the number of objects stored per node. We here (and in Sec. V) consider the worst case for load balancing, which is $k = n$. This makes it easier to formulate and study the problem, and also to explain and interpret the derived results. Results that we present here can be extended using similar arguments for the general case with a fixed $k/n > 1$.

In our study of d-choice allocation, we can find sufficient and necessary conditions on system stability using *maximal d-spacing*, which we discuss in the following subsection.

A. Uniform spacings interlude

Definition 5 ($M_{k,d}$)**.** Maximal d-spacing within k uniform spacings on the unit line is defined as

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

where d is any integer in $[1, k]$.

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

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

While deriving our main results, we extensively use the elegant results presented on $M_{k,d}$ in [19], [21] and in particular [22, Theorem 1] and [23, Theorem 2, 6]. We show that [22, Theorem 1] and [23, Theorem 2, 6] hold also for $M_{k,d}^{(c)}$ when $d = o(k)$. We only state the following result which gives an intuition for why the asymptotic results that were previously known for $M_{k,d}$ carry over to $M_{k,d}^{(c)}$ when $d = o(k)$.

Lemma 4. *For $d = o(k)$, $M_{k,d}^{(c)}/M_{k,d} \to 1$ a.s. as $k \to \infty$.*

B. Evaluating $\mathcal{P}_\Sigma$ and $\mathcal{I}$ for systems with d-choice allocations

A d-choice allocation defines a d-regular balanced bipartite mapping from the set of objects to the set of nodes, which we refer to as *allocation graph*. Its construction can be described

as follows: i) Map the primary object copies to nodes with a bijection f_0, ii) For i going from 1 to $d-1$, map the ith redundant copies for each object to nodes with a bijection f_i such that $f_i(o) \neq f_j(o)$ for any $j < i$ and o. Thus, every node stores one primary and $d-1$ different redundant copies. We refer to a node with the id of the primary object copy stored on it, i.e., o_i is hosted primarily on s_i. Subscript in s_i or o_i will implicitly denote $i \mod n$ throughout. We denote the set of nodes that host o_i with C_i.

Load balancing ability in storage is not only determined by the number of object choices but also on the layout of the content across the nodes. Since $k = n$ and $d > 1$, $|C_i \cap C_j| > 0$ for some $j \neq i$. Overlaps between C_i's might lead to contention when the content popularity is skewed towards the objects that has overlapping choices. Both the number of overlapping C_i's and the size of the overlaps shall be minimized to improve the ability of load balancing. However, the size and number of overlaps cannot be reduced together given a fixed number of nodes. Denoting the cardinality of a set with $|\cdot|$, we have

$$\sum_{i=1}^{k} \sum_{j \neq i} |C_i \cap C_j| = (d-1)d \cdot k, \tag{3}$$

which comes from observing that each node serves as a choice for d different objects and counted in exactly the $d-1$ of the intersections. Given that cumulative cardinality of the overlaps are fixed, i.e., (3), decoupling some of the C_i's can only come at the cost of enlarging the overlap between some other C_j's.

We first consider the two following simple designs for constructing d-choice allocation.

Definition 7 (Clustering design)**.** When $d|n$, the simplest construction is to form *clusters* of d nodes such that each node within the same cluster hosts the same set of d objects. In other words, f_i's are chosen such that the allocation graph is composed by n/d separate d-regular complete bi-partite graphs.

Definition 8 (Cyclic design)**.** The next simplest design follows a *cyclic* construction by picking f_i's such that $f_{i+1}(o) = f_i(o) + 1 \mod n$ for $i = 0, \ldots, d-1$ and every o.

For instance, 3-choice allocation for 7 objects $a, \ldots, g$ with cyclic construction would look like

$$\begin{bmatrix} a \\ g \\ f \end{bmatrix} \quad \begin{bmatrix} b \\ a \\ g \end{bmatrix} \quad \begin{bmatrix} c \\ b \\ a \end{bmatrix} \quad \begin{bmatrix} d \\ c \\ b \end{bmatrix} \quad \begin{bmatrix} e \\ d \\ c \end{bmatrix} \quad \begin{bmatrix} f \\ e \\ d \end{bmatrix} \quad \begin{bmatrix} g \\ f \\ e \end{bmatrix}. \tag{4}$$

For a given set of objects S, union of their choices C_i's 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 offered load for S is greater than $N(S)$. Thus it is desirable to increase the size of 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 overlaps between C_i's for the objects in S, which would imply overlapping C_i's with the C_j's of objects outside of S.

