2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

Scalable Crash Consistency for Staging-based In-situ Scientific Workflows

Shaohua Duan
Rutgers Discovery Informatics Institute
Rutgers University
Piscataway, NJ 08854, USA
shaohua.duan@rutgers.edu

Manish Parashar
Rutgers Discovery Informatics Institute
Rutgers University
Piscataway, NJ 08854, USA
parashar@ored.rutgers.edu

***Abstract*—As applications move towards extreme scales, data-related challenges are becoming significant concerns for scientific workflows, and in-situ/in-transit data processing have been proposed to address these challenges. However, increasing scales are expected to result in an increase in the rate of failures and the cost of resilience. Even worse, since coupled applications in workflows frequently interact and exchange a large amount of data, simply applying state of the art fault tolerance techniques to individual application components can not guarantee data consistency in workflows after failure recovery. Furthermore, naive use of fault tolerance techniques, such as checkpoint/restart, to the entire workflows prohibits the diversity of resilience of application components in workflows, and finally incurs a significant latency, storage overheads, and performance degradation. This paper addressed fault tolerance challenge for extreme scale in-situ scientific workflows. We present a loose coupled checkpoint/restart framework for in-situ workflows. This proposed approach provides a scalable and flexible fault tolerance scheme for in-situ workflows while still maintaining the data consistency and low resiliency cost. Specifically, we introduce a data logging mechanism in data staging which is composed by the queue based algorithm and user interface to keep data/events consistent during failure recovery. We have implemented our approach within the DataSpaces, an open-source data staging middleware, and evaluated it using synthetic workflows on a Cray XC40 system (Cori) at different scales. We demonstrated that, in the presence of failures, uncoordinated checkpoint and hybrid checkpoint with data logging scheme improved the workflow execution time by up to 13.48% in comparison with global coordinated checkpoint/restart approach.**

***Keywords*-crash consistency; checkpointing; in-situ workflows; data staging;**

I. Introduction

Scientific workflows running on emerging extreme-scale system can provide new insights into some of the most important problems in science and society, such as those being addressed by the US Exascale Computing Program (ECP) [1]. However, extreme scales are also leading new challenges, such as data management, resilience, energy efficiency, etc., and research and innovations at all levels from the hardware architecture, the system software, the algorithms, and applications.

In-situ workflow approaches, such as those based on data staging and in-situ/in-transit data-management have emerged as effective solutions to address data-related challenges

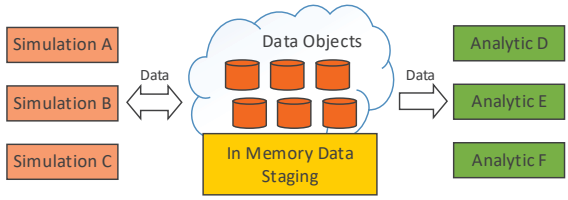


Figure 1: A typical data staging based workflow.

at extreme scales and are being adopted by application workflows across current high-end computing systems [2], [3]. These techniques leverage resources (i.e., compute and memory) on the HPC system itself to create a data staging area and use it to support the interaction and data couplings required by the workflows as well as to execute data-processing components of the workflow close to where the data is being produced, reducing the amount of data that needs to be moved off the system, for example, to a persistent storage [4]. For example, the multi-scale, multi-physics turbulent combustion application S3D [5] has an intricate data-processing workflow with multiple analyses performed at different temporal frequencies on non-overlapping subsets of data, and staging-based in-situ frameworks such as DataSpaces [6] have been effectively used to support these workflow requirements. Figure 1 illustrates a coupled simulation workflow, wherein the primary scientific simulation is the data producer and secondary simulations, analytics, and/or visualization applications serve as data consumers coupled to the producer.

However, workflows on extreme-scale system are also expected to exhibit much higher fault rates than current one does, for various reasons relating to both hardware and software. For extreme-scale system, the performance increases require a commensurate increase in the number of hardware components. If we expect the per-component fault rates in an extreme scale system to be similar to those observed in current devices, a system 1,000 times more powerful will have at least 1,000 times more components and will fail 1,000 times more frequently. Similarly, software architecture of workflows on extreme-scale system will also become more complex and hence more failure-prone. Multiphysics and multiscale codes couple an increasingly large number of distinct modules. Data visualization, simulation, and

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00068

analysis are coupled into increasingly complex workflows. Furthermore, the need to reduce communication between modules, allow asynchrony and memory hierarchy results in more complex software architecture in workflows. Although the mean time between failures (MTBF) for current systems is measured in hours (e.g., 14.51 hours for Titan Cray XK7 as shown in [7]), but it is estimated that the MTBF for an exascale system would be measured in minutes [8].

In order to address fault tolerance at extreme scales with the expected higher rates of failures [9], recent research has explored techniques for minimizing application vulnerability to failures. Various fault tolerance techniques such as checkpoint/restart [10], process replication [11], and algorithm-based fault tolerance (ABFT) [12] have been provided and widely studied.

