Online Code Rate Adaptation in Cloud Storage Systems with Multiple Erasure Codes

Rui Zhu[1], Di Niu[1], Zongpeng Li[2]

[1] Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada.
Emails: {rzhu3, dniu}@ualberta.ca

[2]Department of Computer Science, University of Calgary, Calgary, Alberta, Canada.
Email: zongpeng@ucalgary.ca

***Abstract*—Erasure codes have been adopted for cloud storage systems. While achieving higher reliability at lower storage overhead as compared to replication, erasure codes usually incur high reading cost when recovering an unavailable block. Although local reconstruction code constructions have been proposed to reduce recovery cost, additional benefits can be achieved by adopting erasure codes with different code rates for data blocks with different popularity. In this paper, we study the problem of code rate selection and adaptation in cloud storage systems that adopt multiple erasure codes via online learning. Unlike offline optimization, which requires the knowledge or estimation of future demands, the online learning algorithms can make decisions only based on past observations and dynamically adapt to demand changes. To avoid solving a hard integer program, we perform a stochastic relaxation to the formulated online learning problem and solve it using a exponentiated gradient algorithm, resulting in sparse solutions. We show a regret bound of $O(\sqrt{T})$ of the proposed algorithm by showing that our algorithm is a special case of the FTRL online learning framework. Through trace-driven simulations based on real request traces from Windows Azure Storage, we show that our online algorithm performs close to the best fixed offline policy, and trades off between recovery cost during degraded reads and storage overhead.**

I. Introduction

Cloud storage systems, e.g., Hadoop Distributed File System (HDFS) [1], Google File System (GFS) [2], Windows Azure Storage (WAS) [3], store large amounts of data for both personal and enterprise-level users based on commodity hardware in datacenters. However, these systems may suffer from node failures and frequent data unavailability due to software glitches, I/O hotspots or local congestions on specific data nodes. To introduce fault-tolerance, the first generation of cloud storage systems (e.g., HDFS) adopts 3-way replication, which replicates each data block on three different data nodes. Recently, erasure coding, e.g., a (k, r) Reed-Solomon (RS) code, has been adopted by many production systems, e.g., Windows Azure Storage, Google's ColossusFS, Facebook's HDFS, to offer higher reliability than replication at a lower storage cost [4], [5].

However, one major weakness of erasure-coded storage systems is the high reconstruction or recovery cost which may lead to long request latency tails, as has been widely reported in previous studies [6]–[10]. For example, to expedite normal reads, most production systems today adopt a *systematic* (k, r) RS code, where a single original copy of each data block is stored in the system. As a result, if the data node storing the original copy of a certain requested block B becomes a hotspot and is temporarily unavailable due to congestion, *degraded reads* must be performed to read k other (original or coded) blocks in the same coded group to reconstruct the content in block B. Apparently, degraded reads can incur k times the traffic of normal reads. Due to the reasons above, most storage systems still adopt 3-way replication to store hot content, and convert erasure-coded storage only when the data becomes "cold", i.e., when the access demand for the data drops down.

A large number of prior studies have focused on designing new coding structures to reduce the number of reads required during reconstruction and degraded reads, e.g., [5], [6], [8], [11], [12]. For example, Local Reconstruction Codes (LRC) [6], [11], [12] have been proposed to reduce the necessary number of blocks to be read during reconstruction, while still keeping a low storage overhead. While all the solutions above trade storage overhead off for lowered recovery cost, it is observed [7] that it is fundamentally inefficient to store all the data blocks using one erasure code—the storage modes for different data blocks should be differentiated depending on their demands. Based on this idea, a storage system based on two erasure codes, HACFS [7], is proposed, where hot data are stored using a code with lower code rate for lower recovery cost during degraded reads (yet with higher storage overhead), and cold data are stored using a higher-rate code for lower storage overhead (yet with higher recovery cost). It is shown that with two erasure codes, HACFS can further improve the recovery performance as compared to single-code storage systems at a comparable storage cost.

