Data-Driven Application-Oriented Reliability Model of a High-Performance Computing System

Bentolhoda Jafary, Saurabh Jha, Lance Fiondella, *Member, IEEE*, and Ravishankar K. Iyer, *Fellow, IEEE*

***Abstract*—Reliability analysis and performance evaluation are complementary methods to quantify nonfunctional aspects of a system. However, a range of factors such as concurrency and heterogeneity quickly exacerbate the state-space explosion problem when attempting detailed system-level modeling and simulation of high-performance computing (HPC) systems. To overcome these impediments to modeling and analysis, this article develops a hierarchical model of an application that implements checkpointing running in an HPC environment subject to application, network, and system-wide outages. The modeling approach ensures that the number of states is linear in the number of checkpoints and possesses a low constant factor for the number of recovery states most relevant to the external influences contributing to degraded application performance. We illustrate the types of analysis enabled by the model through a series of examples with parameters determined empirically from data logs of the Blue Waters supercomputer located at the University of Illinois at Urbana–Champaign. A comprehensive comparative analysis of the model parameters indicates that lowering the failure rate of network nodes would most significantly reduce application downtime. We also discuss how the modeling approach can be used to objectively assess both current and hypothetical future systems to identify competitive designs and enhancements.**

***Index Terms*—Application performance, application reliability, high-performance computing (HPC), network outage, utilization.**

NOMENCLATURE

ℓ	Number of intermediate checkpoints.
τ	Compute time between successive checkpoints.
t_n	Computing time required of application.
n_d	Number of compute nodes assigned to application.
n_n	Number of network nodes assigned to application.
n_b	Number of blades assigned to application.
n_c	Number of cabinets assigned to application.
$R_d(t)$	Reliability of compute node.
$R_n(t)$	Reliability of network node.
$R_l(t)$	Reliability of link.
$R_b(t)$	Reliability of blade.
$R_c(t)$	Reliability of cabinet.
m	Number of the application running in a time interval.
T_{total}	Total time to run m applications.
NA	Network and application working.
$N\overline{A}$	Network working and application recovery.
$\overline{N}A$	Network recovery and application working.
$\overline{NA}$	Network and application recovery.
λ_i	Failure rate.
k	Number of retries.
$p_{i_{NA},(i+1)_{NA}}$	Probability of reaching the next checkpoint in system working state.
$p_{i_{NA},i_{N\overline{A}}}$	System working to application recovery.
$p_{i_{NA},i_{\overline{N}A}}$	System working to network recovery.
$p_{i_{NA},i_{\overline{NA}}}$	System working to network and application recovery.
$p_{k-j,i_{NA}}$	Probability of retry to system recovery.
$p_{k-j,k-(j+1)}$	Probability of retry to the next retry.
$p_{k-j,k}$	Probability of restart the retry attempts.
$p_{k-j,i_{\overline{NA}}}$	Probability of retry to network and application recovery.
$p_{nbc}(\tau)$	Network nodes, blades, and cabinets allocated to the application work.
$p_{R_d R_{N\notin A_i}}$	Probability of compute node, network nodes, blades, links, and cabinets not allocated to the application network nodes, blades, and cabinets work.
$p_{F_d R_{N\notin A_i}}$	Probability of compute node fails but network nodes, blades, links, and cabinets not allocated to the application network nodes, blades, and cabinets work.
$\rho_{i_{NA}}$	Total time spent in a working state.
$E[H(\tau)]$	Mean holding time.
ρ_c	Time to perform the checkpoint.
$\rho_{i_{N\overline{A}}}$	Total time spent in an application recovery state.
$\rho_{i_{\overline{N}A}}$	Total time spent in a network recovery state.
$\rho_{i_{\overline{NA}}}$	Total time spent in a network and application recovery state.
ρ	Total time.

Manuscript received May 31, 2020; revised November 1, 2020; accepted May 4, 2021. Associate Editor: H. Jiang. *(Corresponding author: Lance Fiondella.)*

Bentolhoda Jafary is with the Department of Electrical and Computer Engineering, University of Massachusetts Dartmouth, Dartmouth, MA 02747 USA and also with AeroVironment, Inc., Simi Valley, CA 93065 USA (e-mail: bjafary@umassd.edu).

Saurabh Jha is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: sjha8@illinois.edu).

Lance Fiondella is with the Department of Electrical and Computer Engineering, University of Massachusetts Dartmouth, Dartmouth, MA 02747 USA (e-mail: lfiondella@umassd.edu).

Ravishankar K. Iyer is with the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: rkiyer@illinois.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3085582.

