CoREC: Scalable and Resilient In-memory Data Staging for In-situ Workflows

SHAOHUA DUAN, PRADEEP SUBEDI, and PHILIP DAVIS, Rutgers Discovery Informatics Institute, Rutgers University, USA
KEITA TERANISHI and HEMANTH KOLLA, Sandia National Laboratory, USA
MARC GAMELL, Intel, USA
MANISH PARASHAR, Rutgers Discovery Informatics Institute, Rutgers University, USA

The dramatic increase in the scale of current and planned high-end HPC systems is leading new challenges, such as the growing costs of data movement and IO, and the reduced mean time between failures (MTBF) of system components. In-situ workflows, i.e., executing the entire application workflows on the HPC system, have emerged as an attractive approach to address data-related challenges by moving computations closer to the data, and staging-based frameworks have been effectively used to support in-situ workflows at scale. However, the resilience of these staging-based solutions has not been addressed, and they remain susceptible to expensive data failures. Furthermore, naive use of data resilience techniques such as n-way replication and erasure codes can impact latency and/or result in significant storage overheads. In this article, we present CoREC, a scalable and resilient in-memory data staging runtime for large-scale in-situ workflows. CoREC uses a novel hybrid approach that combines dynamic replication with erasure coding based on data access patterns. It also leverages multiple levels of replications and erasure coding to support diverse data resiliency requirements. Furthermore, the article presents optimizations for load balancing and conflict-avoiding encoding, and a low overhead, lazy data recovery scheme. We have implemented the CoREC runtime and have deployed with the DataSpaces staging service on leadership class computing machines and present an experimental evaluation in the article. The experiments demonstrate that CoREC can tolerate in-memory data failures while maintaining low latency and sustaining high overall storage efficiency at large scales.

CCS Concepts: • **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; *Real-time systems*; • **Information systems** → *Data management systems*;

This work was supported in part by the National Science Foundation (NSF) via grants number CCF 1725649, and by Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

Authors' addresses: S. Duan, P. Subedi, P. Davis, and M. Parashar, Rutgers Discovery Informatics Institute, Rutgers University, 100 Brett Road, Piscataway, NJ, 08854; emails: {shaohua.duan, pradeep.subedi, philip.e.davis, manish.parashar}@rutgers.edu; K. Teranishi and H. Kolla, Sandia National Laboratory, Livermore, CA, 94550; emails: {knteran, hnkolla}@sandia.gov; M. Gamell, Intel, Austin, TX, 78746; email: marcgamell@gmail.com.

2329-4949/2020/05-ART12 $15.00
https://doi.org/10.1145/3391448

Additional Key Words and Phrases: Data resilience, erasure codes, replication, in-situ workflows, data staging

ACM Reference format:
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

1 INTRODUCTION

Scientific workflows running on current and emerging extreme-scale systems are providing new opportunities for solving some of the most important problems in science and society, such as those being addressed by the US Exascale Computing Program (ECP) [37]. However, running such workflows at extreme scale presents significant challenges spanning all aspects of data management, including data analysis, movement, and storage.

For example, the S3D [11] extreme scale scientific workflow, which is a coupled, multi-scale, multi-physics turbulent combustion workflow, involves intricate data-processing that includes multiple analyses performed at different temporal frequencies on non-overlapping subsets of data. To address the data-related challenges associated with coupled scientific workflows such as S3D executing at extreme scales, in-situ approaches based on data staging have emerged and are being used by applications on current high-end computing systems [14, 33]. Figure 1 illustrates an in-situ scientific workflow where the application components are coupled via an in-memory data staging framework. In this workflow, the primary scientific simulation is the data producer and the data consumer(s) include secondary simulations, analytics services, and/or visualization applications coupled to the data producer.

However, scientific workflows running on current and emerging extreme-scale systems are also expected to experience higher failure rates for various reasons relating to both increasing scale of hardware and more complexity of software [9]. Even worse, data loss due to failures (e.g., process failures, node failures) in any module of such complex workflows will impact the execution of the entire workflow and can invalidate the final results. Consequently, it is critical to ensure the resilience of the end-to-end workflow. To address fault tolerance at extreme scales with the expected commensurate higher rates of failures and data loss [9], recent research has explored techniques for minimizing application vulnerability to failures. Various fault tolerance techniques such as checkpoint/restart, process replication [17], and erasure coding [42] have been widely studied and have been utilized to mitigate failures in individual software components of the scientific workflow.

Unfortunately, current fault tolerance techniques can not be directly used to implement resilient data staging services due the large amount of data that is exchanged between the different application components via the staging area. For example, using Checkpoint/Restart techniques for an MPI-based application workflow, recovery from a failure in one of the data staging processes/nodes would require all the tightly coupled application components of the workflow to rollback to maintain a consistent status of the workflow. This would result in a large amount of data transfers from remote storage nodes or local disk and require substantial coordination between the application components to reach a consistent state. This, in turn, can lead to significant performance degradation and increase recovery times. Furthermore, application components and data staging servers that have not experienced failures and/or data loss also have to be re-executed, leading to unnecessary data transfer and computation overheads. Similarly, the process replication approach presented in Reference [18], which can potentially provide resiliency for in-situ workflows and data staging services, requires twice the amount of computing and storage resources, which may not be feasible. A more traditional approach based on replication, where multiple copies of the data

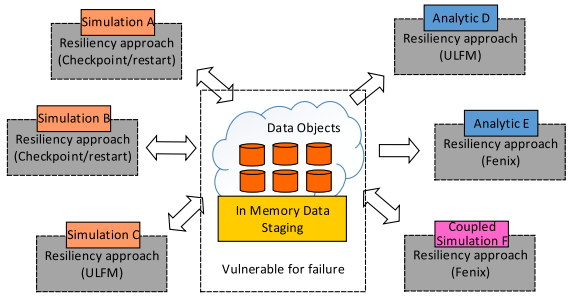


Fig. 1. A typical data staging workflow with fault tolerance.

are maintained, is a viable alternative but can have a large storage overhead [42]. For example, if you want a system to tolerate up to two node failures, using replication would result in a storage overhead of 200%. An alternate approach to achieving data reliability is using erasure coding. While erasure coding can dramatically reduce the storage cost, it incurs the overhead of encoding the data during writes and decoding the data during recovery from failures [34]. Resilience approaches based exclusively on replication or erasure coding would result in large storage overhead or computational overhead, respectively.

Furthermore, the components of an application workflow exhibit different failure probabilities during execution. For example, large-scale, long-running simulations tend to have a higher probability of concurrent failures than smaller-scale and shorter-running components. Since multiple distinct applications and workflows can share the same staging services, using a single approach to data resiliency can be either an overkill or ineffective for some applications/workflows. As a result, it is desirable that the staging runtime dynamically determines the level of data resiliency based on application needs.

An added challenge in implementing resiliency in a data staging framework is process recovery. While simply recovering the data that was lost due to process or node failures guarantees data resiliency, the original failure will still lead to a degradation of the performance of the workflow: When staging servers fail, it is feasible to recover the data and re-distribute the workload across remaining staging servers. While this enables the workflow to progress, the overall performance and the storage capacity of the staging area will be reduced due to the loss of the process/node. This increases the data query and exchange latency experienced by the workflow components. If we consider high failure rates and long-term execution times of the workflows, then the number of available staging servers will progressively reduce for time, leading to degraded performance and an increase in the total workflow execution time.

This article builds on our previous work [16] and makes contributions along multiple dimensions. First, this article explores how multiple replication and erasure coding schemes can provide different levels of data reliability with an understanding of acceptable costs and the associated trade-offs of achieved resilience, overheads, performance, storage, and so on. Second, this article presents a process recovery solution that cooperates with the data resiliency scheme and aims to recover failed staging servers so as to maintain the performance of the data staging framework over the lifetime of the workflow.

In this article, we present CoREC (Combining Replication and Erasure Coding), which is a hybrid approach to data resilience for staging-based in-situ workflows. CoREC provides the benefit of data replication, i.e., high performance, while leveraging erasure coding to reduce storage costs. CoREC uses online data classification, based on spatial/temporal locality, to determine whether to use erasure coding or replication, and balances storage efficiency with low computation overheads

while maintaining desired levels of fault tolerance. Moreover, to satisfy the diverse data resiliency requirements of different workflow components, CoREC supports the use of different data redundancy schemes, such as a hybrid approach combining both replication and triplication with different erasure codes. CoREC with multilevel data redundancy (CoREC-multilevel) can dynamically decide between erasure coding and replication schemes (for, e.g., duplicate, triplicate, and Reed-Solomon codes) based on the data access patterns while maintaining application-specified resiliency requirement and incurring minimal storage overhead. Furthermore, we develop optimized load balancing and conflict-avoiding encoding for CoREC, as well as a low-overhead lazy-recovery scheme for the staging nodes, to alleviate overheads and interference associated with CoREC for both data-writes and data-recovery. We have used CoREC to implement resilient data staging within DataSpaces and have also integrated ULFM (User Level Fault Migration) [7] to efficiently recover from both processes and node failures. We have deployed the resulting resilient DataSpaces on the Titan Cray XK7 production system at Oak Ridge National Laboratory (ORNL), the Cori Cray XC40 system at the National Energy Research Scientific Computing Center (NERSC), and the Caliburn system at Rutgers Discovery Informatics Institute (RDI2). Our experimental evaluations using synthetic workloads and the S3D combustion workflow demonstrate that CoREC maintains good storage efficiency and low latency for various use cases, supporting sustained performance and scalability even in the face of frequent node failures.

The rest of this article is organized as follows: Section 2 presents the low-latency and high-efficiency CoREC approach to data resilience for staging-based in-situ workflows, and Section 3 describes the design of CoREC. In Section 4, we present the implementation and evaluation of CoREC. Section 5 presents related work and Section 6 concludes the article.

2 COREC (COMBINING REPLICATION AND ERASURE CODING)

In this section, we first explore resilience requirements of staging-based in-situ workflows and investigate why traditional mechanisms, such as Checkpoint/Restart, are unable to effectively meet these requirements. We then introduce, model, and analyze CoREC/CoREC-multilevel, our hybrid approach to in-staging data resilience, and present an online approach for data classification based on data access patterns, which underlies CoREC.

2.1 Data Resilience for Staging-based In-situ Workflows

Data staging techniques leverage resources on the HPC system (i.e., cores and storage on simulation nodes as well as on dedicated nodes) to store and process data as it flows (typically memory-to-memory using Remote Direct Memory Access (RDMA)) between components of an in-situ workflow. For example, DataSpaces [13, 14], a data staging service targeting extreme-scale application workflows, uses data staging cores to implement a semantically specialized, virtual shared-space abstraction that can be associatively accessed by all applications and services and provides underlying runtime and RDMA-based asynchronous data transport mechanisms to support in-situ/in-transit workflows. It enables *live data* to be extracted from running applications, indexes this data online, and then allows it to be monitored, queried, and accessed by other applications and services via the shared-space using semantically meaningful operators.