However, what is lacking in the literature is an intelligent decision system beyond simple threshold-based decisions to select the right erasure code for different data blocks based on dynamically changing demands, in order to optimize the overall system cost. We generalize the two-erasure-coded system HACFS [7] to the concept of storage systems with multiple codes (including any erasure code and replication as an extreme case), where each code is characterized by a certain storage overhead and a certain recovery cost. The objective is to select the best code for each coded group of data blocks on the go to minimize an aggregate cost of degraded reads and

storage overhead, as the demand changes. In this paper, we propose an online learning framework to select the best code for each coded group only based on the demands observed in the past. In contrast to offline optimization, an online algorithm does not rely on the knowledge or estimates of future demands and can make dynamic decisions in adaptation to demand changes. As selecting the best out of a finite number of codes involves integer programming and is a hard combinatorial problem, we perform a stochastic relaxation for the problem and propose an online learning algorithm based on exponentiated gradient descent to effectively push solutions onto discrete choices. We show that our algorithm is a special case of the Follow-the-Regularized-Leader (FTRL) online learning framework and has a regret bound of $O(\sqrt{T})$, which is the gap between online cost and the cost of the best fixed code selection policy up to time period T, the latter of which makes the unrealistic assumption that demands up to time T are all known.

We evaluate the proposed online code selection algorithms through trace-driven simulations based on real request rates of 252 blocks collected from Windows Azure Storage in a one-day period. We observe highly skewed demands across different blocks, which echo the similar observations made in prior work [7], [9], [10], [13] and justify the differentiation of code rates between data blocks of different hotness to optimize overall recovery and storage cost. The simulations based on the selection between two LRC codes show that our proposed online learning algorithm can achieve a total amount of traffic cost during degraded reads close to that of the impractical offline optimal algorithm, while achieving an even lower storage overhead than the best fixed offline code selection policy.

II. The Online Code Selection Problem

In this section, we describe our model and formulate the online learning problem of dynamic code selection for different coded groups. We follow some conventions for mathematical notations throughout this paper. Vectors and matrices are mostly denoted as capital letters. A vector V is a column vector by default, with its i-th entry denoted by $v[i]$. The entry at the position (i, j) of a matrix A is denoted by $a[i, j]$.

A. Model Description

The content in a typical cloud storage system is stored in *data blocks*. In an erasure-coded storage system, a *coded group* i consists of k original data blocks and r_i parity blocks which are encoded from the k original blocks according to a certain coding construction, e.g, RS, LRC, etc. We assume that the storage system can support a finite set of n erasure codes (possibly of different types), denoted as $\mathcal{C} = \{c_1, \ldots, c_n\}$. The *degraded read* occurs when the data block requested is unavailable and the storage system will reconstruct the content of this block from other available blocks. For each code $c_j \in \mathcal{C}$, we define the *per-block recovery cost* $l[j]$ as the minimum number of reads needed to retrieve an original block during degraded reads in a coded group adopting code c_j. It is worth noting that the per-block recovery cost of c_j is not always k, depending on the code construction. For example, $(12, 6, 2)$ and $(12, 2, 2)$ LRC codes have per-block recovery cost costs of 2 and 6 [6], respectively, while both having the same $k = 12$ yet different numbers of parity blocks. Furthermore, define the *per-group storage overhead* of each code c_j as $s[j] = r_j$. Define the vectors $L := (l[1], l[2], \ldots, l[n])^\top$, and $S := (s[1], s[2], \ldots, s[n])^\top$.

