H-0211 December 7, 2003
Computer Science

IBM Research Report

Reliability in Large-Capacity Bandwidth-Limited Storage Systems

Ami Tavory, Vladimir Dreizin, Shmuel Gal
IBM Research Division
Haifa Research Laboratory
Haifa 31905, Israel

Meir Feder
Department of EE-Systems
Tel-Aviv University
Israel

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Reliability in Large-Capacity Bandwidth-Limited Storage Systems

Ami Tavory[†], Vladimir Dreizin[†], Shmuel Gal[‡], and Meir Feder[§]
IBM Haifa Research Laboratories
Email:{tavory,dreizin,gals}@il.ibm.com, meir@eng.tau.ac.il

Abstract— **As time goes by, storage systems are required to reliably store increasing amounts of data. Some earlier solutions, designed for relatively small systems, are reaching their limit. The problem of storage reliability is aggravated by ongoing tendencies of (networked) storage systems, *e.g.*, the growth gap between the capacity of each device and its bandwidth, and the advent of mass-produced cheap devices of decreasing reliability.**

In this work we study bounds and solutions for very large systems of high capacity devices. Using simple information-theoretic bounds, we show the eventual necessity of a scalable hierarchical system. We next describe and analyze such a system. The salient points of our system are protection of data by their estimated access-intensity, unequal error-protection by data importance, assumption of a lenient and realistic disk-replacement policy, and competitive transition of data between different levels by predicted access-intensity.

Keywords— **storage, reliability, bounds, codes, unequal error-protection, low-density generation-matrix, competitive algorithms.**

I. Introduction

Storage devices are prone to the failure of an entire device, the failure of some device blocks, and the corruption of some device blocks. This causes storage systems to be susceptible to data loss. To combat the failure of an entire device or device blocks, a systematic *MDS* (maximum distance separable) code [1], [2] is typically used. *E.g.*, mirroring or *RAID* (redundant array of inexpensive devices) level 5 [3], [4] are used to correct a single failure. To combat data corruption, a combination of error detection and erasure correction is typically used.

Much of the work up to date is relevant for devices with limited capacity [3], [4], [5]. The growth gap between device capacity and bandwidth, indicates a longer device-recovery period. As this trend continues, the law of large numbers indicates that concurrent multiple failures will become increasingly common.

Consider a system composed of a large number of devices, each with large capacity and limited bandwidth. We show in Section II that such a group has the reliability behavior of a vector-communication channel with increasing erasure probability, no feedback, and error memory. The system has the same capacity of such a channel [6], [7], and can be protected at most by an optimal code for such a channel [8], [9], [10], [11], [12], [13], [1], [14]. This simple reduction has some consequences: an upper bound on the amount of reliable data as a function of device capacity and failure statistics, a lower bound on the growth of device numbers as a function of the amount of user data, a lower bound on the growth of recovery activity as a function of device capacity, a lower bound on the average coding-group size as a function of the amount of user data, and a lower bound on recovery activity as a function of the amount of user data.

The preceding reduction has consequences for uniform reliability-schemes. In any such scheme, an increase in user-data amount must cause a decrease in access performance.

In practice, data access requirements are not uniform; data can be classified, by activity, into slowly changing sets [15], [16], [17], [18], [19], [20], [21], [22], [23]. This observation appeared in [24] as the rationale for a viable alternative to configuring RAID levels. A two-level system employing mirroring and RAID-5 was used. Data was transited by access-intensity estimation between levels.

We extend this idea further. The lower-intensity level itself can degrade the system performance, since data must constantly transit between the two levels. The above reduction can be applied to the lower level alone. Eventually, the low-intensity level must become a bottleneck. It can be similarly shown that a system employing any fixed number of reliability schemes (*e.g.*, two in the case of [24]), must eventually suffer performance degradation.

We consider, therefore, a scalable hierarchy employing a parameterized family of reliability schemes, as illustrated in Figure 1. Exploiting the law of large numbers, the failure fraction of a large enough group of devices can be predicted accurately. Using this accurate prediction, we code large groups of relatively inactive data using low redundancy codes. Increasingly smaller groups of increasing activity are simultaneously maintained. Groups of increasing activity are protected using higher performance codes, at the cost of higher redundancy.

The storage devices we consider store blocks. It is sometimes beneficial to group blocks into *objects* [25]. An object is a collection of blocks considered related by its creator. The benefits of objects are twofold. The mapping of blocks to objects gives the system an indication of usage correlation. The granularity of objects allows a feasible-complexity attachment of attributes to data.

† Also at the Dept. of EE-Systems, Tel-Aviv University.
‡ Also at the Dept. of Statistics, University of Haifa.
§ At the Dept. of EE-Systems, Tel-Aviv University.

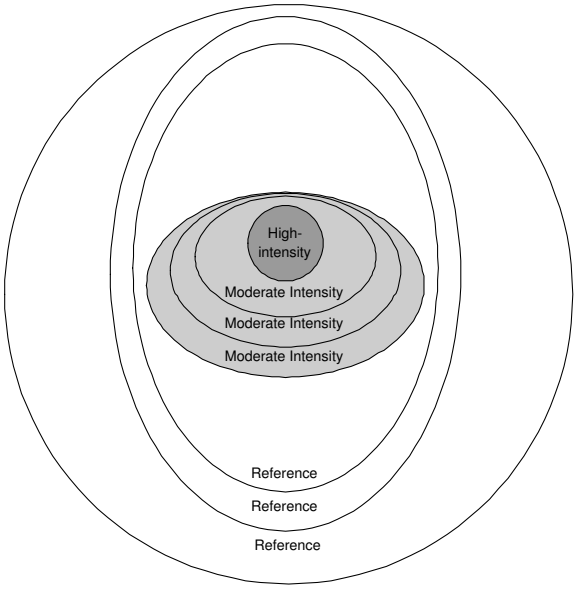

Fig. 1. A hierarchical storage system grouped by data access intensity.

We use the former benefit when considering the transition policy between levels. We use the latter benefit to attach to each object a *UEP* (unequal error-protection) attribute [2], [26]. UEP is a coding scheme wherein data are protected differentially according to user specification. We discuss the importance of UEP in storage, in Sections III and **??**.

The hierarchical scheme requires a policy of data transition between levels. The importance and design of *on-line* competitive algorithms for the case of a memory cache, have been extensively studied [27], [28], [29], [30], [31], [32], [33]. The reliability setting differs somewhat than the memory cache setting. In Section III we discuss this and show a solution adapted from [34].

In Section **??** we discuss a code used in the following sections as a building block. We require a low-complexity code which has a *recovery locality* property, *i.e.*, a code whose number of accessed members in recovery is proportional to the number of failed members, and not to the size of the coding group. We propose the use a variation of *LDGM* (low-density generation-matrix) codes [8], [9], [10], [11]. Independently, the use of standard LDGM codes for storage was proposed [35], [36] for reasons of computational efficiency alone, and under the caveat that these codes are probabilistic. We prove analytically and verify by simulation that under a realistic device-failure assumption, the recovery property of this variation is effectively deterministic.

Reference data are characterized by rarity of updates. This can be exploited for cost effective solutions. We show in Section **??**, that hierarchical large-group coding is applicable. Reference data can be further classified by expectation of read intensity. For reference data rarely read, we show that the bounds from Subsection II can be approached. For reference data often read, we show that while the data is well protected, read activity is not hindered by ongoing recovery activity.

In Section **??** we deal with the bulk of non-reference data. Typically, the data are *striped*, *i.e.*, they are stored on different devices protected in cross-device groups. Different schemes are used to utilize parallelism in both reading and writing recovery activity [5], [37], [38], [39], [40], [35], [36]. We use a hierarchy of random striping meant to minimize failure-correlation between coding groups. Similar schemes were independently proposed in [35], [36]. To the best of our knowledge the reliability analysis is first performed here. This analysis allows to better determine the required code redundancy.

At the extreme of the hierarchical system, highly-active data are stored. Good performance in this level is crucial for good overall system-performance. It is easy to show that the only MDS codes which do not require read-modify-update operations for data modifications are mirroring schemes. Previous work has analyzed this using small-length Markov chains, modelling the deterministic recovery-time by exponentially-distributed random variables [4], [41]. Multi-way mirroring is not well modelled by Markov chains, as the model loses recovery parallelism. The exponential random variable can attain values arbitrarily close to zero. We show new lower bounds and approximations on the system reliability for multi-way mirroring.

Much of the previous work has made use of strong assumptions on error-detection latency and replacement-device availability. Device-failure detection was assumed to be immediate; replacement-device availability was assumed to be unlimited. We consider a model in which device failure is detected with some latency, and replacement devices are inserted into the system once in an *epoch*, which is a pre-determined period of time. The large coding-group and random-sparing schemes explicitly take epochs explicitly into account. The mirroring scheme assumes that replacement devices are available without limit. By keeping the ratio of mirrored-data small, this does not contradict the device-replacement epoch assumption.

A. Paper Layout

We continue the introduction with definitions and notations in Subsection I-B, related work in Subsection I-C, and our contribution in Subsection **??**.

In Section II we show bounds on system reliability and performance which prove the necessity of a scalable hierarchy. In Section III we show a policy for transiting data between levels. In Section **??** we describe a variation of LDGM codes which we use as a building block in the following sections. In Section **??** we describe the *reference-data* hierarchy, used for protecting rarely-updated data. In Section **??** we describe the *moderate-intensity* hierarchy, used for protecting the bulk of non-reference data. In Section VII we describe the *high-intensity* level, used for

protecting the data estimated to be most active. For simplicity, we describe Sections **??**, **??**, and VII in the context of uniform error-protection and in a setting devoid of block-corruption. In Section **??** we extend the solutions in these two directions.

B. Definitions and Notations

Let C be a component of a system (*e.g.*, a storage device, a group of devices, or the entire storage system). The continuous random-variable describing the time to failure of the component is t_C^f. The *reliability* of the component at time t is [42]

$$R_C(t) = \mathbf{P}\left(t_C^f > t\right). \tag{1}$$

The *CDF* (cumulative distribution function) and *PDF* (probability density function) of t_C^f are

$$F_C(t) = 1 - R_C(t), \tag{2}$$

$$f_c(t) = -\frac{d\,R_C(t)}{d\,t}, \tag{3}$$

respectively. Similarly, the *conditional reliability* is

$$R_C(t|t') = \mathbf{P}\left(t_C^f > t | t_C^f > t'\right). \tag{4}$$

We consider a system S storage devices. Each device is composed of c *blocks* containing b consecutive bits, and has a read/write bandwidth of r blocks per time unit.

The devices have i.i.d. (independently and identically distributed) entire-device failure laws. An entire device fails with the exponential PDF $e^{-\lambda t}$. A *block erasure* is a detected corrupted block, and occurs with probability P_E. A *block substitution* is an undetected corrupt block, and occurs with probability P_S. If a block is substituted, we assume each bit has been flipped with probability P_s. *E.g.*, $P_s = \frac{1}{2}$ corresponds to a random earlier version of block contents.

An entire-device failure is detected with latency t_d. In some cases, we assume detection rounds of length t_d, wherein at the start of each round, the devices which have failed throughout the last round are detected. The former model can easily be modelled by the latter, and vice versa. The failure-detection assumption differs from some classical works on reliability, (*e.g.*, [42]).

The amount of user data stored in the entire system is c_{us}. The *MTTF* (mean time to failure) of a component C is the mean of t_C^f. The *MTTDL* (mean time to data loss) of a component C is the mean time until the failure of a component in C will cause data within it to be irreversibly lost. The *system MTTDL* is the MTTDL of S, the entire system. For uniform error-protection, let ν be the ratio between the system MTTDL and the time to sequentially read a single device. We consider systems in which the system MTTDL (equivalently ν) can be made arbitrarily high as c_{us} and $|S|$ grow large. The *storage rate* of the system is $\rho = \frac{c_{\text{us}}}{|S|\cdot c} \leq 1$ (some of the related work term this the *storage efficiency*). The maximal amount of data which can be stored with such ν is $c_{\max}^R(S) = c_{\max}^R(S,\nu)$. For unequal error-protection, we consider d priority levels with corresponding ratios $\nu_1, \ldots, \nu_d$. For any object e, we denote its priority by $\mu(e) \in [d]$.

An *IO* (input output (operation)) is the act of reading from or writing to a storage device. A *user IO* is an IO directly servicing a user's request. A *recovery IO* is a failure-triggered IO serving to recover information from a failed device or block. An *update IO* is an IO used to protect data against possible failure.

C. Related Work

Reliability has been extensively studied for communication channels, most relevantly the erasure channel and the binary symmetric channel [13], [43]. Bounds on channel capacity were found for uniform error-protection [6] and unequal error protection [2], [26]. Well known code-families for uniform error-protection are Hamming codes [1] and BCH codes [1] with some of their variants, *e.g.*, Reed-Solomon codes [44]. More recently, low-complexity codes have been studied [12], in particular, capacity achieving codes *e.g.*, Turbo codes [45], LDPC, and LDGM codes [10], [9], [46].

Reliability in storage systems was originally studied in the context of small-capacity systems [47], [3], [4], and in conjunction with performance improvement via parallelism, *e.g.*, RAID. The schemes were later extended in some directions. Concatenated codes were studied, *e.g.*, two-dimensional codes [48], and new RAID levels [41]. Questions on coding-group placement within devices were studied, *e.g.*, various distributed striping and sparing techniques [49], [40], [39], [38], [50], [37]). Effects of physical device-topologies were studied [51]. Storage reliability via coding was extend in the direction of disaster recovery as well [52].

The important idea of hierarchical protection of data based on data activity was shown in work on HP-AutoRAID [53], a work to which ours is an extension. Differential coding based on data activity was studied in the context of very large, concrete systems [35], [36], [54].

Later work in storage reliability has considered larger-capacity systems, and therefore a failure model taking into account multiple simultaneous errors. Some excellent analytical work can be found in [55], [56], [57]. New work oriented toward overall system design and implementation and the conjunction of several system aspects (*e.g.*, security, load-balancing, and meta-data location) can be found in a series of papers on OceanStore [35], [36], [54].

Our contributions are in bounding the capabilities of very large-capacity storage systems, and showing solutions for them, focusing primarily on the growth gap between device capacity and bandwidth.

We show an upper bound on the amount of data which can be stored reliably as a function of the number of devices and single-device attributes. Using this upper bound,

we show lower bounds on the number of storage devices as a function of system user-data, and lower bounds on the number of recovery and update IOs as a function of user data. Using this, we show the necessity of a *scalable hierarchy*, *i.e.*, a hierarchical system in which the number of levels increases with the number of user data.

We show and analyze a hierarchical reliability system. We modify a competitive algorithm for data transition between levels. We develop a variation of LDGM codes which has asymptotically optimal storage-rate and good recovery-locality. We show a capacity-achieving system for reference data. We develop and analyze a random-sparing scheme. We perform new analysis on multi-way mirroring.

II. Bounds

In this section we deal with bounds on storage-system reliability as the devices' capacity grows. These bounds are absolute limits, in the sense that they cannot be exceeded by any choice of codes, disk-replacement policy, or data-migration policy. The results in this section justify the necessity of a hierarchical reliability system. The bounds in this section hold even for cases in which $P_E = P_S = 0$.

The section is organized as follows. In Subsection II-A we show an upper bound on system reliability, by reducing the problem to one of communication over a faulty channel. Using the upper bound, we show lower bounds on cost and performance. In Subsections II-B we show lower bounds on the number of devices and on recovery activity. In Subsection II-C we show lower bounds on coding group-size and update activity.

A. Upper Bound on System Reliability

In this subsection we show an upper bound on system reliability.

We will use the following definition in the expression of the system reliability. Let S be a system suffering of device failures, block erasures, and block substitutions.

Definition 1: (Effective Volume and Block Capacity) The *effective volume* is defined as

$$\hat{c}\left(S\right) = r \max_{c_1,\ldots,c_n} \left\{ \sum_{i=1}^{n} c_i \mid \exists_{C_1,\ldots,C_n} \mathbf{P}\left(\bigwedge_{i=1}^{n} t^f_{C_i} \geq c_i \right) \geq \frac{1}{\nu} \right\}. \tag{5}$$

The *block capacity* is defined as

$$\begin{aligned} \hat{b}\left(S\right) = {} & b\left(1-P_E\right) + H\left(P_E\right) + P_E \log\left(P_E\right) + \\ & \sum_{i=1}^{b} \left(\binom{2^b}{i} \left(1-P_E\right) P_S P_s^i \left(1-P_s\right)^{b-i} \cdot \right. \\ & \left. \log\left(\binom{2^b}{i} \left(1-P_E\right) P_S P_s^i \left(1-P_s\right)^{b-i} \right) \right) + \\ & \left(1-P_E\right)\left(P_S\left(1-P_s\right)^b + \left(1-P_S\right)\right) \cdot \\ & \log\left(\left(1-P_E\right)\left(P_S\left(1-P_s\right)^b + \left(1-P_S\right)\right)\right). \end{aligned} \tag{6}$$

The main result of the subsection is that regardless of coding techniques, data migration policies, or device replacement policies, the system reliability is determined by the sizes in Definition 1.

Theorem 1: For any system S,

$$c^R_{\max}\left(S\right) \leq \hat{c}\left(S\right)\hat{b}\left(S\right). \tag{7}$$

The proof follows almost immediately from the data-processing inequality and Shannon's channel-capacity theorem[13], [7]. We write it in some detail for two reasons. The proof emphasizes the limit of replacement devices' contribution to the system reliability. The nature of a capacity approaching solution (see Subsection **??**) becomes apparent from the proof.

In the remainder of this subsection, we consider *epochs*, each taking $\frac{c}{r}$ time. An epoch is divided into *steps* of duration $\frac{1}{r}$. We consider an epoch starting at time 1. We define the following sets of devices:

$$\begin{aligned} \mathcal{X} &= \text{set of system devices at time 1,} \\ \mathcal{Y} &= \text{set of replacement devices inserted by time} \frac{c}{r}. \end{aligned} \tag{8}$$

We also define the following sets of blocks:

$$\begin{aligned} X(i) &= \text{set of distinct } \mathcal{X} \text{ blocks read at time } i, \\ \hat{X}(i) &= \text{set of distinct } \mathcal{X} \text{ blocks written at time } i, \\ Y(i) &= \text{set of distinct } \mathcal{Y} \text{ blocks written at time } i, \\ X_{us} &= \text{set of user data at time 1,} \\ X_r &= \text{set of recovered data at time } \frac{c}{r}. \end{aligned} \tag{9}$$

For any k' and k'', we define the sets

$$\begin{aligned} & \left\langle X(i) \right\rangle_{i=k'}^{k''} = \\ & \quad \left\{ X(i) \mid i = k', k'+1, \ldots, k'' \right\}, \\ & \left\langle X(i), \hat{X}(i), Y(i) \right\rangle_{i=k'}^{k''} = \\ & \quad \left\{ X(i), \hat{X}(i), Y(i) \mid i = k', k'+1, \ldots, k'' \right\}. \end{aligned} \tag{10}$$

The following lemma shows that the information pertinent for recovery, written at step j, is contained in the information read and written up to j, and the information read in j.

Lemma 1: For any[1] j

$$\begin{aligned} & H\left(X_{us} \middle| \left\langle X(i) \right\rangle_{i=\frac{c}{r}-j}^{\frac{c}{r}}, \left\langle X(i), \hat{X}(i), Y(i) \right\rangle_{i=1}^{\frac{c}{r}-j-1} \right) = \\ & \quad H\left(X_{us} \middle| \left\langle X(i) \right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}}, \left\langle X(i), \hat{X}(i), Y(i) \right\rangle_{i=1}^{\frac{c}{r}-j-2} \right). \end{aligned} \tag{11}$$

[1] We use $H(\cdot)$ to denote the binary-entropy function

Proof: By the definition of conditional entropy,

$$
\begin{aligned}
&H\left(X_{us}|\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-1}\right)= \qquad (12)\\
&H\left(X_{us},\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)\\
&+H\left(\left(\hat{X}(j),Y(j)\right)|\right.\\
&\qquad\left.X_{us},\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)\\
&-H\left(\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)\\
&-H\left(\left(\hat{X}(j),Y(j)\right)|\right.\\
&\qquad\left.\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right).
\end{aligned}
$$

Note that at time j, all information written to devices in $\mathcal{X}$ or $\mathcal{Y}$, is a function of all the information which had been read up to this point. Therefore,

$$
\begin{aligned}
&H\left(\left(\hat{X}(j),Y(j)\right)|\right. \qquad (13)\\
&\qquad\left.X_{us},\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)=\\
&H\left(\left(\hat{X}(j),Y(j)\right)|\right.\\
&\qquad\left.\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)=\\
&0.
\end{aligned}
$$

Inserting (13) into (12), we obtain

$$
\begin{aligned}
&H\left(X_{us}|\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-1}\right)= \qquad (14)\\
&H\left(X_{us},\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)\\
&-H\left(\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right)\\
&=H\left(X_{us}|\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j-1}^{\frac{c}{r}},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-2}\right).
\end{aligned}
$$

The lemma now follows from the definition of conditional entropy. ■

In the following, we write $H\left(\cdot|\cdot,c'_S\right)$ to denote the conditional entropy when a total of c'_S distinct blocks were read from $\mathcal{X}$ during the epoch.

Lemma 2: For any c'_S,

$$
H\left(X_{us}|X_r,c'_S\right)\leq c'_S\hat{b}\left(S\right). \qquad (15)
$$

Proof: Using the data processing inequality and Lemma 1, we have,

$$
\begin{aligned}
&H\left(X_{us}|X_r,c'_S\right) \qquad (16)\\
&\overset{(a)}{\leq} H\left(X_{us}|\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}},c'_S\right)\\
&\overset{(b)}{=} H\left(X_{us}|X(\frac{c}{r}),\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-1},c'_S\right)\\
&\overset{(c)}{=} H\left(X_{us}|\left\langle X(i)\right\rangle_{i=\frac{c}{r}-1},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-2},c'_S\right)\\
&\vdots\\
&\overset{(d)}{=} H\left(X_{us}|\left\langle X(i)\right\rangle_{i=\frac{c}{r}-j},\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}-j-1},c'_S\right)\\
&\vdots\\
&\overset{(e)}{=} H\left(X_{us}|\bigcup_{i=1}^{\frac{c}{r}}X(i),c'_S\right)\\
&\overset{(f)}{\leq} c'_s\hat{b}\left(S\right).
\end{aligned}
$$

In the above, inequality (a) follows from the fact that $X_{us}\rightarrow\left\langle X(i),\hat{X}(i),Y(i)\right\rangle_{i=1}^{\frac{c}{r}}\rightarrow X_r$ is a Markov chain, and so the data processing inequality applies, and (b) through (e) follow from a repeated application of Lemma 1.

Inequality (f) follows from Shannon's channel-capacity theorem. Assume a sender sends consecutive blocks over a communication channel suffering from the given block-erasure and block-substitution probabilities. Let $B=\{s_1,\ldots s_{2^b}\}$ denote the set of 2^b possibilities of each block, and let $D(s_i,s_j)$ denote the Hamming distance between s_i and s_i. The corresponding channel is shown in Figure 2, where S_e stands for an erasure. The transition matrix of a block, is the $2^b\times(2^b+1)$ matrix

$$
\left(\begin{array}{c}\underline{r}\\ \underline{\pi_2\left(\underline{r}\right)}\\ \vdots\\ \underline{\pi_b\left(\underline{r}\right)}\end{array}\right), \qquad (17)
$$

where

$$
\begin{aligned}
\mathbf{P}\left(s_e\mid s_1\right) &= P_E, \qquad (18)\\
\mathbf{P}\left(s_i\mid s_1\right) &= P_S(1-P_E)P_s^{D(s_i,s_1)}\left(1-P_s\right)^{b-D(s_i,s_1)},
\end{aligned}
$$

and

$$
\underline{r}=\left[p_{1,1},p_{1,2},\ldots,p_{1,2^b-1},p_{1,2^b},,p_{1,e}\right], \qquad (19)
$$

and $\pi_i\left(\underline{r}\right)$ $(i=2,\ldots,b\ )$ are vectors whose entries form a permutation of the entries of $\underline{r}$. Inequality (f) follows by using the Kuhn-Tucker [58] rule. ■

Theorem 1 now follows from Lemma 2. By Definition 1, if $c'_S\gneq\hat{c}_S$, then with probability $\frac{1}{\nu}$, data is lost. It follows that the expected number of epochs until data loss is ν.

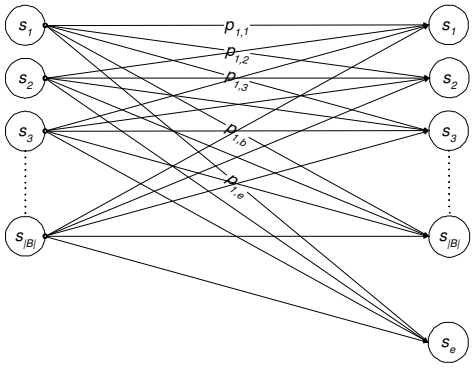

Fig. 2. Channel corresponding to a single block.

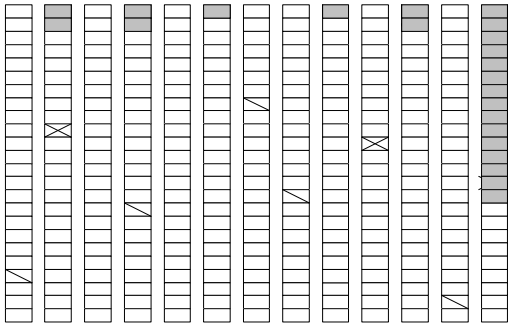

Fig. 3. Effective capacity of a group of devices.