It is not easy to define a knob that regulates both the overlaps between C_i's 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 9 (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 5. *In a d-choice allocation with r-gap design, for set of objects $S = \{o_i, o_{i+1}, \ldots, o_{i+x-1}\}$ with $i = 1, \ldots, n$, we have $r \geq d-1$ and $x \leq |N(S)| \leq x + 2r$.*

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

Lemma 6. *Consider a system with d-choice storage allocation that is constructed with an r-gap design and operating under a cumulative offered load of Σ. Then for system stability, a necessary condition is given as*

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

and a sufficient condition is given as

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

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

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

Notice that clustering or cyclic design is an r-gap design, hence the bounds in Lemma 6 are valid for storage allocation with either design. Their well-defined structure also allows refining the bounds on $\mathcal{P}_\Sigma$ as follows.

Lemma 7. *In a d-choice allocation constructed with clustering or cyclic design, we have*

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

Using the bounds given in Lemma 7, we can find an asymptotic characterization for $\mathcal{P}_\Sigma$ and $\mathcal{I}$ as follows.

Theorem 2. *Consider a system with d-choice storage allocation constructed with clustering or round-robin design.*

When $d = o(\log(n))$, in the limit $n \to \infty$ a.s.

$$\frac{1}{2} \leq \frac{\mathcal{I} \cdot d}{\log(n) + (d-1)(1 + \log_2(n) - \log(d))} \leq 1, \tag{5}$$

and if $\Sigma_n = b_n \cdot n/\log(n)$ for some sequence $b_n > 0$, then

$$\mathcal{P}_{\Sigma_n} = \begin{cases} 1 & \limsup b_n/d < 1, \\ 0 & \liminf b_n/2d > 1. \end{cases} \tag{6}$$

When $d = c\log(n)$ for some $c > 0$, in the limit $n \to \infty$ a.s.

$$\frac{1}{6} \leq \frac{2c\alpha}{3(\alpha+1)} \cdot \frac{\mathcal{I} \cdot \log(n)}{\log_2(n)} \leq 1, \tag{7}$$

where α 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

$$\mathcal{P}_{\Sigma_n} = \begin{cases} 1 & \limsup b_n \cdot 1.5\tau/d < 1, \\ 0 & \liminf b_n \cdot 0.25\tau/d > 1, \end{cases} \tag{8}$$

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

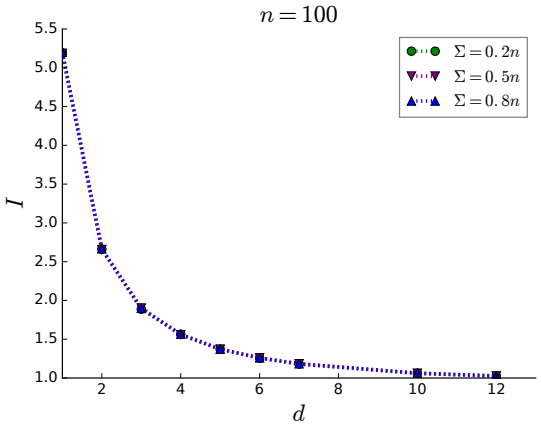

Fig. 1: Load imbalance factor $\mathcal{I}$ for d-choice allocation with cyclic construction for varying cumulative offered load Σ and d. Notice that the three curves generated for different values of Σ lie on top of each other since $\mathcal{I}$ is defined (Def. 3) to be irrespective of Σ.

Remark 2. Theorem 2 implies that i) d choices for each object initially reduces the load imbalance multiplicatively by d, ii) exponential reduction in the load imbalance kicks in as soon as d reaches $\Theta(\log(n))$, i.e., $\mathcal{I}$ goes from $\Theta(\log(n))$ to $\Theta(\log\log(n)/\log(n))$ as d goes from 1 to $\Theta(\log(n))$. These results show that the two observations of [12], which were shown with the dynamic balls-into-bins model under light offered load (i.e., when $O(n)$ balls are sequentially placed into n bins) extend to the static setting under general offered load.

Fig. 1 plots the load imbalance factor $\mathcal{I}$ for the system with $n = 100$ and varying d. Notice that the value of $\mathcal{I}$ is close to $\log(n)$ when $d = 1$ as suggested by Theorem 1, and decays with $1/d$ as suggested by Theorem 2. This illustrates that our asymptotic results are close estimates for the finite case.

Construction with r-gap design decouples C_i's that are r-apart at the cost of enlarging the overlaps between those that are close to each other, e.g., see this in the clustering or cyclic design. Balanced Incomplete Block Designs (BIBD) allows controlling the overlaps between every pair of C_i's.

Definition 10 (BIBD, [24])**.** A (d, λ) block design is a class of equal-size subsets of $\mathcal{X}$ (set of objects), called blocks (nodes), such that every point in $\mathcal{X}$ appears in exactly d blocks (choices), and *every* pair of distinct points is contained in exactly λ blocks.

