On the Practical Application of Long Erasure Codes to Large-Scale Storage Systems

Greg Hood
Pittsburgh Supercomputing Center, Carnegie Mellon University
Pittsburgh, Pennsylvania
ghood@psc.edu

ABSTRACT

While capacities of both disk drives and SSDs have continued to increase, there is still demand for storing ever larger datasets and collections of datasets. Efficiency in how we use large numbers of individual devices to assemble reliable storage arrays remains an important question. Devices have most commonly been combined into RAID 1, RAID 5, or RAID 6 arrays, or, more recently, have employed other types of relatively short erasure codes with $n \leq 16$, n being the total number of devices combined to store both data and parity.

This paper examines trade-offs between performance, space efficiency, and reliability in the use of erasure codes with $3 \leq n < 256$, focusing on the use of Reed-Solomon codes. It discusses the computational costs of generating and recovering data blocks using currently available software packages and hardware, and presents parameters that should work well for various object sizes on current hard drives and SSDs.

CCS CONCEPTS

• **Information systems** → **RAID**; *Disk arrays*; Storage power management;

KEYWORDS

RAID, erasure coding, Reed-Solomon codes, ISA-L

1 INTRODUCTION

The need for high-capacity,reliable storage has been present since the dawn of the information age, and continues to grow annually. Storage system capacities have increased as new technologies have emerged and matured, but the goal of creating a reliable system out of unreliable components has remained the same. The individual device reliability of present technologies such as hard drives and SSDs has improved over time, but these devices are being aggregated into ever larger groups to satisfy the storage needs of research and commercial entities. This paper explores how a large number of storage devices (either rotating drives or SSDs) may be combined in an economically viable way to yield high performance and highly reliable storage systems.

2 BACKGROUND

RAID methods [9] have been widely used for many years, and instances of their application can be found in nearly every datacenter. RAID levels 5 (able to survive one drive failure) and 6 (able to survive two drive failures) have been implemented in both software and hardware. The ZFS filesystem[8] introduced software-based RAIDz3, which has triple parity and can tolerate three drive failures. There have been proposals of quadruple parity RAID [3], but the author is not aware of any in widespread use. RAID methods can be considered a specialized form of erasure coding which operates at the drive level, and all the drives within a set should be of identical capacity and similar in performance, since each stripe is composed of blocks at the same offset on each drive.

Erasure coding methods, and specifically Reed-Solomon codes[14], have had a long history in domains such as deep space transmission, compact discs (CDs,DVDs), and cellular phone communication, and are now making inroads onto storage arrays within datacenters. There are multiple implementations of Reed-Solomon encoding for storage applications, both in software[2, 6, 13] and hardware[7]. The benchmarking runs that were conducted for the current paper used the ISA-L library from Intel[6] because it supports codes up to 255 elements long, is reasonably fast, is written in C, and is actively maintained.

There are other forms of erasure coding which have different theoretical underpinnings than Reed-Solomon. One example is the Mojette transform[5], a discrete analog of the Radon transform used in reconstructing medical CT scans. The RozoFS filesystem[10] uses the Mojette transform to encode data, and the typical ratio of encoded to original data is 1.5:1. Pertin *et al.*[11] report that the speed is up to twice as fast as ISA-L. However, the disadvantage is that in the most common situation of zero errors, RozoFS must decode the encoded data, whereas Reed-Solomon stores data in its original form, so no decoding is required. Also, their longest code mentioned is $(k = 8, n = 12)$, so it would not

be possible to make direct comparison with Reed-Solomon codes for $n > 12$.

3 INDIVIDUAL DEVICE RELIABILITY

Individual storage devices such as hard drives or SSDs have various failure modes, and modeling these in detail would require very device-specific assumptions. For this paper, we adopt a simple exponential failure model based on annualized failure rate (AFR), which abstracts away from these device specifics. Our modeling timespan will be the manufacturer's typical 5-year warranty period, where factors such as wear-out, which would cause significant deviations from this type of failure model, can be ignored. There are various published estimates [1, 12, 15] for AFR values for hard drives, and they generally vary from 1.0% – 5.0%. Our own experience at the Pittsburgh Supercomputing Center with drives used in archival-type configurations has been an AFR toward the lower end of this range. Specific models and/or production lots occasionally have rates above this, which argues for mixing drives from different vendors and lots in a large-scale storage system. SSDs are reported [16] to have a lower overall failure rate, though with a higher rate of uncorrectable errors. To err on the conservative side, we will assume an AFR value of 5% in this paper for all devices, although the equations are parameterized so the impact of other AFRs could easily be calculated. Table 1 shows how different AFRs progress over time, and the reader may estimate what AFR best matches their own experience with drive failures.