Digital Object Identifier 10.1109/TR.2021.3085582

I. Introduction

THIS article models failures and recoveries observed on the Blue Waters,[1] a 13.3-petaflops/s supercomputer at the University of Illinois at Urbana–Champaign, to assess system utility, i.e., the fraction of time spent in performing computations. System component failures (such as processor failures and network router failures) and their associated recoveries significantly impact the running applications in the system, thus decreasing the overall utility. A data-driven failure model is built using data obtained in collaboration with Blue Waters staff. Such a model can help both system designers and application developers and users of the system. System designers can use this model for sensitivity analysis to identify the bottlenecks impeding higher system utility. In practice, users are primarily concerned with their application and, thus, can use this model to improve the overall application performance in the presence of inherent system failure and recovery characteristics. The characteristics of the proposed model are as follows.

1) The proposed model is application oriented, meaning that it takes a bottom-up approach to consider the effects of failures and recoveries of system components on the running application.
2) The application-oriented model is hierarchical. At the lowest level, failure and recovery of system components are modeled using arbitrary life distributions, such that no assumption is made about the specific form of the distribution. This lower level model is solved algebraically, and the outcomes of the lower level model, including successful recovery, further degradation, and failure, are incorporated into the upper-level Markovian model representing the application view of the system as transition probabilities. This general modeling approach enables the mapping of arbitrary continuous distributions characterizing failures and recoveries of the system components to discrete distributions at the top level.
3) Based on the characterization of the data, the proposed model assumes that the system component failures are statistically independent, but that the impact of all failures is not equal. For example, a failure of the compute node running the application would require that application to restart from the last checkpoint, whereas failure of the network node (i.e., network routers) impacts any running application irrespective of its location in the network topology.
4) The solution of the proposed reliability model informs performance analysis through a utility function, which is system independent. This utility function quantifies the fraction of time spent performing computations.

The model is used to calculate a closed-form analytical solution characterizing the utility of the system. Such an analytical solution is preferred because it enables efficient sensitivity analysis on the model parameters to determine the increase in utility attainable by: 1) reducing the failure rate of a hardware element; 2) increasing the probability of application or network recovery; and 3) decreasing the time to recover from an application, network, or system-wide outage. The sensitivity analysis of model parameters enables an objective ranking of alternative possible improvements. Neither the analytical nor the numerical methods widely used in current tools such as Mobius [1] or Figaro [2] can be used to directly obtain the utility of highly complex high-performance computing (HPC) systems. The automated methods provided in these tools fail to obtain solutions due to state-space explosion. While identical solutions of the simplified model presented in this article are possible using numerical methods, analytical methods are desirable for the reasons stated above. The value of the proposed model is as follows.

1) The proposed modeling approach is repeatable for other large-scale systems, thereby providing an approach to objectively compare existing HPC systems and rank the potential effectiveness of alternative design improvements.
2) Unlike alternative approaches [3], [4], our proposed model considers the application view. We perform a bottom-up modeling approach to consider reliability and performance from the perspective of the application. This enables us to avoid the complexity inherent in a top-down model of an HPC system that must consider multiple applications. In doing so, we reduce complexity from millions of states to simulate a system executing multiple applications, to a state space, which is linear in the number of checkpoints times a constant to consider outages that impact the application.
3) Such a modeling approach can help both system designers and application developers and users of the system. Moreover, the proposed model can serve as a guide for future research and development efforts of the HPC community such as novel network architectures, algorithms and hardware for faster recovery times, and scientific advances that lower failure rates to collectively support systematic efforts to scale HPC environments.

We illustrate the value of the model through examples, which employ parameters distilled from measurements collected from the Blue Waters supercomputer. Our results indicate that reducing the time to recover from network outages would improve utility most significantly (10.49%). This observation agrees with intuition because network outages impact all applications regardless of the number of compute nodes to which they are assigned. Moreover, network nodes are the most plentiful system component contributing to network outages and also possess the lowest mean time to failure among these network elements. Our analysis suggests that increasing job size from 100 compute nodes to all nodes (>28 000) without decreasing the checkpointing interval would lower the utility from 57.23% to 29.13%, and that a machine with just 12 cabinets (1152 nodes) to four times the size of Blue Waters (1136 cabinets) would degrade the utility of a job running on 1000 nodes from 82.95% to 33.78%.

II. Failures and Recoveries in Blue Waters

Modern HPC systems are composed of three primary components: compute, network, and storage. In such a design, compute servers (nodes) are interconnected with one another to form

[1]The Blue Waters supercomputer is one of the largest supercomputers in the world in terms of the number of compute nodes and storage servers.