Suppose there are m coded groups. In each time slot $t = 1, 2, \ldots$, we observe a list $D_t = (d_t[1], \ldots, d_t[m])^\top$ of block demands, where $d_t[i]$ denotes the total number of requests for blocks in coded group i. In each time slot t, we need to determine a code selection policy $\pi_t = (\pi_t[1], \ldots, \pi_t[m])$ for all coded groups, where $\pi_t[i] = j$ if and only if coded group i encodes data blocks according to the j-th code $c_j \in \mathcal{C}$. Thus, assuming a degraded read rate of $\epsilon_t[i]$ in each coded group i (which can be measured), the total traffic for degraded reads in the system in time slot t is given by

$$\sum_{i=1}^{m} \epsilon_t[i] d_t[i] \sum_{j=1}^{n} l[j] \mathbb{I}(\pi_t[i] = j), \tag{1}$$

where $\mathbb{I}(\cdot)$ is an identity function that outputs one or zero. In the meantime, we require that the total storage overhead at time t cannot exceed $\mathcal{M}$, i.e.,

$$g_t = \sum_{i=1}^{m} \sum_{j=1}^{n} s[j] \mathbb{I}(\pi_t[i] = j) \leq \mathcal{M}. \tag{2}$$

Since $\epsilon_t[i] d_t[i]$ denotes the number of degraded reads in coded group i at time t and can always be monitored as one entity, in the following, to simply notations, we will just use $d_t[i]$ to represent the degraded reads $\epsilon_t[i] d_t[i]$.

If D_t is known *a priori*, the code selection in each time slot t is to determine π_t given demands D_t and the properties L and S of the available codes, by minimizing (1) subject to (2). However, as we do not know the demands D_t yet when making decisions about π_t, in this paper, we use *online learning* to select codes for each coded group in adaption to demand changes.

B. Online Learning Formulation

We formulate our online learning problem as follows:

(1) in time slot t, the system decides the code selection policy π_t based on data up to $t - 1$;
(2) the system receives and serves the current user demands D_t (amounts of degraded reads in all coded groups);
(3) the system incurs a cost $f_t(\pi_t)$ for the given demands D_t in time slot.
(4) $t := t + 1$; repeats step (1).

In terms of online learning literature, the *regret* is defined as the gap between the *cumulative cost* $\sum_t f_t(\pi_t)$ of online learning and the cumulative cost of the best fixed policy over all the time periods, i.e.,

$$\text{Regret}_T(\mathcal{A}) := \sum_{t=1}^{T} f_t(\pi_t) - \min_{\pi \in \mathcal{A}} \sum_{t=1}^{T} f_t(\pi),$$

where $\mathcal{A}$ is the set of all possible code selection policies. The objective is select the sequence $\pi_1, \ldots, \pi_T$ to bound the regret defined above as T goes to infinity.

Since the code selection $\pi_t[i]$ for group i is a discrete decision to make, minimizing the regret usually involves integer programming and is a hard combinatorial optimization problem. To tackle the integer programming issue, we relax the decisions to be stochastic; that is, we let each coded group i choose each code c_j according to the probability $\mathbb{P}(\pi[i] = j) := \pi_t[i, j]$, where $\sum_j \pi_t[i, j] = 1$. For stochastic policies π_t as a matrix of $\pi_t[i, j]$, we define the total system cost $f_t(\pi_t)$ at time t as the expected total degraded read traffic penalized by the expected storage cost, i.e.,

$$f_t(\pi_t) = \sum_{i=1}^{m} d_t[i] \sum_{j=1}^{n} \pi_t[i, j] l[j] + \frac{\rho}{2}(g_t(\pi_t) - \mathcal{M})^2, \quad (3)$$

where

$$g_t(\pi_t) = \sum_{i=1}^{m} \sum_{j=1}^{n} \pi_t[i, j] s[j] \quad (4)$$

is the total expected storage overhead at time t, and π_t is a matrix of $\pi_t[i, j]$. Now the regret for stochastic decisions is given by

$$\text{Regret}_T(\mathcal{A}) := \sum_{t=1}^{T} f_t(\pi_t) - \min_{\pi \in \mathcal{A}} \sum_{t=1}^{T} f_t(\pi), \quad (5)$$

where $\mathcal{A}$ is the set of all possible stochastic polices.

Unlike deterministic policies which require integer decisions, it is apparent that a stochastic solution only involves fractional solutions, which are easier to handle. In the next section, we propose to use the exponentiated gradient descent algorithm to drive each decision π_t into a sparse matrix, as an approximate solution to the integer code selection problem.