Table 1: Failure rates over time based on AFR

years	First row is annualized failure rate (AFR)				
1	1.00%	2.00%	3.00%	4.00%	5.00%
2	1.99%	3.96%	5.91%	7.84%	9.75%
3	2.97%	5.88%	8.73%	11.53%	14.26%
4	3.94%	7.76%	11.47%	15.07%	18.55%
5	4.90%	9.61%	14.13%	18.46%	22.62%

4 FUNDAMENTALS OF ERASURE CODING

In Reed-Solomon erasure coding we group k data blocks together and generate m parity blocks from these data blocks (see Figure 1). These $n = k + m$ blocks constitute a stripe that will be stored across n drives. Note that there is no requirement that each block of a stripe be stored at the same offset within a drive, or that the same set of n drives be used for every stripe. There is great latitude in how blocks of a stripe may be scattered across a larger set of q drives, where it is possible that $q >> n$.

If up to m blocks become unreadable in the stripe (either data blocks or parity blocks), then the entire stripe may be fully regenerated due to the properties of the Reed-Solomon encoding. An easy-to-understand explanation of how this is done may be found in [2]. A block can be in one of two

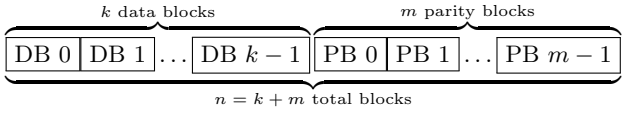

Figure 1: A stripe of size $n * b$ bytes is distributed across n drives; each block is b bytes long

states – *correct* or *erased*, hence the term "erasure coding". Which of these two states it is in can be known by whether the drive returned bytes for this block that had a correct checksum, or whether the drive failed and did not return bytes for this block, or these bytes had an incorrect checksum. If we combine n blocks from different drives together into a stripe, then the stripe has 2^n possible states. The probability of any particular stripe state with i erasures and $n-i$ correct blocks is given by:

$$p^i(1-p)^{n-i}$$

where p is the probability that an individual block was erased. To find the probability that the stripe has i erasures in any position, we simply multiply by the number of ways that pattern can occur – the binomial coefficient of combinatorics:

$$\binom{n}{i} p^i(1-p)^{n-i}$$

Then, in order to compute the probability that the entire stripe is unrecoverable, *i.e.*, has more than m erasures, we sum up these probabilities:

$$p_{\text{stripe failure}} = \sum_{i=m+1}^{n} \binom{n}{i} p^i(1-p)^{n-i} \quad (1)$$

The foregoing makes the critical assumption that all drive failures are *independent*, which is completely untrue in the event of natural disasters such as floods and fires, or man-made disasters such as software bugs or hackers breaking into the storage system. The latter could even be true in a distributed storage system where the disks are located in different datacenters. Thus, it is strongly advisable to have separate offsite (and possibly offline) backups of important information. Still, when designing a storage system we can strive to keep the drive failures more independent by locating the blocks of a stripe on drives on different backplanes, in different chassis, or in different racks. For the purposes of this paper, we are going to ignore the catastrophic events, and compute failure rates based on normal operating conditions.

If we use a drive AFR of 5% then the probability that a single drive has failed after 5 years of service is 22.62%, or 0.2262. We are going to assume that a drive failure affects all blocks stored on it. Substituting this for p into equation 1, and choosing various values of k and m, we can construct table 2. Note that the values in the table represent probabilities in the range 0 to 1, where 1.00e+00 indicates certain failure, and 1.00e-02 indicates a 1% chance of failure.