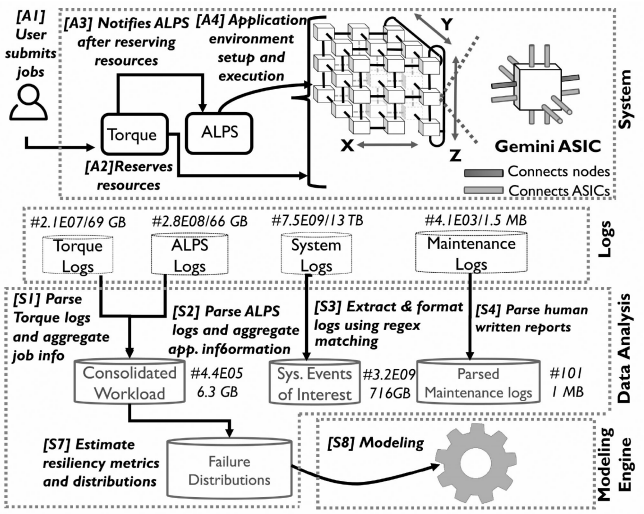


Fig. 1. Data flow of analysis and modeling framework.

compute clusters, storage servers are interconnected with one another to form storage clusters, and network components not only connect servers in storage and compute, respectively, but also connect compute to storage and *vice versa*. Such a design enables fault tolerance and scalability. Moreover, this idea is extended to each of the individual clusters, where compute nodes are further arranged in blades and cabinets, allowing the system to tolerate failures of individual blades and cabinets, while providing horizontal scaling capabilities.

As an example of a modern HPC system, this section provides a brief description of the design and architecture of the Blue Waters supercomputer as well as examples and illustrations of observed failures and their impact on applications. Blue Waters is a large-scale Cray XE system hosted at the University of Illinois at Urbana–Champaign.

A. Blue Waters System Design

Blue Waters is composed of 288 Cray liquid-cooled cabinets hosting 22 640 XE (CPU only nodes) nodes and 4228 XK (CPU + graphical processing unit) nodes. Each cabinet consists of an L1 cabinet controller, several fan trays, power conversion electronics, breakers, a blower and chiller, and related piping. Each cabinet is further composed of three chassis, and each chassis contains eight blades. Each blade is composed of four compute nodes (XE or XK) and two network nodes (Gemini ASICs).

A Cray Gemini [5] is a 3-D torus-based high-speed interconnection network that connects network nodes within Blue Waters. The "X" direction in a 3-D torus provides connectivity between cabinets in a row, while the "Y" direction connectivity between rows and the "Z" direction provides connectivity within the cabinet. The 3-D torus network and the Gemini ASIC are shown in the uppermost system layer of Fig. 1, where each cube corresponds to one Gemini ASIC (henceforth called network nodes). A network node in the torus network is connected to other network nodes by ten connections, two each in $X+$, $X-$, $Z+$, and $Z-$ directions and one each in $Y+$ and $Y-$.

B. Running Applications on Blue Waters

The application on an HPC system is a parallel scientific computing workload composed of multiple computation tasks. On Blue Waters, multiple applications can be run concurrently with one another. However, a compute node is entirely dedicated to the application as opposed to sharing the compute nodes among multiple applications such as the Cori HPC system at the National Energy Research Scientific Computing Center [6].

At any time, application tasks may be performing computation, communication, or I/O. Since most parallel scientific applications are written using the MPI [7], Charm++ [8], or PGAS [9] frameworks, the multiple tasks more or less coordinate their actions, behaving as one cohesive unit.

Steps A1–A4 in Fig. 1 show the steps taken by the user and system to launch and execute jobs on Blue Waters. A job consists of one or more application that users want to run in parallel. A user can submit either interactive jobs or batch jobs through the "qsub" command, a front-end interface to the Torque [10] resource manager. Jobs are immediately queued by the system. Once the requested resources are reserved, applications in the job scripts are executed with the assistance of Application-Level Placement Scheduler [11], which sets application environment variables such as libraries and system paths, executes, and monitors application progress. As the job transitions from submission to the finish state, an event message is written to its logs, which can then be monitored to track the status of the job in the system.

C. Examples and Illustration From Blue Waters

This section describes failure scenarios observed in Blue Waters, explaining how they inform our system modeling priorities. The case studies discussed here include the following.

1) Section II-C1 provides an example of occurrence of failures because of propagation.
2) Section II-C2 discusses the cost of checkpointing and recovery on large-scale applications.

