Load Balancing Performance in Distributed Storage with Regular Balanced Redundancy

Mehmet Fatih Aktas†, Amir Behrouzi-Far*, Emina Soljanin* and Philip Whiting†
* Department of Electrical and Computer Engineering, Rutgers University
† Department of Engineering, Macquarie University,
{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 balance in the system 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 the load balance in the system improves the same way with \( d \) when the service choices are created with XOR’s 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 object copy requires downloading content from \( r \) nodes, which increases the load imbalance in the system additively by \( r \).

I. INTRODUCTION

Distributed computing systems are commonly built on a storage layer that provides data write/read service for the executing workloads. Therefore the overall performance of a compute system is strongly tied to the data access (I/O) performance implemented by the underlying storage system [1]. Data access times in modern large scale systems (e.g., Cloud 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 contention at the system resources [3]. Performance variability exists at any level of load, but it is aggravated by the overloaded storage nodes [4]. It is therefore paramount for the distributed systems to be able to 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 make data objects available in 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 (choices). 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 demand 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 over time. 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 demand for all objects stored in the 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 vectors of object demands are possible and equally likely as long as the cumulative demand for all stored objects remains constant. Each node can serve at most a load of 1, and the demand for each object can be split across its choices (nodes storing that object) as long as the cumulative load at each node remains below 1. We address the following questions:

Q1 What fraction of the object demand vectors (defined over all stored objects) that satisfy the constant cumulative condition can be served by the system?

Q2 If an object demand vector can be served, what is the smallest achievable load imbalance in the system? How does it change with respect to the design parameters: total number of nodes, number of service choices \( d \) provided for each object and the layout of the stored objects across the nodes?

Q3 Can we store XOR’s of multiple objects instead of object replicas in order to improve the load balance in the system with less storage overhead?

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 settings. In dynamic settings, requests for data objects arrive sequentially. Each arrival queries the current load at the nodes storing the requested object, and then goes to one of them according to some strategy based on the load information. Here, the objective is to design a request-to-node assignment strategy that minimizes the maximum load on any node (see e.g., [10], [11]). In static settings, the goal is either 1) to design storage-efficient schemes (e.g., batch codes) that
allow good load balance across the nodes for some set of expected offered loads \(I_2\), or 2) to understand which object demand vectors a given storage scheme (based on, for instance, some known erasure code) can support \(I_3\).

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 \(I_3\), \(I_4\). Unlike the previous work: i) We consider schemes that store more than a single object per node. ii) We find which fraction of a targeted service rate region can the considered schemes achieve without explicitly finding the complete achievable region. We were able to do that by establishing a connection of our problem with the uniform spacings (as described in Sec. II-B), iii) We address the question of load imbalance which is (as mentioned above) important in practice.

This paper is organized as follows: Sec. I presents the storage model, demand (offered load) model and its connection with the uniform spacings, and finally introduces the metrics we use to evaluate the load balancing performance in distributed storage. In Sec. II we discuss storage schemes in which each object is stored in single node and evaluate the system’s load balancing performance. In Sec. IV we discuss storage schemes in which each object is stored in \(d\) nodes, and evaluate the impact of \(d\) and the data layout across the storage nodes on the system’s load balancing performance. In Sec. VI we discuss creating multiple service choices for objects as recovery sets using XOR’s rather than object replicas, and evaluate the impact of this change on the storage overhead and the system’s load balancing performance.

**Note on the proofs, plots, and notation:** We place the proofs in the Appendix in order not to disrupt continuity of the text. Each point in Fig. 1 and 2 are computed by taking the average of \(10^5\) simulation runs. Within each simulation run, object demand vector is sampled according to the offered load model described in Sec. II-B and the load imbalance factor \(I\) for the system is computed according to its definition given 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.

We refer to the offered load for object \(o_i\) as its demand \(\rho_i\), which represents the average number of bytes streamed from the system per unit time to access \(o_i\), divided by a single node’s content access capacity. We refer to a node that hosts the object as a service choice for the object. Multiple choices for an object can be created either by replicating it over several nodes (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. VI). When XOR’ing is used, a 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 shall not interfere with accessing the same object through another choice, thus, different choices for the same object are disjoint.

Demand for an object can be arbitrarily split across its choices. When a load of \(\rho\) is exerted by an object on a recovery set, each node within the set that composes the choice will be exerted a load of \(\rho\). 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 each node within the system is stable. We assume that the object demands \(\rho_i\’s\) are split across their choices so that the load on the maximally loaded node is minimized. This can be done given the storage allocation and the value of \(\rho_i\’s\) by solving a norm minimization problem as described in Sec. II-D.

A storage allocation defines how each object is assigned, possibly with redundancy, to storage nodes. This paper focuses on the 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 VI. In the rest of the paper, unless otherwise noted, allocation itself will mean a regular balanced allocation.

**Connection with the batch codes:** An \((k, N, m, n, t)\) batch code encodes \(k\) objects into \(N\) copies and distributes them across the \(n\) nodes (possibly with redundancy) in such a way that any \(m\) of these objects can be accessed by reading at most \(t\) objects from any node \(I_2\). The goal while designing a batch code is to minimize the total storage requirement. Redundancy can be either in the form of replicating individual objects or encoding multiple objects together (e.g., XOR’ing). Multiset batch codes is concerned with a more general case in which the selection of \(m\) objects for access is done with replacement and each of them should be accessed separately, that is, content that is read for accessing an object cannot be used to access another object.

In Sec. IV we will consider storage allocations that are constructed by replicating objects across multiple nodes. Such allocations yield 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.
for a system denotes the fraction of object demand vectors that lie in the set $S$ for a system with a given storage allocation is the set of all object demand vectors $\{\rho_i\}_{i=1}^k$ under which the system is able to operate under stability. Access capacity for systems that store content with erasure coding was first studied in [17] based on block designs.

B. Offered Load and Uniform Spacing Model

We assume that the system is expected to operate under any object demand (offered load) vector $(\rho_1, \ldots, \rho_k)$ in the set $S = \{(\rho_1, \ldots, \rho_k) | \sum_{i=1}^k \rho_i = \Sigma, \rho_i \geq 0\}$. This connection allows us to use the results on uniform spacings to evaluate the load balancing performance in distributed storage. The results on the properties of uniform spacings available in the literature (see e.g. [18]) 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 2 (see e.g. [18]). Uniform spacings $(S_1, \ldots, S_k)$ is uniformly distributed over the simplex

$$\{(s_1, \ldots, s_k) | \sum_{i=1}^k s_i = 1, s_i \geq 0 \text{ for } i = 1, \ldots, k\}.$$  

Lemma 2 implies that the object demands $\rho_i$'s 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 the load balancing performance in distributed storage in terms of the performance metrics we define in the following Subsection.

C. Performance Metrics

We use two metrics to quantify the load balancing performance in distributed storage. Using these metrics, we answer the questions posed in the Introduction.

Definition 2. $P_S$ for a system denotes the fraction of object demand vectors that lie in $S$ and under which the system can operate under stability. In other words, since the object demand vector $(\rho_1, \ldots, \rho_k)$ is sampled uniformly at random from $S$, $P_S$ denotes the probability that the system defined by $S$ can operate under stability.

Definition 3 (I). In a system of $n$ storage nodes operating under a cumulative load of $\Sigma$, load imbalance factor $I$ for the system is defined as the load on the maximally loaded node divided by its minimum possible value, i.e., $\Sigma/n$. $P_S$ is obviously 0 when $\Sigma > n$, hence we assume $\Sigma \leq n$ implicitly throughout. Notice also that $I$ is always $\geq 1$.

D. Storage Access Capacity

We here explain how to express the access capacity that can be provided jointly for all objects stored within the system. In other words, we are interested in expressing the set of all object demand vectors under which the system with a given storage allocation can operate under stability. Access capacity for systems that store content with erasure coding was first studied in [13] and further studied in [14]. We adopt a formulation similar to the one introduced in [13]. The formulation presented in this Section gives the geometric interpretation of the performance metrics $P_S$ and $I$ that we introduced in Sec. II-C.