As the reader can see, it is almost certain that any stripe consisting of $k >= 64$ blocks will become unrecoverable by the end of 5 years, and, even for much smaller values of k, the

Table 2: Stripe failure rates after 5 years (no replacements)

	m=2	m=3	m=4	m=5	m=6	m=7	m=8
k=1	1.16e-02	2.62e-03	5.92e-04	1.34e-04	3.03e-05	6.86e-06	1.55e-06
k=2	3.85e-02	1.07e-02	2.88e-03	7.56e-04	1.95e-04	4.93e-05	1.24e-05
k=4	1.34e-01	5.07e-02	1.78e-02	5.92e-03	1.88e-03	5.78e-04	1.73e-04
k=8	4.02e-01	2.24e-01	1.13e-01	5.23e-02	2.27e-02	9.33e-03	3.66e-03
k=16	8.09e-01	6.53e-01	4.85e-01	3.33e-01	2.13e-01	1.28e-01	7.24e-02
k=32	9.90e-01	9.71e-01	9.34e-01	8.73e-01	7.88e-01	6.84e-01	5.69e-01
k=64	1.00e+00	1.00e+00	1.00e+00	9.99e-01	9.98e-01	9.95e-01	9.90e-01
k=128	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00
k=224	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00

probability of stripe failure is unacceptably high unless large numbers of parity blocks are used. However, these failure rates are based on a totally passive approach to drive management, where drives that have failed are kept in the system and not replaced with good drives. If we replace drives as soon as they fail (either by physically swapping out the bad drives, or by logically replacing them with online hot-spares), then we can greatly improve the survival rate of our data. When a drive is replaced, the replacement does not instantaneously restore to a pristine condition the stripes that were located on the failed drive. The missing blocks must be reconstructed and written one by one to the replacement drive, and this requires a finite amount of time. If we use a drive size of 10TB and a writing speed of 100MB/s (assuming a fairly large block size), then this process will take slightly over a day. If we round this up to 2 days, the damaged stripes will be vulnerable to additional failures during this 2 day period. We can approximate the probability of a stripe failing over this two-day period by calculating the individual drive failure rate over a two-day period ($p = 2.808 * 10^{-4}$), and then using equation 1. The entire 5-year span can then be treated to a first approximation as a succession of two-day intervals. The number of 2-day intervals in 5 years is 913.125, so we have:

$$p_{\text{5-yr stripe failure}} \approx 1 - (1 - p_{\text{2-day stripefailure}})^{913.125}$$

Probabilities for selected values of k and m are given in Table 3. This sufficiently reduces the probability of stripe failure to where even very long erasure codes become extremely reliable with fewer than 10 parity blocks.

5 ECONOMICS OF EMPLOYING ERASURE CODING

An often-expressed concern about using long erasure codes is the overhead of encoding and decoding each stripe of data. In order to address this concern the author benchmarked the performance of the ISA-L library on 9588 combinations of k, m, and b. This benchmarking was done on an Intel® Xeon® E5-2695v3 CPU with a nominal 2.30GHz clock. While it is true that Reed-Solomon encoding is more cycle-intensive than simple parity operations, increased CPU power, and, specifically, the incorporation of wide vector instructions such as SSE, AVX, AVX2, and AVX-512 into recent processors enable both encoding and decoding to be performed rapidly. For decoding, one should remember that most of the time it does not have to be performed at all, and, when it does, it is usually to recover from a single error, not m errors. The speed of decoding in the presence of a single error is often much greater than decoding with multiple errors. For instance, for $(k = 64, m = 8)$, the decoding speed in our ISA-L benchmarks was 1592MB/s with one error and 505MB/s with 8 errors. Multiple errors are quite rare, and as long as the speed for dealing with them is not orders of magnitude slower, they have much less impact on overall performance.

In this section we outline a cost model for the storage components of an erasure-coded storage system. This model will then be used with optimization techniques to find good values of k, m, and b to use in such a system. To construct this model we developed methods for quantitatively estimating the cost of using erasure codes of various lengths. The particular numbers obtained depend on the read and write frequencies of the data objects being stored, and also on the size of those objects. We do not assume any sort of alignment of the objects with either block or stripe boundaries, since requiring alignment to these large extents (often >>1MiB) would waste a substantial amount of space. We first compute the average number of blocks that an object will be split over, and then the average number of stripes that those blocks will be split over. We then calculate the average number of blocks that will be required to read the object, given the probabililty of block failure, which, if it occurs, entails reading the entire stripe that contains the failed block. Similarly, we calculate the average number of blocks to write an object of that size, taking into account that an entire stripe (including the parity blocks) will have to be written. Code for performing these calculations is available from the author upon request. We include here Table 4 as an example of the results of this step with the block size held fixed at 1MiB. One can see in the middle column of this table that when the object size is comparable to the block size, the I/O is essentially doubled because an object will nearly always span two blocks. This will favor configurations that keep the block size well below the average object size. Another observation from the last column is that the object size needs to be somewhat larger

