2014 IEEE 28th International Parallel & Distributed Processing Symposium

Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications

Sheng Di[1,2], Mohamed Slim Bouguerra[1,2], Leonardo Bautista-gomez[2], Franck Cappello[2]
[1]INRIA, France, [2]Argonne National Laboratory, USA,
sheng.di@inria.fr, slim.bouguerra@imag.fr, {leobago, cappello}@mcs.anl.gov

***Abstract*—HPC community projects that future extreme scale systems will be much less stable than current Petascale systems, thus requiring sophisticated fault tolerance to guarantee the completion of large scale numerical computations. Execution failures may occur due to multiple factors with different scales, from transient uncorrectable memory errors localized in processes to massive system outages. Multi-level checkpoint/restart is a promising model that provides an elastic response to tolerate different types of failures. It stores checkpoints at different levels: e.g., local memory, remote memory, using a software RAID, local SSD, remote file system. In this paper, we respond to two open questions: 1) how to optimize the selection of checkpoint levels based on failure distributions observed in a system, 2) how to compute the optimal checkpoint intervals for each of these levels. The contribution is threefold. (1) We build a mathematical model to fit the multi-level checkpoint/restart mechanism with large scale applications regarding various types of failures. (2) We theoretically optimize the entire execution performance for each parallel application by selecting the best checkpoint level combination and corresponding checkpoint intervals. (3) We characterize checkpoint overheads on different checkpoint levels in a real cluster environment, and evaluate our optimal solutions using both simulation with millions of cores and real environment with real-world MPI programs running on hundreds of cores. Experiments show that optimized selections of levels associated with optimal checkpoint intervals at each level outperforms other state-of-the-art solutions by 5-50 percent.**

I. Introduction

Most of recent scientific problems have to launch a large number of processes simultaneously to process huge amount of workload for a long time. For example, recent studies [1] show that taking advantage of about 83k processors of one of the world's most powerful supercomputers can only mimic just one percent of one second's worth of human brain activity - and even that took 40 minutes.

Any HPC environment may inevitably encounter various hardware failures from time to time, significantly decreasing the reliability [2]. For example, at Lawrence Livermore National Laboratory (LLNL), a BlueGene/L system with 100k nodes encounters one L1 cache bit error every 8 hours [3]. Hence, it is likely to experience fatal failures in the course of large scale executions, because any interruption to one running process will cause the whole execution crash.

In order to resolve the fault-tolerance issue, most of existing fault-tolerance projections [4] adopt periodic checkpoint/restart model [5], [6], not only because it is very easy to apply in practice but also with a satisfied overall performance in general [7], [8], [9]. For instance, for an MPI program, one just needs to periodically store the processes' memories into *Parallel File System* (*PFS*), and roll back the execution to the most recent checkpoint when necessary.

For large scale applications with huge data size to process, classic checkpoint/retart method with single level suffer from huge performance degradation. First, there are many different types of transient/software failures[1] and various levels of hardware failures to strike the application, such that it is hard to predict failure locations with high accuracy. For example, ELSA [10] is an excellent failure prediction tool, yet it still suffers from 50% prediction errors in recall. Second, it is too costly to always store checkpoint files into PFS. On the one hand, large scale applications tend to suffer frequent failures because of a large number of running processes. Even worse, exascale applications [11] might experience multiple simultaneous hardware failures. On the other, large scale applications tend to load larger-size data to process, leading to huge checkpoint/restart overhead (up to 25%) [12], because of I/O bottleneck.

Multi-level checkpoint/restart model [13], [14], [15], [16] has been considered the best-fit solution for the above large scale fault-tolerance issue. Such a model is able to suit various overheads on checkpointing at different storage levels. *Scalable Checkpoint/Retart* (SCR) [15], for example, is a well-known multi-level checkpointing library, which allows to write checkpoints to RAM, Flash, or disk on compute nodes in addition to PFS. Fault Tolerance Interface (FTI) [16] is another outstanding toolkit which provides four-level checkpointing interfaces, including local-disk, partner-copy, Reed-Solomon coding (RS-coding), and PFS. Obviously, multi-level checkpoint/restart model creates a new avenue to refine the optimization of checkpoint intervals on demand, with significantly lower checkpoint/restart cost than single-level checkpoint model. If the application fails due to software errors or user interruption, we just need to restart the execution from the recent checkpoint stored on local-disk. If the failure is due to disk errors or node crash, we have to restart the execution by checkpoints stored on remote disks like nearby partner storage device or PFS.

In this paper, we present the following three contributions based on the multi-level checkpoint/restart model.

- We build a generic mathematical multi-level checkpoint model for optimizing the execution of large scale HPC applications. Our model is suitable for the situation with correlated failures like the cases with

[1]transient failure here refers to software failure, as opposed to hardware failure. That is, local disk on the host is still available.

1530-2075/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPS.2014.122

simultaneous adjacent node failures in a rack. This contrasts the well-known analysis like Young's formula [5] and Daly's work [6], which is subject to single-level checkpoint/failure situation.

- Our solution is split into two stages, which optimize the checkpoint intervals and the selection of levels respectively. For the former, we compute optimal checkpoint intervals for different checkpoint levels, with in-depth analysis of entire wall-clock time for each application. The serious challenge is that failure locations are unpredictable and different levels may not be independent with each other. We have to combine different levels of failures together to optimize the entire performance.
- We evaluate our optimal multi-level checkpoint/restart method on both large scale simulation and a real cluster environment with real-world HPC applications. We compare our solution with optimal selection of levels associated with optimized checkpoint intervals to three other state-of-the-art solutions: (1) single-level checkpoint only with PFS, (2) multi-level checkpoint with all possible levels and Young's formula [5], (3) multi-level checkpoint with all possible levels with our optimal checkpoint intervals. Experiments show that our solution outperforms other solutions by 5-50%.

The rest of the paper is organized as follows. In Section II, we briefly introduce the background of multi-level checkpoint mechanism and present the system overview. In Section III, we formulate the multi-level periodic checkpoint model to be an optimization problem, by aiming to compute the optimal checkpoint intervals for different levels. In Section IV, we derive the optimal checkpoint intervals for different levels with various types of failures, and also optimize the selection of levels based on the optimized checkpoint intervals. We present in Section V our experimental results. We discuss the related works in Section VI, and finally, provide concluding remarks with a vision of the future work in Section VII.

II. System Overview

At present, two most well-known multi-level checkpoint/restart toolkits are SCR [15] and FTI [16], and they adopt similar design models. There are multiple checkpoint levels, each of which corresponds to a particular type of failure event. One can perform different levels of checkpoints onto running MPI processes, and restart them at a particular level upon different types of failures. In this paper, we adopt FTI as the prototype of our multi-level checkpoint/restart model, in that FTI allows to restart MPI processes upon multiple simultaneously failed nodes by leveraging Reed-Solomon coding (RS-coding) technology [17].