While there is an increasing body of work on scalable fault-tolerance mechanisms applicable to individual applications [19], these mechanisms are not directly applicable to in-situ workflows, the data staging service supporting the workflow, or the data being staged by the workflow. In the Checkpoint/Restart approach, checkpoint data are periodically saved during application execution, and when a failure occurs, the application uses these checkpoints to rollback to the most recent consistent state. Using Checkpoint/Restart for fault tolerance of the data staging service presents

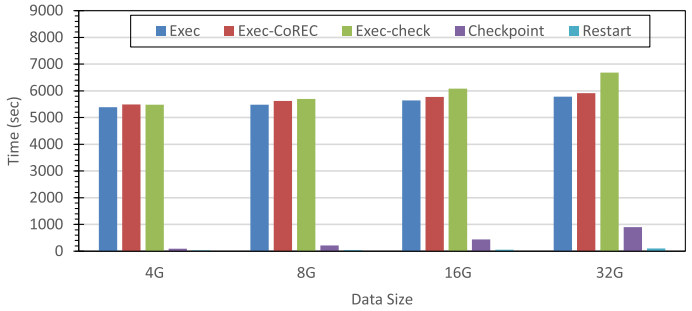


Fig. 2. Impact of checkpointing on staging-based in-situ application workflows. *Exec* is the total execution time of the workflow without checkpointing; *Exec-CoREC* is the total execution time of the workflow using CoREC; *Exec-check* is the total execution time of the workflow with periodic checkpointing of the staged data; *Checkpoint* is the total time required to checkpoint the data staging servers; *Restart* is the time required to perform a global restart of the data staging servers using a checkpoint.

two concerns: The first is the impact on the runtime of application workflows that use the data staging service.

To illustrate this impact, we performed periodic checkpointing of the data stored at the DataSpaces servers to the parallel file system on Titan and measured the total execution time with no server failure. Checkpointing was performed every five minutes, which is similar to the frequency used in the experiments presented in Reference [19], for a total of eight staging servers with varying staged data sizes. This resulted in a range from 17 checkpoints for a data size of 4 G to 20 checkpoints for a data size of 32 G. The results are plotted in Figure 2. From the plots, we can see that even if no failures are present, checkpointing significantly increases the total execution time of the workflow as the staged data size increases. In this case, the maximum time spent to achieve fault tolerance for just the staging servers is ~15.6% of the workflow run time without failures. In addition, this does not include the work lost from rolling back to a previous state. As presented in this article, failure recovery using CoREC increases the total execution time of the workflow by up to 2.23%, which is significantly lower than using Checkpoint/Restart. Furthermore, there is no loss of work in the case of CoREC. The second concern is the overhead due to large amounts of data movement and potential cascading rollback. When using Checkpoint/Restart for MPI applications, rolling back the data staging server can cause the tightly coupled application components of the workflow to become out of sync. Since all of the tightly coupled components in the MPI communication group must be rolled back to an overall consistent state, this can trigger a cascading rollback of the workflow where the rollback of one component triggers other healthy component(s) to rollback, and, in the worst case, cause the entire workflow to restart from the beginning. This process can result in significant coordination and data movement overheads. Process replication or process-redundancy [18] is another mechanism often used for fault tolerance. This approach consists of replicating all processes and their computations. Using replication for fault tolerance of the data staging service would require each staging server and its data to be replicated, which doubles the compute and storage requirements and can make it infeasible.

Since ensuring access to the staged data in spite of failures is most critical for a staging service, data resilience techniques such as data replication or erasure coding are more appropriate. Data replication involves making multiple copies of the data object and distributing them across multiple nodes, which enables efficient recovery by re-routing requests to a replica in case of a failure. However, it can result in increased storage requirements, which may not be feasible for in-memory staging due to limited memory size and increasing data volumes.

An alternate approach to data resilience with lower storage overheads is to employ erasure coding techniques. Erasure codes are constructed using two configurable parameters n and k (where $k < n$). The data are treated as a collection of fixed size units called blocks/objects. Every k original objects (called data objects) are encoded into $n - k$ additional equal size coded objects (called parities) and the set of the n data and parity objects is called a stripe. In case of a data staging service, objects of independently encoded multiple stripes are stored on distinct staging servers, allowing the service to tolerate $n - k$ server failures.

While erasure coding provides lower storage overheads as compared to the replication, it can lead to significant computation and network overheads, as parity has to be re-computed for every object update. If a data object in a stripe is updated, then erasure coding must update the associated parity. This process involves reading old data objects in the stripe, re-computing parities, and updating them. For example, if a stripe has six data objects and two parity objects, updating one data object requires five data object reads (for old data), re-computing two parity objects and two parity object writes. As a result, using erasure coding can be suboptimal for frequently written/updated data objects.

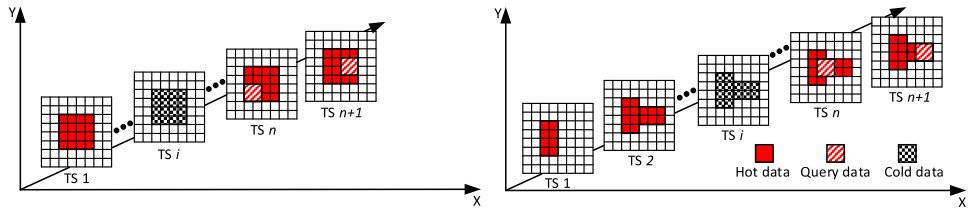
2.2 CoREC, a Hybrid Approach

CoREC is a hybrid approach that dynamically (and intelligently) combines replication with erasure coding based on data access patterns to balance storage efficiency with computation overheads while maintaining desired levels of fault tolerance. Specifically, CoREC uses a robust classification of data access patterns to identify *hot* and *cold* data—the key idea is to replicate the write-hot data while applying erasure coding for write-cold data. Using replication for write-hot data eliminates the expensive parity updates, as we only need to update the replicas. Using erasure coding for write-cold data ensures limited object updates and dramatically reduces storage costs as compared to using a pure replication-based approach. For example, in a two-failure resiliency case, let us assume that 60% of the data are identified as write-cold, which uses erasure code ($n = 8, k = 6$), and the remaining 40% hot data objects are replicated for fault-tolerance. Here, using CoREC, we incur only 100% storage overhead compared to the 200% needed for full replication, but maintain write performance close to that of replication, assuming write-cold data are rarely updated. Note that we do not consider read access patterns in our hot/cold classification, because data encoded with systematic erasure codes do not need to be decoded for reads in the absence of failures [34].

2.3 Classifying Data Access

CoREC utilizes the concept of write-hot and write-cold data to identify data objects as candidates for either replication or erasure coding. If a data object has been recently written/updated more than a threshold number of times within a certain interval, then it is considered to be hot data; otherwise, it is considered to be cold data. While data access patterns in real applications can change as the application evolves, i.e., a hot data object may become cold and vice versa, access patterns in scientific applications typically exhibit high temporal and spatial data localities, as the data and its access is typically defined along some discretization of a physical domain (e.g., a mesh or a grid), and the accesses are iterative in time [4].

During the execution of scientific simulation workflows, the simulation (e.g., S3D) issues a data write request, which writes n-dimensional data, at the end of each time step/iteration. Here, we use *temporal locality of objects* to indicate data objects being written/updated in consecutive time step and *spatial locality of objects* to refer to data objects that are near to each other in the n-dimensional space. As an illustrative example, consider a simulation that uses a 2-dimension Cartesian grid as show in Figure 3(a). The simulation writes data objects in region {(2, 2), (6, 6)} of the grid at time step 1, and this hot data turns cold at time step i (*temporal locality*). At time step n, another



(a) Single time step data locality case (b) Multi time steps data locality case

Fig. 3. An illustration of spatial and temporal data write/update patterns for a 2D data domain with $N + 1$ time steps. The solid red regions and slash regions (i.e., *hot* data) indicate data written into the staging area, while the black dot regions (i.e., *cold* data) are not updated since time step i.

application writes/updates only a portion of that region (say, region {(2,2), (3,3)}). In this case, it is very likely that the surrounding data objects in region {(2, 2), (6, 6)} (due to *spatial locality*) will also be written/updated at subsequent time steps, $n + 1$, $n + 2$, and $n + 3$ [4].

We may go beyond this one step look-ahead prediction and consider several time steps. For example, suppose that the highlighted data objects at time step 1 and step 2 are written by one application in Figure 3(b), and these multiple objects turn cold at time step i. If at time step n another application writes a portion of the combined regions of {(2, 2), (4, 6)}, and {(4, 4), (7, 5)}, then it will likely access objects in the combined region during time steps $n + 1$, $n + 2$, and $n + 3$. This multi-time-step look-ahead mechanism is beneficial, because an application may have several different hot data objects at the same time step in different regions of the grid. CoREC uses these spatial-temporal data locality attributes for multi-time-step data access prediction.

While choosing candidates for replication and erasure coding, we need to consider the properties of both replication and erasure coding as described in Section 2.1. Since replication has advantages in terms of write performance for frequent writes but has storage overhead as compared to erasure coding, we use data access patterns to classify write-hot and write-cold data and apply replication and erasure coding techniques, respectively. Specifically, newly written or updated data objects are classified as hot data. Data objects with spatial coordinates near current hot data are anticipated to be accessed in near future, and thus are also considered hot. The data objects with temporal locality in previous iterations/time steps relative to the current hot data objects are also classified as hot data objects. CoREC replicates these hot data objects while all other cold data objects are erasure coded. We use reference counters to record the access frequency of each data object. From a pool of replicated data objects, the object with the lowest access frequency is selected as a candidate for erasure coding. Once it is erasure coded, its access frequency is reset back to zero and incremented with every future access. The objects in the erasure coding pool with highest access frequencies are selected to be transitioned to replication if and only if the current storage overhead is lower than a user-specified threshold, i.e., CoREC aims to maintain storage efficiency while providing highest performance.

2.4 Modeling the CoREC Approach

In this section, we analyze the trade-off between replication and erasure coding and the impact of data access classification on a simple hybrid approach.