Unfortunately, applying these techniques to each application components separately can not lead to the resiliency of in-situ workflows, which are a composition of multiple interacting applications. Due to the dependencies, interactions and data exchanges between application components, this naive uncoordinated fault tolerance scheme can not maintain the consistent state of workflows during the recovering phase, which finally makes the result of workflows invalid or incorrect. Meanwhile, directly using a single fault tolerance technique, such as global coordinated Checkpoint/Restart, to all application components to achieve data consistency can make the complexity and overhead for addressing failures to be unacceptably high. Furthermore, the uniform fault tolerance strategy restricts the diversity requirement for addressing application resiliency, and potentially increase the overall resiliency cost. As a result, it is important to design a loose coupled workflow-level fault tolerance mechanism to minimize the interference between components during recovery, while still maintain consistent states of workflows.

This paper explores the workflow-level checkpoint/restart strategy for in-situ workflows. We employ a data/event logging mechanism to keep data consistency among application components during the failure recovery while decouple fault tolerance schemes between application components in workflows. Specifically, we perform data/event logging as soon as the data is written or read through the staging area. Also, we introduce global user interface to application components, which works with the queue based algorithm to record and replay data access events when preforming checkpointing and rollback recovery. In this way, our checkpoint/restart with data logging framework allows wide area fault tolerance schemes to be applied in workflows with flexibility and scalability, and minimize the interfere between normal application components and the failed one when performing the recovery strategy.

We have implemented a checkpoint/restart with data logging framework within DataSpaces [6] and have deployed it on Cori, a Cray XC40 production system at Lawrence Berkeley National Laboratory (LBNL). We then have evaluated its effectiveness and performance using synthetic workloads. These evaluations demonstrate that our uncoordinated/hybrid checkpoint/restart with data logging approaches efficiently recover workflows in various use cases and scales, sustaining performance and scalability in spite of frequent failures.

The rest of the paper is organized as follows. In Section II, we discuss background and motivation related to the workflow-level checkpoint framework as well as state of the art fault tolerance approaches for individual applications, and introduce our checkpoint/restart with data logging framework in Section III. In Section IV, we evaluate our approach using various synthetic scientific workflows. Section V provides details of various related work, and we conclude the paper in Section VI.

II. Background and Motivation

In this section, we first explore fault tolerance requirements for in-situ workflows and investigate why traditional mechanisms, such as global coordinated checkpoint/restart, are unable to meet these requirements effectively.

A. Fault Tolerance in Workflows

Extreme scale in-situ scientific workflows involve interactions and data exchanges between the various components composing the workflow. For example, consider the coupled combustion simulation DNS-LES workflow. S3D [5] is a massively parallel computational fluid dynamics (CFD) solver that performs first principles based "direct numerical simulations" (DNS) of turbulent combustion. DNS is very expensive both in terms of flops and data generation, since it resolves the entire range of spatial and temporal scales in the continuum regime of a given problem. Large Eddy Simulation (LES) simulates a combustion environments at a lower resolution, resulting in better performance for simulating features that can tolerate this lower resolution. Many problem classes involve features that cannot be performantly and accurately simulated by DNS or LES alone, but can be effectively simulated by a coupled solution, requiring in-situ coupling of two different solvers running at different resolutions. Furthermore, knowledge discovery and visualization from a S3D simulation run can be a daunting task due to the size of the data set generated and the complexity of the temporally evolving intermittent phenomena. To understand the correlation of scalar fields such as temperature, mixing rates and species concentrations in turbulent flames, simulation-time feature extraction and visualization is necessary.

Meanwhile, there is a large body of work on scalable fault-tolerance mechanisms to individual applications. These techniques range from checkpoint/restart (C/R), process replication to ABFT. They provide fault tolerance for a single application successfully. For example, checkpoint/restart (C/R) is the most widely used general-purpose fault-tolerant technique in high performance applications. The principle of strategy for (C/R) is: checkpoints are periodically saved

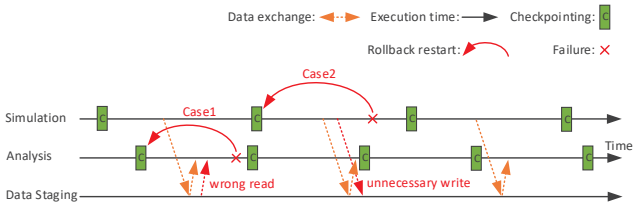


Figure 2: individually checkpoint/restart for the in-situ S3D coupled simulation workflow.

