2018 IEEE International Parallel and Distributed Processing Symposium

Scalable Data Resilience for In-Memory Data Staging

Shaohua Duan*, Pradeep Subedi*, Philip Davis*, Keita Teranishi†, Hemanth Kolla†, Marc Gamell‡, Manish Parashar*
*Rutgers Discovery Informatics Institute, Rutgers University, Piscataway, NJ 08854, USA
†Sandia National Labortory, Livermore, CA 94550, USA
‡Intel, Austin, TX 78746, USA

***Abstract*—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 times 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 paper, we present CoREC, a scalable 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. The paper also 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 Titan at ORNL, and present an experimental evaluation in the paper. The experiments demonstrate that CoREC can tolerate in-memory data failures while maintaining low latency and sustaining high overall storage efficiency at large scales.**

I. Introduction

HPC applications are experiencing a dramatic increase in parallelism and performance as we transition from the current petascale era with many hundreds of thousands of cores [1], toward the exascale era. While the increase in parallelism and performance promises unprecedented computational and data analytics capabilities along with new insights into complex phenomena (e.g., fusion, combustion, astronomy, etc.) [2] [3], it brings forth new challenges. These challenges include growing data volumes and rates, and increasing costs (time and energy) for moving, analyzing and storing this data [4], as well as ensuring reliability in spite of failures affecting system components [5].

Data staging and in-situ/in-transit data-management techniques have emerged as effective solutions for addressing data-related challenges at extreme scales [6]. These techniques leverage resources (compute, storage) on the HPC system itself to stage data and to execute data-processing workflows close to where the data is being produced, which reduces the amount of data that needs to be moved off the system and stored to persistent storage (see Figure 1). For example, a multi-scale, multi-physics turbulent combustion application S3D [7], has an intricate data-processing workflow with multiple analyses performed at different temporal frequencies on

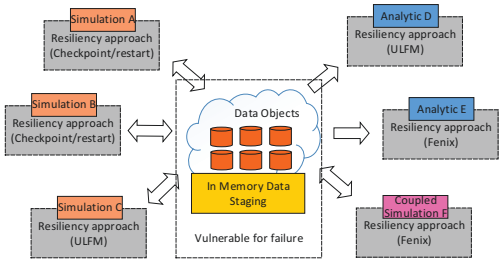


Fig. 1: A typical data staging workflow with fault tolerance.

non-overlapping subsets of data. The DataSpaces [6] staging-based in-situ framework has been effectively used to support these workflow requirements [8], [9].

There is also a rapidly growing body of research addressing the resilience of extreme-scale applications [10]. This includes approaches such as Checkpoint/Restart and runtime extensions to programming systems (e.g., ULFM, Fenix, etc.). However, the end-to-end application workflow, including the analyses components, is as important as the simulation itself: the final results of the overall computation are the outputs of the workflow, and contaminated or missing data due to faults in any component of the workflow can invalidate these outputs. As a result, ensuring the resilience of the end-to-end workflow is critical, and addressing the fault tolerance of staging-based in-situ workflow frameworks is essential.

Unfortunately, traditional HPC fault tolerance techniques, such as Checkpoint/Restart and replication [11], can not be directly used to implement resilient data staging services. For example, using traditional Checkpoint/Restart, recovery from a failure in one of the data staging nodes would require all the components of the workflow to rollback, which can be very expensive and wasteful. Similarly, the process replication approach presented in [12], which can potentially provide resiliency for in-situ workflows and data staging, requires twice the amount of computing and storage resources and may not be feasible. A more traditional approach based on replication, where multiple copies of the data are maintained, is a viable alternative but can have a large storage overhead [13]. For example, if you want a system to tolerate up to two node failures, using replication results 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/decoding the data during writes and recovery from failures [14]. Resilience approaches based exclusively on replication or erasure coding would result in large storage

1530-2075/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPS.2018.00021

overhead or computation overhead respectively. This warrants an approach to data resilience that is capable of maintaining low storage overhead while guaranteeing high performance.

In this paper, 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. We also develop optimizations such as 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 deployed it on the Titan Cray XK7 production system at Oak Ridge National Laboratory (ORNL). Our experimental evaluations using synthetic workloads and the S3D combustion workflow demonstrate that CoREC efficiently maintains storage efficiency and low latency for various use cases and supports sustained performance and scalability in spite of frequent node failures.