III. An Exponentiated Gradient Descent Algorithm

We present an online algorithm based on exponentiated gradient descent to yield sparse stochastic code selection policies π_t with a regret bound of $O(\sqrt{T})$. The basic idea is to use a *preference matrix* H_t such that a higher value of $H_t[i, j]$ shows more preference on code c_j by coded group i. Then, the probability that coded group i selects the code c_j can be determined by a soft-max distribution, i.e.,

$$\pi_t[i, j] = \frac{\exp(H_t[i, j])}{\sum_{j'=1}^{n} \exp(H_t[i, j'])}. \quad (6)$$

Initially, let all coded groups have the same preferences toward all codes, i.e., for all i and j, $H_1[i, j] = 0$.

Now we update the preference matrix H_t in each time slot t. According to (3), we can see that if the coded group i chooses the codebook c_j, it incurs a loss of

$$\mathcal{L}_{ij}(D_t) := d_t[i] l[j] + \rho s[j](g_t(\pi_t) - \mathcal{M}). \quad (7)$$

Since this entry-wise loss can provide a feedback on how well this action was, we can set $H_t[i, j]$ as follows:

$$H_t[i, j] \leftarrow -\eta \sum_{\tau=1}^{t-1} \mathcal{L}_{i,j}(D_\tau), \quad (8)$$

or, in a recursive way, set $H_t[i, j]$ as follows:

$$H_{t+1}[i, j] \leftarrow H_t[i, j] - \eta \mathcal{L}_{i,j}(D_t), \quad \forall t = 1, 2, \ldots. \quad (9)$$

The algorithm is summarized in Algorithm 1. Note that in each time slot t, the total system cost f_t is directly calculated from the stochastic policy π_t and block demands D_t, not from the actual code selection.

Algorithm 1 The Exponentiated Gradient Descent for Code Selection.

1: **Input**: Parameter $\eta > 0$, $\rho > 0$.
2: Initially, $H_1[i, j] = 0$ for all i and j.
3: **for** $t = 1$ to T **do**
4: Determine π_t by (6).
5: Receive demands D_t and observe the cost $f_t(\pi_t)$.
6: Update preference H_{t+1} according to (9).
7: **end for**

A. Regret Analysis

We now analyze the regret bound of Algorithm 1. We first note that the loss $\mathcal{L}_{ij}(D_t)$ in (7) is the gradient $\nabla_t := \nabla f_t(\pi_t)$, where

$$\frac{\partial f_t}{\partial \pi_t[i, j]} = d_t[i] l[j] + \rho(g_t(\pi_t) - \mathcal{M}) s[j] = \mathcal{L}_{ij}(D_t).$$

Thus, we can rewrite the update of the preference matrix as follows:

$$H_t[i, j] \leftarrow H_{t-1}[i, j] - \eta \nabla_{t-1}[i, j], \quad (10)$$

We introduce an auxiliary variable X_t as $X_t[i, j] := \exp(H_t[i, j])$, and the policy update rule becomes:

$$\pi_t[i, \cdot] = \frac{X_t[i, \cdot]}{\|X_t[i, \cdot]\|_1}, \quad (11)$$

where $\pi_t[i, \cdot]$ is the vector of policy distribution for coded group i, $X_t[i, \cdot]$ is the row vector of matrix X_t, and $\|X_t[i, \cdot]\|_1$ denotes the L_1 norm of this vector.