during application initial execution, and when processes are subject to failures, it uses these checkpoints to rollback the application processes to the latest consistent state, and re-execute the program from that point. Unfortunately, applying fault tolerance techniques to the application components individually does not effectually address the resiliency challenge for the whole workflows due to the data coupling at extreme scale. For example, the S3D workflow consists of S3D simulation and visualization coupling application, as illustrated in Figure 2. In each coupling cycle, the workflow first executes S3D simulation for several time steps, and generates the coupling data, which is processed by the analytics/visualization applications for feature extraction. When the workflow passes the coupling data between the S3D simulation components and visualization components, dozens of 3D scalar and vector field components (fluid velocity, molecular species concentrations, temperature, pressure, density, etc) are transferred using staging area. In this scenario, if the S3D simulation and analytics are protected by checkpoint/restart individually, and a failure happens in the analytics, the re-executive analytics process will get the wrong version of data from data staging considering the S3D simulation has updated the data during the analytics re-execution. (the case1 shown in Figure 2). Similarly, if a failure hits the S3D simulation, the re-executive simulation will unnecessarily perform the data updating operation to data staging twice, considering data has already been staged in staging in first execution (the case2 shown in Figure 2). These data/events inconsistency issue finally result in erroneous results of visualization and analysis products. Even worse, in the case of multiphysics coupling even small errors can have catastrophic results, especially to the stability of non-linear PDEs. These errors have the potential to significantly alter the accuracy of the simulation, and make the final workflow's result invalid.

A further challenge to fault tolerance of entire workflows is application components in workflows exhibit different resiliency requirements, program properties and failure characters, which result in diversification of fault tolerance strategy among these components. For example, the stencil-based application which is commonly used as scientific simulations, such as the S3D simulation, employs local recovery strategy [13] to effectively reduce the overhead of recovery in case of frequent process failures. This recovery strategy is based on unique communication pattern of stencil-based applications which implies that multiple independent failures can be masked to effectively reduce the impact on the total time to solution. Meanwhile, as another large group of applications, the linear algebra applications typically utilize algorithm-based fault tolerance (ABFT) [12] to tolerate both fail-stop failures and silent errors. In ABFT, the application uses intricate knowledge of linear algebra to maintain supplementary, redundant data, and can be updated algorithmically to form a recovery dataset in case of failure. Although local recovery and ABFT can exhibit excellent performance and resiliency for the single application, they are less generalist approaches, and can not be appropriate for all application components in workflows. Therefore, when such types of applications with the specific fault tolerance strategies combine together into a in-situ workflow, to enable these resilience techniques working cooperatively is chanllengeable, and constructing a workflow-level loose coupled fault tolerance framework become necessary. Ideally, workflow-level fault tolerance mechanism should enable individual application components to exploit the wide area of fault tolerance techniques.

Unfortunately, naive using state of the art fault tolerance approaches in workflows can not address those challenges discussed above efficiently. One possible solution to fault tolerance of in-situ workflows is using global coordinated checkpoint/restart protocols shown as Figure 3. This method requires that all processes in workflow rollback to the last valid checkpoint place, when a failure occurs. In the case of the Message Passing Interface (MPI), a very simple approach have often been taken to ensure the consistency of the snapshot: a couple of synchronizing MPI barriers can be used, before and after taking the process checkpoints, to guarantee that no application in-flight messages are present at the time of triggering the checkpoint, and thus the causal ordering of communications inside the application is avoided entirely. Global coordinated checkpoint/restart ensures a global state consistency, but it presents two concerns. As the size of the system grows, the probability of failures increase, and the minimal cost to handle such failures also increase due to frequently rollback whole workflow for recovery. Even worse, rollback other healthy components which have no data coupling with the failed component during failures time and last checkpoint time, shown as Figure 3, are unnecessary and wasteful. The second is global coordinated checkpoint/restart constrain the diversity of individual application fault tolerance strategies which can efficiently reduce the resiliency cost.

Ideally, a good candidate approach to the workflow-level fault tolerance should maintain the data consistency between coupled application components efficiently, meanwhile provide a compatibility with diverse state of the art fault tolerance approaches to construct a workflow-level loose coupled fault tolerance mechanism for the entire workflows.

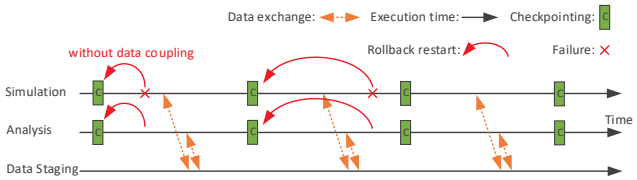


Figure 3: Coordinated checkpoint/restart for the entire in-situ S3D coupled simulation workflow.

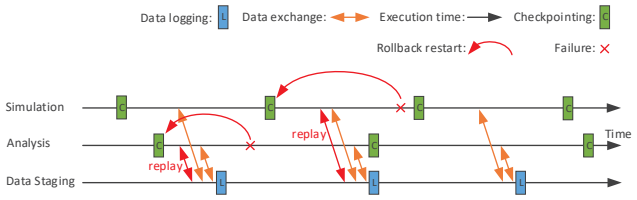


Figure 4: An illustration of uncoordinated checkpointing for a typical workflow with simulation, Analytic.

III. WORKFLOW-LEVEL CHECKPOINTING FRAMEWORK

In this section, we first propose our workflow-level uncoordinated checkpoint framework which is integrated with multiple checkpoint/restart strategies. We then extend the framework with other fault tolerance strategies such as process replication and ABFT, and construct a hybrid checkpoint approach. Finally, we design the global user interface for the framework which is implemented in open source data staging, DataSpaces.