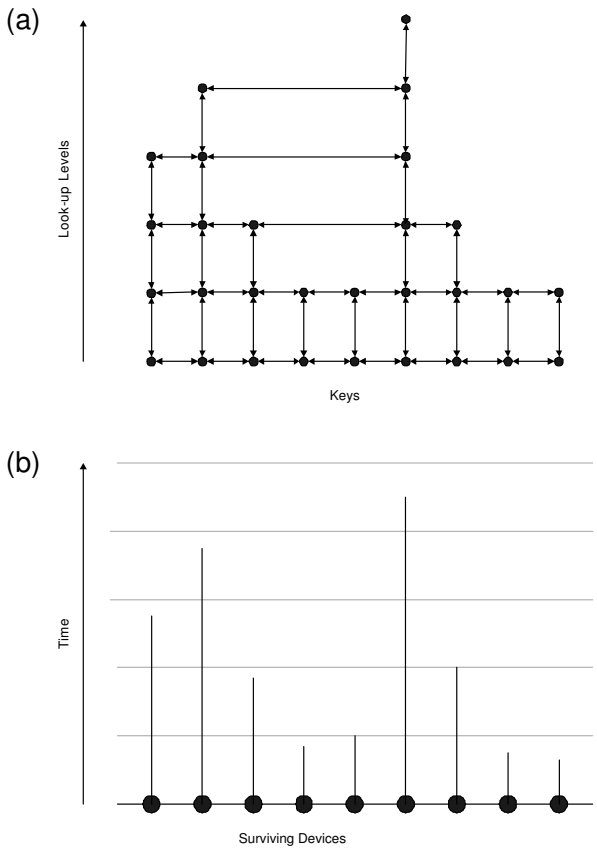

Fig. 4. A skip-list and a storage-device lifeline.

The proof of Theorem 1 gives the following interpretation to the amount of data which can be reliably stored. In Figure 3 we see a diagram of devices read concurrently from bottom to top. A greyed rectangle indicates that the device had failed prior to reading the block. A crossed-out rectangle indicates a substituted block. A partially crossed-out rectangle indicates an erased block. The proof shows that the amount of reliable data in the system is determined by $\langle X(i)\rangle_{i=1}^{\frac{c}{r}}$. That is, the amount of reliable data is equal to that which can be sent over a (non-memoryless) channel accommodating a block per device per time unit, with substitution probability determined by P_S, and erasure probability (with error-memory) determined by P_E and λ.

To achieve the system capacity, each block read should effectively contain user data. For the system to remain reliable, this should be the case for the next epoch as well. It follows that $\langle Y(i)\rangle_{i=k'}^{k''}$ is the only set of blocks which need be written during an epoch. The non-greyed rectangles form a shape with a given area. To achieve capacity, all realistic shapes with the given area should suffice to recover the user data. For a large number of devices, this is not difficult. We return to this point in Subsection **??**.

B. Increase in Storage-Device Number

In this subsection we show that an unlimited growth in c_{us} while keeping n constant, *cannot* be sustained by an unlimited increase in c, since $c_{\max}^{R}(S)$ grows non-negligibly only up to a limited value of c. Rather, an unlimited growth in c_{us} can only be sustained by a corresponding increase in n. We stress that a solution scalable in c_{us} must be scalable in n.

It was shown in Subsection II-A that the amount of data which can be stored reliably is bounded by $\langle X(i)\rangle_{i=1}^{\frac{c}{r}}$. Conceptually, as c grow indefinitely, so can $\langle X(i)\rangle_{i=1}^{\frac{c}{r}}$. Given a realistic failure model, this is not the case. This is similar to the number of levels in a skip list [59], shown in diagram (a) of Figure 4. In a skip-list the number of non-negligible levels is effectively limited by the number of items. Similarly, the maximal time at which a non-negligible number devices still function, is not unlimited, as shown in diagram (b) of Figure 4.

Theorem 2: for a system S with n devices, the effective

capacity is bounded by

$$c^R_{\max}(S) \leq \quad (20)$$
$$2nr\left(1-\left(\frac{1}{2}\right)^{\frac{\frac{c}{r}}{\frac{\ln(2)}{\lambda}}}\right)(1+o(1))\hat{b}(S)$$
$$\stackrel{c\to\infty}{\longrightarrow} 2nr\hat{b}(S)(1+o(1)).$$

Proof: By the properties of the exponential distribution,

$$R_C\left(t+\frac{\ln(2)}{\lambda} \mid t\right) \leq \frac{1}{2}. \quad (21)$$

By the law of large number, with probability one,

$$\left|\left\{C\in S \mid t_C \geq i\frac{\ln(2)}{\lambda}\right\}\right| \leq \quad (22)$$
$$\frac{1}{2^{i-1}}n(1+o(1)),$$

and so, with probability one,

$$\langle X(i)\rangle_{i=1}^{\frac{c}{r}} \leq \quad (23)$$
$$r\sum_{i=0}^{\frac{\frac{c}{r}}{\frac{\ln(2)}{\lambda}}}\left|\left\{C\in S \mid t_C \geq i\frac{\ln(2)}{\lambda}\right\}\right|$$
$$2nr\left(1-\left(\frac{1}{2}\right)^{\frac{\frac{c}{r}}{\frac{\ln(2)}{\lambda}}}\right)(1+o(1)).$$

The theorem follows by combining the above with Theorem 1. ■

The same result can be shown by considering the average number of recovery operations. For a fixed number of devices, an increase in c causes an increase in recovery activity. When the average recovery activity is higher than afforded by the devices, data is lost.

Theorem 3: Let the n devices contain ρnc blocks of user data. The average number of recovery IOs per storage device per time unit is at least

$$\rho c(1-o(1)). \quad (24)$$

Proof: Fix $\rho' \lneq \rho$. Let $t' = \frac{(1-\rho')\cdot n}{\lambda}$. By the Poisson approximation, at time t', the number of original devices which have not failed is $\rho' n(1+o(1))$. As shown in Subsection II-A, the effective volume of the original devices at t' is only

$$\langle X(i)\rangle_{i=t'}^{t'+1+\frac{c}{r}} \leq \rho' nc(1+o(1)). \quad (25)$$

It follows that up to time t', the average number of recovery IOs per original storage device per time unit is at least

$$\frac{(\rho-\rho')cn(1-o(1))}{n} \stackrel{\rho'\to 0}{\longrightarrow} \rho c(1-o(1)). \quad (26)$$

■

C. Disjoint Coding Groups and Increase in Update IOs

For performance reasons, the blocks in a storage system are usually not coded en-mass in a single coding group. Rather, the blocks are partitioned into smaller groups, and the blocks of each group are protected by some code (*e.g.*, in a mirroring scheme, the blocks are partitioned into groups of size 2). In Subsection II-B we showed the necessity of an increase in n as c_{us} increases. In this subsection, we show that there are no k and k', with the property that as n increases, the data can be protected with high system-MTTDL, in coding groups each of size k and resiliency k'.

Theorem 4: Let S be a system in which the data is protected in the following manner. The data is divided into (at most $\frac{cn}{k}$) groups, each of size k and resiliency k'. Let the fraction of blocks used in S be β. Then the system MTTDL is bounded by

$$t_d\left(1+\frac{1}{1-p_r^{\frac{\beta n-\frac{k}{c-1}}{k(k-1)}}}\right) \stackrel{n\to\infty}{\longrightarrow} t_d, \quad (27)$$

where

$$p_r = 1-\sum_{i=0}^{k'}\binom{k}{i}\left(1-e^{-\lambda t_d}\right)^i\left(e^{-\lambda t_d}\right)^{k-i}. \quad (28)$$

Proof: Let $S = S_1 \dot{\bigcup} \ldots \dot{\bigcup} S_m$ be a system be composed of m sub-systems, each containing k devices. Let the data in each subs-system be protected by coding groups each of size k and resiliency k'. In a period of failure detection t_d, it is readily seen that the probability of all groups not suffering data loss is a Bernoulli process with m trials, and so the MTTDL of S' is

$$t_d\left(1+\frac{1}{1-p_r^m}\right). \quad (29)$$

Now consider the system S in the theorem. We show that this system inherently contains a subsystem similar to S'. Let S'_1 be a set of k devices containing at least one coding group in its entirely. In $S\setminus S'_1$, we consider the member of any coding group with members in S'_1 to be protected (which clearly does not decrease the system MTTDL). There still remain $\beta nc-(c-1)k(k-1)$ unprotected blocks in $S\setminus S'_1$. Let S'_2 be a set of k devices from $S\setminus S'_1$, containing at least one unprotected coding group in its entirely, and so forth. It is possible to continue to create the group S'_i as long as

$$\beta nc-i(c-1)k(k-1) \geq k. \quad (30)$$

It follows that there are at least

$$m = \frac{\beta n-\frac{k}{c-1}}{k(k-1)} \quad (31)$$

such groups. Inserting into (29) we prove the theorem. ■

It was observed in [57] that groups with resiliency k' require k' updates per write; if this were not the case, a write,

followed by the $k'' \lneq k'$ updates, would be susceptible to $k''+1$ failures. Combining this with Theorem 4, we obtain the following.

Theorem 5: For any fixed k', ρ and large enough n, the number of IO updates per modification is larger than k'.

III. Data Transition between Levels

A hierarchical reliability scheme composed of increasing-sized levels requires a data transition policy. Ideally, the high performance (and smaller) levels should contain the data required by the users. The challenges here are similar to those of hierarchical-memory data-caching, but are complicated by what we will show are time changing costs.

In this section we show a data transition policy. The setting we assume is relatively restricted. We consider a two-level system, in which data are accessed only via the higher level. We consider policies in which blocks are transferred in entire-object granularity, only entire-object modification takes place, and objects are evicted to a lower level consecutively and to uncorrelated lower-level placements. This corresponds to a system in which inter-object correlation is total, and cross-object correlation is non-existent. We defer a fuller solution to future work.

The section is organized as follows. In Subsection III-A we define precisely the transition costs. In Subsection III-B we show a competitive migration policy. In Subsection III-C we analyze some common solutions in use.

A. Performance Costs in Hierarchy levels

We first precisely define the transition costs and comparison measures.

We consider two levels composed of sets of blocks, S_0 and S_1. They are the higher and lower levels, respectively. The size S_0 is $|S_0| = k$; the size of S_1 obeys $|S_1| \gg |S_0|$. The two levels contain objects. We denote the set of objects they hold by L_0 and L_1, respectively. In general, $L_0 \bigcap L_1 \neq \emptyset$.

The policy handles a sequence of M requests $\underline{\rho} = [\rho_1, \ldots, \rho_M]$. Each request ρ_j is a pair (e_j, t_j). The entry e_j identifies the pertinent object. The entry t_j is either R or W, depending on whether the request is of type read or type write.

The cost of each operation depends on objects' locations and modification state. The cost incurred by a request (e_j, R) is 0 if $e_j \in L_0$; otherwise, the cost is denoted by $f^R_{e_j}$. If an unmodified object e is deleted from L_0, then the deletion cost is 0; otherwise the cost is denoted by f^W_e. The number of blocks required by an unmodified $e \in L_0$ is denoted by $\left|e^R\right|$. If the object is modified, an additional $\left|e^W\right|$ blocks of redundancy are required. We deal with fixed rate codes, and so for any two objects e_i and e_j of the same priority,

$$\frac{\left|e^R_i\right|}{\left|e^R_i\right| + \left|e^W_i\right|} = \frac{\left|e^R_j\right|}{\left|e^R_j\right| + \left|e^W_j\right|}, \tag{32}$$

regardless of the sizes of e_i and e_j.

The data transition problem above has much similarity to the problem of memory-hierarchy online paging.

It is well known [33] that in such settings, absolute performance measures for an algorithm are meaningless. We briefly review two meaningful comparison measures.

Let $\mathcal{A}$ be an *online* paging algorithm, *i.e.*, its response to $\underline{\rho}[j]$ does *not* depend on $\underline{\rho}[j']$ for any $j' \gneq j$. Let $\mathcal{A}(k)$ denote an instance of it for which $|L_0| = k$. *E.g.*, $\mathcal{A}$ is the *LRU* (least recently used) algorithm, and $\mathcal{A}(k)$ is LRU maintaining k items. We denote the cost incurred by $\mathcal{A}(k)$ on $\underline{\rho}$ by $f^{\mathcal{A}(k)}\left(\underline{\rho}\right)$. To assess how relatively good is $f^{\mathcal{A}(k)}\left(\underline{\rho}\right)$, we require the following two costs [34]:

- The off-line cost- let $\mathcal{O}$ denote the optimal *off-line* algorithm (*i.e.*, with advance knowledge of $\underline{\rho}$). Let $\mathcal{O}(h)$ denote its instance when $|L_0| = h$ ($h \leq k$). The cost incurred by this instance due to $\underline{\rho}$, is the off-line cost, $f^{\mathcal{O}(h)}\left(\underline{\rho}\right)$.
- The un-cached cost- let $f_{\underline{\rho}[j]=(e_j,t_j)}$ denote the *un-cached* cost of the jth operation, *i.e.*,

$$f_{\underline{\rho}[j]} = \begin{cases} f_{e^R_j} & , \quad t_j = R \\ f_{e^W_j} & , \quad t_j = W \end{cases} . \tag{33}$$

The un-cached cost of the sequence is $\sum_{i=1}^{M} f_{\underline{\rho}[j]}$.

The following definition [28] defines the competitiveness of an on-line algorithm relative to an optimal off-line algorithm.

Definition 2: An algorithm $\mathcal{A}$ is $\alpha = \alpha(h,k)$ competitive, if there is a constant $\gamma = \gamma(h,k)$, such that for any request sequence $\underline{\rho}$,

$$\mathbf{E}\left[f^{\mathcal{A}(k)}\left(\underline{\rho}\right)\right] \leq \alpha \cdot f^{\mathcal{O}(h)}\left(\underline{\rho}\right) + \gamma. \tag{34}$$

The competitiveness coefficient of $\mathcal{A}$, $\alpha_{\mathcal{A},h,k}$, is the infinimum of any such α which satisfies (34), *i.e.*,

$$\alpha_{\mathcal{A},h,k} = \inf_{\alpha} \exists_{\gamma=\gamma(h,k)} \forall_{\underline{\rho}} \mathbf{E}\left[f^{\mathcal{A}(k)}\left(\underline{\rho}\right)\right] \leq \alpha \cdot f^{\mathcal{O}(h)}\left(\underline{\rho}\right) + \gamma. \tag{35}$$

In the above, our definition differs from that of [34], by the additive γ element.

Subsequently, a modified definition of competitiveness, *loose competitiveness* [29], [34], was created. The new version has advantages in its not allowing $\underline{\rho}$ to be too closely tailored to k, and limiting the effect of any $\underline{\rho}$ for which the absolute cost is too low. The following is a modified version of loose competitiveness.

Definition 3: An algorithm $\mathcal{A}$ is $\hat{\alpha} = \hat{\alpha}(\epsilon, \delta, k)$-loosely-competitive, if there is a constant $\gamma = \gamma(k)$, such that for any request sequence $\underline{\rho}$, at least $(1-\delta)k$ of the values $k' \in [k]$ satisfy

$$\mathbf{E}\left[f^{\mathcal{A}(k')}\left(\underline{\rho}\right)\right] \leq \max\left\{\alpha \cdot f^{\mathcal{O}(k')}\left(\underline{\rho}\right), \epsilon \cdot \sum_{\rho \in \underline{\rho}} f\left(\rho\right)\right\} + \gamma. \tag{36}$$

The (ϵ,δ,k)-loose-competitiveness coefficient of $\mathcal{A}$, $\hat{\alpha}_{\mathcal{A},\epsilon,\delta,k}$, is the infinimum of any such $\hat{\alpha}$ which satisfies (36), *i.e.*,

$$\hat{\alpha}_{\mathcal{A},\epsilon,\delta,k} = \inf_{\alpha} \exists_{\gamma=\gamma(k)} \forall_{\underline{\rho}} \exists_{K'\in[(]k),|(|K')\geq(1-\delta)\cdot k} k' \subseteq K' \Rightarrow \mathbf{E}\left[f^{\mathcal{A}(k')}\left(\underline{\rho}\right)\right] \leq \max\left\{\alpha\cdot f^{\mathcal{O}(k')}\left(\underline{\rho}\right), \epsilon\cdot\sum_{\rho\in\underline{\rho}} f\left(\rho\right)\right\} + \gamma. \quad (37)$$

Definition 4: An algorithm $\mathcal{A}$ is (ϵ,δ)-loosely $\tilde{\alpha}$-competitive, if for any k, except for a finite number of ks, $\mathcal{A}$ is $\hat{\alpha}=\hat{\alpha}(\epsilon,\delta,k)$-loosely competitive, for $\hat{\alpha}\leq\tilde{\alpha}$.

In the above, our definition differs from that of [34], by requiring an algorithm to be $\hat{\alpha}(\epsilon,\delta,k)$-loosely competitive for *almost all* k.

Competitive paging algorithms have previously been studied for the case of multi-level memory hierarchies. This case differs from ours. In a memory hierarchy, the size an object requiring being a constant; in our case modified objects require redundancy. In a memory hierarchy, the cost is, in general, dominated by object retrieval; in our case, modified objects incur an eviction cost, while unmodified objects do not. This is aggravated by UEP. For the memory hierarchy case, algorithms for uniform-size uniform-cost objects were studied [27], [28], [30], algorithms for uniform-size arbitrary-cost objects were studied [32], [30], and algorithms for arbitrary-size and differing-costs were studied [60], [31], [34].

B. A Competitive Algorithm

In this subsection we describe *M-Landlord*, which is a modification of an algorithm from [34]. The act of writing modified data to a lower level is known as *destage*; the act of deleting unmodified from a higher level is known as *demote*. The algorithm works performing a continuing series of destage and demote operations, based on a space-per-cost object assessment. For brevity, we will refer in equations to this algorithm as $\mathcal{LL}^{\mathcal{M}}$.

The main result we prove is the following.

Theorem 6: Fix ϵ and δ, and let $\delta'\gtrsim 0$ be an arbitrarily small constant.
1. $\mathcal{LL}^{\mathcal{M}}$ is $(\epsilon,\delta+\delta')$-loosely $\frac{1-\ln(\epsilon)}{\delta}e$-competitive.
2. $\mathcal{LL}^{\mathcal{M}}$ has computational complexity $O\left(1\right)$ for operations which do not access the lower level, and computational complexity $O\left(\log(k)\right)$, for operations which do.

The subsection is organized as follows. In Sub-subsection III-B.1 we describe the algorithm. In Sub-subsection III-B.2 we study a related hierarchical-memory algorithm, in order to analyze the $\mathcal{LL}^{\mathcal{M}}$ competitiveness. In Sub-subsection III-B.3 we prove the loose-competitiveness and computational-complexity properties of $\mathcal{LL}^{\mathcal{M}}$.

B.1 M-Landlord

In this sub-subsection we describe the algorithm $\mathcal{LL}^{\mathcal{M}}$. This is a modification of the Landlord algorithm [34], extended to the case where some objects are modified, and with lower computational complexity (at a cost to the generality of the original algorithm). Essentially, the algorithm maintains a space-per-cost estimate of each object. Based on this ratio, it decides on the next destage or demote operation used for freeing space. Algorithms 1 and 2 show the algorithm in pseudo-code.

The algorithm maintains the following global variable and array:

$$\begin{aligned} LL &= \text{set of objects in } L_0, \\ c &= \text{a real value describing ``credit history''}. \end{aligned} \quad (38)$$

The algorithm maintains the following object-specific arrays and heap-based PQ (priority queues) [58]. Let $e\in LL$ be a an object, then:

$$\begin{aligned} \underline{m} &= \text{an array s.t. } \underline{m}(e)=\mathbf{T}\Leftrightarrow \text{e is modified}, \\ \underline{c} &= \text{an array s.t. } \underline{c}(e)=\text{object credit},, \\ H^R &= \text{a PQ s.t. } \underline{m}[e]=\mathbf{F}\Leftrightarrow e\in H^r, \\ H_i^W &= \text{a PQ s.t. } \underline{m}[e]=\mathbf{T}\bigwedge \mu\left(e\right)=i\Leftrightarrow e\in H_i^W. \end{aligned} \quad (39)$$

An object e is ordered within its PQ by $\underline{c}[e]$. Initially, $c=0$.

Algorithm 1 shows a high-level description of the algorithm. Let e_j be the object accessed at step j. If e_j need not be retrieved and is not newly modified, no state updates need be done (lines 1 to 5). Space is allocated (by calling Algorithm 2), and the object's queue membership is updated (lines 6 to 11). The object is retrieved if it is not already in place (lines 12 to 20). A modification to the object results in updating the internal data structures (lines 21 to 26).

In all cases, the object's credit, $\underline{c}[e_j]$ is updated according to the credit history c and the operation type t_j. The object's cost-per-space is efficiently stored relative to objects in its class, by means of the appropriate PQ.

The algorithm maintains the following invariants on objects' credit,

Invariant 1:

$$\forall_{e\in LL}\left(\underline{c}\left[e\right]\leq f_e^R\right)\bigvee\left(\underline{m}\left[e\right]=\mathbf{T}\bigwedge\underline{c}\left[e\right]\leq f_e^W\right), \quad \forall_{e\in LL}\underline{c}\left[e\right]\geq 0, \quad (40)$$

and the following invariants on object PQs' location,

Invariant 2:

$$\forall_{e\in LL}\forall_{i\in[d]} e\in H^R\Rightarrow e\notin H_i^W, \quad (41)$$

$\forall_{e\in LL}\forall_{i\in[d]} e\in H_i^W\Rightarrow e\notin H^R\bigwedge e\notin H_j^w\ (j\neq i)$.

Algorithm 2 shows how space is cleared. First, the space needed for eviction is calculated (lines 1 to 6). The algorithm loops until at least that amount has been evicted (lines 7 to 31). First, the credit history is updated (lines 9 to 16). By comparing an objects' credit to the credit history, some objects are possibly demoted (lines 22 to 21), and for each of the priorities, some are possibly destaged (lines 22 to 30).

Algorithm 1 M-Landlord($\underline{\rho}[j]$)

Handles a request $\underline{\rho}[j] = (e_j, t_j)$.

/ Check if $\underline{\rho}[j]$ does not modify objects' state.*/*
if $e_j \in LL \bigwedge (t_j = W \Rightarrow \underline{m}\,[e_j] = \mathbf{T})$ **then**
  Access e_j
  return
end if
/ Evict space needed */*
M-Landlord-Evict($\underline{\rho}[j]$)
/ Remove modified object from read PQ.*/*
if $e_j \in LL \bigwedge t_j = W$ **then**
  PQ-Remove(H^R, e_j)
end if
/ Check if object should be retrieved.*/*
if $e_j \notin LL$ **then**
  Retrieve e_j
  $LL \leftarrow LL \bigcup \{e_j\}$
  if $t_j = R$ **then**
    $\underline{c}\,[e_j] \leftarrow c + \frac{f^R_{e_j}}{|e^R_j|}$
    PQ-Insert(H^R, e_j)
  end if
end if
/ Check if object must be marked as modified.*/*
if $t_j = W$ **then**
  $\underline{c}\,[e_j] \leftarrow c\left(1 + \frac{|e^R_j|}{|e^W_j|}\right) + \frac{f^W_{e_j}}{|e^W_j|}$
  $\underline{m}\,[e_j] \leftarrow \mathbf{T}$
  PQ-Insert($H^W_{\mu(e_j)}, e_j$)
end if
Access e_j

Algorithm 2 M-Landlord-Evict($\underline{\rho}[j]$)

Clears space for a request $\underline{\rho}[j] = (e_j, t_j)$.

Require: $e_j \notin LL \bigvee (t_j = W \bigwedge \underline{m}\,[e_j] = \mathbf{F})$

/ s indicates the space which need be cleared.*/*
$s \leftarrow (t_j = R)?\,|e^R_j| : |e^W_j|$
/ Check if modifying an unmodified existing object.*/*
if $t_j = W \bigwedge e_j \in LL \bigwedge \underline{m}[e_J] = \mathbf{F}$ **then**
  $s \leftarrow s - |e^R_j|$
end if
/ Loop until enough space has been cleared.*/*
while $|LL| \geq k - s$ **do**
  / $\delta^R, \delta^W_1, \ldots, \delta^W_d$ = credit-history changes.*/*
  $\delta^R \leftarrow \underline{c}\left[\text{PQ-Min}(H^R)\right] - c$
  for $i \in [d]$ **do**
    $\underline{\delta^W}[i] \leftarrow \frac{\underline{c}[\text{PQ-Min}_{(H^W_i)}]}{\left(1 + \frac{|e^R_i|}{|e^W_i|}\right)} - c$
  end for
  $\delta \leftarrow \min\left\{\delta^R, \underline{\delta^W}[1], \ldots, \underline{\delta^W}[d]\right\}$
  / Update the credit history.*/*
  $c \leftarrow c + \delta$
  / Demote some objects.*/*
  while $\underline{c}\left[e = \text{PQ-Min}(H^R)\right] = c$ **do**
    $LL \leftarrow LL \setminus \{e\}$
    PQ-Remove(H^R)
  end while
  / Destage some objects.*/*
  for $i \in [d]$ **do**
    while $\underline{c}\left[e = \text{PQ-Min}(H^W_i)\right] = c\left(1 + \frac{|e^R|}{|e^W|}\right)$ **do**
      PQ-Remove(H^W_i)
      $\underline{m}\,[e] = \mathbf{F}$
      $\underline{c}\,[e] \leftarrow c + \frac{f_{e^R}}{|e^R|}$
      PQ-Insert(H^R, e)
    end while
  end for
end while

B.2 RW-Landlord

In this subsection we describe the algorithm RW-Landlord, which we will use for the competitiveness analysis of M-Landlord. It is difficult to analyze M-Landlord directly, because of the time-varying costs and sizes of objects. For brevity in mathematical expressions, we will interchange RW-Landlord with $\mathcal{LL}^{\mathcal{RW}}$.