Since we assume $k = n$, 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 C_i's, we focus on this case and by block design in the remainder we refer to $(d, 1)$ symmetric BIBD. Since every pair of C_i's overlaps at one node, we have

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

Then by (3), a block design is possible only if $k = d^2 - d + 1$.

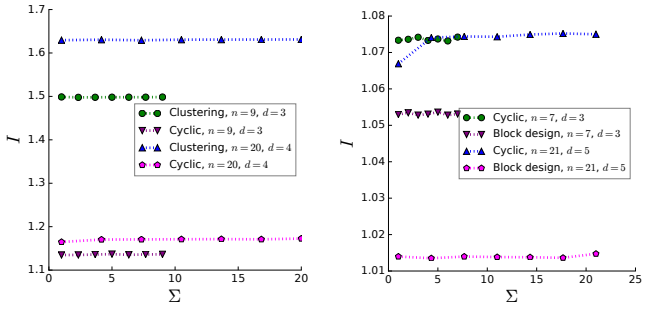

Fig. 2: $\mathcal{I}$ for d-choice allocation with different designs. Note that clustering and block design do not co-exist for the same d and n.

For instance a 3-choice allocation with block design looks like

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \begin{bmatrix} a \\ f \\ g \end{bmatrix} \quad \begin{bmatrix} a \\ d \\ e \end{bmatrix} \quad \begin{bmatrix} b \\ d \\ f \end{bmatrix} \quad \begin{bmatrix} b \\ e \\ g \end{bmatrix} \quad \begin{bmatrix} c \\ d \\ g \end{bmatrix} \quad \begin{bmatrix} c \\ e \\ f \end{bmatrix} \tag{9}$$

Sufficient and necessary conditions presented in Lemma 6 cannot be used on a storage allocation with block design, since it does not comply with the r-gap design. However using ideas that are similar to those used to derive Lemma 6, we can find the following conditions on the system stability.

Lemma 8. *Consider a system with d-choice allocation constructed with block design and operating under a cumulative offered load of Σ. For the 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 $M_{n,d}^{(c)} \leq d/2\Sigma$.*

Stability conditions given in Lemma 8 allow us to find bounds on $\mathcal{P}_\Sigma$ and $\mathcal{I}$ for storage systems with block design, similar to those that were stated in Theorem 2. We do not state them here since they are obtained by simply modifying the multiplicative factors of the bounds given in Theorem 2. The upper bound on $\mathcal{I}$ decays with $1/d$, which says that d choices initially reduce load imbalance at least multiplicatively by d. On the other hand, the lower bound on $\mathcal{I}$ decays in this case with $1/d^2$, that is, block design can possibly implement better scaling of $\mathcal{I}$ in d compared to clustering or cyclic design. However, in the simulations we observe that this is not the case and $\mathcal{I}$ decays as $1/d$ for block design as well.

Our asymptotic analysis does not allow ordering different allocation designs in terms of their load balancing performance. As discussed previously, all d-choice allocations yield the same cumulative overlap between object choices C_i's (recall (3)) and each design gives a different way of distributing the overlaps across C_i's. With simulations we find that it is better to evenly spread the overlaps between C_i's using block design, that is, *many but consistently small overlaps is better than few but occasionally large overlaps*. For instance, Fig. 2 shows $\mathcal{I}$ in 3- and 5-choice allocations constructed with clustering, cyclic or block design. We here see that the largest gain in $\mathcal{I}$ is achieved by moving from clustering to cyclic, while moving to block design yields a smaller gain in $\mathcal{I}$. Currently we don't have a rigorous way to understand how designs with different overlaps between C_i's compare with each other in terms of $\mathcal{P}_\Sigma$ or $\mathcal{I}$.

V. d-FOLD REDUNDANCY WITH XORS

In this section we will answer **Q3**. So far we have only considered d-choice allocations with object replicas. A replicated copy adds a new choice for only a single object, while a *coded* copy can add a new choice simultaneously for multiple objects. For instance, suppose XOR of objects a, b is stored on a node that previously hosted neither a nor b. This adds a choice for both objects; one can now access a by downloading directly a or $(a+b,\, b)$, and access b by downloading directly b or $(a+b,\, a)$. 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.

We here consider d-choice storage allocation with r-XORs, which is implemented by distributing k exact and $k(d-1)/r$ of r-XOR'ed object copies evenly across the storage nodes while complying with Def. 1. Note that XOR'ed sets of objects shall not intersect pairwise at more than one object since this would violate the requirement that sets of choices for each must be disjoint. In addition, we here consider data recovery only from 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 and combinatorial designs [24]–[26]. For instance, 3-choice allocation given in (4) with object replicas is implemented with 2-XORs as

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

Allocation with r-XORs 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, load imbalance factor grows additively with r as stated in the following.