Definition 4. Service capacity region for a system with a given storage allocation is the set of all object demand vectors $\rho = (\rho_1, \ldots, \rho_k)$'s under which the system is able to operate under stability.

Let us consider a storage allocation in which object $o_i$ is stored in $d_i$ nodes. Then its demand $\rho_i$ can be distributed across its $d_i$ choices, each handling a fraction of $\rho_i$. If $\rho_i^{(j)}$ denotes the portion of $\rho_i$ that is assigned to the $j$th choice of $o_i$, then $\rho_i = \rho_i^{(1)} + \cdots + \rho_i^{(d_i)}$. We represent the stacked collection of all these per-node demand portions with the following $(d_1 + \cdots + d_k) \times 1$ vector:

$$x^T = (\rho_1^{(1)}, \ldots, \rho_1^{(d_1)}, \ldots, \rho_k^{(1)}, \ldots, \rho_k^{(d_k)}).$$

Converting back to $\rho$ from $x$ is a matter of matrix-vector multiplication $\rho = T \cdot x$, where $T$ is a binary matrix of size $k \times (d_1 + \cdots + d_k)$. 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

$$M \cdot x \preceq 1, \quad x \succeq 0,$$  

Proof. See Appendix X-A.

Batch codes with object replication are known as combinatorial batch codes and their construction is well studied [15], [16]. 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 that we consider here for the storage allocation. An approach to construct optimal $d$-uniform batch codes has been given in [17] based on block designs.
where \(\preceq\) and \(\succeq\) denote the standard partial orderings in \(\mathbb{R}^n\), and \(\mathbf{0}\) and \(\mathbf{1}\) denote all-zeros and ones vectors of length \(n\). Overall the service capacity region of the system is given as
\[
\mathcal{C} = \{\rho \mid M \cdot x \preceq I, \ T \cdot x = \rho, \ x \succeq 0\}. \tag{3}
\]

\(M\) expresses the storage allocation and is a binary matrix of size \(n \times (d_1 + \cdots + d_k)\). It is constructed by setting \(M[i, j]\) to 1 if the demand portion \(x[j]\) flows into node-\(i\), and to 0 otherwise. When storage redundancy is created with only object replicas, each column of \(M\) becomes a binary representation of a node that stores the corresponding object copy, precisely, each column of \(M\) would consist of a single 1, and the position of this 1 within the column is equal to the position of the reported node within the sequence of all nodes. For instance for the storage allocation \(\{(a, c), \ (b, a), \ (c, b)\}\), which stores \(a, b, c\) across three nodes by allocating two choices for each, we have
\[
x = \left(\rho_a^{(1)}, \rho_a^{(2)}, \rho_b^{(1)}, \rho_b^{(2)}, \rho_c^{(1)}, \rho_c^{(2)}\right), \quad M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.
\]

When storage redundancy is created with coding, then demand portions might be assigned to a recovery set rather than only a single node. When a demand portion of \(\rho\) is assigned to a recovery set, then \(\rho\) much capacity will be used up at each node within the recovery set. Then each column of \(M\) that consists of multiple 1’s represents a recovery set for the corresponding object. For instance for the storage allocation \(\{(a, b + c), \ (b, a + c), \ (c, a + b)\}\), we have
\[
x = \left(\rho_a^{(1)}, \rho_a^{(2)}, \rho_b^{(1)}, \rho_b^{(2)}, \rho_c^{(1)}, \rho_c^{(2)}\right), \quad M = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}.
\]

Lemma 3. Service capacity region for any storage system is a convex polytope.

Proof. Convex polytope expressed by \(\mathbb{L}\) in \(\mathbb{R}^{d_1 + \cdots + d_k}\) consists of all demand portion vectors \(x\)'s under which the system is stable. Capacity region is the linear transformation of this polytope by \(T\), so is another convex polytope in \(\mathbb{R}^n\).

As noted in Sec. II-A, we assume that object demands \(\rho_i\)'s are split across their choices such that the load on the maximally loaded storage node is minimized. This essentially means that for a given object demand vector \(\rho\), out of all demand portion vectors \(x\)'s that satisfy \(\mathbb{L}\), the system is assumed to use \(x^*\) that achieves the best possible load balance across the storage nodes, that is, \(x^*\) is the optimal solution for the following convex optimization problem:
\[
\min \|M \cdot x\|_\infty ; \quad T \cdot x = \rho, \ x \succeq 0, \tag{4}
\]
where \(\|\cdot\|_\infty\) denotes the infinity norm.

We next state a useful interpretation of \(\mathcal{P}_\Sigma\), which directly follows from the definition of \(\mathcal{P}_\Sigma\) (Def. 2) and the definition of service capacity region (Def. 3). It simply says that once the capacity region of a system is determined, evaluating \(\mathcal{P}_\Sigma\) for the system becomes a computational geometry problem.

Lemma 4. For a system with a given storage allocation, let the capacity region be the polytope \(\mathcal{C}\) and let \(S_\Sigma\) be defined for a given \(\Sigma\) as in \(\text{[1]}\). \(\mathcal{P}_\Sigma\) for the system is then given as
\[
\mathcal{P}_\Sigma = \frac{\text{Volume}(\mathcal{C} \cap S_\Sigma)}{\text{Volume}(S_\Sigma)}.
\]

Finding volumes or pairwise intersections of convex polytopes are well studied problems, and numerous efficient algorithms are available to compute both in the literature (see e.g., \cite{19}). Lemma 4 essentially gives a recipe to exactly compute \(\mathcal{P}_\Sigma\) for a system with any given storage allocation. This also 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 storage system, if \(\Sigma > \Sigma'\) then \(\mathcal{P}_\Sigma \leq \mathcal{P}_{\Sigma'}\).

Proof. See Appendix IX-B.

Copying an object to a node that did not previously host the object increments the number of service choices for the object. We study the improvement in the system’s load balancing performance in terms of \(\mathcal{P}_\Sigma\) and \(\mathcal{I}\) by storing objects redundantly across multiple nodes in Sec. IV and VI. We next state a simple but useful fact as the first step to understand the gains of increasing the number of service choices for the objects.

Lemma 5. Let a system’s capacity region be \(\mathcal{C}\) for a given storage allocation. Keeping the number of nodes fixed (i.e., keeping the total system capacity 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’s capacity region for this modified allocation. Then, \(\mathcal{C} \subset \mathcal{C}'\).

Proof. See Appendix IX-C.
Lemma 3 says that storing an additional redundant object copy in the system increases the volume of the service capacity region that can be 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. From this we can see that the added storage redundancy improves the load balancing performance in terms of the metrics defined in Sec. II-C.

Corollary 2. Consider a system with load balancing performance of $P_\Sigma$ and $I$. Suppose a new redundant object copy is stored in the system as described in Lemma 3, and let $P_\Sigma'$, $I'$ represent the metrics of load balancing performance for the modified system. Then we have $P_\Sigma' \geq P_\Sigma$ and $I' \geq I$.

Proof sketch. By Lemma 3, $C \subset C'$. This together with Lemma 4 directly implies $P_\Sigma' \geq P_\Sigma$. In order to see $I' \geq I$, it is enough to observe that for any given object demand vector $\rho$, a demand portion vector $x$ that achieves the best load balance (i.e., minimizes the load on the maximally loaded node as described in (4)) for the unmodified system is also achievable by the modified system because $C \subset C'$.

III. LOAD BALANCING WITH NO REDUNDANCY

Here we consider the allocation 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, where $n/k$. Demand 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 demand for all objects stored on it.