RW-Landlord operates in a standard caching-problem setting (*i.e.*, without object modification). We relate the setting of RW-Landlord to that of M-Landlord as follows. For any object e in M-Landlord, we consider a *read object* e^R, and a *write object* e^W. The retrieval cost of e^R is f_{e^R}, the retrieval cost of e; the retrieval cost of e^W is f_{e^W}, the eviction cost of e. The size of e^R is the size of e; the size of e^W is the size of the redundancy of e. We conceptually partition LL, the cache of $\mathcal{LL}^{\mathcal{RW}}$, into sub-caches LL^R and LL^W, containing read and write objects respectively. *I.e.*, $LL = LL^R \bigcup LL^W$, where

$$\begin{aligned} e^R \in LL \quad &\Rightarrow \quad e^R \in LL^R, \\ e^W \in LL \quad &\Rightarrow \quad e^W \in LL^W. \end{aligned} \tag{42}$$

When studying the competitiveness of RW-Landlord, we similarly partition the cache of $\mathcal{O}$ to $O = O^R \bigcup O^W$ as well.

We differentiate the setting from that of a classical caching problem, by requiring the following invariants on object containment:

Invariant 3:

$$\begin{aligned} \forall_{e^W} e^W \in LL^W \quad &\Rightarrow \quad e^R \in LL^R \\ \forall_{e^W} e^W \in O^W \quad &\Rightarrow \quad e^R \in O^R, \end{aligned} \tag{43}$$

and the requirement that servicing a write request entails that both read object and write objects are in the cache, *i.e.*,

Invariant 4:

$$\underline{\rho}[j] = (e_j, W) \Rightarrow \left(e^R_j \in LL^R \bigwedge e^W_j \in LL^W\right). \tag{44}$$

Specifically, we must take care that the algorithm does not evict a read object before the corresponding write object, and that a read object is not evicted in order to make place for its corresponding write object. The former would be equivalent to evicting an object while retaining its redundancy. The latter would be equivalent to evicting an object in order to make place for its redundancy.

We will show that RW-Landlord has the corresponding invariants to that of M-Landlord's Invariant 1:

Invariant 5:

$$\begin{aligned} \forall_{e^R \in LL^R} \underline{c}\left[e^R\right] &\leq f_{e^R}, \\ \forall_{e^W \in LL^W} \underline{c}\left[e^W\right] &\leq f_{e^W}, \\ \forall_{e \in LL} \underline{c}[e] &\geq 0 \end{aligned} \tag{45}$$

Algorithm 3 shows a high level description of the algorithm. In structure, it is quite similar to that of Algorithm 1. There are some differences. Note that a request $\underline{\rho}[j]$ is now explicitly to e_j^R or to e_j^W, instead of specifying an object and an access type. The credit of each object is maintained directly, instead of the use of the credit history variable c in Algorithm 1. Access to objects is done via Algorithm 4, to make explicit Invariance 4.

Algorithm 3 RW-Landlord($\underline{\rho}[j]$)

Handles a request $\underline{\rho}[j] = e_j^{t_j}$.

1: */* Check if $\underline{\rho}[j]$ does not modify objects' state. */*
2: **if** $e_j^R \in LL^R \bigwedge \left(t_j = W \Rightarrow e_j^W \in LL^W\right)$ **then**
3: RW-Landlord-Service-Request($\underline{\rho}[j]$)
4: **return**
5: **end if**
6: */* Evict space needed */*
7: RW-Landlord-Evict($\underline{\rho}[j]$)
8: */* Check if read object must be retrieved.*/*
9: **if** $e_j^R \notin LL^R$ **then**
10: Retrieve e_j^R
11: $LL^R \leftarrow LL^R \dot{\bigcup} \left\{e_j^R\right\}$
12: $\underline{c}\left[e_j^R\right] \leftarrow f_{e_j^R}$
13: **end if**
14: */* Check if write object must be retrieved.*/*
15: **if** $t_j = W$ **then**
16: Retrieve e_j^W
17: $LL^W \leftarrow LL^W \dot{\bigcup} \left\{e_j^W\right\}$
18: $\underline{c}\left[e_j^R\right] \leftarrow f_{e_j^R}$
19: $\underline{c}\left[e_j^W\right] \leftarrow f_{e_j^W}$
20: **end if**
21: RW-Landlord-Service-Request($\underline{\rho}[j]$)

Algorithm 5 shows how space is cleared for a request. First, the space needed for eviction is calculated (lines 1 to 6). The algorithm loops until at least that amount has been evicted (lines 8 to 31), while maintaining Invariant 3. In each iteration, credit is decreased from all objects (line 11). Objects whose credit is 0, are evicted (line 28).

Algorithm 4 RW-Landlord-Service-Request($\underline{\rho}[j]$)

Handles an existing-object request $\underline{\rho}[j] = e_j^{t_j}$.

Require: $e_j^R \in LL^R \bigwedge \left(t_j = W \Rightarrow e_j^W \in LL^W\right)$
1: */* For reads, access and update a single object.*/*
2: **if** $(t_j = R)$ **then**
3: access e_j^R
4: */* For write, access and update two objects.*/*
5: **else**
6: access e_j^R and e_j^W
7: **end if**

To ensure that invariant 5 is maintained, credit transference is used. When clearing space for a write object, the credit of the corresponding read object is raised to the maximum (line 14). In addition, all write objects transfer credit to their read objects (lines 16 to 23).

We now analyze the competitiveness of the algorithm.

Theorem 7: $\mathcal{LL}^{\mathcal{RW}}$ is $\frac{k}{h-k+1}$ competitive.

To analyze $\mathcal{LL}^{\mathcal{RW}}$, we use a potential function from [34].

Definition 5: Define the potential function

$$\begin{aligned} \Phi = & \\ & (h-1) \\ & \sum_{e'^R \in LL^R} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , \quad e'^W \in LL^W \\ \underline{c}\left[e'^R\right] & , \quad e'^W \notin LL^W \end{cases} + \\ & k \sum_{e'^R \in O^R} \begin{cases} f_{e'^R} + f_{e'^W} & , \quad e'^W \in O^W \\ f_{e'^R} & , \quad e'^W \notin O^W \end{cases} - \\ & k \sum_{e'^R \in O^R} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , \quad e'^W \in O^W \\ \underline{c}\left[e'^R\right] & , \quad e'^W \notin O^W \end{cases} \end{aligned} \tag{46}$$

The following lemma proves Theorem 7.

Lemma 3: Following any series of actions by $\mathcal{LL}^{\mathcal{RW}}$ and $\mathcal{O}$, $\Phi \geq 0$. The following four operations affect Φ:

- $\mathcal{O}$ retrieves e^R: Φ increases by at most kf_{e^R}.
- $\mathcal{O}$ retrieves e^W: Φ increases by at most kf_{e^W}.
- $\mathcal{LL}^{\mathcal{RW}}$ retrieves e^R: Φ decreases by at least $(k-h+1)f_{e^R}$.
- $\mathcal{LL}^{\mathcal{RW}}$ retrieves e^W: Φ decreases by at least $(k-h+1)f_{e^W}$.

No other action by $\mathcal{O}$ or $\mathcal{LL}^{\mathcal{RW}}$ increases Φ.

The proof of Lemma 3 is similar in many points to that in [34]. We focus mainly on the points in which it differs due to Invariants 3, 4, and 5.

Proof: We analyze the effect of the steps taken by $\mathcal{O}$ and $\mathcal{LL}^{\mathcal{RW}}$ on Φ.

- $\mathcal{O}$ evicts $e \in O$: Since Invariant 5 is maintained (see Algorithm 3), Φ cannot increase.
- $\mathcal{O}$ retrieves $e \in O$: In this case $\mathcal{O}$ pays f_e. Since Invariant 5 ($\forall_{e \in LL} \underline{c}[e] \geq 0$) is maintained (see Algorithm 3), then Φ increases by at most kf_e.
- $\mathcal{LL}^{\mathcal{RW}}$ transfers credit from $\underline{c}\left[e^W\right]$ to $\underline{c}\left[e^R\right]$ (lines 21 and 20): In this case $e_j^R \in LL^R$ and $e_j^W \in LL^W$. Rewriting Φ,

Algorithm 5 RW-Landlord-Evict($\underline{\rho}[j]$)

Clears space for a request $\underline{\rho}[j] = e_j^{t_j}$.

Require: $e_j^R \notin LL^R \bigvee \left(t_j = W \bigwedge e_j^W \notin LL^W\right)$

1: /* s indicates the space which need be cleared.*/
2: $s \leftarrow (t_j = R)? \left|e_j^R\right| : \left|e_j^W\right|$
3: **if** $t_j = W \bigwedge e_j^R \notin LL^R$ **then**
4: /* Increment s if a read object is needed.*/
5: $s \leftarrow s + \left|e_j^R\right|$
6: **end if**
7: /* Evict space while maintaining Invariant 3.*/
8: **while** $|LL| \geq k - s$ **do**
9: $\delta \leftarrow \min\left\{\min_{e^R \in LL^R} \frac{\underline{c}[e^R]}{|e^R|}, \min_{e^W \in LL^W} \frac{\underline{c}[e^W]}{|e^R|+|e^W|}\right\}$
10: **for** $e \in LL$ **do**
11: $\underline{c}[e] \leftarrow \underline{c}[e] - \delta|e|$
12: **if** $t_j = W \bigwedge e_j^R \in LL^R$ **then**
13: /* Update corresponding read-object, if in cache.*/
14: $\underline{c}\left[e_j^R\right] \leftarrow f_{e_j^R}$
15: **end if**
16: **for** $e^R \in LL^R$ **do**
17: **if** $e^W \in LL^W$ **then**
18: /* Transfer credit from write objects to read objects.*/
19: $\delta' \leftarrow \min\left\{\delta\left|e^R\right|, \delta\left|e^W\right|\right\}$
20: $\underline{c}\left[e^R\right] \leftarrow \underline{c}\left[e^R\right] + \delta'$
21: $\underline{c}\left[e^W\right] \leftarrow \underline{c}\left[e^W\right] - \delta'$
22: **end if**
23: **end for**
24: **end for**
25: **for** $e \in LL$ **do**
26: **if** $\underline{c}[e] = 0$ **then**
27: /* Evict objects with 0 credit.*/
28: $LL \leftarrow LL \setminus \{e\}$
29: **end if**
30: **end for**
31: **end while**

we have

$$\begin{aligned}\Phi = & \qquad (47)\\ & (h-1)\left(\underline{c}\left[e^R\right] + \underline{c}\left[e^W\right]\right) + \\ & (h-1)\sum_{e'^R \in LL^R \setminus \{e^R\}} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , e'^W \in LL^W \\ \underline{c}\left[e'^R\right] & , e'^W \notin LL^W \end{cases} + \\ & k \begin{cases} f_{e^R} + f_{e^W} - \left(\underline{c}\left[e^R\right] + \underline{c}\left[e^W\right]\right) & , \ e^W \in O^W \\ f_{e^R} - \underline{c}\left[e^R\right] & , \ e^W \notin O^W, \ e^R \in O^R \\ 0 & , \ e^R \notin O^R \end{cases} + \\ & k \sum_{e'^R \in O^R \setminus \{e^R\}} \begin{cases} f_{e'^R} + f_{e'^W} & , \ e'^W \in O^W \\ f_{e'^R} & , \ e'^W \notin O^W \end{cases} - \\ & k \sum_{e'^R \in O^R \setminus \{e^R\}} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , \ e'^W \in O^W \\ \underline{c}\left[e'^R\right] & , \ e'^W \notin O^W \end{cases}, \end{aligned}$$

which shows that Φ cannot increase.

- $\mathcal{LL}^{\mathcal{RW}}$ increments $\underline{c}\left[e^R\right]$ to f_{e^R} (line 14): We can assume in this case that $e^R \in O^R$, and therefore the increase to Φ is $(f_{e^R} - \underline{c}\left[e^R\right])(h-1-k) \leq 0$.
- $\mathcal{LL}^{\mathcal{RW}}$ decreases uniformly $\underline{c}[e']$ for all $e' \in LL$ (line 11): This occurs in one of three cases: either there is an $e^R \in O^R \setminus LL^R$, or there is an $e^W \in O^W \setminus LL^W$, or both of the previous. We prove the first case (which is the only case in [34]). The other two cases are similar. Letting $O'^R = O^R \bigcap LL^R$, and, $O'^W = O^W \bigcap LL^W$ we have

$$\begin{aligned}\Phi \overset{(a)}{=} & \qquad (48)\\ & (h-1)\sum_{e'^R \in LL^R} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , e'^W \in LL^W \\ \underline{c}\left[e'^R\right] & , e'^W \notin LL^W \end{cases} + \\ & k \sum_{e'^R \in O^R} \begin{cases} f_{e'^R} + f_{e'^W} & , \ e'^W \in O^W \\ f_{e'^R} & , \ e'^W \notin O^W \end{cases} - \\ & k \sum_{e'^R \in O'^R} \begin{cases} \underline{c}\left[e'^R\right] + \underline{c}\left[e'^W\right] & , \ e'^W \in O'^W \\ \underline{c}\left[e'^R\right] & , \ e'^W \notin O'^W \end{cases}. \end{aligned}$$

(where (a) follows from the fact that for $e' \notin LL$, $\underline{c}[e'] = 0$. Since $e^R \in O \setminus LL$, $|O \bigcap LL| \leq h - \left|e^R\right|$. Since there is no room for e^R in LL, $|LL| \geq k - \left|e^R\right| + 1$. It follows that the decrease to Φ is at least $(h-1)(k-\left|e^R\right|+1) - k(h-\left|e^R\right|) = (1-\left|e^R\right|)(h-1-k) \geq 0$.

- $\mathcal{LL}^{\mathcal{RW}}$ evicts an object e (line 28): Since this happens when $\underline{c}[e] = 0$ (line 26), Φ does not change.
- $\mathcal{LL}^{\mathcal{RW}}$ retrieves e^R (line 11) and sets its credit to $\underline{c}\left[e^R\right]$ (line 12): In this case, $\mathcal{LL}^{\mathcal{RW}}$ pays a cost of f_{e^R}. Since the retrieval is done in response to a request for e^R or e^W, we can assume that $e_j^R \in O^R$. The increment to the credit therefore increments Φ by $(h-1+k)f_e$, or equivalently, decrease it by $(k-h+1)f_e$.
- $\mathcal{LL}^{\mathcal{RW}}$ retrieves e^W (line 17) and sets its credit to $\underline{c}\left[e^W\right]$ (line 19): This is similar to the previous case.

■

B.3 Loose Competitiveness of M-Landlord

The following lemma shows that $\mathcal{LL}^{\mathcal{M}}$ and $\mathcal{LL}^{\mathcal{RW}}$ work similarly.

Lemma 4: Assume at the starting point $\mathcal{LL}^{\mathcal{M}}$ is operating on an empty level, and $\mathcal{LL}^{\mathcal{RW}}$ is operating on an empty cache. Let $\underline{\rho}^{\mathcal{M}}$ and $\underline{\rho}^{\mathcal{RW}}$ be request sequences to $\mathcal{LL}^{\mathcal{M}}$ and $\mathcal{LL}^{\mathcal{RW}}$, respectively, s.t. at any request $j \in [M]$,

$$\underline{\rho}^{\mathcal{M}}[j] = (e_j, t_j) \Leftrightarrow \underline{\rho}^{\mathcal{RW}}[j] = e_j^{t_j} \qquad (49)$$

Then following the request,

- $\mathcal{LL}^{\mathcal{M}}$ contains an unmodified object $e \leftrightarrow \mathcal{LL}^{\mathcal{RW}}$ contains e^R and does not contain e^W.
- $\mathcal{LL}^{\mathcal{M}}$ contains a modified object $e \Leftrightarrow \mathcal{LL}^{\mathcal{RW}}$ contains both e^R and e^W.
- $\mathcal{LL}^{\mathcal{M}}$ does not contain an object $e \Leftrightarrow \mathcal{LL}^{\mathcal{RW}}$ contains neither e^R nor e^W.

Proof: We first show the relation between credit variables maintained by the algorithms $\mathcal{LL}^{\mathcal{M}}$ and $\mathcal{LL}^{\mathcal{RW}}$.

If LL contains modified e_j, then

$$\underline{c}\left[e_j^W\right] = \left(\underline{c}\left[e_j\right] - c\left(1 + \frac{\left|e_j^R\right|}{\left|e_j^W\right|}\right)\right)\left|e_j^W\right| \tag{50}$$

and

$$\underline{c}\left[e_j^R\right] = f_{e_j^R}. \tag{51}$$

If LL contains unmodified e_j, then

$$\underline{c}\left[e_j^R\right] = \left(\underline{c}\left[e_j\right] - c\right)\left|e_j^R\right|. \tag{52}$$

and there is no e_j^W object.

In other words, the actual credit of an element is given by (50) if it is modified and by (52) if it is unmodified.

if e_j is modified then (51) clearly holds, since its read and write objects exist, and hence whenever $\underline{c}\left[e_j^R\right]$ is decreased in line 11 of Algorithm RW-Landlord-Evict, it is increased back to the initial value in lines 16-21.

Initially (when an element is inserted to priority queues in Algorithm $\mathcal{LL}^{\mathcal{M}}$) (50) and (52) are clearly true (see lines 18 and 25 in Algorithm M-Landlord and line 28 in Algorithm M-Landlord-Evict).

The value of element's credit is effectively decreased in line 16 in Algorithm M-Landlord-Evict (by increasing the counter). If e_j is unmodified, the credit of e_j^R is decreased by $\delta\left|e_j^R\right|$ in line 11 in Algorithm RW-Landlord-Evict, while c is decreased by δ in line 16 in Algorithm M-Landlord-Evict. Hence, Equation 52 still holds. If e_j is modified, the credit of e_j^W is decreased by $\delta\left|e_j^W\right|$ in line 11 and additionally by $\delta\left|e_j^R\right|$ in line 21 in Algorithm RW-Landlord-Evict, while c is increased by δ in line 16 in Algorithm M-Landlord-Evict. Hence, Equation 50 still holds.

Finally, we note that by Equation 52, condition for removal of e_j from LL in line 20 in M-Landlord-Evict corresponds to a condition $f_{e^R} = 0$ in line 26 in RW–Landlord-Evict. Similarly, by Equation 50, condition for marking e_j as unmodified in line 26 in M-Landlord-Evict corresponds to a condition $f_{e^W} = 0$ (for removal of write object) in line 26 in RW–Landlord-Evict. ■

By Theorem 7, $\mathcal{LL}^{\mathcal{RW}}$ is $\frac{k}{k-h+1}$ competitive. We show that this implies the competitiveness of $\mathcal{LL}^{\mathcal{M}}$ as well.

Theorem 8: $\mathcal{LL}^{\mathcal{M}}$ is $\frac{k}{k-h+1}$ competitive.

Proof: Let

$$\gamma'(h,k) = \max_{e^W} f_{e^W}\frac{h}{\left|e^W\right|}. \tag{53}$$

We then have

$$\begin{aligned} &f^{\mathcal{LL}^{\mathcal{M}}(k)}\left(\underline{\rho}^{\mathcal{M}}\right) \overset{(a)}{\leq} \\ &f^{\mathcal{LL}^{\mathcal{RW}}(k)}\left(\underline{\rho}^{\mathcal{RW}}\right) \overset{(b)}{\leq} \\ &\frac{k}{k-h+1}f^{\mathcal{O}^{\mathcal{RW}}(h)}\left(\underline{\rho}^{\mathcal{RW}}\right) \overset{(c)}{\leq} \\ &\frac{k}{k-h+1}f^{\mathcal{O}^{\mathcal{M}}(h)}\left(\underline{\rho}^{\mathcal{M}}\right) + \frac{k}{k-h+1}\gamma'(h,k) \end{aligned} \tag{54}$$

where in the above, (a) follows from the fact that $\mathcal{LL}^{\mathcal{M}}$ "simulates" the actions of $\mathcal{LL}^{\mathcal{RW}}$, and so its cost is not higher than that of $\mathcal{LL}^{\mathcal{RW}}$, (b) follows from Theorem 7, and (c) follows from the fact that if an algorithm $\mathcal{O}'^{\mathcal{RW}}$ for the cache problem would "simulate" the actions of $\mathcal{O}^{\mathcal{M}}$, then

$$f^{\mathcal{O}'^{\mathcal{RW}}(k)}\left(\underline{\rho}^{\mathcal{RW}}\right) \leq f^{\mathcal{O}^{\mathcal{M}}(h)}\left(\underline{\rho}^{\mathcal{M}}\right) + \gamma'(h,k), \tag{55}$$

since for each write object retrieved, $\mathcal{O}^{\mathcal{M}}$ might decide not to flush it to the lower level, but the number of objects for which this is true is bounded by a function of h, and not of $\underline{\rho}$.

From (54) we have that $\mathcal{LL}^{\mathcal{RW}}$ is competitive according to Definition 2, with

$$\begin{aligned} \alpha(h,k) &= \frac{k}{k-h+1}, \\ \gamma(h,k) &= \frac{k}{k-h+1}\gamma'(h,k). \end{aligned} \tag{56}$$

■

The following proof of claim 1 in Theorem 6 follows [34], but differ in the definitions of loose competitive, and of the accompanying constants.

Lemma 5: $\mathcal{LL}^{\mathcal{M}}$ is (ϵ,δ)-loosely $\frac{(1-\ln(\epsilon))e}{\delta}$-competitive.

Proof: For fixed ϵ, δ, and k, let $b \gneq 0$ be a constant, and define

$$\eta = \eta(b,k,\epsilon,\delta) = \frac{k}{b}\epsilon^{-\frac{b}{\delta k - b}}, \tag{57}$$

For a fixed $\underline{\rho}$, let B be a set of s *bad* values consisting of any j for which

$$f^{\mathcal{LL}^{\mathcal{M}}(j)}\left(\underline{\rho}\right) \geq \max\left\{\alpha \cdot f^{\mathcal{O}(j)}\left(\underline{\rho}\right), \epsilon\sum_{i=1}^{M} f_{\underline{\rho}[i]}\right\} + \gamma(j). \tag{58}$$

Specifically, let $B = \{k_1, \ldots, k_s\}$. W.l.o.g., let $k_1 \leq \cdots \leq k_s \leq k$. Define the subset $B' = \{k'_1, \ldots, k'_{s'}\} \subseteq B$, by $k'_i = k_{\lceil ib \rceil}$, for $i \in \left[\left\lceil \frac{s}{\lceil b \rceil} \right\rceil\right]$. Note that for $i \in [s']$, $k'_i - k'_{i-1} \geq b$, and that $s' \geq \frac{s}{b+1} - 1$. We then have

$$\begin{aligned} &f^{\mathcal{LL}^{\mathcal{M}}(k'_i)}\left(\underline{\rho}\right) \overset{(a)}{\leq} \\ &\frac{k'_i}{k'_i - k'_{i-1} + 1}f^{\mathcal{O}(k'_{i-1})}\left(\underline{\rho}\right) \overset{(b)}{\leq} \\ &\frac{k'_i}{\eta(k'_i - k'_{i-1} + 1)} \cdot \\ &\left(f^{\mathcal{LL}^{\mathcal{M}}(k'_{i-1})}\left(\underline{\rho}\right) - \gamma(k'_i - k'_{i-1} + 1, k'_{i-1})\right) \overset{(c)}{\leq} \\ &\epsilon^{\frac{b}{\delta k - b}} f^{\mathcal{LL}^{\mathcal{M}}(k'_{i-1})}\left(\underline{\rho}\right), \end{aligned} \tag{59}$$

where, in the above, (a) follows from Theorem 8, (b) follows from the fact that k'_{i-1} is bad and (58), and (c) follows from the fact that $\frac{k'_i}{\eta(k'_i - k'_{i-1}+1)} \leq \frac{k}{\eta b}$ and (57).

Inductively,

$$f^{\mathcal{LL}^{\mathcal{M}}(k'_{s'})}\left(\underline{\rho}\right) \leq \left(\epsilon^{\frac{b}{\delta k-b}}\right)^{s'} \cdot f^{\mathcal{LL}^{\mathcal{M}}(k'_0)}\left(\underline{\rho}\right). \quad (60)$$

We also have

$$\epsilon \cdot f^{\mathcal{LL}^{\mathcal{M}}(k'_0)}\left(\underline{\rho}\right) \overset{(a)}{\leq} \epsilon \cdot \sum_{i=1}^{M} f_{\underline{\rho}[i]} \overset{(b)}{\leq} f^{\mathcal{LL}^{\mathcal{M}}(k'_{s'})}\left(\underline{\rho}\right), \quad (61)$$

where (a) follows from the fact that the cost of an algorithm cannot be larger than the sequence request, and (b) follows from (58).

It follows from (60) and (61) that $\epsilon \leq \left(\epsilon^{\frac{b}{\delta k-b}}\right)^{s'}$, from which it follows that $s' \leq \frac{\delta k-b}{b}$, and therefore

$$s \leq \delta\frac{b+1}{b}k. \quad (62)$$

From the definition of the set B, and (62), we have that $\mathcal{LL}^{\mathcal{M}}$ is $(\epsilon, \delta\frac{b+1}{b}, k)$-loosely $\frac{k}{b}\epsilon^{-\frac{b}{\delta k-b}}$-competitive, for $b \gneq 0$.