1) Cascaded Failures Due to Fault Propagation: The pump gasket problem is an example of cascading failures in the system, which can occur when the temperature in a cabinet rises. Overheating can trigger emergency power-OFF of the blade, which is a mechanism to protect the blades from permanent damage. This action results in network link failure, triggering network-wide recovery. As this network-wide recovery progresses, additional network links fail in the same cabinet but on a different blade. These failures cascade into further failures in the route computations phase of network recovery since the topology experiences changes between the time the routes were calculated and the time the routes were asserted, producing an inconsistent system state. The failure of the route computation and subsequent retries to establish an alternative path for nodes to communicate with other nodes in the system leads to the failure of the network-wide recovery. The thermal effects propagated to nearby compute cabinets cause failures of routers and other components. To repair the system, a system-wide outage is declared, lasting for several

hours until the problem with the pump gasket can be detected and fixed. The failure of recovery operations can be attributed to the inability to recover from multiple failures occurring in close proximity in time because the time between failures was less than the time required to orchestrate a successful recovery operation. The failure of the recovery procedure was due to the inability of the procedures to recover from multiple failures occurring in close proximity in time, i.e., *the time between the failures was less than the time required to orchestrate a successful recovery operation.*

2) Checkpointing and Recovery of Large-Scale HPC Application: There are two checkpointing approaches employed in HPC systems: one is system-supported checkpointing such as the algorithm used is LAM/MPI checkpoint/restart framework [12]; the other approach is application based, where a global barrier is explicitly used in the application to save a global consistent state, which places the burden of checkpointing on the application. Checkpointing in Blue Waters is application based. The application is instrumented with a number of checkpoint primitives at its safe points (e.g., a global barrier), where it can safely quiesce, such as the end of a loop where it performs a checkpoint. This section discusses the checkpoint and recovery characteristics of MP-Gadget (Massively Parallel Cosmological SPH Simulation Software) application [13],[2] which is used to model galaxies and their evolution over time.

MP-Gadget is one of the largest applications executing on Blue Waters using as much as 20 000 nodes. Each checkpoint in MP-Gadget is about 50 TB, and writing the checkpoint takes 10–15 min. A failure of the application or running it from last checkpoint requires restoration of the checkpoint data and initialization of the application. This loading and initialization of the application can take 20–40 min. For MP-Gadget application run executing on 20 000 nodes, the checkpointing cost is 20k nodes $\times$ 10 min $=$ 3333.3 node hours and the restart cost is 20k nodes $\times$ 30 min $=$ 10 000 node hours. In general, the following observations can be made from the data available to use on checkpointing.

1) Users on a large-scale system tend to decrease checkpoint intervals if their jobs fail repeatedly. Users of MP-Gadget decreased checkpointing intervals by an hour to ensure that their application progresses forward in reasonable time.
2) Not all applications require equal amounts of time to checkpoint/restart. On Blue Waters, application developers take application-specific checkpoints, which can vary from dumping the memory to dumping only the parameters of the simulation models. For example, Parallel Spatial Direct Numerical Simulation [14] checkpoints about 8 TB of data, which require about 1 min, and restoration of checkpoint takes 2–5 min.

These and other complex sequences of events are difficult to exhaustively characterize. Therefore, we quantify the failure rates of elements within the hardware hierarchy within Blue Waters that contribute to application and network outage in order to construct a model of reliability and performance.

[2]https://github.com/bluetides-project/MP-Gadget

TABLE I
EMPIRICAL PARAMETER ESTIMATES

Rate	Parameter	Value
Compute node failure	λ_d	161,242 hours
Network node failure	λ_n	161,252 hours
Link failure	λ_l	2,307,957 hours
Blade failure	λ_b	553,608 hours
Cabinet failure	λ_c	280,000 hours
Network recovery time	t_{r_N}	1/4 hours

TABLE II
ASSUMED PARAMETER

probability/time	Parameter	Value
Probability of recovery	p_{r_A}, p_{r_N}	0.2, 0.1
Application recovery time	t_{r_A}	1/4 hours
Failure recovery time	t_{r_F}	1 hours
Checkpointing time	t_c	1/2 hours

D. Blue Waters Field Failure Data Characterization

The dataset was generated by the production system during the March 2013–January 2015 time frame. All the datasets and their usages in our analysis pipeline are shown in Fig. 1, along with the number of entries such as the number of the line and size on the disk. The datasets acquired directly from the system are shown in cylinders. We consider three different kinds of failures, namely, application, system, and component failures. An application failure occurs when the running application fails, whereas a system failure occurs when the system is shut down, while component failures include hardware and software failure. Component failures occur when the management software or the hardware fails.

