P-ckpt: Coordinated Prioritized Checkpointing

Subhendu Behera[1], Lipeng Wan[2], Frank Mueller[1], Matthew Wolf[2], Scott Klasky[2]
[1]*North Carolina State University*, ssbehera@ncsu.edu, mueller@cs.ncsu.edu
[2]*Oak Ridge National Laboratory*, {wanl,wolfmd,klasky}@ornl.gov

***Abstract*—Good prediction accuracy and adequate lead time to failure are key to the success of failure-aware Checkpoint/Restart (C/R) models on current and future large-scale High-Performance Computing (HPC) systems.**

This paper develops a novel checkpointing technique, called *p-ckpt*, that aims to maintain the performance efficiency of failure-aware C/R models even when failures are predicted with a small lead time. The *p-ckpt* technique is developed for HPC systems with multi-level memory systems to prioritize checkpoints from vulnerable nodes (nodes with predicted failure) in the event of failure prediction. It applies coordination among the nodes within an application so that vulnerable nodes' checkpoint data is stored to the Parallel File System (PFS) first by assigning priorities based on the lead time to failure. Vulnerable nodes thus have low-latency access on the critical path to the PFS before any failure happens. Further, we create the *hybrid p-ckpt* model by integrating Live Migration (LM) because of its cost-effectiveness and to reduce checkpoint frequency. Our *hybrid p-ckpt* C/R model considers prediction lead time and checkpoint latency to the PFS to decide on a feasible proactive action such as *p-ckpt* and LM. Simulations of six real-world applications for the Summit supercomputer show a ≈53-65% reduction in overhead due to the *hybrid p-ckpt* model compared to a ≈31-61% reduction in a state-of-the-art solution. We assess our C/R models against multiple failure distributions and consider lead time variability and failure prediction accuracy. Based on this evaluation and assessment, we discuss the trade-offs of using these models and their impact on application overhead.

***Index Terms*—Fault Tolerance, High-Performance Computing, Failure Prediction, I/O subsystem, Checkpoint/Restart, Live Migration, Burst Buffers**

I. Introduction

Failures and I/O contention add significant overhead to application execution and become the key challenge for C/R efficiency [1]–[6]. In past years, significant progress has been made on failure prediction [7]–[10], live migration (LM) [11], [12], and Burst Buffers (BBs) utilization [5], [13] to address the challenges of fault tolerance and PFS I/O contention. Based on these techniques, many hybrid solutions such as failure-aware safeguard checkpointing [14], [15], and multi-level C/R by orchestrating failure prediction, LM, BBs and periodic checkpointing [16] have been proposed. Multi-level Checkpoint models employ multiple storage layers with different latency to minimize checkpoint latency while improving system efficiency [17]–[19]. Safeguard Checkpoint [14] tries to minimize computation loss by checkpointing just-in-time before an anticipated failure. However, effectiveness of these failure-aware C/R solutions depends on the accuracy of the failure prediction model and the length of the lead times to predicted failures. Proactive techniques, such as LM, require high lead times for a larger memory footprint while safeguard checkpoints require enough lead time to complete a proactive checkpoint until the data is committed to the PFS amid I/O congestion. Without adequate lead time, an effort to complete the safeguard checkpoint is not guaranteed to succeed (i.e., complete before the failure) on vulnerable nodes given the unpredictable nature of I/O completion on large HPC systems. Such challenges require solutions that can effectively meet the deadline to commit checkpoint data to the PFS on vulnerable nodes. Prioritizing checkpoint data bleed-off on individual nodes is a promising direction. A filesystem-level implementation of prioritizing checkpoint data on vulnerable nodes to PFS requires complex changes in the filesystem, I/O server layer, and interconnect network layer. Further, in a system with a high failure rate, simple prioritization of an unhealthy node would fail given the Weibull distribution of failure arrival times on HPC systems [15].

To address these challenges, we propose a novel coordinated prioritized checkpoint method, called *p-ckpt*, that coordinates processes within an application in their effort to store checkpoint data to the PFS in giving vulnerable nodes higher priority for such actions. The core idea is as follows: In the event of failure predictions, a *p-ckpt* request gets initiated by the vulnerable node. It triggers the checkpointing process to, in the first phase, checkpoint to the PFS on only the vulnerable nodes. During this time, the healthy nodes await a checkpoint completion notification from vulnerable nodes to move forward to the next phase, and a new node predicted to fail during this phase gets queued in the node-local priority queue based on its lead time to failure. Once all the vulnerable nodes commit their checkpoint data, the remaining nodes commit their checkpoint data to the PFS in a second phase. Such coordination is facilitated by prioritizing vulnerable nodes based on the lead time to the predicted failure. A lower lead time implies a higher priority. Further, we incorporate LM into the C/R model, thereby creating a so-called *hybrid p-ckpt*. LM is the preferred proactive choice over prioritized checkpoints as it is cost-effective in terms of network traffic and its ability to let the application continue execution while LM is in progress [11].

Our multi-level *hybrid p-ckpt* C/R model for modern HPC systems can benefit in performance from failure prediction and an analysis model for effective use and coordination of multiple resilience techniques such as *p-ckpt*, LM, and periodic checkpoints. Our adaptive C/R model is driven by failure lead time prediction to select an appropriate proactive action with the smallest possible overhead in the presence of failures. Desh's [7] log-based failure chain characterization

technique is utilized to detect instances of likely failures in real-world HPC system logs (three HPC systems) and their lead time distribution, which provides the means for failure prediction and system failure rate calculation. LM and *p-ckpt* checkpoints are integrated into our C/R model to provide just-in-time mitigation of a predicted failure. The approach is unique in that it selects the best possible actions dynamically based on the lead time to failure, either via *p-ckpt*, to checkpoint to the PFS, or via LM with minimal interruption to application execution. The impact of this trade-off is subject to our evaluations.

In summary, we make the following contributions:

- We assess the impact of short lead times to predicted failures comparing existing safeguard checkpoint and live migration [14], [16] solutions.
- We propose a novel checkpoint technique, (*p-ckpt*), that allows coordination at application-level to prioritize the saving of state on nodes with imminent health problems to avoid computational loss due to failures.
- We develop a *hybrid p-ckpt* C/R model that coordinates fault tolerance techniques of LM and coordinated prioritized checkpointing while considering the latency of multiple I/O layers for storing checkpoints, driven by a log-based failure analysis and prediction model to reduce failure overhead.
- We evaluate the *p-ckpt* and *hybrid p-ckpt* C/R models against multiple failure distributions and measure their effectiveness while assessing sensitivity to lead times to failures. We test their robustness against failure prediction accuracy. We discuss these evaluations and make suggestions on their applicability. Further, we evaluate the impact of checkpoint size and LM transfer size on the performance of LM and *p-ckpt* models and provide an analytical model.

II. System Model

Our work is modeled on an HPC system that resembles the Summit supercomputer architecture. Each compute node has a BB serving as an intermediate storage device to absorb I/O write bursts locally. Other HPC architectures historically employed a cluster of dedicated BB nodes, e.g., NERSC's Cori [20]. On Summit, each BB device has 1.6 TB capacity, compared to a 512 GB DRAM size, with up to 2.1 GB/sec write and 5.5 GB/sec read I/O bandwidth [21]. BBs assist in reducing PFS I/O contention in two situations in our C/R model: First, periodic checkpoints are cached on the BBs and later asynchronously bled off to the PFS. The asynchronous bleed off is optimized by limiting the number of nodes that transfer data to the PFS at any time. Second, during recovery from unmitigated failures, only one node, the replacement node, needs to recover checkpoint data from the PFS. The rest of the nodes recover data from their local BB.