Table 3: Stripe failure rates after 5 years (immediate replacements)

	m=2	m=3	m=4	m=5	m=6	m=7	m=8
k=1	2.02e-08	5.68e-12	1.58e-15	0.00e+00	0.00e+00	0.00e+00	0.00e+00
k=2	8.09e-08	2.84e-11	9.55e-15	0.00e+00	0.00e+00	0.00e+00	0.00e+00
k=4	4.04e-07	1.99e-10	8.92e-14	4.95e-17	0.00e+00	0.00e+00	0.00e+00
k=8	2.42e-06	1.87e-09	1.26e-12	7.92e-16	0.00e+00	0.00e+00	0.00e+00
k=16	1.64e-05	2.19e-08	2.46e-11	2.42e-14	0.00e+00	0.00e+00	0.00e+00
k=32	1.20e-04	2.95e-07	5.97e-10	1.03e-12	1.58e-15	0.00e+00	0.00e+00
k=64	9.13e-04	4.29e-06	1.64e-08	5.29e-11	1.48e-13	3.47e-16	0.00e+00
k=128	7.02e-03	6.47e-05	4.79e-07	2.98e-09	1.60e-11	7.58e-14	2.97e-16
k=224	3.60e-02	5.82e-04	7.44e-06	7.96e-08	7.34e-10	5.94e-12	4.30e-14

than the stripe size for most efficient writing. Both considerations underscore the fact that erasure-coded storage is best employed with very large-sized objects.

In the following formulas we estimate the time consumed by either the client or storage server cores to generate block checksums using xxHash[4] (which we benchmarked at 3.375GB/s on our test system). We also estimate (using the stored results of the ISA-L benchmarking) how much time the cores will consume to perform the Reed-Solomon encodings or decodings:

$$\begin{aligned} coreTimePerRead &= \frac{averageBlocksRead \cdot b}{checksum\_rate} \\ &+ \frac{averageStripeCorrections \cdot n \cdot b}{decodingRate[k][m][log2b]} \\ coreTimePerWrite &= \frac{averageBlocksWritten \cdot b}{checksum\_rate} \\ &+ \frac{n \cdot b}{encodingRate[k][m][log2b]} \\ coreFraction &= \frac{coreTimePerRead \cdot reads\_per\_5\_years}{5 \cdot 365.25 \cdot 24 \cdot 3600} \\ &+ \frac{coreTimePerWrite \cdot writes\_per\_5\_years}{5 \cdot 365.25 \cdot 24 \cdot 3600} \end{aligned}$$

We next compute the space cost for that object and the total capital cost associated with storing that object. Note that the capital costs only include the drives and JBODs and a share of the CPU cores on the accessing nodes. It does not include other storage server hardware since we are focusing on just the actual storage components of a storage system.

$$\begin{aligned} driveFraction &= \frac{objectSize}{drive\_capacity} \cdot \frac{k+m}{k} \\ capitalCost &= driveFraction \cdot drive\_cost \\ &+ coreFraction \cdot core\_cost \end{aligned}$$

Finally, we estimate operating costs by computing the energy consumed by both drives and cores in performing the read and write operations:

$$\begin{aligned} wattSeconds_{driveReads} &= \Big(\frac{averageBlocksRead}{drive\_read\_IOPs} + \\ &\quad \frac{averageBlocksRead \cdot b}{drive\_read\_rate}\Big) \\ &\cdot reads\_per\_5\_years \\ &\cdot (drive\_active\_power - drive\_idle\_power) \\ wattSeconds_{coreReads} &= coreTimePerRead \cdot reads\_per\_5\_years \\ &\cdot core\_power \\ wattSeconds_{reads} &= wattSeconds_{driveReads} \\ &+ wattSeconds_{coreReads} \\ wattSeconds_{driveWrites} &= \Big(\frac{averageBlocksWritten}{drive\_write\_IOPs} + \\ &\quad \frac{averageBlocksWritten \cdot b}{drive\_write\_rate}\Big) \\ &\cdot writes\_per\_5\_years \\ &\cdot (drive\_active\_power - drive\_idle\_power) \\ wattSeconds_{coreWrites} &= coreTimePerWrite \cdot writes\_per\_5\_years \\ &\cdot core\_power \\ wattSeconds_{writes} &= wattSeconds_{driveWrites} \\ &+ wattSeconds_{coreWrites} \\ wattSeconds_{idle} &= driveFraction \cdot drive\_idle\_power \\ &\cdot 5 \cdot 365.25 \cdot 24 \cdot 3600 \\ operatingCost_{idle} &= \frac{wattSeconds_{idle}}{1000 \cdot 3600} \cdot energy\_cost\_per\_KWH \\ operatingCost_{reads} &= \frac{wattSeconds_{reads}}{1000 \cdot 3600} \cdot energy\_cost\_per\_KWH \\ operatingCost_{writes} &= \frac{wattSeconds_{writes}}{1000 \cdot 3600} \cdot energy\_cost\_per\_KWH \\ operatingCost &= operatingCost_{idle} \\ &+ operatingCost_{reads} + operatingCost_{writes} \\ &+ \frac{driveFraction}{drives\_per\_U} \cdot \frac{5 \cdot annual\_cost\_per\_rack}{42} \end{aligned}$$

The above formulas obtain the capital and operating cost for a single data object. To obtain overall cost values we multiply these by how many objects of that size the storage system can contain.

6 RESULTS OF PARAMETER OPTIMIZATION

In any optimization process it is important to be clear on what is being optimized. In the case of a storage system, are we optimizing for performance, or for cost of storing x petabytes, or some combination? Computer optimization

Table 4: Stripes / Blocks_read / Blocks_written for objects of various sizes with block size 1MiB

	4KiB objects	64KiB objects	1MiB objects	16MiB objects	256MiB objects
k=1,m=4	1.00/1.01/5.02	1.06/1.06/5.31	2.00/2.00/10.00	17.00/17.02/85.00	257.00/257.29/1285.00
k=2,m=4	1.00/1.01/6.01	1.03/1.06/6.19	1.50/2.00/9.00	9.00/17.02/54.00	129.00/257.29/774.00
k=4,m=4	1.00/1.01/8.01	1.02/1.06/8.12	1.25/2.00/10.00	5.00/17.02/40.00	65.00/257.29/520.00
k=8,m=4	1.00/1.01/12.01	1.01/1.07/12.09	1.12/2.01/13.50	3.00/17.02/36.00	33.00/257.29/396.00
k=16,m=4	1.00/1.01/20.00	1.00/1.07/20.08	1.06/2.01/21.25	2.00/17.04/40.00	17.00/257.31/340.00
k=32,m=5	1.00/1.01/37.00	1.00/1.07/37.07	1.03/2.02/38.16	1.50/17.11/55.50	9.00/257.45/333.00
k=64,m=6	1.00/1.02/70.00	1.00/1.08/70.07	1.02/2.04/71.09	1.25/17.26/87.50	5.00/257.81/350.00
k=128,m=7	1.00/1.04/135.00	1.00/1.10/135.07	1.01/2.07/136.05	1.12/17.57/151.87	3.00/259.02/405.00
k=224,m=8	1.00/1.07/232.00	1.00/1.13/232.06	1.00/2.13/233.04	1.07/18.03/248.57	2.14/262.19/497.14

will often yield nonsensical results if there is a sole focus on optimizing a particular quantity, and that is indeed the case here. For example, if we optimize on just storage cost (TB/$), we end up with a system that uses very long erasure codes even for small objects. This has minimized the storage cost of those objects, but writing a single small object requires writing a huge number of blocks to many drives, and the performance is poor. On the other hand, if we optimize on just performance (GB/s/$), then we end up with shorter codes with a large m to k ratio assuming reads greatly outnumber writes, because reading usually does not require the parity blocks. The data blocks get spread over a larger number of drives $n = k + m$, and this increases the performance, but the overall cost is excessive.

