# Trapezoid Quorum Protocol Dedicated to Erasure Resilient Coding Based Schemes

Théodore J. R. Relaza, Jacques Jorda and Abdelaziz M'zoughi
*Institut de Recherche en Informatique de Toulouse (IRIT)*
*Université Paul Sabatier, Toulouse, France*
{*relaza* | *jorda* | *mzoughi*}*@irit.fr*

***Abstract*—In distributed storage systems like parallel filesystems or storage virtualization middleware, data replication is the mainly used solution to implement data avaialability. The more replicas are distributed among nodes, the more robust is the storage system. However, the price to pay for this dependability becomes significant, due to both direct costs (the price of disks) and indirect costs (the energy consumption of this large amount of disks needed).**

**In order to lower the disk space needed for a given availalbility, Erasure Resilient Codes (referred to as ERC after this) are of interest and start to be implemented in this context. However, the use of such codes involves some new problems in data management. In fact, if some constraints like data concurrency can be solved using classical ways, others like coherency protocols need some adaptations in order to fit this context.**

**In this paper, we present an adaptation of trapezoid protocol in the context of ERC schemes (instead of full replication [12]). This new quorum protocol shows an increase of storage space efficiency while maintaining a high level of availability for read and writes operations.**

***Keywords*-Data availability; error codes; consistency; quorums;**

## I. INTRODUCTION

The most used way to implement reliability in distributed filesystems is data replication: when a data is stored, $n$ replicas ($n \geq 1$) are created and distributed among nodes. The higher is $n$, the more robust is the filesystem. The most commonly used replication level is triple block replication ($n = 2$), because of its balance between cost and reliability [11]. However, the cost of this replication is becoming unbearable as data stored grows. While 5 Exabytes of data are produced every 2 days (i.e. 0,8 ZB for the next year!), multiplying data replicas to ensure storage reliability is no longer a solution.

The most efficient solution to balance between redundancy and storage space consumed in parallel storage systems is the use of erasure coding [2]. For trivial performances reasons, only systematic codes are used: the original data is part of the encoding process output and thus distributed as is among nodes. To ensure an optimal failure resilience, MDS code are commonly used. In such codes, $k$ blocks are mapped into $n$ total blocks called a stripe (the $k$ original blocks and $n - k$ redundant blocks), and any $k$ blocks chosen over the $n$ may be used to reconstruct any of the $k$ original blocks. These MDS codes may be categorized according to their repair capabilities. When one node fails, the blocks it owned have to be reconstructed. These new blocks just have to ensure that they still form a valid $(n, k)$ MDS code. Thus, these new blocks may be either identical to the original ones (*exact repairs) or different, but still allowing to retrieve the original blocks using some computations (*functional repairs). An hybrid category uses an exact repair for the $k$ original blocks, and a functional repair for the $n - k$ redundant blocks. This question is of importance because an exact repair allows to quickly retrieve the original data when a read is performed, but it requires either more storage or additional bandwidth for repair operations. In contrast, a functional repair will only require a message broadcast to an information flow graph to reconstruct data blocks [5] but a more heavy processing will be required to retrieve or update the original data. Thus, these code are of less interest in the context of distributed storage.

The main problematic is then: how to efficiently support nodes failures. In fact, ERC-based schemes involves a significant network and IO overhead to reconstruct the original data when some blocks are missing. Thus, new storage schemes have been proposed, almost all sharing the same goals and making the same assumptions. First, they wish to optimize the update process. A basic approach requires read and write operations on $n - k + 1$ blocks of the stripe [2]. For exemple, a (9,6)-MDS will require 8 read and write operations for a single block update: one read and one write for the target block, and one read and one write for each of the three redundant blocks. The goal is then to create new ERC codes allowing in-place updates using the commutativity of Gallois fields operations. Secondly, they wish to optimize the recovery process. In fact, if a node failure occurs, the blocks stored on that node have to be computed back using the redundant blocks and the reverse algorithm used. This process may be very compute-intensive and may have a significant impact on the storage system performances.