A. Uncoordinated Checkpointing

To mitigate data inconsistency issue discussed in Section II, we introduce workflow-level uncoordinated checkpoint with data logging framework. In this framework, the workflow logs data transportation events and payload between application components as it proceeds along the initial execution; without strong coordination, application components in workflows checkpoint their state independently. In case of application component failure, the workflow collects all its data/event log history, and enters the replay mode. Replay consists in following the log history, enforcing all data transportation events of the failed component to produce the same effect they had during the initial execution, and the corresponding data is re-provided to this process for this purpose. Therefore, the data dependency and consistency between coupled application components will keep during the replay mode. Once the history has been entirely replayed, the application component reaches a state that is compatible with the state of the other components in workflows, that can continue its progress from this point on.

1) Data Logging in Staging: One way to implement data logging mechanism for workflows is to perform data logging in a data resilience staging area. Figure 4 shows a typical workflow with a uncoordinated checkpoint scheme combined with data staging. In this workflow, application components send the data communication requests to data staging. For data write requests, applications offload the

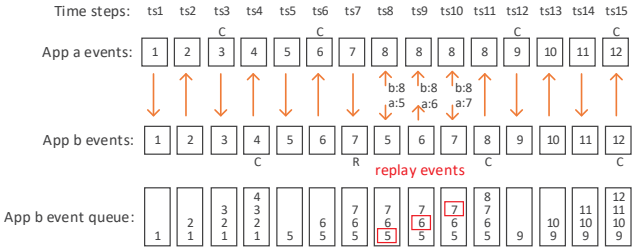


Figure 5: An illustration of queue based data consistency algorithm for a coupled applications workflow. Simulation b fails and performs rollback recovery at time step 7, then during time step 8 to 10, staging area relays the events in the queue for the simulation b which are recorded from time step 5 to 7.

data to staging area for data transportation to coupled components later. For data read requests, applications receive the data from staging area which are generated by coupled components in early time. Data staging logs the data communication requests and corresponding payload, and records the fault tolerance events such as chechpointing and failure recovery during the initial execution. In case of failure, data staging switches to the recovery phrase. It cooperates with the application recovery scheme, and reproduces the data communication requests for the recovered component, which are determined by the log history in the initial execution. To guarantee the data availability in staging, the data staging can contain data resilience mechanism such as data replication or erasure coding. It can also be integrated with the third part framework such as FTI [14] for data resilience.

Specifically, we employ a queue based data consistency algorithm in staging area. Figure 5 illustrates a queue based data consistency algorithm for a coupled applications workflow. The workflow consists of two coupled applications, and in each coupling cycle (1 time step), coupled simulations a, b exchange the data through data staging. The data staging creates a event queue for each application, and pushes the data communication request events which related with the application into queue. In case of failure, staging area will replay the events from the queue during the application recovery phase. In this example, simulation b failed and performs a rollback recovery at time step 7, and during time step 8 to 10, staging area relays the events in the queue for the simulation b which are recorded from time step 5 to 7. At the end of checkpoint cycle($ts4$, $ts9$, $ts12$), data staging will clean the event queue and reload the following event from the front of queue. By maintaining the data request event queue, data staging can keep data consistency between coupled applications during failure recoveries.

As well as uncoordinated checkpoint with multiple checkpoint periods shown in Figure 5, this data logging mechanism can easily adapt to other checkpoint/restart strategies such as proactive checkpointing [15] and multi-level checkpointing [16] under minor changes.

2) Storage Cost and Garbage Collection: To reducing the storage cost, a garbage collecting mechanism is provided in

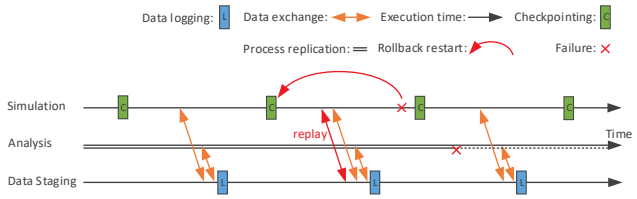


Figure 6: An illustration of hybrid checkpoint (integrated with a process replication) for a typical workflow with simulation, analytic.

workflow_check()	send a checkpoint event to data staging.
workflow_restart()	recover data staging client and notify the recovery event to data staging.
dspaces_put_with_log()	log data to data staging.
dspaces_get_with_log()	retrieve the logged data specified by geometric descriptor from data staging.

Table I: User interface for checkpoint/restart in workflows.

staging area. Specifically, data staging servers periodically delete logged data which are related with previous checkpoint periods without data dependency to other application components, and only keep the latest version of data in staging area.