As discussed in Sec. II-B, object demand 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 without loss of generality that node $s_i$ stores the set of objects $o_{(i-1)m+1}, \ldots, o_{im}$. Then the 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 $l_i$’s must be $\leq 1$, and thus $\max\{l_1, \ldots, l_n\} \leq 1$ is necessary and sufficient for system stability. Therefore, $P_\Sigma$ is given by $\Pr\{\max\{l_1, \ldots, l_n\} \leq 1\}$. In the uniform spacing literature, $l_i$’s have been studied, and are called non-overlapping $m$-spacings. Their maximum has also been studied, and is called maximal non-overlapping $m$-spacing.

Definition 5 ($M_{k,m}^{(n)}$). Maximal non-overlapping $m$-spacing for $k$ uniform spacings on the unit line is defined for $k = mn$ as

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

Notice that $M_{k,m}^{(n)}$ denotes the load on the maximally loaded node in a system of $n$ nodes storing $k$ objects when the cumulative demand for all objects is 1. Using a combination of the ideas presented in [20–22], we can derive the following convergence results for $M_{k,m}^{(n)}$.

Lemma 6. 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

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

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

Proof. See Appendix IX-D.

Now we are ready to express the metrics $P_\Sigma$ and $I$ for the system with single-choice allocation in terms of $M_{k,m}^{(n)}$.

Lemma 7. In a system with single-choice storage allocation,

$$P_\Sigma = \Pr\left\{M_{k,m}^{(n)} < 1/\Sigma\right\}, \quad I = M_{k,m}^{(n)} \cdot n.$$

Proof. When the system operates under a cumulative offered load of $\Sigma$, the load on the maximally loaded node is given by $M_{k,m}^{(n)} \cdot \Sigma$. This together with the definition of $P_\Sigma$ (Def. 2) and $I$ (Def. 3) gives us 7.

Using Lemma 7 and Lemma 6 we determine the behavior of $P_\Sigma$ and $I$ for large $n$ as follows:

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

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

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

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 \( n \to \infty \)

\[
\mathcal{P}_{\Sigma_n} = \begin{cases} 
1 & \limsup b_n < m, \\
0 & \liminf b_n > m.
\end{cases}
\] (10)

Proof. See Appendix IX-1

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 the load imbalance factor \( \mathcal{I} \) decays multiplicatively with \( m \). The scaling of \( \mathcal{I} \) with \( \log(n) \) is aligned with the well-known result derived for the dynamic load balancing setting in 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 [10].

IV. LOAD BALANCING WITH \( d \)-FOLD REDUNDANCY

In this section we focus on systems with \( d \)-choice storage allocation, in which each of the \( k \) objects is stored on \( d \) different nodes (choices) and each of the \( n \) nodes stores \( kd/n \) different objects.

As discussed at the end of Sec. III, load imbalance \( \mathcal{I} \) in the system decays with the number of objects \( (m) \) stored per node. We here (and in Sec. VI) 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 for the general case with a fixed value of \( k/n > 1 \) using arguments that are very similar to those we discuss here.

In our study of systems with \( d \)-choice allocation, we find sufficient and necessary conditions for the system stability using the mathematical object that is known as maximal \( d \)-spacing, which we discuss in the following subsection.

A. Uniform spacings interlude

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

\[
M_{k,d} = \max_{i=0,\ldots,k-1} U_{i+d} - U_i \quad \text{or} \quad \max_{i=1,\ldots,k-1} \sum_{j=i}^{i+d-1} S_j,
\]

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

While deriving our main results, we extensively use the elegant results presented in [21], [23]–[25] on \( M_{k,d} \). We state the ones that we mainly use in the following.

Lemma 8. ([25] Theorem 1) For any integer \( d \geq 1 \), we have as \( k \to \infty \)

\[
\Pr \{ M_{k,d} \cdot k - \log(k) - (d-1)\log_2(k) + \log((d-1)!)) \leq x \} \to G(x).
\] (11)

Lemma 9. ([23] Theorem 2, 6) For \( d = o(\log(k)) \), we have as \( k \to \infty \)

\[
\frac{M_{k,d} \cdot k - \log(k)}{(d-1)(1 + \log_2(k) - \log(d))} \to 1 \quad \text{a.s.}
\] (12)

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

\[
\limsup_{k \to \infty} V_k = \beta^*(1 + \alpha)/\alpha \quad \text{a.s.}
\]

\[
\liminf_{k \to \infty} V_k = -\beta^!(1 + \alpha)/\alpha \quad \text{a.s.}
\] (13)

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

In order to evaluate the load balancing performance in systems with \( d \)-choice storage allocation, we need to consider the uniform spacings on the unit circle rather than the unit line segment that is previously defined in Def. 6

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

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

As we show in Appendix IX-E the results stated in Lemma 8 and 9 for the maximal \( d \)-spacing on the unit line carry over to its counterpart that is defined on the unit circle.
We next define a class of allocations in which the overlaps and node expansions are loosely controlled by a single parameter.

Definition 8 (Clustering design). Suppose \( d|n \). Then 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 9 (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

\[
\begin{bmatrix}
a \\
g \\
f \\
a \\
g \\
a \\
g \\
\end{bmatrix}.
\]

(15)

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 demand 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 10 (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 10. In a d-choice allocation with r-gap design, \( r \geq d-1 \) and \( x \leq |N(S)| \leq x+2r \) for any \( S = \{o_i, o_{i+1}, \ldots, o_{i+x-1}\} \) for \( i = 1, \ldots, n \).

Proof. See Appendix IX-F.

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

Lemma 11. Consider a system with d-choice storage allocation that is constructed with an \( r \)-gap design and operating under a cumulative offered load of \( \Sigma \). Then for system stability, a necessary condition is given as

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

(16)

and a sufficient condition is given as

\[
M^{(c)}_{n,r+1} \leq d/\Sigma.
\]

(17)

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

\[
\text{Pr}\left\{M^{(c)}_{n,i+1} \leq d/\Sigma \right\} \leq \mathcal{P}_\Sigma \leq \text{Pr}\left\{M^{(c)}_{n,i} \leq (i + 2r)/\Sigma \right\}.
\]
Proof. See Appendix [IX-G].

Notice that clustering or cyclic design is an \(r\)-gap design, hence the bounds in Lemma [11] are valid for storage allocation with either design. Their well-defined structure also allows refining the bounds on \(P_{\Sigma}\) as follows.

**Lemma 12.** In a \(d\)-choice allocation constructed with clustering or cyclic design

\[
\Pr\left\{ M_{n,d}^{(c)} \leq d/\Sigma \right\} \leq P_{\Sigma} \leq \Pr\left\{ M_{n,d+1}^{(c)} \leq 2d/\Sigma \right\}.
\]

**Proof.** See Appendix [IX-H].

Using the bounds given in Lemma [12], we can find an asymptotic characterization for \(P_{\Sigma}\) and \(I\) as follows.

**Theorem 2.** Consider a system with \(d\)-choice storage allocation constructed with clustering or cyclic design.

When \(d = o(\log(n))\), in the limit \(n \to \infty\) we have almost surely

\[
\frac{1}{2} \leq \frac{I \cdot d}{\log(n) + (d - 1)(1 + \log_2(n) - \log(d))} \leq 1,
\]

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

\[
P_{\Sigma_n} = \begin{cases} 1 & \limsup b_n/d < 1, \\ 0 & \liminf b_n/2d > 1. \end{cases}
\]

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

\[
\frac{1}{6} \leq \frac{2c\alpha}{3(\alpha + 1)} \cdot \frac{I \cdot \log(n)}{\log_2(n)} \leq 1,
\]

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

\[
P_{\Sigma_n} = \begin{cases} 1 & \limsup b_n \cdot 1.5\tau/d < 1, \\ 0 & \liminf b_n \cdot 0.25\tau/d > 1. \end{cases}
\]

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

**Proof.** See Appendix [IX-J].

**Remark 2.** Theorem [2] implies that \(I = \Theta(\log(n)/d)\) when \(d = o(\log(n))\), and \(I = \Theta(\log \log(n)/\log(n))\) when \(d = \Theta(\log(n))\). These imply that i) \(d\) choices for each object initially reduces load imbalance multiplicatively by \(d\), ii) exponential reduction in load imbalance kicks in as soon as \(d\) reaches \(\Theta(\log(n))\), i.e., \(I\) 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 [11], which were shown with the dynamic balls-into-bins model under the light offered load (when \(O(n)\) balls are sequentially placed into \(n\) bins) extend to the static setting under general offered load.