If N_{level} is the data resilience level, i.e., the maximum number of simultaneous node failures that system should be able to recover from, then using replication for fault tolerance requires N_{level} copies of each object. Therefore, the storage efficiency, which is the ratio of the size of original data objects to the size of original data object plus redundant data objects, for replication is:

$$E_r = \frac{1}{N_{level} + 1}.$$

Assume that data are transferred between servers using a streaming approach and it takes c seconds to transfer one object from the current server to the remote server. Further assuming that these servers have an l second latency before sending the object to other servers to make copies, the time required to transfer N_{level} replica objects to guarantee data resiliency for one object is:

$$C_r = l \times N_{level} + c.$$

Using Reed Solomon Code [30], supporting N_{level} fault tolerance with a group of N_{node} servers involves both encoding and data transfer between servers. It requires a computation overhead of $O(N_{level} \times N_{node})$ and data transfer of $N_{level} + N_{node} - 1$ data objects for N_{node} objects. Thus, the storage efficiency is:

$$E_e = \frac{N_{node}}{N_{level} + N_{node}}.$$

and the time required to encode one data object is:

$$C_e = O(N_{level} \times N_{node}) + \frac{l \times (N_{level} + N_{node})}{N_{node}} + c.$$

2.4.1 Simple Hybrid Erasure Coding. In this article, we use simple hybrid erasure coding to refer to a hybrid approach where candidate data objects for replication and erasure coding are selected randomly without any data classification. Suppose that an application stages n disjoint objects and runs for a duration T while uniformly updating each object t times. Then, the resulting object update frequency is $f = \frac{T}{t}$. If the probability that an object will be replicated is P_r and the probability that an object will be erasure coded is $P_e = 1 - P_r$, then the storage efficiency for simple hybrid erasure coding (E_{hybrid}) can be computed as:

$$\frac{N_{node}}{(N_{node} \times (N_{level} + 1) \times P_r + (N_{level} + N_{node}) \times P_e)}.$$

The corresponding time complexity is given by:

$$C_{hybrid} = (P_r \times C_r + P_e \times C_e) \times f \times n. \tag{1}$$

2.4.2 CoREC. In CoREC, we classify data objects as hot or cold based on the data update frequency f. Assuming that the object update frequency is non-uniform for hot and cold data, let these frequencies be f_h and f_c, respectively, and that $f_h > f_c$. For n disjoint data objects, $P_h \times n$ hot data objects are replicated and $P_c \times n$ cold data objects are encoded in CoREC, where P_h and P_c are the percentages of hot and cold data objects in the data staging service, respectively. Therefore, the time complexity for CoREC can be computed as:

$$C_{CoREC} = P_h \times C_r \times f_h \times n + P_c \times C_e \times f_c \times n. \tag{2}$$

Since each data object in the data staging service is classified as either hot or cold, $P_c = 1 - P_h$. From Equation (1), we have:

$$C_{CoREC} = (C_r \times f_h - C_e \times f_c) \times n \times P_h + C_e \times f_c \times n. \tag{3}$$

Accordingly, the time complexity for exclusively using erasure coding $C_{erasure}$ and replication $C_{replica}$ are:

$$C_{replica} = (f_h - f_c) \times C_r \times n \times P_h + C_r \times f_c \times n, \tag{4}$$

$$C_{erasure} = (f_h - f_c) \times C_e \times n \times P_h + C_e \times f_c \times n. \tag{5}$$

The advantage of CoREC as compared to simple hybrid erasure coding in terms of time complexity can be computed as:

$$Gain = C_{hybrid} - C_{CoREC} = (C_e - C_r) \times P_h \times P_c \times (f_h - f_c) \times n. \tag{6}$$

The storage efficiency for CoREC, which depends on percentage of hot and cold data (E_{CoREC}), is given by:

$$\frac{N_{node}}{(N_{node} \times (N_{level} + 1) \times P_r + (N_{level} + N_{node}) \times P_e)}. \tag{7}$$

The prediction and classification of hot data objects depend upon the accuracy of the classifier. If the classifier is not accurate, then it might classify cold data as hot data (or vice versa). Even if the accuracy of the classifier is perfect, replicating all hot data objects might be infeasible due to limited memory size. Since we can tolerate a limited storage overhead for data resiliency, in CoREC, we introduce two parameters: *miss ratio* r_m and *storage efficiency constraint* S. We use miss ratio, i.e., the ratio of misclassified data objects to total hot data objects, as a measure of the accuracy of data access classification. Then, $P_h r_m n$ real hot data are classified as cold data and encoded. Thus, the time complexity for CoREC under miss ratio r_m can be computed as:

$$\begin{aligned} &C_{CoREC} = P_h(1 - r_m)C_r f_h n + P_h r_m C_e f_h n + P_c C_e f_c n = \\ &(C_r f_h - C_e f_c + (C_e - C_r) f_n r_m) n P_h + C_e f_c n \end{aligned} \tag{8}$$

The storage efficiency constraint S is used as an upper bound for the storage overhead that can be tolerated, which is a lower-bound for E_{hybrid} and E_{CoREC}. When $E_{CoREC} = S$, the storage efficiency constraint limit is reached and Equation (7) can be solved to obtain value of P_r as:

$$P_r = \frac{E_r \times (S - E_e)}{S \times (E_r - E_e)}.$$

When $P_r < P_h$ and $P_e > P_c$, $(P_h - (1 - r_m)P_r)n$, real hot data are encoded under constraint S. Thus, when CoREC hits the storage efficiency constraint, the time complexity for CoREC with miss ratio r_m can be computed as:

$$\begin{aligned} &C_{CoREC} = P_r(1 - r_m)C_r f_h n + (P_h - (1 - r_m)P_r)C_e f_h n + P_c C_e f_c n = \\ &(f_h - f_c)C_e n P_h + C_e f_c n - (C_e - C_r)(1 - r_m)P_r f_h n. \end{aligned} \tag{9}$$

Using the time complexity equations (1), (3), (4), (5), (8), and (9), we plot relative write/update cost versus the hot data percentage in Figure 4. When all of the data objects are cold (Marker 1 in the figure), the write performance of CoREC is the same as simple hybrid erasure coding, because data are written/updated rarely. With the increase in the hot data percentage, the time complexity for CoREC increases linearly, i.e., performance is gained due to the replication of hot data objects. If we assume that classification is accurate and there is no constraint on storage, then all hot objects are replicated and all cold objects erasure coded. In this case, the write cost will be similar to replication. When storage constraint limit S is reached (Marker 2 in the figure), some of the hot data objects will be erasure coded, irrespective of their classification, which will lead to an increase in the cost. In addition to this, if the classifier is not accurate, then there will be misclassifications, and write/update performance will be further degraded. In conclusion, between points 1 and 2 in Figure 4, the performance of CoREC increases due to the increase in hot data objects, but beyond

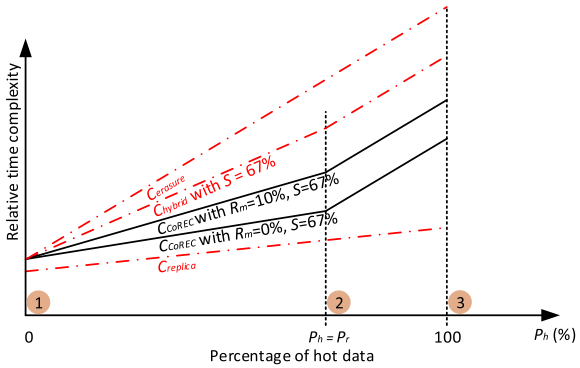


Fig. 4. An analytic study of the relative time complexity of CoREC (C_{CoREC}) with $RS(4,3)$ and varying miss ratios(R_m) and percentages of hot data objects (P_h). The time complexity for erasure coding ($C_{erasure}$), replication ($C_{replica}$), and simple hybrid erasure coding (C_{hybrid}) is noted by red dotted lines, as baselines.

point 2, the storage overhead limit is reached and objects are erasure coded irrespective of their classification, leading to a constant difference in time complexity with the full erasure coding approach, i.e., $C_{erasure}$.

Based on Equation (6) and Figure 4, we can deduce that CoREC's time complexity depends on the following factors: (i) The difference in the data access frequencies of hot and cold data objects, i.e., $f_h - f_c$. The larger the difference, the greater the benefit of CoREC. (ii) The difference in the time complexity of replication and erasure coding, i.e., $C_e - C_r$. The larger the difference, the greater the benefit of CoREC. (iii) The scale of workload n. The larger the workload, the greater the benefit of CoREC. (iv) The miss ratio, i.e., r_m. The lower the miss ratio, the greater the benefit of CoREC.

2.5 CoREC-multilevel, CoREC with Multilevel Data Redundancy

When dealing with in-situ workflows, each application and variable potentially has different data resilience requirements. For example, large-scale, long-term simulation applications require high data resiliency due to a higher probability of failures. Meanwhile, applications in a small-scale workflow may have lower data resilience requirement. Since different workflows can share the same data staging resource, using the same data resiliency scheme across the whole data staging framework is not efficient. Amplifying this concern, the level of data resilience might vary for each *variable* of an application. For example, in machine learning workflows performing hyperparameter optimization, the hyperparameter variable is the key result of the hyperparameter tuning, and failures affecting this dataset will significantly affect the entire workflow. In contrast, variables used for logging purpose are less important and unavailability/corruption of such data rarely impacts the final result of the workflow. This warrants a need to support the ability to set the data redundancy level at the granularity of variables.

In the following section, we introduce CoREC with multilevel data redundancy (CoREC-multilevel). Unlike CoREC, which only cares about data access frequency and applies a universal data redundancy for all data, CoREC-multilevel takes into account the resilience requirement of applications and variables. Specifically, we enable a varying data redundancy scheme, which corresponds to different n-way replications and erasure coding schemes based on the data resilience requirements. In CoREC-multilevel, each variable has an individual level of data redundancy, which is set by the application, and the global storage efficiency constraint is set as an upper bound of storage cost in the staging area.

Assuming that the overall cost of multilevel replications is the sum of the cost of each individual replication scheme (C_{ri}) weighted by its percentage (P_{ri}), the expected cost of the multilevel method that combines n replication schemes $\widetilde{C}_r$ is:

$$\widetilde{C}_r = P_{r1}C_{r1} + P_{r2}C_{r2} + \cdots + P_{rn}C_{rn}.$$

In the same way, the expected cost for the n erasure coding schemes $\widetilde{C}_e$ is:

$$\widetilde{C}_e = P_{e1}C_{e1} + P_{e2}C_{e2} + \cdots + P_{en}C_{en}.$$

From these equations and Equation (8), the time complexity for CoREC-multilevel C_{CoRECM} under average replication $\widetilde{C}_r$ and erasure coding $\widetilde{C}_e$ costs can be computed as:

$$C_{CoRECM} = (\widetilde{C}_r f_h - \widetilde{C}_e f_c + (\widetilde{C}_e - \widetilde{C}_r) f_n r_m) n P_h + \widetilde{C}_e f_c n. \tag{10}$$

Similarly, we can get the average storage efficiency for replication $\widetilde{E}_r$ and erasure coding $\widetilde{E}_e$ as:

$$\widetilde{E}_r = P_{r1}E_{r1} + P_{r2}E_{r2} + \cdots + P_{rn}E_{rn},$$

$$\widetilde{E}_e = P_{e1}E_{e1} + P_{e2}E_{e2} + \cdots + P_{en}E_{en}.$$

The storage efficiency for CoREC-multilevel is then given by:

$$E_{CoRECM} = \frac{\widetilde{E}_e \widetilde{E}_r}{\widetilde{E}_e P_r + \widetilde{E}_r P_e}. \tag{11}$$

3 COREC SYSTEM DESIGN

CoREC is composed of three key components, i.e., the grouped replication & erasure coding based data placement scheme, the load balancing & conflict-avoid encoding workflow, and the lazy recovery strategy. In this section, we present the overall design and implementation details of CoREC and describe these three components.

3.1 Data Placement

3.1.1 Grouped Replication & Erasure Coding Scheme. To tolerate concurrent staging server failures (i.e., node failure), we divide staging servers into replication groups and erasure coding groups. A replication group includes the data object and its replica, and an erasure coding group includes data objects and their parities. The grouped replication and erasure coding scheme overcomes the limitation of random replication and makes data objects able to survive concurrent failures with higher probability. Figure 5 shows an example of how two-way replication and erasure coding group ($k = 3, n = 4$) scheme work in a 16-server data staging.