B. Hybrid Checkpointing

Beside checkpoint/restart approaches, this workflow-level framework can also support the wide area fault tolerance mechanism, and construct a hybrid checkpointing scheme for the workflow resiliency. Figure 6 illustrates a hybrid checkpoint framework which integrated with process replication and checkpoint/restart approaches. In this example, a simulation employs checkpoint/restart approach meanwhile the analytic uses process replication for resiliency. To make checkpoint/restart collaborate with a replication approach, during checkpointing periods, the data communication requests between these coupled applications will be logged in staging area. If a failure happens in the application with checkpoint/restart, the application will rollback to last checkpoint place and re-execute through the latest checkpoint, and the data staging switches to recovery phase. The data logging in the data staging guarantees the restarted application can always get the correct version of data from the coupled application. Since the application with process replication can tolerate failures without rollback recovery, the failures will not trigger the data staging to switch to recovery phase and replay the events.

C. Global User Interface

In this section, we describe more details about global user interface, and present an example to illustrate how this interface can be used to integrate multiple application fault tolerance approaches to a workflow-level fault tolerance scheme.

As shown in Figure 7(a), When the application component performs checkpointing, it first saves process states and user-level data to the reliable storage devices. The checkpoints can be stored through a centralized parallel file system, assumed to be fault-free. Other options include storing the

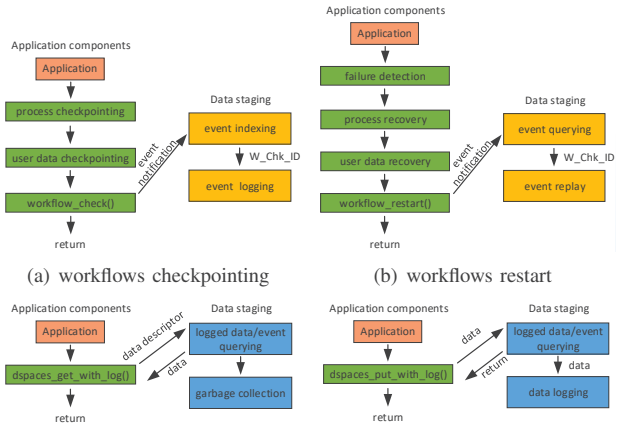


Figure 7: User interface for checkpoint/restart in workflows.

checkpoints in the node-local storage (such as NVRAM and SSD) or bust-buffer if the hardware architecture provides these devices. After that, the application component calls **workflow_check()** function, and notices a checkpoint event to data staging. When data staging receives the event notification, it creates a checkpointing ID: W_Chk_ID for this event, and then inserts it into the event queue. Since application components may have different checkpoint time spots, we assign an unique W_Chk_ID for each checkpoint event which refer to the same application component.

For the application recovery, it involves four key steps shown in Figure 7(b): failure detection, process recovery, data recovery, and data staging client recovery with event notification. To enable to recover applications from failures, application components will delete failed processes and recover the MPI communicator through ULFM [17] [18]; a proposed extension of the MPI standard which includes mechanisms for MPI applications to tolerate fail-stop failures. After that, the equal number of spare processes join the old communicator to construct the new one. An alternative approach is to spawn new processes instead of using processes from a previously prepared process pool, if this is supported by the job scheduler. After the failed application component recovered from the latest checkpoint spot, it calls **workflow_restart()** function. This function firstly initializes the data staging client, and tries to build RDMA connection to data staging servers, and send the recovery event notification to the servers. After receiving the notification, the data staging servers will update the event queue, and generate a replay script for the recovered application.

During a workflow execution, when an application component tries to read the coupled data from data staging, it calls **dspaces_get_with_log()** function. This function sends data read request with the data descriptor to data staging. Based on the log history in the event queue, the data staging identifies whether the read request comes from a rollback execution or an initial execution of applications, and return

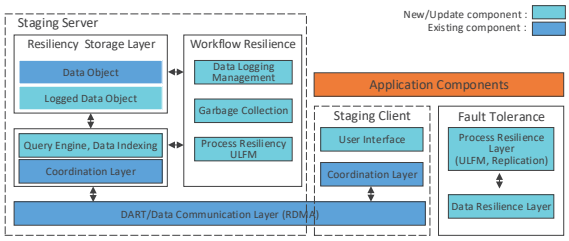


Figure 8: Implementation of workflow-level checkpoint framework

the correct version of logged data. Finally, data staging performs garbage collection operation to erase those data which are no longer in use.

Similarly, when an application component tries to write the coupled data into data staging, it calls **dspaces_put_with_log()** function, and sends the data with the data descriptor to data staging. After receiving the request, data staging will query the events in the event queue. If the request is from initial execution, data staging will store data as the logged data otherwise omit the write request due to the redundant write request from the rollback recovering application. Table I summarize the global user interface for application components to perform checkpoint/restart and data communication.

IV. Experimental Evaluation

In this section, we describe the implementation details of workflow-level checkpoint framework and perform an experimental evaluation using synthetic benchmarks in the presence of failures.