Further, each node has an instance of a fault predictor using the Aarohi [22] model running on a separate core than the application. Aarohi suggests placing predictors on Hardware Supervisory System (HSS) that manages a chassis on Cray systems. Their observations are based on the grounds of application interference from the predictor. However, notifying the prediction handler subsystem/thread on a compute node from a chassis controller can be challenging in terms of latency, particularly when prediction lead times are in the range of a few seconds. Placing the predictor on the node itself eliminates this problem, preferably on a spare core or otherwise by core sharing with applications. Aarohi itself is a lightweight monitor that predicts a failure by analyzing 18 different logs within 0.31 msecs on average. Checkpoint data is transferred from BBs to the PFS asynchronously, e.g., via the Spectral library on Summit [23]. We assume that the integrity of the checkpoint data stored on local BBs is maintained, and the checkpoint size per node never exceeds the DRAM or BB size.

Checkpoint Model: Our proactive checkpoint technique (applicable to both *p-ckpt* and safeguard checkpoint) mandates that all the nodes commit their checkpoints to the PFS in the event of failure prediction, thereby bypassing the BBs. In contrast, periodic checkpoints are staged on to the BBs first and later drained to the PFS asynchronously. Other strategies have been pursued in multi-level checkpointing models, such as neighbor checkpointing and local disk/BB checkpointing [17], [24]. Evaluating these methods are beyond the scope of this paper, but as they are orthogonal, can themselves benefit from prioritization. Further, we mandate all the nodes in an application to save their state to avoid application restart and synchronization issues. Thus, if recovery happens from a non-handled failure, then all the healthy nodes recover checkpoint data, which was stored in a periodic checkpoint on their local BBs, while the replacement node recovers from the PFS. If a failure is mitigated with proactive checkpointing, then all the nodes recover from the PFS.

Failure Model: We make the following assumptions in our failure model:

- Failures can happen at any point in time.
- The impact of failure is limited to a single node.
- Another failure, predicted or not, can occur on a node that already had a previous failure predicted, but still with some lead time left before the failure is predicted to occur.

The Optimal Checkpoint Interval (OCI) is the near-optimal time gap between two consecutive checkpoints that aims to lower the checkpoint frequency while minimizing the computation loss due to failures. Young's formula [25] for OCI applies to single-level C/R models. Previous work by Di et al. [18], [19] and Benoit et al. [26] focused on the optimal checkpoint interval for multiple types of checkpoints, each of them stored on a separate storage medium. However, our HPC system model employs intermediate storage devices (BBs in this case) that stage the checkpoints before being bled off to the PFS asynchronously. Failures during the asynchronous checkpointing can cause loss of computation performed during the current and previous iterations as shown in Fig. 1(B). However, during our evaluation, we found that this asynchronous checkpoint window is negligible compared to the OCI because of the high performance of the PFS on Summit (see Section IV). So we use Young's formula in (1) for OCI calculation, where t_{cmpt}^{opt} is the OCI, λ the failure rate, c the

number of compute nodes a job is running on, and t_{ckpt}^{bb} the time required to write one checkpoint to the BBs by a job.

$$t_{cmpt}^{opt} = \sqrt{\frac{2t_{ckpt}^{bb}}{\lambda c}} \tag{1}$$

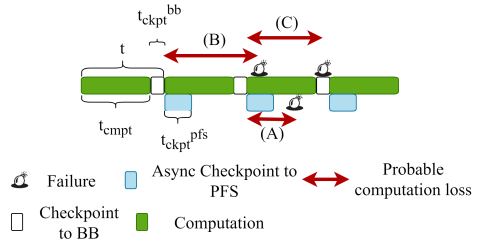


Fig. 1. Computation loss upon failure during (A) computation post checkpointing to PFS, (B) asynchronous checkpointing to PFS, and (C) synchronous checkpointing to BB

The OCI in our checkpoint model further includes a rigorous analysis of failures logs. The study analyzes the system logs collected from three real-world HPC systems from Desh [7] over a period of six months. Using the Desh approach, the most common sequences of phrases in logs that may lead to failure are considered. Our assumption in this work is that any sequence of phrases, so-called failure chains, results in an actual failure. The time difference between the first phrase and the last phrase in a chain is calculated as the lead time. Fig. 2a shows the distribution of lead times as box plots for different failure instances (sequence 1-10), each of which occurs repeatedly in these logs. Failure sequence ID and the number of occurrences in the logs are on the x-axis. The y-axis represents the lead time in seconds. Mean lead time is on the left side of each boxplot. We observe that most failures are bounded by the whiskers, only a few outliers exist, with an exception of failure sequences 3 and 4. In the following experiments, we consider the actual lead time of any failure during simulation.

We introduce a parameter σ that represents the percentage of failures that can be predicted with enough lead time in excess of the time required to migrate a process from a faulty node to a new and healthy node. By considering a σ percent decrease in the rate of failures, we further improve the OCI. That means σ percent of failures can be predicted with a lead time in excess of θ seconds and thus can be avoided with proactive live migration. We calculate the value θ by assuming that the total amount of data transferred during live migration is equal to three times the processes' checkpoint data and is bounded by RAM size (512 GB). We account for a 3x higher footprint for LM as it migrates an entire process rather than just a subset of application data. Consider a stencil with a temporal domain of t-1, t, t+1, i.e., any particle point has 3 values in time (needed by LM whereas *p-ckpt* only needs one as others can be recalculated). This approximates the overhead assuming that these data structures dominate the memory consumption of an application. Note that (1) is used for the *p-ckpt* model while (2) is applicable to the *hybrid p-ckpt* model. We do not incorporate the percentage of failures handled by *p-ckpt* in the OCI as they cause the application to recover after failure. In contrast, with live migration, failures are avoided, i.e., no recovery process is required.

$$t_{cmpt}^{opt} = \sqrt{\frac{2t_{ckpt}^{bb}}{\lambda c\,(1-\sigma)}} \tag{2}$$

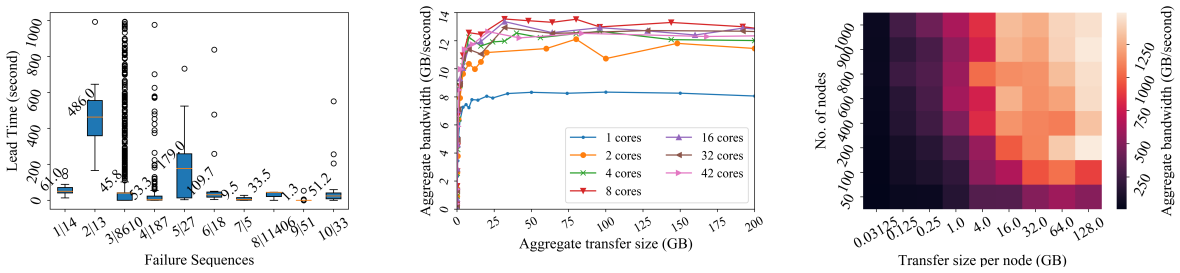
III. Simulation Framework

For evaluating the C/R models developed in this paper, we rely on simulation. SimPy [27], a process-based discrete-event simulation framework in Python, is used for developing our simulation framework. With SimPy, we simulate the time spent during computation, checkpointing to BBs and PFS, proactive operations, and inject failure events. Our simulation framework comprises multiple components (see Fig. 3). Components boxed with dotted lines run as a SimPy process during simulation. The boxes with solid lines represent the input to these components. Arrows of dotted edges indicate either actions or input at runtime, arrows with solid lines are inputs during initialization time. Each simulated application runs as a SimPy process performing computation and periodic checkpointing iteratively. The OCI of each application SimPy process is updated periodically using (1) and (2) to better account for a dynamically changing system failure rate. The checkpoint period is constant as the applications store checkpoints to BBs while asynchronously draining them to PFS.