Fig. [1] plots \(I\) for \(n = 100\). Notice that the value of \(I\) close to \(\log(n)\) when \(d = 1\) as suggested by Theorem [1] and decays as \(1/d\) while incrementing \(d\) as suggested by Theorem [2]. This illustrates that our asymptotic results are close estimates of 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) allow controlling the overlaps between every pair of \(C_i\)'s.

**Definition 11 (BIBD, [16]).** 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 \(k = n\), 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 [14], block designs are possible only if \(k = d^2 - d + 1\). For instance a 3-choice allocation with block design looks like

\[
\begin{bmatrix} a & a & a \\ b & f & d \\ c & g & e \end{bmatrix}
\]

(22)
Lemma 13. Consider a system with \( d \)-choice allocation constructed with block design and operating under a cumulative offered load of \( \Sigma \). 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 \( M_{n,d}^{(c)} \leq d/2\Sigma \).

Proof. See Appendix [X-K].

Stability conditions given in Lemma [13] allow us to find bounds on \( \mathcal{P}_\Sigma \) and \( \mathcal{I} \) for storage allocations 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 in the bounds given in Theorem [2]. 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 design can possibly implement better scaling of \( \mathcal{I} \) in \( d \) compared to clustering or cyclic design.

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 \( C_i \)'s (recall [14]) 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 fewer but occasionally large overlaps. For instance, Fig. [2] shows load imbalance factor \( \mathcal{I} \) for systems with 3- and 5-choice allocations that are constructed using clustering, cyclic or block design. We see here 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 object choices \( C_i \)'s compare with each other in terms of \( \mathcal{P}_\Sigma \) or \( \mathcal{I} \). In the following Subsection, we present our intuitive reasoning on why it is better in terms of load balancing performance to implement consistently small overlaps across all \( C_i \)'s (as in block design) performs better than to shrink (or decouple) the overlaps between some objects and make them larger for others (as in clustering).

C. On the impact of overlaps between the service choices of objects in load balancing performance

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 \( (C_i)'s \) for the objects that are expected to be 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. [11-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.

Since the goal of storage redundancy is to allow good load balance in the presence of skews in object popularities, we first need to understand the popularity skew characteristics captured by our demand (offered load) model. Without loss of generality let us assume that the cumulative demand \( \Sigma \) is 1. Let \( N(\alpha, \beta) \) be the number of objects with a demand of \( \geq \alpha \) and \( \leq \beta \). Then
With cyclic design for smaller overlapping service choices performs on average worse than treating all objects the same and minimizing the overlaps between the choices of some objects come at the expense of enlarging the overlaps between the choices of others. However, as explained in detail previously in this Section, reducing the overlap of the choice of the objects that are close to the corners represent the load scenarios with skewed object popularities. Setting \( \rho \) unit length at the skew extends the skew corners by \( \frac{1}{2} \) and yields a simplex capacity region, which 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.

If we decouple the service choices for a set of objects, then the system can handle the load even when the popularity is extremely skewed towards those “lucky” objects. However, as explained in detail previously in this Section, reducing the overlaps between the choices of some objects come at the expense of enlarging the overlaps between the choices of others. Thus decoupling the choices for some hurts the performance for many others, that is why decoupling should be done only when its benefit outweighs the harm done, that is when some of the objects are known to be more popular than others.

\( N(\alpha, \beta) \) is given by the number of uniform spacings on the unit line that are within \([\alpha, \beta]\). An asymptotic characterization of this distribution has been given in [26] as follows.

**Theorem 3** \( R1 \). \( N(\alpha/k, \beta/k) \) is asymptotically normally distributed with an asymptotic mean and variance

\[
\mu_k \sim k \left( e^{-\alpha} - e^{-\beta} \right), \quad \sigma_k^2 \sim k \left( e^{-\alpha} - e^{-\beta} - (ae^{-\alpha} - \beta e^{-\beta})^2 \right).
\]

**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 3 tell us great deal about the object popularities implemented by our demand model: As \( k \to \infty \), i) \( R2 \) implies that almost surely, only finitely many of the object demands \( \rho_i \)'s are of the order of (small) magnitude \( 1/k^2 \), ii) \( R3 \) implies that almost surely, only finitely many of \( \rho_i \)'s are of the order of (large) magnitude \( \log(k)/k \), iii) \( R7 \) implies that most of \( \rho_i \)'s are of the order of (medium) magnitude \( 1/k \). These imply that 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 (\( \rho \sim 1/k \)).

If we decouple the service choices for a set of objects, then the system can handle the load even when the popularity is extremely skewed towards those “lucky” objects. However, as explained in detail previously in this Section, reducing the overlaps between the choices of some objects come at the expense of enlarging the overlaps between the choices of others. Thus decoupling the choices for some hurts the performance for many others, that is why decoupling should be done only when its benefit outweighs the harm done, that is when some of the objects are known to be more popular than others. Decoupling the choices for a particular set of \( p \) objects will mean enlarging the overlaps for others. What we observe in the simulations is what is expected; allocating choices for a selected set of objects with smaller overlapping service choices performs on average worse than treating all objects the same and minimizing the overlaps across the choices of all.

The rationale of favoring many but consistently small overlaps (as in block design) over fewer but occasionally larger overlaps (as in clustering) has very recently been observed to perform well also in the context of scheduling compute jobs with bi-modal job size distribution. Authors in [27] consider replicating every job arrival across \( r \) nodes, in which the overlaps between the set of nodes assigned to subsequent jobs impact queueing times at the nodes. 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 assignment strategies.

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

Fig. 5 plots the capacity region \( \mathcal{C} \) for a system of three servers and three objects with \( d \)-choice allocation that is constructed with cyclic design for \( d = 1, 2, 3 \), (Def. 9). 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, which 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.

We previously 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. 1). We here look into this phenomenon through the geometric interpretation of \( \mathcal{P}_\Sigma \) that was given in Lemma 4. As a Corollary of Lemma 5 capacity region for \( d \)-choice allocation is contained by that for \((d + 1)\)-choice allocation (e.g., as can be seen in Fig. 5). 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’s capacity.
region. $\mathcal{P}_\Sigma$ is proportional to $\text{Vol}(\mathcal{A})$ where $\mathcal{A} := \mathcal{S}_\Sigma \cap \mathcal{C}$ (by Lemma 4), 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) center (examples in Fig. 3 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. 3. Suppose $\Sigma = 3$, then $\text{Vol}(\mathcal{S}_\Sigma) = 9\sqrt{3}/2$. 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 with equal amount 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, the increase in $\text{Vol}(\mathcal{A})$ so 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 the problem posed in the Introduction. So 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 adds a new service choice for only a single object, while a $2$-XOR’s are $\Sigma = 3$, $n = 3$, $d = 1$.

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

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, 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 $d = o(\log(n))$, in the limit $n \to \infty$ we have almost surely

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

(24)

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

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

(25)
When \( d = c \log(n) \) for some constant \( c > 0 \), in the limit \( n \to \infty \) we have almost surely
\[
\frac{1}{2} \leq \frac{\mathcal{I}}{(\alpha + 1) \left( \frac{3}{2} \frac{\log_2(n)}{\log(n)} + r \right)} \leq 1,
\]
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
\[
\mathcal{P}_{\Sigma_n} = \begin{cases} 
1 & \limsup 1.5 \tau \cdot b_n / d < 1, \\
0 & \liminf 0.25 \tau \cdot b_n / d > 1,
\end{cases}
\]
where \( \tau = c (1 + \alpha)^2 / \alpha \).