Maintenance logs are generated by Blue Waters maintenance specialists and consolidated by Cray. Events are added to the failure report upon: 1) any failure that requires special corrective actions such as a manual reboot or repair of faulty hardware and 2) events that cause system downtime in the form of system-wide outages. Over 4000 incidents were reported during this study, including 101 system-wide outages.

System logs contain system events logged by the OS and by the Cray Hardware Supervisor System.

Table I provides the failure rates, probabilities, and time parameters of the model determined from empirical analysis of Blue Waters maintenance and system log data.

However, some application and system parameters of the model could not be estimated, due to inadequate field-measurements. Therefore, Table II provides numerical values assumed to illustrate how the model can be used to conduct sensitivity analysis.

In Table II, Pr_A and Pr_N are the probability of successful recovery from application and network failure, respectively, t_{r_A} and t_{r_F} are the times to recover from application and system failure, respectively, and t_c is the time to perform a checkpointing operation. It should be noted that the model can accommodate other values and is, therefore, not dependent on these assumptions.

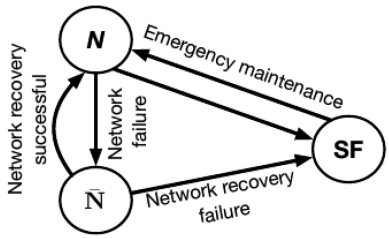


Fig. 2. System state representation. SF denotes system failure.

III. Modeling

This section develops a hierarchical model to quantify the impact of application, network, and system outages on an application implementing checkpointing. The top-level model (see Section III-A) is characterized by a discrete-time Markov chain (DTMC) possessing states to represent computation as well as application, network, network and application, and system-wide outages. Recovery from the application, network, and network and application failure states is also characterized by a set of lower level DTMC models (see Sections III-A2–III-A4), each with a finite number of retries.

Expressions for the transition to recovery and nonrecovery states are modeled by a combination of success in the underlying logic of the recovery attempt as well as life distributions for the hardware elements. These life distributions are general and can take any form determined from empirical data collected in system logs. The duration of retry attempts maps these continuous distributions to discrete transition probabilities, and the DTMC is solved to obtain the probability of recovering, experiencing a more serious failure, or suffering a system-wide outage. These probabilities are substituted into the top-level model, which also maps continuous distributions to discrete transition probabilities based on the time between checkpoints. The solution of this top-level model produces the average number of visits to each working and failure state. Combining this information with the time between application and network recovery retry attempts and holding times in states prior to failures enables performance analysis, decomposition of downtime to the various failure states, and quantifies the fraction of the time spent performing useful calculation.

The lower level model is solved algebraically, and the results are substituted into the top-level model, which is also solved algebraically. This approach enables efficient point calculations with system-specific numbers as well as sensitivity analysis to assess the relative benefit of increasing the failure time of the different hardware elements comprising the system, increasing the probability of successful recovery, and decreasing the time spent recovering from various failure states or performing checkpointing operations.

A. Markov Model

Fig. 2 shows possible states of the network, namely, working, recovering, and failed. If the network experiences a disruption, it enters the recovery state. This disruption impacts all applications in progress and prevents communication within the network.

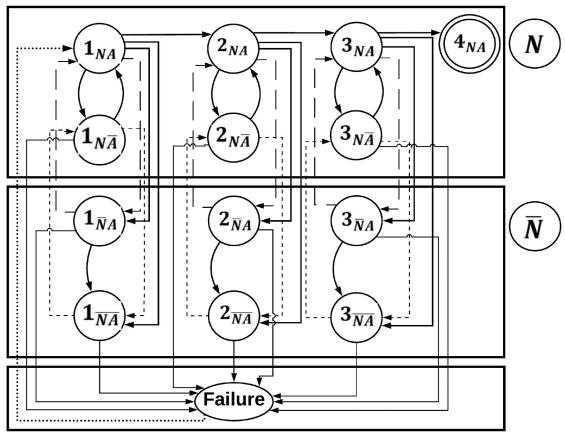


Fig. 3. State representation of application subject to network outages and failure.

If network recovery is successful, the network returns to the working state; otherwise, it enters the failure state and must undergo a restart. All applications that were executing at the time must restart from the previous checkpoint.