The system and application configuration file contains input describing application characteristics, PFS I/O performance statistics, failure distribution parameters (Table III), and failure analysis (lead times) in detail. These data are fed into the simulation framework creating the static and dynamic components required for the simulation. The failure generation and prediction component uses the failure distribution parameters to generate one of the failures along with its prediction lead time using failure analysis described in Section II that is then injected into an application. For each failure generation, a node is randomly selected from a uniform probability distribution. The application on that node gets a failure prediction notification before the actual failure is triggered. Upon prediction, the application selects one of two proactive actions; which one depends on the C/R model algorithm being simulated.

When a failure is injected, the interrupted application uses the SimPy framework's time measurement APIs and the PFS I/O performance model to determine its state at the time of failure and calculates the amount of computation loss. Hence, the I/O performance model is an integral part of our simulation framework, which is described in the following section. Further, we make the following assumptions in our simulation:

- The rate of failures is lower than the rate of recovery for failed nodes so that reserved nodes are always available to the resource manager, such as Slurm [28] or Flux [29].
- No distinction is made between soft failures and hard/node failures, i.e., both are handled uniformly, except during the recovery process. A failed node is always replaced by a new and healthy node.
- Checkpointing resembles application-level checkpointing.



(a) Failure prediction lead time distribution (b) I/O performance on single compute node (c) Impact of scaling on I/O bandwidth

Fig. 2. Model and validation

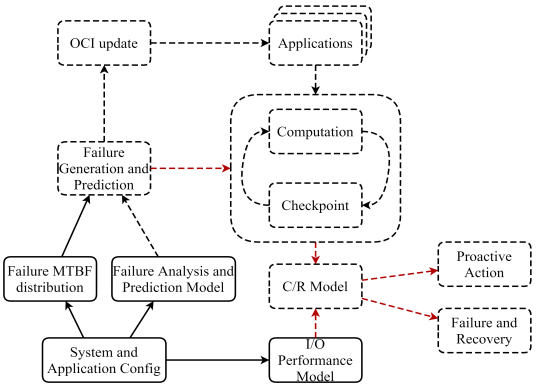


Fig. 3. Simulation framework

IV. I/O Performance Model

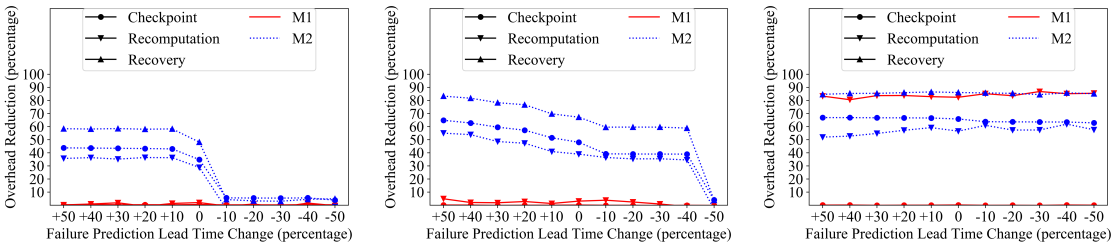
I/O performance is known to be variable due to I/O contention between different jobs. Even on modern HPC systems, the I/O bandwidth of an application is severely impacted by concurrent I/O operations performed by other applications. This causes significant variability in I/O performance. On Summit, IBM's SpectrumScale $GPFS^{TM}$ PFS handles application I/O using IBM's $GL4^{TM}$ Elastic Storage Servers as I/O nodes. The I/O subsystem evaluation in [21] shows an aggregate bandwidth of 2.5 TB/sec can be realized. However, the evaluation measures the performance of the I/O node server. It does not measure the I/O performance realized within an application. The objective here is to characterize the *actual* I/O performance seen by an application.

To characterize the I/O performance of the $GPFS^{TM}$, two experiments are conducted. The first experiment determines the optimal number of MPI processes that can achieve maximum aggregate I/O bandwidth on a single compute node. A compute node on Summit has 42 physical cores, which are evenly distributed over two sockets along with DRAM. This experiment measures the average I/O bandwidth for different aggregate data transfer sizes over multiple MPI tasks from 1 to 42 in 10 different runs. These MPI tasks are evenly distributed over the two sockets on the compute node and use POSIX write to transfer data. I/O buffers are flushed via the fsync() call to ensure that the data is not cached but rather committed to the devices. Fig. 2b depicts the aggregate I/O bandwidth (y-axis) for different transfer sizes (x-axis) on curves ranging from 1 to 42 processes. These results indicate that 8 MPI tasks on a single compute node result in the maximum I/O bandwidth. Hence, 8 MPI tasks are used to store checkpoints in the C/R model.

The second experiment assesses the effect of weak scaling on aggregate I/O bandwidth for different sizes of aggregate data transfer per node. 8 MPI tasks are used to perform I/O on a node and its aggregate bandwidth is averaged over 10 runs. Effectively, the I/O performance of the $GPFS^{TM}$ parallel file system is modeled. Fig. 2c shows the effect of scaling (nodes on the y-axis, transfer size on the x-axis) on aggregate I/O bandwidth (indicated by the heat map). For weak scaling, a fixed data size (each column) is exposed to an increasing number of nodes. We increase the data size along the x-axis and construct the I/O performance matrix. In our simulation, this performance matrix is used to calculate the time required to store checkpoint data in the PFS. Our simulation is based on the assumption that the aggregate bandwidth of a job is not affected by the I/O traffic generated by other running applications for now. I/O congestion will add more overhead for the non-frequent and failure prediction driven proactive checkpoints (safeguard and *p-ckpt*) as they checkpoint to the PFS directly, but not for the asynchronous periodic checkpoints from BBs to PFS resulting in minimal impact on performance overhead across all the C/R models. Adding the effect of background traffic impacts the checkpoint overhead across all models. For evaluation purposes, we assume the same performance matrix for the I/O read operations. PFS write achieves better throughput than read because data is cached. But checkpoints must be committed to the PFS before recovery. Hence, our I/O experiments use fsync() to purge caches. Further, as stated in Section II, all nodes recover checkpoint data from BBs, except for the new replacement node in case of non-handled failures. This reduces PFS reads to a single node and thus no longer results in PFS contention, i.e., I/O performance is well below the thresholds of the aggregate scenario as discussed before. So recovery mainly depends on BBs speed, PFS is not the bottleneck.

V. Impact of Lead Time Variability

We simulated the execution of six real-world scientific applications listed in Table I to assess the impact of prediction lead time variability. Previous works [15], [30] use these application characteristics with the OLCF's Titan supercomputer as their platform. Since our experiments are based on



(a) CHIMERA (b) XGC (c) POP

Fig. 4. Impact of lead time variability on safeguard checkpointing and LM for applications (a) CHIMERA (b) XGC (c) POP

Summit, we scale up the checkpoint size for each application proportionately to the change in DRAM size using (3).

$$Size_{new} = \frac{Size_{old} * \#Nodes_{new} * DRAMSIZE_{new}}{\#Nodes_{old} * DRAMSIZE_{old}} \quad (3)$$

To generate failures, we use the Weibull distribution parameters of Table III for OLCF's Titan in place of Summit's because of unavailability of the latter. A total of 1000 simulation runs were performed and then averaged.

TABLE I
HPC WORKLOAD CHARACTERISTICS

Application	Number of Nodes	Checkpoint Size (GB) on Summit	Computation Time (hour)
CHIMERA	2,272	646,382	360
XGC	1,515	149,625	240
S3D	505	20,199	240
GYRO	126	197.2	120
POP	126	102.5	480
VULCAN	64	3.27	720

We performed our analysis using three existing C/R models:

- *Model B:* Periodic checkpointing + No prediction (base model);
- *Model M1:* Periodic checkpointing + Failure prediction & analysis model + Safeguard checkpointing; and
- *Model M2:* Periodic checkpointing + Failure prediction & analysis model + Live migration.