The remainder of this paper is organized as follows. Section II presents the low-latency and high-efficiency CoREC approach to data resilience for staging-based in-situ workflows, and Section III describes the design of CoREC. In Section IV, we present the implementation and evaluation of CoREC. Finally, Section V presents related work and Section VI concludes the paper.

II. 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, our hybrid approach to in-staging data resilience, and present an online approach for data classification based on data access patterns, which underlies CoREC.

A. 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 RDMA) between components of an in-situ workflow. For example, DataSpaces [6], [15], 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 [5], these mechanisms are not directly applicable to in-situ workflows, and to the data staging service supporting the workflow and the data being staged by the workflow. In the Checkpoint/Restart approach, checkpoint data is 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 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 4 seconds (based on the discussion for S3D presented in [5]) for a total of 8 staging servers with varying staged data sizes, which resulted in 12 checkpoints for data sizes of 1G to 4G and 13 checkpoints for a 8G data size. 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 time spent to achieve fault tolerance for just the staging servers is $\sim 40\%$ of the total 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 paper, failure recovery using CoREC increases the total execution time of the workflow by up to 2.3%, 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, rolling back the data staging server can cause the components of the workflow to become out of sync. Since all of the components of the workflow need to rollback 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 [12] 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.

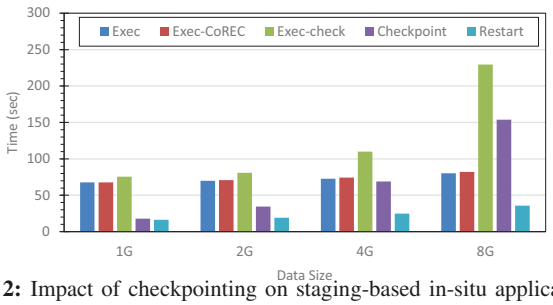
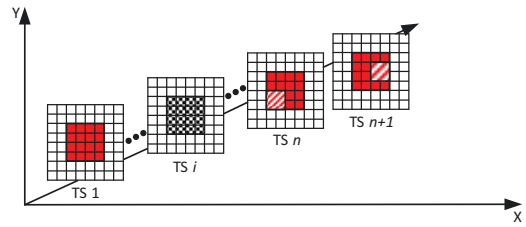
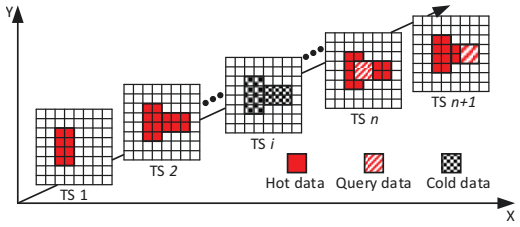


Fig. 2: Impact of checkpointing on staging-based in-situ application workflows. *Exec* is the total execution time of the workflow without checkpointing; *Exec-CoREC* is the total execution time of the workflow using CoREC; *Exec-check* is the total execution time of the workflow with periodic checkpointing 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..



(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.

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 is 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 recomputed for every object update. If a data object in a stripe is updated, 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 6 data objects and 2 parity objects, updating one data object requires 5 data object reads (for old data), recomputing 2 parity objects and 2 parity object writes. As a result, using erasure coding can be suboptimal for frequently written/updated data objects.

B. 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 is 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 [14].

C. 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 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 [8].

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 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 $n + 3$ [8].

We may go beyond this one step lookahead 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 $\{(4, 4), (7, 5)\}$, 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 sub-section II-A. 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.

D. 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, using replication for fault tolerance requires N_{level} copies of each object. Therefore, the storage efficiency, which is ratio of the size of original data objects to the size of original data object plus redundant data objects, for replication is:

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

Assuming that data is transferred between servers using a streaming approach and it take c seconds to transfer one object from the current server to the remote server. Further assuming that these servers have a 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 [16], 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:

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

*1) **Simple Hybrid Erasure Coding**:* In this paper, 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:

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

*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:

$$C_{CoREC} = P_h \times C_r \times f_h \times n + P_c \times C_e \times f_c \times n \quad (2)$$

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:

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

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

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

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

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

$$\begin{aligned} Gain &= C_{hybrid} - C_{CoREC} = \\ &(C_e - C_r) \times P_h \times P_c \times (f_h - f_c) \times n \end{aligned} \quad (6)$$

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

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

The prediction and classification of hot data objects depends upon the accuracy of the classifier. If the classifier is not accurate, 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:

$$C_{CoREC} = P_h(1-r_m)C_r f_h n + P_h r_m C_e f_h n + P_c C_e f_c n = (C_r f_h - C_e f_c + (C_e - C_r) f_n r_m) n P_h + C_e f_c n \quad (8)$$

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:

$$C_{CoREC} = P_r(1-r_m)C_r f_h n + (P_h - (1-r_m)P_r)C_e f_h n + P_c C_e f_c n = (f_h - f_c)C_e n P_h + C_e f_c n - (C_e - C_r)(1-r_m)P_r f_h n \quad (9)$$

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 is 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 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.

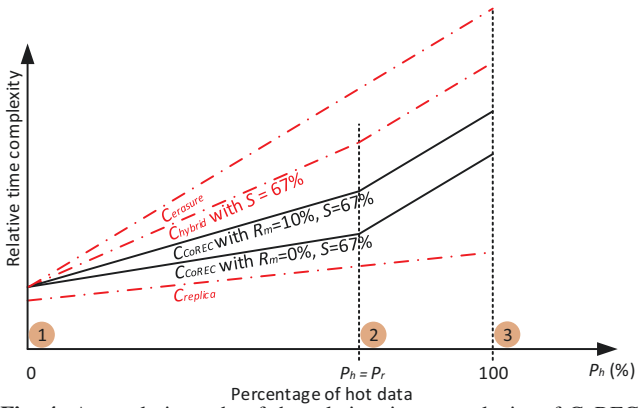


Fig. 4: An analytic study of the relative time complexity of CoREC (C_{CoREC}) with $RS(4,3)$, and varying miss ratios(R_m) and percentages of hot data objects (P_h). The time complexity for erasure coding ($C_{erasure}$), replication ($C_{replica}$) and simple hybrid erasure coding (C_{hybrid}) is noted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

III. 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.

A. Data Placement

1) Grouped Replication & Erasure Coding Scheme: In order to tolerate concurrent staging server failures, 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 ($k = 2, n = 3$) work in a twelve-servers data staging.

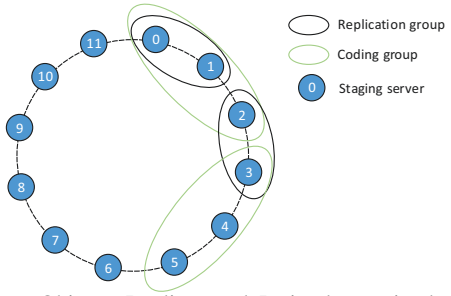


Fig. 5: Data Objects, Replicas and Parity layout in data staging. Servers 0 and 1 are in the same replication group while servers 0, 1, and 2 belong the same coding group. It is assumed that they are physically located in different cabinets.

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, as depicted in Figure 5. 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.

B. 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, 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.

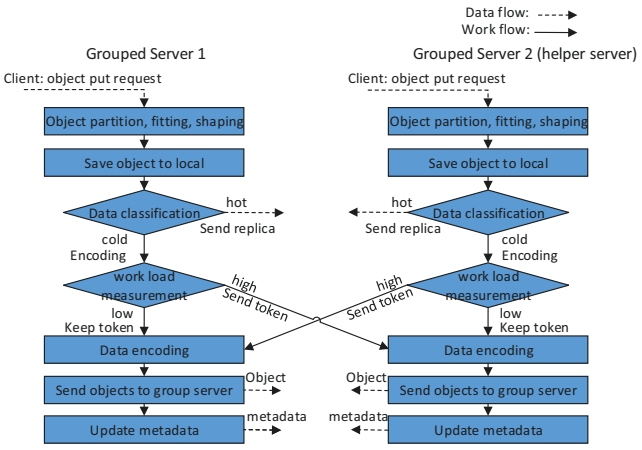


Fig. 6: Encoding workflow in CoREC.

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 make 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 comprises 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 objects. 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 operation than the busier one and workload is balanced throughout the coding group.