With RS-coding technology, FTI [16] is able to quickly recover an MPI execution, even though half number of checkpoint files are missing due to crashed nodes in the worst case. In addition to RS-coding technology, FTI also integrates partner-copy method [16], [18] which has lower checkpoint/restart overhead than RS-encoding. Hence, there are 4 levels in FTI based on different storage technologies: local storage, partner-copy, RS-encoding, and PFS.

Figure 1 presents the system architecture of the HPC execution environment deployed with FTI. Based on this figure, the whole HPC architecture can be split into five layers. Physical hardware infrastructure (e.g., execution nodes and network) serves as the bottom layer, on top of which is the resource allocation layer like Parallel Batch System (PBS). FTI is a particular library or toolkit which supports multi-level checkpoint/restart interfaces for MPI library. Any MPI programmer just needs to add a few FTI function calls (or annotations) into programs to specify the heap or stack memories to save in case of failures. Unlike the system level checkpoint tool (e.g., Berkeley Lab Checkpoint/Restart (BLCR) [19]), such a design with inserted annotations in MPI programs can effectively reduce checkpoint overheads because of on-demand critical memory states to store.

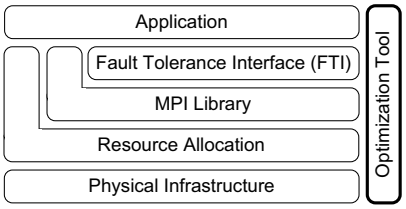


Figure 1. System Architecture Deployed with FTI

In this paper, we focus on how to optimize the checkpoint intervals based on different levels of checkpoint/restart technologies, shown as the *optimization tool* in Figure 1. This optimization work is non-trivial in that it has to be related to different layers in the whole system. We also need to construct a mathematical model which can fit the real checkpoint/restart cases, and try to find the optimal solution with in-depth analysis on the multi-level checkpoint model.

III. Problem Formulation

In this section, we formulate our research as an optimization problem based on multi-level checkpoint/restart model. We mainly focus on the periodic checkpoint model (a.k.a., equidistant checkpoint model), as it is a de-facto standard in the fault-tolerance research. Although a hybrid model with both periodic checkpoints and proactive checkpoints [20], [21] has been studied recently, it requires practitioners to predict the locations of failures with a high precision, which would be non-trivial in practice.

Suppose we are given a large scale MPI program, whose processes are running on K separate processors like nodes. Without loss of generality, transient failures (level 1) are independent with hardware failures (other levels), since their root causes lie in different layers (application layer and physical infrastructure layer) as shown in Figure 1.

In general, an application user can estimate with a certain high precision his/her own HPC application length (denoted

as T_e) based on experiential analysis. Note that such a length refers to the productive time, excluding any failure-related costs like roll-back loss and checkpoint overhead.

There are two types of basic failures, transient failure (a.k.a., software failure) and hardware failure. The expected number of transient failures (denoted as n_s) during the execution of the HPC application is able to be predicted with the mean time to interruption (MTTI) based on historical statistics or the knowledge of possible interruptions. The hardware failure rate per processor is denoted as λ. That is, for a particular period t, there are about λt hardware failures on a processor.

Here, we propose a generic multi-level checkpoint/restart model with L checkpoint levels. The checkpoint level 1 corresponds to transient failures. The remaining higher checkpoint levels (2,3,$\cdots$,L) correspond to different cases of node failures. It is easy to tune such a model to fit practical cases. In FTI, for example, there are totally four checkpoint levels (local storage, partner-copy, RS-coding, and PFS) as shown in Figure 2. The four checkpoint levels correspond to "no hardware failure", "nonadjacent node failures", "a certain number of node failures with adjacent failure cases", and "the situations that lower levels cannot take over" respectively. Upon any type of hardware failure, the system will reallocate a new resource (such as a compute node) for the interrupted application, and the resource allocation period in our model is a constant period, denoted by A, which is far shorter than application execution time T_e.

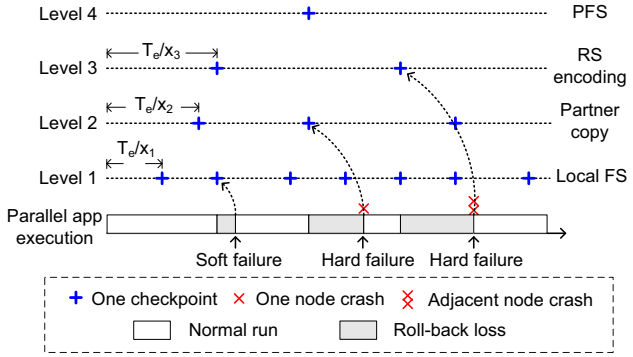


Figure 2. The Process of An HPC Application with Failures

In Figure 2, we present the overall procedure in running an application with three failure cases (software failure, one-node-crash failure, and adjacent-node-crash failure), where x_i refers to the number of checkpoint intervals at checkpoint level i during the normal execution period T_e. For simplicity of presentation, this figure does not present resource allocation period, checkpoint overheads and restart overheads.

The checkpoint overhead and restart overhead are different from level to level. We denote the checkpoint overhead at the checkpoint level i by C_i, where $C_1 \leq C_2 \leq \cdots \leq C_L$ in general. Similarly, the restart overhead at level i is denoted by R_i, where $R_1 \leq R_2 \leq \cdots \leq R_L$ in general.

Optimization of multi-level checkpoint model is non-trivial because of the mutual impact among levels and unpredictable failure locations on particular levels. In order to make the problem tractable, we introduce a key random variable called the number of failures (denoted by Y) during the application's execution, as opposed to other common metrics like interval between failures.

Our objective is to minimize the expected wall-clock length for each given application. The expected wall-clock length $E(T_w)$ can be written as Formula (1), where L, Γ_{ij} and $P_i(Y=N)$ denote the total number of checkpoint levels, the roll-back loss due to the jth failure in the execution and the probability of experiencing N failures[1] at checkpoint level i respectively. On one hand, we need to determine the optimal values of x_1,x_2,$\cdots$,x_L with minimized $E(T_w)$ regarding all possible overheads and roll-back losses. On the other hand, we need to determine which levels are supposed to be selected/removed for the optimal performance.

$$E(T_w) = T_e + \sum_{i=1}^{L} C_i(x_i - 1) + \sum_{i=1}^{L}\left[\sum_{N=1}^{\infty}\left(P_i(Y=N)\sum_{j=1}^{N}(\Gamma_{ij} + A + R_i)\right)\right] \quad (1)$$

Some key notations are summarized in Table I.