Both models M1 and M2 are driven by failure predictions to perform proactive actions to avoid losses due to failures. Model M2 represents the LM-C/R model [16] and starts the LM process with adequate time before failure. Model M1 [14] performs just-in-time checkpoints or safeguard checkpoints before a failure.

Fig. 4 illustrates the impact of prediction lead time variability on the applications (results for S3D, VULCAN, and GYRO omitted as they behave similarly to POP). The curves represent the percent change of overhead for each phase of M1 (red) and M2 (blue) relative to the base model B (y-axis) over percent lead time variation (x-axis). When lead times are varied, failure prediction timing is impacted. For example, with a 50% increase in lead time, failures are predicted 1.5x earlier than the original lead times. At 0% (y-axis), the overhead remains unchanged, at 100% the overhead is completely removed, i.e., higher is better. The phase of each model is indicated by the legend and defined as follows:

•*Checkpoint Overhead:* Duration for which application execution is blocked for checkpointing.

•*Recomputation Overhead:* Duration to recompute the portion of execution that was lost due to a failure.

•*Recovery Overhead:* Duration to recover from all failures.

Observation 1: Model M2 shows moderate improvement in reducing resilience overhead when lead times increase for large applications, but its performance diminishes once lead times are shorter than their reference values. In contrast, M1 reduces recomputation overhead by a larger amount than M2, but only for the smallest of applications; other overheads, and recomputation for larger applications remain unchanged.

For the large applications, CHIMERA and XGC, safeguard checkpoints (M1) do not add any benefit while they eliminate 85% of recomputation cost for smaller applications (even for a 50% decrease in lead time) and tolerate the impact of lead time variability. For S3D, in particular, M1's recomputation cost reductions gradually decrease from 77% (for 50% increased lead time) to 50% reductions (for 40% decreased lead times) and evaporates with further decrements in lead times. Safeguard checkpoints (M1) have no impact on checkpoint and recovery overhead regardless of application size.

We observe a more differentiated pattern under model M2. The support of LM in M2 reduces all types of overheads and changes with lead time variability at different rates depending on application size. For the largest application, CHIMERA, M2 sees all types of reductions rise by 8-10% (for a 10% increase in lead times relative to the reference) resulting in 35-60% savings over the base model, and then remaining stagnant for longer leads. However, a mere 10% decrease in lead times diminishes all types of benefits provided by LM in M2. Similarly, for XGC, the second largest application, benefits for all types of reductions gradually increase with longer lead times, and these benefits rise at a faster rate than for CHIMERA. With a decrease in lead time, these benefits diminish only after lead times decrease by 50% or more. For smaller applications, M2 provides consistent reductions in all types of overheads that are not affected by lead time variability.

To understand the impact of lead time variability on M1 and M2, we define two terms: FT latency and FT ratio. FT latency is the time required by M1 or M2 to complete its proactive action to mitigate failures. FT ratio is the ratio of successfully mitigated failures to the total number of failures for an application. Application size and FT latency are two key factors that impact the performance benefits in M1 and M2 when lead time is varied. Table II represents the FT ratio

in M1 and M2 for CHIMERA, XGC, and POP under varied lead times. As application size increases, both M1 and M2 require larger lead times to mitigate failures. So there is a drop in FT ratio for a lead time reference resulting in decreasing overhead reductions. This drop is also seen with increased reference lead times. This suggests that both M1 and M2's FT latencies are too high for large applications. Also, a decrease in lead time brings the FT ratio for M2 to near zero for large applications (CHIMERA and XGC) resulting in near-zero overhead reductions. However, since M2's FT latency is lower than M1's, it results in a higher FT ratio in M2 for large applications. In contrast, for smaller applications, the FT ratios remain similar and unchanged for both M1 and M2.

TABLE II
FT RATIO FOR APPLICATIONS UNDER M1 AND M2

Lead Time Change	**FT Ratio**					
	CHIMERA		***XGC***		***POP***	
	M1	M2	M1	M2	M1	M2
+50%	0.007	0.57	0.04	0.83	0.84	0.85
+10%	0.006	0.57	0.04	0.69	0.82	0.85
0%	0.006	0.47	0.04	0.66	0.84	0.85
-10%	0.004	0.04	0.04	0.58	0.83	0.86
-50%	0	0.04	0.009	0.04	0.83	0.85

This experiment illustrates that lead time variability can have a severe impact on failure prediction-assisted fault tolerance solutions. First, Safeguard Checkpointing (M1) fails at providing any benefits for large applications and only provides reductions in recomputation overhead for smaller applications. Second, a small decrease in lead time can reduce the performance benefits of LM (under M2) for large applications. Given these results, our proposed p-ckpt solution aims to tackle these challenges of short lead times as described in the following section, followed by an evaluation with the same experimental methodology as discussed so far.

VI. PRIORITY-BASED COORDINATED CHECKPOINTING

In this section, we describe the overall design of our priority-based coordinated checkpointing method, *p-ckpt*, and the *hybrid p-ckpt* model. The core idea behind *p-ckpt* is that it applies coordination among the nodes within an application before checkpointing to the PFS. It supports the prioritization of vulnerable nodes during the checkpointing to guarantee them contention-free access to the PFS. The *hybrid p-ckpt* model orchestrates *p-ckpt* with another proactive choice LM. However, LM is the preferred choice in our C/R model over *p-ckpt* as it allows the application with a vulnerable node to continue its execution while its pages are being copied to a replacement node [11]. Further, checkpointing/restarting (to/from PFS) is more costly than LM in terms of network traffic for medium to large applications.

Fig. 5 depicts the state transitions of a node in the *hybrid p-ckpt* model. The square boxes with solid lines represent different states of a node. The ellipses with dotted transitions represent notifications as required. The solid arrows represent state transitions. When a failure is predicted, a node transitions from the normal state of periodic computation and checkpointing to the vulnerable state. In this state, based on the predicted lead time, a decision is made on the proactive action. If there is enough time to migrate the process from the vulnerable node to a new and healthy node, then the live migration process starts. Otherwise, the vulnerable node sends a *p-ckpt* notification to all other nodes and the *p-ckpt* process begins. When a *p-ckpt* notification is received, healthy nodes transition to the waiting state and wait for the vulnerable nodes to finish checkpointing to the PFS. Once the vulnerable nodes finish storing their state to the PFS, they broadcast the *pfs-commit* message to all other nodes within the application. When the healthy nodes receive this notification, they proceed with checkpointing to the PFS. The *p-ckpt* process is implemented with node-local priority queues, where vulnerable nodes with lower lead time to failures have higher priority while all healthy nodes have equal lower priorities. When live migration is in progress and another failure prediction occurs with lower lead time, live migration is aborted and the *p-ckpt* process begins (see state diagram).

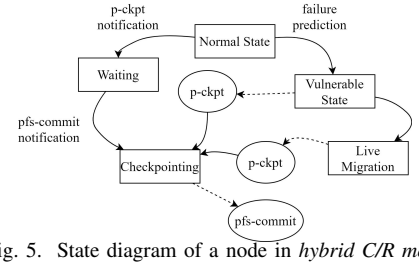


Fig. 5. State diagram of a node in *hybrid C/R model*

P-ckpt performs a few global synchronizations and broadcast operations, which adds performance overhead. However, these operations are in the order of microseconds on Summit [21]. A global barrier with 2048 nodes takes only $\approx 8\mu$secs. We do not account for these small overheads during simulation. Further, the *p-ckpt* threads run only when a *p-ckpt* is taken but otherwise do not impact applications during execution. LM's execution interleaves with application execution. However, the overhead is quite low adding just 0.08-2.98% in runtime during live migration [11].