We now show that our algorithm is a special case of Follow-The-Regularized-Leader (FTRL) algorithm. In FTRL, the policy π_t is updated as follows:

$$\pi_{t+1} = \arg\min_{\pi \in \mathcal{A}} \left\{ \eta \sum_{\tau=1}^{t} \nabla_\tau^\top \pi + R(\pi) \right\}. \quad (12)$$

Let $R(\pi) = \pi \log(\pi)$ be the negative entropy function, i.e.,

$$R(\pi) = \sum_{i=1}^{m} \sum_{j=1}^{n} \pi[i, j] \log \pi[i, j],$$

and its derivative is $\nabla R(\pi)[i, j] = 1 + \log \pi[i, j]$ for all i and j. Using the Lagrangian multiplier and the constraint of π as

$\sum_j \pi[i,j] = 1$ for all coded group i and some fixed $\vec{\lambda}$, we have

$$F_{t+1}(\pi, \vec{\lambda}) := \eta \sum_{\tau=1}^{t} \nabla_\tau^\top \pi + R(\pi) + \sum_{i=1}^{m} \lambda[i](\sum_{j=1}^{n} \pi[i,j] - 1). \tag{13}$$

Taking the partial derivatives on π and setting the derivatives to zero, we have

$$\frac{\partial F_{t+1}}{\partial \pi[i,j]} = \eta \sum_{\tau=1}^{t} \nabla_\tau[i,j] + 1 + \log \pi[i,j] + \lambda[i] = 0. \tag{14}$$

To simplify the analysis, we use the auxiliary variable X_{t+1} such that

$$\log X_{t+1}[i,j] = -\eta \sum_{\tau=1}^{t} \nabla_\tau[i,j] - 1. \tag{15}$$

Then, (14) becomes $\log \pi[i,j] = \log X[i,j] - \lambda[i]$. We first evaluate the recursive property of X_{t+1}. Since (15) holds for all time slot t, we can take $t \leftarrow t-1$, the previous time slot, and X_t satisfies:

$$\log X_t[i,j] = -\eta \sum_{\tau}^{t-1} \nabla_\tau[i,j] - 1. \tag{16}$$

Now we have:

$$\log X_{t+1}[i,j] = \log X_t[i,j] - \nabla_t[i,j], \tag{17}$$

and

$$X_{t+1}[i,j] = X_t[i,j] \exp(-\nabla_t[i,j]). \tag{18}$$

Now we evaluate the value of $\vec{\lambda}$. For a coded group i, we take sum over $j = 1, \ldots, n$ of the above equation and we have

$$1 = \sum_{j=1}^{n} \pi[i,j] = \sum_{j=1}^{n} X[i,j] \exp(-\lambda[i]). \tag{19}$$

Therefore, the update of π_t is equivalent to (6), showing Algorithm 1 is a special case of FTRL. Based on the following theorem, we can find the regret bound of Algorithm 1:

Theorem 1 ([14]). *For every policy $\pi \in \mathcal{A}$, the FTRL algorithm attains the following bound on the regret:*

$$\text{Regret}_T(\mathcal{A}) \leq 2\sqrt{2 D_R G_\infty^2 T}, \tag{20}$$

where D_R is the diameter of set $\mathcal{A}$ relative to the regularizer function R as

$$D_R = \max_{\pi_1, \pi_2 \in \mathcal{A}} \{R(\pi_1) - R(\pi_2)\},$$

G_∞ is the upper bound of gradient, i.e., $\|\nabla_t\|_\infty \leq G_\infty$ for all t, and the learning rate η is determined as

$$\eta = \sqrt{\frac{D_R}{2 G_\infty^2 T}}.$$

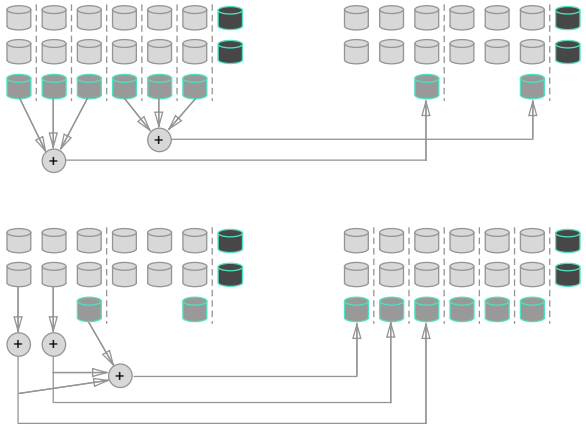


Fig. 1. Two LRC codes used in simulation with Upcode and Downcode operations [7]. It shows how to change code rate with a few simple and fast operations. The gray blocks are local parity blocks, and the black blocks are global parity blocks.