The placement of replicas and data/parity objects on staging servers in the physical organization can also have a critical effect on data resilience. In many cases, a single event such as a power failure or a physical disturbance will affect multiple devices and greatly increases the risk of data loss. By reflecting the underlying physical organization of data staging servers, our approach can model and thereby address potential sources of correlated staging server failures. Specifically, in CoREC, we reorder the data staging server ID based on network topology and organize them in a logical ring. Each server is followed in the logical ordering by a server on a different node or cabinet so that as many as n contiguous servers belong to n different nodes or cabinets. By encoding this information into the logical network topology, our data placement policy can separate the data object, its replicas, and parity objects across different failure groups while maintaining the desired distribution. As depicted in Figure 5, a 16-server data staging that is located in four dedicated compute nodes can tolerate arbitrary one node failure.

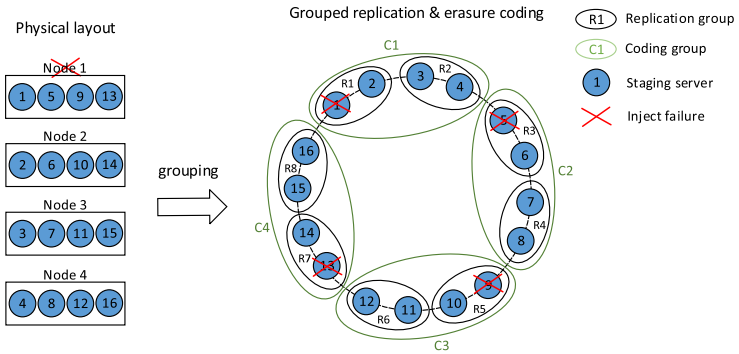


Fig. 5. Data Objects, Replicas, and Parity layout in data staging. Servers 1 and 2 are in the same replication group, while servers 1, 2, 3, and 4 belong to the same coding group. This topology-aware data layout can tolerate arbitrary single node failure.

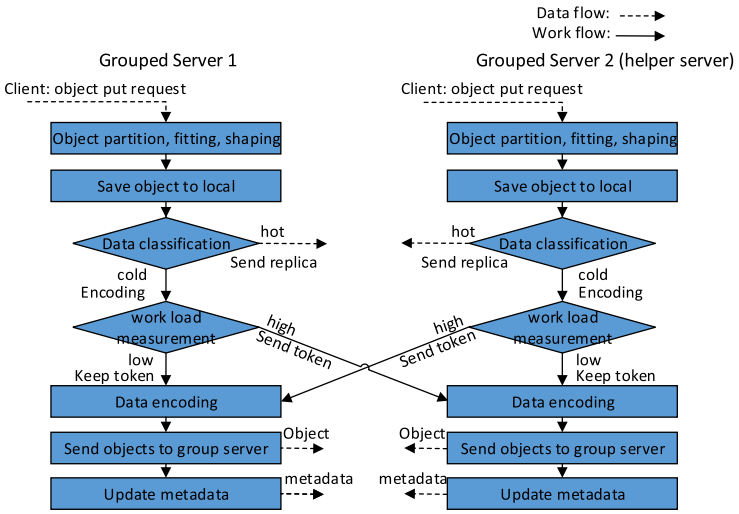


Fig. 6. Encoding workflow in CoREC.

3.2 Load Balancing & Conflict-avoid Encoding Workflow

In CoREC, data objects are encoded in staging servers during transition from replication to erasure coding. If one staging server is currently busy with a large read-write workload, then assigning the encoding task to this server will impact other requests being served, as well as the encoding time. CoREC addresses this interference with a load-balancing & conflict-avoid encoding workflow. Since hot data objects are always replicated, CoREC can simply select the staging server with the lightest workload in the replication group to perform data classification and encoding operation.

Figure 6 illustrates an encoding workflow with one server and one paired server, also called helper server, executing on a replication group of size 2. The encoding workflow is triggered by the server when it receives an object-put request from a client. Once server receives and pre-processes the data object, data classification component classifies data objects and makes decision for the data resilience approach based on data frequency and storage efficiency constraint. After that, the workload measurement component decides whether to encode locally or let the helper

server encode based on its workload level. If local node's workload is high, then it sends the replica node (node with replica data) an encoding token to perform erasure coding. Otherwise, the server performs encoding locally. After the server performs the encoding operation, it sends data & parity objects to other servers in the erasure coding group.

The encoding workflow is composed of four principal components. First, a data fitting and partition component pre-processes the data objects into a specific size and shape. Second, a data classification and encoding component classifies data object and makes it resilient. Third, a workload measurement component measures a server's workload level based on the frequency of client read-write requests. Finally, a data/parity object consistency mechanism provides atomic encoding processing for each data object. In a replication group, all servers share one encoding token and the server can get the encoding token only if it has a low workload. Only the server that holds an encoding token can perform an encoding operation, which ensures that exactly one stripe is placed in the coding grouped servers. It also ensures that the less busy server in the group performs more encoding operations than the busier one and workload is balanced throughout the coding group.

3.3 Data Size & Geometric Shape

While very small data objects suffer from metadata overheads, larger data objects have relatively smaller metadata overheads and achieve better throughput during asynchronous communication such as RDMA [14]. However, large-sized data objects increase the processing time required for data encoding, decoding, replication, and transportation [40]. This leads to longer data access latencies. Thus, an appropriate object size is required to balance metadata overhead and data access latency.

To fit data objects into desirable size and shape on the servers, the data fitting and partition component in CoREC uses Algorithm 1. In this algorithm, we first set a range of target data object sizes. When a staging server receives a data object that is larger than the range, we partition the object into halves along the longest geometric dimension. This is done repeatedly until all sub-objects fall into the range of target size. This simple binary partition algorithm ensures that data objects do not exceed a threshold size. Partitioning in this way ensures a balance between the size of objects and the quantity of objects. Under perfect conditions, every object can be partitioned into regular and uniform n-dimensional objects.

ALGORITHM 1: Geometric partitioning and fitting of an object

Input: Data Object (*object*), metadata, dimension (n), fitting size (*size*);
Output: Fitting data objects ($object[m]$), metadata ($metadata[m]$);

```
N ⇐ 1
object[m] ⇐ object
while N ≠ 0 do
  if ∃ obj in object[m] > size then
    get maximum boundary size of obj in dimension n
    partition boundary to half
    partition obj to half
    metadata[m] ⇐ metadata
    object[m] ⇐ obj
  else {Object is fitting}
    return object[m], metadata[m]
  end if
end while
```

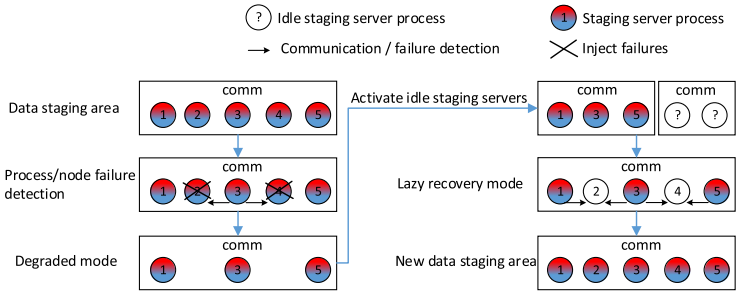


Fig. 7. Data and process recovery in data staging area.

3.4 Recovering Data Staging Server Failures

Existing large-scale resilient storage solutions typically use an aggressive data recovery strategy[12]. Whenever a failure on one or more servers is detected, all lost objects are recovered and re-generated onto active servers immediately. The problem with such an aggressive data recovery scheme is that it requires significant resources to recover data from a failure. Decoding operations and data transportation may consume considerable network and computing resources in a short time window. These overheads eventually hinder the application read-write requests. In CoREC, we propose a new lazy recovery scheme with a time limit on delayed data recovery. As shown in Figure 7, recovering data staging servers from failures involves four key steps: (*i*) failure detection, (*ii*) data recovery in the degraded mode, (*iii*) process recovery, and (*iv*) data recovery in the lazy recovery mode. CoREC introduces two data recovery modes (the degraded mode and lazy recovery mode) for data recovery.

3.4.1 Failure Detection. In CoREC, the function of detection and handling of failures is delegated to ULFM-enabled MPI. As a proposed extension of the MPI standard, ULFM [7, 8] includes mechanisms to tolerate fail-stop failures without the need to restart all processes linked to the MPI communicator. We leverage ULFM to tolerate and recover from such failures in the data staging area. ULFM guarantees that MPI operations involving communication should return an ERR_PROC_FAILED error code if the runtime detects a process failure in the data staging communication area. ULFM-specific return codes are captured using MPI's profiling interface and no changes in the MPI runtime itself are required. In data staging frameworks, the data exchange between applications happen via reads/writes from/to the staging area. The reads/writes are facilitated via asynchronous RDMA. When staging server processes fail or are unavailable, RDMA error codes can also be used to detect these failures.

3.4.2 Degraded Mode. Once a data staging server detects a process failure, it distributes failure notifications to the remaining data staging servers. The data staging area is then shrunk to remove the failed staging servers and the rest of the staging servers switch to a degraded mode, as shown in Figure 7. In this mode, only the requested data are re-constructed and transferred to the client. These temporarily re-constructed data are discarded once the read request is served. The reconstruction of failed data objects in the read-path increases read latency. Experimental evaluation results for the reading performance in degraded mode are presented in Section 4.

3.4.3 Process Recovery. To be able to recover from process failures, CoREC reserves a few staging server processes as backup processes. The number of backup processes is determined by the choice of erasure code and server process density on a node. For example, if CoREC is initialized to use erasure code with $n = 8, k = 6$, the goal is to tolerate 2 node failures per 6 nodes. If the each

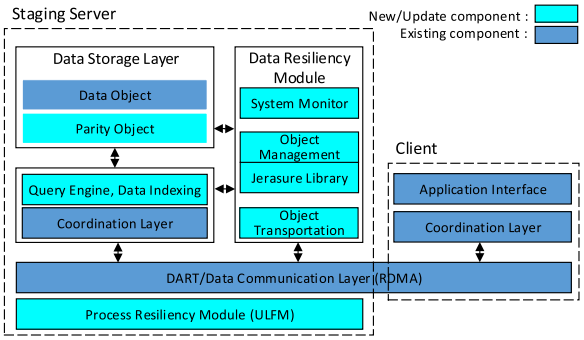


Fig. 8. System architecture.

node has 8 server processes, then for process recovery, CoREC initiates 16 backup processes per 48 staging processes. When failures are detected, these idle data staging processes will be activated and merged with the old data staging processes group. These newly activated processes will be reassigned the same rank numbers as failed ones. An alternative approach is to spawn new processes instead of use processes from a previously prepared process pool, if this is supported by the job scheduler.

3.4.4 Lazy Recovery Mode. After a replacement server joins data staging, CoREC switches to the lazy recovery mode. In this mode, each object on the failed server will be recovered immediately after it is queried or updated. The recovery of all other remaining objects are triggered based on the time limit set for delayed data recovery. The time-limit setting depends on the fault tolerance requirement for data objects and the overall MTBF of the system. Normally, too long of a time-limit constraint results in an unacceptably high risk of permanently losing the data, as it increases the chance of multiple failures in the same group. However, too short time-limit constraint risks interfering with the application's regular requests in the same way as aggressive recovery. Specifically, CoREC uses $\frac{1}{4}MTBF$ as the recovery timeline constraint. In many data-intensive simulation applications, most of the failed objects will be recovered much earlier than the end of the timeline due to high frequency of update and query requests.

4 EXPERIMENTAL EVALUATION

This section describes the implementation details of CoREC and presents an experimental evaluation using synthetic benchmarks as well as the S3D combustion simulation and analysis workflow [11].

