Reliability Stripe Coagulation in Two Failure Tolerant Storage Arrays

Thomas Schwarz
Department of Computer Science
Marquette University
Milwaukee, Wisconsin, USA
tschwarz@calprov.org, orcid: 0000-0003-4433-3360

John Rose Santiago
Department of Information Technology
Xavier Institute of Engineering
Mahim, Mumbai, Maharasthra, India
johnrose@xavier.ac.in, orcid: 0000-0001-8487-5370

***Abstract*—As the industry slowly transitions to data centers made up of electronic instead of magnetic storage components, the rate of device failure and page corruption will decrease, but not vanish. In this emerging environment, redundant storage is still required to safeguard data. We argue that the use of two-failure resilient linear codes with an exclusive-or (xor) based P-parity and a Q-parity calculated using a finite field operations is appropriate. We use the algebraic property of finite fields to show how to coagulate a number of small "constituent" reliability stripes into larger coagulated stripes without recalculating parities. This allows to protect largely inactive data with more storage efficient larger reliability stripes. The procedure is reversible.**

***Index Terms*—Erasure Correcting Codes, Extension of Codes, Linear Codes, Storage Systems**

I. Introduction

Storage and memory systems are made up of fallible components. With increasing numbers, the probability of failure also goes up. Faced with the possibility of failure, systems designers can opt to do nothing, to detect failures in order to prevent secondary damage, or to store data redundantly. The first course is examplified by phone owners who do not back up their data, the second is taken by DRAM manufacturers whose devices store a parity bit for each memory word to prevent an invalid word to be interpreted as valid data. Data centers that store petabytes of data will use erasure correcting codes, whereas high value data that cannot go through a reconstruction process after loss is mirrored or even triplicated.

The use of erasure correcting code does not come for free. One cost factor is the increased complexity of operations. It takes more effort to update data as now also parity data has to be updated. The other cost factor is financial, it costs money to buy and operate the additional devices needed. We can control the first by the number of parity pages that have to be updated and the second by the ratio of storage devoted to parity over the total number of storage, i.e. by adjusting the length of a reliability stripe.

In storage arrays using hard drives, silent corruption of disk blocks has been recognized as a source of data loss [6], [11]. If we try to access data stored in a corrupted disk block, we can usually recover the data because we still can access all other blocks in the reliability stripe. However, if we need the data in a corrputed block in order to deal with a failure in another block, then we need the capacity to recover from more than one failure. This observation lead to the development of RAID Level 6, where blocks located in $n-2$ of the n disks making up the storage array are placed in reliability stripes to which two blocks, the P- and the Q-parity, located in the remaining two disks are added. Of these, the P-parity is the exclusive-or (XOR) of the data blocks in the stripe, whereas the Q-parity is the result of a more involved calculation, frequently obtained using calculations in a finite field [9]. These type of stripes are also used in larger disk arrays where they incorporate blocks on n different disks, but the number of disks in the array is much larger.

As storage and memory moves to the use of very large non-volatile RAM technologies, the failure rates for these components will be less, but the same failure mechanisms experienced in large disk-based data centers will still happen, namely the loss of communication to a range of devices, the failure of enclosures, the failure of components, and the failure of individual storage units (pages). The size of the reliability stripe will still be an important design issue. If stripes are long, then as recovery of a lost block or two in a stripe involves reading all other blocks in the stripe, the load generated by recovery is high. If stripes are short, then the storage overhead for the parity is bad. Maintaining parity is also a problem. In the "small write", we update a block by reading its old contents, reading the contents of the P and Q-parity, calculate the new contents from the exclusive-or (XOR) of the old and new contents before we write the data and the newly calculated parity.. (This assumes that the Q-parity is obtained from a linear code, see below.) In a modern disk storage systems, incoming blocks of data (which can be much larger than the 4KB physical blocks) are streamed into stripes and when the stripe is full, the P and Q-parity data is written. This is much more efficient.

Data in storage exhibit strong access characteristics. First, in many situations, data is actually never accessed after it is written or after a certain time after creation. For instance, medical images are rarely consulted after the end of treatment, but for legal reasons or for rare medical cases, need to stay available. Secondly, much data goes from an active phase after creation to an inactive phase where accesses are very rare. The HP AutoRAID was an example of such a system [14]. Fresh data was stored in mirrored form, where recovery and updates

978-1-6654-5987-7/23/$31.00

are simple and efficient, but after a while data was migrated to a RAID Level 5 configuration. Unfortunately, the commercial version was not a success because the product could not overcome the problem of trashing where data blocks oscillate between the mirrored and the erasure correction protected states, but the principal idea is sound.