C. 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 [6]. However, large-sized data objects increase the processing time required for data encoding, decoding, replication and transportation [17]. This leads to longer data access latencies. Thus, an appropriate object size is required to balance metadata overhead and data access latency.

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 ($metadata[m]$);
$N \Leftarrow 1$
$object[m] \Leftarrow object$
while $N \neq 0$ **do**
  if $\exists$ obj in $object[m] > size$ **then**
    get maximum boundary size of obj in dimension n
    partition boundary to half
    partition obj to half
    $metadata[m] \Leftarrow metadata$
    $object[m] \Leftarrow obj$
  else {Object is fitting}
    return $object[m]$, $metadata[m]$
  end if
end while

In order 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.

D. Data Recovery

Existing large-scale resilient storage solutions typically use an aggressive recovery strategy [18]. Whenever a failure on

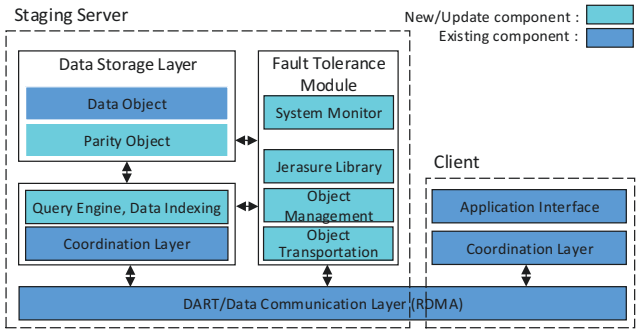


Fig. 7: System Architecture

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 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.

There are two modes in our recovery scheme: the degraded mode and lazy recovery mode. When a transient failure occurs and there is no replacement staging server, CoREC switches to the degraded mode. In this mode, only the requested data is re-constructed, sent to the client and discarded. The reconstruction of failed data objects in the read-path increases the read-latency. Experimental evaluation results for the reading performance in degraded mode are presented in Section IV.

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. On the other hand, 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.

IV. 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 [7].

CoREC is implemented on the top of DataSpaces [6], an open-source data staging framework. The schematic overview of the runtime system is presented in Figure 7. In addition to modifying several existing components of DataSpaces for the integration, the system architecture introduces three key new components: 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 [19] 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 workload monitoring, failure detection and recovery initiation is performed by the System Status Monitor component.

1) Synthetic Experiments: Our synthetic experiments were performed on the ORNL Titan Cray XK7 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 is written in multiple iterations (*time-steps*) as described in five test cases below. We compared our results with three other fault tolerance mechanisms: Replication (replicates all data objects and places replicas on remote staging servers), Erasure Coding (encodes all data objects locally and places data/parity objects on remote staging servers), and Hybrid Erasure Coding (without data classification – data objects are randomly selected for replication/erasure coding within a defined constraint on storage overhead). We additionally compared our results to the performance of data staging without any fault tolerance. In order to evaluate the balance between the write response time and the storage cost for various data resilience technique, 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 I. The experimental results are presented in Figure 8 and Figure 9, followed by a detailed discussion and analysis of each.

Total number of cores	$64 + 32 + 8 = 104$
No. of parallel writer cores	$4 \times 4 \times 4 = 64$
No. of staging cores	8
No. of parallel reader cores	32
Volume size	$256 \times 256 \times 256$
In-staging data size (20 TSs)	320MB
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%

TABLE I: Experimental setup for synthetic tests.

1) **Case 1** - *Write the entire data domain in each time step*: In this case, the data of the entire domain is written at every simulation time step. Since, there is no data repli-

