RAPIDS: Reconciling Availability, Accuracy, and Performance in Managing Geo-Distributed Scientific Data

Lipeng Wan
Department of Computer Science
Georgia State University
Atlanta, GA, USA
lwan@gsu.edu

Jieyang Chen
Department of Computer Science
University of Alabama at Birmingham
Birmingham, AL, USA
jchen3@uab.edu

Xin Liang
Department of Computer Science
University of Kentucky
Lexington, KY, USA
xliang@uky.edu

Ana Gainaru
Oak Ridge National Laboratory
Oak Ridge, TN, USA
gainarua@ornl.gov

Qian Gong
Oak Ridge National Laboratory
Oak Ridge, TN, USA
gongq@ornl.gov

Qing Liu
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ, USA
qliu@njit.edu

Ben Whitney
Oak Ridge National Laboratory
Oak Ridge, TN, USA
whitneybe@ornl.gov

Joy Arulraj
School of Computer Science
Georgia Institute of Technology
Atlanta, GA, USA
arulraj@gatech.edu

Zhengchun Liu
Argonne National Laboratory
Lemont, IL, USA
liuzhengchun@gmail.com

Ian Foster
Argonne National Laboratory
Lemont, IL, USA
foster@anl.gov

Scott Klasky
Oak Ridge National Laboratory
Oak Ridge, TN, USA
klasky@ornl.gov

ABSTRACT

In modern science, big data plays an increasingly important role. Many scientific applications, such as running simulations on supercomputers or conducting experiments on advanced instruments, produce huge amount of data at unprecedented speed. Analyzing and understanding such big data is the key for scientists to make scientific breakthroughs. However, data might become unavailable for scientists to access when outages or maintenance of the storage system occur, which severely hinders scientific discovery. To improve the data availability, data duplication and erasure coding (EC) are often used. But as the scientific data gets larger, using these two methods can cause considerable storage and network overhead.

In this paper, we propose RAPIDS, a hybrid approach that combines the multigrid-based error-bounded lossy compression with erasure coding, to significantly reduce the storage and network overhead required for maintaining high data availability. Our experiments show that RAPIDS reduces the storage overhead by up to 7.5x and network overhead by up to 3x to achieve the same level of availability compared to the regular EC method. We improve RAPIDS by building two models to optimize the fault tolerance configurations and data gathering strategy. We demonstrate that RAPIDS significantly improves performance when running on many CPU cores in parallel or on GPUs.

CCS CONCEPTS

• Information systems → Data management systems; **Storage replication**; **• Computer systems organization → Availability**; **Redundancy**.

KEYWORDS

scientific data Management, data availability

ACM Reference Format:
Lipeng Wan, Jieyang Chen, Xin Liang, Ana Gainaru, Qian Gong, Qing Liu, Ben Whitney, Joy Arulraj, Zhengchun Liu, Ian Foster, and Scott Klasky. 2023. RAPIDS: Reconciling Availability, Accuracy, and Performance in Managing Geo-Distributed Scientific Data. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23), June 16–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3588195.3592983

1 INTRODUCTION

As more advanced technologies and equipment are heavily involved in scientific discovery, massive volume of scientific data is being generated at enormous speed in modern science. For instance, the XGC gyrokinetic particle-in-cell code [27], which simulates the edge region of magnetically confined thermonuclear fusion plasma, can generate 1 PB of output data every 24 hours when running on the fastest supercomputers in the U.S., and is expected to produce 10 PB per day in the near future. Similarly, in astrophysics, although dramatic data reduction will be performed during observation, the Square Kilometer Array (SKA) [60], which is an ongoing project with the aim of building the largest radio telescope in the world, is still projected to produce data at 1 PB/s in the next 10–20 years.

These humongous scientific datasets are usually stored on the high-performance, large-capacity data storage systems maintained by the government-funded research institutes, such as the national laboratories. Although extensive resources have been invested in building these storage systems to improve their reliability, system outages caused by hardware/software failures, security issues, or even human errors, are still inevitable. Besides these unplanned downtime, the storage system might also be inaccessible during the scheduled maintenance time. All these incidents that prevents scientists from accessing their data in timely manner, can lead to disastrous consequences. For example, once ITER (the world's largest experimental tokamak nuclear fusion reactor) [24] is in production, scientists expect to control the experiment running on ITER by analyzing the historical experiment data as well as the simulation data. If the storage system is suddenly out of service and the data is unavailable for access, the experiment has to be terminated or postponed. Therefore, improving the availability for large scientific data has become extremely critical since the productivity of scientific discovery heavily relies on the data accessibility, which must be assured.

Prior Work. In practice, two approaches are commonly used to improve the data availability. The first approach is data duplication, which produces multiple copies of the data and distributes them to storage systems that are located at different places. For example, we duplicate the original data on the local storage system twice and transfer the two copies to two independently operated remote storage systems. As a result, we can always access the data unless all of these three storage systems are unavailable simultaneously. Apparently, this approach can lead to significant storage and network overhead especially when the size of the original data is large. In our example, if the original data is 20TB, not only do we need 40TB extra storage capacity on the remote systems to store the two data copies, but transferring them also consumes a lot of network bandwidth.

The second approach is based on erasure coding, which has been widely used in distributed storage systems. When the erasure coding is applied to the original data, the data is split into fragments and additional fragments called "parity fragments" are also created to provide redundancy. For example, let us assume the original data is still 20TB, and 4 data fragments and 2 parity fragments are generated after erasure coding. Then we distribute these fragments to 6 independently operated storage systems. The mathematics behind the erasure coding guarantees that the original data can be fully reconstructed with any 4 of these 6 fragments. In this case, we can also tolerate two concurrent system outages while the wasted storage capacity for storing the 2 parity fragments is reduced to $20 \times 2/4 = 10$TB. Although adopting erasure coding reduces the storage and network overhead compared to duplicating the data, it is still expensive to create, transfer and store many parity fragments to achieve high availability for large scientific datasets.

Our Approach. In this paper, we propose RAPIDS, a hybrid method that combines the error-bounded lossy compression with erasure coding, to further reduce the storage and network overhead while maintaining the same level of data availability. The rationale behind our method comes from two observations: 1) Due to the large data volume, scientists have already been adopting lossy compression techniques to reduce their data as long as the errors or distortions in the reduced representation of their data are under control [5, 33]. 2) Since the lossy compression can reduce the data size by 10-1000x depending on the error bound, the total overhead might still be small even more redundancies are added to the compressed data.

Specifically, RAPIDS leverages the unique features provided by a multigrid-based error-bounded lossy compression algorithm, called pMGARD [34], to decompose and refactor the scientific data into a hierarchical reduced representation, where the size of each level increases from top to the bottom. The original data can be reconstructed using only the top level and the error of the reconstructed data decreases with more lower levels being used in the reconstruction. Erasure coding with different fault tolerance configurations are then applied to each level so that the number of parity fragments created for each level decreases from top to the bottom. On the one hand, adding more redundancies to the upper levels does not incur too much overhead due to their small sizes. On the other hand, the overall data availability is improved when creating more parity fragments for the upper levels since they are more critical to the data construction than the lower ones. More importantly, rather than letting the user intuitively choose the fault tolerance configuration for each level, RAPIDS can automatically determine the optimal amount of redundancies that should be applied to each level in a systematic and quantitative way, which takes system failure rates, data accuracy, as well as storage/network overhead into account. The evaluation results demonstrate that, compared to the regular erasure coding method which applies the same configuration to the entire data, RAPIDS reduces the storage overhead by up to 7.5x and network overhead by up to 3x to achieve the same level of availability.

Moreover, three particular challenges are addressed in our method. The first two are: 1) how to determine the fault tolerance configuration for each level to minimize the expected relative error of the restored data given certain overhead budget, and 2) how to select the fragments that should be gathered for data restoration to minimize the gathering latency. We build two optimization models to formulate these two challenges. To solve the first optimization problem, we propose a heuristic algorithm which achieves superior performance compared to the brute-force search. We address the second optimization problem by leveraging a mixed integer nonlinear programming (MINLP) solver. The evaluation results show that, using the optimized fault tolerance configurations does achieve the minimum expected relative error under the storage overhead budget, while adopting the optimal data gathering strategy also significantly reduces the gathering latency. The third challenge is the data refactoring and reconstruction operations our method relies on can be time consuming especially when the original data is large.

We conduct extensive evaluations to demonstrate that running RAPIDS on many CPU cores in parallel or on GPUs can significantly accelerate these operations, and thus improve the overall performance.

2 BACKGROUND

2.1 Data Availability and Fault Tolerance

As mentioned above, two commonly used approaches for improving the data availability are data duplication and erasure coding. In order to better understand the availability improvements that these two methods can provide, we need to quantify the likelihood of incidents in which the data becomes unavailable to the users when each of these two approaches is used.

Let us assume the probability of losing one of the n storage systems is p. If the data duplication method is used and m data replicas in total (including the original dataset) are stored on m of the n geo-distributed storage systems, the data will become unavailable when all the m storage systems that store the data replicas are out of service. Since we assume these storage systems are independently operated, the probability of this incident can be calculated through the following formula:

$$P_{\text{UNAVBL}} = \sum_{i=0}^{n-m} \binom{n-m}{i} p^{m+i} (1-p)^{n-m-i} \tag{1}$$

In this formula, $\binom{n-m}{i} p^{m+i}(1-p)^{n-m-i}$ represents the probability of an incident where $m+i$ storage systems are out of service simultaneously. Particularly, the m storage systems, on which the m data replicas are stored, are within these out-of-service ones for certain, while the remaining i unavailable storage systems can be any one of the $\binom{n-m}{i}$ combinations. In this case, the storage overhead, which is defined as the ratio of the storage capacity occupied by the redundant replicas to that by the original data, is $m-1$.