We propose a similar mechanism where data blocks migrate back and forth between reliability stripes of small and large length. Our mechanism allows more than two stripe sizes. We envision its use in large data storage devices using solid state technology, either Flash or one of the storage class memory technologies currently under development such as phase change memories or second generation magnetic RAM. Our work is inspired by that of Pâris, Estrada-Galiñas, Amer, and Rincón, who showed how the use of entanglement codes [3], [2] can be used to expand and shrink reliability stripes defined by two-dimensional erasure correcting codes (2D codes) [5], [12]. The contribution of these authors is stronger, as Pâris et al. use the entanglement coding idea in a clever way to counter-act the decrease in reliability that results from using larger reliability stripes and is the cost of lower storage overhead. The other inspiration is the work by Plank, Greenan and Miller [8], [16] who showed how one can exploit features of modern CPU by vectorizing Galois field calculations. We exploit here the fact previously observed by Zhou and Tian [16] that calculation in small Galois fields are more efficient than in large Galois fields.

Our proposal is based on the mathematics of finite fields, used in the definition of linear codes. We review this in the next section. We then discuss how these codes are and will be used in the future. We then describe our proposal on how to stitch together ("coagulate") several small reliability stripes into a larger ones, using the previously calculated parities of the smaller stripes. Finally, we discuss how to integrate this proposal into large storage installations.

II. Mathematical Preliminaries

General linear codes are defined over finite fields, which we have to explain first. Because of the binary nature of data storage in Information Storage, i.e. because we store data as arrays of bits, the finite fields whose elements can be naturally interpreted as bit strings are the most important. (There is nothing fundamental in Computer Science to this, as many modern storage and memory technologies use cells that in principal can use any number of levels, for example multi-level flash drives. However, current semi-conductor technology uses bits forcing storage technologies to use a small power of two as the number of levels.) Mathematically, (up to isomorphy), there is only one finite field with 2^l elements. We write it as $\mathscr{F}_{2^l}$. It is also called the Galois field with 2^l elements.

A. Galois Field Definition

The elements of $\mathscr{F}_{2^l}$ are bit strings of length l. The actual operations will depend on the exact definition of the field, but the following is standard. The addition of two field elements is given by the exclusive-or (XOR) of two bit-strings. This operation is standard in modern computer architectures and among the fastest to be executed on a modern micro-processor. It is also an operation that smart storage devices might be capable of handling. As a consequence, the zero bit string $00\ldots0$ is the zero of the field. For the multiplication, we need to distinguish between implementation and mathematically most convenient definition. The latter interprets a bit string $a_0a_1a_2\ldots a_{l-1}$ as a polynomial $a_0+a_1\cdot t+a_2\cdot t^2+\ldots+a_{l-1}t^{l-1}$ with unknown t and coefficients in the field $\mathscr{F}_2=\{0,1\}$; in the latter the addition is the logical eXclusive OR (XOR) and the multiplication is the logical AND of two operands. This just means that $0+0=0,\ 0+1=1,\ 1+0=1,\ 1+1=0,$ $0\cdot0=0\cdot1=1\cdot0=0,$ and $1\cdot1=1$. The multiplication in $\mathscr{F}_{2^l}$ is then defined as the product of the two polynomials resulting from the two bit strings, modula a *generator* polynomial $g(t)\in\mathscr{F}_2[t]$, which is irreducible (not the non-trivial product of smaller polynomials), and has degree l. There is actually quite a number of generator polynomials to choose from. Choosing the right generator polynomial simplifies the calculation of the product of two bit strings, but this is the case only if we use the definition for the implementation. Using pre-processing of various kinds we can simplify the calculation [4]. Plank and Miller found that using the Intel SIMD instructions yields multiplication implementations that are blazingly fast and published the GF-Complete library in C++ that implements them and other operations useful for erasure coding for bit-strings of size 4, 8, 16, 32, 64, and 128 [8], [10]

The non-zero elements of $\mathscr{F}_{2^l}$ form a cyclic group with multiplication. Thus, every non-zero element can be given as a power $\alpha^i, 0\le i<2^l-1$ with a certain element $\alpha\in\mathscr{F}_{2^l}$. In fact, we can always achieve that α is the bit string $010\ldots0$. Of course, the one in $\mathscr{F}_{2^l}$ is the bit string $10\ldots0$. If we can represent two non-zero elements as powers of α, namely $x=\alpha^i$ and $y=\alpha^j$, then obviously $xy=\alpha^{i+j}$, where the addition is taken modulo 2^{l-1}. This mathematical connection between Galois field multiplication and addition in the integers modulo 2^{l-1} is the key to the coagulation of two reliability stripes, as we will see.