**Proof.** See Appendix [X-L].

**Remark 3.** Theorem 3 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 (24) and (26).

**A. A note on the 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 (25) is constructed based on a symmetric BIBD with \( \lambda = 1 \) (see Def. 11 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. CONCLUSIONS**

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 modern 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 a 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 multiplicatively by \( r \).

**VIII. ACKNOWLEDGEMENTS**

Authors would like to thank Gala Yadgar for the fruitful discussions and recommendations in the early stages of this work. This research is supported by the National Science Foundation under Grants No. CIF-1717314.

**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 Townsend, 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.
Lemma 14. For $k = mn$, we have
\[ M_{k,m}^{(n)} = \max \{ \Gamma_1, \ldots, \Gamma_n \} / \Gamma_1 + \cdots + \Gamma_n \text{ in distribution}, \]
where $\Gamma_i$'s are i.i.d. 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 14

\[ \text{Proof.} \] 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 the allocation graph. First, by König’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 $(k, kd, n, n, 1)$ multiset batch code. Given that the graph is $d$ regular, it can only qualify for a $(k, kd, d, n, 1)$ multiset batch code.

B. Proof of Corollary 1

\[ \text{Proof.} \] Let the system store $k$ objects, and let us denote its capacity region with $C$, and denote $C \cap \Sigma \cap T_c$ with $T_c$. Notice that $S_{\Sigma'}$ is obtained by scaling $S_{\Sigma}$ down with $\Sigma' / \Sigma$, hence we have
\[ \text{Volume}(S_{\Sigma'}) / \text{Volume}(S_{\Sigma}) = (\Sigma' / \Sigma)^k. \]
For any \( x \in \mathcal{T}_z \subseteq \mathcal{C} \), by the convexity of \( \mathcal{C} \) its scaled version \( \Sigma' / \Sigma : x \) also lies in \( \mathcal{C} \) (notice that \( 0 \in \mathcal{C} \)). Thus the scaled down version of \( \mathcal{T}_z \) with \( \Sigma' / \Sigma \), which we denote by \( \mathcal{T}_z' \), will also lie in \( \mathcal{C} \). We also know that
\[
\text{Volume}(\mathcal{T}_z') / \text{Volume}(\mathcal{T}_z) = (\Sigma' / \Sigma)^k.
\]
We have \( \mathcal{T}_z' \subseteq \mathcal{C} \cap \mathcal{S}_{\Sigma'} \), then
\[
\mathcal{P}_{\Sigma'} = \frac{\text{Volume}(\mathcal{C} \cap \mathcal{S}_{\Sigma'})}{\text{Volume}(\mathcal{S}_{\Sigma'})} \geq \frac{\text{Volume}(\mathcal{T}_z')}{\text{Volume}(\mathcal{S}_{\Sigma'})} = \frac{\text{Volume}(\mathcal{S}_{\Sigma'}) \cdot (\Sigma' / \Sigma)^k}{\text{Volume}(\mathcal{S}_{\Sigma'})} = \mathcal{P}_{\Sigma'}.
\]

\[\square\]

C. Proof of Lemma 5

**Proof.** Let the initially given storage allocation that yields the capacity region \( \mathcal{C} \) be described by the matrices \( T \) and \( M \) (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 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 \( T \) and \( M \). 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 \( T' \) and \( M' \) respectively.

First, we show that any point in \( \mathcal{C} \) also lies in \( \mathcal{C}' \). Let us define \( D = \{ x \mid M \cdot x \leq 1, \ x \geq 0 \} \), and \( D' \) similarly with \( M' \). Let \( p \in \mathcal{C} \), then there is an \( x \) in \( D \) such that \( p = T \cdot x \). Let us generate \( x' \) by appending a 0 at the end of \( x \). Then, \( x' \in D' \) since \( M' \cdot x' = M \cdot x \leq 1 \), and \( T' \cdot x' = p \). Thus, \( p \) 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 \((d + 1)/C\) of demand for the tagged object and zero demand for all other objects, while it could not supply this before the modification is implemented on the storage allocation. This together with the fact that \( \mathcal{C} \subseteq \mathcal{C}' \) implies \( \mathcal{C} \subseteq \mathcal{C}' \).

**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 both allocation matrices \( T \) and \( M \). The same arguments used above for the case with replication can be easily repeated here to show \( \mathcal{C} \subseteq \mathcal{C}' \) holds for this case as well. \[\square\]

D. Proof of Lemma 6

**Proof.** Proof of (5). By Lemma 14 we have
\[
M_{k,m}^{(n)} = \frac{\max \{ \Gamma_1, \ldots, \Gamma_n \} }{ \Gamma_1 + \cdots + \Gamma_n } \text{ in distribution,}
\]
where \( n = k/m \) and \( \Gamma_i \)'s are i.i.d. as Gamma with a shape parameter of \( m \) and a rate of 1.

From Darling [20, Sec. 3], we know for fixed \( m \), as \( n \to \infty \)
\[
\Pr \left\{ \frac{1}{M_{k,m}^{(n)}} > m \frac{n}{\log(n)} - (m-1)m \frac{n \log_2(n)}{\log^2(n)} + m \log(\Gamma(m)) \frac{n}{\log(n)} + m - \frac{n}{\log^2(n)} \right\} \to G(x). \tag{28}
\]
From this we get
\[
\Pr \left\{ M_{k,m}^{(n)} < \frac{\log(n)}{mn} \left( 1 - (m-1) \frac{\log_2(n)}{\log(n)} + \frac{\log(\Gamma(m))}{\log(n)} - \frac{x}{\log(n)} \right)^{-1} \right\} \to G(x).
\]
Defining
\[
\alpha_n = \frac{x}{\log(n)} + (m-1) \frac{\log_2(n)}{\log(n)} - \frac{\log(\Gamma(m))}{\log(n)}
\]
we can write
\[
\Pr \left\{ M_{k,m}^{(n)} < \frac{\log(n)}{mn} (1 - \alpha_n)^{-1} \right\} \to G(x).
\]
Using Taylor expansion on \(1/(1-\alpha_n)\), we can write
\[
\Pr \left\{ M_{k,m}^{(n)} < \frac{1}{mn} \left( x + \log(n) + (m-1) \log_2(n) - \log(\Gamma(m)) \right) \right\} \to G(x),
\]
which gives us (5).
Proof of \([6]\): Here we apply the approach taken in the proof of \([\Sigma]\) Theorem 2.1. Let us first define
\[
X_n = \frac{M_{k,m}^{(n)} \cdot mn - f_n}{\log(n)}.
\]
Then \([5]\) becomes
\[
\Pr \{(X_n - 1) \log(n) < x\} \to G(x).
\]
Let \(G_n\) be the distribution function of \((X_n - 1) \log(n)\), and if \(\delta_n = O(1)\) and \(x_n = \delta_n \log(n) \to \infty\) as \(n \to \infty\), then
\[
\Pr \{|X_n - 1| > \delta_n\} = G_n(-x_n) + 1 - G_n(x_n)
\]
\[
\leq G_n(-x_n) - G(-x_n) + G(x_n) - G_n(x_n) + 1 - G(x_n)
\]
\[
\leq 2 \sup_x |G_n(x) - G(x)| + G(-x_n) + 1 - G(x_n).
\]
Let us write
\[
G(x_n) = \exp(-\exp(-\delta_n \log(n))) = \exp(-n^{-\delta_n}) = 1 - n^{-\delta_n}.
\]
where \((a)\) comes from the fact that \(\exp(-x) \geq 1 - x\). Then we have in the limit \(n \to \infty\)
\[
\Pr \{|X_n - 1| > \delta_n\} \leq 2 \sup_x |G_n(x) - G(x)| + G(-x_n) + 1 - G(x_n) = O(n^{-\delta_n}).
\]
Let us define the subsequence \(n_k\) for positive integer \(k\) where \(n_k\) is the greatest integer \(\leq \exp(\sqrt{2}\sqrt{k})\). Taking
\[
\delta_{n_k} = \frac{\log(k)}{(\alpha \sqrt{k})}
\]
for \(\alpha < \sqrt{2}\) (necessary for \(\delta_{n_k} \log(n_k) \to \infty\) as \(k \to \infty\)), we find
\[
\sum_{k=1}^{\infty} n_k^{-\delta_{n_k}} = \sum_{k=1}^{\infty} n_k^{-\log(k)/(\alpha \sqrt{k})} \leq \sum_{k=1}^{\infty} k^{-\sqrt{2}/\alpha} < \infty.
\]
Hence by Borel-Cantelli’s Lemma, \(|X_{n_k} - 1| > \delta_{n_k}\) only finitely often. Recall we defined \(n_k\) as
\[
\exp(\sqrt{2}\sqrt{k} - 1) \leq n_k \leq \exp(\sqrt{2}\sqrt{k}),
\]
using which we find
\[
\sqrt{k} \geq \log(n_k)/\sqrt{2}, \quad \log(k - 1) \leq 2 \log_2(n_k) - \log(2).
\]
Then we can write
\[
\delta_{n_k} \leq \frac{\log(k - 1)}{\alpha \sqrt{k}} \leq \frac{\sqrt{2}}{\alpha} \cdot \frac{2 \log_2(n_k) - \log(2)}{\log(n_k)}.
\]
This together with the above result implies that
\[
|X_{n_k} - 1| > \frac{\sqrt{2}}{\alpha} \cdot \frac{2 \log_2(n_k) - \log(2)}{\log(n_k)}
\]
only finitely often. This implies that \(X_{n_k} \to 1\) a.s.