If the regular erasure coding method is used and m parity fragments are created, the $n-m$ data and m parity fragments are stored on those n geo-distributed storage systems. Since we can always restore the original data using any $n-m$ of the n fragments, the data will become unavailable when more than m storage systems are out-of-service at the same time. The probability of this incident can be calculated through the following formula:

$$P_{\text{UNAVBL}} = \sum_{i=m+1}^{n} \binom{n}{i} p^{i} (1-p)^{n-i} \tag{2}$$

In this case, the storage overhead caused by erasure coding is m/k, which is the ratio of the storage capacity occupied by those parity fragments to that by data fragments.

2.2 Scientific Data Refactoring

Scientific data differs from other types of data (such as texts, images, audios, videos, etc.) in the sense that it usually contains large multi-dimensional arrays which are mainly composed of floating-point values and exhibit strong spatial and temporal locality. On the one hand, using lossless compressors (such as Gzip [46], bzip2 [13], FPC [12], and FPZIP [37]), which can ensure that the decompressed data is exactly the same as the original data, usually cannot achieve satisfactory compression ratio on scientific data, due to the randomness of the tailing mantissa bits of those floating-point values [21]. On the other hand, a few lossy compression techniques for scientific data (such as SZ [15, 33, 54], ZFP [36], ISABELA [29], etc.) have also been proposed and extensively studied. Prior studies have shown that although certain amount of errors are involved during compression, lossily compressed data can still be used in many scenarios across a wide range of applications without impacting data integrity [16]. For instance, it has been used in computational chemistry code GAMESS to store intermediate data in memory for avoiding recomputation [20]. It has also been applied in full-state quantum circuit simulation to increase the size of the simulation [61]. In addition, it is applied to address exascale data challenges, including accelerating the checkpoint/restart process in parallel iterative solvers [55], reducing stream intensity for crystallography [56], and mitigating storage and I/O pressures for climate applications [9].

Different from regular lossy compressors [4–6, 10, 14, 29, 33, 36, 38, 51] which typically only respect a singular user-specified error bound, data refactoring enables the capability that the original data can be decomposed into multiple components such that the approximation of the original data can be reconstructed using a certain number of those components. For example, JPEG2000 [49] uses a block-based wavelet transformation approach to support flexible progressive data reconsecration. ZFP [36] also supports progressive data reconstruction, but it is only limited to a fixed-rate mode. Progressive data reconstruction was also extensively studied for exploring the cost-accuracy trade-offs in scientific visualizations [28, 30, 47, 48, 53]. Despite previous works have made extensive progress, only a few of them can guarantee error bound, which is essential to ensuring valuable scientific features contained in the data are not lost.

pMGARD, which is an extension of the MGARD multilevel decomposition approach [4–6], comes up with an error-controlled data refactoring and reconstruction framework for scientific data. Specifically, by using the multigrid-based decomposition and L2 projection, the original data is decomposed into multiple levels. Each level consists of the multilevel coefficients, which can be used to reconstruct the approximation of the original data. According to the mathematical theory of MGARD, the error of the reconstructed data can be bounded by $e \leq (1+\sqrt{3}/2) \times \sum_{l=0}^{L} \max_{x \in N_l^*} |u\_mc[x] - \widetilde{u}\_mc[x]|$, where e is the estimated error; L is the total number of decomposed levels; N_l^* is the set of nodes of level l; $u\_mc[x]$ is the original multilevel coefficient at position x; and $\widetilde{u}\_mc[x]$ is the multilevel coefficient after incurring error for the purpose of reducing data volume (please refer to [5] for more details). By controlling how much error that is incurred to each multilevel coefficient, the error of reconstructed data can be bounded. To achieve fine-grained error control, pMGARD applies bitplane encoding to the multilevel coefficients. Since different bitplanes at different levels have different relative importance to the precision of the reconstructed data, after encoding those bitplanes are further reordered in to a new set of levels for efficient and progressive data reconstruction level by level. In this way, the final refactored data will be in a hierarchical representation that consists of series of levels. The reconstruction process takes a certain number of levels to rebuild the approximation of the original data. Depending on how many levels are used, the reconstructed data will be bounded by different errors.

3 OPTIMIZATION IN MANAGING GEO-DISTRIBUTED SCIENTIFIC DATA

In this section, we provide an overview of our approach and present how to 1) find the proper fault tolerance configuration for the refactored scientific data to maximize the data fidelity given limited storage resources and 2) minimize the data access latency under constrained network bandwidth. All the mathematical symbols used in the model and their description are listed in Table 1.

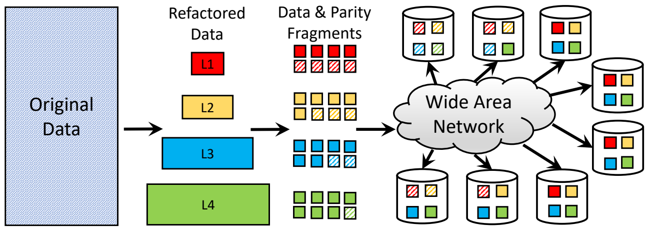
Table 1: Symbols used in our model

Symbol	Description
n	Total number of data storage systems
p	Probability of losing access to one storage system
N	Total number of concurrent failures
S	Total size of the original data
l	Total number of levels after refactoring
s_j	Data size of level j
e_j	Relative L-infinity error when using level $1, 2, \ldots, j$ to reconstruct the data
m_j	Number of failures level j can tolerate
B_i	Bandwidth when requesting data from storage system i
x_{ij}	$x_{ij} = \begin{cases} 1, & \text{if level } j\text{'s fragment is obtained from system } i \\ 0, & \text{otherwise} \end{cases}$

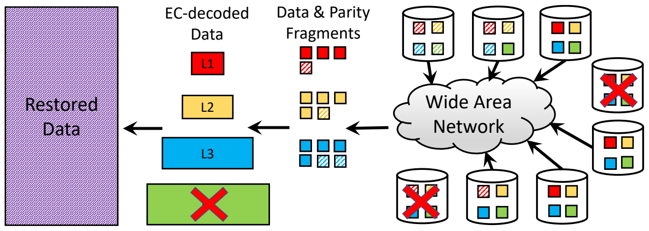
3.1 Overview of Our Approach

In order to achieve high data availability without incurring too much storage/network overhead, we propose to first leverage the multigrid-based error-bounded lossy compression technique to compress and refactor the original data into a hierarchical representation. Then we apply erasure coding to each level of this hierarchy with different fault tolerance configurations and the generated data and parity fragments are distributed to different remote storage systems.

Here we use an example to demonstrate the benefit of adopting our approach. As shown in Fig. 1(a), the original data is refactored into a hierarchical representation with four levels and the number of storage systems is eight. If the size of the original data is S and the sizes of these four levels are denoted by s_1, s_2, s_3 and s_4, we have $S > s_1 + s_2 + s_3 + s_4$ and $s_1 \ll s_2 \ll s_3 \ll s_4$. We then apply erasure coding to each level with different fault tolerance configurations. If k and m represent the number of data and parity fragments, then in this example we have: $k = 4$, $m = 4$ for the first level; $k = 5$, $m = 3$ for the second level; $k = 6$, $m = 2$ for the third level; $k = 7$, $m = 1$ for the forth level. As shown in Fig. 1(b), if two storage systems become unavailable at the same time due to failures or maintenance, we can fetch 4, 5, 6 fragments from the 6 remaining storage systems to restore the first, second and third level of the hierarchy respectively. However, the fourth level cannot be restored since the restoration requires at least 7 fragments of this level. Once the first three levels of the hierarchy are restored, we can reconstruct the original data within certain error bound through data decompression. Since the sizes of the upper levels are much smaller than the lower ones of the hierarchy, creating more parity fragments for the upper levels and less for the lower ones significantly reduces the overall storage and network overhead compared to directly applying erasure coding to the original data to achieve the same degree of availability.



(a) The original data is refactored into a hierarchical representation. Erasure coding is applied to each level of this hierarchy. The data and parity fragments of each level are distributed to different systems through the wide area network.



(b) The data and parity fragments are obtained from different remote systems based on their EC configuration and availability, to recover all or certain levels of the hierarchical representation. An approximation of the original data is restored using the recovered hierarchical representation.

Figure 1: An illustration of our approach

In our example, the number of parity fragments created for each level of this hierarchy are chosen just for the purpose of illustration. In practice, we need to carefully select the fault tolerance configurations for each level to achieve a good trade-off between availability, data accuracy and storage/network overhead. Moreover, when the data and parity fragments need to be gathered from the remote storage systems to reconstruct the original data, the data gathering strategy that determines which fragments should be fetched from which storage systems can significantly affect the transfer performance and must be optimized. We address these two problems in the following two subsections.

3.2 Optimizing Fault Tolerance Configurations for Refactored Scientific Data

If N storage systems fail simultaneously at certain time, which levels of the refactored data are available depends on the erasure coding scheme applied to each level, or in other words how many failures each level can tolerate. If the j^{th} level is encoded into $n - m_j$ data and m_j parity EC-fragments (n EC-fragments in total which are distributed to n storage systems, one per each system), it means level j can tolerate m_j concurrent failures. When we have $N \leq m_j$, then level j can be recovered from these failures. In order to reconstruct the original data with error e_j, level $1, 2, \ldots, j$ of the refactored data all need to tolerate at least N concurrent failures. Here we use the relative L-infinity error to quantify the error of reconstructed data, which is defined as

$$e = \frac{\max(\|d - \tilde{d}\|)}{\max(\|d\|)} \tag{3}$$

In this formula, $\max(\|d - \tilde{d}\|)$ represents the maximum absolute value of element-wise difference between the original data d and reconstructed data $\tilde{d}$, while $\max(\|d\|)$ represents the maximum absolute value in the original data.