These goals and constraints have led to some assumptions shared by almost all existing studies. The main one is the consideration of append-only filesystems. To be more precise, the updates are considered as new writes, the old versions of blocks being garbage-collected using a lazy process. This assumption is not problematic since as of now,

these storage systems are mainly used to store infrequently changed data (e.g. using a REST API, or backing up classical replicated data over an ERC storage system). As a direct result, no consistency protocol is used, or rely on a weak protocol. However, this assumption is not suited to all kind of storage. For exemple, when users' data stored on virtual disks is accessed by several virtual machines, a strict consistency protocol is required in any case to avoid incoherent data.

Although data consistency protocols have been extensively studied, they all target classical data replication schemes. These protocols cannot be used as is on ERC-based schemes, as the access to a single block of the stripe is not a sufficient condition: since an operation may fail on some nodes due to hardware issues, the conditions for read and writes are enforced. Thus, even if data availability and storage space is significantly improved using such ERC-based schemes, the real availability of read and write operations regarding nodes failures should be computed.

Our objective is to propose a new coherency protocol dedicated to such storage schemes. Built upon the classical trapezoidal protocol [12], it is extended to handle $(n, k)-$MDS code. Read and write availability are studied and compared to the space gained

## II. CONTEXT AND RELATED WORK

Replication has been used for a long time to implement data availability in distributed storage systems. However, the information explosion draws the limits of this technique [14]. To ensure data availability with a lower storage resources usage, Erasure Resilient Codes (ERC codes) are being studied in the context of distributed storage systems [6]. Maximum Distance Separable codes (MDS codes) are optimal to that end since they allow an adequate tradeoff between reliability and storage space used: an $(n, k)$ MDS code splits the data in $k$ blocks and adds $n - k$ additional redundant blocks to overcome up to $n - k$ nodes failures. Thus, $n$ and $k$ may be chosen with respect to the storage needs.

Some very recent works have been published in this domain. In [10], specific encoding / decoding techniques are used on top of RS codes to decrease both network traffic and disk IO during reconstruction stage. However, this scheme is used to backup in an erasure-encoded storage data already stored in a replicated way. Thus updates are not handled and no consistency mechanism is proposed. The same assumptions are made in [11], and a similar constraint is assumed in [9]: the storage systems targeted by their scheme are append-only, and no coherency protocol is implemented. The assumptions are slightly differents in [3]. Outlining that a strong consistency is often required by end users (especially enterprise customers), an attempt to conciliate consistency and availability if proposed. However the storage is always append-only, and no updates are allowed on node failures.

To allow a real strong consistency among blocks, some protocol must be implemented. In the full replication context, many works have been published on the subject. The most basic approach, named ROWA (Read One Write All), requires a successful write of all blocks, thus implying that any single block read will give the latest value. The two main drawbacks of this system are the write penalty (each replica must be written before the operation to complete) and the lack of reliability of the write operations (since all target nodes are required, any failure prevent these operations).

Quorum systems have been introduced to overcome these difficulties. In these protocols, the nodes required to read and/or write blocks are restricted to sets named quorums. The most trivial implementation, called Majority Quorum, requires a strict majority of nodes for both read and write operations [13]. Given that condition, the set of nodes used for any read operation will intersect the set of nodes used for any write operation. Many logical structured have been proposed for these sets: in diamonds [7], grids [4], trapeze [12], trees [1] or even hypercubes [8]. However, all these works only deal with full replication and do not encompass ERC schemes.

The reason why they are not suited to distributed storage using erasure codes is the difference between the original data block and the other ones. In fact, on full replication, any node giving the adequate latest version of a block can be used to retrieve the corresponding data. The case is different on ERC-based schemes: if the original data block have the latest version, it can be used to retrieve the data. But in the opposite case, $k$ blocks having the latest version must be found to reconstruct the original block. This leads to significant differences in the computation of read and write availabilities for these protocols, and thus has to be studied.

## III. MODEL

Our work focus on consistency protocols dedicated to distributed storage systems using ERC instead of full replication. After a brief recall on a classical implementation of Erasure Resilient Codes, we are going to present our data consistency protocol.