Table I
SUMMARY OF KEY NOTATIONS

Notation	Description
L	number of checkpoint levels each with different failure types
K	number of processes of the studied application
T_e	the execution length of the studied application
n_s	expected number of soft failures of the studied application
λ	hardware failure rate per node (a constant)
C_i	checkpoint overhead per checkpoint at level i (a constant)
R_i	restart overhead per recovery at level i (a constant)
A	resource allocation period (a constant)
x_i	number of checkpoint intervals of the application at level i
$P_i(Y=N)$	the probability of encountering N failures at level i
Γ_{ij}	roll-back loss of the application due to jth failure at level i

IV. Optimizing Multi-level Checkpoint Model

The roll-back loss in multi-level checkpoint model is different from that in traditional single-level checkpoint model. As the application is restarted based on a checkpoint at level i, the total roll-back loss has to include all checkpoint overheads at lower levels in addition to the lost execution time. For the example shown in Figure 2, when the application rolls back to a level-3 checkpoint, both of the level-1 checkpoint overheads and level-2 checkpoint overheads should also be counted in the roll-back loss. On the other hand, since all failures are unpredictable with random arrival locations and the checkpoints are taken periodically with equal distances, the expected roll-back execution time without considering checkpoint overheads should approach half of the checkpoint interval, i.e., $\frac{T_e}{2x_i}$. As such, the expected value of Γ_{ij} can be represented in Formula (2), where

[1]The N failures here are evaluated with respect to productive time. If the productive time is long, then it is close to wall-clock time, so $N \approx \frac{T_e}{T_{fi}}$, where T_{fi} refers to mean time between failures (MTBF) at level i.

$\frac{T_e/(2x_i)}{T_e/x_k}$ refers to the number of checkpoints at level k during a roll-back period.

$$E(\Gamma_{ij}) = \frac{T_e}{2x_i} + \sum_{k=1}^{i-1}\left(\frac{T_e/(2x_i)}{T_e/x_k}C_k\right) = \frac{T_e}{2x_i} + \sum_{k=1}^{i-1}\frac{C_k x_k}{2x_i} \quad (2)$$

Since the unpredictable hardware failures occur randomly during the execution, the roll back loss in each checkpoint interval is supposed to be independent with the total number of failures. As a result, we can convert the expected wall-clock time to be Formula (3), where n_i (i=1,2,$\cdots$,L) refers to the expected number of failures at checkpoint level i during the execution.

$$E(T_w) = T_e + \sum_{i=1}^{L} C_i(x_i - 1) + \sum_{i=1}^{L}\left(\frac{T_e}{2x_i} + \sum_{j=1}^{i-1}\frac{C_j x_j}{2x_i} + A + R_i\right)\cdot n_i \quad (3)$$

Our objective is to minimize $E(T_w)$ based on the above formula with variables x_i (i=1,2,$\cdots$,L). Since $\forall i$ $\frac{\partial^2 E(T_w)}{x_i^2} > 0$, there must exist an unique minimum point, when $\frac{\partial E(T_w)}{x_i} = 0$. Hence, we can derive L equations, i.e., Formula (4), where i=1,2,$\cdots$,L. That is, we can get the optimal checkpoint intervals for different levels as long as such simultaneous equations are solved.

$$C_i = \frac{n_i}{2x_i^2}\left(T_e + \sum\nolimits_{j=1}^{i-1} C_j x_j\right) - \frac{C_i}{2}\sum\nolimits_{j=i+1}^{L}\frac{n_j}{x_j} \quad (4)$$

However, it is non-trivial to directly solve such simultaneous equations due to two key factors.

- According to Abel-Ruffini theorem (a.k.a., Abel's impossibility theorem) [22], there is no generic formula to directly solve a system of quintic equations with single variable, not alone to say the above simultaneous equations (Formula (4)) with higher degrees and multiple variables x_i (i=1,2,$\cdots$,L).
- We need to compute the expected number of failure events[1] for different checkpoint levels (i.e., n_i). Which levels the failure events should belong to are determined by the mutual occurrence intervals between hardware failures or the state of node connection. As shown in Figure 3, a *failure event* is considered higher-level failure if and only if there are multiple nodes crashing simultaneously in a short period. Such a failure event in literatures is also known as *correlated failure event* [8], [23], [24]. The short period (a.k.a., correlated failure window) can be set to a constant overhead, such as resource allocation period. For example, in [23] and [24], the correlated failure windows are set to 1 minute and 2 minutes respectively. In Figure 3, the first blue cross implies a soft failure due to no hardware failures occurring around it, so it belongs to the checkpoint level 1. When there are two nonadjacent node failures occurring during the resource allocation, the failure event belongs to level 2, while a failure event with three simultaneous hardware failures will be considered checkpoint level 4 (e.g., CPL4 in the figure). On the other hand, simultaneous node failures may occur due to the malfunctioned switch or power board shared by multiple nodes.

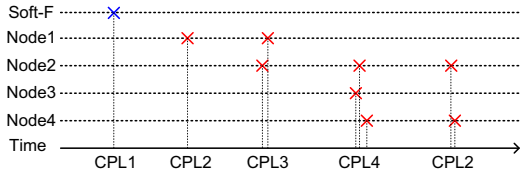


Figure 3. Illustration of CheckPoint Levels (CPL)

In the following text, we first present a very effective method to iteratively obtain the solution to the above simultaneous equations (4). We then describe how to optimize the selection of checkpoint levels based on such an iterative method. After that, we use a real use-case (FTI prototype) to illustrate how to extend our multi-level checkpoint/restart model and compute the optimal solution in practice. Finally, we illuminate some ideas on how to derive the expected number of failure events belonging to different checkpoint levels, based on different types of failures.

A. Optimizing Checkpoint Intervals for Different Levels

As analyzed previously, the solution to the simultaneous equations presented by Formula (4) leads to the optimal checkpoint intervals for different checkpoint levels. That is, we need to find a set of values for $\{x_1,x_2,\cdots,x_L\}$ such that Equation (4) always holds for i=1,2,$\cdots$,L. In fact, Abel-Ruffini theorem [22] reveals that there is no formula to get the solution.

Fortunately, we find an iterative method which is able to solve the Equations (4) very quickly, with just a few iterative steps. The idea is similar to Jacobi method [25] which is commonly used to solve a system of linear equations.