Since we always generate more parity fragments for top levels in the hierarchy to have $m_1 > m_2 > \ldots > m_l$, level $1, 2, \ldots, j$ of the refactored data can all be recovered from the failures as long as $N \leq m_j$. Moreover, if we have $m_{j+1} < N \leq m_j$, it means the original data can be reconstructed with error e_j and the error cannot be further reduced. Therefore, if N storage systems become unavailable concurrently due to failures or maintenance and we assume these failures or maintenance are independent to each other, the probability that we can still reconstruct the original data with error e_j is

$$P(m_{j+1} < N \leq m_j) = \sum_{i=m_{j+1}+1}^{m_j} \binom{n}{i} p^i (1-p)^{n-i} \tag{4}$$

where p is the probability of one system failure or maintenance occurs. Particularly, if $N > m_1$, all levels of the refactored data will be unavailable for reconstruction due to the concurrent system failures. In such a scenario, although it is impossible to reconstruct the data to get $\tilde{d}$, we still define a relative error which will be used in our optimization model as a penalty for being unable to access any level of the refactored data. Basically, we create an array of the same size as the original data but contains only zeros. We use this array as $\tilde{d}$ in formula 3 and calculate the "relative L-infinity error" e_0 for the scenario when all levels of the refactored data are unavailable. Apparently, we have $e_0 = \frac{\max(\|d\|)}{\max(\|d\|)} = 1$, which can be interpreted as the "upper bound" of the relative L-infinity error. In fact, no data compression algorithm should allow the relative L-infinity error of the reconstructed data to be equal to or greater than 1, because that means the reconstructed data is completely useless. Given the definition of e_0, the expectation of the relative L-infinity error when using all the available levels to reconstruct the data can be calculated by

$$\begin{aligned} E =& e_0 P(N > m_1) + e_l P(N \leq m_l) \\ &+ \sum_{j=1}^{l-1} e_j P(m_{j+1} < N \leq m_j) \\ =& \sum_{i=m_1+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} + e_l \sum_{i=0}^{m_l} \binom{n}{i} p^i (1-p)^{n-i} \\ &+ \sum_{j=1}^{l-1} e_j [\sum_{i=m_{j+1}+1}^{m_j} \binom{n}{i} p^i (1-p)^{n-i}] \end{aligned} \tag{5}$$

Therefore, this can be formulated as an optimization problem and the overarching goal is to find the proper fault tolerance configuration for each level $(m_1, m_2, \ldots, m_l)$ to minimize the expectation of the relative L-infinity error under a few constraints.

The first constraint is that the storage overhead caused by enabling fault tolerance should not exceed certain threshold. Particularly, in order to make the j^{th} level of refactored data tolerate m_j storage system failures, we need to use the erasure coding to generate $n - m_j$ data fragments and m_j parity fragments. These data and parity fragments have the same size and are distributed among n storage systems. If the size of the refactored data at the j^{th} level is s_j, the size of each parity fragment of level j is $\frac{s_j}{n-m_j}$. Thus, the total size of parity fragments of that level is $\frac{m_j}{n-m_j} s_j$. The storage overhead caused by saving those parity fragments, which is quantified by the ratio of the total size of parity fragments to the total size of the original data, cannot exceed the threshold (ω) specified by the user:

$$W = \frac{\sum_{j=1}^{l} \frac{m_j}{n-m_j} s_j}{S} \leq \omega \tag{6}$$

The second constraint is that the number of failures each level of refactored data can tolerate should follow $n > m_1 > m_2 > \ldots > m_l \geq 1$. The assumption behind this constraint is that the size of refactored data at each level increases from the top to the bottom ($s_1 < s_2 < \ldots < s_l$). Therefore, adding more redundancies to the top levels will not incur a significant increase in the overall storage overhead.

If we put everything together, the complete optimization model we built is presented as follows, which is a nonlinear integer programming model.

$$\begin{aligned} \underset{m_j, \forall j \in \{1,\ldots,l\}}{\arg\min} \quad & e_0 \sum_{i=m_1+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} + \\ & e_l \sum_{i=0}^{m_l} \binom{n}{i} p^i (1-p)^{n-i} + \\ & \sum_{j=1}^{l-1} e_j [\sum_{i=m_{j+1}+1}^{m_j} \binom{n}{i} p^i (1-p)^{n-i}] \\ \text{s.t.} \quad & \frac{\sum_{j=1}^{l} \frac{m_j}{n-m_j} s_j}{S} \leq \omega, \\ & n > m_1 > m_2 > \ldots > m_l \geq 1 \end{aligned} \tag{7}$$

The most straightforward approach for solving this optimization model is to use brute-force search, especially when the number of candidate solutions is not large. In fact, the total number of candidate solutions for this model can be calculated by:

$$\sum_{k=1}^{n-l} (n-l-k+1) \frac{(k+1)k}{2} \tag{8}$$

If we have $U = n - l$, the time complexity of solving this optimization model with brute-force search is $O(U^4)$.

To mitigate the computational cost of brute-force search when U is large, we also propose a heuristic algorithm which significantly reduces the search space. First, we need to determine proper initial values for the fault tolerance configuration of each level. Let us use $M^i = [m_1^i, m_2^i, \ldots, m_l^i]$ to denote the initial fault tolerance configurations. If $m_l^i = m^*$ and we want to minimize the overall storage overhead, then we have $m_1^i = m^* + l - 1$, $m_2^i = m^* + l - 2$, $\ldots$, $m_{l-1}^i = m^* + 1$. We need to find the maximum value of m^* so that it satisfies the following inequality:

$$\frac{\frac{m^*+l-1}{n-(m^*+l-1)} s_1 + \frac{m^*+l-2}{n-(m^*+l-2)} s_2 + \cdots + \frac{m^*}{n-(m^*)} s_l}{S} \leq \omega \tag{9}$$

This set of initial values help us eliminate the searching within the candidate solutions that have $m_l < m^*$, since it is guaranteed that the optimal solution must have $m_l \geq m^*$.

Once the initial fault tolerance configurations are determined, we can use our heuristic algorithm (shown in Algorithm 1) to find the optimal fault tolerance configurations for each level. The design of

Algorithm 1: Finding Optimal FT Configurations

```
Input: Initial fault tolerance configurations $M^i = [m_1^i, m_2^i, \dots, m_l^i]$
Output: Optimal fault tolerance configurations $M^o = [m_1^o, m_2^o, \dots, m_l^o]$
$M = [m_1, m_2, \dots, m_l] = M^i$, $l_{curr} = l$, $M_{prev} = []$;
while True do
    $W = \frac{\sum_{j=1}^{l} \frac{m_j}{n-m_j} s_j}{S}$;
    if $W < \omega$ then
        foreach $1 \le x < l_{curr}$ do
            $m_x = m_x + 1$ ;
        end
        $M = [m_1, m_2, \dots, m_l]$ ;
    else
        foreach $1 \le x < l_{curr}$ do
            $m_x = m_x - 1$ ;
        end
        $M = [m_1, m_2, \dots, m_l]$ ;
        $l_{curr} = l_{curr} - 1$ ;
    end
    if $M == M_{prev}$ then
        break;
    end
    $M_{prev} = M$ ;
end
$M^o = M$;
```

this algorithm is based on two assumptions: 1) $s_1 \ll s_2 \ll s_3 \ll s_4$, and 2) $e_1 \gg e_2 \gg e_3 \gg e_4$. Basically, this algorithm intends to incrementally increase the number of parity fragments for each level starting from the bottom level. If increasing the number of parity fragments for current level leads to the violation of the storage overhead constraint, it starts increasing the number of parity fragments for the level above in the hierarchy. The time complexity of our heuristic algorithm is $O(U)$, which is a significant improvement compared to brute-force search.

3.3 Optimizing Data Gathering Strategy for Faster Data Reconstruction

When the original data is requested by the user, a subset of these fragments need to be gathered through the wide area network for data restoration. On the one hand, it is natural to request as many fragments as possible from the storage systems that have the highest network bandwidth. However, on the other hand, launching concurrent data transfer requests to the same storage system might cause bandwidth contention, leading to performance degradation even that system has a high network bandwidth. Therefore, we need to optimize the data gathering strategy to minimize the transfer latency, and thus shorten the overall data restoration time.

First of all, let us define a binary variable x_{ij} to indicate whether a fragment of level j should be transferred from storage system i or not. If $x_{ij} = 1$, it means the user should request a fragment of level j from storage system i; otherwise $x_{ij} = 0$. Now determining the data gathering strategy can be formulated as an optimization problem whose objective is to find a set of values for x_{ij} ($\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, l\}$) so that the data transfer time can be minimized.

Secondly, we need to calculate the data transfer time for each potential transfer request. Since requesting multiple fragments from the same storage system is allowed, the network bandwidth will be shared by multiple data transfer requests under that circumstance. For the sake of simplicity, here we assume the network bandwidth is equally shared among these requests. Under this assumption, if the total network bandwidth of storage system i is B_i, then the bandwidth each of these requests can get is $\frac{B_i}{\sum_{j=1}^{i} x_{ij}}$. Since the size of each fragment of level j is $\frac{s_j}{n-m_j}$, the time it takes to transfer a fragment of level j from storage system i to the user is $\frac{s_j}{n-m_j} \Big/ \frac{B_i}{\sum_{j=1}^{l} x_{ij}}$.