CoREC is implemented on top of DataSpaces [14], an open-source data staging framework. The schematic overview of the runtime system is presented in Figure 8. In addition to modifying several existing components of DataSpaces for the integration, the system architecture introduces three key new components in data resiliency module: Local Object Management, Object Transportation, and System Status Monitor. The Local Object Management component maintains local data objects, replicas, parity objects, and metadata. It also stores the data object classification information in addition to performing the encoding, decoding, and object preprocessing tasks. We use the Jerasure open-source library [32] to perform encode/decode operations. While evaluation results demonstrate the efficacy of CoREC when using Reed-Solomon code, the Jerasure library offers a variety of erasure codes to choose from and it is straightforward to change the erasure code used in CoREC. The Object Transportation component synchronizes data objects, replicas, parities, and metadata while managing the transportation of objects between different staging servers. Server's

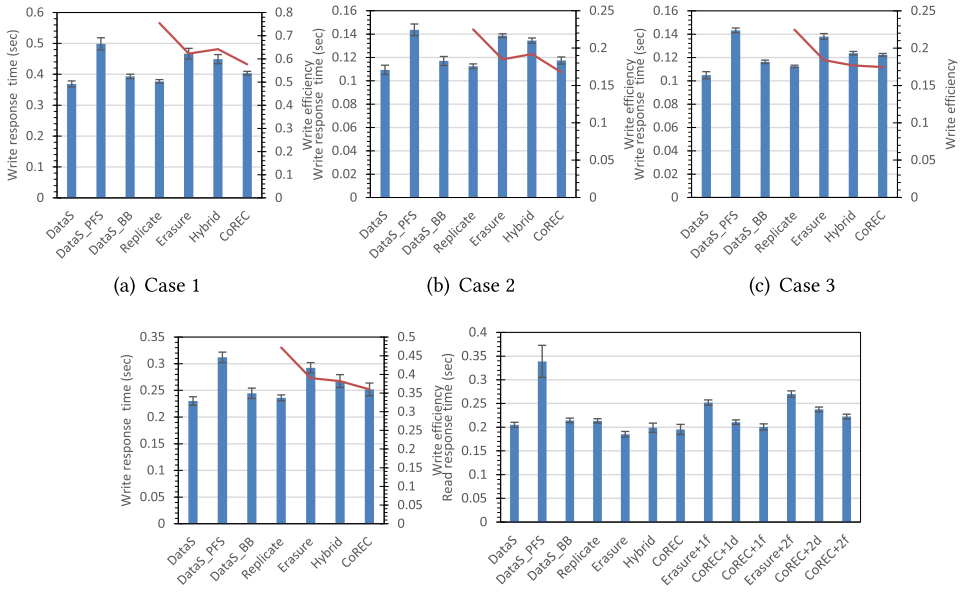
Table 1. Experimental Setup for Synthetic Tests

Total number of cores	64 + 32 + 8 = 104
No. of parallel writer cores	4 × 4 × 4 = 64
No. of staging cores	8
No. of parallel reader cores	32
Volume size	256 × 256 × 256
In-staging data size (20 TSs)	2560 MB
No. of replica	1
No. of data objects	3
No. of parity objects	1
Coding technique	Reed-Solomon Code
Storage efficiency for hybrid erasure coding	67%
Storage efficiency lower-bound for CoREC	67%

workload monitoring and failure detection is performed by the System Status Monitor component. To recover from failed staging server processes, we also introduce an additional process resiliency module. This module manages a spare process pool and implements the detection and handling of staging server failures using ULFM, which offers a set of fault-tolerance mechanisms for MPI applications.

4.0.1 Synthetic Experiments. Our synthetic experiments were performed on both the ORNL Titan Cray XK7 system and the NERSC Cori Cray XC40 system. These experiments evaluate the read and write performance of applications with different data read and write patterns when they use CoREC for resilient data staging. To better understand the performance and effectiveness of our approach, we selected five test cases with common data reading and writing patterns used by real scientific simulation workflows. In these cases, we assume that scientific applications write data to a 3-dimensional global space (*data domain*). We also assume that data are written in multiple iterations (*time steps*) as described in five test cases below. We compared CoREC with five other fault tolerance mechanisms: DataS_PFS (replicates all data objects and places replicas on the parallel file system), DataS_BB (replicates all data objects and places replicas on burst buffer nodes), Replication (replicates all data objects and places replicas on peer staging servers), Erasure Coding (encodes all data objects locally and places data/parity objects on peer staging servers), and Hybrid Erasure Coding (data objects are classified and selected for replication/erasure coding under the LRU algorithm and a defined constraint on storage overhead). We additionally compared our results to the performance of data staging without any fault tolerance. To evaluate the balance between the write response time and the storage cost for various data resilience techniques, we introduce *write efficiency*, which is a ratio of application's observed write response time to the storage efficiency of the data resiliency technique. The low write efficiency value indicates a better balance between time and storage cost for data resilience. The setup of these experiments is described in Table 1. The experimental results are presented in Figure 9 (collected on Cori) and Figure 10 (collected on Titan), along with a detailed discussion and analysis of these results.

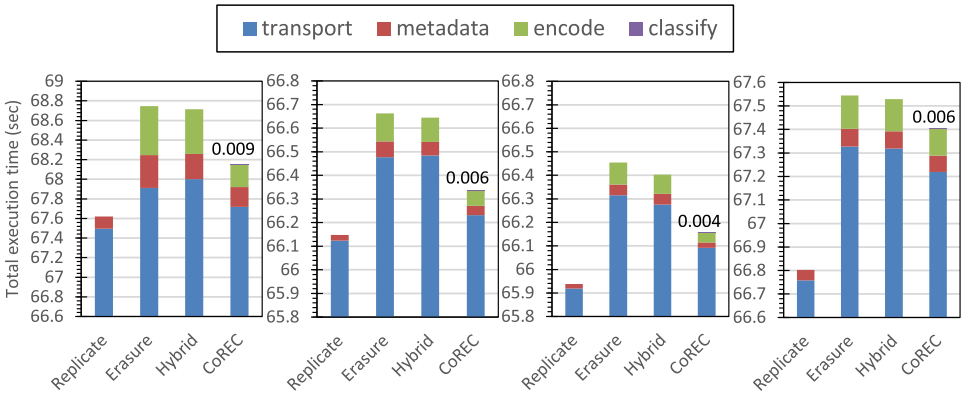
(1) **Case 1** - *Write the entire data domain in each time step*: In this case, the data of the entire domain are written at every simulation time step. Since there is no data replication, encoding, data movement, or metadata synchronization overhead, the data staging without fault tolerance has the best relative data write response time. For the fault tolerance approaches, although the replication approach with unlimited storage constraint on peer memory or burst buffer does not incur the overhead of data encoding, I/O operation, or extra data transportation overhead, these approaches have only achieved 6.9% and 2.6% smaller write response time in comparison to CoREC,



(a) Case 1 (b) Case 2 (c) Case 3

(d) Case 4 (e) Case 5

Fig. 9. Average data write and read response time *(blue bars)* and Write Efficiency = *Write response time/Storage Efficiency (red line)* of different data resilience mechanisms for the five test cases using different writing patterns. *DataS:* Data staging without fault tolerance; *DataS_PFS:* Data are stored in PFS for resilience; *DataS_BB:* Data are stored in Burst Buffer for resilience; *Replicate:* Data are replicated in peer memory for resilience; *Erasure:* Data are erasure coded for resilience; *Hybrid:* hybrid erasure coding with LRU data classification; *CoREC+1d and CoREC+2d:* CoREC in degraded mode with 1 and 2 server failures; *CoREC+1f and CoREC+2f:* CoREC in lazy recovery mode with 1 and 2 server failures; *Erasure+1f and Erasure+2f:* Erasure coded data staging with an aggressive recovery strategy under 1 and 2 server failures.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Fig. 10. Breakdown of the total execution time (in seconds) for the workflows in Figure 9. *transport:* Time spent in data movement; *metadata:* Time spent to update the distributed metadata; *encode:* Time spent to perform data encoding; *classify:* Time spent for data classification in CoREC (listed as number).

respectively. Meanwhile, storing the replicated data into PFS gets the worst write access performance and the longest total workflow execution time due to intensive I/O operations. Also, the result for the erasure coding method shows the second worst performance for both write access and total workflow execution time, because of the overhead associated with frequently encoding the original data objects and the placement of data/parity objects on peer servers. Although only a portion of data objects are erasure coded in hybrid erasure coding, frequently switching between replication and erasure coding approach on the same data object makes this approach's write performance just slightly better than the erasure coding approach and has longest total transportation time. For CoREC, due to the write-intensive workload, the workload balance and conflict-avoid encoding workflow plays a vital role in minimizing the interference to regular request. CoREC achieves a decrease of 48.7% and 53.2% in encoding time and an improvement of 13.5% and 10.1% in the write response time, relative to erasure coding and hybrid erasure coding, respectively. The lower-bound constraint for storage efficiency in CoREC causes some data objects to be erasure coded, even if they are hot, and this leads to a 6.9% increase in write-time as compared to replication.

(2) **Case 2** - *Write the entire data domain in multiple time steps*: In this case, the entire data domain is divided into four subdomains, and each subdomain is written in a time step. This means that in every four time steps, the entire data domain is written. Since each subdomain has the same write access frequency, all data objects in that subdomain are either hot or cold. However, CoREC leverages its multi-time-step look-ahead mechanism to efficiently convert data objects from cold to hot, i.e., moving from erasure coding to replication. Thus, CoREC has better (around 12.7%) performance improvement for write response time and 38.4% decrease in the encoding time with respect to hybrid erasure coding, while incurring an overhead of 4.3% in the write response time over replication. In addition, the conflict-avoid encoding workflow and less data conversion from replication to erasure coding contributes to having smaller data transportation overhead than hybrid erasure coding.

(3) **Case 3** - *Write a subset of the data domain at a higher frequency than others*: In this case, data objects of a subdomain in a particular domain are written at higher frequency and data objects in other subdomains are written just once. This setup addresses the presence of hot spots in the data domain. Both CoREC and hybrid erasure coding with LRU can easily identify these hot data objects and apply the corresponding resiliency technique. Since erasure coding always selects all data objects as candidates for erasure coding, CoREC improves the write response time by 11.3% and 1.1% and decreases encoding time by 50.4% and 16.5%, respectively, while increasing the write response time by just 8.8% as compared to replication.

(4) **Case 4** - *Write subsets of the data domain with random access pattern*: This case differs from the previous case, as the subdomains of the data domain are randomly chosen for writing/updating. The random access pattern reduces the accuracy of the data classifier, which is based on temporal and spatial locality. However, the workload balance and conflict-avoid encoding workflow optimizations enhance the performance of CoREC by 13.8% and 5.8% and decrease encoding time by 14.6% and 17.8% compared to erasure coding and hybrid erasure coding, respectively.

Figure 10 shows the breakdown of the normalized execution time for workflows in aforementioned cases in failure-free case. The plots show that CoREC has lower overheads compared to hybrid erasure coding and pure erasure coding in all cases. CoREC has less data transport time than erasure coding and hybrid technique, because fewer erasure coded objects incur updates and it minimizes the parity update operations, which leads to less encoding time also. While replication has better performance, it should be noted that it suffers from high storage overhead.