### A. Erasure Resilient Codes

In our storage system, we use Erasure Resilient Codes (ERC) to store the data. An (n, k) MDS (Maximum Distance Separable) erasure code, stores the original $k$ data blocks into the $k$ nodes out of $n$ nodes and generates $n - k$ redundant blocks such that any $k$ nodes out of $n$ nodes can reconstruct the original data.
Let $\{N_1, N_2, ..., N_k, N_{k+1}, ..., N_n\}$ be the $n$ nodes where $\{N_1, N_2, ..., N_k\}$ store the original $k$ data blocks and $\{N_{k+1}, ..., N_n\}$ store the $n - k$ redundant blocks.
Let $\{b_1, b_2, ..., b_k, b_{k+1}, ..., b_n\}$ be the $n$ data blocks where $\{b_1, b_2, ..., b_k\}$ are the original $k$ data blocks and $\{b_{k+1}, ..., b_n\}$ are the $n - k$ redundant blocks generated.

For $k+1 \leq j \leq n$,

$$b_j = \sum_{i=1}^{i=k} \alpha_{j,i}.b_i \quad (1)$$

Where $\alpha_{j,i}$ are carefully chosen constants, and arithmetic is over some finite field, usually **GF**$(2^h)$ [2].

### B. Data Consistency Protocol

In this section, we introduce an adapted version of the trapezoidal quorum protocol dedicated to ERC schemes. We will define the necessary and sufficient conditions for read and write operations to work properly, and propose the respectives algorithms.

*1) Defintions:* A write quorum ($WQ$) is a set of nodes required to write data. More precisely, $WQ$ is the set of nodes to be updated successfully in order to complete a write operation. $|WQ|$ denotes the size of the write quorum. Similarly, a read quorum ($RQ$) is a set of nodes required to read data. More precisely, the replicas of a data being all retrieved from the nodes of $RQ$, one is sure to have among them at least one replica with the latest version. $|RQ|$ denotes the size of the read quorum.
For a given $RQ$ and $WQ$ in a same quorum system, there is at least one node which belongs simultaneously to $RQ$ and $WQ$ [12], in the other words the following condition must be held:

$$RQ \cap WQ \neq \emptyset \quad (2)$$

This condition ensures that a read quorum contains at least one node with a chunk updated (i.e. with the latest version). If WQ1 and WQ2 are two write quorums in the same quorum system, then

$$WQ1 \cap WQ2 \neq \emptyset \quad (3)$$

This condition ensures that two successive write operations on an object will have at least one node in common, and thus that each write quorum contains at least one node with the latest version.

*2) Design:* In the trapezoidal protocol [12], the nodes are arranged on a logical trapezoid that has $h+1$ levels ($0 \leq l \leq h$). The level $l = 0$ contains $b$ nodes and the $l$-th level ($1 \leq l \leq h$) contains $s_l = a.l + b$ nodes, where $a$ and $b$ are two integers with $a \geq 0$ and $b \geq 1$. Therefore, the number of nodes used to store one block in this protocol is equal to

$$Nb_{node} = \sum_{l=0}^{l=h} s_l \quad (4)$$

Using ERC as a data distribution technique, the nodes $\{N_i, N_{k+1}, ..., N_n\}$ are used to store the block $b_i$ where $1 \leq i \leq k$. The original data block $b_i$ is stored in $N_i$ and the redundant blocks $\alpha_{j,i}.b_i$ are stored in $N_j$ for $k+1 \leq j \leq n$. Therefore, in order to ensure the consistency of block $b_i$, the protocol will organize the nodes $\{N_i, N_{k+1}, ..., N_n\}$ in a logical trapezoid. The node $N_i$ (which contains the original data block) is placed in level $l = 0$ of trapezoid. Thus,

$$Nb_{node} = n - k + 1 \quad (5)$$

Figure 1 in page 3 shows an example of nodes organized in a logical trapezoid $Nb_{node} = n - k + 1 = 15$.

*3) Write quorum:* An ***Erasure Resilient Code compliant write quorum*** is any set of nodes constituted by $w_l$ arbitrary-chosen nodes in each level.

$$|WQ| = \sum_{l=0}^{l=h} w_l \quad (6)$$

Where $w_0 = \lfloor \frac{b}{2} \rfloor + 1$, $1 \leq w_l \leq s_l$ for $1 \leq l \leq h$.
With this definition of a write quorum, for any two quorum WQ1 and WQ2, the property $WQ1 \cap WQ2 \neq \emptyset$ is always verified.