We need to optimize on a combination of both capacity and performance, but using a sum of TB and GB/s is somewhat like counting apples and oranges. To have some common basis when forming a weighted sum of these quantities, we scale each to be in units of what a single drive provides. For example, if our hard drive candidate has a capacity of 10TB, and a sustained transfer rate of 200MB/s, then we scale the overall storage system capacity to units of 10TB, and the overall performance to units of 200MB/s, and then combine. If we choose to give equal weight to both goals, we would then use a weighted sum with coefficients of 0.5 as the value to optimize on.

In this paper, we optimize on this equal weighting scheme of both capacity and performance for a selection of object sizes in a range from 256 bytes to 1GiB, subject to the following constraints:

- The drives are usable at full capacity. In other words, with the expected read and write frequencies for objects of a given size, the drives have sufficiently low latencies and sufficiently high bandwidth that the entire drive may be populated with objects.
- The erasure coding parameters yield a failure rate less than or equal to 10^{-15} over 5 years. We choose a rate this low because the storage system may be holding billions of stripes, and each stripe might potentially involve a different subset of drives.
- $n < 256$ in order to use available EC libraries.

We define performance as the steady-state GB/s achievable when reading or writing objects of the given size. The performance itself is a weighted sum of the read and write bandwidth for objects read from or written to storage, *i.e.*, it is not raw drive bandwidth. The performance weighting is determined by the ratio of expected reads and writes over a 5 year period.

The optimization was performed by constructing a computer program in C to go though many possible combinations of parameters k, m, and b. This program was also given a set of constants defining characteristics of the drives, a range of object sizes for reads and writes, and the expected number of reads and writes over a 5 year period. It also used the benchmark results obtained for the ISA-L library to estimate encoding and decoding costs. The program selected the parameter set which maximized our capacity-performance metric per dollar, discarding any sets that did not satisfy the above constraints. We ran the program first with these typical hard drive constants:

- drive_annualized_failure_rate = 0.05
- drive_read_IOPs = 240.0
- drive_write_IOPs = 240.0
- drive_read_rate = 190MB/s
- drive_write_rate = 190MB/s
- drive_cost = 387.0 dollars *(includes JBOD overhead)*
- drive_capacity = 10TB
- drive_idle_power = 4.5 watts
- drive_active_power = 8.0 watts
- drives_per_U = 11.0
- writes_per_5_years = 20 *(per object)*
- reads_per_5_years = 200 *(per object)*
- checksum_rate = 3.375GB/s
- core_cost = 200.0 dollars
- core_power = 10.0 watts
- energy_cost_per_KWH = 0.1 dollars
- annual_cost_per_rack = 1200.0 dollars

This yielded the best k, m, b parameters for various object sizes, as shown in Table 5. For small objects (64KiB and less), the program could find no suitable parameter combinations because the hard drives could not keep up with the specified number (200) of object reads per 5 years, while still being filled to capacity. Due to the small size of the objects the

Table 5: Parameters and Estimated Costs for Hard Drives

Object Size	EC Parameters	Read GB/s	Write GB/s	Capital Cost/TB	Operating Cost/TB	Total Cost/TB	Total Cost/(GB/s)	Perf. Effic.	Cap. Effic.
64KiB	(for 64KiB and lower, no parameter combinations satisfied the constraints)								
256KiB	k=2,m=4,b=512KiB	0.15	0.03	$116.46	$11.05	$127.51	$18188.68	0.12	0.33
1MiB	k=8,m=5,b=512KiB	0.65	0.12	$63.09	$5.95	$69.05	$9119.50	0.25	0.62
4MiB	k=26,m=6,b=1MiB	2.75	0.38	$47.89	$4.48	$52.37	$5372.14	0.42	0.81
16MiB	k=52,m=6,b=2MiB	7.01	0.96	$43.57	$4.02	$47.58	$3828.50	0.59	0.90
64MiB	k=112,m=7,b=4MiB	17.40	2.24	$41.66	$3.83	$45.49	$3179.73	0.71	0.94
256MiB	k=120,m=7,b=8MiB	20.82	4.39	$41.29	$3.71	$45.00	$2794.57	0.80	0.94
1GiB	k=120,m=7,b=16MiB	22.23	7.57	$41.17	$3.65	$44.82	$2573.75	0.87	0.94

drives would spend most of their time seeking to the proper position instead of performing actual I/O. The sensible alternative for these object sizes would be to choose drives with faster seek times (such as SSDs). The effect of long seek times also shows up for larger objects and their correspondingly larger block sizes. The fact that a 1GiB object does not fill a single stripe at $k = 120$ explains why even longer erasure codes were not chosen.