Our workflow-level checkpoint framework is implemented on the top of CoREC [19], an open-source data staging with scalable data resilience. CoREC is a branch version of DataSpaces [6], and provides data resilience for staging area in the case of both process and node failures while still maintaining low latency and sustaining high overall storage efficiency at large scales. The schematic overview of the runtime system is presented in Figure 8. In addition to modifying several existing components of CoREC for the integration, the system architecture introduces four key new components: Data Logging Component, Garbage Collection Component, Global User Interface, and Process/Data Resilience Component. The Data Logging Component stores, indexes and maintains the log data from coupled applications. Garbage Collection Component regularly cleans the unused historical log data. User Interface provides a set of checkpoint/restart and data logging interfaces for the applications in workflows. The Process/Data Resilience Component manages recovering applications from failures. This Component manages a spare process pool and implements the detection and handling of the process failures using ULFM, which offers a set of fault tolerance mechanisms for MPI applications.

Total No. of cores	$256 + 64 + 32 = 352$
No. of simulation cores	$8 \times 8 \times 4 = 256$
No. of staging cores	32
No. of analytic cores	64
Volume size	$512 \times 512 \times 256$
Data size (40 ts)	$20GB$
Data access pattern	write immediately followed by read
Coordinated checkpoint period (ts)	4
Simulation checkpoint period (ts)	4
Analytic checkpoint period (ts)	5

Table II: Experimental setup for synthetic test cases.

A. Synthetic Experiments

Our synthetic experiments were performed on the NERSC Cori Cray XC40 system, and evaluated the write performance and memory usage of data staging with data/event logging. We also measured the total workflow execution time to evaluate the benefit from workflow-level checkpoint/restart framework in case of failures. To better understand its performance and effectiveness, we selected two test cases with common data access patterns and resilience schemes used by real scientific workflows.

In each case, the simulation wrote the coupled data into the data staging, and the analytic read the data right after simulation write. Checkpoint/restart and/or process replication method were applied to the individual application to construct either uncoordinated checkpoint or hybrid checkpoint scheme for the entire synthetic workflows. In the synthetic workflows, simulation and analytic applications have different scales and resiliency requirements which correspond to checkpointing data to the parallel file system with different frequencies. For the process replication method, we use process duplication to tolerance one process failure. In case of failures, the application tolerated by checkpoint/restart will be rolled back to the last checkpoint place and re-executed from that point. For the application with replication scheme, it will be tolerated failures by switching the task from the failed process to the replicated process. We ran the synthetic workflow with different data access pattern and various frequency of checkpointing. Then, we measure write response time and memory usage of data staging and the total execution time of workflows. In these experiments, a failure was randomly introduced into the application process within 40 time steps, which corresponds to $MTBF = 10min$. This simulates frequent failures on an extreme-scale supercomputer system. We compared our approach with two other fault tolerance mechanisms: global coordinated checkpoint (checkpoint application components in workflows coordinately, and restart them globally) as a baseline and individual checkpoint (individually checkpoint/restart application components without guarantee of correctness results) as the theoretical optimal lower bound. All experiments ran on the Cori, Cray XC40 system, a 12,076-node supercomputer located at the NERSC center at Lawrence Berkeley National Laboratory (LBNL). The set-up of these experiments is described in Table II. The experimental results are presented in Figure 9 followed by a detailed discussion and analysis of each.

Total No. of cores	704	1408	2816	5632	11264
No. of simulation cores	512	1024	2048	4096	8192
No. of staging cores	64	128	256	512	1024
No. of analytic cores	128	256	512	1024	2048
Data size (40 ts)(GB)	40	80	160	320	640
Coordinated checkpoint period (ts)	8	8	8	8	8
Simulation checkpoint period (ts)	8	8	8	8	8
Analytic checkpoint period (ts)	10	10	10	10	10
$MTBF$(sec) / No. of failures	600 / 1, 300 / 2, 200 / 3				

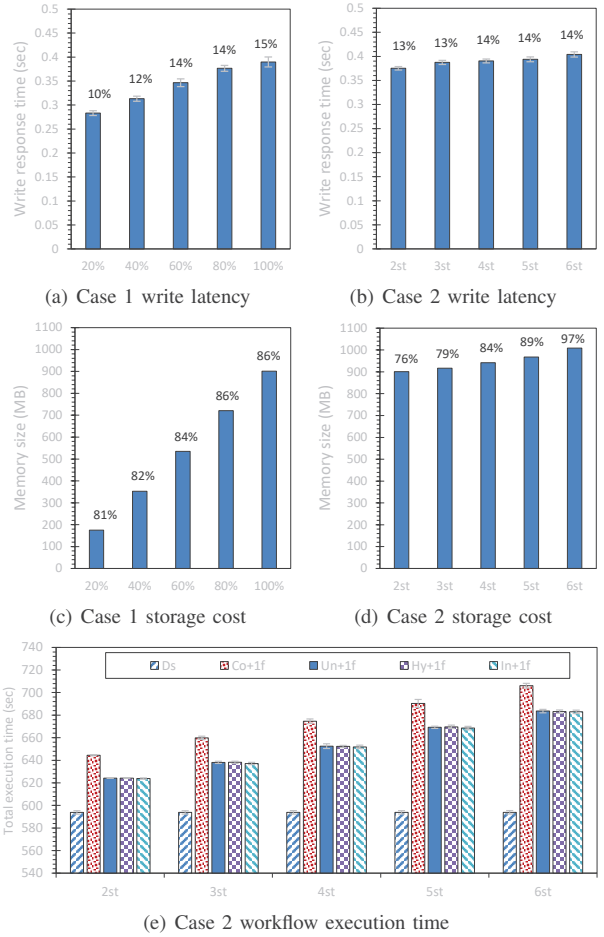
Table III: Configuration of core-allocations, data sizes, and failure characters for the scalability test scenarios on 704, 1408, 2816, 5632, and 11264 cores.