Our goal is to show \(X_n \to 1\) a.s. In order to do this, let us first write \(X_n\) as \(\alpha_n/\beta_n\) for
\[
\alpha_n = M_{k,m}^{(n)} \cdot mn - f_n, \quad \beta_n = \log(n).
\]
Both \(\alpha_n\) and \(\beta_n\) are increasing in \(n\). Let us then pick a sample path \(w\) so that \(X_{n_k}(w) \to 1\), and observe that if \(n_k \leq n < n_{k+1}\) then
\[
\frac{\alpha_{n_k}(w)}{\beta_{n_k+1}} \leq \frac{\alpha_n(w)}{\beta_n} \leq \frac{\alpha_{n_{k+1}}(w)}{\beta_{n_k}}.
\]
The terms the right and left can be written as
\[
\frac{\beta_{n_k}}{\beta_{n_k+1}} \cdot \frac{\alpha_{n_k}(w)}{\beta_{n_k}} \leq \frac{\alpha_n(w)}{\beta_n} \leq \frac{\alpha_{n_{k+1}}(w)}{\beta_{n_{k+1}}} \cdot \frac{\beta_{n_{k+1}}}{\beta_{n_k}}.
\]
We have
\[
\sqrt{2}\sqrt{k} - 1 \leq \beta_{n_k} \leq \beta_{n_{k+1}} \leq \sqrt{2}\sqrt{k} + 1.
\]
Given that \(\sqrt{k + 1}/\sqrt{k} - 1 \to 1\) as \(k \to \infty\), we have \(\beta_{n_{k+1}}/\beta_{n_k} \to 1\). This shows that \(X_n(w) \to 1\), which implies \(X_n \to 1\) a.s. \(\square\)
E. Maximal d-spacing on the unit circle

We here show that the maximal d-spacing $M_{k,d}$ for $k$ ordered uniform samples on the unit circle (see Def. 7) converge to its counterpart $M_{k,d}^c$ that is defined on the unit line. We are aware that almost sure convergence implies, convergence in probability, which then implies convergence in distribution. However in the following, we present the convergence first in distribution, then in probability, finally in almost sure sense because we believe it contributes to a better understanding of the arguments.

Lemma 15. For $d < k$,

$$\Pr \{ M_{k,d} > x \} \leq \Pr \left\{ M_{k,d}^c > x \right\} \leq \frac{k}{k-d} \Pr \{ M_{k,d} > x \}.$$  \hspace{1cm} (29)

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

First inequality is easy to see; if a sequence of spacings $s = (s_1, s_2, \ldots, s_k) \in L$ 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 $m = (s_1, \ldots, s_{i+d-1})$. Shifting (by feeding what is shifted out back in the sequence at the opposite end) $s$ to left by at most $i-1$ times will preserve $m$, hence each of the $i-1$ shifted versions will also lie in $L$. Similarly, shifting $s$ to right by at most $k - (i + d - 1)$ times will also preserve $m$. We call such permutations, which are obtained by shifting with wrapping around, a cyclic permutation.

Let us introduce a set $L' \subseteq L$ 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 $L' \subseteq L \subseteq C$ 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 (29) is given as follows. Recall that the uniform samples, together with the 0 point, are ordered on the unit circle as $0, U_1, \ldots, U_{k-1}$. 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 $1, \ldots, k$.

From Lemma 15 the convergence in distribution as stated in the following easily follows.

Corollary 3. For $d = o(k)$, $M_{k,d}^c \rightarrow M_{k,d}$ in distribution as $k \rightarrow \infty$.

Lemma 16. 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 15 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.

Lemma 16 allows us to invoke the well known fact and conclude that if $M_{k,d} \rightarrow X$ in distribution then $M_{k,d}^c \rightarrow X$ in distribution as well. In order to use the results known for the convergence of $M_{k,d}$ in probability or a.s. while we work with $M_{k,d}^c$, we need the following Lemma.

Lemma 17. 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 17 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 $0 \leq U_{(1)} \leq \ldots \leq U_{(2m+1)} \leq 1$ 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 $M_{k,d_k}^{(c)}$ 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 unit 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 $P$,

$$M_{k,d_k}^{(c)} \leq \max \{ M_{k,d_k}^{(O)}, M_{k,d_k}^{(P)} \}.$$  \hfill (30)

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

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

by the strong law of large numbers. Therefore $\exists K_N$ such that $\forall k \geq K_N, N_k \geq \frac{2a}{k}$ and $a > 0$ a.s. Since $d_k = o(k)$ it follows that $\exists K_N$ 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 (30) holds $\forall k \geq K_d$ a.s.

Now we are ready to prove Lemma 17 that is to show $M_{k,d}/M_{k,d} \to 1$ a.s.

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

$$\liminf_{k \to \infty} M_{k,d}^{(c)} / M_{k,d} = 1.$$  \hfill (31)

Next using (30), we have

$$M_{k,d}^{(c)} / M_{k,d} \leq \max \{ M_{k,d}^{(O)}, M_{k,d}^{(P)} \}.$$  \hfill (32)

This allows us to find

$$\limsup_{k \to \infty} M_{k,d}^{(c)} / M_{k,d} \leq \max \{ \limsup_{k \to \infty} M_{k,d}^{(O)} / M_{k,d}, \limsup_{k \to \infty} M_{k,d}^{(P)} / M_{k,d} \}.$$  \hfill (33)

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

$$\limsup_{k \to \infty} M_{k,d}^{(c)} / M_{k,d} \leq 1.$$  \hfill (34)

Putting (31) and (32) together completes the proof.

In the following, we show that the results given in Lemma 8 and 9 for $M_{k,d}$ carry over to $M_{k,d}^{(c)}$.