For any k, setting $b = \frac{\delta k}{1-\ln(\epsilon)}$ in (57), yields $\eta = \frac{1-\ln(\epsilon)}{\delta}e$. For all but a finite number of k,

$$\delta\frac{b+1}{b} = \delta\left(1 + \frac{1}{\frac{\delta k}{1-\ln(\epsilon)}}\right) \leq \delta + \delta'. \quad (63)$$

■

C. Comparison to Some Common Solutions

Present systems commonly employ two types of solutions for the eviction problem in Subsection III-A. In the first, *maximal protection*, all data is protected to the maximal requirement. In the second, *static partitioning* devices, or devices' blocks, are statically partitioned s.t. each partition element is dedicated to elements of a single priority. In both cases, an eviction policy for uniform priority is used (either according to the highest priority, or within each partition element). It is clear that these solutions are not efficient in terms of storage efficiency (in particular the first one). We show in this subsection that, in addition, they are not competitive in any definition in Subsection III-A.

Theorem 9: Let d be a number of priorities. Let $\mathcal{A}_m$ be any eviction algorithm based on maximal protection (while the ratio between the highest priority space consumption per one element to the lowest priority one is $\frac{\nu_d}{\nu_1}$) , and $\mathcal{A}_s$ be any eviction algorithm based on static partitioning, where either or both of $\mathcal{A}_m$ and $\mathcal{A}_s$ can be random algorithms.

1. For any h, k s.t. $h \gneq \frac{k}{d}$, $\alpha_{\mathcal{A}_s,h,k} = \infty$.
2. For any $\delta > \frac{1}{d}$, $\epsilon < 1 - \frac{1}{d\delta}$ and k, $\hat{\alpha}_(\mathcal{A}_s, \epsilon, \delta, k) = \infty$.
3. For any h, k s.t. $h \gneq \frac{k}{\frac{\nu_d}{\nu_1}}$, $\alpha_{\mathcal{A}_m,h,k} = \infty$.
4. For any $\delta > \frac{1}{d}$, $\epsilon < 1 - \frac{1}{\delta\frac{\nu_d}{\nu_1}}$ and k, $\hat{\alpha}_(\mathcal{A}_m, \epsilon, \delta, k) = \infty$.

We first prove claim 1 in Theorem 9.

Proof: Let $E = \left\{e_1, \ldots, e_{\frac{k}{d}+1}\right\}$ be arbitrary elements of priority j. Consider the request sequence $\underline{\rho} = \rho_1, \ldots, \rho_M$ s.t. $\rho_i = (e_{i \mod (\frac{k}{d}+1)}, R)$. Clearly, the cost of the optimal algorithm on the request sequence is $f^{\mathcal{O}(h)}\left(\underline{\rho}\right) = \sum_{i\in\left\{\frac{k}{d}+1\right\}} f_{e_i}$. Dividing the request series into rounds, each of size $\frac{k}{d}$, it is easy to see that in each round, $\mathcal{A}_s$ incurs a cost of at least $\min_{i\in\left[\frac{k}{d}+1\right]} f_{e_i}$. Clearly,

$$\frac{f^{\mathcal{A}_s(k)}\left(\underline{\rho}\right)}{f^{\mathcal{O}(h)}\left(\underline{\rho}\right)} \geq \frac{\sum_{j=0}^{\frac{M}{\frac{k}{d}+1}} \min_{i\in\left[\frac{k}{d}+1\right]} f_{e_i}}{\sum_{i\in\left\{\frac{k}{d}+1\right\}} f_{e_i}} \overset{M\to\infty}{\longrightarrow} \infty. \quad (64)$$

The proof now follows by Yao's Minimax principle [33]. ■
The proof of claim 3 in Theorem 9 is similar.

We now prove claim 2 in Theorem 9.

Proof: Let $\gamma \in (1, d)$ be a real number. Let $\mathcal{A}_s^{det}$ be the best online paging deterministic algorithm, which uses static partitioning. Clearly, for each cache size $k' \leq k$ there is some j such that Algorithm $\mathcal{A}_s^{det}$ allocates for priority j at most $\frac{k}{d}$ space.

Set $E = \left\{e_1, \ldots, e_{\gamma\frac{k}{d}}\right\}$ be arbitrary elements of priority j (having cost f_e each one). The read sequence $\underline{\rho}$ of length M is chosen in the following way: $\rho[i]$ is chosen uniformly from all the items of E. Then for cache sizes k' s.t. $\gamma\frac{k}{d} \leq k' \leq k$ we have

$$\mathbf{E}\left[f^{\mathcal{A}_s^{det}(k')}\left(\underline{\rho}\right)\right] \geq M\frac{\gamma-1}{\gamma}f_e, \quad (65)$$

$$\sum_{i=1}^{M} f_{\rho[i]} \leq Mf_e, \quad (66)$$

$$f^{\mathcal{O}(k')}\left(\underline{\rho}\right) \leq \gamma\frac{k}{d}f_e. \quad (67)$$

Note that (67) holds if $k' \geq \gamma\frac{k}{d}$. Hence,

$$\frac{\mathbf{E}\left[f^{\mathcal{A}_s^{det}(k')}\left(\underline{\rho}\right)\right]}{f^{\mathcal{O}(k')}\left(\underline{\rho}\right)} \geq \frac{M\frac{\gamma-1}{\gamma}}{\gamma\frac{k}{d}} \overset{M\to\infty}{\longrightarrow} \infty \quad (68)$$

for all $\gamma\frac{k}{d} \leq k' \leq k$.

$$\frac{\mathbf{E}\left[f^{\mathcal{A}_s^{det}(k')}\left(\underline{\rho}\right)\right]}{\sum_{i=1}^{M} f_{\rho[i]}} \geq \frac{\gamma-1}{\gamma} = \epsilon. \quad (69)$$

Hence, for $\epsilon < \frac{\gamma-1}{\gamma}$ and $\delta > \frac{\gamma}{d}$, the algorithm $\mathcal{A}_s^{det}$ is not (ϵ, δ, k) loosely-competitive. In other words, the algorithm $\mathcal{A}_s^{det}$ is not (ϵ, δ, k) loosely-competitive for $\delta > \frac{1}{d}$ and $\epsilon < 1 - \frac{1}{d\delta}$. By theorem **??**, the algorithm $\mathcal{A}_s$ is not loosely-competitive too. ■

The proof of claim 4 in Theorem 9 is similar.

IV. LDGM Codes for Storage-Systems

In this section we discuss a variation of *LDGM* (low-density generation-matrix) codes. *LDPC* (low-density parity-check) and LDGM codes were extensively studied in the field of communication [8], [9], [10], [11], [61], due to their low computational-complexity and nearly-optimal

rate. We propose a variation suitable for storage systems. The use of LDGM codes for storage was independently proposed previously [35], [36], [54], [61], for reasons of computational-complexity alone, and under the caveat that they are probabilistic.

Consider two coding alternatives with equal storage rates. In the first, all bits are encoded as a single group. In the second, the bits are partitioned into groups, and each group is coded. By the law of large numbers, it is clear that the former alternative is superior in terms of loss probability to the latter (whose loss probability approaches 1, as the number of groups grows). The former alternative might have other drawbacks. Reed-Solomon coding [1], for example, would necessitate accessing nearly all bits for the recovery of nearly any failure configuration. We are interested in codes for large data-groups which have a recovery penalty determined by the number of errors which took place, rather than being always proportional to the group size.

The complementary question, of bounded update penalty (defined similarly), was previously addressed in an excellent paper [57], where an inverse relationship between recovery locality and update locality was also shown.

This section is organized as follows. In Subsection IV-A we give the relevant definitions and notations of LDGM codes. We next show two codes with low block error-probability, good recovery locality, and low computational-complexity. In Subsection IV-B we show the first code, which has a sub-optimal storage rate. Using this code, we show in Subsection IV-C a code with an asymptotically-optimal storage rate, but higher update penalty. In Subsection IV-D we show simulation results.

A. Definitions and Notations

In this subsection we give the relevant definitions of LDGM codes. We consider a recursive LDGM code C composed of p levels, $C^{(1)},\ldots,C^{(p)}$, each of which can be *un-augmented* or *augmented*. In Sub-subsection IV-A.1 we give the definitions for a single level. In Sub-subsection IV-A.2 we give the definitions for the recursive code.

A.1 A single level

An un-augmented level of an LDGM code is described by a *Tanner-graph*, which is a bipartite graph

$$G=(V,E)=(L\dot{\bigcup}R,E). \tag{70}$$

Let $m=|L|$. Each node in L represents a data bit; each node in R represents a parity bit. The value of any node v is denoted by $value\,(v)$. An example of such a graph is shown in Figure 5. For any parity node $v^r\in R$, its *left-neighbor* set, $\overleftarrow{V}\,(v^r)\subseteq L$, is the set

$$\overleftarrow{V}\,(v^r)=\left\{v^\ell\in L\mid (v^\ell,v^r)\in E\right\}. \tag{71}$$

For any data node $v^\ell\in L$, its *right-neighbor* set, $\overrightarrow{V}\left(v^\ell\right)$, is defined similarly. The parity bit corresponding to v^r is set to be the *XOR* (exclusive or) of the data bits corresponding to the nodes in $\overleftarrow{V}\,(v^r)$, *i.e.*,

$$value\,(v^r)=\bigoplus_{v^\ell\in\overleftarrow{V}(v^r)} value\left(v^\ell\right). \tag{72}$$

The edges in E thus represent parity constraints. The graph is *sparse*, *i.e.*, $|E|=O\left(|L|\right)=O\left(|R|\right)$.

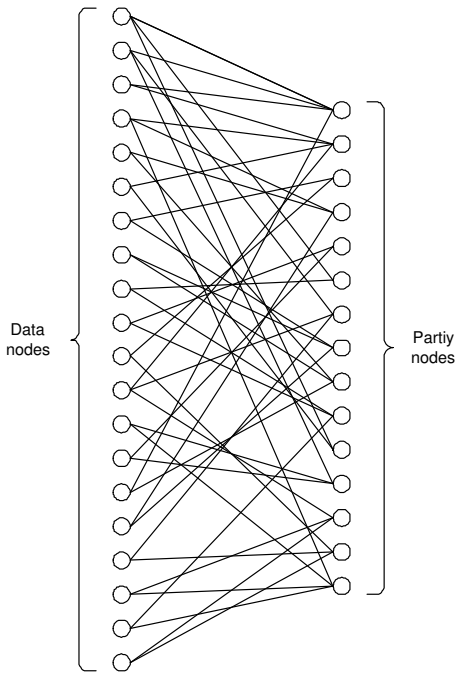

Fig. 5. A Tanner graph.

The average left-node and right-node degrees are

$$\begin{aligned} a_\ell &= \frac{\sum_{v^\ell\in L}\left|\overrightarrow{V}\left(v^\ell\right)\right|}{|L|}, \\ a_r &= \frac{\sum_{v^r\in R}\left|\overleftarrow{V}\,(v^r)\right|}{|R|}, \end{aligned} \tag{73}$$

respectively. The fractions of edges whose left and right node-degree is i, are

$$\begin{aligned} \lambda_i &= \frac{\left|\left\{e=(v^\ell,v^r)\in E\mid \left|\overrightarrow{V}\left(v^\ell\right)\right|=i\right\}\right|}{|E|}, \\ \rho_i &= \frac{\left|\left\{e=(v^\ell,v^r)\in E\mid \left|\overleftarrow{V}\,(v^r)\right|=i\right\}\right|}{|E|}, \end{aligned} \tag{74}$$

respectively. These degrees are usually put into the form of generating functions

$$\begin{aligned} \lambda(x) &= \sum_{i=1}^{\infty}\lambda_i x^{i-1}, \\ \rho(x) &= \sum_{i=1}^{\infty}\rho_i x^{i-1}. \end{aligned} \tag{75}$$

Note that $\lambda(x)$ and $\rho(x)$ completely define the un-augmented level.

Assume that all nodes in R are valid, and that some nodes in L have failed. The recovery process is iterative. In each step, a node $v^r \in R$ is chosen s.t. a single node $v^\ell \in \overleftarrow{V}(v^r)$ is failed, and the value of v^ℓ is then corrected using (72). Of course, the recovery process terminates prematurely if such a node v^r cannot be found. We denote by $\mathbf{P}_{\text{block}}(L)$ the block error-probability, which is the probability that the recovery process terminates while some bits in L have not been recovered. We denote by $\mathbf{P}_{\text{bit}}(L,\alpha)$ the α-bit error probability, which is the probability that the recovery process terminates while a fraction of at least α bits in L has not been recovered.

We will use the following theorem from [10].

Theorem 10: Assume a fraction of δ nodes from L originally fail.

1. In the family of codes for which

$$\forall_{x\in(0,\delta]}\delta\cdot\lambda(1-\rho(1-x))<x, \tag{76}$$

a fraction of $1-o(1)$ of the codes have

$$\forall_{\alpha<1}\mathbf{P}_{\text{bit}}(L,\alpha)=o\left(e^{-\alpha m}\right). \tag{77}$$

2. In the family of codes for which (76) holds and

$$\lambda_1=\lambda_2=0, \tag{78}$$

a fraction of $1-o(1)$ of the codes have

$$\mathbf{P}_{\text{block}}(L)=o\left(e^{-\Omega(m)}\right). \tag{79}$$

As seen in Theorem 10, a code for which (78) does not hold, does not guarantee the recovery of all nodes. On the other hand, it was shown in [62] that if (78) holds, then the storage rate is suboptimal. For this reason, an augmented level is commonly used. In an augmented level, the nodes in L are protected by both an LDGM code for which (78) does not hold, and by an expander-based code [10], [63]. The LDGM code corrects all but a negligible number of failed nodes. The expander-based code corrects the rest.

A.2 The recursive code

The recovery process of $C^{(i)}$, described in Sub-subsection IV-A.1, requires that all the nodes in R be valid. Since these nodes can fail, a recursive code is used. This is shown in Figure 6. In the recursive code, each level's R is taken to be the next level's L. A sequence of p levels is built, $C^{(1)},\ldots,C^{(p)}$. The last level, $C^{(p)}$, is composed of a non-recursive code (*e.g.*, Reed-Solomon).

Note that in general, it is not required that the $C^{(i)}$ be of the same type. Typically, however, a recursive LDGM code is composed of same-type levels (except for the last). Let the storage-rate of a level be denoted by β, and let $\beta'=\frac{1-\beta}{\beta}$. Let the number of original data bits be n. Level

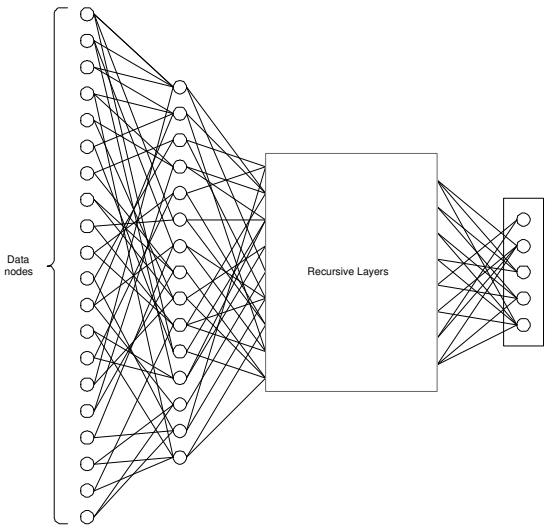

Fig. 6. A recursive LDGM encoding.

i is then an LDGM code with $\beta'^{i-1}n$ left nodes and $\beta'^{i}n$ right nodes. Assume that p is set s.t. $\beta'^{p-1}n\approx\sqrt{n}$. It follows that the total number of redundant nodes is

$$n\sum_{i\in[p]}\beta'^{i}\approx \left(n-\beta'\sqrt{n}\right)\frac{\beta'}{1-\beta'}=\left(n-\frac{1-\beta}{\beta}\sqrt{n}\right)\frac{1-\beta}{2\beta-1}. \tag{80}$$

B. A Truncated Right-Regular Code

In this subsection we define a right-regular code which is a variation of [46], and analyze its performance. The code is *truncated*, in its power series $\lambda(x)$ having a 0 coefficient of the x term. This code has low block error-probability, good recovery locality, and low computational-complexity. We use this code in Subsection IV-C as well.

Definition 6: Let $\hat{C}^{(i)}$ be an un-augmented level of an LDGM code, defined by the generating functions:

$$\begin{aligned}\hat{\lambda}(x) &= \frac{\sum_{k=2}^{\hat{q}-1}\binom{\hat{\alpha}}{k}(-1)^{k+1}x^k}{1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}},\\ \hat{\rho}(x) &= x^{\hat{a}-1},\end{aligned} \tag{81}$$

where $\hat{a}\geq 2$ and $\hat{q}\geq 3$, are arbitrary integers, and $\hat{\alpha}=1/(\hat{a}-1)$. Let the average left-node degree be denoted by $\hat{a}_\ell$ (the average right-node degree is obviously $\hat{a}$).

Note that $\hat{\lambda}(x)$ is a normalized sum of x^i terms of the Taylor expansion of $1-(1-x)^{\hat{\alpha}}$, for $2\leq i\leq\hat{q}$.

The main result in this subsection is the following.

Theorem 11: Let

$$\hat{\delta}_{\max} = \frac{1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}}{1-\hat{\alpha}}. \tag{82}$$

Then we have the following attributes of $\hat{C}^{(i)}$:

1. Let δ denote the fraction of errors in L.

$$\delta \leq \hat{\delta}_{\max} \Rightarrow \mathbf{P}_{\text{block}}(L) = o\left(e^{-\Omega(m)}\right). \tag{83}$$

2. • If the number of failed data-nodes is $|L_f|$ and the number of accessed data-nodes is $|L_a|$, then

$$|L_a| \leq \hat{a}\,|L_f|\,. \tag{84}$$

• Let $\hat{C}$ be a recursive LDGM code composed of $\hat{C}^{(i)}$s. Then the expected number of accesses per single (data or parity) failed-node recovery approaches $\hat{a}+1$.

3. • If the number of modified data-nodes is $|L_u| \ll |L|$ and the number of accessed parity-nodes is $|R_a|$, then

$$|R_a| \approx |R| - |R|\left(1 - \frac{\hat{a}_\ell\,|L_u|}{\hat{a}\,|R|}\right)^{\hat{a}} \tag{85}$$

• Let $\hat{C}$ be a recursive LDGM code composed of $\hat{C}^{(i)}$s, protecting n original data-bits. Let a single data-node be modified. Then the ratio between the expected number of accessed nodes and total number of redundant nodes is

$$o\left(\frac{1}{n^{\Omega(1)}}\right). \tag{86}$$

4. The storage rate of the code is greater than

$$1 - \frac{2\left(1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}\right)}{1 - \frac{2}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}}. \tag{87}$$

5. The ratio between the fraction of errors corrected by $\hat{C}^{(i)}$, and the fraction corrected by an optimal code of the storage rate in (87), is

$$\frac{1}{2} - \frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}. \tag{88}$$

6. The computational complexity of encoding and decoding is linear.

For much of the proof, we modify proofs from [10] and [46].

We first prove item 1 of Theorem 11,

Proof: To apply item 2 of Theorem 10, we first must show that the degree distributions are valid ones. To do so, it is sufficient to show that the coefficients of $\hat{\lambda}(x)$ are positive, and that $\hat{\lambda}(1) = \hat{\rho}(1) = 1$.

To show the positivity of the coefficients, note that

$$\binom{\hat{\alpha}}{k} = \frac{\hat{\alpha}(\hat{\alpha}-1)\cdots(\hat{\alpha}-k+1)}{k!} = \tag{89}$$
$$(-1)^{k-1}\frac{\hat{\alpha}}{k}\left(1 - \frac{\hat{\alpha}}{k-1}\right)\cdots\left(1 - \frac{\hat{\alpha}}{2}\right)(1-\hat{\alpha}).$$

To show that the sum of the coefficients is 1, note that

$$\begin{aligned}\hat{\lambda}(1) &= \frac{\sum_{k=2}^{\hat{q}-1}\binom{\hat{\alpha}}{k}(-1)^{k+1}1^k}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \overset{(a)}{=} 1, \qquad (90)\\ \hat{\rho}(1) &= 1^{\hat{a}-1} = 1,\end{aligned}$$

where (a) follows from the fact that, by induction, [10]

$$\sum_{k=1}^{\hat{q}-1}\binom{\hat{\alpha}}{k}(-1)^{k+1} = 1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}. \tag{91}$$

We now show that the condition of item 2 in Theorem 10 holds.

Expanding $\hat{\lambda}(1-\hat{\rho}(1-x))$, we have

$$\begin{aligned}&\hat{\lambda}(1-\hat{\rho}(1-x)) \overset{(a)}{=} \qquad (92)\\ &\frac{\sum_{k=2}^{\hat{q}-1}\binom{\hat{\alpha}}{k}(-1)^{k+1}(1-\hat{\rho}(1-x))^k}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \overset{(b)}{\leq}\\ &\frac{\sum_{k=2}^{\infty}\binom{\hat{\alpha}}{k}(-1)^{k+1}(1-\hat{\rho}(1-x))^k}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \overset{(c)}{=}\\ &\frac{1 - (1-(1-\hat{\rho}(1-x)))^{\hat{\alpha}} - \hat{\alpha}(1-\hat{\rho}(1-x))}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \overset{(d)}{=}\\ &\frac{1 - \left(1-\left(1-(1-x)^{\hat{a}-1}\right)\right)^{\hat{\alpha}} - \hat{\alpha}\left(1-(1-x)^{\hat{a}-1}\right)}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \overset{(e)}{=}\\ &\frac{x - \hat{\alpha} + \hat{\alpha}(1-x)^{\frac{1}{\hat{\alpha}}}}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}}, \qquad (93)\end{aligned}$$

where in the above, (a) follows from (81), (b) follows from the fact that by (89), for $y \leq 1$,

$$\binom{\hat{\alpha}}{k}(-1)^{k+1}(1-y)^k \geq 0, \tag{94}$$

(c) follows from the fact that the Taylor expansion of $1-(1-y)^{\hat{\alpha}}$ is

$$1 - (1-y)^{\hat{\alpha}} = \sum_{k=1}^{\infty}\binom{\hat{\alpha}}{k}(-1)^{k+1}y^k, \tag{95}$$

and (d) and (e) follow from (81).

We also note for the above, that it was shown in [46] that

$$\breve{\lambda}(x) = \frac{\sum_{k=1}^{\hat{q}-1}\binom{\hat{\alpha}}{k}(-1)^{k+1}x^k}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}} \tag{96}$$

converges in the relevant range of x, and so

$$\hat{\lambda}(x) = \frac{\breve{\lambda}(x)\left(1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}\right) - \hat{\alpha}x}{1 - \frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1} - \hat{\alpha}} \tag{97}$$

converges as well.

Using the above expansion, we now have that for $x \in (0, \hat{\delta}_{\max}]$,

$$\begin{aligned}\delta\hat{\lambda}(1-\hat{\rho}(1-x)) &\overset{(a)}{\leq} \\ \frac{\delta\left(x-\hat{\alpha}+\hat{\alpha}\cdot(1-x)^{\frac{1}{\hat{\alpha}}}\right)}{1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}} &\overset{(b)}{\leq} \\ \frac{\delta\left(1-\hat{\alpha}\right)x}{1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}} &= \\ \frac{\delta}{\hat{\delta}_{\max}}x \leq x,\end{aligned} \tag{98}$$

where, in the above, (a) follows from (93), and (b) follows from the fact that

$$\forall_{0\leq x\leq 1} x-\hat{\alpha}+\hat{\alpha}\cdot(1-x)^{\frac{1}{\hat{\alpha}}} \leq x(1-\hat{\alpha}). \tag{99}$$

■

We now prove item 2 of Theorem 11.

Proof: The first part follows easily from description, in Sub-subsection IV-A.1, of the recovery process. For the second part, assume that the number of original data-bits is n. Note that the number of accesses required for a failed node v is at most

$$\begin{cases} \sqrt{n} & , \quad v \in \hat{C}^{(p)} \\ \hat{a} & , \quad \text{otherwise} \end{cases}. \tag{100}$$

Let n' be the total number of nodes not in $\hat{C}^{(p)}$. The average number of accessed nodes is

$$\frac{\hat{a}n' + \sqrt{n}\cdot\sqrt{n}}{n'+\sqrt{n}}. \tag{101}$$

Using (80), we get that the average number of accessed nodes is at most

$$\hat{a} + \frac{2\beta-1}{\beta} < \hat{a}+1, \tag{102}$$

since $\frac{2\beta-1}{\beta} < 1$ for $\frac{1}{2} < \beta < 1$. ■

We now prove item 3 of Theorem 11.

Proof: To prove the first part, we use the differential-equation approach from [10]. Some edges emanate from the $L_m \subset L$ modified nodes. Let the number of such edges be m'; let the edges be $e_1, \ldots e_{m'}$.

Assume a process of m' steps. Let each step take $\Delta t = \frac{1}{m'}$ time. Let r_t denote the average number of un-accessed parity-nodes at time t. At step i, the node v_i^r is accessed, where $v_i^r \in R$ is the terminating node of e_i. The value of $r_{i\Delta t}$ is updated, if necessary. It can be shown that the difference-equation system for r_t is:

$$\begin{aligned} r_{t+\Delta t} - r_t &= -\frac{\hat{a}r_t}{\hat{a}|R| - \frac{t}{\Delta t}}, \\ r_0 &= |R|. \end{aligned} \tag{103}$$

Manipulating and taking $\Delta t \to 0$, we get the differential-equation system:

$$\begin{aligned} \frac{d\,r_t}{d\,t} &= -\frac{m'\hat{a}r_t}{\hat{a}|R| - tm'}, \\ r_0 &= |R|. \end{aligned} \tag{104}$$

Solving for r_t and setting $t=1$, we get

$$r_1 = |R|\left(1-\frac{m'}{\hat{a}|R|}\right)^{\hat{a}}. \tag{105}$$

It remains to approximate m'. This random variable is distributed hyper-geometrically. For $|L_m| \ll |L|$, $m' \approx \hat{a}_l L_m$.