It is easy to convert Formula (4) to Formula (5).

$$x_i = \sqrt{\frac{n_i(T_e + \sum_{j=1}^{i-1} C_j x_j)}{C_i(2 + \sum_{j=i+1}^{L}\frac{n_j}{x_j})}} \quad (5)$$

Based on Formula (5), we can further derive an iterative function shown below, where $x_i^{(k)}$ and $x_i^{(k+1)}$ are referred to as the converged results at the kth step and the (k+1)th step respectively.

$$x_i^{(k+1)} = \sqrt{\frac{n_i(T_e + \sum_{j=1}^{i-1} C_j x_j^{(k)})}{C_i(2 + \sum_{j=i+1}^{L}\frac{n_j}{x_j^{(k)}})}} \quad (6)$$

Similar to Jacobi method, we first set the variables $\{x_1,x_2,\cdots,x_L\}$ to some initial values, and then iteratively perform Formula (6) until each equation in Formula (4) approximately holds with little errors. The initial values of $\{x_1,x_2,\cdots,x_L\}$ can be obtained by Formula (7), in that it leads to the sub-optimal checkpoint interval result for a

[1]A failure event is a failure case with one or more simultaneous hardware failures occurring in a short period before restarting execution.

particular level i by overlooking the impact of checkpoint overheads at other levels.

$$x_i^{(0)} = \sqrt{\frac{n_i T_e}{2C_i}} \tag{7}$$

In fact, Young's formula [5] can be considered a special case of Formula (7). That is, when assuming that failure intervals follow Exponential distribution and the checkpoint overhead is very small, then we can get Young's formula from Formula (7) through the following derivation, where T_{ci} and T_{fi} refer to the checkpoint interval at level i and the mean time between failures (MTBF) at level i respectively. $T_{ci} \approx \frac{T_e}{x_i^{(0)}} = \frac{T_e}{\sqrt{\frac{1}{2} n_i T_e / C_i}} = \frac{T_e}{\sqrt{\frac{1}{2} \frac{T_e}{T_{fi}} \cdot T_e / C_i}} = \sqrt{2C_i T_{fi}}$

Suppose there are millions of running processes with 8 checkpoint levels (from level 1 to level 8): their checkpoint overheads are 10,30,45,50,55,60,65,240 seconds and the expected number of failures are 30,10,20,25,18,15,8,2 (transient failures at the first level are independent with hardware failures and the last checkpoint level is PFS). When the productive time T_e is set to 1000, 5000, and 9000 seconds, the total loss (excluding productive time) induced by Formula (7) is higher (worse) than that of Formula (6) by 17.8%, 5.6%, and 4.2% respectively. The numbers of iterative steps used in our iterative method (error threshold=10^{-6}) are 27, 21, and 19 respectively, confirming fairly high efficiency of our iterative algorithm.

B. Optimizing Selection of Checkpoint Levels

Considering the non-negligible checkpoint overheads at different checkpoint levels, it is likely that the overall execution performance can be further improved if we remove some lower levels. For example, if checkpoint costs at level 1 and level 2 are comparative, we can try removing level 1 such that all previous failures belonging to this level will be rolled back to the most recent level-2 checkpoints. Due to decreased total number of checkpoints in the system, the overall checkpoint overhead will decrease, which may improve the entire performance in turn.

Our basic idea of finding the optimal selection of levels, is traversing all of possible cases each with a specific combination of checkpoint levels and compute optimal checkpoint intervals for each case. The case with shortest wall-clock length estimated will serve as the final optimal solution.

It is plausible that such an approach in finding the optimal combination of checkpoint levels may suffer from high computation complexity due to exponential number (2^L) of possible cases. In effect, the practical time complexity is still very low due to two factors. On one hand, the number of checkpoint levels is small in general. For example, there are no more than four checkpoint levels in both FTI and SCR. On the other hand, there are few iterative steps in computing the optimal checkpoint levels for each case. As for the example in the last subsection, there are 8 checkpoint levels, and it just requires at most 30 iterative steps to obtain a converged result (with maximum error = 10^{-6}). There are at most 2^8-1=255 different possible combinations of different levels selected, thus there are at most 255×30=7650 iterations in total, which means a fairly low computation complexity.

C. Analysis of A Practical Case − FTI

We here illustrate how to make our multi-level checkpoint model fit a real implementation - FTI [16], and how to compute the optimal multi-level checkpoint intervals for it.

In FTI, all nodes are split into several groups and there are totally four checkpoint levels for each group: local storage (no hardware failure), partner-copy (nonadjacent node failures), RS-encoding (adjacent node failures with $\leq M$ failed nodes), and PFS (adjacent node failures with M+ failed nodes). M is the number of RS-encoding files to be generated per group during execution, thus M is the maximum number of node failures RS-coding technology can tolerate in a group. That is, even with M crashed processors during the execution, the M missing checkpoint files can still be restored based on the remaining valid checkpoint files and encoding files. As for the four checkpoint levels, their checkpoint overheads are denoted as C_{lf}, C_{pc}, C_{rs}, and C_{pf} respectively. Similarly, their restart overheads are denoted by R_{lf}, R_{pc}, R_{pc}, and R_{pf} respectively.

The target expected wall-clock length based on the FTI four-level checkpoint model can be written as follows.

$$\begin{aligned} E(T_w) = {} & T_e + C_{lf}(x_{lf}-1) + C_{pc}(x_{pc}-1) \\ & + C_{rs}(x_{rs}-1) + C_{pf}(x_{pf}-1) \\ & + (\tfrac{T_e}{2x_{lf}} + R_{lf}) n_{lf} \\ & + (\tfrac{T_e}{2x_{pc}} + \tfrac{C_{lf}x_{lf}}{2x_{pc}} + R_{pc}) n_{pc} \\ & + (\tfrac{T_e}{2x_{rs}} + \tfrac{C_{lf}x_{lf}}{2x_{rs}} + \tfrac{C_{pc}x_{pc}}{2x_{rs}} + R_{rs}) n_{rs} \\ & + (\tfrac{T_e}{2x_{pf}} + \tfrac{C_{lf}x_{lf}}{2x_{pf}} + \tfrac{C_{pc}x_{pc}}{2x_{pf}} + \tfrac{C_{rs}x_{rs}}{2x_{pf}} + R_{pf}) n_{pf} \end{aligned} \tag{8}$$

The optimal checkpoint intervals should be the solution of Equations (9), where n_{lf}, n_{pc}, n_{rs}, and n_{pf} are expected numbers of failure events handled by different levels.