(5) **Case 5** - *Read entire data domain in each time step*: The data of the entire domain are read for every time step. In this case, the replication method, either in peer memory or burst buffer, has a

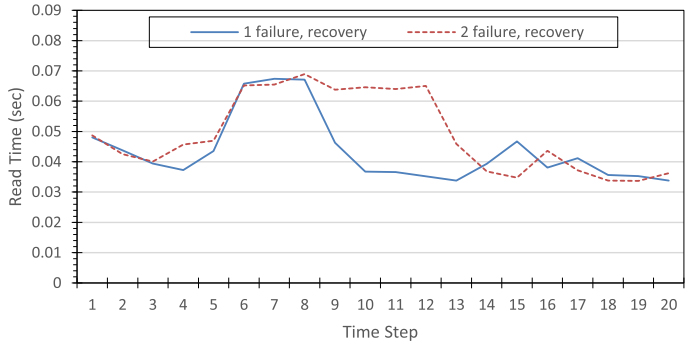


Fig. 11. The average read response time for reading the entire data domain with 1 and 2 failures, along with failure recovery, for 20 time steps. The first failure occurs at time step 4, and second failure occurs at time step 6. First failure-recovery begins at the 8th time step and another recovery is initiated at the 12th time step, and they end at time steps 9 and 13, respectively.

similar read response time to data staging without fault tolerance, meanwhile replicating data to the PFS results in the worst performance among all fault tolerance approaches. Since erasure coding splits original data objects into small objects and distributes them among the staging servers, a single read request can be distributed across multiple servers and consequently erasure coding, hybrid erasure coding, and CoREC have better read response times than both replication and the original data staging technique. We also performed experiments for various cases of reads as we did for writes, but the results are not presented in this article due to the similar patterns as case 5. We also evaluated read response time of CoREC in the presence of failures. In degraded mode, the read response time increases by 7.93% and 21.7% for single and double server failures, respectively, as compared to failure-free case. However, when using lazy recovery, the read response time increases by 2.66% for single failure and 13.9% for double server failures as compared to failure-free case.

While we demonstrated that CoREC performs better on average than both erasure coding and hybrid erasure coding, replication might seem like a good choice for fault tolerance. However, we also need to consider the storage overhead associated with each fault tolerance mechanism. We also plot the ratio of write-response time and storage efficiency in Figure 9. It can be seen that data staging without fault-tolerance provides best performance along with best storage efficiency. However, fault-tolerance introduces overheads on both write response time and storage efficiency. Among the fault-tolerant mechanisms, CoREC provides the best balance for storage efficiency and write response time in all the data access patterns studied.

To study the impact of lazy recovery in CoREC, we also perform the experiment on Titan and plot the read response time at every time step for 20 time steps in Figure 11. For a single failure case, we inject a staging server failure at time step 4 and recover it at time step 8. For the two failure cases, we inject a first staging server failure at time step 4 and a second failure at time step 6 and then start recovering them at time steps 8 and 12, respectively. In both the cases, the entire data domain was read for all time steps. We observe that, unlike aggressive recovery, our lazy recovery approach does not trigger data recovery for the failed server immediately, which may result in an increased data read response time. From time step 8 to time step 9, our approach gradually recovers unavailable data objects, which leads to a nominal increase in the data read response time for recovery from multi-server failure. After time step 14, the data read response time resets back to what it had been before the failure was injected.

Table 2. Configuration of Core-allocations, Data Sizes, and Data Resilience for the Three Test Scenarios on 4,480, 8,960, and 17,920 Cores

No. of cores	4,480	8,960	17,920
No. of simulation cores	$16 \times 16 \times 16 = 4{,}096$	$32 \times 16 \times 16 = 8{,}192$	$32 \times 32 \times 16 = 16{,}896$
No. of staging cores	256	512	1,024
No. of analysis cores	128	256	512
Volume size	$1{,}024 \times 1{,}024 \times 1{,}024$	$2{,}048 \times 1{,}024 \times 1{,}024$	$2{,}048 \times 2{,}048 \times 1{,}024$
Data size (GB)	160	320	640
No. of replica	1	1	1
No. of data objects	3	3	3
No. of parity objects	1	1	1
Storage efficiency	67%	67%	67%

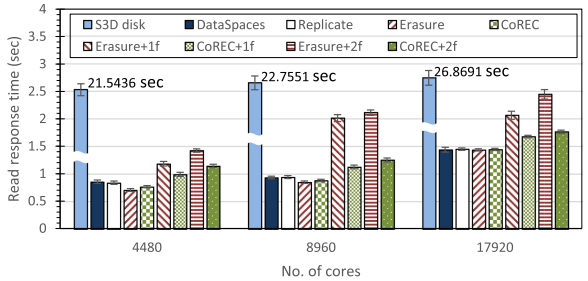


Fig. 12. Comparison of the cumulative data read response time using the S3D and coupled analysis workflow on Titan.

4.0.2 Large Scale S3D Experiment. We also performed large-scale tests for CoREC using the lifted hydrogen combustion simulation workflow using S3D [11] and an analysis application on Titan and compared it to pure replication and erasure codes. CoREC was tested using three different core counts (4,480, 8,960, and 17,920) and corresponding grid domain sizes so that each core was assigned a spatial sub-domain of size $64 \times 64 \times 64$. In terms of the data access pattern, in this experiment, the S3D simulation wrote the vector field component pressure of entire domain to data staging at each time, which was processed for feature extraction by the visualization application later. For comparison purpose, we also ran S3D without data staging, S3D with data staging but without resilience, and S3D with data staging and resilience. The cumulative time for reading/writing data over 20 time steps was measured. The core configurations, the data region assignments, and data resilience for our experimental setup are summarized in Table 2.

Figure 12 and Figure 13 illustrate the experimental results for the S3D coupled simulation and analysis application workflow for various resiliency settings. Since the PFS (parallel filesystem) based S3D does not have data staging and the data are saved to disk, it has the longest read and write response time. While data staging without resilience shows best performance, it is not able to recover from failures. Among the resilient data staging techniques studied, CoREC reduces the write response time by 7.3%, 14.8%, and 5.4% as compared to pure erasure coding on 4,480, 8,960, and 17,920 cores, respectively. In comparison to replication, CoREC has an overhead of 4.2%, 5.3%, and 17.2% in write response time on 4,480, 8,960, and 17,920 cores, respectively. It can also be seen that in the presence of failures, CoREC reduces the read response time by up to 40.8% and 37.4% for one and two server failures, respectively, as compared to pure erasure coding.

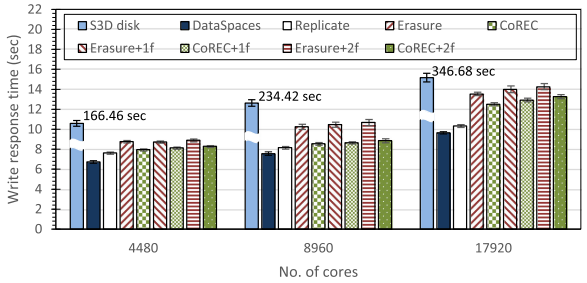


Fig. 13. Comparison of the cumulative data write response time using the S3D and coupled analysis workflow on Titan.

These results show that CoREC demonstrates good overall scalability and better storage efficiency with small overheads for different processor counts and data sizes while providing data resiliency for extreme-scale HPC systems.

4.1 Experiments with Node Failures

The goal of these experiments is to demonstrate that CoREC is capable of handling data and process recovery under high-frequency node failures.

4.1.1 Experimental Setup. We have deployed CoREC with ULFM on the Caliburn cluster, which consists of 560 compute nodes, each containing two 18-core Intel Xeon E5-2695v4 processors, 256 GB of main memory, and an Intel Omni-Path Host-Fabric interface adapter at Rutgers Discovery Informatics Institute (RDI2). In this experiment, we evaluate the overhead related to data recovery for staging node failures. In our experiments, a staging node failure is equivalent to N staging server processes failures at same time, where N is the total number of processes per dedicated staging node. To perform these experiments, node failures are injected by simultaneously sending SIGKILL signals to all the staging server processes running on a particular node. Since the data objects, parity objects, replicas, and metadata are stored in process memory, killing server processes make such data unavailable. This is consistent with the behavior of real node failures. In our experiments, one compute node of the Caliburn cluster runs 8 staging processes, which translates to $N = 8$ for node failures. Unless specified otherwise, all tests have been repeated five times. We ran some preliminary experiments for high values of MTBFs (such as 5 minutes, 10 minutes, or 30 minutes, etc.) and observed negligible recovery overheads relative to the total execution time. Subsequent experiments were run under MTBFs for less than a minute. The data size for each data staging server was 50 MB. We also ran this workflow with a failure-free case as the baseline. The setup of these experiments is described in Table 3.

4.1.2 Experiment Description and Results. For node failure experiments, we studied the read/write response time for different data/process recovery strategies in the data staging. We also evaluated the total overhead for workflows to empirically demonstrate the low cost of data recovery and small latency impact. We performed these experiments on a 256-core (32 node) data staging area. Figures 14 and 15 plot the average read/write response time for different frequencies of node failures injected for a synthetic workflow with a total execution time of about 150 seconds. The synthetic failure rates range from 18 to 150 seconds, and the corresponding total number of failures range from 8 processes (1 node) to 64 processes (8 nodes), as noted on top of each bar, over a total time period of about 150 seconds.

Figure 14 shows the cumulative read-response time for 50 time steps. The read-response time increases with the failure rate. In the worst case (8 node failures), the read-response time increases

Table 3. Experimental Setup for Node Failures Tests

Total number of cores	1,024 + 256 + 128 = 1,408
No. of parallel writer cores	8 × 8 × 16 = 1,024
No. of staging cores	256 (32 nodes)
No. of parallel reader cores	128
Volume size	128 × 128 × 256
In-staging data size (50 TSs)	3.2 GB
No. of replica	1
No. of data objects	3
No. of parity objects	1
Coding technique	Reed-Solomon Code
Storage efficiency lower-bound	67%

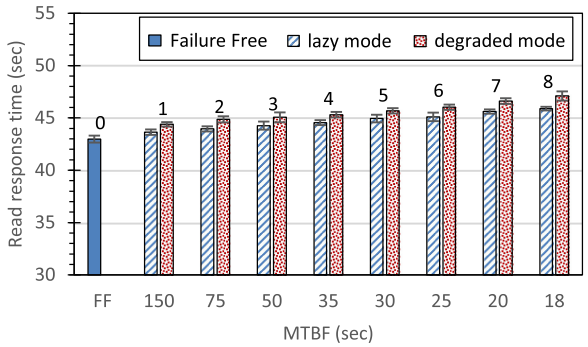


Fig. 14. Comparison of the cumulative data read response time using the synthetic workflow on Caliburn. *FF:* in the x-axis represents CoREC in failure-free case.