We now prove the second part. On the average, when updating a left node, $\hat{a}_l$ right nodes should be updated. For each level, the maximal number of nodes updated is not larger than the number of parity nodes in the level. It follows that for any $j \in [p]$, the average number of accessed nodes is bounded by

$$\sum_{i=0}^{j} {\hat{a}_l}^{i+1} + \sum_{i=j+1}^{\hat{p}} \beta^{i+1} n. \tag{106}$$

Minimizing by j yields that the number of accessed nodes is

$$\begin{aligned} &\approx n^{\log_{\frac{\hat{a}_l\beta'}{1-\beta'}}(\hat{a}_l)}\left(\frac{\hat{a}_l}{\hat{a}_l-1}+\frac{1-\beta'}{2\beta'-1}\right) \\ &-\sqrt{n}\frac{1-\beta'}{2\beta'-1} - \frac{\hat{a}_l}{\hat{a}_l-1}. \end{aligned} \tag{107}$$

The combination of (80) and (107) shows that the ratio of the number of accessed nodes to the number of redundant nodes is

$$o\left(n^{\log_{\frac{\hat{a}_l\beta'}{1-\beta'}}(\hat{a}_l)-1}\right). \tag{108}$$

Note that since $\frac{\beta'}{1-\beta'} > 1$, the power of n in (108) is negative. ■

We now prove item 4 of Theorem 11.

Proof: By definition, the storage rate of the code is the ratio between the size of the left-node set and the total number of nodes.

$$\begin{aligned} r = \frac{|L|}{\left|L \dot{\bigcup} R\right|} = \frac{1}{1+\frac{|R|}{|L|}} \geq 1 - \frac{|R|}{|L|} = \\ 1 - \frac{\int_0^1 \hat{\rho}(x)\,dx}{\int_0^1 \hat{\lambda}(x)\,dx} \overset{(a)}{=} \\ 1 - \frac{2\left(1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}\right)}{1-\frac{2}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}} \end{aligned} \tag{109}$$

where (a) follows from the fact that

$$\sum_{k=1}^{\hat{q}-1}\binom{\hat{\alpha}}{k}\frac{(-1)^{k+1}}{k+1}=\frac{\hat{\alpha}-\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}}{\hat{\alpha}+1}, \tag{110}$$

$$\int_0^1\hat{\lambda}(x)\,dx=\frac{\hat{\alpha}}{\hat{\alpha}+1}\frac{1-\frac{2}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}}{2(1-\frac{\hat{q}}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha})},$$

$$\int_0^1\hat{\rho}(x)\,dx=\frac{\hat{\alpha}}{\hat{\alpha}+1}.$$

■

We now prove item 5 of Theorem 11.

Proof: For any storage rate r, an optimal code could then correct a fraction of $1-r$ errors [1]. The ratio of correction capabilities is therefore,

$$\frac{\hat{\delta}_{\max}}{1-R}\geq \frac{1-\frac{2}{\hat{\alpha}}\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}-\hat{\alpha}}{2(1-\hat{\alpha})}\overset{(a)}{=}\frac{1}{2}-\frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}, \tag{111}$$

where in the above, (a) uses the fact that [46]

$$\exists_c\forall_{\hat{q}\geq 2}\forall_{\hat{\alpha}<\frac{1}{2}}\frac{c\hat{\alpha}}{\hat{q}^{\hat{\alpha}+1}}\leq\binom{\hat{\alpha}}{\hat{q}}(-1)^{\hat{q}+1}\leq\frac{\hat{\alpha}}{\hat{q}^{\hat{\alpha}+1}}. \tag{112}$$

■

Item 6 of Theorem 11 is a known property of LDGM codes. It follows from the sparseness of G.

C. An Augmented Right-Regular Code

In this subsection we define an augmented right-regular code based on [46] and Subsection IV-B. This code has low block error-probability, good recovery locality, and low computational-complexity. Its storage rate is asymptotically optimal; its update penalty is higher than the code from Subsection IV-C.

We will require the following code from [46].

Definition 7: Let $\tilde{C}^{(i)}$ be an un-augmented level of an LDGM code, defined by the generating functions:

$$\begin{aligned}\tilde{\lambda}(x) &= \frac{\tilde{\alpha}\sum_{k=1}^{\tilde{q}-1}\binom{\tilde{\alpha}}{k}(-1)^{k+1}x^k}{\tilde{\alpha}-\tilde{q}\binom{\tilde{\alpha}}{\tilde{q}}(-1)^{\tilde{q}+1}},\\ \tilde{\rho}(x) &= x^{\tilde{a}-1},\end{aligned} \tag{113}$$

where $\tilde{a}\geq 2$ and $\tilde{q}\geq 2$, are arbitrary integers, and $\tilde{\alpha}=\frac{1}{\tilde{a}-1}$.

We now define a code that combines the previous two.

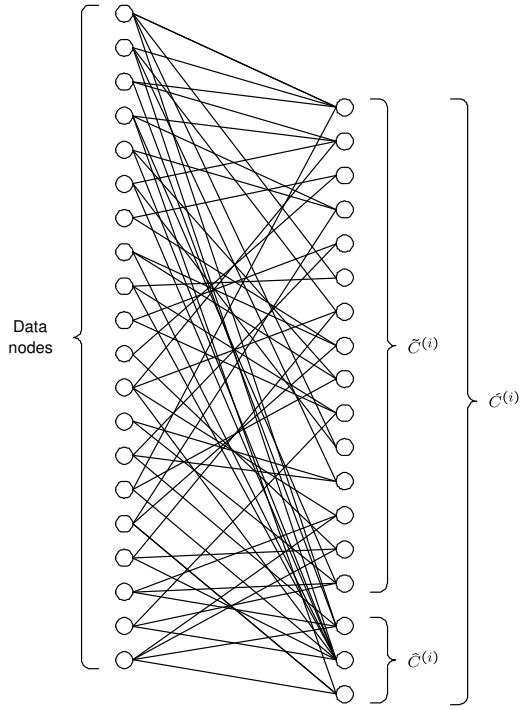

Fig. 7. The augmented level-code.

Definition 8: Let $\acute{C}^i=\acute{C}^i(\epsilon)$ be an LDGM level defined by $\tilde{C}^i$, and augmented by $\hat{C}^i$ (from Subsection IV-B) coded at storage rate

$$1-\frac{\epsilon}{\frac{1}{2}-\frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}}. \tag{114}$$

A level of this code is shown schematically in Figure 7. This idea is similar to augmenting an LDGM code by an Expander-based code [10], but has much lower recovery penalty.

The main result in this subsection is the following.

Theorem 12: Let $\tilde{\alpha}$ and $\tilde{q}$ be defined as in Definition 7. Let

$$\begin{aligned}\acute{\delta}_{\max} &= \frac{\tilde{\alpha}-\tilde{q}\binom{\tilde{\alpha}}{\tilde{q}}(-1)^{\tilde{q}+1}}{\tilde{\alpha}},\\ \tilde{r} &= 1-\frac{\tilde{\alpha}-\tilde{q}\binom{\tilde{\alpha}}{\tilde{q}}(-1)^{\tilde{q}+1}}{\tilde{\alpha}-\binom{\tilde{\alpha}}{\tilde{q}}(-1)^{\tilde{q}+1}}.\end{aligned} \tag{115}$$

Then we have the following attributes of $\acute{C}^{(i)}$:

1. Let the fraction of errors which occurred be δ.

$$\delta\leq\acute{\delta}_{\max}\Rightarrow\mathbf{P}_{\text{block}}(L)=o\left(e^{-\Omega(m)}\right). \tag{116}$$

2. If the number of failed data-nodes is $|L_f|\gg 1$ and the number of accessed data-nodes is $|L_a|$, then

$$|L_a|\lessapprox\tilde{a}\,|L_f|\,. \tag{117}$$

3. • If the number of modified data-nodes is $|L_m| \ll |L|$ and the number of accessed parity-nodes is $|R_a|$, then

$$|R_a| \lessapprox |R| \left(1 - \left(1 - \frac{\tilde{a}_\ell \, |L_m|}{\tilde{a} \, |R|} \right)^{\tilde{a}} + \frac{\epsilon}{\frac{1}{2} - \frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}} \right) \quad (118)$$

• Let $\acute{C}$ be a recursive LDGM code protecting n original data-bits, whose each layer is $\acute{C}^{(i)}$. Let a single data-node be modified. Then the ratio between expected accessed parity-nodes and total parity-nodes is

$$o\left(\frac{1}{n^{\Omega(1)}} \right). \quad (119)$$

4. The storage rate of the code is greater than

$$\tilde{r} \left(1 - \frac{\epsilon}{\frac{1}{2} - \frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}} \right). \quad (120)$$

5. The ratio between the fraction of errors corrected by the code and the fraction corrected by an optimal code of the same storage rate is

$$\left(1 - \frac{1}{\tilde{q}^{\tilde{\alpha}+1}} \right) \left(1 - \frac{\epsilon}{\frac{1}{2} - \frac{1}{\hat{q}^{\hat{\alpha}+1}(1-\hat{\alpha})}} \right). \quad (121)$$

6. The computational complexity of encoding and decoding is linear.

We first prove items 1 of Theorem 12.

Proof: Using Item 1 of Theorem 10, it can be shown that $\tilde{C}^{(i)}$ corrects all but a negligible fraction of errors ϵ' s.t. $\epsilon' \ll \epsilon$. Using Item 2 of Theorem 10 and the proof of Theorem 11, it follows that $\hat{C}^{(i)}$ corrects at least an ϵ fraction of errors. ■

Items 2 and 3 of Theorem 12 can be proved as in Theorem 11.

We now prove item 4 of Theorem 12. Item 5 follows in a similar manner.

Proof: Let $\acute{L}$ denote the set of data nodes. Let $\tilde{R}$ denote the set of parity nodes defined by the code $\hat{C}^{(i)}$, $\hat{R}$ denote the set of parity nodes defined by the code $\hat{C}^{(i)}$, and $\acute{R}$ be the total set of parity nodes, *i.e.*, $\acute{R} = \tilde{R} \bigcup \dot{\hat{R}}$.

Let $\tilde{d}$ and $\hat{e}$ be defined by

$$\begin{aligned} \tilde{d} &= 1 - \frac{1}{\tilde{q}^{\tilde{\alpha}+1}}, \quad (122) \\ \hat{e} &= \frac{1}{2} - \frac{1}{\frac{1}{\hat{q}^{\hat{\alpha}+1}}(\hat{\alpha}+1)}. \end{aligned}$$

Then

$$\begin{aligned} \frac{\tilde{\delta}_{\max}}{1 - \frac{|\acute{L}|}{|\acute{L}|+|\tilde{R}|}} &= \tilde{d}, \quad (123) \\ \frac{\epsilon}{1 - \frac{|\acute{L}|}{|\acute{L}|+|\tilde{R}|}} &= \hat{e}. \end{aligned}$$

Using (123), we obtain relationships between $\left|\tilde{R}\right|$ and $\left|\acute{L}\right|$, and between $\left|\hat{R}\right|$ and $\left|\acute{L}\right|$:

$$\begin{aligned} \left|\tilde{R}\right| &= \left|\acute{L}\right| \frac{\frac{\acute{\delta}_{\max}}{\tilde{d}}}{1 - \frac{\acute{\delta}_{\max}}{\tilde{d}}}, \quad (124) \\ \left|\hat{R}\right| &= \left|\acute{L}\right| \frac{\frac{\epsilon}{\hat{e}}}{1 - \frac{\epsilon}{\hat{e}}} \geq \left|\acute{L}\right| \frac{\epsilon}{\hat{e}}. \end{aligned}$$

Inserting (124) into the storage-rate expression of the augmented code, we obtain:

$$\begin{aligned} &\frac{\left|\acute{L}\right|}{\left|\acute{L}\right| + \left|\acute{R}\right|} = \quad (125) \\ &\frac{\left|\acute{L}\right|}{\left|\acute{L}\right| + \left|\tilde{R}\right|} \frac{1}{1 + \frac{|\hat{R}|}{|\acute{L}|+|\hat{R}|}} = \\ &\tilde{r} \frac{1}{1 + \frac{\frac{\epsilon}{\hat{e}}(1 - \frac{\acute{\delta}_{\max}}{\tilde{d}})}{1 - \frac{\epsilon}{\hat{e}}}} \geq \\ &\tilde{r} \left(1 - \frac{\epsilon}{\hat{e}} \right). \end{aligned}$$

Item 6 of Theorem 12 follows as in Theorem 11. ■

D. Simulation Results

In this subsection we show simulation results for a single-layer of the truncated right-regular LDGM codes discussed in Subsection IV-B:

• Figure 8 shows the single-layer storage-rate and maximal fraction of fixable nodes as a function of left and right edge-degrees (derived in (87) and (82) respectively). As noted, the rate of this code is sub-optimal, necessitating the use of the augmented code from Subsection IV-C.

• We have performed 500 tests on the block error-probability. In each test a random code with $m = 10000$ left nodes, $\hat{q} = 4$, $\hat{a} = 4$, $\hat{\alpha} = \frac{1}{3}$, and 8236 right nodes was first built. Then, 10^6 iterations were performed. In each iteration 3900 left nodes were initially set to be corrupted, and were then attempted to be fixed. The code succeeded in every single iteration of every single test.

• Figures 9 and 10 show the number of updated nodes as a function of the fraction of modified left nodes, for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes. Note that as predicted, simulation results coincide with the expected value, given by (85). For small fractions of left nodes being updated, the number of update-IOs can be approximated as the number of updated left nodes times average left degree (a_l).

• Figure 11 shows the number of nodes, accessed by the recovery procedure, as a function of the fraction of failed left nodes, for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes. Note that the assumption of a accesses per each failed node gives the upper bound on the number of accessed nodes, as shown in (84).

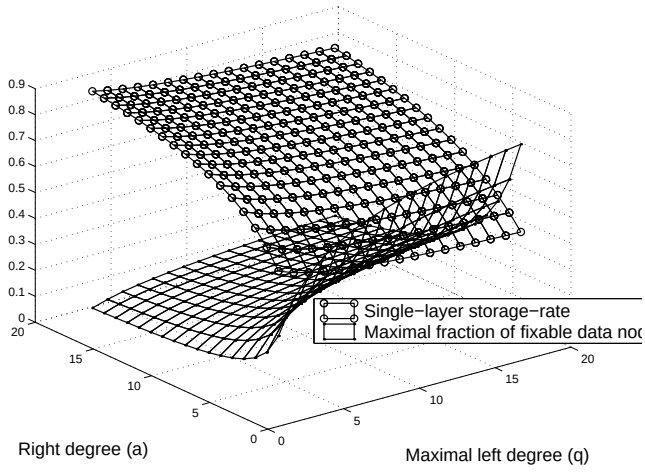

Fig. 8. Rates and error-correction capabilities as a function of edge degrees.

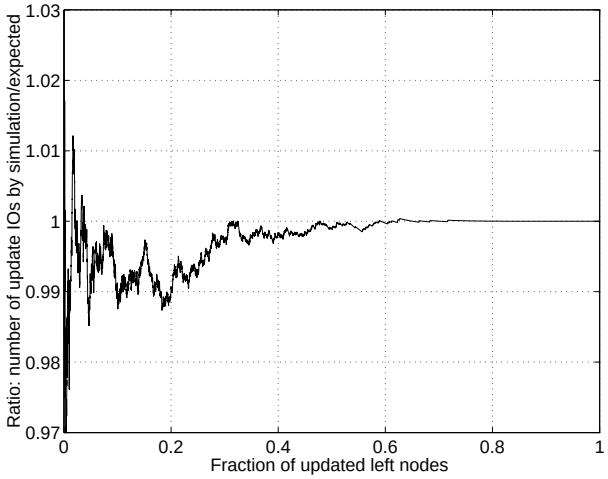

Fig. 10. Ratio $\frac{\text{the number of updated nodes by simulation}}{\text{the number of updated nodes by (85)}}$ as a function of the fraction of modified left nodes for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes.

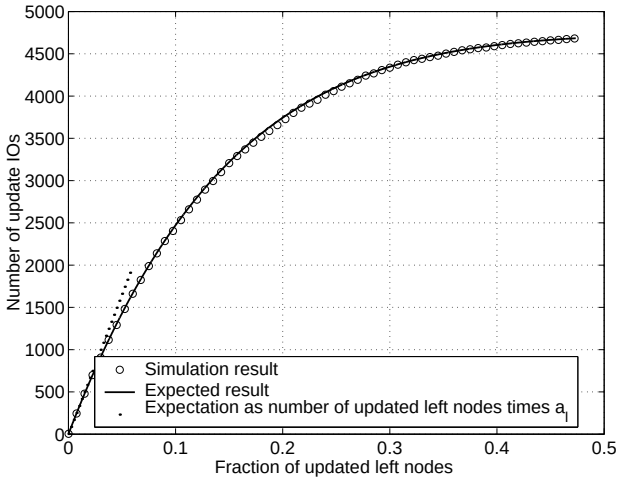

Fig. 9. Number of updated nodes as a function of the fraction of modified left nodes for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes.

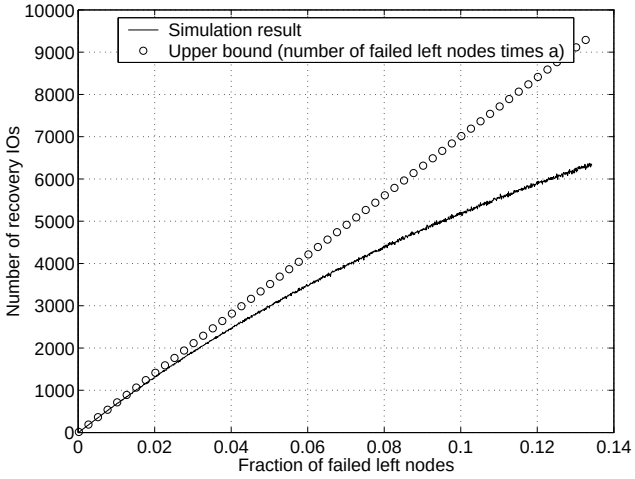

Fig. 11. Number of nodes, accessed by the recovery procedure, as a function of the fraction of failed left nodes, for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes.

V. Applications to Reference Data

The first simple application of the LDGM code is for protecting large amounts of reference data (*i.e.*, data which is not often updated). In this application, the bounded proportionality of accessed bits per erased bits, comes into effect. *E.g.*, consider a three-site reference-data system storing a copy of each datum in two of the three sites. While this system can survive a disaster obliterating a single site, or ongoing failures affecting some devices in all three sites, the MTTDL can be shown to decrease linearly in the number of devices. Using an LDGM code to protect all devices, would result in a system with similar performance, but with MTTDL increasing in the number of devices. Using a classic Reed-Solomon code to protect all devices, would result in a system whose devices would be engaged in much of the time in recovery-related operations.

VI. Applications to Random Sparing

Data which are active (as opposed to reference data), are not best protected by the method in Section V; The update penalty is too high. Commonly, active data-groups are distributed in some manner over devices, in an attempt to parallelize request handling [49], [40], [39], [38], [50], [37]. We consider a *random-sparing* scheme, apparently similar to an independent OceanStore solution [35], but differing in the random location-decision and use of bounded-penalty codes.

We consider a *random-sparing* scheme, apparently similar to an independent OceanStore solution [35]but differing in the random location-decision and use of bounded-penalty codes. Using queuing theory and the concepts of random-variable negative dependence, we perform approximate analysis on the attributes of this scheme: MTTDL,

coding-group average size and rate, required device memory, and reliability-related workload.

This section is organized as follows. In Subsection VI-A we describe the scheme. In subsection VI-B we analyze it. In Subsection VI-C we show simulation results. In Subsection VI-D we discuss some implications of the analysis.

A. Scheme Description

In this subsection we describe the random sparing scheme.

Initially, there are n devices in the system. The data is divided into data groups. For simplicity, we assume the size of each group is ℓ blocks, of which $\ell - \ell'$ can be corrected. We assume the code used has bounded recovery-locality. Specifically, we assume that the ratio of accessed blocks per failed blocks of a group, is at most k. We assume an *average fill-ratio* of β, *i.e.*, of the nc blocks in the system, βnc are taken (and so the number of groups is $\frac{\beta nc}{\ell}$). We place the restriction that a fraction of at most μ of all operations can be dedicated to recovery operations.

We divide the operating time into *rounds* of duration t_s, and *epochs*, each consisting of s rounds. For simplicity, we assume that $st_s = \frac{c}{\mu r}$, *i.e.*, each epoch lasts the time that would be required to sequentially read a device from start to end, normalized by μ.

Naturally, a request to a device can be blocked due to its servicing previous requests. We assume that each device has a queue of read and write requests, which it services subject to its bandwidth and constraint on fraction of recovery-operations. Memory is required for write requests in the queue, and for read requests which have been completed but whose contents are still needed. We later analyze the system-wide amount of such memory.

At some points, the scheme requires writing a (recovered) group element to a one of the operating devices. The selection of the operating device is random, but differs from the standard uniform selection between all operating devices which are not full [35]. Rather, a uniform selection is made between all operating devices (full or non-full), which does not contain a member of the data group. If a full device is selected, a *reassignment* takes place. A random element from the selected device is chosen to be reassigned to a different device, the recovered element is written to the device instead of it, and the process continues recursively with the reassigned element. We explain the rationale for this later.

The scheme is composed of two concurrent processes, a *contracting* process and an *expanding* process, which we describe next.

The contracting process works as follows. In each round, the system observes the set of devices which have failed in the round. For each data group, the system identifies the members that are on failed devices. If these members cannot be recovered, then data has been lost at this time. Otherwise, the system randomly selects, for each failed member, a subset of k devices out of all subsets of k devices which can recover the element. For each of these devices, it inserts into its queue a request to read the required element. If the system has enough elements to recover an element, it recovers it, randomly selects a device, and inserts into its queue a request to write the element (possibly triggering reassignment).

We term this a contracting process, since it attempts to maintain the data groups into a continually contracting group of devices (those which are still operating).

The expanding process works as follows. At the beginning of each epoch, replacement devices are inserted into the system. The number of replacement devices is determined s.t. the expected number of operating devices in the end of the epoch will remain n. In each round, the system chooses, for each replacement device, $\frac{\beta c}{t_s}$ random data groups which are not represented in the device. For each such group, it randomly selects a non-replacement device containing a member of the group, and inserts into its queue a request to read the required element. When the element has been read, the replacement devices writes it.

At the end of an epoch, all replacement devices which have not failed during an epoch, become (new) operating devices. For each element written to a replacement device during an epoch, the system modifies its marked location. It is now marked as being located in the (new) operating device. Its old location is marked as empty.

We term this an expanding process, since it attempts to maintain the data groups into a continually expanding group of devices (the union of operating devices and replacement devices).

From the above description, it is clear that the number of operating devices (originally n), and the average fill ratio of each device (originally β), change with time. In an arbitrary point in an epoch, we denote these by n' and β', respectively.

The main result of this section is the following theorem.

Theorem 13: Assume

$$\ell \gg \log(n), \quad (126)$$

$$c\left(1-\beta\left(1+2\lambda st_s+2(\lambda st_s)^2\right)\right) \gg 1, \quad (127)$$

$$\frac{1}{\lambda} \gg \frac{c}{\mu r}. \quad (128)$$

For some $\delta \gtrsim 0$, let the storage rate of each coding group be

$$\approx 1-(1+\delta)\lambda\left(t_s+\frac{k+1}{\mu r}\right). \quad (129)$$

Then the system MTTDL of random sparing grows like

$$\frac{\frac{c}{s\mu r}}{e^{\left(-\frac{1}{2\frac{\beta nc}{\ell}e^{-\lambda\left(t_s+\frac{k+1}{\mu r}\right)}\frac{\ell\delta^2}{2}}\right)}}. \quad (130)$$

B. Analysis

Following is the proof-outline of Theorem 13. In Lemma 7 we bound the expectation of the number of full devices in a round. In Lemma 8 we bound the probability of a failure in a round, given the effective number of failed devices in the round. In Lemma 9 we approximate the number of pending recovery-requests, using Lemma 7. Using the number of pending requests, we approximate the effective number of failed devices in a round.

The analysis of random sparing is complicated by the fact that the devices' and groups' states are not independent, *e.g.*, if some data-groups have very many representatives in some set of devices, then the number of representatives of other data-groups is probably not very large. This makes it difficult to apply directly the Chernoff bound. For this reason, we use in some places in the analysis, the notion of *negative dependence* [64].

Definition 9: Let $X = \left\{x_1, \ldots, x_{|X|}\right\}$ be an ordered set of random variables. The elements of X are *negatively dependent* if for every disjoint index-sets $I \bigcup \cdot\, J \subseteq [|X|]$, and any functions $f : \mathcal{R}^{|I|} \to \mathcal{R}$ and $g : \mathcal{R}^{|J|} \to \mathcal{R}$ that are both non-increasing or non-decreasing,

$$\mathbf{E}\left[f\left(x_i, i \in I\right) \cdot g\left(x_j, j \in J\right)\right] \leq \mathbf{E}\left[f\left(x_i, i \in I\right)\right] \cdot \mathbf{E}\left[g\left(x_j, j \in J\right)\right]. \tag{131}$$

The following lemma contains useful properties of negatively-associated random variables which we will use. The statements of the lemma appear in [64], or are slight variations of them.