Now let us derive the specific values of the constants D_R and G_∞ for the proposed Algorithm 1. We can derive the diameter D_R for Algorithm 1 as follows:

$$\begin{aligned} R(\pi_1) - R(\pi_2) &\leq R(\pi_2) && (21)\\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} -\pi_2[i,j] \log \pi_2[i,j] && (22)\\ &\leq m \log n, && (23) \end{aligned}$$

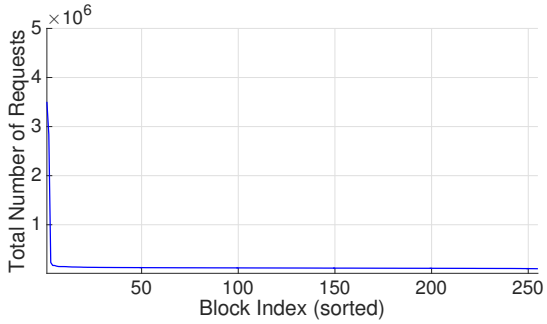
where (21) follows from the fact that $\pi_1[i,j] \log \pi_1[i,j]$ is non-positive for all i and j, and (23) follows from the entropy inequality, which shows that the random variable with uniform distribution has the maximum entropy. The next step is to determine the upper bound of gradient G_∞. We assume that the demand in any coded group is upper bounded by D. For any codebook c_j, we can find its recovery cost and its storage overhead immediately. We denote $L_{max} := \max_j l[j]$ and $S_{max} := \max_j s[j]$. Therefore, we have

$$G_\infty := D L_{max} + \rho S_{max} |mn S_{max} - \mathcal{M}| \geq \|\nabla_t\|_\infty \tag{24}$$

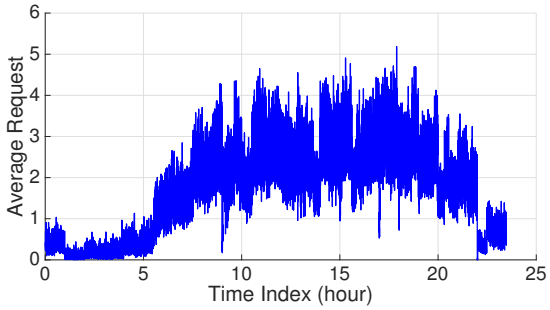
for Algorithm 1. Both (23) and (24) are constant with time T and we can get a regret bound of $O(\sqrt{T})$ of Algorithm 1 from Theorem 1.

IV. Performance Evaluation

We evaluated our online learning algorithm based on real workload traces collected from a production cluster in the Windows Azure Storage (WAS) system. Our trace dataset contains 252 equal-sized original data blocks *in every second* for a 22-hour period. We use two erasure codes, both are in the family of Local Reconstruction Codes [6], with different coding parameters. One of them is $(12, 2, 2)$ LRC with $k = 12$ data blocks, 2 local parity blocks and 2 global parity blocks, achieving a per-block recovery cost of 6 [6] and a storage overhead of 4. Another code is $(12, 6, 2)$ LRC with $k = 12$ data blocks, 6 local parity blocks and 2 global parity blocks, achieving a per-block recovery cost of 2 [6] and a larger storage overhead of 8. Fig. 1 shows the code structures of