**Lemma 18.** For any integer $d \geq 1$, we have as $k \to \infty$

$$\Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - (d-1) \log_2(k) + \log((d-1)!) \leq x \right\} \to G(x).$$  \hfill (35)

**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 $U_0, \ldots, U_{k-1}$, 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

$$\Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} = \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E \right\} + \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E^c \right\}.$$  \hfill (36)

Left hand side of the sum above can be bounded as

$$\Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E \right\} \leq \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\} - \Pr \{ E^c \},$$

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

$$\Pr \{ A; B \} = \Pr \{ B \} - \Pr \{ B; A^c \} \geq \Pr \{ B \} - \Pr \{ A^c \}.$$  \hfill (37)

Putting this in (36) gives us

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

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 $P$.
where
\[ \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x; E^c \right\} \to 0 \]
since \( \Pr \{ E^c \} = d/k \to 0 \) as \( k \to \infty \). Overall this gives us
\[ \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \geq \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\}. \]

Given that \( M_{k,d}^{(c)} \geq M_{k,d} \), we have the lower bound
\[ \Pr \left\{ M_{k,d}^{(c)} \cdot k - \log(k) - f_k < x \right\} \leq \Pr \left\{ M_{k,d} \cdot k - \log(k) - f_k < x \right\}. \]

Both the lower and upper bounds given above are equal and converge to \( G(x) \) by Lemma \( \ref{lem:G} \) hence showing (33). Another way to show (33) is given as follows. By Lemma \( \ref{lem:19} \) (stated below), we know
\[ \frac{M_{k,d}^{(c)} \cdot k - \log(k) - f_k}{M_{k,d} \cdot k - \log(k) - f_k} \to 1. \]

This together with
\[ M_{k,d} \cdot k - \log(k) - f_k \to G \] in distribution, and Slutsky’s theorem gives us (33).

\begin{lemma}
For \( d = o(\log(k)) \), we have as \( k \to \infty \)
\[ \frac{M_{k,d} \cdot k - \log(k)}{(d - 1) (1 + \log_2(k) - \log(d))} \to 1 \quad \text{a.s.} \]
\end{lemma}

\begin{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
\[ f \left( M_{k,d}^{(c)} \right) \geq f \left( M_{k,d} \right) \]
for \( k \) sufficiently large a.s. By (12) in Lemma \( \ref{lem:9} \) the right hand side of the above inequality \( \to 1 \) as \( k \to \infty \) a.s. Then we have
\[ \lim \inf_{k \to \infty} f \left( M_{k,d}^{(c)} \right) \geq 1. \] (35)

Inequality (30) gives us
\[ f \left( M_{k,d}^{(c)} \right) \leq f \left( \max \left\{ M_{k,d}^{(O)}, M_{k,d}^{(P)} \right\} \right) \]
for \( k \) sufficiently large a.s. This implies that
\[ \limsup_{k \to \infty} f \left( M_{k,d}^{(c)} \right) \leq \max \left\{ \limsup_{k \to \infty} f \left( M_{k,d}^{(O)} \right), \limsup_{k \to \infty} f \left( M_{k,d}^{(P)} \right) \right\}. \]

By (12) in Lemma \( \ref{lem:9} \)
\[ \limsup_{k \to \infty} f \left( M_{k,d}^{(O)} \right) = 1 \quad \text{and} \quad \limsup_{k \to \infty} f \left( M_{k,d}^{(P)} \right) = 1 \quad \text{a.s.} \]

Hence we have
\[ \limsup_{k \to \infty} f \left( M_{k,d}^{(c)} \right) \leq 1. \]

This together with (35) concludes the proof.
\end{proof}

\begin{lemma}
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
\[ \limsup_{k \to \infty} V_k = c^* (1 + \alpha)/\alpha \quad \text{a.s.} \]
\[ \liminf_{k \to \infty} V_k = -c^* (1 + \alpha)/\alpha \quad \text{a.s.} \] (36)
\end{lemma}
where \( \alpha \) is the unique positive solution of \( e^{-1/c} = (1 + \alpha) e^{-\alpha} \), and \( c^* \) and \( c^1 \) 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 19 given above.

\[ \]

F. Proof of Lemma 10

Proof. Suppose \( r < d - 1 \). Pick an arbitrary object \( o_i \). Then object \( i - r \) is co-located together with object \( i \) on one of the nodes in \( C_i \), which we refer to as \( n^* \). Given that \( n^* \) is a choice for \( o_i \), any object stored on it need to be one of \( o_{i-r}, \ldots, o_{i+r} \). Then, the remaining \( d - 2 \) storage slots of \( n^* \) (other than \( o_i \) and \( o_{i-r} \)) need to be occupied by all objects between \( i - r + 1 \) and \( i - 1 \), because otherwise \( C_{i-r} \) would have to overlap with a \( C_j \) such that \( i < j \leq i + r \) and this would mean the choices of \( o_{i-r} \) overlap with the choices of an object that is further than \( r \) objects, hence violate the \( r \)-gap design requirement. Thus there needs to be at least \( d - 2 \) different objects between \( o_{i-r+1} \) and \( o_{i-1} \), implying \( r \geq d - 1 \).

Let us pick an arbitrary set of \( x \) consecutive objects \( S = \{o_i, \ldots, o_{i+x-1}\} \). Allocation defines a regular bipartite graph, then by Hall’s theorem \( |N(S)| \geq x \). The copies of \( o_i \) can expand across at most \( n_{i-r}, \ldots, n_{i+r} \), and the copies of \( o_{i+r-1} \) can expand at most across \( n_{i+r-1}, \ldots, n_{i+x-1+r} \). Then \( S \) can expand at most across \( n_{i-r}, \ldots, n_{i+x-1+r} \), meaning \( |N(S)| \leq x + 2r \). \[ \]

G. Proof of Lemma 17

Proof. Necessary condition: System is surely unstable if a set of objects \( S \) has a cumulative offered load larger than \( |N(S)| \). Lemma 10 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 (16).

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 \( M_n^{(c)} \cdot \Sigma \leq d \) (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 \( \rho_{p_j} \) for \( o_i \) is \( d \) when \( i = x + (r + 1) j, j = 0, 1, \ldots, [n/(r + 1)] \) and 0 otherwise. In this case, offered load of \( d \) for each spiky object \( o_i \) can be supplied by using up the capacity in all the nodes in its \( d \) choices since all other objects that overlap with \( o_i \) in their choices have 0 offered load (by the \( r \)-gap design property). System can supply the spiky load regardless of the value for \( x \). Given that the system’s service capacity region is convex (Lemma 8), any convex combination of any set of spiky load scenario’s 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 (17).

H. Proof of Lemma 12

Proof. Lower bounds come from substituting \( r = d - 1 \) in those given in Lemma 11. 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.

I. Proof of Theorem 2

Proof. Recall that the load at the maximally loaded node \( l_{n}^{\max} \) is given by \( M_{k,m}^{(n)} \cdot \Sigma \). Almost sure convergence given in (6) implies for \( \Sigma = b_n \cdot n / \log(n) \) that

\[
l_{n}^{\max} \cdot m/b_n \rightarrow 1 \text{ a.s.}
\]

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

\[
\Pr \{l_{n}^{\max} \cdot m/b_n - 1 > \delta \} = \Pr \{l_{n}^{\max} > b_n/m \cdot (1 + \delta) \} + \Pr \{l_{n}^{\max} < b_n/m \cdot (1 - \delta) \} \rightarrow 0.
\]

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

\[
\Pr \{l_{n}^{\max} > b_n/m \cdot (1 + \delta) \} \rightarrow 0, \quad \Pr \{l_{n}^{\max} < b_n/m \cdot (1 - \delta) \} \rightarrow 0.
\]

Recall from (7) that \( \mathcal{P}_{\Sigma} \) is given by \( \Pr \{l_{n}^{\max} < 1 \} \). Then the convergence of probabilities given above implies (10). (8) and (9) come from substituting \( I = M_{k,m}^{(n)} \cdot n \) (by (7)) in the convergence results given in Lemma 6.