For the sake of exposition, Fig. 3 shows a more detailed state representation of an application implementing two intermediate checkpoints, which is subject to hardware failures as well as network outages that prevent communication and may induce failure because of timeouts. The upper two sets of states enclosed in rectangles in Fig. 3 correspond to the network state shown in Fig. 2. For example, the network is working properly (N) in the states in the uppermost rectangle, while the rectangle below refers to the states where the network is undergoing recovery ($\overline{N}$).

Without loss of generality, the system begins in state 1_{NA}, where both the network and the application are working, and transitions to state $1_{N\overline{A}}$, when hardware resources allocated to the application such as compute nodes fail, or $1_{\overline{NA}}$, when hardware resources allocated to the application such as network nodes induce both application and network recovery. The other possible failure transition from 1_{NA} is to the network recovery state ($1_{\overline{N}A}$) when hardware resources not allocated to the application induce network recovery. If no failure occurs, the system transitions to state 2_{NA}, which represents the first checkpoint where the application has successfully performed one-third of the required computation.

States 2_{NA} and 3_{NA} represent the first and second intermediate checkpoints, respectively, and possess similar transitions with interpretations identical to those described for state 1_{NA}. In the ideal case, computation proceeds through ℓ intermediate checkpoints represented by the states i_{NA} $(2 \leq i \leq (\ell+1))$ until it reaches the final state (4_{NA} in this example), which is surrounded by two concentric circles and represents successful completion of the application.

State $1_{N\overline{A}}$, where the network is working but the application is recovering, possesses three possible transitions. The first is a successful recovery attempt, transition to 1_{NA}. The second transition is to state $1_{\overline{NA}}$, where both the network and application attempt recovery. Later application recovery states such as $2_{N\overline{A}}$

and $3_{N\overline{A}}$ transition to 2_{NA} and 3_{NA}, respectively, upon successful recovery or their corresponding network recovery state $2_{\overline{NA}}$ and $3_{\overline{NA}}$ if a network outage occurs before application recovery. However, both go to the $Failure$ state if application recovery fails.

Ideally, states such as $1_{\overline{N}A}$, where the network is recovering but the application is working, will experience network recovery and return to 1_{NA}. However, application failure may occur, in which case both the network and application must attempt recovery from $1_{\overline{NA}}$. When the network fails to recover, the system enters the $Failure$ state from which all applications that were running must be restarted from 1_{NA}. From $1_{\overline{NA}}$, only two transitions are possible, namely, application recovery ($1_{N\overline{A}}$) or $Failure$ because the network must recover before an application can recover.

To enable efficient quantitative analysis, this section constructs an algebraic Markov model corresponding to the application subject to network outages and failure described in Fig. 3. Toward this end, we develop expressions for the transitions of the state model.

1) System Working: Checkpoint ($p_{i_{NA},(i+1)_{NA}}$)*:* We begin by formulating an expression for transition between two checkpoint states i_{NA} and $(i+1)_{NA}, i \in \{1, \ldots (\ell+1)\}$, where both the network and the application are working. For this to occur, the network must remain in the working state for duration

$$\tau = \frac{t_n}{\ell + 1} \tag{1}$$

where t_n is the computing time required of the application after distributing the application over n_d nodes. In the worse case, when computations require all compute nodes to be reliable, all elements involved in the computation must be reliable for time τ between subsequent checkpoints, including compute nodes, network nodes, and links as well as blades and cabinets.

The system under consideration consists of 284 cabinets, and each cabinet consists of 24 blades. Two network nodes reside in each blade, and these network nodes connect two compute nodes. Thus, each blade contains four compute nodes, while each cabinet contains 96 compute nodes. If a cabinet fails, the blades in that cabinet become inaccessible. Similarly, if a blade fails, the corresponding network and compute nodes of that blade will not be available. However, if a node fails, the network continues to work. To reach a checkpoint, it is necessary that resources allocated to the application do not fail as well as resources not allocated to the application that would induce network failure. Therefore, we construct the overall expression for the transition between i_{NA} and $(i+1)_{NA}$ (see Fig. 3) in two steps to distinguish between failures that induce both network and application recovery and those that trigger network or application recovery but not both.

Only failures of network nodes, blades, and cabinets allocated to the application result in application and network recovery. This is expressed as

$$p_{nbc}(\tau) = R_n(\tau)^{n_n} \times R_b(\tau)^{n_b} \times R_c(\tau)^{n_c} \tag{2}$$

where $R_n(\tau)$ is the probability that a network node remains in the working state for duration τ, while $R_b(\tau)$ and $R_c(\tau)$ are the corresponding reliability expressions for a blade and cabinet, respectively, while n_n, n_b, and n_c are the corresponding number of network nodes, blades, and cabinets, respectively. The common reliability expression for each element implicitly assumes that failures of elements such as compute nodes are independent and identically distributed. However, the failures of compute nodes, network nodes, blades, and cabinets are not assumed to be identical as indicated by the numerical parameters reported in Table I. The assumption of independence is common in hardware. In cases where a system is composed of heterogeneous hardware, more general expressions may be required.