B. Erasure Tolerant Linear Codes

A *binary linear code* is defined by a generator matrix $\mathbb{G}\in\mathscr{F}_{2^l}^{r\times s}$ of size $r\times s$ with coefficients in the Galois field $\mathscr{F}_{2^l}$ with 2^l elements.

A generator matrix maps an *information word*, a column vector $\mathbf{x}=^t(x_1,x_2,\ldots,x_s)\in\mathscr{F}_{2^l}^s$ to a code word $\mathbf{y}=^t(y_1,y_2,\ldots,y_r)\in\mathscr{F}_{2^l}^r$ by left multiplication

$$\mathbf{y}=\mathbb{G}\cdot\mathbf{x}.$$

The generator matrix is *systematic* if $s<r$ and the top r rows

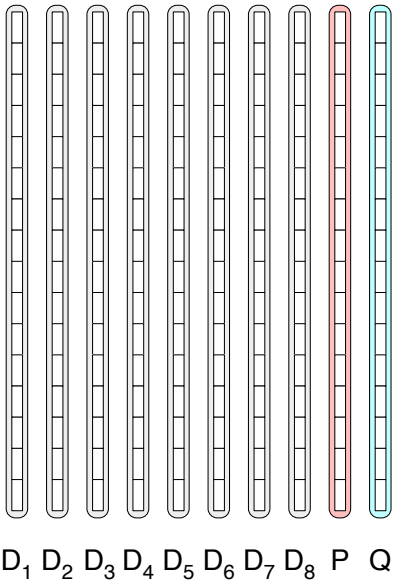

Fig. 1. Two Failure Tolerant Storage Array with eight data elements and two parity elements.

form an identity matrix, i.e. if

$$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & 0 & & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ c_{s+1,1} & c_{s+1,2} & c_{s+1,3} & \dots & c_{s+1,s-1} & c_{s+1,s} \\ \vdots & \vdots & & & \vdots & \vdots \\ c_{r,1} & c_{r,2} & c_{r,3} & \dots & c_{r,s-1} & c_{r,s} \end{pmatrix}$$

If $\mathbb{G}$ is systematic, then the first s coordinates of $\mathbb{G} \cdot \mathbf{x}$ are the same. Thus, multiplication by $\mathbb{G}$ merely adds $r-s$ coordinates to $\mathbf{x}$.

We employ linear codes in order to store data redundantly. The data itself is a stream of bits stored in the pages of memory or a storage device. We assume that this stream of bits is divided into finite field elements. Figure 1 depicts eight data pages D_1, …, D_8 as tall, rounded rectangles. The sixteen smaller rectangles inside represent field elements, bit strings of a given, constant size. To the ensemble of eight data pages, we add two *parity pages* P and Q. We use a systematic generator matrix to calculate the contents of the parity pages. The first element in each data page is arranged into a row vector $\mathbf{x} \in \mathscr{F}_{2^l}^8$. We then multiply $\mathbf{x}$ with a systematic generator matrix $\mathbb{G} \in gf^{10\times 8}$. The result, $\mathbf{y}$ is a ten-dimensional row vector, whose first eight coefficients are the same as the corresponding coefficients in $\mathbf{x}$. The two additional, i.e. the nineth and tenth coefficient become the first part of the contents of the parity pages P and Q, respectively. We continue the procedure with the second field element in each data page, until we reach the end. In other words, the contents of the P and Q parities are calculated as a linear combination of the contents of the data pages.