Lemma 6: Let X be an ordered set of random variables. Then

1. If $f : \mathcal{R} \to \mathcal{R}$ is a non-increasing or non-decreasing function, then the Chernoff bound can be applied to $\sum_{x \in X} f\left(x\right)$.
2. If Y is a set of negatively-associated random variables, and X and Y are independent, then $X \bigcup \cdot\, Y$ is negatively dependent.

We first bound the expectation of the number of full devices in a round. This in turn, serves to bind the expected number of reassignments performed.

Lemma 7: At some round, let the fraction of operating devices' blocks be β'. Then, with high probability, the fraction of full devices is at most

$$\left(\frac{2}{\beta'\left(\frac{1}{\beta'}-1\right)^3 c}\right)^{\frac{1}{3}}. \tag{132}$$

Proof: Let the number of devices be n', and define the vector $\underline{c}$, s.t. $\underline{c}[i]$ denotes the number of elements in device i. For some $\delta' \leq 1$ and $\beta'' \leq \beta'$, assume that a subset $S \subseteq [n']$ exists, s.t. $|S| \geq \delta' n'$, and $\forall_{i \in S} \underline{c}[i] \geq \beta'' c$. A straightforward calculation shows that the maximum fraction of full devices, is at most

$$\delta \leq \frac{\frac{\beta' n' c - \delta' n' \beta'' c}{c - \beta'' c}}{n'} = \frac{\beta' - \delta'\beta''}{1 - \beta''}. \tag{133}$$

From (133), to show that δ is small, we can show that $\delta' \overset{\beta'' \approx \beta'}{\approx} 1$.

Consider three processes $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, each inserting $\beta' n' c$ blocks into the n' devices. Each process inserts blocks in $\frac{\beta' n' c}{\ell}$ iterations. In each iteration, process $\mathcal{A}$ inserts a coding group into ℓ distinct devices. If a full device is encountered, reassignment is performed. Process $\mathcal{B}$ does the same for the case $c \to \infty$, ad so reassignment are not performed. Process $\mathcal{C}$ does the same as $\mathcal{B}$, except that at each iteration, the ℓ devices are chosen with replacement.

Define the vector $\underline{w}$ as the indicator of $\underline{c}[i]$ being less than $(1-\epsilon)\beta' c$ elements, *i.e.*,

$$\underline{w}[i] = I\left(\underline{c}[i] \leq (1-\epsilon)\beta' c\right). \tag{134}$$

We would like to show that at the termination of process $\mathcal{A}$, with high probability, most entries of $\underline{w}$ are 0. It clearly suffices to show that at the termination of process $\mathcal{B}$, with high probability, most entries of $\underline{w}$ are 0.

At the termination of any of the three processes,

$$\forall_{i \in [n']} \mathbf{E}\left[\underline{c}[i]\right] \overset{(a)}{=} \frac{\sum_{i \in [n']} \mathbf{E}\left[\underline{c}[i]\right]}{n'} \overset{(b)}{=} \beta' c, \tag{135}$$

where (a) follows from the symmetry between devices, and (b) follows from linearity of expectation and the fact that $\mathbf{E}\left[\sum_{i \in [n']} \underline{c}[i]\right] = n'\beta' c$. At the termination of process $\mathcal{B}$, it follows from the Chernoff bound that for $i \in [n']$,

$$\mathbf{P}\left(\underline{w}[i] = 1\right) \leq e^{-\frac{\beta' c \epsilon^2}{2}}. \tag{136}$$

We cannot directly deduce from the low probability of the event $\underline{w}[n'] = 1$ in (136), the high-probability of a low-fraction of entries of $\underline{w}$ being 1. The Chernoff bound does not apply immediately , as the entries are not independent. If (136) would result from process $\mathcal{C}$, then condition 2 in Lemma 6 would hold, and the Chernoff bound would apply. For process $\mathcal{B}$, however, condition 2 in Lemma 6 does not hold, as the placements of the ℓ blocks in each iteration are not independent. Rather than using Lemma 6 directly, we show the Chernoff bound applies, by applying the Harris inequality in a slightly different way than used in [64].

Let $c'_n = \underline{c}[n']$. For any t,

$$\begin{aligned}
&\mathbf{E}\left[\Pi_{i\in[n']}e^{t\cdot\underline{w}[i]}\right] = \qquad (137)\\
&\quad \mathbf{E}\left[e^{t\cdot\underline{w}[n']}\cdot\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}\right] =\\
&\quad \mathbf{E}\left[\mathbf{E}\left[e^{t\cdot\underline{w}[n']}\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}|c'_n\right]\right] \overset{(a)}{=}\\
&\quad \mathbf{E}\left[\mathbf{E}\left[\mathbf{E}\left[e^{t\cdot\underline{w}[n']}|c'_n\right]\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}|c'_n\right]\right] \overset{(b)}{=}\\
&\quad \mathbf{E}\left[\mathbf{E}\left[e^{t\cdot\underline{w}[n']}|c'_n\right]\mathbf{E}\left[\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}|c'_n\right]\right] \overset{(c)}{\leq}\\
&\quad \mathbf{E}\left[\mathbf{E}\left[e^{t\cdot\underline{w}[n']}|c'_n\right]\right]\cdot\\
&\quad \mathbf{E}\left[\mathbf{E}\left[\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}|c'_n\right]\right] =\\
&\quad \mathbf{E}\left[e^{t\cdot\underline{w}[n']}\right]\mathbf{E}\left[\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}\right] \overset{(d)}{\leq}\\
&\quad \mathbf{E}\left[e^{t\cdot\underline{w}[n']}\right]\Pi_{i\in[n'-1]}\mathbf{E}\left[e^{t\cdot\underline{w}[i]}\right] \leq\\
&\quad \Pi_{i\in[n']}\mathbf{E}\left[e^{t\cdot\underline{w}[i]}\right].
\end{aligned}$$

In the above, (a) and (b) follow from the fact that given c'_n, $e^{t\cdot\underline{w}[n']}$ is obviously a constant. Inequality (c) follows from the Harris inequality [65]. Inequality (d) follows from a repeated application of the same idea to $\mathbf{E}\left[\Pi_{i\in[n'-1]}e^{t\cdot\underline{w}[i]}\right]$.

By (137), the Chernoff bound can be applied to $\sum_{i\in[n']}\underline{w}[i]$. It follows that for large enough n', we can approximate, with high probability $1-\frac{\sum_{i\in[n']}\underline{w}[i]}{n'}$ by the right side of (136). Inserting into (133), we obtain

$$\begin{aligned}
&\frac{\beta'-\left(1-e^{-\frac{\beta' c\epsilon^2}{2}}\right)\beta'(1-\epsilon)}{1-\beta'(1-\epsilon)} \overset{(a)}{\approx} \qquad (138)\\
&\quad \frac{e^{-\frac{\beta' c\epsilon^2}{2}}+\epsilon}{\frac{1}{\beta'}-1} \overset{(b)}{\leq} \left(\frac{2}{\beta'\left(\frac{1}{\beta'}-1\right)^3 c}\right)^{\frac{1}{3}},
\end{aligned}$$

where (a) follows from neglecting the second order term $\epsilon\cdot e^{-\frac{\beta' c\epsilon^2}{2}}$, and (b) follows from taking $\epsilon=\left(\frac{2}{\beta' c}\right)^{\frac{1}{3}}$. ■

We next bound the error probability in a round, assuming that all groups failing up to the round's start have been recovered.

Lemma 8: At some round, let the number of functioning devices be n', and let the effective number of failed device in the round be n'_f. Assume that for some δ, each group can be recovered if at most

$$\ell' = (1+\delta)n'_F\frac{\ell}{n'} \qquad (139)$$

blocks of it are lost.

Then the probability of failure in the round, is

$$e^{\left(-\frac{1}{2\frac{\beta nc}{\ell}e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}}\right)}. \qquad (140)$$

Proof: Let the set of devices failing in the round be F (*i.e.*, $n'_f=|F|$). Define the vector $\underline{z}$ whose ith entry represents the number of elements of the ith group in the failed devices of the round, *i.e.*, for $i\in[n']$,

$$\underline{z}[i] = \left|\left\{j \mid \exists_{C\in F}\underline{x}^i[j]\in C\right\}\right|. \qquad (141)$$

It is easy to see that

$$\mathbf{P}\left(\underline{z}[i]=x\right) = \frac{\binom{\ell}{x}\binom{n'-\ell}{n'_f-x}}{\binom{n'}{n'_f}}. \qquad (142)$$

Since the distribution is hyper-geometric, the $\underline{z}[i]$ are negatively dependent. From the Chernoff bound,

$$\mathbf{P}\left(\underline{z}[i]\geq(1+\delta)n'_F\frac{\ell}{n'}\right)\leq e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}. \qquad (143)$$

We define a vector $\underline{w}$ whose ith entry is the indicator of the group-failure event, *i.e.*,

$$\underline{w}[i] = I\left(\underline{z}[i]\geq(1+\delta)n'_F\frac{\ell}{n'}\right). \qquad (144)$$

We are interested in the event that $\underline{w}$ is the all-0 vector, *i.e.*, no group had many elements in F. Noting that the elements of $\underline{w}$ are negatively dependent, and that a group failed if

$$\begin{aligned}
&\sum_{i\in\frac{\beta nc}{\ell}}\underline{w}[i]\geq 1 = \qquad (145)\\
&\quad \frac{\beta nc}{\ell}\cdot e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}\left(1+\frac{1}{\frac{\beta nc}{\ell}e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}}-1\right),
\end{aligned}$$

we have, by the Chernoff bound,

$$\begin{aligned}
&\ln\left(\mathbf{P}\left(\sum_{j\in\left[\frac{\beta nc}{\ell}\right]}w[j]\geq 1\right)\right) \overset{(a)}{\leq} \qquad (146)\\
&\quad -\frac{\frac{\beta nc}{\ell}e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}\left(\frac{1}{\frac{\beta nc}{\ell}e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}}-1\right)^2}{2} \overset{(b)}{\approx}\\
&\quad -\frac{1}{2\frac{\beta nc}{\ell}e^{-\frac{n'_F\frac{\ell}{n'}\cdot\delta^2}{2}}},
\end{aligned}$$

where (a) follows from (145), and (b) follows from (126) and the fact that $n'_f=O(n')$. ■

A block is *pending* if it is waiting to be read from or written to some device. The number of pending blocks affects the effective number of failures per round.

Lemma 9: Let m be the total number of system-wide pending blocks in steady state. Then

$$m\lessapprox n\left(\lambda\beta c t_s+\frac{e^{\frac{\Lambda}{\Xi}}-1}{2-e^{\frac{\Lambda}{\Xi}}}\right)\approx n\lambda\left(t_s+\frac{k+1}{\mu r}\right)\beta c, \qquad (147)$$

where

$$\begin{aligned}\Lambda &\approx \lambda\beta c\left(k+1\right), \\ \Xi &= \mu r.\end{aligned} \tag{148}$$

Proof: The inter-failure time of devices from S is distributed. With high probability, the number of failed functioning devices in an epoch approaches $\left(1 \pm o\left(1\right)\right)\lambda n s t_s$; the number of functioning devices is always in the approximate range $\left[n - n\lambda s t_s, n\right]$. It follows that the failure rate of functioning devices normalized by the remaining number of functioning devices can be upper bounded by

$$\left(1 \pm o\left(1\right)\right)\frac{\lambda n s t_s}{\left(n - n\lambda s t_s\right) s t_s}. \tag{149}$$

The failure of each failed functioning-device's block generates k read requests approximately uniformly distributed among functioning devices. Since the inter-failure time of failed devices is exponential, and a random splitting of an exponential process is itself an exponential process [66], the read-request process per device is approximately distributed exponentially with rate

$$\frac{\lambda n\beta c s t_s k}{\left(n - n\lambda s t_s\right) s t_s}. \tag{150}$$

The failure of each failed functioning-device's block generates a single write requests and possibly reassignment requests. By the same reasoning, the process generated by these requests per functioning device is approximately distributed exponentially, with rate

$$\frac{\lambda n\beta c s t_s k}{\left(n - n\lambda s t_s\right) s t_s}\frac{1}{k(1-\gamma)} \overset{(a)}{\approx} \frac{\lambda n\beta c s t_s}{\left(n - n\lambda s t_s\right) s t_s}, \tag{151}$$

where

$$\beta' = \beta + 2\frac{n_R\beta}{n} = \beta\left(1 + 2\lambda s t_s + 2(\lambda s t_s)^2\right), \tag{152}$$

$$\gamma = \left(\frac{2}{\beta'\left(\frac{1}{\beta'} - 1\right)^3 c}\right)^{\frac{1}{3}}, \tag{153}$$

and (a) follows from the fact that c is large.

To maintain equilibrium between the contraction and expansion processes, the number of replacement devices should be approximately

$$\left(n\lambda s t_s + n\left(\lambda s t_s\right)^2\right), \tag{154}$$

where the first term is due to the expected number of failed devices from S, and the second term is due to the expected number of failed replacement devices. By the same reasoning as above, the average rate of requests per functioning device due to the expansion process is

$$\frac{\left(n\lambda s t_s + n\left(\lambda s t_s\right)^2\right)\beta c}{\left(n - n\lambda s t_s\right) s t_s}, \tag{155}$$

which by (128) is negligible in comparison to (150) or (151).

Since the sum process of two exponential processes is exponential [66], the load on each functioning device can be modelled by an M/D/1 queue. The birth rate is given by the sum of (150) and (151), which by (128) is approximately $\lambda\beta c(k+1)$. The death rate is μr.

In Section VII we deal with bounds on such queues. Corollary 1 of Subsection VII-D gives the steady state distribution of each queue, given the birth and death rates. Using this and the fact that the lengths of the queues are negatively dependent, we obtain (147). ■

C. Simulation Results

In this subsection we show simulation results for the random-sparing scheme:

- Figure 8 shows the single-layer storage-rate and maximal fraction of fixable nodes as a function of left and right edge-degrees (derived in (87) and (82) respectively). As noted, the rate of this code is sub-optimal, necessitating the use of the augmented code from Subsection IV-C.
- We have performed 500 tests on the block error-probability. In each test a random code with $m = 10000$ left nodes, $\hat{q} = 4$, $\hat{a} = 4$, $\hat{\alpha} = \frac{1}{3}$, and 8236 right nodes was first built. Then, 10^6 iterations were performed. In each iteration 3900 left nodes were initially set to be corrupted, and were then attempted to be fixed. The code succeeded in every single iteration of every single test.
- Figures 9 and 10 show the number of updated nodes as a function of the fraction of modified left nodes, for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes. Note that as predicted, simulation results coincide with the expected value, given by (85). For small fractions of left nodes being updated, the number of update-IOs can be approximated as the number of updated left nodes times average left degree (a_l).
- Figure 11 shows the number of nodes, accessed by the recovery procedure, as a function of the fraction of failed left nodes, for the case $\hat{q} = 4$, $\hat{a} = 7$, and $m = 10004$ left nodes. Note that the assumption of a accesses per each failed node gives the upper bound on the number of accessed nodes, as shown in (84).

D. Discussion

In this section we have shown how to create a level employing random sparing. This scheme appears to be similar to one independently proposed in OceanStore [35]. From Theorem 13, though, we deduce that the storage rate for small and large coding-groups must differ, as opposed to the uniform coding-rate in OceanStore.

The scheme relies on the use of codes with recovery locality. It was previously shown that codes with good

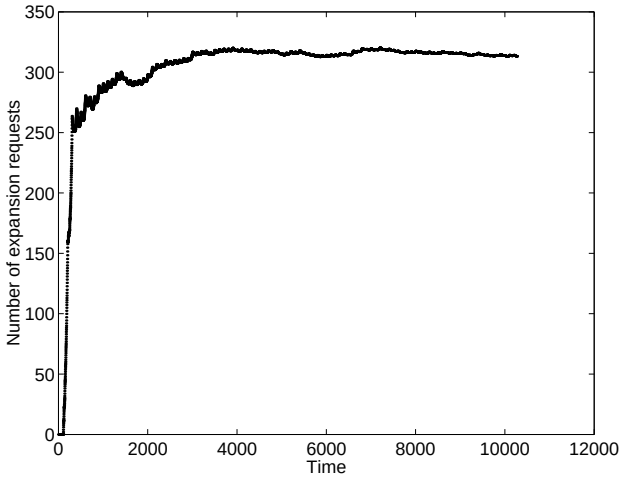

Fig. 12. num bal r reqs as f of time.

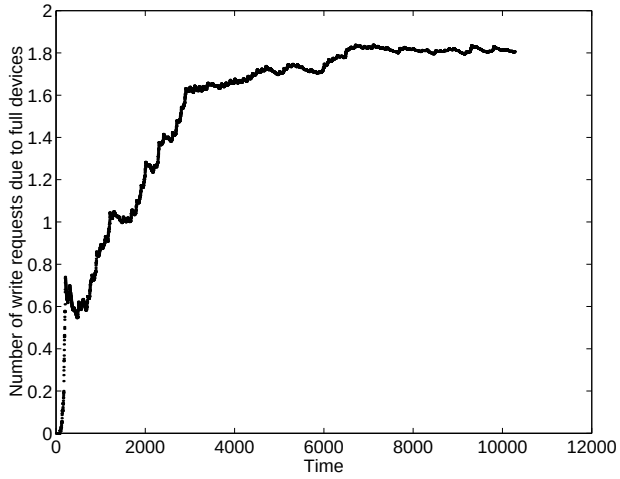

Fig. 13. num cong as f of time.

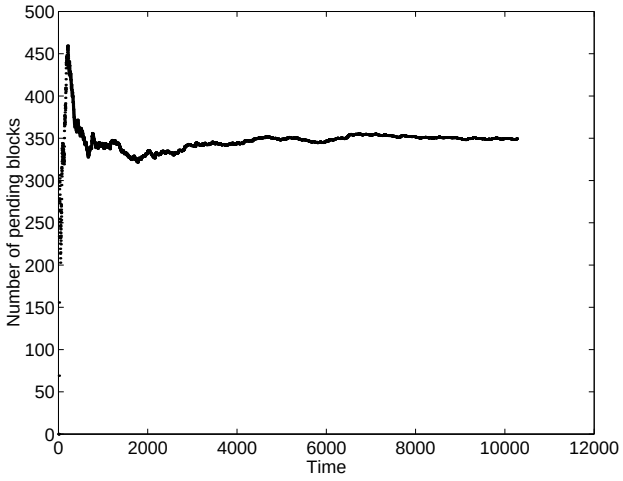

Fig. 14. num int mem as f of time.

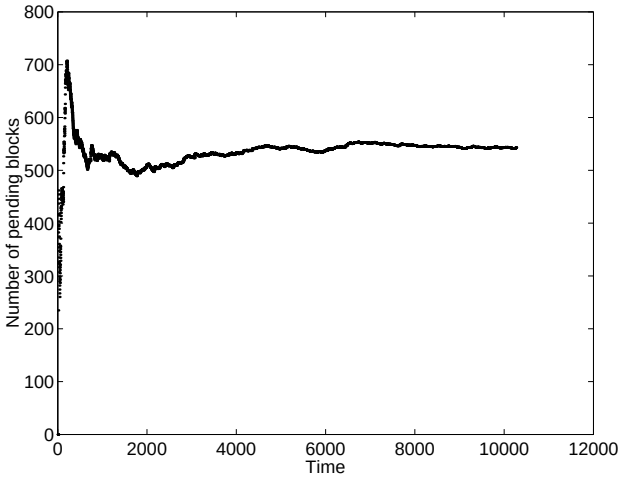

Fig. 15. num pend blks as f of time.

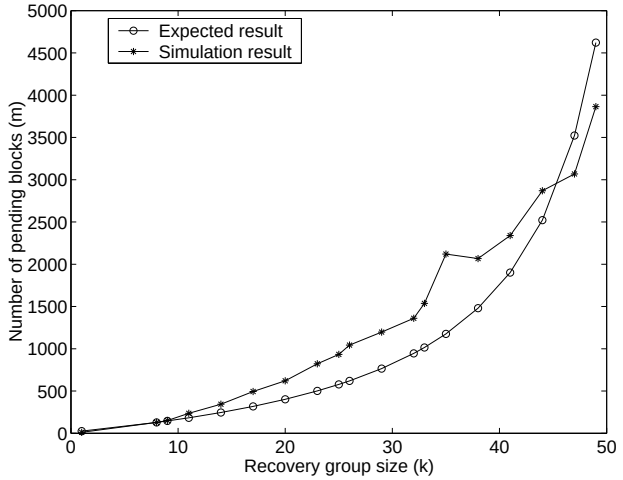

Fig. 16. p blcks f of k.

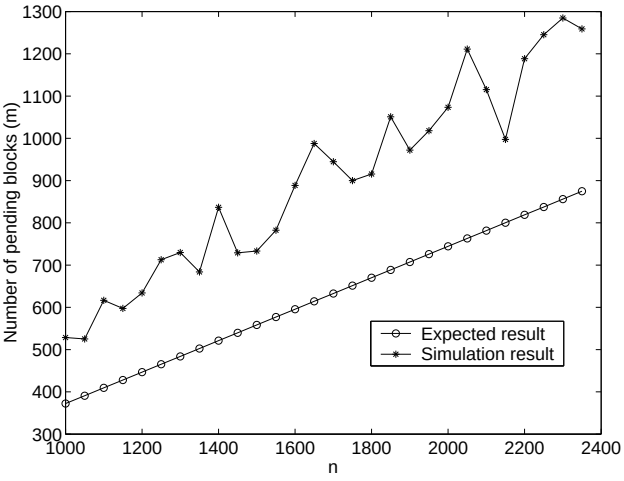

Fig. 17. p blcks f of n.

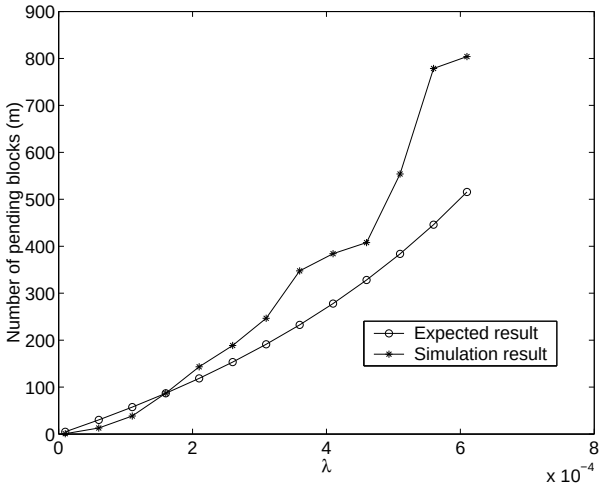

Fig. 18. p blcks f of lambda.

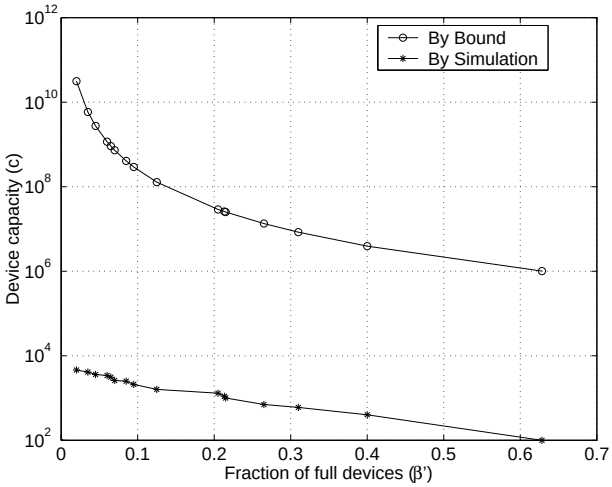

Fig. 21. full devs as f of beta tag.

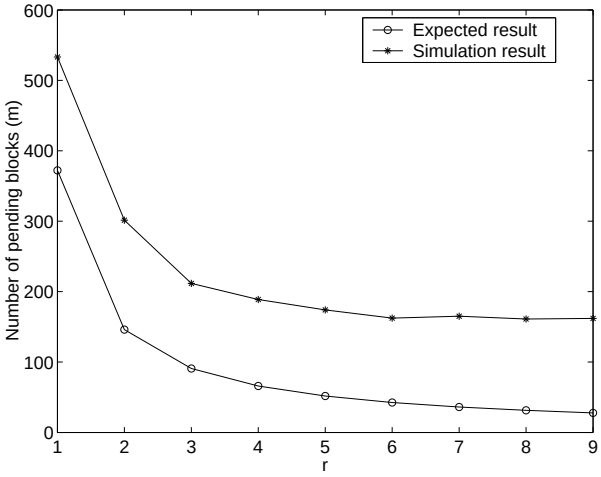

Fig. 19. p blcks f of r.

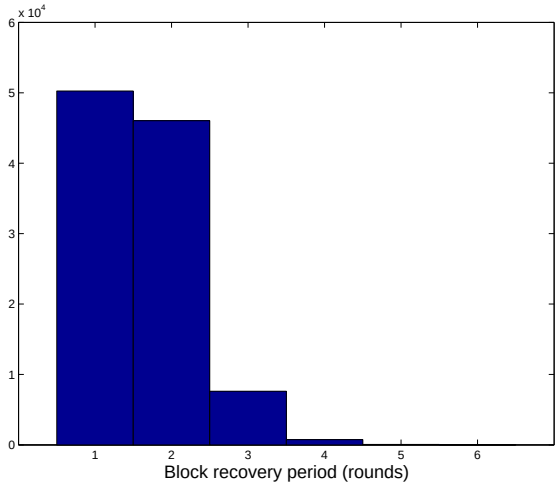

Fig. 22. block in rec time rounds hist.