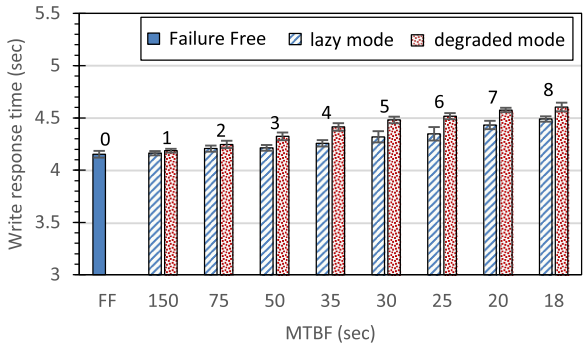


Fig. 15. Comparison of the cumulative data write response time using the synthetic workflow on Caliburn. *FF:* CoREC with failure-free case.

9.58% in degraded mode and 6.77% in lazy mode as compared to the failure-free case (FF). The write-response time shown in Figure 15 shows the similar trend with the read-response time. The write-response time increases by only 2.41% in degraded mode and 1.13% in the lazy mode for an MTBF of 150 seconds. For the MTBF of 18 seconds, the write-response time increases 10.88% in degraded mode and 8.11% in lazy mode as compared to FF. As a whole, the experiment results

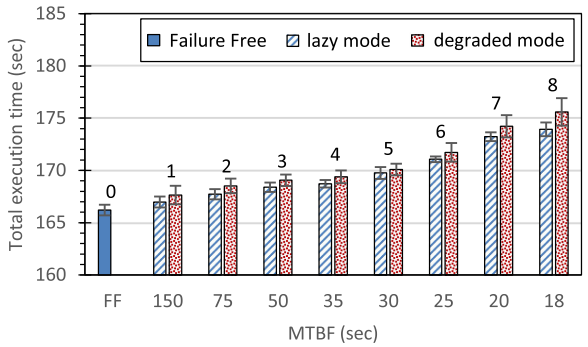


Fig. 16. Comparison of the total execution time using the synthetic workflow on Caliburn. *FF:* CoREC in failure-free case.

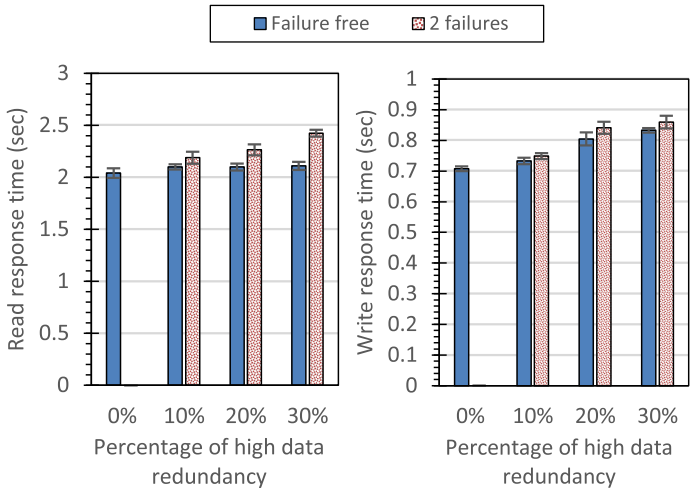
demonstrate that grouped replication and coding scheme with lazy recovery mode can tolerate frequent node failures with minimal overhead.

We also study the impact of data and process recovery on the total workflow execution time, which includes the overhead of data replication, encoding, decoding, and process recovery. Specifically, we compare the end-to-end workflow execution time under varying MTBFs and the failure-free case. The leftmost bar in Figure 16 shows the total workflow execution time for the failure-free case. By using lazy recovery, we obtain much lower performance penalties even at the higher failure rates. In comparison to the baseline failure-free case, the total execution time is increased by only about 0.8% in degraded mode and by 0.5% in lazy recovery mode for the MTBF of 150 seconds. For the case of failures occurring every 18 seconds, the workflow execution time increases by about 5.69% in degraded mode and by 4.25% in lazy recovery mode. In short, CoREC can handle data recovery under frequent node failures with total overheads of up to 5.69% in degraded mode and 4.25% in lazy mode for the worst-case scenario.

4.2 Experiments for CoREC with Multilevel Data Redundancy

In this section, we demonstrate that CoREC-multilevel can efficiently provide varying levels of data redundancy based on application requirements. Although CoREC-multilevel uses different redundancy schemes to satisfy resiliency requirements, the storage efficiency constraint is maintained by managing the trade-off between storage and performance, and duplication/triplications and erasure coding. Since CoREC-multilevel is intelligent enough to make dynamic decisions based on data access characteristics, we did not observe a significant impact on application performance. The synthetic workflow writes two variables into data staging. These two variables are intended to representative high- and low-redundancy datasets. Within the workflow, the coupled components write and read these two variables into the data staging area concurrently in each time step, and the response time is measured.

For high data redundancy, we use $RS(6, 4)$ and triplication to tolerance-concurrent failures in two arbitrary staging servers. For low data redundancy, we use $RS(6, 5)$ and duplication to tolerate one staging server failure. To study the impact of lazy recovery in the presence of concurrent failures for CoREC-multilevel, we insert failures into two staging servers concurrently. During a total runtime of 20 time steps the failures were inserted only once and then recovered in subsequent time steps using the lazy recovery mechanism. Since two staging servers have failed, only high-redundancy data can be recovered. As CoREC-multilevel can re-direct write requests to other staging servers when failures are detected, the low-redundancy data sets for all time steps are available except for the particular time step when the failures were introduced and detected. In



(a) Read response time of variables (b) Write response time of variables

Fig. 17. Read and write response time of variables under the percentage of high data redundancy in CoREC-multilevel. *Failure-free:* CoREC-multilevel in failure-free case.

Table 4. Experimental Setup for Multilevel Redundancy Tests

Total number of cores	512 + 120 + 512 = 1,144
No. of parallel writer cores	$8 \times 8 \times 8 = 512$
No. of staging cores	120
No. of parallel reader cores	512
Volume size	$256 \times 256 \times 256$
In-staging data size (20 TSs)	3,200 MB
Replication for high data redundancy	Triplication
Replication for low data redundancy	Duplication
Coding for high data redundancy	RS(6, 4)
Coding for low data redundancy	RS(6, 5)
Storage efficiency lower-bound	66.7%

our experiments, the total data exchanged between readers and writers is kept constant. We vary the percentage of high-redundancy data with regards to total data from 0 to 30 while keeping a constant storage efficiency constraint in CoREC-multilevel. Since the total data are composed of high- and low-redundancy data, it can also be viewed as varying low-redundancy data from 100% to 70%. Figure 17 shows both cumulative read-response and write-response time. A failure-free case is considered as baseline in our tests. The details of the experimental setup is listed in Table 4.

In Figure 17(b). We observed that when the percentage of high-redundancy data increases, the cumulative write-response time increases in Figure 17(b). This increase corresponds to the higher complexity or overheads of erasure coding and replication for higher data redundancy. When the amount of high redundancy increases, more data can be recovered and correspondingly the write response is affected to a higher degree in presence of failures. When all of the data are low redundancy, the data cannot be recovered when multiple failures are inserted. Thus, no write and read-response time is reported for 0% high-redundancy data. As compared to the failure-free case,

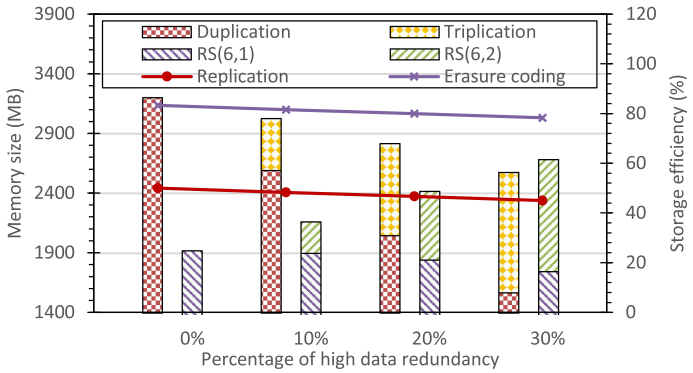


Fig. 18. Storage cost and efficiency for the percentage of high data redundancy in test scenarios.

CoREC-multilevel has an increase in cumulative write response time by around 2.2%, 4.5%, and 3.2% as compared to failure-free case.

When no failures are present, the read-response time remains fairly constant across varying percentages of high and low dataset. In the presence of failures, read-response times are directly impacted, because data need to be recovered first and then sent to the reader application. Under the presence of concurrent failures and data recovery, the read response time of CoREC-multilevel increases by around 4.1%, 7.9%, and 15.5% as compared to the failure-free cases, when high-redundancy data are set to 10%, 20%, and 30%, respectively.

To understand the impact of different data resiliency techniques on the overall storage capacity of the staging area, we measure the total memory consumed by the data in the staging area and also show the storage efficiency in Figure 18. When the percentage of high-redundancy data increases, the storage efficiency of replication decreases from 50% to 45%. Similarly, the storage efficiency of erasure coding also decreases from 83.3% to 78.3%. Correspondingly, the memory consumption for replication decreases from 3,200MB to the least 2,576MB, and the consumption for erasure coding increases from 1,920MB to the largest 2,683MB so as to maintain total storage usage is constant. Since the total amount of raw data written to the staging area is fixed, the overall storage efficiency also remains the same for different percentages of high- and low-redundancy data. From these results, we can infer that CoREC-multilevel is performant and can provide various data resiliency levels based on application needs.

5 RELATED WORK

The increasing performance gap between compute and I/O capabilities has motivated recent developments in both in-situ and in-transit data processing paradigms. In-situ and in-transit data processing allows analytics to directly access simulation data and has been used for visualization, indexing building, data compression, statistical analysis [5, 41], and so on. A number of in-memory data staging solutions such as DataSpaces [13] /ActiveSpaces [15], PreDatA [43] provide services for supporting in-situ and in-transit approaches, e.g., Reference [4], with a primary focus on fast and asynchronous data movement off simulation nodes. An alternative solution [6, 10, 31, 36] for in-situ and in-transit data processing is to perform data analysis on storage node storage controllers such as solid-state device (SSD), where the data already reside. Unfortunately, these frameworks do not address the resilience of in-situ/in-transit data processing, which is an important concern at extreme scales.

While supporting resilience in contexts other than in-situ/in-transit data analytics—such as Checkpointing [1, 20, 21, 38] and Replication/Erasure Coding [12, 39]—has been widely studied,

there are limited research efforts focused on in-situ/in-transit data processing systems. The study in Reference [26] exploits the reduction style processing pattern in analytics applications and reduces the complications of keeping checkpoints of the simulation and the analytics consistent. Research efforts in Reference [27] use a synchronous two-phase commit transactions protocol to tolerate failures in high performance and distributed computing systems. In comparison to these efforts, our data resilience approach specifically targets data staging based in-situ workflows and is more flexible, asynchronous, and scalable. Furthermore, it can handle dynamic execution and failure patterns across multiple applications that are part of in-situ/in-transit workflows.

Burst buffers are being increasingly used in HPC systems [2, 24, 25, 35] with the initial goal of relieving the bandwidth burden on the parallel file systems by providing an extra layer of low-latency storage between compute and storage resources. CoREC can easily be extended to use burst buffers. In this setting, CoREC would store the hot data in local DRAM memory and keep the cold data in the non-volatile storage layer on the burst buffer nodes to achieve faster read/write performance for workflows generating large amount of data . However, burst buffers themselves are not immune to failures. Furthermore, due to the difference in the physical typologies in burst buffers provided by vendors, the fault model and resiliency for burst buffers are complicated and remain an open challenge. The fault tolerance of burst buffers needs to explored further before any of the data resiliency methods, explored in the article, can be used on such devices.