The purpose of the parity pages is of course to recover from the failure or unavailability of data pages. Since the content of the data pages are independent of each, the information in an ensemble of s data pages and $r-s$ parity pages is exactly that of s pages. Thus, information has to be lost if less than s pages are available. However, if the generator matrix is chosen well, than any combination of s pages among the r contains the original information in the data pages. Assume that a data page is no longer readable. For example, we have encountered a corrupted page. We select s readable pages in the ensemble. One of them is a parity page. The contents of the parity page is calculated from the data pages using a linear function. If we call d_i the finite field symbol in the i^{th} data page at the same offset and similar p for the corresponding parity symbol, then

$$p = a_1 \cdot d_1 + a_2 \cdot d_2 + \ldots + a_s \cdot d_s.$$

Assume that data page x has failed. We can calculate d_x from the other d_i and p if (and only if) a_x is not-zero. Thus, by making sure that all a_i are non-zero, we can use this parity page to recover the contents of a single bad data disk.

In general, if all $s \times s$ sub-matrices of $\mathbb{G}$ are invertible, then any s pages in the ensemble of $r+s$ pages can be used to regenerate the data in the original s data pages. In this case, the resulting code is *$r-s$-erasure correcting*.

C. Two Failure Correcting Codes

In most settings, it turns out that the capability of restoring a single data page in a group is not sufficient. On the other hand, a capacity of restoring three inaccessible pages is usually not needed. We therefore concentrate here on two-failure correcting codes and systems.

Subject to the last observation, different generator matrices will have different performance. First, the first parity page can be calculated using the "true parity", meaning choosing all coefficients to be equal to 1. This leaves the selection of the second parity row in $\mathbb{G}$.

We call the generator matrix and the resulting code *two-failure correcting*, if the contents of any two pages can be recalculated from the other s pages in an $s+2$ ensemble. We have the following, surprisingly simple criterion:

Lemma 1: Let

$$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & 0 & & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 1 & 1 & \dots & 1 & 1 \\ c_1 & c_2 & c_3 & \dots & c_s - 1 & c_s \end{pmatrix}$$

Then $\mathbb{G}$ is two-failure correcting if and only if the coefficients c_1, c_2, …, c_s are all different and non-zero.

Proof: We assume that two pages are not available. By what we have already observed, we can assume that these are two

data pages. We can recover these data pages if and only if the $s \times s$ square matrix $\mathbb{G}'$ obtained by removing the corresponding rows from $\mathbb{G}$ is invertible. We can use repeatedly cofactor expansion along the first $s-2$ rows to show that

$$\det(\mathbb{G}') = \begin{vmatrix} 1 & 1 \\ c_i & c_j \end{vmatrix} = c_j - c_i,$$

where i and j are the missing rows. This shows that all coefficients in the last row have to be different in order to reconstruct two missing data pages. If we instead loose the first parity page and a data page, we can reconstruct if (and only if) the last-row coefficient corresponding to the data page is not zero, as we have already seen.

D. Multiplicative Structure and Field Extensions

The algebraic structure imposed by multiplication is surprisingly simple. In Mathematical language, the multiplicative group (all non-zero elements of the field with the multiplication) is cyclic. This means that all element $x \in \mathscr{F}_{2^l}$, $x \neq 0$ can be written as β^i, $i \in \{0,\ldots,2^l-2\}$ with a primitive element $\beta \in \mathscr{F}_{2^l}$. By choosing a primite element β, we can quickly generate all 2^l-2 elements.

The complexity of the implementation of multiplication depends on the size of the elements, i.e. their length as bit strings. Sometimes it is advantageous to see a given field as a "field extension" $\mathbb{F}_{2^{lk}}$ of $\mathscr{F}_{2^l}$. The multiplicative group of the former has $2^{lk}-1$ elements and the latter 2^k-1. Not surpisingly, 2^k-1 divides $2^{lk}-1$ as $d = \sum_{\nu=0}^{l-1}(2^\nu)^k = \frac{2^{kl}-1}{2^k-1}$. If β is a primitive element of $\mathbb{F}_{2^{lk}}$, then β^d and all of its powers is an element of $\mathscr{F}_{2^l} \subset \mathbb{F}_{2^{lk}}$. It follows that β^d is a primitive element for $\mathscr{F}_{2^l}$.