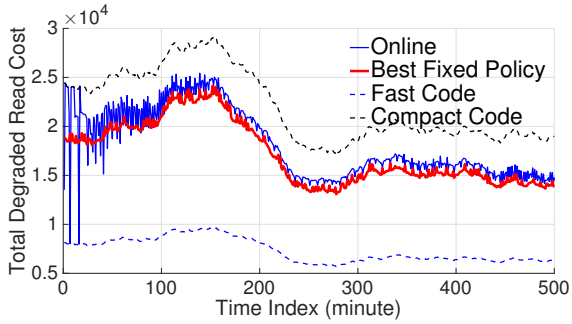


(a) Distribution of total requests per block

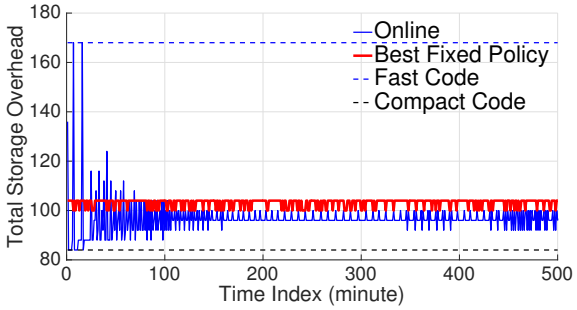


(b) Average request per block

Fig. 2. The statistics of the trace data collected from Windows Azure Storage (WAS).



(a) Total degraded read traffic in each minute



(b) Total storage overhead in each minute

Fig. 3. The performance of the Exponentiated Gradient Algorithm compared with other approaches.

these two codes and an efficient scheme to change code rates between them.

Fig. 2(a) shows the distribution of the total number of requests for each block in the trace data. We can see that the distribution of data access is highly skewed. The majority of data blocks are cold with only a few read accesses. However, there are a small fraction of data blocks that have high request demands. Fig. 2(b) shows the average request per block at different times. We can see that the trend of demand varies as the time evolves.

A. Performance of the Exponentiated Gradient Algorithm

We conduct our performance evaluation with each time slot representing 1-minute and we set the entire evaluation period to be $T = 500$ minutes. In each minute, we perform Algorithm 1, determine the policy matrix π_t, and for each coded group G_i, we pick a code c_j according to the probability distribution $\pi_t[i, \cdot]$. We compare the performance of Algorithm 1 with the following approaches:

- **Best fixed policy (offline)**: it learns the best fixed policy π^* by minimizing $\sum_{t=1}^{T} f_t(\pi)$ assuming all demands $D_1, \ldots, D_T$ are known. Note that this policy is also stochastic, and in each minute, each coded group G_i should choose one code according to the probability $\pi^*[i, \cdot]$.
- **A single fast code**: all coded groups choose the fast code $\text{LRC}(12, 6, 2)$.
- **A single compact code**: all coded groups choose the compact code $\text{LRC}(12, 2, 2)$.

Fig. 3(a) and Fig. 3(b) show the performance comparison from two perspectives, namely, total degraded read traffic and total storage overhead. We assume that 5% of all requests may encounter the events of block unavailability at random and thus triggering degraded reads. During each degraded read, the requests are randomly directed to the combinations of blocks that can reconstruct the original block, contributing to the total degraded read traffic. Fig. 3(a) illustrates that after about 100 minutes, the online algorithm can perform close to the best fixed code selection policy, which shows the ability of our algorithm to closely track the demand. It is surprising to see that Algorithm 1 can even save more storage as compared to the best fixed policy.

We also show the performances of two single coding schemes as comparisons, which are also illustrated in Fig. 3. The single compact code has the highest recovery cost, while the single fast code has the least recovery cost. However, this doesn't show the benefit of the single fast code, since such performance is at the expense of its storage overhead. In other words, neither of the two single codes do not achieve a proper tradeoff between recovery cost during degraded read and storage overhead.

V. Relationship to Prior Work

A number of recent studies [6]–[8] show that aside from node failures, the majority of degraded reads are due to the *temporary unavailability* of single original blocks. For example, Over 98% of all failure modes in Facebook's data-warehouse and other production HDFS clusters require recovery of a single temporary block failure [8], [15] instead of

node failures. And only less than 0.05% of all failures involve three or more blocks simultaneously. It is thus important to optimize the latency and cost associate with the reconstruction of individual data blocks, especially hot ones.