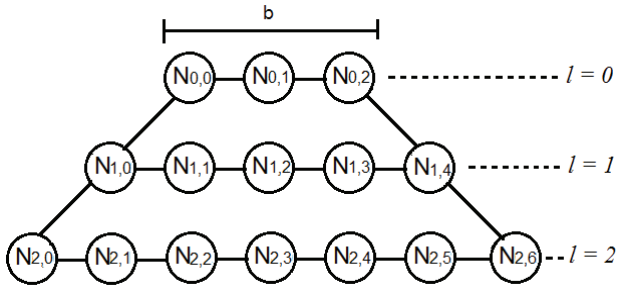


Figure 1. Trapezoid protocol for $Nb_{node} = 15$ and $s_l = 2l + 3$ ($a = 2$, $b = 3$, $h = 2$)

**Proof.** Let WQ1 and WQ2 be two write quorums in a system with N nodes in which we select $n$ ($n \leq N$) nodes to store the blocks. Then, each write quorum contains an absolute majority of nodes residing at level $l = 0$. This guarantees that these two write quorums contain at least one common element in level $l = 0$. Therefore, equation (3) is verified. □

The algorithm 1 describes the way write operations are implemented for a trapezoidal protocol dedicated to (n,k) MDS erasure code. The protocol starts to write the different bocks from level $l = 0$ up to level $l = h$ (see algorithm 1 line 16). To validate a write operation in a level $l$, write operation must be done successfully in at least $w_l$ nodes out of $s_l$ nodes residing in this level. Therefore, the write operation fails if the protocol can't write at least $w_l$ nodes in any level $l$ where $0 \leq l \leq h$) (see algorithm 1 lines 35 to 37).

*4) Read quorum:* We want to read the data block $i$ where $0 \leq i \leq k$. There are two steps in order to read this block.

1) **Step 1 (Checking version):** The latest version can be checked in one of the $h+1$ levels ($0 \leq l \leq h$) by comparing the version of the block in the $s_l - w_l + 1$ nodes out of $s_l$ nodes residing in this level. If the latest version is found then go to step 2. Otherwise, the read operation fails.
2) **Step 2 (Read or decode the data block):** If node $N_i$ contains the latest version then the protocol read directly the block value from this node. Otherwise, the data block is decoded using any $k$ nodes out of $n$ nodes with the latest version.

The algorithm 2 describes the way read operations are implemented for a trapezoidal protocol dedicated to (n,k) MDS erasure code. The protocol starts to check the latest version of data block in a given level $l$ by comparing the version of $s_l - w_l + 1$ nodes out of $s_l$ nodes residing in this level. The checking version starts from level $l = 0$ to level $l = h$ (see algorithm 2 line 11). If the latest version is found in a given level then one of the two following cases will apply depending on the version of block storing in the node which contains the original data block(see algorithm 2 lines 30 to 36), the Case 1 will be launched if this node contains the latest version and the Case 2 otherwise:

- **Case 1:** The protocol read directly from the node which contains the original data block.
- **Case 2:** The decode operation will be launched using any $k$ updated nodes out of $n$ nodes in order to reconstruct the orginal data block.

## IV. ANALYSIS

We are now going to evaluate the availability of the read and write operations and the space used depending on the values of

**Algorithm 1** Write data block $x$ in node $N_i$

**Definitions**
$s_l$, $w_l$, $h$ ▷ Trapezoid protocol parameters
$n$, $k$ ▷ ERC parameters
$U_l$ ▷ List of nodes residing at level $l$
▷ Where $U_0 \cup U_1 \cup ... \cup U_l = \{N_i, N_{k+1}, N_{k+2}, ..., N_n\}$
$V$ ▷ $kx(n-k)$ matrix : version of redundant nodes
▷ Where $V(i, j-k)$ denotes version of $\alpha_{i,j}.b_i$ for $1 \leq i \leq k$ and $k+1 \leq j \leq n$
▷ $V(:, j-k)$ denotes all elements of $V$ in column $j-k$