Recent research [3, 22, 23, 28, 29] indicates temporal and spatial properties of failures in different software and hardware components. CoREC can easily tolerate such concurrent and correlated process/node failures using topology-aware grouped replication and erasure coding schemes discussed in Section 3.1. We believe CoREC can also be adapted to tolerate other classes of failures such as a GPU failure, which will potentially result in data loss at staging servers by extending the failure detection and handler mechanism to those failures.

While aspects of CoREC may appear conceptually similar to Cocytus [42], where replication is used for small-sized and scattered data (e.g., metadata and key) and erasure coding is used for large data (e.g., value), CoREC uses data access frequency rather than data size for data classification. In contrast to Cocytus, which is designed for cluster storage systems, CoREC targets in-situ/in-transit data processing on large-scale HPC systems.

6 CONCLUSION AND FUTURE WORK

Data-staging frameworks have emerged as effective solutions for addressing data-related challenges at extreme scale and supporting in-situ/in-transit workflows. However, the resilience of these frameworks remains a challenge. This article addresses data resiliency for staging-based in-situ/in-transit workflows. In this article, we presented CoREC and CoREC-multilevel, a scalable hybrid approach to data resilience for data staging frameworks that used online data access classification to effectively combine replication and erasure codes and to balance computation and storage overheads. CoREC-multilevel can support different data resiliency techniques to satisfy the varying data resiliency requirements of multiple applications. Furthermore, utilizing lazy recovery and conflict-avoid encoding workflow optimizations, we reduced the interference due data-resiliency on the simulation/analysis components of the workflow.

We have implemented CoREC on top of the DataSpaces data staging services and deployed it on the Titan Cray XK7 at OLCF, Cori Cray XC40 at NERSC, and the Caliburn system at RDI2. To evaluate its effectiveness and performance, we used both synthetic benchmarks and real-world large-scale S3D application. Our experiments demonstrated that CoREC can dynamically classify data objects based on data-driven access pattern and provide efficient data recovery in the presence of frequent process and node failures. The source code for our prototype implementation of CoREC is publicly available at https://github.com/shaohuaduan/datastaging-fault-tolerance.

As future work, we plan to expand CoREC to support multiple storage layers, for example, using NVRAM and SSD, and designing new models for data resilience that incorporate utility-based data placement across these layers.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback and suggestions.

REFERENCES

[1] Leonardo Arturo, Bautista Gomez, Naoya Maruyama, and Franck Cappello. 2010. Distributed diskless checkpoint for large scale systems. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*. 263–272.

[2] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. 2019. Sizing and partitioning strategies for burst-buffers to reduce IO contention. In *Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'19)*. 631–640.

[3] Leonardo Bautista-Gomez, Ana Gainaru, Swann Perarnau, Devesh Tiwari, Saurabh Gupta, Christian Engelmann, Franck Cappello, and Marc Snir. 2016. Reducing waste in extreme scale systems through introspective analysis. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. 631–640.

[4] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, Tong Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, Hongfeng Yu, Fan Zhang, and J. Chen. 2012. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*. 1–9.

[5] Janine C. Bennett, Vaidyanathan Krishnamoorthy, Shusen Liu, Ray W. Grout, Evatt R. Hawkes, Jacqueline H. Chen, Jason Shepherd, Valerio Pascucci, and Peer-Timo Bremer. 2011. Feature-based statistical analysis of combustion simulation data. *IEEE Trans. Vis. Comput. Graph.* 17, 12 (2011), 1822–1831.

[6] E. Wes Bethel, Martin Greenwald, Kersten Kleese van Dam, Manish Parashar, Stefan M. Wild, and H. Steven Wiley. 2016. Report of the DOE workshop on management, analysis, and visualization of experimental and observational data—The convergence of data and computing. In *2016 IEEE 12th International Conference on e-Science (e-Science)*. 213–222.

[7] Wesley Bland, Aurelien Bouteiller, Thomas Herault, George Bosilca, and Jack Dongarra. 2013. Post-failure recovery of MPI communication capability: Design and rationale. *Int. J. High Perform. Comput. Applic.* 27, 3 (2013), 244–254.

[8] Wesley Bland, Aurelien Bouteiller, Thomas Herault, Joshua Hursey, George Bosilca, and Jack Dongarra. 2012. An evaluation of user-level failure mitigation support in MPI. In *Proceedings of the 19th European MPI Users' Group Meeting (EuroMPI'12)*.

[9] Franck Cappello, Geist Al, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. 2014. Toward exascale resilience: 2014 update. *Supercomput. Front. Innov. Int. J.* 1 (2014), 5–28.

[10] Alexis Champsaur, Jay Lofstead, Jai Dayal, Matthew Wolf, Greg Eisenhauer, Patrick Widener, and Ada Gavrilovska. 2017. SmartBlock: An approach to standardizing in situ workflow components. In *Proceedings of the IEEE 31st International Symposium on Parallel and Distributed Processing Symposium Workshops (IPDPSW'17)*.

[11] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. 2009. Terascale direct numerical simulations of turbulent combustion using S3D. *Comput. Sci. Discov.* 2, 1 (2009).

[12] Asaf Cidon, Ryan Stutsman, Stephen Rumble, Sachin Katti, John Ousterhout, and Mendel Rosenblum. 2013. MinCopysets: Derandomizing replication in cloud storage. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*.

[13] Ciprian Docan, Manish Parashar, and Scott Klasky. 2010. DataSpaces: An interaction and coordination framework for coupled simulation workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. 25–36.

[14] Ciprian Docan, Manish Parashar, and Scott Klasky. 2012. DataSpaces: An interaction and coordination framework for coupled simulation workflows. *Cluster Comput.* 15, 2 (01 June 2012), 163–181.

[15] Ciprian Docan, Fan Zhang, Tong Jin, Hoang Bui, Qian Sun, Julian Cummings, Norbert Podhorszki, Scott Klasky, and Manish Parashar. 2014. ActiveSpaces: Exploring dynamic code deployment for extreme scale data processing. *Concurr. Comput. Pract. Exper.* 27, 14 (2014).

[16] Shaohua Duan, Pradeep Subedi, Keita Teranishi, Philip Davis, Hemanth Kolla, Marc Gamell, and Manish Parashar. 2018. Scalable data resilience for in-memory data staging. In *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*. 105–115.

[17] Ifeanyi P. Egwutuoha, David Levy, Bran Selic, and Shiping Chen. 2013. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *J. Supercomput.* 65, 3 (2013), 1302–1326.

[18] James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. 2012. Combining partial redundancy and checkpointing for HPC. In *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS'12)*.

[19] Marc Gamell, Keita Teranishi, Michael A. Heroux, Jackson Mayo, Hemanth Kolla, Jacqueline Chen, and Manish Parashar. 2015. Local recovery and failure masking for stencil-based applications at extreme scales. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'15)*.

[20] Shen Gao, Bingsheng He, and Jianliang Xu. 2015. Real-time in-memory checkpointing for future hybrid memory systems. In *Proceedings of the 29th ACM International Conference on Supercomputing*. 263–272.

[21] Leonardo Bautista Gomez, Dimitri Komatitsch, and Naoya Maruyama. 2012. FTI: High performance fault tolerance interface for hybrid systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 728–740.

[22] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: Long-term measurement, analysis, and implications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'17)*.

[23] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. 2015. Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems. In *Proceedings of the 45th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. 37–44.

[24] Anthony Kougkas, Hariharan Devarajan, Xian-He Sun, and Jay Lofstead. 2018. Harmonia: An interference-aware dynamic I/O scheduler for shared non-volatile burst buffers. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'18)*.

[25] Anthony Kougkas, Matthieu Dorier, Rob Latham, Rob Ross, and Xian-He Sun. 2016. Leveraging burst buffer coordination to prevent I/O interference. In *Proceedings of the IEEE 12th International Conference on e-Science (e-Science'16)*.

[26] Jiaqi Liu and Gagan Agrawal. 2017. Supporting fault-tolerance in presence of in-situ analytics. In *Proceedings of the 17th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'17)*. 304–313.

[27] Jay Lofstead, Jai Dayaly, Ivo Jimenezz, and Carlos Maltzahn. 2014. Efficient, failure resilient transactions for parallel and distributed computing. In *Proceedings of the International Workshop on Data Intensive Scalable Computing Systems*. 17–24.

[28] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H. Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. 519–530.

[29] Bin Nie, Ji Xue, Saurabh Gupta, Tirthak Patel, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2018. Machine learning models for GPU error prediction in a large scale HPC system. In *Proceedings of the 48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*. 95–106.

[30] I. S. Reed and G. Solomon. 1960. Polynomial codes over certain finite fields. *J. Soc. Industr. Appl. Math.* Vol. 8, 2 (1960), 300.

[31] Hyogi Sim, Youngjae Kim, Sudharshan S. Vazhkudai, Devesh Tiwari, Ali Anwar, Ali R. Butt, and Lavanya Ramakrishnan. 2015. AnalyzeThis: An analysis workflow-aware storage system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'15)*.

[32] James S. Plank, J. Luo, Catherine D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. 2009. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)*. 263–272.

[33] Pradeep Subedi, Philip Davis, Shaohua Duan, Scott Klasky, Hemanth Kolla, and Manish Parashar. 2018. Stacker: An autonomic data movement engine for extreme-scale data staging-based in-situ workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 73.

[34] Pradeep Subedi and Xubin He. 2013. A comprehensive analysis of XOR-based erasure codes tolerating 3 or more concurrent failures. In *Proceedings of the IEEE 27th International Symposium on Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW'13)*.

[35] Kun Tang, Ping Huang, Xubin He, Tao Lu, Sudharshan S. Vazhkudai, and Devesh Tiwari. 2017. Toward managing HPC burst buffers effectively: Draining strategy to regulate bursty I/O behavior. In *Proceedings of the IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'17)*.

[36] Devesh Tiwari, Simona Boboila, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter J. Desnoyers, and Yan Solihin. 2013. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*. 119–132.

[37] U.S. Department of Energy, Office of Science. 2018. Exascale Computing Project. Retrieved from https://www.exascaleproject.org/exascale-computing-project/.

[38] Dirk Vogt, Cristiano Giuffrida, Herbert Bos, and Andrew S. Tanenbaum. 2013. Techniques for efficient in-memory checkpointing. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*. 263–272.

[39] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*. USENIX Association, 307–320.

[40] Matt M. T. Yiu, Helen H. W. Chan, and Patrick P. C. Lee. 2017. Erasure coding for small objects in in-memory KV storage. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR'17)*.

[41] Hongfeng Yu, Chaoli Wang, Ray W. Grout, Jacqueline H. Chen, and Kwan-Liu Ma. 2010. In situ visualization for large-scale combustion simulations. *IEEE Comput. Graph. Applic.* 3 (2010), 45–57.

[42] Heng Zhang, Mingkai Dong, and Haibo Chen. 2016. Efficient and available in-memory KV-store with hybrid erasure coding and replication. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*.

[43] Fang Zheng, H. Abbasi, C. Docan, J. Lofstead, Qing Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. 2010. PreDatAPreparatory data analytics on peta-scale machines. In *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS'10)*. 1–12.

Received June 2019; revised December 2019; accepted February 2020