Thirdly, we set "minimizing the average data transfer time of all transfer requests" as the objective of our optimization model. In particular, the average data transfer time can be denoted by $\frac{\sum_{i=1}^{n}\sum_{j=1}^{l}\left(\frac{x_{ij}s_j}{n-m_j} \Big/ \frac{B_i}{\sum_{j=1}^{l} x_{ij}}\right)}{\sum_{i=1}^{n}\sum_{j=1}^{l} x_{ij}}$, where $\frac{x_{ij}s_j}{n-m_j} \Big/ \frac{B_i}{\sum_{j=1}^{i} x_{ij}}$ is the time it takes if a fragment is transferred from storage system i to the user (if $x_{ij} = 0$, it means no transfer happens, thus the transfer time is 0), and $\sum_{i=1}^{n}\sum_{j=1}^{l} x_{ij}$ is the total number of transfer requests. There are also a few constraints we need to satisfy in this optimization problem. Basically, for each level of the refactored data, we need to gather enough number of fragments to reconstruct that level. For instance, if level j needs to be reconstructed, we need to at least have $n - m_j$ fragments transferred to the user, meaning $\sum_{i=1}^{n} x_{ij} \ge n - m_j$. Moreover, if failures occur and certain storage systems are unavailable, we need to set the value of associated x_{ij} to 0. For example, if storage system i is unavailable, we have $x_{ij} = 0$ ($\forall j \in \{1, \dots, l\}$). If the total number of out-of-service storage systems $m_{j-1} \le N < m_j$, level $j, j+1, \dots, l$ cannot be recovered. In this case, only the inequality constraints associated with level $\{1, \dots, j-1\}$ are included. The complete optimization model for determining the optimal data gathering strategy is shown as follows:

$$
\begin{aligned}
\underset{x_{ij}}{\arg\min} \quad & \frac{\sum_{i=1}^{n}\sum_{j=1}^{l}\left(\frac{x_{ij}s_j}{n-m_j} \Big/ \frac{B_i}{\sum_{j=1}^{l} x_{ij}}\right)}{\sum_{i=1}^{n}\sum_{j=1}^{l} x_{ij}} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} \ge n - m_j,\ \forall j \in \{1, \dots, l\}, \\
& x_{ij} = 0,\ \text{if system } i \text{ is unavailable}
\end{aligned}
$$

4 IMPLEMENTATION DETAILS

The implementation of our approach is composed of four software components, which are the data refactoring and erasure coding component, the data distribution and gathering component, the metadata management component, and the data restoration component. We briefly introduce each of these components in this section.

4.1 Data Refactoring and Erasure Coding

This component executes the following operations in sequence: 1) It reads in the original data object from a scientific dataset stored on the local file system. 2) It refactors the original data to generate a hierarchical representation. The data refactoring code is implemented based on pMGARD (see introduction in Section 2.2). The source codes of pMGARD developed in C++ and CUDA are both available on GitHub [42]. 3) It uses the heuristic algorithm proposed in Section 3.2 to determine the fault tolerance configuration for each level of that hierarchy. 4) It applies erasure coding to each level based on the fault tolerance configuration given by the solver. The erasure coding is implemented by calling "liberasurecode" [57], which is an erasure coding library written in C that supports Reed-Solomon codes. 5) It writes the data and parity fragments of each level to separate files with self-describing formats through the HDF5 [26] or ADIOS [41] library. Using the self-describing file formats allows us to embed the information of the original data object (e.g., the data object name) into those data and parity fragments files. 6) It stores the refactoring and erasure coding related information (e.g., number of levels, number of data and parity fragments at each level, etc.) to a key-value database through the metadata management component.

4.2 Data Distribution and Gathering

The implementation of this component is a script that controls the file transferring between the local and remote storage systems through Globus. Particularly, finding the transfer destination, establishing the connection, and managing the transfer tasks are all realized by calling the Command Line Interface (CLI) of Globus [1]. Since we need to track which data and parity fragments are stored on which storage systems, this component also needs to interact with the metadata management component so that such information can be stored to or retrieved from the key-value database. For example, if a data or parity fragment is permanently lost or damaged, it can be restored by other available data and parity fragments through the erasure decoding process. The restored fragment might be stored on a new storage system. This component then informs the metadata management component to update the new location information of the restored fragment in the key-value database by this component. This component calls an MINLP solver to optimize the data gathering strategy if needed. The MINLP solver used is called MIDACO [43], which is developed based on an evolutionary algorithm known as Ant Colony Optimization.

4.3 Metadata Management

This is a critical component in our implementation. Not only does this component track all the data and parity fragments of each level of the refactored data for every data object, but it also maintains the information needed for reconstructing the original data using the refactored data. Furthermore, the throughput of each data transfer is also recorded by this component, which can be used to update the bandwidth parameters in our data gathering strategy optimization model so that the results of our model can adapt to any network bandwidth variation. This component adopts RocksDB [50], a lightweight key-value store optimized for fast, low latency storage, as the database backend. When the user need to restore certain data object, the metadata can be fetched from the RocksDB through this component and used to guide the data gathering and reconstruction. Currently, the metadata is only maintained on one system, which is prone to failures. In future development, the metadata duplication and distributed metadata management will be added.

4.4 Data Restoration

This component first reads in the gathered data and parity fragments and calls "liberasurecode" library to erasure decode the data to restore the hierarchical representation. Then it interacts with the metadata component to retrieve the information needed for the data reconstruction from the database. Finally, it calls the decompression function in pMGARD to reconstruct the original data.

5 EVALUATION

In this section, we present the evaluation results of the proposed approach to demonstrate its advantages and limitations compared to existing methods.

5.1 Experimental Setup

5.1.1 Evaluation Platform. All our experiments are conducted on Andes, which is a 704-compute node commodity-type linux cluster built for data processing/analysis purposes at Oak Ridge Leadership Computing Facility (OLCF). Each of these 704 compute nodes on Andes has two AMD EPYC 7302 16-Core Processors (32 cores in total) and 256GB memory. Besides these regular compute nodes, Andes also has 9 GPU nodes. Each of these GPU nodes has two NVIDIA K80 GPUs, two Intel Xeon E5-2695 14-core processors and 1TB memory. Andes and OLCF's Summit supercomputer share the same POSIX-based IBM Spectrum Scale parallel Filesystem called Alpine. Therefore, once the data produced by those large-scale scientific simulations running on Summit are stored into Alpine, users can access and process the data from Andes as well.

5.1.2 Globus Data Transfer Logs. In our experiments, the transfer of data and parity fragments between local and remote storage systems relies on Globus, which is a widely used software-as-a-service system that provides reliable and high-performance file transfer through the wide area network. Due to the lack of access authorization, we are not able to transfer data to many geo-distributed storage systems and measure the real-time network bandwidth in our evaluation. In order to address this issue, we collaborate with the Globus team who shared with us the anonymized transfer logs collected by Globus transfer service from 2014 to 2018 [40]. Among these logs, we focus on the data transfers between the Globus Connect Servers (GCS), which are GridFTP servers deployed on high-performance storage systems and can handle many concurrent data transfer requests. We first select one GCS which had a large number of transfer records with other GCSs. Then we extract logs of transferring data between the selected GCS and 16 other remote GCSs. We can calculate the user-perceived throughput for each transfer using the information embedded in the logs (transfer times and data sizes). Finally, we calculate the average throughput of all these transfers and use it as an estimate of the bandwidth between the selected GCS and each of those 16 remote GCSs. Based on our calculation, the estimated network bandwidth to/from different remote GCSs can vary from 400MB/s to more than 3GB/s. The estimated network bandwidth numbers obtained from the historical Globus logs are only used to evaluate the time it takes to transfer data and parity fragments to/from remote storage systems (Fig. 3 and Fig. 4). All the other evaluation results are obtained by running our codes on Andes cluster.

5.1.3 Scientific Datasets. The datasets used in our evaluation come from three different scientific domains. Particularly, these datasets are generated by Hurricane Isabel climate simulation [23], NYX cosmology simulation [45], and SCALE-LETKF weather simulation [52]. Each of these datasets contains multiple data objects and each data object is a 3D array that represents certain field (e.g., temperature, velocity, etc.) in the 3D space targeted by the simulation. The elements in all these arrays are 32-bit floats. In a weak scaling setup, the size of each data object produced by the simulation running on each CPU core is fixed. Therefore, multiplying the size of data objects on each CPU core by the number of CPU cores used for the simulation gives us the total size of the simulation data. In all our evaluations, we assume the complete datasets are all produced by simulations running with 32,768 CPU cores (1,024 nodes, 32 cores/node) on OLCF's Summit supercomputer. For example, in the "NYX" dataset the size of each data object produced on each CPU core is 512MB. Thus the total size of each "NYX" data object used in our evaluation is $512 \times 32768/1024/1024 = 16$TB. Similarly, we can calculate the total size of each data object in the other two datasets, which are 16.82TB (SCALE-LETKF) and 2.98TB (Hurricane Isabel). The details of the datasets we used are listed in Table 2.