$$\begin{cases} C_{lf} = \frac{n_{lf}}{2x_{lf}^2} T_e - \frac{C_{lf}}{2} \frac{n_{pc}}{x_{pc}} - \frac{C_{lf}}{2} \frac{n_{rs}}{x_{rs}} - \frac{C_{lf}}{2} \frac{n_{pf}}{x_{pf}} \\ C_{pc} = \frac{n_{pc}}{2x_{pc}^2} (T_e + C_{lf} x_{lf}) - \frac{C_{pc}}{2} \frac{n_{rs}}{x_{rs}} - \frac{C_{pc}}{2} \frac{n_{pf}}{x_{pf}} \\ C_{rs} = \frac{n_{rs}}{2x_{rs}^2} (T_e + C_{lf} x_{lf} + C_{pc} x_{pc}) - \frac{C_{rs}}{2} \frac{n_{pf}}{x_{pf}} \\ C_{pf} = \frac{n_{pf}}{2x_{pf}^2} (T_e + C_{lf} x_{lf} + C_{pc} x_{pc} + C_{rs} x_{rs}) \end{cases} \tag{9}$$

Finally, the optimal multi-level checkpoint intervals $\{x_{lf}^*, x_{pc}^*, x_{rs}^*, x_{pf}^*\}$ can be effectively obtained by our iterative method based on Formula (6).

D. Estimating Expected Number of Failure Events

There are two cases that nodes fail simultaneously. The first one (called failure case A) is that the different nodes fail consecutively in a short period (i.e., during resource allocation duration) before the uniform recovery/restart operation. In this case, we assume each node failure occurs

independently. The second one (called failure case B) is due to tightly-coupled structures inside the supercomputer. For example, when a switch or an outlet power board is malfunctioned, all of nodes connected to it will be disconnected or crashed. We will discuss the two cases respectively and derive the expected numbers of failure events for different checkpoint levels based on the above FTI design model.

1) Analysis of Failure Case A: In FTI [16], there are totally four checkpoint levels based on different types of failures. Since software failures are independent with hardware failures, the expected number (n_1=n_s) of software failures must be independent with that of hardware failures (n_i, i=2,3,4). In the following, we intensively derive the expected numbers of failure events for the other three checkpoint levels, partner-copy (level 2), RS-coding (level 3) and PFS (level 4), based on given hardware failure rate λ per processor per time unit. For simplicity, we assume that there is no iterative failure propagation issue, i.e., there are no new hardware failures occurring in a restart period R_i after the resource allocation step.

There are three steps in our estimate of expected number of failures for specific levels.

1) Compute the expected total number of failure events (denoted as y) during the application execution.
2) Compute the probability of one failure event belonging to a particular checkpoint level i, denoted as P_i.
3) Since each failure event occurs independently, we can treat them Bernoulli trials, thus the expected number of failure events at checkpoint level i is equal to $y \cdot P_i$.

In the following, we illuminate how to derive y and P_i.

- To derive expected total number of failure events y

With respect to a single processor, λA is supposed to be very small, where A is the resource allocation period, then we can derive that the probability that there exist failures during A is approximately equal to λA. In fact, according to [12], the number of hardware failures per year per processor is about 0.2-0.7. Suppose the resource allocation period is 60 seconds, then, $\lambda A \approx \frac{0.7}{360*24*60} \approx 1.35 \times 10^{-6}$. Due to "law of rare events" [26], the failure probability follows a Poisson distribution. Then, the probability that there do not exist any failures in a short interval A (such as 1 minute) is equal to $e^{-\lambda A} \approx 1-\lambda A$. That is, the probability that there exists at least one failure during A approaches λA.

Since K processors are independent with each other, the expected number of hardware failures during a resource allocation period A is approximately equal to $K\lambda A$. Similarly, the total expected number of hardware failures during the whole execution among the K processors is $\sum_{j=1}^{K} \lambda T_w = K\lambda T_w$, where T_w is application running time. Hence, $y = \frac{K\lambda T_w}{K\lambda A} = \frac{T_w}{A}$.

- To derive the probability P_i **of one failure event belonging to level** i

We derive the value of P_i based on three cases, partner-copy, RS-coding, and PFS respectively.

- *Partner-copy* is widely used in the fault-tolerance of exascale HPC applications [16], [18]. Figure 4 shows the basic principle of partner-copy technology. In a nut shell, partner-copy maintains two copies of the checkpoint file for each process of the application, on local disk and on a partner-node disk respectively. Obviously, upon a failure event with multiple simultaneous hardware failures, the whole execution cannot be restarted through the partner-copy method if and only if there exist two adjacent nodes crashing.

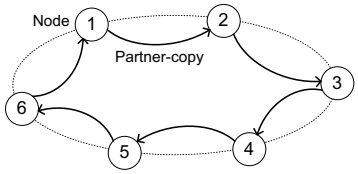


Figure 4. Principle of Partner-copy Technology

The probability of failure event belonging to the parter-copy level can be derived as follows. Upon a node failure occurring, the system will allocate one or more available nodes to the interrupted application. The hardware failure event belongs to the partner-copy checkpoint level if and only if there are no adjacent failed nodes in the end of the resource allocation period. For the example in Figure 4, suppose Node 1 is the initial failed node, then the hardware failures striking Node 2, Node 6, or any other two adjacent nodes (e.g., node 3 and node 4) will make the whole failure event beyond the partner-copy level. Recall that the probability of a node failing in the resource allocation period with a length of A is λA, so the probability of the failure event belonging to partner-copy level is:

$$\begin{aligned} P_{\text{pc}} &= 1 - \{2\lambda A(1-\lambda A) + (\lambda A)^2 \\ &+ (1-\lambda A)^2 \left[(K-4)(\lambda A)^2(1-\lambda A)\right]\} \\ &= 1 - \lambda A(2 - \lambda A + \lambda A(1-\lambda A)^3(K-4)) \end{aligned} \quad (10)$$

Obviously, in the situation with independent node failures, P_{pc} is very close to 1 when λA is extremely small (e.g., in the order of 10^{-6}).

- *RS-encoding* can tolerate the adjacent node failures, yet it has an upper-bound on the number of node failures, which is equal to the number M of extra RS-encoding files generated per checkpoint. Hence, we need to focus on the node-failure situation that partner-copy level cannot take over and there are no more than M failed nodes in the failure event. Based on the discussion about partner-copy, the probability of the failure event handled by RS-encoding is: P_{rs} = $2\lambda A(1-\lambda A)\sum_{j=0}^{M-2} P(K-3,j) + (\lambda A)^2 \sum_{j=0}^{M-3} P(K-3,j)$, where $P(K,j)$ is the probability with j node failures occurring during the resource allocation period A when

an application is running with K mutually-independent nodes, derived in Formula (11).

$$P(K,j) = \binom{K}{j} (\lambda A)^j (1-\lambda A)^{K-j} \qquad (11)$$

- *PFS* is used to tolerate the failure events with multiple node failures that cannot be restored using lower levels like partner-copy or RS-encoding. So, the probability of such failure events is: $P_{\text{pf}} = 1 - P_{\text{pc}} - P_{\text{rs}}$.