1) **Case 1** - *Write different subsets of the entire data domain in each time step*: In this case, 20%, 40%, 60%, 80%, 100% percentages of the entire data domain are exchanged between application components via data staging in each time step. Meanwhile, we perform checkpointing every 4 iterations for large simulation application, and 5 iterations for the small analytic application. With more data exchanging through data staging, both computation cost and storage cost for data/event logging increased. Data/event logging increased the write response time by 10%, 12%, 14%, 14%, and 15% as compared to the original data staging respectively. For the storage overhead of data/event logging in data staging, as shown in Figure 9(c), data/event logging increased the memory usage by 81% for 20%, 82% for 40%, 84% for 60%, 86% for 80%, and 86% for 100% subset, as compared to the original data staging's.

2) **Case 2** - *Write the entire data domain and perform checkpointing with different frequencies*: In this case, we write the entire data domain in data staging, and change the checkpoinnting periods from every 2 time steps to every 6 time steps. As seen in Figure 9(b), we got slight performance degradation when performing data/event logging in data staging. Data/event logging increased the write response time by maximum 14% as compared to original data staging under five different checkpoint frequencies. Since the less frequent checkpoint indicates the longer data/event queue size in staging area, the higher storage cost can be expected. Therefore, as shown in Figure 9(d), the memory usage for data logging increases by 76% for $2ts$, 79% for $3ts$, 84% for $4ts$, 89% for $5ts$, and 97% for $6ts$ checkpoint period, as compared to the memory usage of original data staging.

In these two cases, both uncoordinated checkpoint and hybrid checkpoint achieved nearly same execution time as individual checkpoint's which is theoretical optimal lower bound for the execution time of workflows with fault tolerance. Also, they achieve a decrease of 3.06% and 3.05% in the total execution time relative to global coordinated checkpoint in case 1. In case 2, as seen in Figure 9(e), we get similar performance improvement with case 1. The uncoordinated checkpoint and hybrid checkpoint reduce the total execution time around 3.15% for $2st$, 3.28% for $3st$, 3.26% for $4st$, 3.05% for $5st$ and 3.18% for $6st$ relative to global coordinated checkpoint respectively.

In order to study the scalability of workflow-level uncoordinated/hybrid checkpoint, we also plot the total workflow execution time at different workflow scales and MTBF.



(a) Case 1 write latency (b) Case 2 write latency

(c) Case 1 storage cost (d) Case 2 storage cost

(e) Case 2 workflow execution time

Figure 9: Comparison of the cumulative data write response time, storage cost, and total workflow execution time using the synthetic workflow on Cori. *Ds:* The workflow with original data staging and failure free; *Co:* Global coordinated checkpoint/restart; *Un:* Uncoordinated checkpoint/restart; *Hy:* Hybrid checkpoint/restart with process replication; *In:* Individual checkpoint/restart; *+1f:* with one synthetic process failure. Percentages on top of the bars indicate the ratio of memory usage of data logging to the original data staging's, and the ratio of write response time delay of data staging with data logging to the original ones.

The setup of these experiments is described in Table III. Figure 10 summarizes the total workflow execution time in case of different numbers of failures (from 1 to 3) and scales (704, 1408, 2816, 5632, and 11264 cores). It can be seen that in the presence of multiple failures, workflow-level uncoordinated checkpoint reduced the total execution time by up to 7.89%, 10.48%, 11.5%, 12.03%, and 13.48% on 704, 1408, 2816, 5632, and 11264 cores scales in comparison to global coordinated checkpoint.

These results show that workflow-level checkpoint framework demonstrates good overall scalability and flexibility with small storage overheads for different data coupling pattern, fault tolerance schemes and processor counts.

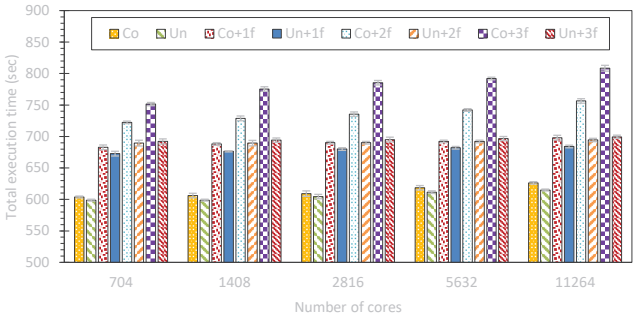


Figure 10: Summary of the total workflow execution time in case of failures (1, 2, and 3) and at different scales (704, 1408, 2816, 5632, and 11264 cores).