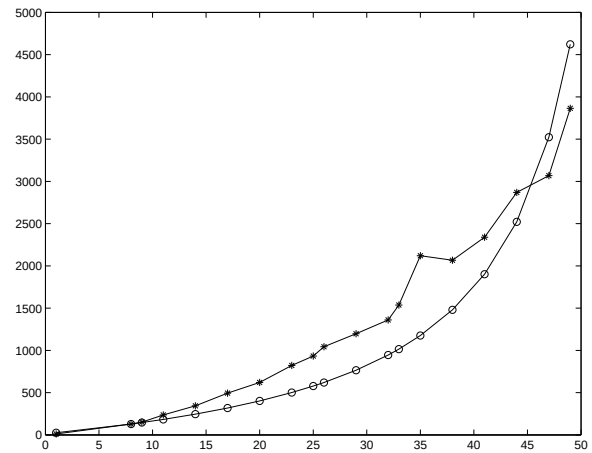

Fig. 20. p blcks f of rec grp sz.

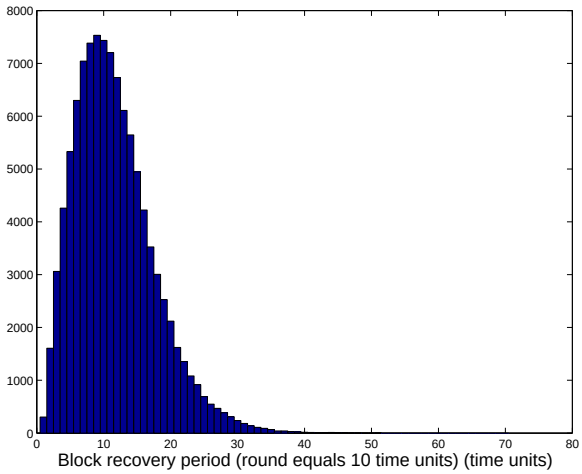

Fig. 23. Recovery histogram ticks.

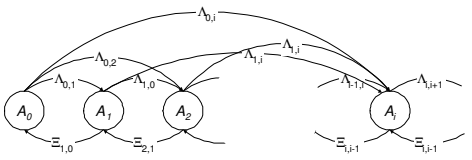

Fig. 24. Discrete-time Markov-graph of failed devices.

update locality have bad recovery locality, and vice versa [57]. While this is true when considering a single code, the conclusion is somewhat different when considering random sparing. The analysis in Lemma 9 shows that if codes do not have recovery locality, the load on each device-queue grows indefinitely. It follows that the redundancy of each coding group must grow as well. In this case, the update performance must degrade.

VII. High-Intensity Data

For high-intensity data, high performance codes should be used. Assume a group of size n is coded in an MDS code of storage rate $\frac{1}{n}$. One such code is the repetition code, wherein an item is replicated n times. In the setting of storage, the repetition code is usually termed *(multi-way) mirroring*. The repetition code does not require a read-modify-write sequence of operations to all redundant items, when a data item is modified. It is easy to show that it is the only such code (up to trivial variations). This makes the repetition code a good candidate for use in high intensity data level.

A possible semi-Markov modelling of the system is shown in Figure 24. The model consists of $n+1$ states, $A_0, \ldots, A_n$. The state A_i indicates that i devices are failed. The model begins in state A_0; and terminates in state A_n, when data is lost. The various $\Lambda_{i,j}$ and $\Xi_{j,i}$ indicate the transition rates between states, and we define them later on (they do not have the exact interpretation of the corresponding M/M/1 queue values).

At each discrete multiple of t_d, the system detects the set of failed devices. Let A_i and A_j be arbitrary states, where $i \lneq j$. A transition from A_i to A_j takes place when $j-i+1$ devices fail in time t_d. Note that neglecting the occurrence of two simultaneous events, valid in the M/M/1 queue, is no longer valid. A transition from state A_j to state A_i takes place when $j - i + 1$ devices have been recovered. The recovery of a device is performed by reading a copy of the data from operating devices, and writing it to a replacement device (and therefore requires $\frac{c}{r}$ time).

Note that the assumption that the model state completely determines the system state, valid in the M/M/1 queue, is not valid. The transition from A_j to A_i depends on the length of time the system has been in state A_j, as well as the path by which A_j has been reached.

In general, a more complicated setting may be considered, where the number of *copies* is considered, rather than the number of devices. Exploiting recovery-parallelism, the number of copies recovered in a given time may depend on the number of existing copies. We do not analyze this setting in this work.

The use of birth-death processes for finding the MTTDL has been previously used for the case where the number of states is small [4], [41], [50]. When the number of states is large, the ability to recover many devices in parallel, is lost in the model. Also, as noted above, the exponential distribution does not coincide with the deterministic recovery-time, nor does it take into account the detection latency.

The section is organized as follows. In Subsection VII-A we model the system in Figure 24 in a way in which is amenable for finding a lower bound on the MTTDL. In Subsection VII-B we show a solution to the model in Subsection VII-A. In Subsection VII-C we show an improved approximation for the lower bound. In Subsection VII-D we use the same ideas to find a simple bound on steady-state distributions for the M/D/1 queue, a result needed for Section VI. In Subsection VII-E we show simulation results on the MTTDL of multi-way mirroring.

A. Modelling the System State

The system in Figure 24 shows the model from the viewpoint of the system controller. In a time interval t_d, many devices can fail. Devices can be in different stages of recovery. This means that each state can be reached from many different states. The resulting system of equations is complex. Rather than solving it directly, we consider a different system and viewpoint.

We consider a system composed of n devices, with failure CDFs $1 - e^{-\hat{\lambda}_1 t}, \ldots, 1 - e^{-\hat{\lambda}_n t}$ $(t \geq 0)$, respectively. We will require in particular two cases: the case $n = 2$ with arbitrary $\hat{\lambda}_1$ and $\hat{\lambda}_2$, and the case of an arbitrary n with $\hat{\lambda}_1 = \cdots = \hat{\lambda}_n = \hat{\lambda}$. We consider the same failure-detection latency and recovery time as in the original system. If a device fails during the recovery of some other devices, the recovery process is aborted and re-initiated. An imaginary observer with zero-delay knowledge of the state of each device observes the system. We consider the view of this observer. The model then becomes that in Figure 25.

In this model, it can be shown that neglecting two simultaneous events is valid. Note that a solution for this model is a lower bound on the solution of a original system in which device failure during recovery does not re-initiate the recovery process.

Let $A(t)$ be the state at time t. Let

$$\begin{aligned} p_{i,j}(h+t,t) &= \mathbf{P}\left(A(h+t) = A_j \mid A(t) = A_i\right), \\ p_i(h+t,t) &= p_{0,i}(h+t,t). \end{aligned} \tag{156}$$

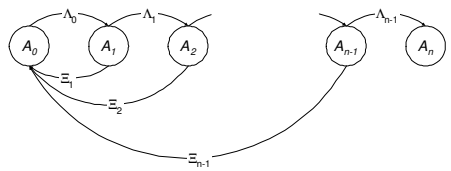

Fig. 25. Markov-graph of failed devices.

As opposed to an M/M/k type of setting, it is not possible to precisely define here

$$\begin{aligned} p_{i,j}(h) &= p_{i,j}(h+t,t), \\ p_i(h) &= p_{0,i}(h+t,t), \end{aligned} \tag{157}$$

since the model state alone does not determine the system state. By equating the transition rates between states, we show that (157) can hold as an approximation, and be found by a system of approximate differential equations.

Definition 10: Let

$$\Xi(\Lambda) = \Lambda\left(\frac{1}{1-e^{-\Lambda\frac{c}{r}}\frac{1-e^{-\Lambda t_d}}{\Lambda t_d}}-1\right). \tag{158}$$

Let $p_i(t) = p_i(t,\hat{\lambda}_1,\ldots,\hat{\lambda}_n,\frac{c}{r},t_d)$ be defined by the system of differential equations:

$$\begin{aligned} &p_i'(t) = \\ &\quad \Lambda_{i-1}p_{i-1}(t) + \Xi_{i+1}p_{i+1}(t) - (\Lambda_i+\Xi_i)p_i(t), \\ &p_0(0) = 1, \\ &p_1(0) = \cdots = p_n(0) = 0, \end{aligned} \tag{159}$$

for $i \in \{0,\ldots,n\}$, and Λ_i and Ξ_i defined as:

- For the case of system of two devices:

$$\begin{aligned} &\Lambda_0 = \hat{\lambda}_0 + \hat{\lambda}_1, \\ &\Lambda_1 = \frac{\hat{\lambda}_1\hat{\lambda}_2\left(\hat{\lambda}_1+\hat{\lambda}_2+\Xi(\hat{\lambda}_1)+\Xi(\hat{\lambda}_2)\right)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}, \\ &\Xi_1 = \\ &\quad \frac{\hat{\lambda}_1^2\Xi(\hat{\lambda}_2)+\hat{\lambda}_2^2\Xi(\hat{\lambda}_1)+\hat{\lambda}_1\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)+\hat{\lambda}_2\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}, \\ &\Lambda_{-1} = \Lambda_2 = \Xi_0 = \Xi_2 = 0. \end{aligned} \tag{160}$$

- For the case of n identical devices:

$$\begin{aligned} &\Lambda_i = \begin{cases} 0 & , \quad j \in \{-1,n\} \\ (n-i)\hat{\lambda} & , \quad \text{otherwise} \end{cases}, \\ &\Xi_i = \\ &\quad \begin{cases} 0 & , \quad i \in \{0,n,n+1\} \\ \Xi(\Lambda_i) & , \quad \text{otherwise} \end{cases}. \end{aligned} \tag{161}$$

By equating transition rates, we show the following.

Theorem 14: The probability of data loss at time t is lower bounded by $p_n(t)$, given by Definition 10.

Proof: In the new model, recovery does not commence immediately once devices fail (due to the detection latency t_d). If it is known that an exponential event occurred up to time $t+t_d$, then the a-posteriori probability of its taking place at any time $t' \in [t,t+t_d]$ is equally likely.

Consider two random variables, t_i^b and t_i^d, with respective PDFs:

$$\begin{aligned} \Lambda_i e^{-\Lambda_i t} &\quad , \quad (t>0), \\ \frac{1}{t_d} &\quad , \quad \left(t \in \left[\frac{c}{r},\frac{c}{r}+t_d\right]\right). \end{aligned} \tag{162}$$

We define a third process $t_i^\ell = \min\left\{t^{d_i}, t^{b_i}\right\}$. The rate of the renewal process generated by t_i^ℓ, is then $\frac{1}{\mathbf{E}\left[t_i^\ell\right]}$. It can be shown that

$$\mathbf{E}\left[t_i^\ell\right] = \frac{1}{\Lambda_i}\left(1-e^{-\Lambda_i\frac{c}{r}}\frac{1-e^{-\Lambda_i t_d}}{\Lambda_i t_d}\right). \tag{163}$$

Each renewal in the process generated by t^{l_i} is caused by either t^{b_i} or t^{d_i}. We define the probabilities

$$\begin{aligned} p_i^b &= \mathbf{P}\left(\min\left\{t_i^d,t_i^b\right\} = t_i^b\right), \\ p_i^d &= \mathbf{P}\left(\min\left\{t_i^d,t_i^b\right\} = t_i^d\right). \end{aligned} \tag{164}$$

It can be shown that

$$p_i^b = e^{-\Lambda_i\frac{c}{r}}\frac{1-e^{-\Lambda_i t_d}}{\Lambda_i t_d}, \tag{165}$$

$$p_i^d = 1-e^{-\Lambda_i\frac{c}{r}}\frac{1-e^{-\Lambda_i t_d}}{\Lambda_i t_d}. \tag{166}$$

By using the standard technique [66] of equating the entry and exit rate of each state using the Chapman-Kolmogorov equations, it can be shown that in this case,

$$p_{i,j}'(t) = \sum_{k\neq j} q_{k,j}p_{i,k}(t) - \nu_j p_{i,j}(t) \tag{167}$$

holds approximately, where ν_j is the rate of the renewal process generated by t_j^ℓ, and $q_{i,k}$ is the rate of the renewal process generated by transferring from A_i to A_k.

It follows that we can insert into (167) the following:

$$\begin{aligned} q_{i,i+1} &= \Lambda_i = p_i^b\frac{1}{\mathbf{E}\left[t_i^\ell\right]}, \\ q_{i,i-1} &= \Xi_i = p_i^d\frac{1}{\mathbf{E}\left[t_i^\ell\right]}, \\ \nu_i &= \Lambda_i+\Xi_i = \frac{1}{\mathbf{E}\left[t_i^\ell\right]}. \end{aligned} \tag{168}$$

It now remains to find the various Λ_i and Ξ_i:

- For the system of 2 devices, the failure rate when in state A_0 is clearly $\Lambda_0 = \hat{\lambda}_1+\hat{\lambda}_2$. With probability $\frac{\hat{\lambda}_1}{\hat{\lambda}_1+\hat{\lambda}_2}$, the

device with $\hat{\lambda}_2$ fails first. In this case, $\Lambda_1 = \hat{\lambda}_2$ and $\Xi_1 = \Xi(\hat{\lambda}_2)$. Similarly, with probability $\frac{\hat{\lambda}_2}{\hat{\lambda}_1+\hat{\lambda}_2}$, $\Lambda_1 = \hat{\lambda}_1$ and $\Xi_1 = \Xi(\hat{\lambda}_1)$. By applying the Bayes rule, we get:

$$\mathbf{E}\left[t_1^{\ell}\right] = \frac{\hat{\lambda}_1}{\hat{\lambda}_1+\hat{\lambda}_2}\frac{1}{\hat{\lambda}_2+\Xi(\hat{\lambda}_2)} + \frac{\hat{\lambda}_2}{\hat{\lambda}_1+\hat{\lambda}_2}\frac{1}{\hat{\lambda}_1+\Xi(\hat{\lambda}_1)} = \frac{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}{\left(\hat{\lambda}_1+\Xi(\hat{\lambda}_1)\right)\left(\hat{\lambda}_1+\hat{\lambda}_2\right)\left(\hat{\lambda}_2+\Xi(\hat{\lambda}_2)\right)}, \quad (169)$$

$$p_i^b = \frac{\hat{\lambda}_1\hat{\lambda}_2\left(\hat{\lambda}_1+\hat{\lambda}_2+\Xi(\hat{\lambda}_1)+\Xi(\hat{\lambda}_2)\right)}{\left(\hat{\lambda}_1+\Xi(\hat{\lambda}_1)\right)\left(\hat{\lambda}_1+\hat{\lambda}_2\right)\left(\hat{\lambda}_2+\Xi(\hat{\lambda}_2)\right)},$$

$$p_i^d = \frac{\left(\hat{\lambda}_1^2\Xi(\hat{\lambda}_2)+\hat{\lambda}_2^2\Xi(\hat{\lambda}_1)+\hat{\lambda}_1\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)+\hat{\lambda}_2\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)\right)}{\left(\hat{\lambda}_1+\Xi(\hat{\lambda}_1)\right)\left(\hat{\lambda}_1+\hat{\lambda}_2\right)\left(\hat{\lambda}_2+\Xi(\hat{\lambda}_2)\right)}.$$

Combining (168) and (169), we have

$$\Lambda_1 = \frac{\hat{\lambda}_1\hat{\lambda}_2\left(\hat{\lambda}_1+\hat{\lambda}_2+\Xi(\hat{\lambda}_1)+\Xi(\hat{\lambda}_2)\right)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}, \quad (170)$$

$$\Xi_1 = \frac{\hat{\lambda}_1^2\Xi(\hat{\lambda}_2)+\hat{\lambda}_2^2\Xi(\hat{\lambda}_1)+\hat{\lambda}_1\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)+\hat{\lambda}_2\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}.$$

• For the system of n identical devices, when in state A_i, the time to device failure is clearly distributed exponentially with rate $\Lambda_i = (n-i)\hat{\lambda}$. Ξ_i can be found as before. ■

B. Estimating the MTTDL

In this subsection we modify the method in [42] to solve (159) for the MTTDL.

Theorem 15: Assume that

$$\frac{1}{\frac{c}{r}+\frac{t_d}{2}} \gg n\lambda. \quad (171)$$

Then the MTTDL of the system in Figure 25 is approximately lower bounded by

$$\sum_{j=1}^{n}\frac{1}{\Lambda_{j-1}}\prod_{i=j}^{n-1}\left(1+\frac{\Xi_i}{\Lambda_i}\right) \approx \frac{\left(\frac{1}{\frac{c}{r}+\frac{t_d}{2}}\right)^{n-1}}{n!\lambda^n}, \quad (172)$$

where $\Xi_i = \Xi(\Lambda_i)$.

Proof: For the system in Figure 25, the equation system (159) becomes the following:

$$\begin{aligned} p_0'(t) &= -\Lambda_0 p_0(t) + \sum_{i=1}^{n-1}\Xi_i p_i(t), \quad (173)\\ p_j'(t) &= -\left(\Xi_j+\Lambda_j\right)p_j(t) + \Lambda_{j-1}p_{j-1}(t),\\ p_n'(t) &= \Lambda_{n-1}p_{n-1}(t),\\ p_1(0) &= 1, \end{aligned}$$

where $j \in [n-1]$.

Using the Laplace transform:

$$\begin{aligned} \int_0^\infty p_j(t)\,e^{-st}\,dt &= a_j(s), \quad (174)\\ \int_0^\infty p_j'(t)\,e^{-st}\,dt &= -p_j(0) + sa_j(s), \end{aligned}$$

we obtain the equation system:

$$\begin{aligned} (-s-\Lambda_0)a_0(s) + \sum_{i=1}^{k-1}\Xi_i a_i(s) = -p_0(0) &= -1, \quad (175)\\ (-s-\Xi_j-\Lambda_j)a_j(s) + \Lambda_{j-1}a_{j-1}(s) = -p_j(0) &= 0,\\ -sa_n(s) + \Lambda_{n-1} &= 0. \end{aligned}$$

Note that once the system enters A_n, it will not leave it. It follows that defining the MTTDL by $\bar{t}_n$, we have

$$\bar{t}_n = \int_0^\infty R_S(t)\,dt = \int_0^\infty tp_n'(t)\,dt. \quad (176)$$

By properties of the reverse Laplace-transform, this is

$$\bar{t}_n = -\left.\frac{d\,(sa_n(s))}{d\,s}\right|_{s=0}. \quad (177)$$

We therefore need to solve the equation system (175) for $a_n(s)$. To do so, we apply Cramer's rule, obtaining

$$a_n(s) = \frac{g_n(s)}{g(s)}, \quad (178)$$

where

$$g(s) = \Delta\begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} & \\ \Lambda_0 & -R_1^O & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \Lambda_{n-2} & -R_{n-1}^O & \\ & & & & \Lambda_{n-1} & -s \end{pmatrix}, \quad (179)$$

$$g_n(s) = \Delta\begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} & -1 \\ \Lambda_0 & -R_1^O & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \Lambda_{n-2} & -R_{n-1}^O & \\ & & & & \Lambda_{n-1} & \end{pmatrix}, \quad (180)$$

and

$$R_i^O = \Lambda_i + \Xi_i + s. \quad (181)$$

We first develop $g(s)$ and $g_n(s)$. For the former, we have

$$g(s) = -s\gamma_n(s), \quad (182)$$

where

$$\gamma_n(s) = \Delta \begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} \\ \Lambda_0 & -R_1^O & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \Lambda_{n-2} & -R_{n-1}^O \end{pmatrix}. \quad (183)$$

For the latter, we have

$$g_n(s) = (-1)^{n+1} \prod_{i=0}^{n-1} \Lambda_i. \quad (184)$$

Inserting (182) and (184) into (178) and (177), we have

$$\bar{t}_n = \frac{g_n'(0) + \gamma_n'(0)}{\gamma_n(0)}. \quad (185)$$

In the matrix within the determinant in (179), we may add all rows to the last without altering $g(s)$. To the matrix within the determinant in (180), we may append an all-0 column and the row $-s, -s, \dots, -1$, without altering $g_n(s)$. After doing so, we have

$$g_n(s) + \gamma_n(s) = \Delta \begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} & -1 \\ \Lambda_0 & -R_1^O & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \Lambda_{n-2} & -R_{n-1}^O & \\ -s & -s & -s & -s & -s & -1 \end{pmatrix} - \Delta \begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} & 0 \\ \Lambda_0 & -R_1^O & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \Lambda_{n-2} & -R_{n-1}^O & \\ -s & -s & -s & -s & -s & -1 \end{pmatrix} \quad (186)$$

$$\stackrel{(a)}{=} \Delta \begin{pmatrix} -R_0^O & \Xi_1 & \Xi_2 & \dots & \Xi_{n-1} & -1 \\ \Lambda_0 & -R_1^O & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \Lambda_{n-2} & -R_{n-1}^O & \\ -s & -s & -s & -s & -s & 0 \end{pmatrix} \quad (187)$$

where (a) follows from the fact that the determinant is distributive over matrices identical in all rows but one.

We note that the determinant in (187) is

$$s(-1)^n B_n(s), \quad (188)$$

where

$$B_n(s) = \Delta \begin{pmatrix} \Lambda_0 & -R_1^O & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \Lambda_{n-2} & -R_{n-1}^O \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \quad (189)$$

Furthermore, the following recurrence system is satisfied at $s = 0$:

$$\begin{aligned} B_k(0) &= \prod_{i=0}^{k-2} \Lambda_i + (\Lambda_{k-1} + \Xi_{k-1})\, B_{k-1}(0), \quad (190) \\ B_1(0) &= 1. \end{aligned}$$

Solving (190), we obtain

$$B_k(0) = \sum_{j=1}^{k} \prod_{i=0}^{j-2} \Lambda_i \prod_{i=j+1}^{k} (\Lambda_{i-1} + \Xi_{i-1}). \quad (191)$$

Setting $s = 0$ in (183), we have

$$\gamma_n(0) = (-1)^{n+1} \prod_{i=0}^{n-1} \Lambda_i. \quad (192)$$

Inserting (191) and (192) into (185), we get

$$\begin{aligned} \bar{t}_n = & \quad (193) \\ & \frac{1}{\gamma_n(0)} \left. \frac{d\,(g_n(s) + \gamma_n(s))}{d\,s} \right|_{s=0} = \\ & (-1)^k \frac{B_n(0)}{\gamma_n(0)} = \\ & \sum_{j=1}^{n} \frac{1}{\Lambda_{j-1}} \prod_{i=j}^{n-1} \left(1 + \frac{\Xi_i}{\Lambda_i}\right). \end{aligned}$$

By applying (171) to (158), we have

$$\min_i \left\{ \frac{\Xi_i}{\Lambda_i} \right\} \gg 1, \quad (194)$$

and (193) simplifies to

$$\bar{t}_n \approx \frac{\prod_{i=1}^{n-1} \Xi_i}{\prod_{i=0}^{n-1} \Lambda_i}. \quad (195)$$

This can be further simplified by substituting

$$\begin{aligned} \Lambda_i &= (n-i)\lambda, \quad (196) \\ \Xi_i &\stackrel{(a)}{\approx} \left(\frac{1}{\frac{c}{r} + \frac{t_d}{2}} \right), \end{aligned}$$

where (a) follows from the application of (171) to (158). ■

C. Improving the MTTDL Estimation

The system model from Subsection VII-B assumes that any device failure while recovery, restarts the recovery process. This is needed for the definition of the state-transition rates. This is clearly a drawback. The MTTDL found in Section VII-B therefore only lower bounds the true MTTDL. In this subsection we attempt to rectify this by considering "compound devices", *i.e.*, devices composed of a sub-group of devices.

The main point we prove in this subsection is the following.

Theorem 16: Assume that

$$\frac{1}{\frac{c}{r}+\frac{t_d}{2}} \gg 2\lambda. \tag{197}$$

Then the MTTDL of the system in Figure 25 is approximately

$$\frac{\left(\frac{1}{\frac{c}{r}+\frac{t_d}{2}}\right)^{n-1}}{2^{n-1}\lambda^n}. \tag{198}$$

We first analyze the failure-time distribution for the general case of two devices.