Table 2: Scientific datasets for evaluation

Dataset	Object name	Size/object
NYX	temperature	16TB
NYX	velocity_x	16TB
SCALE-LETKF	PRES	16.82TB
SCALE-LETKF	T	16.82TB
Hurricane Isabel	Pf48.bin	2.98TB
Hurricane Isabel	TCf48.bin	2.98TB

5.1.4 *Availability of Storage Systems.*

The probability of losing the access to one storage system, p, is an important parameter in our model for optimizing the fault tolerance configurations. If p is large, more redundancies must be added to the data to achieve the same level of availability compared to when p is small. Therefore, we need to assign a realistic value to p in our evaluation. According to the operational assessment report from OLCF for year 2020 [2], the Alpine storage system was available for 98.93% of the time, which means the probability that Alpine became unavailable in 2020 is 0.0107. In the operational assessment report from the Argonne Leadership Computing Facility (ALCF) for year 2020 [7], the Theta Lustre storage system was available for 94.8% of the time, meaning the system outage probability is 0.052. Theta Lustre storage system has higher outage probability than Alpine might be because of its longer service time (it has been in operation since 2017). In our all evaluations, we set $p = 0.01$ to be consistent with the OLCF's operational assessment report.

5.2 Comparison with Existing Methods

5.2.1 *Data Quality vs. Storage Overhead.*

Different from using the two existing methods, where the data is either available or unavailable, our approach allows the data to be restored with certain error given the availability of all the levels in that hierarchy. Since now the restored data might contain errors, the data quality depends not only on its availability but also the accuracy. Thus, we use the "the expected relative L-infinity error" (defined in Section 3.2), which combines both the availability and errors of the data, as a metrics, to quantify the data quality that can be achieved by different methods.

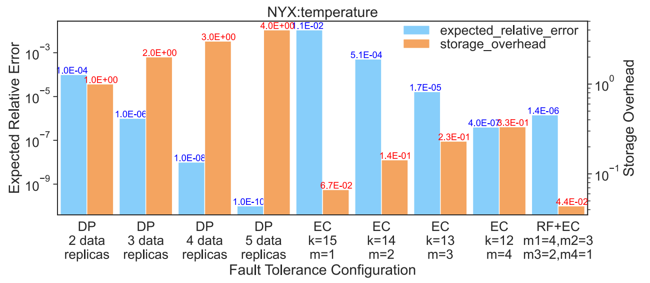


Figure 2: For our approach in this case, the relative L-infinity errors of using the first j levels of the hierarchy to reconstruct the data, e_j, are set as: $e_1 = 0.004$, $e_2 = 0.0005$, $e_3 = 0.00006$, $e_4 = 0.0000001$.

The expected relative L-infinity error of the restored data when our approach is used can be calculated through Equation 5 in Section 3.2. In Equation 5, we assume the the relative L-infinity error achieves the "upper bound", which is 1.0, if the entire data is unavailable (see explanation in Section 3.2). Similarly, we also calculate this metrics for the two existing methods. With Equation 1, we can easily get the expected relative L-infinity error for the data duplication method: $E_{DP} = 1.0 \times P_{\text{UNAVBL}} + 0.0 \times (1 - P_{\text{UNAVBL}}) = \sum_{i=0}^{n-m} \binom{n-m}{i} p^{m+i} (1-p)^{n-m-i}$. Using Equation 2, we can obtain the expected relative L-infinity error when the regular erasure coding method is used: $E_{EC} = 1.0 \times P_{\text{UNAVBL}} + 0.0 \times (1 - P_{\text{UNAVBL}}) = \sum_{i=m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$.

As shown in Fig. 2, we apply two existing methods and our approach ("DP" stands for data duplication, "EC" stands for regular erasure coding, "RF+EC" stands for our approach as it combines data refactoring and erasure coding) to the "temperature" object in the "NYX" dataset. For our approach in the first case demonstrated by Fig. 2, the original data is decomposed and refactored into a hierarchy with four levels, where the relative L-infinity errors of using the first j levels of the hierarchy to reconstruct the data, e_j, are set to be: $e_1 = 0.004$, $e_2 = 0.0005$, $e_3 = 0.00006$, $e_4 = 0.0000001$. We then create 4, 3, 2, and 1 parity fragments for each level from the top to the bottom of that hierarchy using erasure coding, which means the concurrent storage systems failures each level can tolerate are 4, 3, 2, and 1. As we can see, our approach achieves better expected relative error compared to data duplication with 2 data replicas and regular erasure coding with 3 parity fragments, with much lower storage overhead.

5.2.2 *Network Overhead.*

Not only does storing extra data copies and parity fragments incur storage overhead, but transferring them to those remote storage systems also requires more time due to the extra network bandwidth consumption. As shown in Fig. 3, we evaluate the overall latency of distributing the data and parity fragments to 15 remote storage systems when different methods are applied to six different data objects from the three datasets. Since the data transfers to different remote storage systems can be launched in parallel, the overall transfer latency depends on when the slowest transfer is finished. For the regular erasure coding and our approach, each EC-fragment is transferred to one of the 15 remote storage systems. For data duplication method, the extra data copies are always transferred to the storage systems that have the highest network bandwidth. From Fig. 3, we can see that it takes

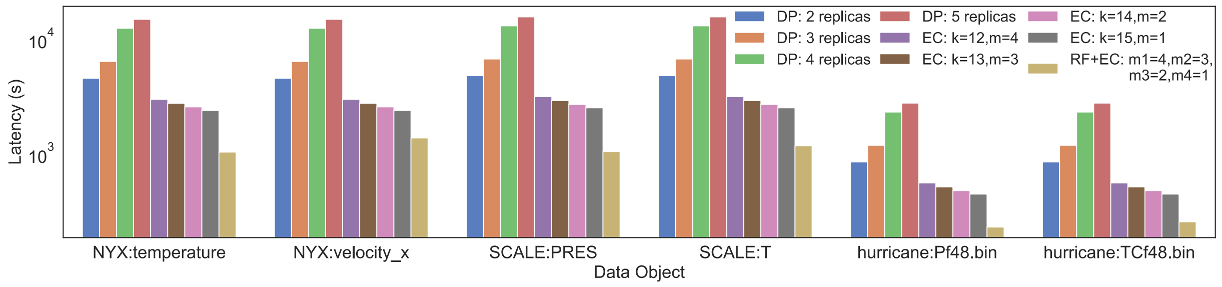


Figure 3: Latency of distributing data and parity fragments to remote storage systems

huge amount of time to complete the transfer when the data duplication method is used even only one extra data copy is created (2 data replicas in total). Adopting the regular erasure coding reduces the transfer time since the much smaller data and parity fragments instead of the original data are transferred to each remote system. Similar to the storage overhead, our approach significantly outperforms the two existing methods in terms of transfer latency due to the data reduction results from the multigrid-based error-bounded lossy compression and heterogeneous fault tolerance configurations for the refactored data.

5.3 Fault Tolerance Configuration Optimization

In Fig. 2 and Fig. 3, when our approach is used, the number of parity fragments created for each level of the refactored data using erasure coding are 4, 3, 2, and 1. On the one hand this configuration causes the least amount of storage overhead since the number of parity fragments each level has is minimized. But on the other hand, it also offers the worst expected relative error. In practice, users might be willing to sacrifice certain amount storage space to minimize the expected relative error. Moreover, the data might need to be decomposed and refactored into a hierarchy with more than four levels, or the user wants to distribute the data and parity fragments to more than 16 remote systems. All these requirements can be addressed by solving the optimization model we proposed in Section 3.2.

We build the optimization models using 6 different data objects, then use the heuristic algorithm proposed in Section 3.2 to solve the optimization problems to determine the optimal fault tolerance configurations for each data object. As shown in Table 3, not only can our heuristic algorithm find the same optimal fault tolerance configurations, but it is also more than 100 times faster than the brute-force approach when applied to all these data objects.

Table 3: Effectiveness of our heuristic algorithm

Data object	Optimal FT Configuration		Speedup (t_{BF}/t_H)
	Brute-Force	Heuristic	
NYX:temperature	[8,5,4,2]	[8,5,4,2]	124
NYX:velocity_x	[5,4,3,1]	[5,4,3,1]	152
SCALE:PRES	[9,6,4,2]	[9,6,4,2]	114
SCALE:T	[7,4,3,2]	[7,4,3,2]	137
hurricane:Pf48.bin	[11,10,8,7]	[11,10,8,7]	152
hurricane:TCf48.bin	[11,9,8,6]	[11,9,8,6]	137

5.4 Data Gathering Strategy Optimization

In our evaluation, the model for optimizing the data gathering strategy has $4 \times 16 = 64$ binary variables. Due to the large solution space of this problem, there is no guarantee that the MINLP solver can find the global optimal solution quickly. Therefore, we set the maximum amount of time for running the solver to be 60 seconds in all our evaluations. The solution given by the solver at the end of the 60-second run is used as our "Optimized" strategy. Once the fragments that need to be transferred are selected, we calculate the time it takes to transfer every of them and use the slowest time plus the 60-second optimization time as the overall latency.

To demonstrate the superiority of the "Optimized" data gathering strategy found by our model, we also evaluate two other strategies, which are the "Random" and "Naive" strategy. When the "Random" strategy is adopted, the fragments to be transferred are randomly selected. A combination of randomly selected fragments that can fully restore all the levels of the refactored data is called a selection. To understand the variability brought by the randomness, we generate 50 different selections by varying the random seed in our code. For each of these selection, we calculate the time it takes to complete all the transfers as the overall transfer latency. In Section 3.3, we use the error bars to represent the standard deviation of the transfer latency of these 50 selections.

When the "Naive" strategy is adopted, we first sort the network bandwidth of all 16 remote storage systems in descending order to form a list. Then we select the fragments always starting from the storage system that has the highest network bandwidth to restore each level of the refactored data. For instance, let us assume 8 fragments are needed to restore the first level, all these fragments will be transferred from the first 8 remote storage systems on that list. Similarly, if 6 fragments are needed for the second level, they will be transferred from the first 6 remote storage systems on that list. As we can see, this is a greedy strategy which intends to take advantage of the remote storage systems that have higher network bandwidth. However, transferring multiple fragments from the same storage system causes bandwidth contention, leading to performance degradation. To accelerate the searching for the "Optimized" data gathering strategy, we also use the "Naive" strategy as the initial value for the MINLP solver.