The second-to-last column in Table 5 represents *performance efficiency*, or what fraction of the drive set's raw I/O capability is potentially deliverable to clients. The last column represents *capacity efficiency*, or what fraction of the drive set's raw capacity is potentially usable by clients. With large objects and long erasure codes, the efficiency of the storage system in both respects becomes very good. One may compare this efficiency with that of typical RAID-6 and RAIDz3 storage systems, where efficiencies are typically in the 0.7 to 0.8 range.

We also ran the program with typical SSD constants, as listed below:

- drive_annualized_failure_rate = 0.05
- drive_read_IOPs = 72000.0
- drive_write_IOPs = 20000.0
- drive_read_rate = 500MB/s
- drive_write_rate = 330MB/s
- drive_cost = 290.0 dollars *(includes JBOD overhead)*
- drive_capacity = 480GB
- drive_idle_power = 0.9 watts
- drive_active_power = 3.6 watts
- drives_per_U = 18.0
- writes_per_5_years = 20 *(per object)*
- reads_per_5_years = 2000 *(per object)*
- *all other parameters as before*

Note that we increased the read frequency by a factor of 10 to correspond to a more frequent access pattern which would favor the use of SSDs over rotating drives. We obtained the results shown in Table 6. Because of the high IOPs of an SSD, relatively small block sizes (b) emerged as winners. This leads to very long erasure codes becoming efficient with object sizes as small as 64KiB. The stripe lengths also result in extremely high (>100GB/s) potential I/O rates, which would be a challenge to deliver to clients. The much increased operating costs of an SSD-based system as compared with one based on hard drives is mostly due to the higher power consumption of SSDs on a watt/TB basis. (Individual devices consume less power but more devices are needed to yield the same capacity.)

7 CONCLUSIONS

The present paper has shown that the overheads of using long erasure codes are reasonable, and that, especially for systems such as archival storage, they offer a means of achieving high reliability while reducing the overall number of drives required. Of course, choice of a particular storage architecture will depend on many factors not discussed above, such as installation constraints, compatibility with pre-existing hardware, expected usage patterns (or uncertainty about what they will be), and object store and/or filesystem software availability. The chief downside to long erasure codes is that they require many individual drives, making them more suitable for arrays of 1PB or larger. The early adopters of erasure coding have indeed been large online enterprises such as Backblaze and Facebook. It is likely that erasure coding will begin to displace RAID-style storage in more mid-level applications in the near future.

Storage architects designing systems today that will hold large objects (*e.g.*, images) without a lot of "churn" should consider employing longer erasure codes to most economically store large volumes of data. The results in this paper also emphasize that making the objects to be stored as large as possible (perhaps by bundling smaller objects together) will result in benefits to both storage efficiency and performance.

ACKNOWLEDGMENTS

This work is supported under grant 1R24MH114793 by NIH-NIMH. Benchmarking was done on the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC).

REFERENCES

[1] Backblaze, Inc. 2018. Hard Drive Stats for 2017. (Feb. 2018). https://www.backblaze.com/blog/hard-drive-stats-for-2017/

[2] Brian Beach. 2015. Backblaze Open Sources Reed-Solomon Erasure Coding Source Code. (June 2015). https://www.backblaze.com/blog/reed-solomon/