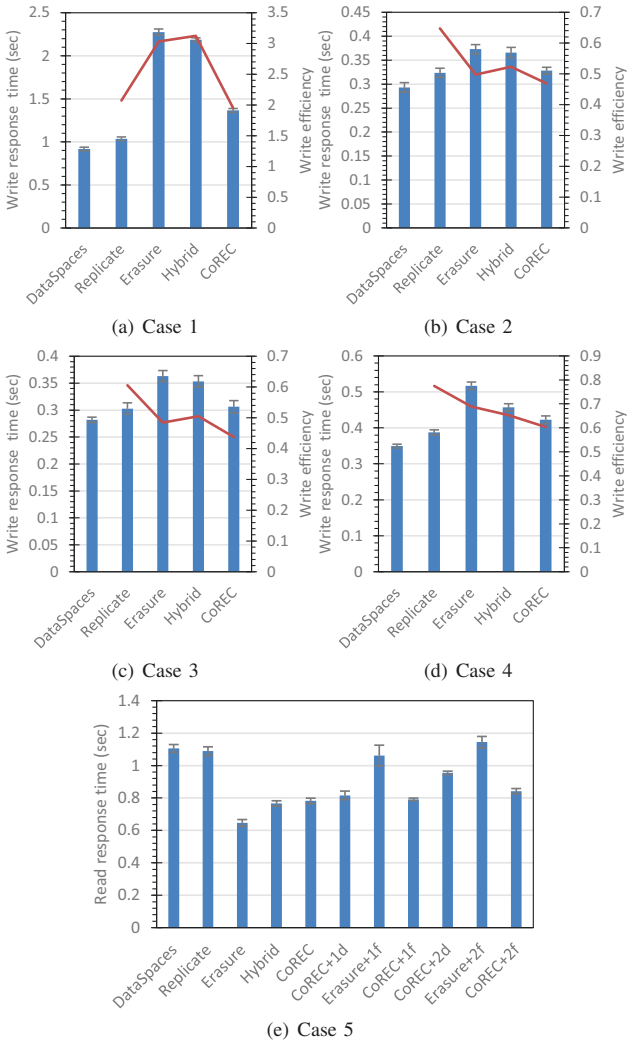


Fig. 8: Average data write and read response time *(blue bars)* and Write Efficiency = *Write response time/Storage Efficiency (red line)* of different data resilience mechanisms for the five test cases using different writing patterns. *DataSpaces:* Data staging without fault tolerance; *Replicate:* Data is replicated for resilience; *Erasure:* Data is erasure coded for resilience; *Hybrid:* Simple hybrid erasure coding without any 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.

cation, encoding, data movement and metadata synchronization overhead, the data staging without fault tolerance has the best relative data write response time. In our tests, the replication approach has smaller write response time and total workflow execution time in comparison to other fault tolerance approaches because it does not have the overhead of data encoding, which also leads to a smaller data transportation overhead. In contrast, the result for the erasure coding method shows the worst write access performance and the longest total workflow execution time because of the overhead associated with frequently encoding the original data objects and the placement of data/parity objects on remote servers. Although only a portion of data objects are erasure coded in hybrid erasure coding, frequently switching between replication and

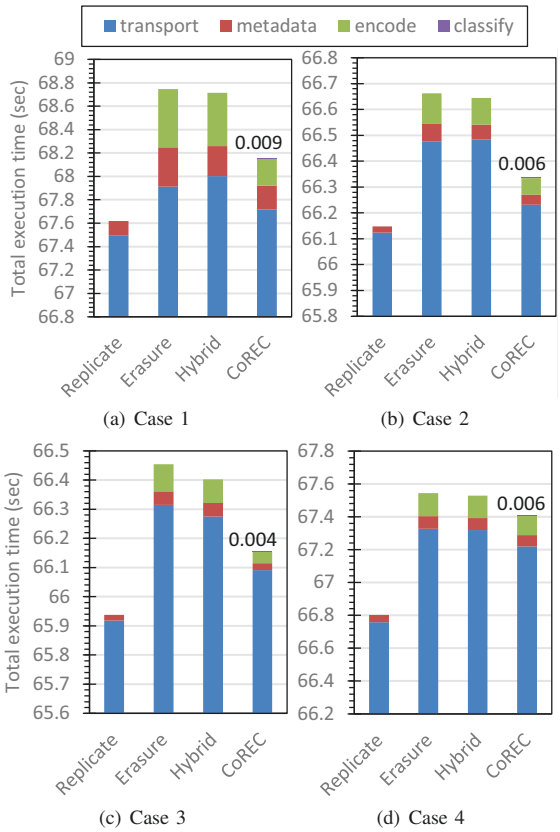


Fig. 9: Breakdown of the total execution time (in seconds) for the workflows in Figure 8. *transport:* Time spent in data movement; *metadata:* Time spent to update the distributed metadata; *encode:* Time spent to perform data encoding; *classify:* Time spent for data classification in CoREC (listed as number).