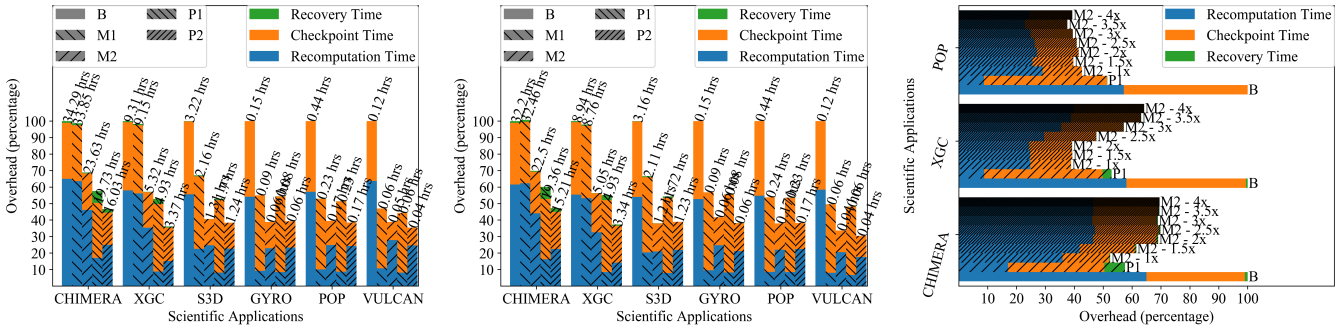
VII. EVALUATION

The simulator used in Section V is also used for the evaluation of two new models as below relative to the same base model B as before:

- *Model P1:* Periodic checkpointing + Failure prediction & analysis model + *p-ckpt*.
- *Model P2:* Hybrid of periodic checkpointing + Failure prediction & analysis model + *p-ckpt* + LM.

Model P2 combines two different proactive fault tolerance techniques, LM and *p-ckpt*. The objective is to showcase our contributed models' benefits over fault tolerance models in prior work. No prior work combined M1+M2, and benefits may be limited for large applications (CHIMERA and XGC) as M1 is ineffective for large applications (see Section V).

Fig. 6 depicts for each application (x-axis) the overhead of fault tolerance in percent (y-axis) normalized to the base model (B) with periodic checkpoints (first bar) compared to



(a) Summit under Titan's Failure Distribution (b) System 18 (c) LM vs *p-ckpt*

Fig. 6. Reduction in overhead for Summit, LANL System 18, and LM vs *p-ckpt*

failure prediction models M1, M2, P1 (*p-ckpt*) and P2 (*hybrid p-ckpt*). All models are annotated with rounded total overhead (in hours) on top of each bar. To test the robustness of our C/R model, we applied three different failure distributions from systems referenced in Table III [15], [30]. Here, we make the assumption that OLCF's Titan's failure distribution applies to Summit, i.e., Fig. 6a depicts the overhead distribution for Summit under Titan's failure distribution.

Observation 2: *p-ckpt* (P1) and *hybrid p-ckpt* (P2) help reduce application overhead over the base model by ≈42-55% and ≈53-65% on Summit, respectively.

TABLE III
WEIBULL DISTRIBUTIONS FOR FAILURE GENERATION

HPC System	Shape	Scale
LANL System 8 (164 nodes)	0.7111	67.375
LANL System 18 (1024 nodes)	0.8170	6.6293
OLCF Titan (18868 nodes)	0.6885	5.4527

In related work [16], the LM-C/R model (M2) was guided by failure prediction and reduced the application overhead by ≈31-61%. This reduction was due to the assistance of LM. The safeguard checkpoint model (M1) by Bouguerra et al. [14] when driven by lead time-based failure prediction, reduced overall application overhead by ≈0-52% without providing any benefits for large applications. With *hybrid p-ckpt*, we observe a significantly higher reduction in cost, by ≈53-65% (see Fig. 6a), than in [14], [16]. The savings can be attributed to a combination of prioritized coordinated checkpointing (*p-ckpt*) against failures with short lead times (model P1) and lower failure rates due to prediction and successful mitigation via LM (model P2). The assistance of *p-ckpt* alone brings a ≈42-55% reduction in application overhead (see Fig. 6a), which is higher than model M2 for large applications. Table IV represents the FT ratio in P1 and P2 for CHIMERA, XGC, and POP under varied lead times. As can be seen, the lower FT latency of *p-ckpt* allows both P1 and P2 to obtain a higher FT ratio compared to models M1 and M2 (see Table II). Model M1 cannot handle failures with short lead times for large applications with safeguard checkpoints, and its FT ratio remains near zero. However, *p-ckpt* successfully handles such failures as it commits the checkpoint on the vulnerable nodes without any congestion in a prioritized manner. While M2's LM yielded an FT ratio of 0.5 and above for the base lead times and above for large applications, *p-ckpt* pushed the FT ratio in P1 and P2 even higher, resulting in better overhead reductions. Notice that the FT ratios for P1 and P2 are almost equal for all the applications. That means both P1 and P2 can handle an equal amount of faults, but the overhead reduction difference between them is significant, as discussed later.

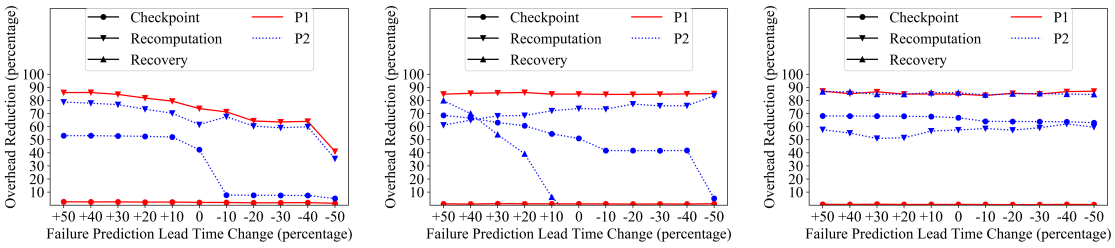
TABLE IV
FT RATIO FOR APPLICATIONS UNDER P1 AND P2

Lead Time Change	FT Ratio					
	CHIMERA		*XGC*		*POP*	
	P1	P2	P1	P2	P1	P2
+50%	0.84	0.83	0.85	0.84	0.88	0.86
+10%	0.76	0.76	0.84	0.84	0.87	0.85
0%	0.70	0.69	0.84	0.83	0.86	0.85
-10%	0.67	0.67	0.84	0.84	0.84	0.87
-50%	0.36	0.37	0.84	0.84	0.86	0.86

The stacked bars break down overhead that can be attributed to checkpointing and, after a failure, recovery to reload a checkpoint plus recomputation time to catch up with the execution to the point of failure. Notice that recovery overhead is negligible for all the models except for P1. This is due to our proactive checkpointing model, where all nodes commit their checkpoint to the PFS bypassing the BBs unlike regular checkpointing. A mitigated failure by a proactive checkpoint takes longer to recover, whereas failures unhandled are recovered faster with the assistance of BBs. We observe that recovery contributes ≈2.5-6% of total overhead for P1 compared to less than 1% for other models.

Observation 3: Both *p-ckpt* and *hybrid p-ckpt* can tolerate the impact of prediction lead time variability better than prior models for large applications.