III. Storage Array Organization

Erasure coding was first deployed for storage systems, with Patterson et al. invention of Redundant Arrays of Inexpensive Disks (RAID) that proposed to gather a small number of small, inexpensive disks to take on the role of a large, expensive disk. Originally intended to be a faster replacement of a single disk, the reliability aspect became the most important one and the "inexpensive" disks became "independent" disks. The original RAID Level 5 consisted of a small number m of disk drives to which was added a parity disk for an ensemble of $m+1$ disks. A dedicated parity disk however needs to be updated whenever any data changes. To equalize the otherwise unbalanced load, each disk is divided into a multiple of $m+1$ regions, so that each disk carries the same amount of parity regions than all others [7].

In large storage installations, this balancing can be left to random placements. A stream of incoming data is placed in pages, these pages are assigned to storage components (using the meta-data server, see below), placed into reliability stripes, and as the stripe fills up with user data, the parity data is calculated and placed eventually into parity pages. Ideally, once a user page is written, it will no longer be changed. However, if a user page changes, it is still possible to calculate the new parity pages from (1) the content of the previous contents of the user page, (2) the current content of the parity page to be calculated, and (3) the new content of the user page. In a disk based system, this means reading the one data block and the two parity blocks and then writing all these blocks. Between reading and writing, the disk needs to make a full rotation, making this a cumbersome operation. In a large storage installation, we do not need to store old and new data pages in the same location, which incidentally allows us to order the operations in such a way that a crash during an update never looses data, even if a failure occurs.

IV. Our Proposal

Our task is to coagulate a number of stripes $d_{i,1}, d_{i,2}, \ldots d_{i,m}, p_i, q_i \quad i = 0,1,\ldots,r$ into a single stripe without wasting the calculation of the parities p_i and q_i. Furthermore, the calculation of the Q-parities for the constituent stripes that will make up the coagulated stripe need to be the same.

A. Definition

Since the P-parity of the new stripe is the exclusive-or (XOR) of the data pages, this is trivially solved. The P-parity of the coagulated stripe is the XOR of the P-parities of the constituent stripes. The Q-parities are calculated using non-zero, mutually different Galois field elements, which are given as powers β^i of a primitive element, thus are determined by their *discrete logarithm* i where i is in $\{0,1,\ldots,2^l-2\}$.

We calculate the Q-parities of the constituent parities with coefficients β^d, β^{2d}, β^{3d}, …, β^{md}. The Q-parity of the coagulated stripe is then calculated with coefficients

$$\begin{matrix} \beta^d & \beta^{2d} & \ldots & \beta^{md} \\ \beta^{d+1} & \beta^{2d+1} & \ldots & \beta^{md+1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta^{d+r-1} & \beta^{2d+r-1} & \ldots & \beta^{md+r-1} \end{matrix}$$

which presupposes that $r < d$ and $md+r$ is smaller than the number of non-zero elements in the underlying Galois field. As a result, the Q-parity q of the coagulated stripe is calculated from the Q-parities q_i as

$$q = \beta^0 \cdot q_0 + \beta \cdot q_1 + \beta^2 \cdot q_2 + \ldots \beta^{r-1} \cdot q_{r-1}.$$

This is particularly useful if the underlying Galois field is composite and β^d is the primitive element of the subfield, because the calculation of the constituent stripe Q-parities are then performed in a smaller field that can be optimized.

B. Operations

Under normal circumstances, incoming data is organized into streams and each stream is written into data pages. The data pages are placed into constituent stripes. As the pages are written, the P- and Q-parities are calculated. The reasons to break incoming data into streams are two-fold: We want to place related data into the same stripes, because related data is more likely to change in the same transaction. We want to have several streams because the calculation of Q-parities at

least is slower than writing to a page (unless magnetic disks are used) we might need to paralize the calculation.

Fresh or freshly changed data stays in small constituent stripes as they are more likely to be changed or deleted. At a convenient time, constituent stripes are collected and coagulated.

When a page (or a set of pages) in a coagulated stripe is changed, its constituent stripe will be removed from the coagulated stripe. To this end, we first calculate the P- and Q-parities of the constituent stripe. To fix notation, assume that this is constituent stripe i in the notation above with P-parity p_i and Q-parity q_i. The data pages in the constituent stripe are $d_{i,1}$, $d_{i,2}$, ..., $d_{i,m}$ and the corresponding coefficients β^{d+i}, β^{2d+i}, ..., β^{md+i} for the calculation of the Q-parity in the coagulated stripe. The Q-parity of the constituent stripe is

$$q_i = \beta^d \cdot d_{i,1} + \beta^{2d} \cdot d_{i,2} + \ldots + \beta^{md} \cdot d_{im}.$$

The contribution to the Q-parity of the complete coagulated stripe is, as we have seen above, $\beta^i \cdot q_i$. Similarly, for the P-parity, the contribution of the constituent string is just p_i, the P-parity of the constituent string.