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 gets achieves a decrease of 48.7% and 53.2% in encoding time and an improvement of 35.2% and 38.7% in the write response time, relative to simple hybrid erasure coding and erasure coding. 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 31.7% 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 4 subdomains, and each subdomain is written in a time step. This means that in every 4 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 slightly better (around 6.51%) performance improvement for write response time and 38.4% decrease in the encoding time with respect to simple hybrid erasure coding, while incuring an overhead of 0.8% in the write response

time over replication. In addition, the conflict avoid encoding workflow and fewer data conversion from replication to erasure coding contributes to having smaller data transportation overhead than simple 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 is 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. CoREC can easily identify these hot data objects and apply the corresponding resiliency technique. Since erasure coding and simple hybrid erasure coding select either all data objects or randomly as candidates for erasure coding, CoREC improves the write response time by 7.2% and 9.6% and decreases encoding time by 50.4% and 56.5% respectively, while increasing the write response time by just 1.51% 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 3.14% and 13.4% and decrease encoding time by 14.6% and 17.8% compared to simple hybrid erasure coding and pure erasure coding respectively.

Figure 9 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 simple hybrid erasure coding and pure erasure coding in all cases. CoREC has less data transport time than erasure coding and simple 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 is read for every time step in this case. The replication-only method has a slightly better read response time than the original method because having multiple copies of the data at separate nodes/servers can increase data access bandwidth for concurrent data access requests. 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, simple 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 paper due to lack of space. These results show similar patterns as case 5. We also evaluated read response time in the presence of failures. In degraded mode, the read response time increases by 4.11% and 23.4% for single and double server failures respectively, as compared to failure-free case. However, when using lazy recovery, the read response time increases by 2.41% for single failure and 8.43% for double server failures as compared to failure-free case.

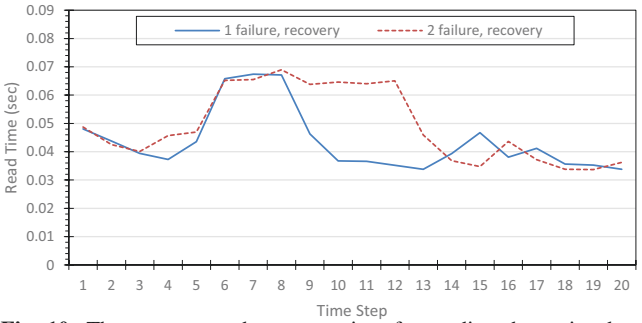


Fig. 10: The average read response time for reading the entire data domain with 1 and 2 failures, along with failure recovery, for 20 time steps. The first failure occurs at time step 4, and second failure occurs at time step 6. First failure-recovery begins at the 8^{th} time step and another recovery is initiated at the 12^{th} time step, and they end at time steps 9 and 13 respectively.

While we demonstrated that CoREC performs better on average than both erasure coding and simple 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 8. It can be seen that, data staging without fault-tolerance provides best performance along with best storage efficiency. On the other hand, 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.

In order to study the impact of lazy recovery in CoREC, we also plot the read response time at every time step for 20 time steps in Figure 10. 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 case, 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 step 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.

No. of cores	4480	8960	17920
No. of simulation cores	$16 \times 16 \times 16 = 4096$	$32 \times 16 \times 16 = 8448$	$32 \times 32 \times 16 = 16896$
No. of staging cores	256	512	1024
No. of analysis cores	128	256	512
Volume size	$1024 \times 1024 \times 1024$	$2048 \times 1024 \times 1024$	$2048 \times 2048 \times 1024$
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%

TABLE II: Configuration of core-allocations, data sizes, and data resilience for the three test scenarios on 4480, 8960 and 17920 cores.

2) Large Scale S3D Experiment: We also performed large-scale tests for CoREC using the lifted hydrogen combustion simulation workflow using S3D [7] and an analysis application on Titan, and compared it to pure replication and erasure codes. CoREC was tested using three different core count

(4480, 8960 and 17920) and corresponding grid domain sizes so that each core was assigned a spatial sub-domain of size $64 \times 64 \times 64$. 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 II.