**procedure** INIT
  **for** $l \leftarrow 0, h$ **do**
    $U_l = \{u_1^l, u_2^l, ..., u_{s_l}^l\}$ ▷ from l-th level
  **end for**
**end procedure**

**procedure** WRITEBLOCK($i, x$) ▷ Write block value $x$ in node $N_i$
  $[chunk, version] \leftarrow$ READBLOCK($i, id$) ▷ Read old block value
  **for** $l \leftarrow 0, h$ **do**
    $counter \leftarrow 0$
    **for all** $u \in U_l$ **do**
      **if** $(u = N_i)$ **then**
        $state \leftarrow u.write(x)$ ▷ Write $x$ in node $N_i$
        **if** $state = VALID$ **then** ▷ Write operation is done
          $counter \leftarrow counter + 1$
        **end if**
      **else** ▷ $u = N_j$ where $k < j \leq n$
        $V[:, j-k] \leftarrow u.version(id)$
        **if** $V[i, j-k] = version$ **then**
          $state \leftarrow u.add(\alpha_{i,j}.(x - chunk))$
          ▷ Where $N_j.add(buf)$ denotes $b_j \leftarrow b_j + buf$ and write it in $N_j$
          **if** $state = VALID$ **then**
            $counter \leftarrow counter + 1$
          **end if**
        **end if**
      **end if**
    **end for**
    **if** $counter < w_l$ **then**
      **return** $FAIL$ ▷ Write block is failed
    **end if**
  **end for**
  **return** $SUCCESS$ ▷ Write block is done
**end procedure**

**Algorithm 2** Read data block in node $N_i$

**Definitions**
$s_l$, $w_l$, $h$ ▷ Trapezoid protocol parameters
$n$, $k$ ▷ ERC parameters
$U_l$ ▷ List of nodes residing at level $l$

**procedure** INIT ▷ List of nodes in the system
  **for** $l \leftarrow 0, h$ **do**
    $U_l = \{u_1^l, u_2^l, ..., u_{s_l}^l\}$ ▷ from l-th level
  **end for**
**end procedure**

**procedure** READBLOCK($i, id$) ▷ $id$: id of data
  **for** $l \leftarrow 0, h$ **do**
    $counter = 0$
    $version \leftarrow -1$
    **for all** $u \in U_l$ **do**
      **if** $(u = N_i)$ **then**
        $temp \leftarrow u.version(id)$
        **if** $temp \neq INVALID$ **then**
          $version \leftarrow max(version, temp)$
          $counter \leftarrow counter + 1$
        **end if**
      **else** ▷ $u = N_j$ where $k < j \leq n$
        $V[:, j-k] \leftarrow u.version(id)$
        **if** $version \neq INVALID$ **then**
          **if** $V[i, j-k] > version$ **then**
            $version \leftarrow V[i, j-k]$
          **end if**
          $counter \leftarrow counter + 1$
        **end if**
      **end if**
      **if** $(counter = s_l - w_l + 1)$ **then**
        **if** $(version = N_i.version(id))$ **then**
          **return** $[N_i.read(id), version]$ ▷ Data block is available in $N_i$
        **else** ▷ Need to reconstruct the data block
          **return** decode($i, id, V$) ▷ Using $k$ nodes out of n
        **end if**
      **end if**
    **end for**
  **end for**
  **return** $\emptyset$ ▷ Data is not readable
**end procedure**

various parameters. We are going to compute this probability for the two following cases:

1) A full replication storage system ensuring that each data block is stored on $n-k+1$ nodes for given parameters $n$ and $k$ ;
2) A $(n, k)$ MDS ERC based storage system.

These two cases implement the same level of availability, and thus are well suited to compare the capabilities of each scheme. We will then show some numerical evaluation to outline the balance between the operations availability and the storage space saved.

**Notations**

- $p$: denotes node availability, i.e. the probability that one given node is available in the system.
- $r$: denotes size of the read quorum for the trapezoid protocol in the general threshold scheme context.
- $w_l$: denotes the minimal number of nodes required to write the block in level $l$.
- $r_l$: denotes the minimal number of nodes required to check the latest version of data in level $l$ ($r_l \equiv s_l - w_l + 1$).
- TRAP-FR: makes reference to Trapezoid Protocol in the full replication context.
- TRAP-ERC: makes reference to Trapezoid Protocol dedicated to erasure resilient codes context.