2) Analysis of Failure Case B: As discussed above, if the node failures occur independently, the probability of the correlated failures occurring beyond the partner-copy level is very low. Whereas, in practice, the compute nodes in a supercomputer are always tightly-coupled, which leads to a potentially strong correlations of failure events among them.

In this subsection, we analyze the probability and expected number of correlated failures based on Failure Case B, which takes into account the possible correlations among compute nodes. Our key concern is whether the partner-copy takes a dominant role as in the case with independent nodes. Hence, we mainly study how to estimate the expected number of failure events for the partner-copy checkpoint level based on Failure Case B.

Partner-copy technology saves each process's checkpoint file on both local host disk and its adjacent node disk. Hence, partner-copy can tolerate multiple hardware failures unless some node and its adjacent node fail together.

In general, all of nodes are organized in a very tightly-coupled pair-wise manner. Dongarra's recent report [27] shows that the totally 16,000 nodes in Tianhe-2 (the most powerful supercomputer currently) are placed in 8,000 compute blades (i.e., two nodes per blade). Its entire network adopts a fat-tree topology to connect all of nodes, connected through $13 \times 576 \approx 7500$ ports, which means a malfunction of any one port will lead to about two adjacent-node failures. As for the world-wide rank-2nd supercomputer Titan [28], the network is organized as Torus topology composed of $25 \times 16 \times 24$=9,600 *Gemini* components. Each such component combines two nodes via an embedded 48-port router, which means the adjacent node failure will arise whenever a Gemini component encounters an error.

Suppose there are S network-connection components (such as Gemini in Titan) and W power boards in the whole cluster. Denote the failure rate of one connect component by λ_S and denote that of each power board by λ_W. Based on Bernoulli-trial model and Poisson Equation, we can write the probability (denoted P'_{pc}) of experiencing a failure event that partner-copy can handle during the task execution length T_e as Formula (12), where Δt refers to a small time unit and P_{pc} is given in Formula (10).

$$\begin{aligned} P'_{\text{pc}} &= (1-\lambda_S \Delta t)^{\frac{T_e}{\Delta t} S} \cdot (1-\lambda_W \Delta t)^{\frac{T_e}{\Delta t} W} \cdot P_{\text{pc}} \\ &= P_{\text{pc}} e^{-(\lambda_S S + \lambda_W W) T_e} \end{aligned} \qquad (12)$$

Obviously, if an application scale is large enough (e.g., with large values of S and W), it is likely to encounter a failure event beyond the recovery ability of partner-copy level.

V. Performance Evaluation

A. Experimental Setting

Since our fault-tolerance research is designed for large scale (or even exascale) HPC applications, it is best to perform the evaluation over a large scale test-bed with millions of nodes or cores. At the moment, however, we only have up to 128 nodes with totally up to 1024 cores to perform the experiment, so we have to evaluate our solutions by leveraging simulations to a certain extent.

Our evaluation can be split into two types, called *evaluation type A* and *evaluation type B*. Evaluation type A is based on numerical simulation with millions of emulated cores, which can evaluate our solution from the perspective of real system scale. In the simulation, we also take into account the possible prediction errors (with 20% error ratio) in checkpoint/restart cost. For instance, if the checkpoint cost is set to 10 seconds, the real cost used in the simulation is a random value in [8,12] seconds. Evaluation type B is based on a real experimental environment over a cluster with hundreds of cores, and the node failures occur randomly over time in accordance with that on a large scale situation. Evaluation type B can confirm the feasibility of our optimized multi-level checkpoint model in practice. We comprehensively evaluate the solutions with different failure rates, including both the optimistic cases with rare failures and the very pessimistic cases with frequent failures.

The application prototype used in our experiment is a well-known MPI program, *Heat-Distribution*, which is commonly used in real scientific research like Ocean Simulation [29]. This application aims to compute the heat distribution over time based on a set of initial heat sources. Its execution length is determined by a preset expected precision or the simulation length (the number of iterations). The checkpoint cost and recovery cost (or restart cost) are both dependent upon two factors: (1) the program's memory sizes, which are determined by the problem size (such as a square grid with 10000×10000 points or 15000×15000 points); and (2) the execution scale, i.e., the number of processes to raise in parallel. We present in Table II the checkpoint overhead of FTI characterized with hundreds of cores on FUSION cluster [30]. Basically, checkpoint overheads are determined by the group size set by FTI [16] and checkpoint file size to store. The checkpoint file sizes to be stored per process for the problem size 10000×10000 and 15000×15000 are 25MB and 56.6MB respectively. Note that with small execution scale (e.g., 64 cores), the checkpoint overhead on PFS is smaller than that on other lower levels. This is because of the special fast storage device (such as SSD) used by PFS in the FUSION cluster. However, the checkpoint overhead of PFS increases significantly with number of cores used in the execution due to inevitable bottleneck, which means PFS is unsuitable for large scale (or exa-scale) HPC applications.

Table II
CHECKPOINT OVERHEAD OF FTI (IN SECONDS)

Problem Scale	Execution Scale	Ckpt Cost (level 1 – level 4)			
10000×10000	64 cores	3.3	7.5	13.2	4.9
15000×15000	128 cores	4.3	9.8	13.7	6.9
15000×15000	256 cores	3.2	7.1	11.7	12.2
15000×15000	512 cores	5.8	8.3	9.6	20.8
15000×15000	1024 cores	6.6	9.5	10.6	43.1

Two indicators will be mainly used in our evaluation. The first one is called *Workload Processing Ratio* (*WPR*), which is defined as the ratio of the application's productive time to its wall-clock time. The second one is called *total overhead*, which includes all types of overheads like rollback loss and checkpoint/restart cost. Total overhead excludes application's productive time, which can clearly quantify the impact of different checkpointing methods to the overhead.

B. Experimental Results

1) Evaluation based on Simulation: We first compare our optimal solution based on Formula (6) to the suboptimal approach adopting Young's formula separately on multiple levels. Then, we present the different effects when using different selections of levels.

- Iterative Optimal Solution vs. Young's Formula

In Section IV-A, we present that the performance gain of WPR can reach up to 17.8% in that example case with 8 checkpoint levels. In the following, we mainly focus on the practical use cases in terms of FTI [16] and SCR [15]. In our simulation, there are up to 5 potential levels with different types of storage devices: RAM, local disk, partner-copy, RS-encoding, and PFS.