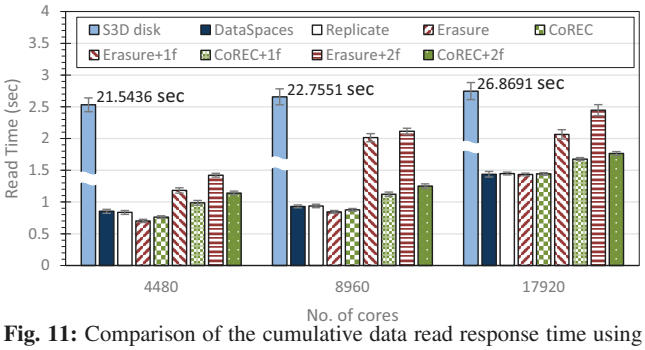


Fig. 11: Comparison of the cumulative data read response time using the S3D and coupled analysis workflow on Titan.

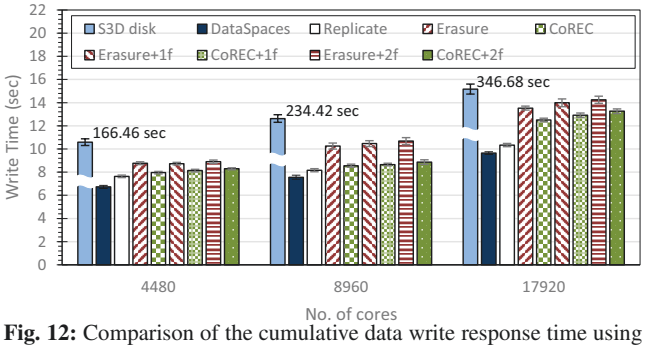


Fig. 12: Comparison of the cumulative data write response time using the S3D and coupled analysis workflow on Titan.

Figure 11 and Figure 12 illustrate the experimental results for the S3D coupled simulation and analysis application workflow, for various resiliency settings. Since the PFS (parallel filesystem based) based S3D does not have data staging and the data is 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 4480, 8960, and 17920 cores respectively. In comparison to replication, CoREC has an overhead of 4.2%, 5.3%, and 17.2% in write response time on 4480, 8960, and 17920 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.

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

V. 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 in-memory simulation data, and has been used for visualization, indexing building, data compression, statistical analysis [20] [21], etc. A number of data staging solutions such as DataSpaces [15] /ActiveSpaces [22], PreDatA [23] provide services for supporting in-situ and in-transit approaches, e.g., [8], with a primarily focus on fast and asynchronous data movement off simulation nodes. Unfortunately, these frameworks do not address the resilience of staged data, which is an important concern at extreme scales.

While supporting resilience in contexts other than in-situ/in-transit data analytics, such as Checkpointing [24], [25], [26], [27] and Replication/Erasure Coding [18], [28] has been widely studied, there are limited research efforts focussed on in-situ/in-transit data processing systems. The study in [29] 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 [30] use a synchronous two-phase commit transactions protocol to tolerate failures in high performance and distributed computing system. 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.

While aspects of CoREC may appear conceptually similar to Cocytus [13], 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 system, CoREC targets in-situ/in-transit data processing on large-scale HPC systems.

VI. 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 paper addresses data resiliency for staging-based in-situ/in-transit workflows. In this paper we presented CoREC, a scalable hybrid approach to data resilience for data staging frameworks that used online data access classification to effectively combines replication and erasure codes, and to balance computation and storage overheads. 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. 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 fault-tolerance. 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 a new models for data resilience that incorporate utility-based data placement across these layers.

VII. Acknowledgments

This worked 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 Energys National Nuclear Security Administration under contract DE-NA-0003525. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

References

[1] S. Ahern et al., "Scientific discovery at the exascale, a report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization," in *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*, 2011, pp. 1–4.

[2] E. F. D'Azevedo, J. Lang, P. H. Worley, S. A. Ethier, S.-H. Ku, and C. Chang, "Hybrid mpi/openmp/gpu parallelization of xgc1 fusion simulation code," in *Supercomputing Conference 2013*, 2013.

[3] S. Ku, C. Chang, and P. Diamond, "Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry," *Nuclear Fusion*, vol. 49, no. 11, p. 115021, 2009.

[4] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling in-situ execution of coupled scientific workflow on multi-core platform," in *Proc. 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS'12)*, 2012.