Applications commonly utilize collocated resources. Due to this utilization of collocated resources, (2) can be further simplified by recalling that each network node contains two compute nodes, while each blade and cabinet contain 4 and 96 compute nodes respectively. Thus, the reliability or probability that the application executes for duration τ without experiencing a failure that would initiate both network and application recovery can be expressed in terms of n_d as

$$p_{nbc}(\tau) = R_n(\tau)^{\lceil \frac{n_d}{2} \rceil} \times R_b(\tau)^{\lceil \frac{n_d}{4} \rceil} \times R_c(\tau)^{\lceil \frac{n_d}{96} \rceil}. \tag{3}$$

Similarly, failure of compute nodes only leads to application recovery, whereas any link failure or failure of network nodes, blades, or cabinets not belonging to the application's resources results in network recovery. Thus, the probability of not entering the $N\overline{A}$ or $\overline{N}A$ states may be expressed as

$$\begin{aligned} p_{R_d R_{(N-n)}}(\tau) = {} & R_d(\tau)^{n_d} \times R_n(\tau)^{N_n - \lceil \frac{n_d}{2} \rceil} \times R_l(\tau)^{\lceil \frac{N_d}{12} \rceil} \\ & \times R_b(\tau)^{N_b - \lceil \frac{n_d}{4} \rceil} \times R_c(\tau)^{N_c - \lceil \frac{n_d}{96} \rceil} \end{aligned} \tag{4}$$

where $R_d(\tau)$ and $R_l(\tau)$ are the probabilities that a compute node and a network link remain in the working state for duration τ, while n_d is the number of compute nodes allocated to the application, and N_n, N_b, and N_c are the total number of network nodes, blades, and cabinets in the system.

Combining (3) and (4), the probability of successfully reaching a checkpoint is

$$p_{i_{NA},(i+1)_{NA}}(\tau) = p_{nbc}(\tau) \times p_{R_d R_{(N-n)}}(\tau). \tag{5}$$

a) System working to application recovery ($p_{i_{NA},i_{N\overline{A}}}$)*:* Transitions from states where both the network and the application are working (i_{NA}) to the state where the application undergoes recovery ($i_{N\overline{A}}$) occur when (i) a compute node allocated to the application fails, but no other failures that would lead to network outage take place, including nodes, blades, and cabinets (ii) internal and (iii) external to the application as well as network links. Equation (3) expresses (ii), while (i) and (iii) may be expressed as

$$\begin{aligned} p_{F_d R_{(N-n)}}(\tau) = {} & (1 - R_d(\tau)^{n_d}) \times R_n(\tau)^{\left(N_n - \lceil \frac{n_d}{2} \rceil\right)} \\ & \times R_l(\tau)^{\lceil \frac{N_d}{12} \rceil} \times R_b(\tau)^{\left(N_b - \lceil \frac{n_d}{4} \rceil\right)} \\ & \times R_c(\tau)^{\left(N_c - \lceil \frac{n_d}{96} \rceil\right)}. \end{aligned} \tag{6}$$

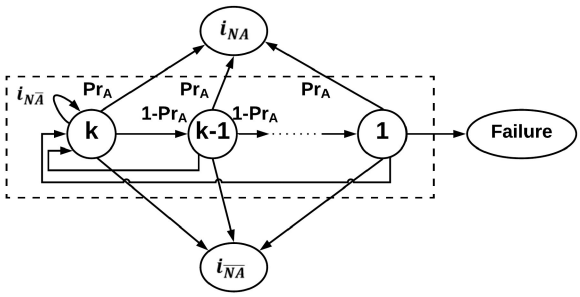


Fig. 4. States internal to application recovery ($i_{N\overline{A}}$).