Lemma 10: A system of 2 devices whose failure times have respective CDFs $1-e^{-\hat{\lambda}_1 t}$ and $1-e^{-\hat{\lambda}_2 t}$, has a failure time having an approximate CDF

$$1-e^{-\frac{2\hat{\lambda}_1\hat{\lambda}_2}{\Xi}t} \tag{199}$$

where

$$\Xi = \frac{1}{\frac{c}{r}+\frac{t_d}{2}}, \tag{200}$$

assuming that

$$\frac{1}{\frac{c}{r}+\frac{t_d}{2}} \gg \max\left\{\hat{\lambda}_1, \hat{\lambda}_2\right\}. \tag{201}$$

Proof: For this case, (159) becomes

$$\begin{aligned} p_0'(t) &= \Xi_1 p_1(t) - \Lambda_0 p_0(t), \\ p_1'(t) &= \Lambda_0 p_0(t) - (\Xi_1+\Lambda_1) p_1(t), \\ p_2'(t) &= \Lambda_1 p_1(t), \end{aligned} \tag{202}$$

where

$$\Lambda_0 = \hat{\lambda}_0 + \hat{\lambda}_1, \tag{203}$$

$$\Lambda_1 = \frac{\hat{\lambda}_1\hat{\lambda}_2\left(\hat{\lambda}_1+\hat{\lambda}_2+\Xi(\hat{\lambda}_1)+\Xi(\hat{\lambda}_2)\right)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)},$$

$$\Xi_1 = \frac{\hat{\lambda}_1^2\Xi(\hat{\lambda}_2)+\hat{\lambda}_2^2\Xi(\hat{\lambda}_1)+\hat{\lambda}_1\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)+\hat{\lambda}_2\Xi(\hat{\lambda}_1)\Xi(\hat{\lambda}_2)}{\hat{\lambda}_1^2+\hat{\lambda}_2^2+\hat{\lambda}_1\Xi(\hat{\lambda}_1)+\hat{\lambda}_2\Xi(\hat{\lambda}_2)}.$$

Solving (202) for $p_0(t)$ and $p_1(t)$, we get

$$\begin{aligned} p_0(t) &= -\frac{r_2+\Lambda_0}{r_1-r_2}e^{r_1 t} + \frac{r_1+\Lambda_0}{r_1-r_2}e^{r_2 t}, \\ p_1(t) &= \frac{(\Lambda_0+r_1)(\Lambda_0+r_2)}{\Xi_1(r_1-r_2)}\left(-e^{r_1 t}+e^{r_2 t}\right), \end{aligned} \tag{204}$$

where

$$r_{1,2} = \frac{-\Lambda_0-\Lambda_1-\Xi_1 \pm \sqrt{(\Lambda_0+\Lambda_1+\Xi_1)^2-4\Lambda_0\Lambda_1}}{2} = \frac{\Lambda_0+\Lambda_1+\Xi_1}{2}\left(-1\pm\sqrt{1-\frac{4\Lambda_0\Lambda_1}{(\Lambda_0+\Lambda_1+\Xi_1)^2}}\right). \tag{205}$$

Using the approximation in (201), we derive the following approximations for $i=1,2$: $\hat{\lambda}_i t_d \ll 1$ and $\hat{\lambda}_i\frac{c}{r} \ll 1$. Applying these approximations to (158), and using the power-series approximation of e^x for small x, we get $\Xi(\hat{\lambda}_2) \approx \Xi(\hat{\lambda}_2) \approx \Xi = \frac{1}{\frac{c}{r}+\frac{t_d}{2}}$. Applying this to (203), we get

$$\Xi_1 \approx \Xi = \frac{1}{\frac{c}{r}+\frac{t_d}{2}}. \tag{206}$$

Applying (206) and using $\Xi \gg \max\{\hat{\lambda}_1, \hat{\lambda}_2\}$ in (203), we get

$$\Lambda_1 \approx \frac{2\hat{\lambda}_1\hat{\lambda}_2}{(\hat{\lambda}_1+\hat{\lambda}_2)} \overset{(a)}{\leq} 2\max\left\{\hat{\lambda}_1, \hat{\lambda}_2\right\}, \tag{207}$$

where (a) follows from the Arithmetic - Geometric Mean inequality. Combining (207) with (206), we get $\frac{\Xi_1}{\max\{\Lambda_1,\Lambda_2\}} \gg 1$. Using this fact, we approximate

$$\begin{aligned} r_1 &\approx -(\Lambda_0+\Lambda_1+\Xi_1) \\ r_2 &\approx -\frac{\Lambda_0\Lambda_1}{\Lambda_0+\Lambda_1+\Xi_1} \approx -\frac{\Lambda_0\Lambda_1}{\Xi_1}. \end{aligned} \tag{208}$$

The lemma follows by the fact that

$$|r_1| \gg |r_2| \Rightarrow e^{r_1 t} \ll e^{r_2 t}. \tag{209}$$

■

We now prove Theorem 16.

Proof: Lemma 10 gives the approximate failure-CDF of a system composed of two devices. From (199) we see that this distribution is exponential with inverse mean

$$\tilde{\lambda} = \frac{2\hat{\lambda}_1\hat{\lambda}_2}{\Xi} \approx 2\hat{\lambda}_1\hat{\lambda}_2\left(\frac{c}{r}+\frac{t_d}{2}\right). \tag{210}$$

Given a system of $n \gneq 2$ devices, we can recursively consider the system as composed of two devices, one simple, and one "compound", both with failure times distributed exponentially. The corresponding state diagram is shown in Figure 26. In general, we can decompose a system of n devices recursively, s.t. each device represents a compound device, an example of which is shown in Figure 27.

When analyzing the reliability of such a system, it is convenient to view it as a *full binary* tree (*i.e.*, a tree where each node has 0 or 2 children), as in diagrams (a) and (b) of

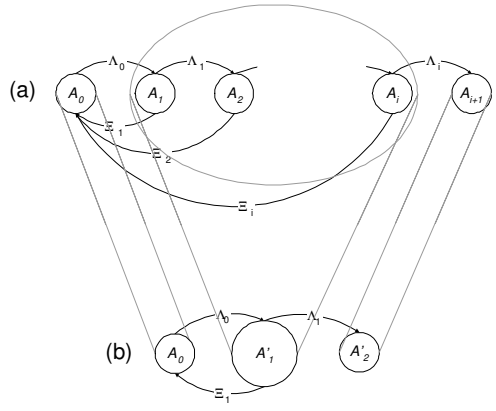

Fig. 26. Reduction to a compound system.

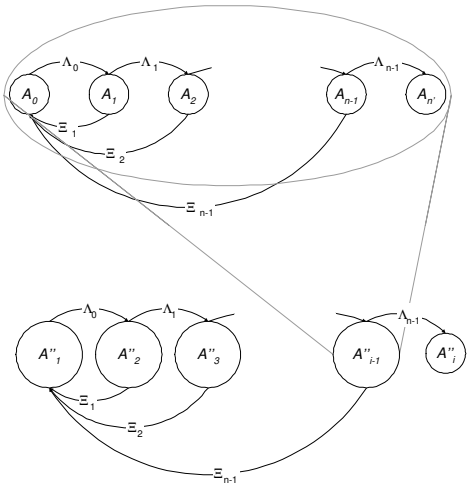

Fig. 27. A recursive analysis.

Figure 28. In the tree of each diagram, any leaf is a device node (indicated by D). Any inner node is a compound node (indicated by C), the reliability of which is determined by its children nodes. The system reliability is that of the root. Lemma 10 shows how to combine the MTTDL of two subtrees. Surprisingly, the MTTDL of the entire tree depends only on the number of leaves, and not on the tree topology chosen.

■

D. Lower Bounds on M/D/1 Steady-State Distributions

In this subsection we find a simple bound on steady-state distributions for the M/D/1 queue. This is needed for Section VI.

For the case of the M/D/1 queue, equating the transition rates yields the following simple lower-bound approximation. In Figure 29 we see a graph corresponding to the

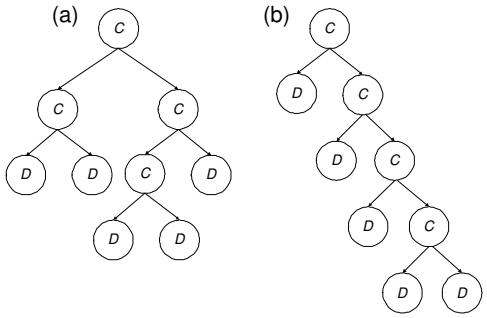

Fig. 28. A tree formed by the recursive analysis

Fig. 29. Markov-graph of birth-death process of failed devices.

M/D/1 process. Specifically, assume the process is a birth-death process, with births generated by a Poisson process with mean interval times $\frac{1}{\lambda}$, and deaths occurring deterministically after $\frac{1}{\xi}$ time.

Corollary 1: For the given M/D/1 queue, let P_i be the steady-state distribution of state A_i. Then

$$P_i = \left(e^{\frac{\lambda}{\xi}} - 1\right)^i \left(2 - e^{\frac{\lambda}{\xi}}\right). \tag{211}$$

Proof: Following [66], we equate the transition rates, resulting, in a manner similar to the one used in Subsection VII-A, in

$$P_i = \begin{cases} \frac{\Xi P_1}{\lambda} & , \quad i = 0 \\ \frac{\Xi P_{i+1} + \lambda P_{i-1}}{\Xi + \lambda} & , \quad i \neq 0 \end{cases}, \tag{212}$$

where

$$\Xi = \lambda \left(\frac{1}{1 - e^{-\frac{\lambda}{\xi}}} - 1 \right) \tag{213}$$

by 158.

The proof follows by solving (212), subject to the constraint $\sum_{i=0}^{\infty} P_i = 1$.

■

E. Simulation Results

In this subsection we show simulation results for multi-way mirroring:

• Figures 30, 31, and 32 show the behavior of a system of two devices (Subsection VII-C) for different settings of $\hat{\lambda}_1$, $\hat{\lambda}_2$, and Ξ. Each of the figures compares the CDFs obtained

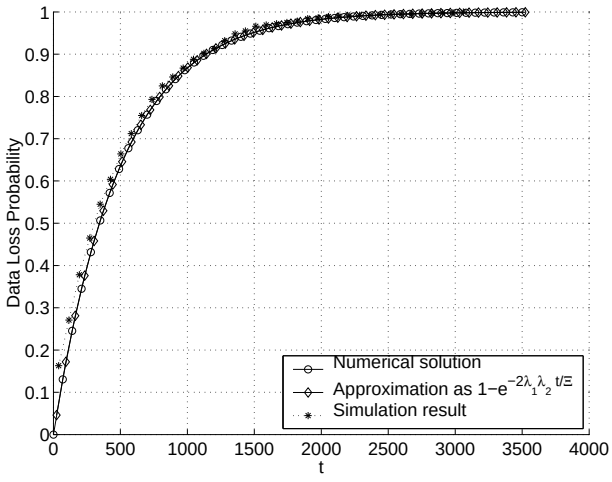

Fig. 30. The CDF of the two-devices system MTTDL, for the case $\hat{\lambda}_1 = \hat{\lambda}_2 = \frac{1}{30}$ and $\Xi = 1$.

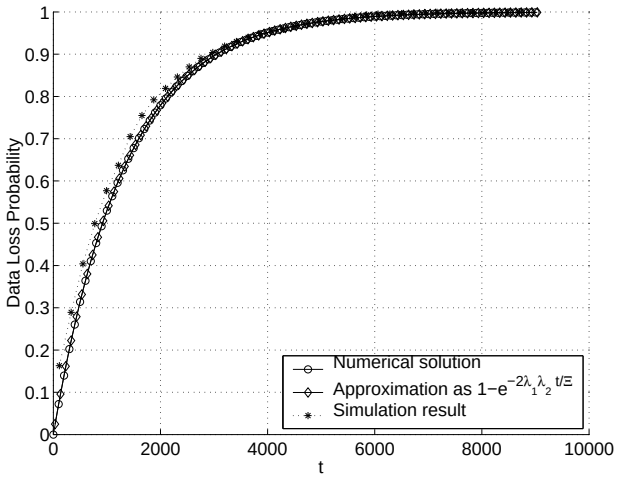

Fig. 31. The CDF of the two-devices system MTTDL, for the case $\hat{\lambda}_1 = \hat{\lambda}_2 = \frac{1}{50}$ and $\Xi = 1$.

by the following: the numerical solution of a differential-equation model of the system, the approximated solution shown in Lemma 10, and the results obtained by a software simulation of the system of devices. Note that the simulation results coincide with the approximation of Lemma 10 for both $\hat{\lambda}_1 = \hat{\lambda}_2$ and $\hat{\lambda}_1 \gg \hat{\lambda}_2$ settings. This result justifies the assumption that the MTTDL of a "compound device" has exponential distribution.

- Figure 33 and 34 show the behavior of a system as a function of the number of devices (n). Figure 33 compares the MTTDLs obtained by the following: the results obtained by a software simulation of the system of devices, the exact and approximated expression from Theorem 15, and the approximation of Theorem 16. Note that Theorem 16, based on the "compound device" approach, achieves the best estimation. Figure 34 compares, for two different settings, the MTTDLs obtained by the following: the results obtained by a software simulation of the system of devices, and the approximation of Theorem 16. Note that Theorem 16 indeed gives a lower bound on the system MTTDL.

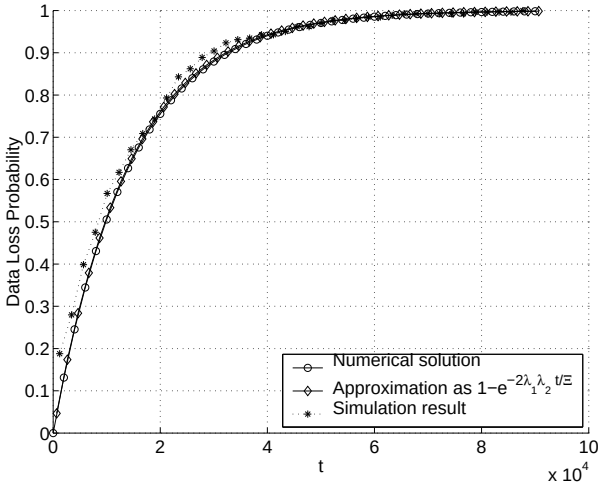

Fig. 32. The CDF of the two-devices system MTTDL, for the case $\hat{\lambda}_1 = \frac{1}{30} \gg \hat{\lambda}_2 = \frac{1}{900}$ and $\Xi = 1$.

VIII. Conclusions and Future Work

In this work we have presented and analyzed codes for storage systems. We have shown that it is easy to construct LDGM codes which have bounded recovery penalty. We have shown algorithms which incorporate codes of this type. Finding bounds and optimal codes subject to bounded penalty constraints, is left to future research. We have also analyzed the distribution of the time to data loss of multi-way mirroring.

The fact that such radically different codes and schemes can be applied to storage systems, suggests that storage reliability should be hierarchical [24]. This in turn raises many questions on the number of levels the hierarchy should contain, the code appropriate to each level, and the transition policy between levels. We leave these interesting questions to future research.

The failure model we considered is one in which entire devices fail. Other errors which occur in practice are those of the erasure and substitution of the content of blocks of devices. The former is very similar to that considered in this paper. The latter presents formidable difficulties, even in terms of detection that an error took place. We leave its treatment to future research.

IX. Acknowledgements

Thanks to Dana Ron and David Burshtein of the Dept. of EE-Systems at Tel-Aviv University, and Alain Azagury, Michael Factor, Kalman Meth, Julian Satran, and Dafna Sheinwald of IBM Haifa Research Laboratories, for useful discussions.

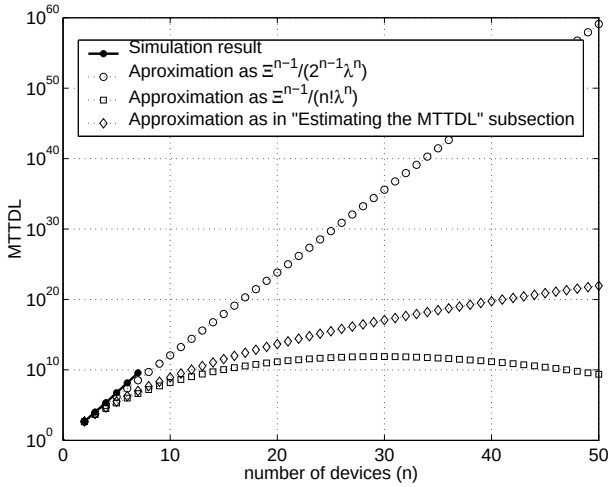

Fig. 33. The system MTTDL as a function of the number of devices (n), for the case $\hat{\lambda}_1 = \ldots = \hat{\lambda}_n = \frac{1}{30}$ and $\Xi = 1$.

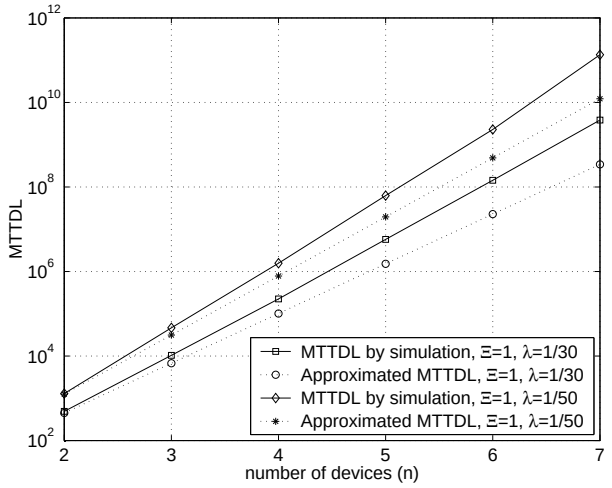

Fig. 34. The system MTTDL as a function of the number of devices (n).

References

[1] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, ser. North-Holland Mathematical Library. North-Holland, 1977, vol. 16.

[2] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission," in *Proceedings: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico*. IEEE Computer Society Press, Nov. 1994, pp. 604–612.

[3] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proceedings of the ACM Conference on Management of Data (SIGMOD)*, June 1988, pp. 109–116.

[4] G. A. Gibson, *Redundant Disk Arrays: Reliable Parallel Secondary Storage*, ser. ACM Distinguished Dissertations. Cambridge, MA: MIT Press, 1992.

[5] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures," in *Proc. of the 21st Symp. on Computer Architecture (21st ISCA'94), Computer Architecture News*. Chicago: ACM SIGARCH, Apr. 1994, pp. 245–254.

[6] C. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[7] R. Gallager, *Discrete Stochastic Processes*. Kluwer Academic Publishers, Boston, 1995.

[8] M. Luby, "LT codes," in *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS-02)*. Los Alamitos: IEEE COMPUTER SOCIETY, Nov. 16–19 2002, pp. 271–282.

[9] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[10] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC '97)*. New York: Association for Computing Machinery, May 1997, pp. 150–159.

[11] A. Shokrollahi, "An introduction to low-density parity-check codes," *Lecture Notes in Computer Science*, vol. 2292, pp. 175–197, 2002.

[12] N. Alon and M. Luby, "A linear time erasure-resilient code with nearly optimal recovery," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1732–1736, Nov. 1996.

[13] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, ser. Wiley Series in Telecommunications. New York, NY, USA: John Wiley & Sons, 1991.

[14] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed. Cambridge, MA, USA: The M.I.T. Press, 1972.

[15] S. Akyürek and K. Salem, "Adaptive block rearrangement," *ACM Transactions on Computer Systems*, vol. 13, no. 2, pp. 89–121, 1995.

[16] M. B. Deshpande and R. B. Bunt, "Dynamic file management techniques," in *Proceedings of 7th IEEE Phoenix Conference on Computers and Communication*. New York: IEEE, 1988, pp. 86–92.

[17] R. A. Floyd and E. C. Ellis, "Directory reference patterns in hierarchical file systems," in *IEEE Trans. Know. Data Eng.*, june.

[18] R. Geist, R. Reynolds, and D. Suggs, "Minimizing mean seek distance in mirrored disk systems by cylinder remapping," *Performance Evaluation*, vol. 20, no. 1–3, pp. 97–114, May 1994.

[19] S. Majumdar, "Locality and file referencing behaviour: Principles and applications," Department of Computer Science, University of Saskatchewan, Saskatoon, Saskatchewan, Tech. Rep., 1984.

[20] M. S. McDonald and R. B. Bunt, "Improving file system performance by dynamically restructuring disk space," in *Proceedings of Phoenix Conference on Computers and Communication*. New York: IEEE, 1989, pp. 264–269.

[21] B. McNutt, "Background data movement in a log-structured disk subsystem," *IBM Journal of Research and Development*, vol. 38, no. 1, pp. 47–58, 1994.

[22] C. Ruemmler and J. Wilkes, "Disk shuffling," Hewlett-Packard Laboratories, Palo Alto, California, Tech. Rep. HPL91156, 1991.

[23] A. J. Smith, "Optimization of i/o systems by cache storage devices and file migration: a summary," *Performance Evaluation*, vol. 1, no. 4, p. 249262, 1981.

[24] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," *ACM Transactions on Computer Systems*, vol. 14, no. 1, pp. 108–136, Feb. 1996.

[25] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, and L. Yerushalmi, "Towards an object store," in *In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003, pp. 165–176.

[26] S. Boucheron and K. Salamatian, "About priority encoding transmission," *IEEE Trans. Inform. Theory*, vol. 46, pp. 697–705, Mar. 2000.

[27] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, no. 4, pp. 685–699, Dec. 1991.

[28] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Comm. ACM*, vol. 28, no. 2, pp. 202–208, Feb. 1985.

[29] L. A. McGeoch and D. D. Sleator, "A strongly competitive randomized paging algorithms," *Algorithmica*, vol. 6, pp. 816–825, 1991.

[30] N. E. Young, "The k-server dual and loose competitiveness for paging," *Algorithmica*, vol. 11, no. 6, pp. 525–541, june 1994.

[31] S. Irani, "Page replacement with multi-size pages and applications to web caching," *Algorithmica*, vol. 33, no. 3, pp. 384–409, July 2002.

[32] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, "New results on server problems," *SIAM Journal on Discrete Mathematics*, vol. 4, no. 2, pp. 172–181, May 1991.

[33] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press.

[34] Y. Neal, "On-line file caching," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 28–30 1998, pp. 82–86.

[35] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 190–201, Nov. 2000.

[36] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz, "Oceanstore: An extremely wide-area storage system," in *Proceedings of the Nine International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, Nov. 2000.

[37] J. Chandy and A. L. N. Reddy, "Failure evaluation of disk array organizations," in *Proceedings of the 13th International Conference on Distributed Computing Systems*. Pittsburgh, PA: IEEE Computer Society Press, May 1993, pp. 319–327.

[38] A. Thomasian and J. Menon, "RAID5 performance with distributed sparing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 6, pp. 640–657, June 1997.

[39] J. M. Menon and R. L. Mattson, "Comparison of sparing alternatives for disk arrays," in *Proceedings the 19th Annual International Symposium on Computer Architecture,ACM SIGARCH*, Gold Coast, Australia, May 1992, pp. 318–329.

[40] J. Menon and D. Mattson, "Distributed sparing in disk arrays," in *Proceedings of the COMPCOM Conference*, Feb. 1992, pp. 410–421.

[41] Q. Xin, E. Miller, D. Long, S. Brandt, T. Schwarz, and W. Litwin, "Reliability mechanisms for very large storage systems," in *In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, Apr. 2003, pp. 146–156.

[42] B. A. Kozlov and I. A. Ushakov, *Reliability Handbook*. New York: Holt, Rinehart and Winston Inc., 1970.

[43] P. Elias, "Coding for two noisy channels," in *Information Theory Third London Symposium*. London: Buterworth's Scientific Publications, Sept. 1955, pp. 61–76.

[44] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, June 1960.

[45] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding:turbo-codes," in *Proceedings of IEEE ICC'93*, vol. 2. Geneva: IEEE, may 1993, pp. 1064–1070.

[46] A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," in *Proceedings of AAECC-13, Lecture Notes in Computer Science 1719*, 1999, pp. 65–76.

[47] J. A. Katzman, "System architecture for nonstop computing," in *14th IEEE Computer Society International Conference (COMPCON)*, Mar. 1977, pp. 77–80.

[48] Q. M. Malluhi and W. E. Johnston, "Coding for high availability of a distributed-parallel storage system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 12, pp. 1237–1252, Dec. 1998.

[49] M. Holland, G. A. Gibson, and D. P. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Journal of Distributed and Parallel Databases*, vol. 2, no. 3, pp. 295–335, July 1994.

[50] X. Wu, J. Li, and H. Kameda, "Reliable analysis of disk array organizations by considering uncorrectable bit errors," in *Proceedings of The 16th Symposium on Reliable Distributed Systems (SRDS '97)*. Washington - Brussels - Tokyo: IEEE, Oct. 1997, pp. 2–9.

[51] R. G. S. Kirkpatrick, W. Wilcke and H. Huels, "Percolation in dense storage arrays," in *Physica A*, vol. 314, Nov. 2002, pp. 220–229.

[52] F. Chang, M. Ji, S.-T. Leung, J. MacCormick, S. Perl, and L. Zhang, "Myriad: Cost-effective disaster tolerance," in *Proceedings of the FAST '02 Conference on File and Storage Technologies (FAST-02)*. Berkeley, CA: USENIX Association, Jan. 28–30 2002, pp. 103–116.

[53] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, H. Jin, T. Cortes, and R. Buyya, Eds. New York, NY: IEEE Computer Society Press and Wiley, 2001, pp. 90–106.

[54] H. Weatherspoon, M. Delco, and S. Zhuang, "Typhoon: An archival system for tolerating high degrees of file server failure," 1999, available through http://www.cs.berkeley.edu/~hweather/Typhoon/TyphoonReport.html.

[55] G. A. Gibson, L. Hellerstein, R. M. Karp, R. H. Katz, and D. A. Patterson, "Failure correction techniques for large disk arrays," *Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 123–132, Apr. 1989.

[56] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding techniques for handling failures in large disk arrays," *Algorithmica*, vol. 12, no. 2/3, pp. 182–208, Aug./Sept. 1994.

[57] Y. Chee, Colbourn, and Ling, "Asymptotically optimal erasure-resilient codes for large disk arrays," *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 102, no. 1-2, pp. 3–36, 2000.

[58] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[59] K. Mulmuley, *Computational Geometry, An Introduction Through Randomized Algorithms*. Prentice-Hall, Inc., New-Jersey.

[60] S. Albers, S. Arora, and S. Khana, "Page replacmenet for general caching problems," in *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 31–40.

[61] J. A. Cooley, J. L. Mineweaser, L. D. Servi, and E. T. Tsung, "Software-based erasure codes for scalable distributed storage," in *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*. IEEE, April 2003, p. 157.

[62] P. Oswald and M. A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, no. 12, pp. 3017–3028, Dec. 2002.

[63] M. Sipser and D. Spielman, "Expander codes," *IEEE Transactions on Information Theory (special issue devoted to coding theory)*, pp. 1710–1722, 1996.

[64] D. Dubashi and D. Ranjan, "Balls and bins: A study in negative dependence," *Random Structures and Algorithms*, vol. 13, no. 2, pp. 99–124, 1998.

[65] T. E. Harris, "A lower bound for the critical probability in a certain percolation process," in *Proc. Cam. Phil. Soc.*, vol. 56, 1960, pp. 13–20.

[66] S. M. Ross, *Introduction to Probability Models*, 8th ed. Harcourt Publishers Ltd, Dec. 2002.