With no loss of generality, we assume that:

1) node availability is the same and equal to $p$ for all nodes in the system.
2) nodes fail independently of each other.
3) each node stops on failure (fail-stop).
4) there is no failure on communication links.

For readability, we use the following expression to refer the probability that at least *i* nodes and less than or equal to *j* nodes out of $z$ would be available.

$$\Phi_z(i, j) \equiv \sum_{k=i}^{k=j} \binom{z}{k} p^k (1-p)^{z-k} \tag{7}$$

### A. Write availability

The ***write availability*** $P_{write}$ represents the probability that one data block can be written into the system.

*1) TRAP-FR:* For the write operation of data block to succeed in a full replication system, the validation of blocks' write operation on at least $w_l$ nodes out of $s_l$ for each level $l$ ($0 \leq l \leq h$) is required. Consequently, write availability $P_{write}$ is equal to:

$$P_{write} = \prod_{l=0}^{l=h} \Phi_{s_l}(w_l, s_l) \tag{8}$$

*2) TRAP-ERC:* For the write operation of data block to succeed in this system, the validation of blocks' write operation

on at least $w_l$ nodes out of $s_l$ for each level $l$ ($0 \leq l \leq h$) is required. Consequently, write availability $P_{write}$ is equal to:

$$P_{write} = \prod_{l=0}^{l=h} \Phi_{s_l}(w_l, s_l) \tag{9}$$

### B. Read availability

The ***read availability*** $P_{read}$ represents the probability that one data block can be read from the system.

*1) TRAP-FR:* The probability that the data is readable using the nodes in level $l$ is equal to $\Phi_{s_l}(r_l, s_l)$.
Therefore,

$$P_{read} = 1 - \prod_{l=0}^{l=h} (1 - \Phi_{s_l}(r_l, s_l)) \tag{10}$$

*2) TRAP-ERC:* For readability, we use the following expressions:

$$\beta_l \equiv \begin{cases} \max(0, r_l - 2) & \text{for } l = 0 \\ r_l - 1 & \text{for } 1 \leq l \leq h \end{cases} \tag{11}$$

$$\lambda_l \equiv \begin{cases} s_l - 1 & \text{for } l = 0 \\ s_l & \text{for } 1 \leq l \leq h \end{cases} \tag{12}$$

Let $P_1$ and $P_2$ be respectively the probability that the data block can be read without recovery operation (i.e. the protocol can found directly a quorum read) and the probability that the data block can be read after recovery operation.

$$P_1 = p.\left(1 - \prod_{l=0}^{l=h} \Phi_{\lambda_l}(0, \beta_l)\right)$$

$$P_2 = (1-p).\Phi_{n-1}(k, n-1)$$

Then, $P_{read} = P_1 + P_2$

$$P_{read} = p.\left(1 - \prod_{l=0}^{l=h} \Phi_{\lambda_l}(0, \beta_l)\right) + (1-p).\Phi_{n-1}(k, n-1) \tag{13}$$

### C. Storage space used

$D_{used}$ and $blocksize$ represent respectively the size of disk used to store a data block and the size of one data block.

*1) TRAP-FR:* The data is replicated in the $n - k + 1$ nodes. Each contains the same block which equal to the original data block.
Therefore,

$$D_{used} = (n - k + 1).blocksize \tag{14}$$

*2) TRAP-ERC:* In order to store the block $b_i$ ($1 \leq i \leq k$) using this protocol, we need to write $\{b_i$ , $\alpha_{k+1,i}.b_i$, $\alpha_{k+2,i}.b_i$, ..., $\alpha_{n,i}.b_i\}$. The size of $b_i$ and $\alpha_{j,i}.b_i$ ($k + 1 \leq j \leq n$) are respectively equal to $blocksize$ and $\frac{blocksize}{k}$. Then,

$$\begin{aligned} D_{used} &= blocksize + (n-k).\frac{blocksize}{k} \\ &= (1 + \frac{n-k}{k}).blocksize \\ &= \frac{n}{k}.blocksize \end{aligned}$$

$$D_{used} = \frac{n}{k}.blocksize \tag{15}$$

### D. Simulations

In the following figures, $w$ parameter ($1 \leq w \leq s_1$) refers to:

$$w_l = \begin{cases} \lfloor \frac{b}{2} \rfloor + 1 & \text{for } l = 0 \\ w & \text{for } 1 \leq l \leq h \end{cases} \tag{16}$$