Theorem 3. *Consider a system with d-choice storage allocation created with r-XOR's, where $r \geq 2$ is an integer. When $d = o(\log(n))$, in the limit $n \to \infty$ we have almost surely*

$$\frac{1}{2} \leq \frac{\mathcal{I} \cdot d}{\log(n) + \beta_{n,d}} \leq 1, \tag{10}$$

where $\beta_{n,d} = r(d-1)\left(1 + \log_2(n) - \log\left(1 + r(d-1)\right)\right)$.

When $d = c\log(n)$ for some constant $c > 0$, in the limit $n \to \infty$ we have almost surely

$$\frac{1}{2} \leq \frac{\mathcal{I}}{(\alpha+1)\left(\frac{3}{2c\alpha} \cdot \frac{\log_2(n)}{\log(n)} + r\right)} \leq 1, \tag{11}$$

where α is the unique positive solution of $e^{-1/c} = (1+\alpha)e^{-\alpha}$.

Remark 3. Theorem 3 implies that d-choice allocation with r-XORs achieves the same scaling of $\mathcal{I}$ in d as if the service choices were created with replicas (as stated in Remark 2), while also reducing the storage requirement r times. However, accessing an object by a recovery set requires downloading r objects to recover one, thus, increasing the object access overhead r times. Consequently, $\mathcal{I}$ increases additively in r.

REFERENCES

[1] Jeffrey Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 1–1. ACM, 2009.

[2] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB*, 2010.

[3] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[4] Xue Ouyang, Peter Garraghan, Renyu Yang, Paul Townend, and Jie Xu. Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters. In *Fast Abstracts in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE, 2010.

[6] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Operating Systems Review*, 2010.

[7] Salvatore Sanfilippo. *Redis: An open source (BSD licensed), in-memory data structure store.*, 2019.

[8] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Proceedings of the sixth conference on Computer systems*, pages 287–300. ACM, 2011.

[9] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment*, 2012.

[10] Mohsen Sardari, Ricardo Restrepo, Faramarz Fekri, and Emina Soljanin. Memory allocation in distributed storage networks. In *IEEE International Symposium on Information Theory, ISIT 2010, June 13-18, 2010, Austin, Texas, USA, Proceedings*, pages 1958–1962, 2010.

[11] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

[12] P. Godfrey. Balls and bins with structure: balanced allocations on hypergraphs. In *Proceedings of the 19th annual ACM-SIAM symposium on discrete algorithms*, pages 511–517. Society for Industrial and Applied Mathematics, 2008.

[13] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271. ACM, 2004.

[14] M. Aktaş, S. Anderson, A. Johnston, G. Joshi, S. Kadhe, G. Matthews, C. Mayer, and E. Soljanin. On the service capacity region of accessing erasure coded content. In *, 2017 55th Annual Allerton Conference on Communication, Control, and Computing*, pages 17–24, 2017.

[15] Sarah E Anderson, Ann Johnston, Gauri Joshi, Gretchen L Matthews, Carolyn Mayer, and Emina Soljanin. Service rate region of content access from erasure coded storage. In *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2018.

[16] Mehmet Fatih Aktaş, Amir Behrouzi-Far, Emina Soljanin, and Philip Whiting. Load balancing performance in distributed storage with regular balanced redundancy. *arXiv preprint arXiv:1910.05791*, 2019.

[17] Ronald Pyke. Spacings. *Journal of the Royal Statistical Society: Series B (Methodological)*, 27(3):395–436, 1965.

[18] DA Darling. On a the test for homogeneity and extreme values. *The Annals of Mathematical Statistics*, pages 450–456, 1952.

[19] Eric Slud. Entropy and maximal spacings for random partitions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 1978.

[20] Luc Devroye. Laws of the iterated logarithm for order statistics of uniform spacings. *The Annals of Probability*, pages 860–867, 1981.

[21] Luc Devroye. Uniform and exponential spacings. In *Non-Uniform Random Variate Generation*, pages 206–245. Springer, 1986.

[22] Aleksandar Mijatović and Vladislav Vysotsky. On the weak limit law of the maximal uniform k-spacing. *Advances in Applied Probability*, 48(A):235–238, 2016.

[23] Paul Deheuvels and Luc Devroye. Strong laws for the maximal k-spacing when k c log n. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 66(3):315–334, 1984.

[24] Douglas R Stinson. *Combinatorial designs: constructions and analysis*. Springer Science & Business Media, 2007.

[25] DR Stinson, Ruizhong Wei, and Maura B Paterson. Combinatorial batch codes. *Advances in Mathematics of Communications*, 3(1):13–27, 2009.

[26] Natalia Silberstein and Anna Gál. Optimal combinatorial batch codes based on block designs. *Designs, Codes and Cryptography*, 2016.