We then replace the P-parity and Q-parity of the coagulated stripe by

$$\begin{aligned} p &= p + p_i \\ q &= q + \beta^i \cdot q_i. \end{aligned}$$

(As these calculations are made in the Galois field, addition is the same as subtraction.) With these changes, the new P- and Q-parities of the coagulated stripe are the correct parities for a coagulated stripe where the data pages of the i^{th} constituent stripes are zero pages (i.e. made up only of zero bytes).

We call a coagulated stripe where one or more constituent stripes are made up of zero pages a coagulated stripe with holes. To fill a hole, we calculate the P- and Q-parities of a new constituent stripe, select a hole i and use the same calculations as before.

C. Use of Field Extensions

While our calculations work in any Galois field $\mathscr{F}_{2^l}$ of characteristic two, we can make use of field extensions to simplify Q-parity calculations in the constituent stripes. Then β^d and its powers are in a smaller GF field. We can decompose each symbol in a data page, representing an element of the larger Galois field into smaller bit strings each representing an element of the smaller Galois field $\{0, \beta^d, \beta^{2d}, \ldots\}$ and use the faster implementation of the multiplication there.

As an example consider the field extension $\mathscr{F}_{2^8} \subset \mathscr{F}_{2^{16}}$. The element of the first are bit-strings of length eight, whereas the elements of the latter are bit-strings of length sixteen. If β is a primitive element of $\mathscr{F}_{2^{16}}$, then β^{257} is a primitive element of $\mathscr{F}_{2^8}$. With this choice of fields, a constituent stripe has a maximum length of 255 whereas the coagulated stripe can consists of up to 257 constituent stripes. If we actually were to select maximum stripe sizes, the coagulated stripe could consist of 65535 data pages (plus two parity pages), sizes that would exceed the needs and capabilities of most storage systems.

For a more realistic example, we consider $\mathscr{F}_{2^4} \subset \mathscr{F}_{2^8}$. With these choices, constituent reliability stripes contain up to fifteen user pages and the coagulated stripes contain up to 255 user pages. The choice of such a small base field is motivated by the efficiency with which the Plank-Greenan-Miller method [8] can generate the Q-parity, as the Intel PSHUFB instruction (and its correspondent instructions in other instruction sets) uses arrays of nibbles for its operation. The parity overhead goes from 11.8% to 0.8% through coagulation. Finally, there is no technical reason to combine both examples and have two levels of coagulation, which reduces the overhead to as little as 0.003%, though in practice, the length of reliability stripes is limited by the total number of independent devices and the cost of reading all the other pages in the stripe during data reconstruction.

V. Implementation

The original RAIDs were composed of a small number of disks and today still remain a popular product for the small office or home office market. However, our proposal takes aim at data centers with a large number of constituent devices. Because we assume them to be no longer magnetic hard drives but electronic, the failure characteristics will be different. We will still have to contend with rare page failures, where a page cannot be read, with rare device failures, and more importantly, with communication failures where the overall system can no longer communicate with a device. We assume that the underlying storage network is fault tolerant. This assumption excludes the possibility of a network partition, where devices in each partition still function, but cannot communicate with devices in the other partition while they still might be able to process requests from some clients. Our assumption means that only a few devices will be inaccessible at a time.

Even if the underlying storage technology changes, the principal architecture of the system as examplified with Ceph [13] will remain. A distributed storage architecture is supervised by a meta-data server. This meta-data server is responsible for breaking up user data into pages, place pages into reliability stripes, insure that all pages in a reliability stripe are physically stored at different devices, move pages between devices whenever the size of the installation changes, and also to coagulate stripes with inactive data. As the experience with Ceph has shown, the underlying data storage has to be using a specialized file system [1].