This parameter is compliant with the condition imposed by write quorum in page 3 ($1 \leq w_l \leq s_l$ for $1 \leq l \leq h$).

The first noticeable point is that the write availability is the same in the case of full replication and ERC schemes (see equations 8 and 9). In fact, the use of Erasure Resilient Codes do not alter the number of nodes required to successfully write a block on the system. Fig. 2 shows various cases for $n = 15$. It should be noted that like on the full replication case, the write availability is not significantly impacted by the number of replicas for usual values of $p$ (i.e. for $p > 0.9$).

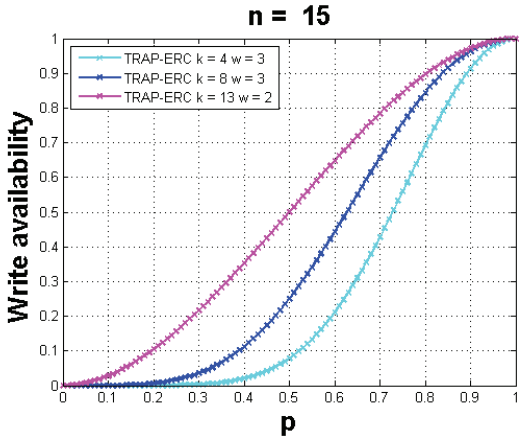


Figure 2. Write availability of TRAP-ERC as a function of the node availability $p$

Conversely, using ERC scheme impacts the read availability, as shown on fig. 3. For exemple, when $p = 0.5$, the write availability of the full replication scheme is about 75% while it is just 63% when an ERC scheme is used. However, there is no difference when $p \geq 0.8$, i.e. for usual values of nodes availability.

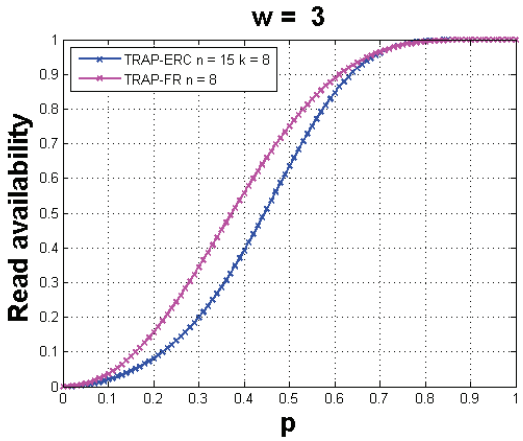


Figure 3. Read availability of TRAP-ERC and TRAP-FR as a function of the node availability $p$

Read availability is also impacted by both the parameters used to construct the trapezoid (the parameter $w$), and the number of redundant blocks (the difference $n-k$). The greater this difference is, i.e. the higher is the number of redundant blocks, the better is the read availability (see fig. 4).

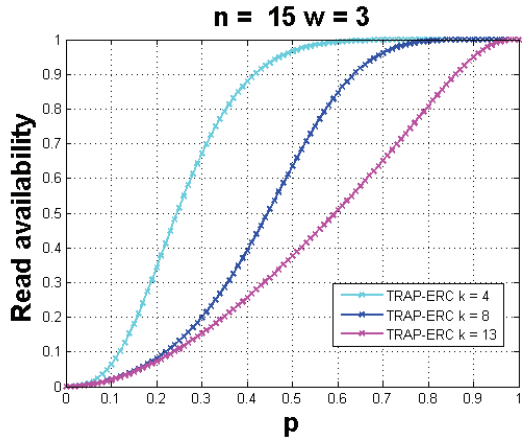


Figure 4. Read availability of TRAP-ERC as a function of the node availability $p$