Fig. 7 assesses the impact of varied prediction lead time on models P1 and P2 for all the applications (results for S3D, VULCAN, and GYRO omitted as they behave similarly to POP) with the same x- and y-axes as in Fig. 4 before. *p-ckpt* (P1) does not provide any additional benefits for recovery and checkpoint overheads like the M1 model (Section V). However, for the largest application CHIMERA, it produces more recomputation overhead reductions than M2 and P2 and can tolerate up to a negative 50% change (i.e., reduction) in lead times while still providing some savings in recomputation relative to the base model due to the prioritization of vulnerable nodes. In contrast, Model M1 (safeguard checkpoints) does not provide performance benefits for CHIMERA, and M2's



(a) CHIMERA (b) XGC (c) POP

Fig. 7. Impact of lead time variability on *p-ckpt* and *hybrid p-ckpt* for applications (a) CHIMERA (b) XGC (c) POP

benefits diminish when lead time decreases by 10% relative to the reference. For XGC, P1 nearly eliminates the entire recomputation overhead regardless of lead time variations. In contrast, M2's performance benefits diminish with a 50% reduction in lead time while M1 is not effective at all.

The overhead reduction pattern for checkpointing in model P2 (*hybrid p-ckpt)* with respect to lead time changes follows model M2 for both CHIMERA and XGC. The pattern for recomputation overhead follows M2 largely when lead time increases, but follows P1 when the lead time shrinks. With the support of coordinated prioritized checkpointing (*p-ckpt*), P2 achieves a similar recomputation overhead reduction pattern as model P1 gaining a significant advantage over model M2. For large applications, both models, P1 & P2, not only achieve better recomputation overhead reductions, but increase their tolerance against prediction lead time variability. Further, because of our checkpointing model (see Sec. II) in *p-ckpt* and the recovery process after proactive checkpoints, reductions in recovery overhead for P2 are not seen for XGC when lead time is less or equal to the reference. These reductions completely diminish for CHIMERA. Patterns for P1 and P2 for smaller applications follow M1 and M2, respectively.

Finding: *Variability in prediction lead time has a significant impact on the performance benefit of prediction-based C/R models, and our hybrid p-ckpt model outperforms prior related models under such circumstances.*

Observation 4: *p-ckpt* is more effective for large applications compared to LM. Higher lead times favor LM; conversely, when lead times are low, *p-ckpt* takes over.

Fig. 8 demonstrates the difference in FT ratio by LM and *p-ckpt* in percent (y-axis) in model P2 over lead time variation (x-axis) for all the applications. In this experiment, the lead time variation range was within (-90%, +90%) expanding the earlier range (-50%, +50%). If the percent difference is positive, then LM is the dominant proactive choice; otherwise, *p-ckpt* is more dominant. As can be seen, for smaller applications, the FT ratio difference between LM and *p-ckpt* remains consistently high (above 75%) across the lead time variation range. Since LM is the preferred choice ahead of *p-ckpt* and its FT latency is small enough, it can tolerate the lead time changes for smaller applications. When application size increases, the FT ratio difference between *p-ckpt* and LM decreases for the base lead time (0% change in lead time). That means *p-ckpt* is more effective for large applications compared to LM because of its lower FT latency. For larger applications, as lead times decrease, the dominance of *p-ckpt* as the proactive choice increases. *p-ckpt*'s dominance over LM is seen earlier for the largest application like CHIMERA followed by XGC and S3D. As lead time changes become negative, *p-ckpt* completely takes over LM before the FT ratio difference reaches zero as lead times completely diminish.

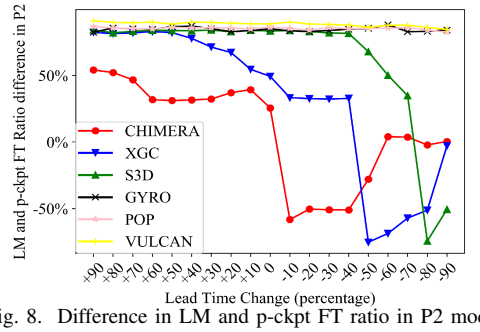


Fig. 8. Difference in LM and p-ckpt FT ratio in P2 model

Observation 5: Under *hybrid p-ckpt* (P2), checkpoint overhead is reduced by ≈42-70% across applications. In contrast, LM (M2) results in reductions of 34% compared to P2's 42% for the largest application.

Even though both P1 and P2 yield equal FT ratios (Table IV), P2 performs better than P1 in reducing overhead as LM helps model P2 to reduce checkpoint overhead. There is a negligible change in time spent in storing checkpoints under model P1 because of the adaptive nature of our checkpoint model. That means the schedule of checkpoints is variable in our model depending on the factors such as the time and location of failure prediction, and the proactive action chosen. For example, the scheduled checkpoint changes due to a *p-ckpt* triggered by and completed before a predicted failure. What is more significant is the reduced failure rate resulting from the failure analysis model, which yields a ≈42-70% decrease in checkpoint overhead in the *hybrid p-ckpt* (P2) model. Further, for the largest application, CHIMERA, P2 reduces checkpoint overhead by 42% compared to just a 34% reduction by M2. Even though LM in both M2 and P2 have the same configuration, the assistance of *p-ckpt* helps P2 in completing the execution earlier (2% faster than M2) and thus reducing checkpoint overhead.

Observation 6: In the presence of frequent faults, applications can suffer higher recomputation overhead with *hybrid p-ckpt* compared to *p-ckpt*.

For all the applications, the recomputation overhead increases under (compare blue bars between P1 and P2 in Fig. 6a and Fig. 6b) due to the inclusion of LM and elongated checkpoint intervals derived from our extended failure analysis model as per (2). The reduced failure rate increases the optimal checkpoint interval by ≈54-340%, which indirectly impacts the computation losses due to failures that could not be predicted or avoided even if predicted in advance. The elongated checkpoint interval increases the hours of computation loss when failures are not proactively avoided. P2 experiences a ≈11-27% increase in recomputation overhead relative to the base model when compared to P1. However, P2's loss in performance benefits is compensated by the reduced checkpoint overhead. This gives rise to the requirement of a careful selection of the C/R model for fault tolerance.

Recommendation: *Based on the analysis in observations 4 and 6, we suggest that HPC systems with a high fault rate and low lead times should utilize p-ckpt (P1) for large applications with short runtimes because of its ability to handle failures with short lead times and reduced computation loss derived from more frequent checkpointing. In contrast, applications with long runtimes should use the hybrid p-ckpt (P2), irrespective of size and failure rate, as checkpoint overhead can eclipse the recomputation overhead.*

Observation 7: Reductions in overheads for model P2 are robust across different Weibull failure distributions.

Fig. 6b depicts the reduction in overhead on the same x- and y-axes as before (Fig. 6a), yet for Systems 18 with its failure distribution. The reduction in overhead follows a similar pattern for all three failure distributions. For LANL System 8 (figure not presented due to lack of space), the decrease in overhead is ≈44-73% while System 18 results in ≈52-69% reduced overhead. Furthermore, the same pattern of increasing gains with decreasing checkpoint sizes is observed. This result is significant as it demonstrates that our model is robust and generalizes to other failure distributions. *In principle, our C/R model can be deployed on any HPC system that supports BBs, LM, and failure analysis plus prediction. It also shows that orchestrating failure prediction within a C/R model to drive decisions about when and how to checkpoint and when to live migrate reduces the impact of failures and shortens application execution over simpler failure models.*

Observation 8: The larger an application's checkpoint size is, the larger the advantage of *p-ckpt* over LM will be.