We perform a large number of tests and find that the multi-level checkpoint effect differs with the checkpoint overhead and application execution length. Table III shows that different performance gains are obtained on WPR with the iterative optimal solution based on different representative cases, where *MNOF* refers to Mean (or expected) Number of Failures at some level (e.g., n_i at level i).The first case is allowed to use all of five levels (including RAM disk level) and the other two cases are only allowed to use four higher levels (which is consistent with FTI architecture). It is observed that the improvement of WPR and reduction of total overheads are prominent when the checkpoint overheads are relatively huge compared to the application's productive time, such as the case #2 and #3, where the reduction of total overhead is up to about 8%.

The key reason that our iterative optimal method exhibits better effects is due to the impact of checkpoint overheads across different levels ignored by the suboptimal solutions with separate Young's formula used onto multiple levels, especially when checkpoint overheads are relatively huge.

- Different Selections of Checkpoint Levels

The optimized selection of levels in our design can significantly improve performance. As for the example used in Section IV-A, if the application's productive length is

Table III
ITERATIVE OPTIMAL SOLUTION VS. YOUNG'S FORMULA

Cases	Description of Setting (costs are in seconds)						Ipvt WPR	Rdct Ovhd
		ram	ld	pc	rs	pfs		
Case #1: length= 3 hours	ckpt-cost	2	20	30	40	50	1.33%	1.92%
	restart-cost	2	4	6	8	10		
	MNOF	50	50	30	30	30		
Case #2: length= 3 hours	ckpt-cost	-	40	45	50	55	4.0%	4.8%
	restart-cost	-	5	8	10	13		
	MNOF	-	200	100	80	60		
Case #3: length= 5 min.	ckpt-cost	-	12	13	14	15	6.7%	7.8%
	restart-cost	-	2	3	4	5		
	MNOF	-	30	20	20	20		

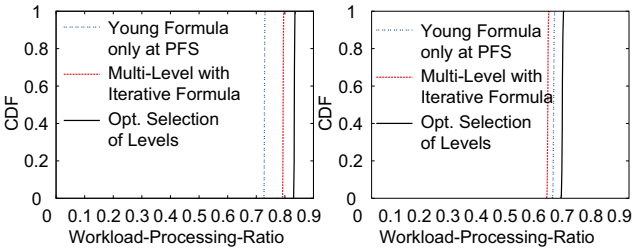
one day, our simulation shows the total overhead (including checkpoint/restart cost and rollback loss) using our iterative method with all 8 levels is 93712 seconds. In contrast, our algorithm shows that the optimal solution should just select level 7 and level 8, and the entire overhead can be reduced down to 47399 seconds (being reduced by about 50%).

We also evaluate the performance with different selections of checkpoint levels in two typical cases based on FTI multi-level checkpoint model. The settings of the two cases are shown in Table IV. This table also presents the solutions (i.e., the checkpoint interval lengths for different levels) based on different methods.

Table IV
SIMULATION SETTINGS AND OPTIMAL CKPT INTERVALS

Case	Metric	Level (1-4)			
Case #A: length= 12 hours	checkpoint-cost	8	10	80	90
	restart-cost	8	10	80	90
	Mean Num of Fails (MNOF)	20	30	5	2
	Young Form. only on PFS (sec)	0	0	0	369
	Opt. Sol. with all Levels (sec)	201	179	1178	1884
	Opt. Sel. of Levels (sec)	0	137	0	1054
Case #B: length= 12 hours	checkpoint-cost	1	20	60	70
	restart-cost	1	10	30	35
	Mean Num of Fails (MNOF)	50	50	40	30
	Young Form. only on PFS (sec)	0	0	0	189
	Opt. Sol. with all Levels (sec)	50	214	382	425
	Opt. Sel. of Levels (sec)	47.6	0	0	225

Figure 5 presents the Cumulative Distribution Function (CDF) of WPR of the optimized checkpointing method with different selections of levels. We observe that all curves look relatively vertical, which means that for a particular case, the evaluation with 20% prediction errors on checkpoint/restart cost lead to very close results. On the other hand, it is



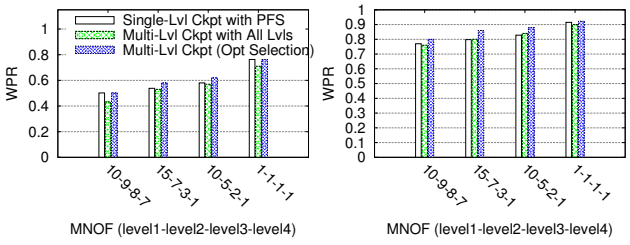
(a) Case A (b) Case B

Figure 5. Evaluation Results with Different Selections of Levels

observed that the multi-level checkpoint model with all possible levels does not always outperform single-level checkpoint model with only PFS. The former outperforms the latter by about $\frac{0.8-0.725}{0.725} \approx 10\%$ in Case A while the

latter outperforms the former by $\frac{0.63-0.61}{0.61}$=3.3% in Case B. However, the optimal multi-level checkpoint model with optimized selection of levels will always exhibit best results, which outperforms other two solutions by about 10-20%.

2) Evaluation based on Real Experiment: Figure 6 presents the WPR of running HeatDistribution MPI program on 512 cores and 1024 cores respectively, based on different failure rates at various levels. The productive times are about 1300 seconds and 12 hours respectively. It is observed that our solution with optimal selection of levels always exhibit the best effects, outperforming the optimized solution with all levels by about 5-10% in general. The single level checkpoint effect approaches the optimal-selection solution in Figure 6 (a) due to comparative checkpoint overheads among different levels. For example, as shown in Table II, when using 256 cores to run the program, the checkpoint overheads among levels are not largely different; the checkpoint overhead at level 4 is even smaller than that of level 3. This will cause the optimal solution will only select one or two levels eventually based on our algorithm.



(a) 512 cores (T_e=1300 secs) (b) 1024 cores (T_e=12 hours)

Figure 6. Evaluation Results on A Real Cluster

VI. Related Work

With ever-increasing demand on large scale or even exascale systems, the fault tolerance issue has been extensively studied recently [7], [8], [9], [31], [18]. Checkpoint-restart [9], [16], [15] is still a classic and effective model, as long as the checkpoint overheads can be properly controlled, e.g., diskless checkpoint [14], [18]. In addition to checkpoint-restart model, an alternative in attaining exascale computing fault tolerance is process replication [31], which generates multiple replica processes or machine states in case of failures. These two models are relatively orthogonal to each other, which means that one can combine them in practice. In this paper, we focus on checkpoint/restart model and optimize checkpoint intervals for multiple levels.