The figure (fig. 5) outline the interest of ERC based schemes. It compares the storage space used (total space used divided by the block size) in the case of an ERC based system and a full replication system, for various values of $k$. For example, when $n = 15$ and $k = 8$, a full replication system uses $8$ blocks while an ERC based system uses only $4$ block. Thus, with these parameters, the use of Erasure Resilient Codes allows the storage space to be reduced by $50\%$ compared that of a full replication system

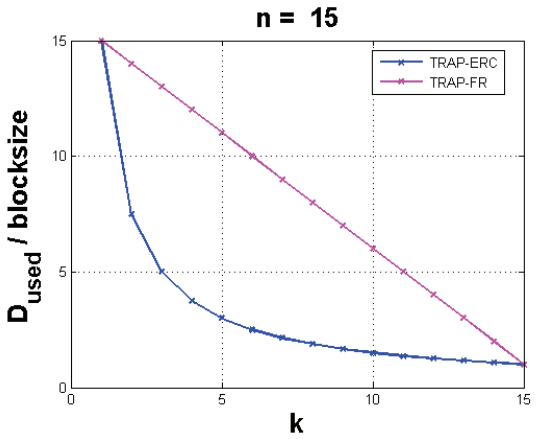


Figure 5. Storage space used by TRAP-ERC and TRAP-FR divided by $blocksize$ as a function of the node availability $k$

## V. CONCLUSION

Erasure Resilient Codes are being used in the context of distributed systems to overcome the data explosion. In fact, while more and more data are produced every day (and thus have to be stored) the use of a classical replication system is no longer a solution.

Most of the works published as of now focus on the optimisation of the code itself: the objectives are to reduce the network and IO bandwidth used on the recovery process. And most of the works published as of now are based on the same assumptions: data is globally immutable (i.e. read only or append only) and no consistency is necessary. If these assumptions are true for some cloud storage applications, it misses a large part of storage needs...

In this paper, we have studied an adaptation of the trapezoidal protocol in the context of ERC based distributed storage systems. Our goal is to implement a strong consistency mechanism even when some nodes fail. Using such a protocol allows to enlarge the use of ERC based storage systems to a number of applications such as virtual machines data storage. We have shown that write availability is not impacted by these codes, whereas read availability is slightly decreased. However, we have outlined the difference of storage space used in both cases, which confirms the interest of Erasure Resilient Codes.

## REFERENCES


[1] Divyakant Agrawal and Amr El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comput. Syst.*, 9(1):1–20, February 1991.

[2] Marcos K. Aguilera, Ramaprabhu Janakiraman, and Lixao Xu. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the 2005 International Conference onDependable Systems and Networks*, DSN'05, pages 336–345. IEEE, 2005.

[3] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[4] Shun Yan Cheung, Mostafa H. Ammar, and Mustaque Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 438–445, Washington, DC, USA, 1990. IEEE Computer Society.

[5] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theor.*, 56(9):4539–4551, September 2010.

[6] Alexandros G. Dimakis, Kannan Ramchandran, Yunnan Wu, and Changho Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.

[7] Ada Wai-Chee Fu, Yat Sheung Wong, and Man Hon Wong. Diamond quorum consensus for high capacity and efficiency in a replicated database system. *Distrib. Parallel Databases*, 8(4):471–492, October 2000.

[8] Ada Waichee Fu, Fu Wai, Kwong Lau, Fuk Keung, Ng Man, and Hon Wong. Hypercube quorum consensus for mutual exclusion and replicated data management. *Computers and Mathematics with Applications, An International Journal*, 36(5):45–59, 1998.

[9] Cheng Huang, Minghua Chen, and Jin Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *Trans. Storage*, 9(1):3:1–3:28, March 2013.

[10] K.V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan 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*, SIGCOMM '14, pages 331–342, New York, NY, USA, 2014. ACM.

[11] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment, 2013.

[12] Tabito Suzuki and Mamoru Ohara. Analysis of probabilistic trapezoid protocol for data replication. *Proceeding of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, 2005.

[13] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, June 1979.

[14] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 328–338, London, UK, UK, 2002. Springer-Verlag.