The practical length of a coagulated reliability stripe will be determined by (1) the capability to find pages on different components (Otherwise a component failure will lead to data loss), (2) the costs of reconstructions since all but one of the pages need to be read, and finally, (3) the overall increase in the probability of loosing more than two pages in a stripe including those whose failure was discovered during the reconstruction attempt. Of these, the second reason is likely to be the most restricting one. After each failure, the storage array enters a period of vulnerability where any unrelated failure is

more likely to lead to dataloss. For good overall reliability, this period of vulnerability needs to be kept as short as possible [15]. If a reliability stripe has n data pages, and a container with many pages fails, then we need to read n times the contents of a container. As we can assume that assignment of pages to reliability stripes is governed by random assignments, this reconstruction load is reasonably evenly distributed over the whole storage array. Nevertheless, this additional load needs to be managed. As we cannot set apart a large part of the IO capacity of the array for reconstruction load, the stripe size n is limited. This limit can be extended by smart caching that redirects data influx to a cache for a limited time. As the underlying devices become more reliable, we will also see this limit increase.

VI. Conclusion

Currently, much data storage in the world is stored on devices, using magnetic technology (disks, tapes), but the future is electronic. While electronic devices tend to be more failure resistent, we still will have to face their fallability. This means storing data redundantly. Device failure can be silent, for instance, corruption of a single page. Such a failure can only be detected by accessing the page. We can assume that easily detectable failures such as loss of network access are more frequent.

Under these circumstances, using a single parity for redundancy is not sufficient as a hidden failure is likely to be only discovered during a reconstruction triggered by an open failure. However, unlike for disk based storage facilities where time between disk failures is measured in hours instead of days, adding more redundancy is not needed. As parity generation is involved and its storage is costly, we foresee the need for 2-failure tolerant codes. These codes will pack data pages into a reliability stripe and add two parities to it. The best choice for parity generation is the exclusive-or for one, the P-parity, and a linear combination of the data pages calculated over a Galois field, the Q-parity.

The length of the reliability stripe is one of the factors that determines data survival rates. As stripes get longer, more data needs to be accessed to deal with a missing page, increasing the "reconstruction load". However, the parity overhead also goes down.

In this paper we have shown how to aggregate a number of small reliability stripes into a larger, "coagulated" stripe. The increase is appropriate for stripes with (largely) inactive data. The coagulation process only uses already calculated parities and is reversible. It is simple and can be integrated into existing storage systems like Ceph. Our method is based on the basic algebraic structure of Galois fields.

References

[1] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File systems unfit as distributed storage backends: lessons from 10 years of ceph evolution," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 353–369.

[2] V. Estrada-Galinanes, E. Miller, P. Felber, and J.-F. Pâris, "Alpha entanglement codes: practical erasure codes to archive data in unreliable environments," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 183–194.

[3] V. Estrada Galinanes and P. Felber, "Helical entanglement codes: An efficient approach for designing robust distributed storage systems," in *Stabilization, Safety, and Security of Distributed Systems: 15th International Symposium, SSS 2013, Osaka, Japan, November 13-16, 2013. Proceedings 15*. Springer, 2013, pp. 32–44.

[4] K. M. Greenan, E. L. Miller, and T. Schwarz, "Optimizing Galois field arithmetic for diverse processor architectures and applications," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*. IEEE, 2008, pp. 1–10.

[5] 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, 1994.

[6] I. Iliadis, R. Haas, X.-Y. Hu, and E. Eleftheriou, "Disk scrubbing versus intradisk redundancy for RAID storage systems," *ACM transactions on storage (TOS)*, vol. 7, no. 2, pp. 1–42, 2011.

[7] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, 1988, pp. 109–116.

[8] J. Plank, K. Greenan, and E. L. Miller, "Screaming fast Galois field arithmetic using Intel SIMD extensions," in *Proceedings of the 11th Conference on File and Storage Systems (FAST 2013)*, Feb. 2013.

[9] J. S. Plank, "A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems," *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, 1997.

[10] J. S. Plank, E. L. Miller, K. M. Greenan, B. A. Arnold, J. A. Burnum, A. W. Disney, and A. C. McBride, "Gf-complete: A comprehensive open source library for Galois field arithmetic version 1.02," University of Tennessee, Tech. Rep. UT-CS-13-716: [PMG+13], 2014.

[11] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012022, jul 2007. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/78/1/012022

[12] T. J. E. Schwarz, "Reliability and performance of disk arrays," Ph.D. dissertation, UC San Diego, 1994.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 307–320.

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

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

[16] T. Zhou and C. Tian, "Fast erasure coding for data storage: A comprehensive study of the acceleration techniques," *ACM Transactions on Storage*, mar 2020.