As mentioned in Sec. II, we assume that the amount of data transferred for successful LM is three times that of the checkpoint data size per process. To understand how this factor impacts the performance comparison of LM (M2) and *p-ckpt*, we varied the amount of data transfer for LM and created multiple models designated with M2-*, where * indicates the factor of checkpoint data for transfer. Fig. 6c shows the impact of varying transfer size for LM. The horizontal bars represent overhead reductions (similar to Fig. 6a) for all the models (B, P1, and M2-*) along the x-axis for three applications on the y-axis. We observe that for large applications (CHIMERA and XGC), *p-ckpt* (P1) performs better than LM (M2) overall until the LM transfer size becomes 1x and 2.5x times the checkpoint size, respectively. For smaller applications, LM always performs better than *p-ckpt*. Furthermore, reductions in recomputation overhead for *p-ckpt* (P1) are significantly larger than for LM (M2).

Based on this analysis, we provide an analytical model to compare LM and *p-ckpt*. We observe that LM (M2) reduces checkpoint overhead significantly (Observation 5), whereas *p-ckpt* (P1) yields better recomputation reductions than LM (Observation 4). For *p-ckpt* to perform better than LM, the difference in recomputation overhead reductions between *p-ckpt* and LM must be greater than the checkpoint overhead reduction by LM. This is captured by (4). Notice that we consider the recovery overhead in model P1 as negligible.

$$ckpt^{LM}_{reduction} < \left(recomp^{P-CKPT}_{reduction} - recomp^{LM}_{reduction}\right) \quad (4)$$

The first term can be expressed as (5), where the first term represents the total checkpoint overhead in the base model (B) and the third term represents a fractional reduction in checkpoint frequency due to LM (see (2)).

$$ckpt^{LM}_{reduction} = ckpt^{B}_{overhead} * \left(1 - \sqrt{(1-\sigma)}\right) \quad (5)$$

Similarly, $recomp^{P-CKPT}_{reduction}$ and $recomp^{LM}_{reduction}$ can be represented as $\left(recomp^{B}_{overhead} * \beta\right)$ and $\left(recomp^{B}_{overhead} * \sigma\right)$, respectively. σ and β represent the fraction of failures that can be handled by LM and *p-ckpt*, respectively, and $recomp^{B}_{overhead}$ represents the total recomputation overhead of model B.

The right side of (4) can be simplified as $recomp^{B}_{overhead} * (\beta - \sigma)$. If we consider a uniform distribution of lead times of failures, an equal inter-node network bandwidth and single node PFS write bandwidth (which is the case for Summit with 12.5 GB/sec and 13-13.5 GB/sec, respectively), then β can be expressed using (6). α is the ratio of LM's transfer size to checkpoint data size.

$$\beta = \frac{\alpha - 1 + \sigma}{\alpha} \quad (6)$$

Equation (4) can be re-written as

$$\frac{\left(1 - \sqrt{(1-\sigma)}\right)}{\frac{\alpha-1+\sigma}{\alpha} - \sigma} < \frac{recomp^{B}_{overhead}}{ckpt^{B}_{overhead}} \quad (7)$$

Assuming application overhead is split in half between recomputation and checkpointing, (7) is further simplified to

$$\frac{\sigma + 1}{\sigma + \sqrt{(1-\sigma)}} < \alpha \quad (8)$$

Based on the constraint that the sum of $recomp^{LM}_{reduction}$ and $ckpt^{LM}_{reduction}$ must be less than $recomp^{B}_{overhead}$, $\sigma < 0.61$. Equation (8) suggests that α must increase non-linearly as σ grows. Under the constraints of $0 <= \sigma < 0.61$, the LM transfer size to checkpoint size ratio implies $1.04 <= \alpha < 1.30$ for *p-ckpt* to perform better than LM.

Observation 9: All models (M1/M2/P1/P2) experience a steady decline in total overhead reduction as the false negative rate increases. However, LM-supported models (M2/P2) experience larger declines in recomputation overhead reductions than the safeguard checkpoint and *p-ckpt* models (M1/P1).

To observe the impact of false negatives, we kept the false positive rate constant at 18% (see [31]) and varied the false negative rate (figures omitted due to space) up to 40%. Models

TABLE V
C/R MODEL COMPARISON

C/R Model	Failure Awareness	Coordinated prioritized checkpoint	Safeguard Checkpoint	Periodic Checkpoint	Live Migration	PFS I/O Model	Failure Prediction	Async Ckpt. Interval
Hybrid p-ckpt	Failure Lead Time Prediction	✓	✗	✓	✓	✓	✓	✓
Wang et al.	Health Monitoring	✗	✗	✗	✓	✗	✗	✗
Bouguerra et al.	Failure Lead Time Prediction	✗	✓	✓	✗	✗	✓	✗
Tiwari et al.	Failure Locality	✗	✗	✓	✗	✗	✗	✗
Behera et al.	Failure Lead Time Prediction	✗	✗	✓	✓	✓	✓	✗

M2 and P2 observe a ≈91-180% and ≈71-174% decline in recomputation overhead reduction, respectively, when the false negative rate reaches up to 40%. However, M1 and P1 observe smaller reductions of ≈48-54% and ≈35-40%, respectively. This means they actually can handle a fewer number of failures with an increasing false negative rate. LM-assisted models, M2 and P2, overestimate the number of failures they can handle and keep the checkpoint interval larger than models M1 and P1 (see (2)). It confirms our recommendation in Observation 6 that on failure-prone systems, P1 holds an advantage over P2. To improve P2, the failure prediction accuracy factor needs to be included in (2), which is part of our future work.

Drawbacks: We rely on simulation for evaluating the models as special privileges are required to reserve nodes on Petascale systems (e.g., Summit) for large-scale experiments. To mitigate this, our evaluated applications are compute-intensive and use I/O during checkpointing based on the real machine data from Summit with a validated I/O write performance model [16]. Simulated failures are based on real-world HPC failure logs [7], [8]. Some operations such as synchronization and broadcast introduce overhead, but these are negligible. A *p-ckpt* barrier with 2048 Summit nodes take 8 microseconds. We also ignore the overhead of failure prediction as Aarohi [32] predicts failures within 0.31 msec.

Feasibility: Utilization of two different proactive options under a single FT model (P2) requires coordination among multiple software systems like LM, *p-ckpt*, and periodic checkpointing. Further, to use a different proactive choice for individual applications, LM requires a global system view to avoid migrations that can create conflicts. *p-ckpt* applies to individual applications only by coordinating its processes. *p-ckpt* with a global system view is beyond the scope of this paper, as is a complete implementation of the whole system.