V. Related Work

The most commonly deployed strategy to cope with application failures is checkpointing, in which processes periodically save their state, so that computation can be resumed from that point when some failure disrupts the execution. Checkpointing strategies are numerous, and rang from fully coordinated checkpointing [20], uncoordinated checkpointing with message logging [21], multilevel checkpointing [16], on-line checkpointing with local recovery [13] to proactive checkpointing [15]. Simultaneously, other techniques such as process replication or redundancy techniques [22] and algorithm-based fault tolerance (ABFT) [12] also introduce fault tolerance mechanisms to scientific applications. In contrast to these efforts, we employ the data logging mechanism for tight coupled in-situ workflows to cooperate with these techniques together, and provide fault tolerance for application components in workflows effectually and efficiently.

Although In-situ processing paradigms and data staging techniques [6] [23] [24] can effectually address the data related challenge in scientific workflow, there are limited research efforts focused on fault tolerance in scientific in-situ workflows. The study in [25] exploits the reduction style processing pattern in analytic applications and reduces the complications of keeping checkpoints of the simulation and the analytic consistent. Research efforts in [26] use a synchronous two-phase commit transactions protocol to tolerate failures in high performance and distributed computing system. In comparison to these efforts, our checkpoint/restart with data logging approach specifically targets tight coupled in-situ workflows, and is more flexible, asynchronous and scalable.

VI. Conclusion and Future Work

Cutting-edge in-situ workflows are applied at an ever-growing scale. Fault tolerance is an important issue that needs to be addressed in order to allow these workflows to continue scaling efficiently. In this paper we have presented a checkpoint/restart with data logging framework for tight coupled in-situ workflows to enable diverse fault tolerance schemes to be used in workflows effectively and efficiently. Specifically, we apply data logging in staging area to effectively decouple fault tolerance schemes between application components while maintaining data consistency. We have also provided a user interface for integrating this framework with application fault tolerance schemes.

We have implemented uncoordinated and hybrid checkpoint framework on top of the DataSpaces, and deployed them on Cori. We have evaluated the effectiveness and performance of our approach through synthetic tests. Our experiments demonstrate that compared with global coordinated checkpoint, our uncoordinated checkpoint and hybrid checkpoint with data logging framework can effectively reduce the execution time of scientific in-situ workflows.

As future work, we plan to integrate the approach described in this paper with other fault tolerance methods such as proactive checkpointing and hierarchical checkpointing, and experimentally evaluate them using real application workflows.

Acknowledgment

This worked was supported by the National Science Foundation (NSF) via grant 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 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).

References

[1] U.S. Department of Energy, Office of Science. (2018) Exascale computing project. https://www.exascaleproject.org/exascale-computing-project/.

[2] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[3] 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

[4] 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.

[5] 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.

[6] 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.

[7] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "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)*, November 2017.

[8] J. Dongarra and et al., "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[9] 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.

[10] 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.

[11] 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.

[12] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410–416, 2009.

[13] 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.

[14] 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.

[15] M. S. Bouguerra, A. Gainaru, L. B. Gomez, F. Cappello, S. Matsuoka, and N. Maruyam, "Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*, May 2013, pp. 501–512.

[16] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'10)*, November 2010, pp. 1–11.

[17] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra, "Post-failure recovery of mpi communication capability: Design and rationale," in *International Journal of High Performance Computing Applications*, vol. 27, August 2013, pp. 244–254.

[18] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. Dongarra, "An evaluation of user-level failure mitigation support in mpi," in *Proceedings of the 19nd European MPI Users' Group Meeting (EuroMPI'12)*, September 2012.

[19] S. Duan, P. Subedi, K. Teranishi, P. Davis, H. Kolla, M. Gamell, and M. Parashar, "Scalable data resilience for in-memory data staging," in *Proceedings of the 32th IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*, May 2018, pp. 105–115.

[20] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 375–408, September 2002.

[21] A. Bouteiller, T. Ropars, G. Bosilca, C. Morin, and J. Dongarra, "Reasons for a pessimistic or optimistic message logging protocol in mpi uncoordinated failure, recovery," in *2009 IEEE International Conference on Cluster Computing and Workshops*, August 2009, pp. 1–9.

[22] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, November 2012.

[23] P. Subedi, P. Davis, S. Duan, S. Klasky, H. Kolla, and M. Parashar, "Stacker: An autonomous data movement engine for extreme-scale data staging-based in-situ workflows," in *High Performance Computing, Networking, Storage and Analysis (SC), 2018 International Conference for*. ACM, 2018.

[24] S. Duan, P. Subedi, P. Davis, and M. Parashar, "Addressing data resiliency for staging based scientific workflows," in *High Performance Computing, Networking, Storage and Analysis (SC), 2019 International Conference for*. ACM, 2019.

[25] 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.

[26] 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.