SCR [15] is the first library based on multi-level checkpoint/restart mechanism and also explores a Markov model to fit it. There are four differences between SCR and our work. (1) The Markov model used by SCR assumes that the failure rates at different checkpoint levels are completely independent, while the failure probabilities among checkpoint levels (i=2,$\cdots$,L) are actually correlated to a certain extent, as we discussed in Section IV-D. For example, the RS-encoding level can only cope with the failure cases that partner-copy level cannot take over. (2) SCR assumes that the failure rates at different levels are all pre-known beforehand, while we analyze the failure probability for each level and expected number of failure events belonging to different levels in depth. (3) We propose a more generic multi-level checkpoint/restart model which can suit the checkpoint demand with various numbers of simultaneous node failures, and also optimize the checkpoint intervals for each level via a very effective iterative method. (4) We optimize the selection of levels to further improve the performance.

There are some other models [20], [21] related to multi-level checkpoint/restart mechanism, and they mainly focus on the advantage of combining the periodic checkpoint and proactive checkpoint (a hybrid model). In our previous work [20] and Aupy et al.'s work [21], optimal proactive checkpoint locations are derived based on such a hybrid model. However, both works did not discuss how to optimize the periodic checkpoint intervals for different levels, but just adopt Young's formula [5] for simplicity. In our paper, we prove that Young's formula used on different levels separately is just a sub-optimal approach with higher total overheads than our solution.

VII. Conclusion and Future Work

In this paper, we devised a novel multi-level periodic checkpoint model based on various types of location-unpredictable failures and proposed an iterative method to find optimal checkpoint intervals. We confirm our iterative method can converge very quickly with only a few iterations. We also optimize the selection of checkpoint levels to further improve the performance. We not only studied a real-world case - FTI, but also analyzed in depth the expected number of simultaneous/correlated failure events in the course of task execution. Some key findings about our experiments are listed below.

- Our iterative optimal solution outperforms Young's formula by 1.92-8%. The performance gains differ with checkpoint overheads and application length.
- The optimization of the selection of levels can further improve the performance by 10-20% based on FTI scenario, and by up to 50% based on 8-level checkpoint/restart model.
- Using a real experiment on a cluster environment with 512 cores and 1024 cores, our solution outperforms others by 5-10% in general, which strongly confirms the validity of our solution in practice.

In the future, we plan to evaluate our optimal methods using more types of Peta-Flop parallel applications on various real supercomputers with larger scales.

Acknowledgments

This work was supported by PRACE First implementation Phase (PRACE-1IP) as the AMFT prototype under

contract RI-261557 and by GENCI, also in part by the Advanced Scientific Computing Research Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the ANR RESCUE, ANR G8 ECS projects and the INRIA-Illinois Joint Laboratory for Petascale Computing.

REFERENCES

[1] A. Feinberg, "An 83,000-Processor Supercomputer Can Only Match 1% of Your Brain," online at http://gizmodo.com/an-83-000-processor-supercomputer-only-matched-one-perc-1045026757.

[2] J. Dongarra et al. The International Exascale Software Project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, February 2011.

[3] J.N. Glosli, K.J. Caspersen, J.A. Gunnels, D.F. Richards, R.E. Rudd, and F.H. Streitz, "Extending Stability Beyond CPU Millennium: A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'07)*, 2007, pp. 1–11.

[4] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," in *International Journal of High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 374–388, 2009.

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

[6] J.T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," in *Future Generation Computater Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[7] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, Y. and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 1–11.

[8] K. Pattabiraman, C. Vick, and A. Wood, "Modeling Coordinated Checkpointing for Large-Scale Supercomputers," in *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, 2005, pp. 812–821.

[9] K. Ferreira, "Keeping Checkpoint/Restart Viable for Exascale Systems," Ph.D Thesis, Computer Science, University of New Mexico, 2011.

[10] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-scale HPC Systems," in *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'12)*, 2012, pp. 1530-2075.

[11] E. Vivek Sarkar, et al., "Exascale Software Study: Software Challenges in Exascale Systems, " technical report, 2009.

[12] B. Schroeder and G. Gibson, "Understanding Failure in Petascale Computers," in *Journal of Physics Conference Series: SciDAC*, vol. 78, pp. 11–22, June 2007.

[13] L. Bautista-Gomez, A. Nukada, N. Maruyama, and F. Cappello, and B. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems,", in *Proceedings of International Conference on High Performance Computing (HiPC'10)*, 2010, pp. 1-10.

[14] L. Bautista-Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed Diskless Checkpoint for Large Scale Systems," in *proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*, 2010, pp.63-72.

[15] A. Moody, G. Bronevetsky, K. Mohror, and B.R. Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010, pp. 1-11.

[16] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, and N. Maruyama, S. Matsuoka, "FTI: high performance fault tolerance interface for hybrid systems," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 32:1–32:32.

[17] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," in *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp.300–304, 1960.

[18] G. Zheng, X. Ni, and L.V. Kale, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *EEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W12)*, 2012, pp 1-6.

[19] P.H. Hargrove and J.C. Duell, "Berkeley lab checkpoint/restart (BLCR) for Linux clusters", in *Journal of Physics: Conference Series*, vol. 46, no. 1, pp. 494, 2006.

[20] M.S. Bouguerra, A. Gainaru, L.B. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing," in *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'13)*, 2013.

[21] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," Research Report RR-8237, INRIA, February 2013.

[22] E. Dehn, *Algebraic equations: an introduction to the theories of Lagrange and Galois*. Columbia University Press, 1930.

[23] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis(SC'11)*, 2011, pp. 45:1-45:12.

[24] D. Ford, F. Labelle, F.I. Popovici, M. Stokely, Murray, V.A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'11)*

[25] Jacobi method in solving linear equestions: on line at http://en.wikipedia.org/wiki/Jacobi_method

[26] M. Falk, J. Huesler, and R.D. Reiss, "Laws of Small Numbers: Extremes and Rare Events, " in *Birkhauser, Boston*, 1994.

[27] J. Dongarra, "Visit to the National University for Defense Technology Changsha, China," technical report, 2013, online at http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf.

[28] Buddy Bland, "Titan - Early experience with the Titan system at Oak Ridge National Laboratory," sccompanion, in *SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 2189-2211.

[29] R. Smith, et al. "The Parallel Ocean Program (POP) Reference Manual: Ocean Component of the Community Climate System Model (CCSM)," technical report, Los Alamos National Laboratory (LAUR-10-01853), 2010.

[30] FUSION Cluster: http://www.lcrc.anl.gov/

[31] K. Ferreira, et al., "Evaluating the viability of process replication reliability for exascale systems," in *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 44:1-44:12.