Extensive recent studies have been conducted to reduce the repair cost (i.e., the number of reads required) for degraded reads when the original data is unavailable, through coding theory and system optimization. Local Reconstruction Code (LRC) [6] is proposed to reduce the IOs required for reconstruction reads over Reed-Solomon (RS) codes, while still allowing a significant reduction in storage overhead as compared to 3-way replication. Similar locally recoverable codes have been presented in [11], [12]. HitchHiker [8] is a another erasure-coded storage system that reduces both network traffic and disk I/O during reconstruction of unavailable data, riding on top of RS codes based on new encoding and decoding techniques. [16] presents an algorithm that finds the optimal number of codeword symbols needed for recovery with any XOR-based erasure code and produces recovery schedules to use a minimum amount of data. [5] proposes FastDR, a system that addresses node heterogeneity and exploits I/O parallelism, to enhance the performance of degraded reads.

[7] presents HACFS, which uses two different erasure codes, i.e., a fast code is used to encode frequently accessed data for low recovery cost, and a compact code is used to encode the majority of data to maintain a low overall storage overhead. HACFS [7] is similar to our work in the sense that it tries to use two erasure codes to differentiate the coding schemes for hot and cold blocks. However, what is lacking in [7] is an intelligent scheme to determine code selection based on demand. In this paper, we propose a data-driven online learning framework which can perform code selection for different coded groups given any finite number of erasure codes (possibly of different types or constructions) in a storage system. We show that our algorithm is fast and achieves a regret bound of $O(\sqrt{T})$ as T grows.

Optimized code selection based on the demand of each coded group can significantly reduce recovery traffic cost while maintaining a similar storage overhead. Such a benefit is especially salient for skewed demands, which have also been observed in other studies [7], [9], [10], [13].

VI. Concluding Remarks

Erasure codes have been widely deployed in cloud storage systems. Although erasure codes can provide more reliability with less storage overhead, yet they incur higher reading traffic when reconstructing an unavailable block. Since degraded read may happen frequently in a coded storage system due to software glitches and server congestion, the higher reconstruction cost is the major reason that leads to long latency tails in erasure coded storage systems. In this paper, we study the problem of code rate adaption and codebook selection in storage systems that adopt multiple erasure codes to for service differentiation between hot and cold content. Since it is impossible to know the block demands before the code selection decisions are made, we propose an online learning algorithm to make dynamic decisions only based on the past observations. To avoid solving a hard integer program, we perform a stochastic relaxation for code selection and solve the online learning problem using an exponentiated gradient descent algorithm, leading to sparse solutions. We prove a regret bound $O(\sqrt{T})$ of our algorithm by showing that our algorithm is a special case of the FTRL online learning framework. Through trace-driven simulations based on real request traces from Windows Azure Storage, we show that our online learning algorithm performs close to the best fixed offline code selection policy, and can achieve fine-tuned tradeoff between degraded read cost and storage overhead.

References

[1] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[3] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci *et al.*, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[4] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Peer-to-Peer Systems*. Springer, 2002, pp. 328–337.

[5] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads." in *FAST*, 2012, p. 20.

[6] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in windows azure storage." in *Usenix annual technical conference*. Boston, MA, 2012, pp. 15–26.

[7] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in hdfs," in *To appear in Proceedings of 13th Usenix Conference on File and Storage Technologies*, 2015.

[8] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 331–342.

[9] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.

[10] K. Ren, Y. Kwon, M. Balazinska, and B. Howe, "Hadoop's adolescence: an analysis of hadoop usage in scientific workloads," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 853–864, 2013.

[11] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 325–336.

[12] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *Information Theory, IEEE Transactions on*, vol. 60, no. 8, pp. 4661–4676, 2014.

[13] C. L. Abad, N. Roberts, Y. Lu, and R. H. Campbell, "A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 100–109.

[14] S. Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2011.

[15] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster," *Proc. USENIX HotStorage*, 2013.

[16] Y. Zhu, J. Lin, P. P. Lee, and Y. Xu, "Boosting degraded reads in heterogeneous erasure-coded storage systems."