From Fig. 4, we can see that for most of the data objects, our optimized strategy reduces the data gathering latency by up to 2x compared to the "Random" strategy and 1.5x compared to the "Naive" strategy. Such performance improvements are particularly valuable when the data objects are large. For example, when gathering the fragments of data object "SCALE:T", using our optimized

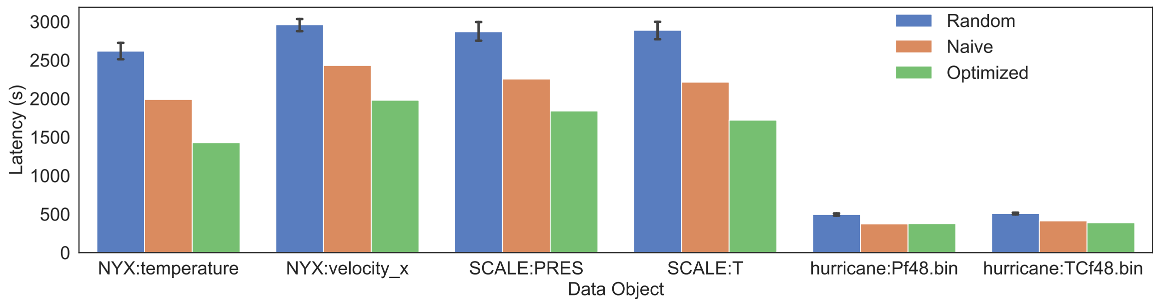


Figure 4: Latency of gathering data and parity fragments from remote storage systems with different strategies

strategy can save more than 1,000 seconds. However, for small data objects such as "hurricane:Pf48.bin", using the "Optimized" data gathering strategy does not give us significant performance improvement due to the 60-second optimization time.

5.5 Performance Acceleration

When different methods are used, different number of operations are involved and they are executed in certain orders. During the data preparation phase: 1) If the data duplication method is used, the original data is simply transferred to multiple remote storage systems, thus only the distribution operation is executed. 2) If the regular erasure coding method is adopted, the original data needs to be read into memory from the local storage system, erasure coded, written to the local storage system, and then distributed. 3) If our approach is used, the operations needed are: reading the data, refactoring the data, optimizing the fault tolerance configurations, writing the data and parity fragments to the local storage system, and distributing the fragments.

Similarly, during the data restoration phase: 1) If the data duplication method is used, the only operation needed is gathering one data replica from one remote storage system. 2) If the regular erasure coding method is adopted, the operations needed are: gathering the required fragments, reading the data into memory, and erasure decoding the data. 3) If our approach is used, the operations involved are: optimizing the data gathering strategy, gathering the required fragments, reading the data into memory, erasure decoding the data, and reconstructing the data.

Next we evaluate the performance of all these operations with a varying number of CPU cores. We also show that the performance of the particular operations needed by our approach, the data refactoring and reconstruction, can be improved by leveraging GPUs. The fault tolerance configurations we used for our approach in the following evaluations are the same as those optimal ones shown in Table 3.

5.5.1 Performance Acceleration with More CPU cores. In Fig. 5, we evaluate our approach with different number of CPU cores during the data preparation phase and measure the time it takes to complete each of those involved operations. As we can see, the overall execution time is dominated by the data refactoring operation if only a small number of CPU cores (128 cores or less) are used, especially when the original data object is large. Since the data refactoring operation can be executed in an embarrassingly parallel fashion, increasing the number of CPU cores does significantly reduce its execution time. Besides data refactoring, the performance of other operations, such as erasure coding, data read and write, are also improved with the increase of CPU cores.

Table 4: Overall data preparation performance

Data object	DP	64 cores		256 cores		1024 cores	
		EC	RF+EC	EC	RF+EC	EC	RF+EC
NYX:temperature	6412	5838	7936	3818	2850	3177	1469
NYX:velocity_x	6412	5338	8251	3693	3187	3146	1808
SCALE:PRES	6742	6144	8455	4015	2988	3341	1508
SCALE:T	6742	5911	8310	3958	3052	3326	1622
hurricane:Pf48.bin	1194	985	1537	686	575	585	314
hurricane:TCf48.bin	1194	1004	1699	690	635	587	348

In practice, computer clusters with many CPU cores are not always available for access. Therefore, we also evaluate the two existing approaches to find out which one achieves the best overall performance during the data preparation phase when different number of CPU cores can be used. In order to have a fair comparison, we choose the fault tolerance configurations for the data duplication and regular erasure coding method so that they can achieve comparable expected relative errors. In this case, the duplication method creates 3 data replicas (2 extra copies) while the regular erasure coding method generates 12 data fragments and 4 parity fragments.

We list the performance numbers (the time it takes to complete all the needed operations) for all the three methods in Table 4, and all the performance numbers are in seconds. As we can see, when 64 CPU cores (2 compute nodes of Andes cluster) are used, using the regular erasure coding method achieves the best overall performance. Due to the complexity of the multigrid-based error-bounded lossy compression algorithm, our approach takes much more time to complete in this case. Therefore, even our approach reduces the storage and network overhead (see Section 5.2), it is not recommended if limited computing resources are available. With the increase of CPU cores, our approach outperforms the other two methods. For example, our approach can achieve about 4x and 2x execution time reduction compared to the data duplication and regular erasure coding method.

Similarly, we also evaluate our approach with different number of CPU cores during the data restoration phase. As shown in Fig. 6, the data reconstruction operation takes a huge amount of time to complete compared to other operations if small number of CPU

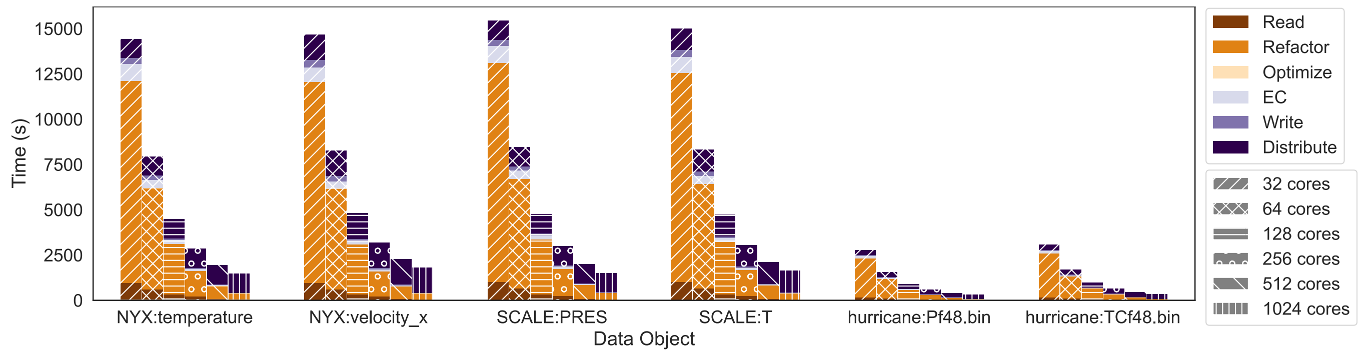


Figure 5: Performance of each operation during data preparation

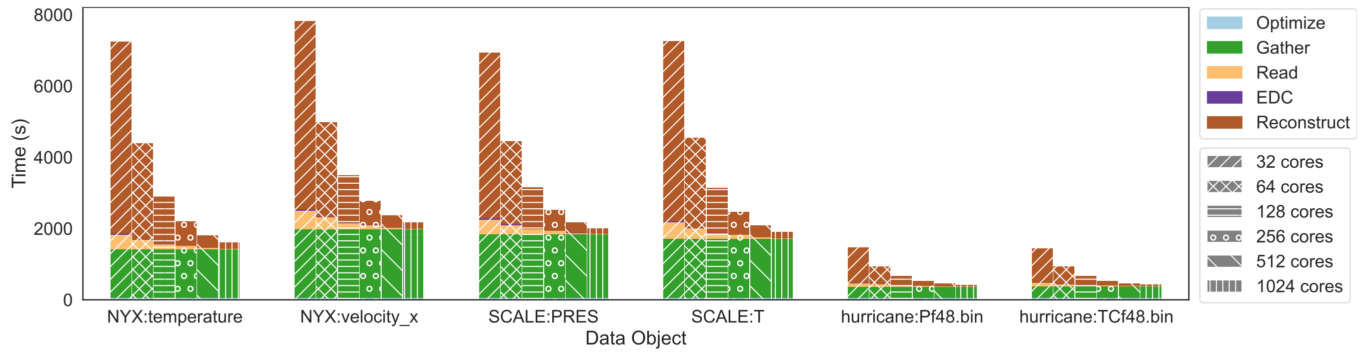


Figure 6: Performance of each operation during data restoration

cores are used. But just like the data refactoring operation, the data reconstruction operation can also be greatly accelerated by using more CPU cores.

Table 5: Overall data restoration performance

Data object	DP	64 cores		256 cores		1024 cores	
		EC	RF+EC	EC	RF+EC	EC	RF+EC
NYX:temperature	4583	3416	4391	2853	2200	2584	1612
NYX:velocity_x	4583	3407	4972	2850	2767	2583	2164
SCALE:PRES	4819	3598	4444	3001	2522	2717	2001
SCALE:T	4819	3588	4548	2998	2462	2716	1895
hurricane:Pf48.bin	854	634	936	531	524	481	412
hurricane:TCf48.bin	854	636	925	531	529	481	422

In Table 5, we list the overall performance for all the three methods during the data restoration phase. As we can see, the regular erasure coding method still achieves the best overall performance when 64 CPU cores are used. If we increase the number of CPU cores to 256, our approach start outperforming the regular erasure coding method. If we further increase the number of CPU cores to 1,024, the performance improvement can be more significant especially when data objects are large.