VIII. Related Work

Several C/R solutions leverage failure awareness [11], [14], [15], [33]–[35]. Wang et al. [11] monitor healthy nodes and migrate processes if the node's health deteriorates. However, the evaluation of their model excludes failures and only evaluates the efficiency of the live migration technique. Bouguerra et al. [14] use proactive checkpoints upon failure prediction along with preventive checkpoints to reduce computation waste. They use FTI's [24] level 0 checkpointing strategy for proactive checkpoints and regular periodic checkpointing to PFS for periodic/preventive checkpoints. At level 0, checkpoint data of the failing node is stored on a neighbor node. Our model, for both *p-ckpt* and safeguard checkpoint, mandates the checkpoint data of all the nodes to be committed to PFS. Bouguerra et al. [14]'s proactive checkpointing adds 2-6% overhead to checkpoint time, whereas the adaptive nature of our model limits *p-ckpt*'s overhead to less than 1%. Further, *hybrid p-ckpt* reduces the checkpointing overhead by ≈42-70% due to reduced failure rate and faster execution completion. Bouguerra et al. [14]'s model relies on the failed node to restart for recovery purposes, whereas our model utilizes reserved nodes. Bouguerra et al. [14]'s model achieves a 22% reduction in total overhead due to proactive checkpointing compared to a ≈53-65% reduction with our *hybrid p-ckpt* model. Recomputation overhead reduction in Bouguerra et al. [14]'s model is 17%, whereas our model's impact on recomputation overhead is a ≈56-73% reduction. Tiwari et al. [15] increase the checkpoint interval until failure and skip selected checkpoints post-failure using a temporal distribution of failures. Our C/R model's uniqueness comes from the use of failure prediction that selects the best mitigation action based on lead time. Further, we developed *p-ckpt* that replaces proactive checkpoints and improves fault tolerance for predictions with short lead time, even for large applications. Tiwari et al. [15]'s evaluation platform is based on OLCF's Titan while ours is on Summit. Their scheme reduces checkpoint time up to 70%, ours by ≈42-70%. However, our checkpoint overhead accounts for the commits to the BBs while theirs commits to the PFS. Further, their recomputation overhead offsets most of the gains made with checkpoint overhead reductions. In contrast, our model provides more overhead reductions with less recomputation. George et al. [33] deploy partial replication of process sets and predict failures to change the replicated process group. Garg et al. [34] exploit failure locality to schedule applications with higher checkpoint overhead during lower failure rates and applications with lower checkpoint overhead during higher failure rates. In contrast, our model relies on dynamically predicted failures. Behera et al. [16] developed a C/R model with live migration and assessed its benefit under failure prediction. However, their work could not handle faults with low prediction time. Tab. V compares our C/R model with other C/R models and illustrates the uniqueness and comprehensiveness of our approach in contrast to prior work.

IX. Conclusion

We developed a multi-level C/R model that provides fault tolerance by orchestrating failure prediction with proactive

actions of coordinated prioritized checkpoints (*p-ckpt*) and live migration via prioritization along critical failure paths, which results in reduced application overhead by ≈53-65% compared to a ≈31-61% reduction by conventional LM-C/R.

Overall, proactive actions should resort to *p-ckpt* in an HPC system with short lead times and high failure rates for short-running, large applications. In contrast, *hybrid p-ckpt* should be used for long-running applications, irrespective of application size and system failure rate. Our *hybrid p-ckpt*'s coordination of multiple fault tolerance techniques through failure prediction is unprecedented while providing better tolerance against failures with short lead times.

ACKNOWLEDGMENT

We would like to thank the reviewers for their valuable feedback. This research was supported in part by NSF grants 1525609, 1813004, 1818914, DOE ASCR SIRIUS-2 project and Exascale Computing Project (17-SC-20-SC). This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] A. Geist and C. Engelmann, "Development of naturally fault tolerant algorithms for computing on 100,000 processors," 01 2003. [Online]. Available: https://www.csm.ornl.gov/ geist/Lyon2002-geist.pdf

[2] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka, "Design and modeling of a non-blocking checkpointing system," in *Supercomputing*, Nov 2012, pp. 1–10.

[3] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, p. 012022, jul 2007.

[4] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, and G. Chiu, "Doe advanced scientific computing advisory subcommittee (ascac) report: Top ten exascale research challenges," 2 2014.

[5] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Symposium on Mass Storage Systems and Technologies (MSST)*, April 2012, pp. 1–11.

[6] K. Iskra, J. W. Romein, K. Yoshii, and P. Beckman, "Zoid: I/o-forwarding infrastructure for petascale architectures," in *Symposium on Principles and Practice of Parallel Programming*, 2008, pp. 153–162.

[7] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep learning for system health prediction of lead times to failure in hpc," in *Symposium on High Performance Distributed Computing*, Jun. 2018, pp. 40–51.

[8] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, "Doomsday: Predicting which node will fail when on supercomputers," in *Supercomputing*, Nov. 2018, pp. 9:1–9:14.

[9] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems," *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 1168–1179, 2012.

[10] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *Supercomputing*, 2012.

[11] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in hpc environments," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, 2008.

[12] ——, "Proactive process-level live migration and back migration in hpc environments," *Journal of Parallel Distributed Computing*, vol. 72, no. 2, pp. 254–267, Feb. 2012.

[13] A. Fang and A. A. Chien, "How much ssd is useful for resilience in supercomputers," in *Workshop on Fault Tolerance for HPC at eXtreme Scale*, pp. 47–54.

[14] M. Bouguerra, A. Gainaru, L. Bautista-Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing," in *International Parallel and Distributed Processing Symposium*, May 2013, pp. 501–512.

[15] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *International Conference on Dependable Systems and Networks*, June 2014, pp. 25–36.

[16] S. Behera, L. Wan, F. Mueller, M. Wolf, and S. Klasky, "Orchestrating fault prediction with live migration and checkpointing," in *Symposium on High Performance Distributed Computing*, Jun. 2020.

[17] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Supercomputing*, 2010, p. 1–11.

[18] S. Di, Y. Robert, F. Vivien, and F. Cappello, "Toward an optimal online checkpoint solution under a two-level hpc checkpoint model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 244–259, Jan 2017.

[19] S. Di, M.-S. Bouguerra, L. A. Bautista-Gomez, and F. Cappello, "Optimization of multi-level checkpoint model for large scale hpc applications," in *International Parallel and Distributed Processing Symposium*, May 2014, pp. 1181–1190.

[20] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, and M. a. Bryson, "Accelerating science with the nersc burst buffer early user program," 05 2016. [Online]. Available: https://www.nersc.gov/assets/Uploads/Nersc-BB-EUP-CUG.pdf

[21] S. S. Vazhkudai, B. R. de Supinski, A. S. Bland, A. Geist, J. Sexton, J. Kahle, C. J. Zimmer, S. Atchley, S. Oral, D. E. Maxwell, and et al., "The design, deployment, and evaluation of the coral pre-exascale systems," in *Supercomputing*, 2018.

[22] A. Das, F. Mueller, and B. Rountree, "Aarohi: Making real-time node failure prediction feasible," in *International Parallel and Distributed Processing Symposium*, 2020, pp. 1092–1101.

[23] ORNL. (2020, 02) Spectral library. [Online]. Available: https://www.olcf.ornl.gov/spectral-library/

[24] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "Fti: High performance fault tolerance interface for hybrid systems," in *Supercomputing*, 2011.

[25] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, no. 9, pp. 530–531, 1974.

[26] A. Benoit, A. Cavelan, V. Fevre, Y. Robert, and H. Sun, "Towards optimal multi-level checkpointing," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1212–1226, July 2017.

[27] S. Team. (2020, 02) Simpy: Discrete-event simulation for python. [Online]. Available: https://pypi.org/project/simpy/

[28] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., 2003, pp. 44–60.

[29] D. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, J. Koning, P. Tapasya, T. Scogland, B. Springmeyer, and M. Taufer, "Flux: Overcoming scheduling challenges for exascale workflows," 11 2018, pp. 10–19.

[30] L. Wan, Q. Cao, F. Wang, and S. Oral, "Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems," *Journal of Parallel and Distributed Computing*, vol. 100, pp. 16 – 29, 2017.

[31] A. Das, A. Vishnu, C. Siegel, and F. Mueller, "Desh: Deep learning for hpc system health resilience," in *SC Poster Session*, Nov. 2017.

[32] A. Das, "Aarohi: Making real-time node failure prediction feasible," in *International Parallel and Distributed Processing Symposium*, Apr. 2020.

[33] C. George and S. Vadhiyar, "Fault tolerance on large scale systems using adaptive process replication," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2213–2225, Aug 2015.

[34] R. Garg, T. Patel, G. Cooperman, and D. Tiwari, "Shiraz: Exploiting system reliability and application resilience characteristics to improve large scale system throughput," in *International Conference on Dependable Systems and Networks*, June 2018, pp. 83–94.

[35] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *International Parallel and Distributed Processing Symposium*, April 2008, pp. 1–9.