[5] M. Gamell, K. Teranishi, M. A. Heroux, J. Mayo, H. Kolla, J. Chen, and M. Parashar, "Local recovery and failure masking for stencil-based applications at extreme scales," in *High Performance Computing, Networking, Storage and Analysis (SC), 2015 International Conference for*, November 2015.

[6] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, Jun 2012.

[7] 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, "Terascale direct numerical simulations of turbulent combustion using s3d," *Computational Science & Discovery*, 2009.

[8] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–9.

[9] M. Parashar, "Addressing the petascale data challenge using in-situ analytics," in *Proceedings of the 2Nd International Workshop on Petascal Data Analytics: Challenges and Opportunities*, ser. PDAC '11. New York, NY, USA: ACM, 2011, pp. 35–36. [Online]. Available: http://doi.acm.org/10.1145/2110205.2110212

[10] F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," in *Supercomputing Frontiers and Innovations: an International Journal*, vol. 1, no. 1, 2014, pp. 5–28.

[11] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," in *The Journal of Supercomputing*, vol. 65(3), 2013, pp. 1302–1326.

[12] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for hpc," in *at the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[13] H. Zhang, M. Dong, and H. Chen, "Efficient and available in-memory kv-store with hybrid erasure coding and replication," in *the Fourteenth USENIX Conference on File and Storage Technologies (FAST) for*, February 2016.

[14] P. Subedi and X. He, "A comprehensive analysis of xor-based erasure codes tolerating 3 or more concurrent failures," in *Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013 IEEE 27th International Symposium on*, April 2013.

[15] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10, 2010, pp. 25–36.

[16] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," in *Journal of the Society for Industrial & Applied Mathematics*, vol. 8(2), 1960, p. 300.

[17] M. M. T. Yiu, H. H. W. Chan, and P. P. C. Lee, "Erasure coding for small objects in in-memory kv storage," in *in Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR)*, May 2017.

[18] A. Cidon, R. Stutsman, S. Rumble, S. Katti, J. Ousterhout, and M. Rosenblum, "Mincopysets: derandomizing replication in cloud storage," in *at the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[19] J. S.Plank, J.Luo, C. D.Schuman, L.Xu, and Z.-O. Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *the Seventh USENIX Conference on File and Storage Technologies (FAST)*, Dec 2009, pp. 263–272.

[20] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *IEEE Computer Graphics and Applications*, no. 3, pp. 45–57, 2010.

[21] J. C. Bennett, V. Krishnamoorthy, S. Liu, R. W. Grout, E. R. Hawkes, J. H. Chen, J. Shepherd, V. Pascucci, and P.-T. Bremer, "Feature-based statistical analysis of combustion simulation data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1822–1831.

[22] C. Docan, F. Zhang, T. Jin, H. Bui, Q. Sun, J. Cummings, N. Podhorszki, S. Klasky, and M. Parashar, "Activespaces: Exploring dynamic code deployment for extreme scale data processing," *Concurrency and Computation: Practice and Experience*, 2014.

[23] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata - preparatory data analytics on peta-scale machines," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.

[24] L. B. Gomez, D. Komatitsch, and N. Maruyama, "Fti: high performance fault tolerance interface for hybrid systems," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2012, pp. 728–740.

[25] L. Arturo, B. Gomez, N. Maruyama, and F. Cappello, "Distributed diskless checkpoint for large scale systems," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010, pp. 263–272.

[26] D. Vogt, C. Giuffrida, H. Bos, and A. S. Tanenbaum, "Techniques for efficient in-memory checkpointing," in *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*, Nov 2013, pp. 263–272.

[27] S. Gao, B. He, and J. Xu, "Real-time in-memory checkpointing for future hybrid memory systems," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, Nov 2015, pp. 263–272.

[28] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *in Proceedings of the 7th symposium on Operating Systems Design and Implementation OSDI'06*. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[29] J. Liu and G. Agrawal, "Supporting fault-tolerance in presence of in-situ analytics," in *2017 17th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2017, pp. 304–313.

[30] J. Lofstead, J. Dayaly, I. Jimenezz, and C. Maltzahn, "Efficient, failure resilient transactions for parallel and distributed computing," in *2014 International Workshop on Data Intensive Scalable Computing Systems*, November 2014, pp. 17–24.