5.5.2 Performance Acceleration with GPUs. As shown in the above evaluation results, the data refactoring and reconstruction operations of our approach can be time-consuming if they cannot be executed on many CPU cores in parallel. An alternative way to accelerate these two operations is to leverage GPUs. As illustrated in Fig. 7, by running the data refactoring operation on a GPU, the data refactoring throughput can be improved by 3.7x on average. If we run the data reconstruction operation on a GPU, the throughput improvement will be 20.3x on average.

6 RELATED WORK

Resilience is a well-explored topic in both HPC and cloud computing. In the field of HPC, most of existing studies focus more on the system/application resilience rather than the data resilience, since the main portion of the data produced by traditional HPC applications is checkpoint data [8, 18, 44, 59], which has an ephemeral lifetime (It is only used for restarting the execution when failures happen and will be discarded if the execution finishes successfully). However, with advanced techniques such as machine learning and AI being used in science, more data is produced for analytics purposes by the scientific applications running on HPC systems [27]. Similarly, the data collected from experimental and observational instruments is often moved to and stored in the HPC storage systems for further analysis and study as well [60]. These types of scientific data are extremely valuable and must be available for scientists to repeatedly access in the long term.

In cloud computing, data resilience receives equal attention as the system resilience. The commonly used approaches include data duplication [11, 31, 58] and erasure coding [35, 39]. In general, erasure coding-based approaches are more favorable compared to

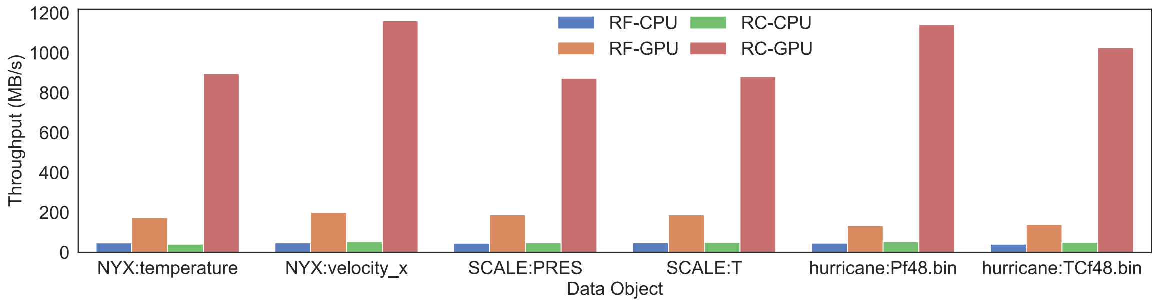


Figure 7: Data refactoring and reconstruction performance on a single CPU core vs. on a GPU

data duplication, as they incur less storage overhead. For instance, large-scale distributed storage systems like Swift [3], Google file system [19], and Windows Azure storage [22], have implemented solutions based on erasure coding as alternatives to data duplication. Some distributed storage systems, such as HDFS [25] allow data to be stored in a multi-tiered fashion with data at different tiers being stored using different erasure codes or with different number of replicas [62]. However, applying data duplication or erasure coding-based approaches directly to the forementioned scientific data faces significant challenges. Since the volume of these scientific datasets are huge and they are often produced at high velocity, duplicating or erasure-coding the data leads to excessively large storage and network overhead.

A few methods have been proposed and developed to reduce such overheads. For instance, burst buffer and caching strategies have been deployed to decrease the high service latency due to remote accesses in Geo-distributed cloud storage systems [39]. A scalable and resilient in-memory data staging runtime for large-scale in-situ workflows called CoREC is designed and implemented [17], which provides data duplication and erasure coding capabilities similar to previous solutions but in the context of in-situ and in-transit scientific data processing. Moreover, in order to achieve a good trade-off between the user-perceived data availability and storage overhead, CoREC diversifies the number of data replicas or erasure coding schemes for each individual data object based on the data access frequency so that more redundancies are added for popular data objects.Another data resilience framework for distributed storage systems called Zebra [32] also categorizes data objects into multiple tiers based on their demand and applies erasure coding to each tier with different parameters. However, unlike CoREC, Zebra does not need users to provide predetermined erasure coding parameters as input. Instead, given the overall storage overhead and the number of failures to tolerate, it automatically determines the parameters of erasure coding in each tier by solving a geometric programming problem.

All the forementioned solutions either use data duplication or erasure coding in static ways which often result in significant storage and network overhead, or rely on predicting the data access patterns (such as CoREC and Zebra) which not only is difficult (the data access pattern can vary substantially over time) but also requires extra computing and storage resources to collect and maintain data access history during runtime. Moreover, none of them takes the information content of the data into consideration when determining the amount of redundancies that should be added. By contrast, the approach we proposed in this paper does not need to learn the data access patterns. It leverages data refactoring technique to decompose scientific data into a hierarchical representation and automatically determine the optimal amount of redundancies that should be applied to each level of that hierarchy in a systematic and quantitative way.

7 CONCLUSION

Improving the availability of scientific data is important as scientists rely on their data to make scientific breakthroughs. However, as scientific data gets large, using conventional methods such as data duplication or erasure coding has become infeasible in many cases since these methods can cause significant storage and network overhead. To address this challenge, in this study we propose RAPIDS, a hybrid approach that combines the multigrid-based error-bounded lossy compression with erasure coding and can greatly reduce the storage and network overhead required for maintaining high data availability. The evaluation results demonstrate that when our approach is adopted, the storage overhead can be reduced by up to 7.5x while the network overhead can be reduced by up to 3.5x compared to the regular erasure coding method. To make our approach more feasible in practice, we further improve it by optimizing its fault tolerance configurations and data gathering strategy, and accelerating the data refactoring/reconstruction operation by leveraging many CPU cores and GPUs.

ACKNOWLEDGMENTS

This research was supported by the ECP CODAR, Sirius-2, and RAPIDS-2 projects through the Advanced Scientific Computing Research (ASCR) program of Department of Energy, and the LDRD project through the DRD program of Oak Ridge National Laboratory. This research used resources of the Oak Ridge Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

REFERENCES

[1] A Command Line Interface to Globus. 2022. https://github.com/globus/globus-cli. Online.

[2] Subil Abraham, J. Paul Abston, Ryan M. Adamson, Valentine Anantharaj, Aaron Barlow, Ashley D. Barker, Katie L. Bethea, Kevin S. Bivens, Adam Carlyle, Daniel Dietz, Christopher Fuson, Rachel Harken, Benjamin Hernandez, Jason J. Arreguin, Hill, Jason Kincl, Ryan Landfield, Don Maxwell, Veronica Melesse Vergara, Bronson Messer, J. Robert Michael, Ross Miller, Sheila Moore, Sarp Oral, Thomas Papatheodore, Ryan Prout, Sherry Ray, William Renaud, Mallikarjun Shankar, Woong Shin, Scott Simmerman, Kevin G. Thach, Aristeidis Tsaris, Georgia Tourassi, Coury Turczyn, Joseph Voss, Justin L. Whitt, and Junqi Yin. 2021.

US Department of Energy, Office of Science High Performance Computing Facility Operational Assessment 2020. Technical Report. Oak Ridge Leadership Computing Facility.

[3] Cheongjin Ahn, Mehdi Pirahandeh, and Deok-Hwan Kim. 2020. Dynamic Allocation of Replication and Erasure Codes for Enhancing Storage Efficiency in OpenStack Swift. In *2020 International Conference on Electronics, Information, and Communication (ICEIC).* 1–2. https://doi.org/10.1109/ICEIC49074.2020.9051133

[4] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2018. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science* 19, 5 (2018), 65–76.

[5] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2019. Multilevel techniques for compression and reduction of scientific data—The multivariate case. *SIAM Journal on Scientific Computing* 41, 2 (2019), A1278–A1303.

[6] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2019. Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities. *SIAM Journal on Scientific Computing* 41, 4 (2019), A2146–A2171.

[7] ALCF. 2021. *2020 Operational Assessment.* Technical Report. Argonne Leadership Computing Facility.

[8] Marc Perelló Bacardit, Leonardo Bautista-Gomez, and Osman Unsal. 2021. FPGA Checkpointing for Scientific Computing. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS).* 1–7. https://doi.org/10.1109/IOLTS52814.2021.9486693

[9] Allison H. Baker, Dorit M. Hammerling, Alex Pinard, and Haiying Xu. 2022. Lossy Data Compression and the Community Earth System Model. In *EGU General Assembly Conference Abstracts (EGU General Assembly Conference Abstracts).* Article EGU22-8762, EGU22-8762 pages. https://doi.org/10.5194/egusphere-egu22-8762

[10] Rafael Ballester-Ripoll, Peter Lindstrom, and Renato Pajarola. 2019. TTHRESH: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics* 26, 9 (2019), 2891–2903.

[11] Rabindra K. Barik, Sudhansu Shekhar Patra, Rasmita Patro, Sachi Nandan Mohanty, and Abdulsattar Abdullah Hamad. 2021. GeoBD2: Geospatial Big Data Deduplication Scheme in Fog Assisted Cloud Computing Environment. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom).* 35–41.

[12] Martin Burtscher and Paruj Ratanaworabhan. 2009. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Comput.* 58, 1 (jan 2009), 18–31. https://doi.org/10.1109/TC.2008.131

[13] Bzip2. 2022. http://www.bzip.org/. Online.