J. Proof of Theorem 2

Proof. We first need to recall Lemma 12 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 the system stability. Here we will refer to \( d \) as \( d_n \) to make it explicit that it is a sequence in \( n \).

Proof of (19): In this case \( d = o(\log(n)) \). Almost sure convergence given in (12) together with Lemma 17 implies for \( \Sigma = b_n \cdot n / \log(n) \) that

\[
M_{n,d}^{(c)} \cdot \Sigma/b_n \rightarrow 1 \text{ a.s.}
\]
Recall that $M^{(c)}_{n,d} \cdot \Sigma_n/d_n \leq 1$ is sufficient and $M^{(c)}_{n,d} \cdot \Sigma_n/2d_n \leq 1$ is necessary for system stability, which respectively imply that $\mathcal{P}_{\Sigma_n} \rightarrow 1$ if $\lim \sup_{n \rightarrow \infty} b_n/d_n < 1$, and $\mathcal{P}_{\Sigma_n} \rightarrow 0$ if $\lim \inf_{n \rightarrow \infty} b_n/2d_n > 1$, hence (19).

**Proof of (21):** In this case $d = o(\log(n))$. Almost sure convergence given in (13) together with Lemma 17 implies for $\Sigma_n = b_n \cdot n/\log(n)$ that in the limit $n \rightarrow \infty$ we have almost surely

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

Then the sufficient and necessary conditions (as used in the previous step while showing (19)) for system stability imply (21).

**Proof of (18) and (20):** In order to prove (18), 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 $M^{(c)}_{n,d} \cdot \Sigma \leq dC$ and $M^{(c)}_{n,d} \cdot \Sigma \leq 2dC$. Using these we find that $C \geq M^{(c)}_{n,d} \cdot \Sigma/d$ is sufficient and $C \geq M^{(c)}_{n,d} \cdot 2\Sigma/d$ is necessary for system stability. This means that the maximum load on any node in the system will lie in $[M^{(c)}_{n,d} \cdot \Sigma/2d, M^{(c)}_{n,d} \cdot \Sigma/d]$, which implies that the load imbalance factor $T$ for the system lies in $[M^{(c)}_{n,d} \cdot n/2d, M^{(c)}_{n,d} \cdot n/d]$.

Finally using the results of almost sure convergence given for $M_{n,d}$ in Lemma 9 (hence given for $M^{(c)}_{n,d}$ as well due to Lemma (17)), we find (18) and (20).

**K. Proof of Lemma 13**

**Proof.** We use the following fact, which we refer to as $F$ here: in a storage allocation with block design, every pair 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 pick an arbitrary object $a_i$ 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 $S \setminus a_i$ is hosted on the same node. However, this does not prevent all objects within $S \setminus a_i$ 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 11 let $x$ be an integer in $[0, n]$, and the offered load for $a_i$ is $\rho$ if $i = x + dj$ for some $j = 0, 1, \ldots, \lfloor n/d \rfloor$ 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, system is stable only if $\rho \leq 1 + (d - 1)/2$, which gives us the sufficient condition for stability.

**L. Proof of Theorem 2**

**Proof.** This proof is very similar to that of Theorem 2 except that it is more subtle due to the fact 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 subtle and lengthy), then we derive our results by studying these auxiliary cyclic allocations.

We start by showing sufficient and necessary conditions for the 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 $a_i$ that is primarily stored on node $s_i$, $d - 1$ choices (recognition 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 $[1, 2, 3, 4, 5, 6]$ with $r = 2$, pairs of nodes $(s_2, s_4)$ and $(s_1, s_2)$ can jointly serve the object $a_0$ that is primarily stored on $s_1$ (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 truly 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, $s_4$ and $s_3$ form a choice for $a$, which is a proper cyclic choice, while $s_3$ and $s_4$ 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. Capacity region of non-cyclic (our regular balanced) allocation with XOR’s contains that of its cyclic counterpart. It is slightly subtle to see why this statement is true; we explain it as follows. 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

\[
M = \begin{bmatrix}
a & b & c & d & e & f \\
e + f & [a + b] & [a] & [d] & [e] & [f] \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.
\] (37)

We next briefly explain what \(M\) represents. 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 exerted 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). 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 (37) would be

\[
M^c = \begin{bmatrix}
a & b & c & d & e & f \\
e + f & [a + b] & [a] & [d] & [e] & [f] \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.
\] (38)

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 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 implementing. Thus, it is better to have less overlap between choices in terms of achieving greater capacity region. This is the “inefficiency” of cyclic allocation that causes it to achieve smaller capacity region than its non-cyclic counterpart.

Let \(D = \{x \mid M \cdot x \leq 1, \ x \succeq 0\}\), and \(D^c\) be defined similarly with \(M^c\). It is easy to see that any \(x\) in \(D^c\) will also lie in \(D\) (recall the additional 1’s in \(M^c\)). In addition, non-cyclic \(d\)-choice allocation and its cyclic counterpart share the same \(T\) (i.e., the other allocation matrix that yields the capacity region \(C\) (or \(C^c\)) by transforming \(D\) (or \(D^c\); recall from Sec. II-D). Thus, we have \(C \supseteq C^c\). This together with Lemma 4 implies that probability \(P_{\Sigma}\) 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 1 how we found 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 \(M^{(c)}_{n,1+r(d-1)} \cdot \Sigma \leq d\), where \(\Sigma\) is the cumulative offered load on the system and \(M^{(c)}_{n,1+r(d-1)}\) 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 following up 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, \( P \) for cyclic \( d \)-choice allocation is a lower bound for that of its non-cyclic counterpart. Thus, the sufficient condition \( M^{(c)}_{n,1+r(d-1)} \cdot \Sigma \leq d \) for the stability of cyclic allocation 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 with those given in part (i).

**Step 1: Cyclic-plus allocation.** Recall in part (i) that we created a cyclic \( d \)-choice allocation by firstly 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 (37) and its cyclic counterpart in (38)). Let us also consider and use the incurred non-cyclic choices for object access here, and refer to this form of the allocation as cyclic-plus.

**Cyclic-plus achieves greater capacity region than non-cyclic.** Capacity region of cyclic-plus allocation will contain that of its non-cyclic counterpart, which together with Lemma 4 implies that \( P \) for cyclic-plus allocation will be at least as large as that of its non-cyclic counterpart. This is because cyclic-plus allocation implements all the choices that its non-cyclic counterpart implements and plus some additional choices (e.g., compare (37) with (38)), which will yield at least as large of 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 allocation, 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 \( s_{i+1} \mod n \) 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 (38), its first cyclic choice is \((b, a+b)\) and its only non-cyclic choice is \((f+\alpha, \alpha)\), where both \( b \) and \( f+\alpha \) 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 choice need to 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’s capacity.

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 of \( >2d-1 \), that is when \( M^{(c)}_{n,1+r(d-1)} \cdot \Sigma > 2d-1 \), hence a necessary condition for its stability is given as \( M^{(c)}_{n,1+r(d-1)} \cdot \Sigma \leq 2d \). This is easy to see using the exact same arguments we used to show the corresponding necessary stability condition in Lemma 11 for \( d \)-choice allocation with object replicas.

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

In the remainder, we will refer to \( d \) as \( d_n \) to make it explicit that it is a sequence in \( n \). We will also refer to \( 1+r(d-1) \) as \( D \).

**Proof of (25) and (27).** Follows from the exact same arguments used in the Proof of respectively (19) and (21) (Theorem 2).

**Proof of (24) and (26).** Using the same arguments used in the Proof of (18) and (20) (Theorem 2), we can conclude here that load imbalance factor \( \bar{T} \) for the system lies in \([M^{(c)}_{n,D} \cdot n/2d, M^{(c)}_{n,D} \cdot n/d]\). And again similarly, using the results of almost sure convergence given for \( M_{n,D} \) (hence for \( M^{(c)}_{n,D} \) due to Lemma 17 in Lemma 9) we find (24) and (26).