Table 6: Parameters and Estimated Costs for SSDs

Object Size	EC Parameters	Read GB/s	Write GB/s	Capital Cost/TB	Operating Cost/TB	Total Cost/TB	Total Cost/(GB/s)	Perf. Effic.	Cap. Effic.
256B	(no parameter combinations satisfied the constraints)								
1KiB	k=2,m=4,b=2KiB	0.23	0.01	$1814.89	$79.35	$1894.24	$8067.23	0.08	0.33
4KiB	k=9,m=5,b=4KiB	1.29	0.06	$941.49	$40.85	$982.34	$3310.15	0.18	0.64
16KiB	k=32,m=6,b=8KiB	6.79	0.21	$718.87	$30.82	$749.69	$1712.67	0.36	0.84
64KiB	k=96,m=7,b=16KiB	28.16	0.63	$649.91	$27.72	$677.63	$1119.65	0.54	0.93
256KiB	k=120,m=7,b=32KiB	45.08	1.65	$640.80	$27.04	$667.84	$861.55	0.71	0.94
1MiB	k=120,m=7,b=64KiB	52.41	3.72	$640.51	$26.85	$667.37	$740.24	0.82	0.94
4MiB	k=120,m=7,b=128KiB	56.93	7.40	$640.37	$26.77	$667.14	$680.82	0.89	0.94
16MiB	k=120,m=7,b=256KiB	59.68	12.96	$640.33	$26.73	$667.07	$648.82	0.94	0.94
64MiB	k=208,m=8,b=256KiB	102.98	35.63	$628.41	$26.25	$654.65	$638.84	0.95	0.96
256MiB	k=224,m=8,b=512KiB	112.99	49.85	$626.70	$26.16	$652.86	$624.68	0.97	0.97
1GiB	k=224,m=8,b=1MiB	114.36	59.71	$626.65	$26.15	$652.80	$616.68	0.98	0.97

[3] Ming-Shing Chen, Bo-Yin Yang, and Chen-Mou Cheng. 2013. RAIDq: A Software-friendly, Multiple-parity RAID. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. USENIX, San Jose, CA. https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/Chen

[4] Yann Collet. 2017. xxHash. (2017). http://www.xxhash.com/

[5] JeanPierre Guedon and Nicolas Normand. 2005. The Mojette Transform: The First Ten Years. In *Discrete Geometry for Computer Imagery* (2005-04) *(Lecture Notes in Computer Science)*, Eric Andres, Guillaume Damiand, and Pascal Lienhardt (Eds.), Vol. 3429. Springer Berlin / Heidelberg, 79–91. https://doi.org/10.1007/b135490

[6] Intel. 2018. Intel Intelligent Storage Acceleration Library (Intel ISA-L). (2018). https://software.intel.com/en-us/storage/ISA-L

[7] Mellanox. 2018. Understanding Erasure Coding Offload. (2018). https://community.mellanox.com/docs/DOC-2414

[8] Oracle. 2012. Oracle Solaris ZFS Administration Guide. (April 2012). https://docs.oracle.com/cd/E23823_01/html/819-5461/index.html

[9] David A. Patterson, Garth A. Gibson, and Randy H. Katz. 1987. *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. Technical Report UCB/CSD-87-391. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/5853.html

[10] Dimitri Pertin, Sylvain David, Pierre Evenou, Benoît Parrein, and Nicolas Normand. 2014. Distributed File System based on Erasure Coding for I/O Intensive Applications. In *4th International Conference on Cloud Computing and Service Science (CLOSER) (Proceedings of the 4th International Conference on Cloud Computing and Services Science)*, Vol. 1. INSTICC, SciTePress, Barcelone, Spain, 451–456. https://doi.org/10.5220/0004960604510456

[11] Dimitri Pertin, Didier Féron, Alexandre van Kempen, and Benoît Parrein. 2015. Performance evaluation of the Mojette erasure code for fault-tolerant distributed hot data storage. *CoRR* abs/1504.07038 (2015). arXiv:1504.07038 http://arxiv.org/abs/1504.07038

[12] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz Andr Barroso. 2007. Failure Trends in a Large Disk Drive Population. In *5th USENIX Conference on File and Storage Technologies (FAST 2007)*. 17–29. https://research.google.com/archive/disk_failures.pdf

[13] James S. Plank and Kevin M. Greenan. 2014. *Jerasure: A Library in C Facilitating Erasure Coding for Storage Applications (Version 2.0)*. Technical Report UT-EECS-14-721. EECS Department, University of Tennessee, Knoxville. http://web.eecs.utk.edu/~plank/plank/papers/UT-EECS-14-721.html

[14] I. S. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960), 300–304. https://doi.org/10.1137/0108018 arXiv:https://doi.org/10.1137/0108018

[15] Bianca Schroeder and Garth A. Gibson. 2007. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)*. USENIX Association, Berkeley, CA, USA, Article 1. http://dl.acm.org/citation.cfm?id=1267903.1267904

[16] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. 2016. Flash Reliability in Production: The Expected and the Unexpected. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 67–80. https://www.usenix.org/conference/fast16/technical-sessions/presentation/schroeder