[14] Zhengzhang Chen, Seung Woo Son, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. 2014. NUMARCK: machine learning algorithm for resiliency and checkpointing. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 733–744.

[15] Sheng Di and Franck Cappello. 2016. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS).* 730–739. https://doi.org/10.1109/IPDPS.2016.11

[16] Jack Dongarra, Bernard Tourancheau, Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. 2019. Use Cases of Lossy Compression for Floating-Point Data in Scientific Data Sets. *Int. J. High Perform. Comput. Appl.* 33, 6 (nov 2019), 1201–1220. https://doi.org/10.1177/1094342019853336

[17] Shaohua Duan, Pradeep Subedi, Philip Davis, Keita Teranishi, Hemanth Kolla, Marc Gamell, and Manish Parashar. 2020. CoREC: Scalable and Resilient In-Memory Data Staging for In-Situ Workflows. *ACM Trans. Parallel Comput.* 7, 2, Article 12 (may 2020), 29 pages. https://doi.org/10.1145/3391448

[18] Shen Gao, Bingsheng He, and Jianliang Xu. 2015. Real-Time In-Memory Checkpointing for Future Hybrid Memory Systems. In *Proceedings of the 29th ACM on International Conference on Supercomputing* (Newport Beach, California, USA) *(ICS '15).* Association for Computing Machinery, 263–272.

[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP '03).* Association for Computing Machinery, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[20] A. Gok, S. Di, Y. Alexeev, D. Tao, V. Mironov, X. Liang, and F. Cappello. 2018. PaSTRI: Error-Bounded Lossy Compression for Two-Electron Integrals in Quantum Chemistry. In *2018 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE Computer Society, Los Alamitos, CA, USA, 1–11. https://doi.org/10.1109/CLUSTER.2018.00013

[21] Leonardo A. Bautista Gomez and Franck Cappello. 2013. Improving floating point compression through binary masks. In *2013 IEEE International Conference on Big Data.* 326–331. https://doi.org/10.1109/BigData.2013.6691591

[22] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure Coding in Windows Azure Storage. In *2012 USENIX Annual Technical Conference (USENIX ATC 12).* USENIX Association, Boston, MA, 15–26. https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang

[23] Hurricane ISABEL simulation data. 2004. http://vis.computer.org/vis2004contest/data.html. Online.

[24] ITER. 2022. https://www.iter.org/. Online.

[25] Anju Kaushik and Vinod Kumar Srivastava. 2021. An Approach To Secure Sensitive Attributes Stored On HDFS Using Blowfish. In *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N).* 1260–1264. https://doi.org/10.1109/ICAC3N53548.2021.9725545

[26] Sandeep Koranne. 2011. *Hierarchical Data Format 5 : HDF5.* Springer US, Boston, MA, 191–200. https://doi.org/10.1007/978-1-4419-7719-9_10

[27] S. Ku, R. Hager, C.S. Chang, J.M. Kwon, and S.E. Parker. 2016. A new hybrid-Lagrangian numerical scheme for gyrokinetic simulation of tokamak edge plasma. *J. Comput. Phys.* 315 (2016), 467–475. https://doi.org/10.1016/j.jcp.2016.03.062

[28] Sidharth Kumar, John Edwards, Peer-Timo Bremer, Aaron Knoll, Cameron Christensen, Venkatram Vishwanath, Philip Carns, John A Schmidt, and Valerio Pascucci. 2014. Efficient I/O and storage of adaptive-resolution data. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 413–423.

[29] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. 2013. ISABELA for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience* 25, 4 (2013), 524–540.

[30] Daniel Laney and Valerio Pascucci. 2004. Progressive compression of volumetric subdivision meshes. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.* IEEE, 680–687.

[31] Waraporn Leesakul, Paul Townend, and Jie Xu. 2014. Dynamic Data Deduplication in Cloud Storage. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering.* 320–325. https://doi.org/10.1109/SOSE.2014.46

[32] Jun Li and Baochun Li. 2021. Demand-Aware Erasure Coding for Distributed Storage Systems. *IEEE Transactions on Cloud Computing* 9, 2 (2021), 532–545. https://doi.org/10.1109/TCC.2018.2885306

[33] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data).* IEEE, 438–447.

[34] Xin Liang, Qian Gong, Jieyang Chen, Ben Whitney, Lipeng Wan, Qing Liu, David Pugmire, Rick Archibald, Norbert Podhorszki, and Scott Klasky. 2021. Error-controlled, progressive, and adaptable retrieval of scientific data with multilevel decomposition. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–13.

[35] Hsiao-Ying Lin and Wen-Guey Tzeng. 2012. A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding. *IEEE Transactions on Parallel and Distributed Systems* 23, 6 (2012), 995–1003. https://doi.org/10.1109/TPDS.2011.252

[36] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 2674–2683.

[37] Peter Lindstrom and Martin Isenburg. 2006. Fast and Efficient Compression of Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250. https://doi.org/10.1109/TVCG.2006.143

[38] Peter Lindstrom and Martin Isenburg. 2006. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics* 12, 5 (2006), 1245–1250.

[39] Kaiyang Liu, Jun Peng, Jingrong Wang, Zhiwu Huang, and Jianping Pan. 2022. Adaptive and Scalable Caching with Erasure Codes in Distributed Cloud-Edge Storage Systems. *IEEE Transactions on Cloud Computing* (2022), 1–1. https://doi.org/10.1109/TCC.2022.3168662

[40] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Nageswara S. V. Rao. 2018. Cross-Geography Scientific Data Transferring Trends and Behavior. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC '18).* Association for Computing Machinery, New York, NY, USA, 267–278. https://doi.org/10.1145/3208040.3208053

[41] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments* (Boston, MA, USA) *(CLADE '08).* Association for Computing Machinery, New York, NY, USA, 15–24. https://doi.org/10.1145/1383529.1383533

[42] MGARD MultiGrid Adaptive Reduction of Data. 2022. https://github.com/CODARcode/MGARD. Online.

[43] MIDACO Mixed Integer Distributed Ant Colony Optimization. 2022. http://www.midaco-solver.com/. Online.

[44] Bogdan Nicolae, Jiali Li, Justin M. Wozniak, George Bosilca, Matthieu Dorier, and Franck Cappello. 2020. DeepFreeze: Towards Scalable Asynchronous Checkpointing of Deep Learning Models. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID).* 172–181. https://doi.org/10.1109/CCGrid49817.2020.00-76

[45] NYX simulation. 2019. https://amrex-astro.github.io/Nyx. Online.

[46] P. Deutsch. 1996. https://www.rfc-editor.org/info/rfc1952. Online.

[47] Alberto Paoluzzi, Valerio Pascucci, and Giorgio Scorzelli. 2004. Progressive dimension-independent Boolean operations.. In *Symposium on Solid Modeling*

and Applications. Citeseer, 203–211.

[48] Valerio Pascucci and Randall J Frank. 2001. Global static indexing for real-time exploration of very large regular grids. In *SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*. IEEE, 45–45.

[49] Majid Rabbani. 2002. Book Review: JPEG2000: Image Compression Fundamentals, Standards and Practice.

[50] RocksDB. 2022. http://rocksdb.org/. Online.

[51] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of lossy compression for application-level checkpoint/restart. In *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 914–922.

[52] SCALE-LETKF weather model. 2019. https://github.com/gylien/scale-letkf. Online.

[53] Ariel Shamir and Valerio Pascucci. 2001. Temporal and spatial level of details for dynamic meshes. In *Proceedings of the ACM symposium on Virtual reality software and technology*. 77–84.

[54] D. Tao, S. Di, Z. Chen, and F. Cappello. 2017. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, Los Alamitos, CA, USA, 1129–1139. https://doi.org/10.1109/IPDPS.2017.115

[55] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2018. Improving Performance of Iterative Methods by Lossy Checkponting. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC '18)*. Association for Computing Machinery, New York, NY, USA, 52–65. https://doi.org/10.1145/3208040.3208050

[56] Robert Underwood, Chun Yoon, Ali Gok, Sheng Di, and Franck Cappello. 2022. ROIBIN-SZ: Fast and Science-Preserving Compression for Serial Crystallography. arXiv:2206.11297 [cs.DC]

[57] Unified Erasure Coding interface for common storage workloads. 2022. https://github.com/openstack/liberasurecode. Online.

[58] D. Veeraiah and J. Nageswara Rao. 2020. An Efficient Data Duplication System based on Hadoop Distributed File System. In *2020 International Conference on Inventive Computation Technologies (ICICT)*. 197–200. https://doi.org/10.1109/ICICT48043.2020.9112567

[59] Dirk Vogt, Cristiano Giuffrida, Herbert Bos, and Andrew S. Tanenbaum. 2014. Techniques for Efficient In-Memory Checkpointing. *SIGOPS Oper. Syst. Rev.* 48, 1 (may 2014), 21–25. https://doi.org/10.1145/2626401.2626406

[60] Ruonan Wang, Rodrigo Tobar, Markus Dolensky, Tao An, Andreas Wicenec, Chen Wu, Fred Dulwich, Norbert Podhorszki, Valentine Anantharaj, Eric Suchyta, Baoqiang Lao, and Scott Klasky. 2020. Processing Full-Scale Square Kilometre Array Data on the Summit Supercomputer. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 11–22.

[61] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-State Quantum Circuit Simulation by Using Data Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 80, 24 pages. https://doi.org/10.1145/3295500.3356155

[62] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. 2015. A Tale of Two Erasure Codes in HDFS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 213–226. https://www.usenix.org/conference/fast15/technical-sessions/presentation/xia