Combining (3) and (6) provides the probability the system enters the application recovery state

$$p_{i_{NA},i_{N\overline{A}}}(\tau) = p_{nbc}(\tau) \times p_{F_d R_{(N-n)}}(\tau). \tag{7}$$

b) System working to network recovery ($p_{i_{NA},i_{\overline{NA}}}$): The system enters the network recovery state when network resources external to the application fail because failure of internal resources leads to application recovery or both application and network recovery. This probability of entering network recovery can be written in terms of previous expressions such that

$$p_{i_{NA},i_{\overline{NA}}}(\tau) = p_{nbc}(\tau) \times (R_d(\tau)^{n_d} - p_{R_d R_{(N-n)}}(\tau)) \tag{8}$$

where resources internal to the application are reliable, but a resource external to the application induces network recovery because the two terms in parentheses contain an $R_d(\tau)^{n_d}$ term, which factors so that what remains in parentheses is the complement of the reliability of the resources external to the application that correspond to entering network recovery.

c) System working to network and application recovery ($p_{i_{NA},i_{\overline{NA}}}$): The system enters both application and network recovery if a failure inducing application recovery and a failure inducing network recovery occur within the interval τ between two checkpoints

$$\begin{aligned} p_{i_{NA},i_{\overline{NA}}}(\tau) &= (1 - p_{nbc}(\tau)) \\ &\quad + p_{nbc}(\tau)\left((1 - R_d(\tau)^{n_d}) - p_{F_d R_{(N-n)}}(\tau)\right) \end{aligned} \tag{9}$$

which corresponds to failures of application resources that induce network and application recovery, and both terms in parentheses contain $(1 - R_d(\tau)^{n_d})$, which factors, meaning that no application resource that would induce both application and network recovery occurs (R_{nbc}), but a compute node allocated to the application fails ($(1 - R_d(\tau)^{n_d})$) triggering application recovery, and a network resource external to the application also fails triggering network recovery.

2) Application Recovery: Fig. 4 shows the states internal to application recovery, denoted by $i_{N\overline{A}}$ in Fig. 3, as well as its one-step transitions, including k retries down to the $Failure$ state, recovery to the network and application working state (i_{NA}), and network failure to the network and application recovery state ($i_{\overline{NA}}$).

a) Retry to system working ($p_{k-j,i_{NA}}, j \in \{0, \ldots (k-1)\}$): In addition to the logic of application recovery, which is successful with probability (p_{r_A}) or unsuccessful, as shown in Fig. 4, it is also necessary to consider the probability of other sources of failure prior to recovery. Therefore, the probability of transition from the initial (k) or subsequent retries after j unsuccessful attempts ($k - j$) to the application and network working state i_{NA} is

$$p_{k-j,i_{NA}} = p_{r_A} \times R_d(t_{r_A})^{n_d} \times p_N(t_{r_A}) \tag{10}$$

where

$$\begin{aligned} p_N(t_{r_A}) &= R_n(t_{r_A})^{N_n} \times R_l(t_{r_A})^{N_l} \times R_b(t_{r_A})^{N_b} \\ &\quad \times R_c(t_{r_A})^{N_c} \end{aligned} \tag{11}$$

because the logic of application recovery must complete successfully, and compute nodes belonging to the application as well as all network resources including those not allocated to the application must not fail.

b) Retry to next retry ($p_{k-j,k-(j+1)}$): The application performs k retry attempts each of which requires time t_{r_A}. If all k retry attempts are unsuccessful, the application transitions to the failure state and must then restart from the beginning (1_{NA}). A transition between two retries occurs with probability

$$p_{k-j,k-(j+1)} = (1 - p_{r_A}) \times R_d(t_{r_A})^{n_d} \times p_N(t_{r_A}) \tag{12}$$

because the logic of application recovery does not complete successfully, yet compute nodes belonging to the application and network resources must not fail, as these failures would lead to other state transitions described below.

c) Restart retry attempts ($p_{k-j,k}$): When a compute node experiences failure, the number of retry attempts is reset to k, which occurs with probability

$$p_{k-j,k} = (1 - p_{r_A}) \times (1 - R_d(t_{r_A}))^{n_d} \times p_N(t_{r_A}). \tag{13}$$

This transition makes the implicit assumption that the number of compute node failures, which occur prior to job completion, is small. In practice, applications request a small fraction of additional resources, which may support this assumption.

d) Retry to network and application recovery ($p_{k-j,i_{\overline{NA}}}$): The only remaining transition is to the network and application recovery state, which occurs with probability

$$p_{k-j,i_{\overline{NA}}} = 1 - (p_{k-j,i_{NA}} + p_{k-j,k(j+1)} + p_{k-j,k}) \tag{14}$$

when the retry is unsuccessful and the compute nodes allocated to the application are reliable, but a resource that induces network recovery occurs before the end of the present recovery attempt.

Table III summarizes the possible transitions between the states internal to application recovery shown in Fig. 4.

TABLE III
APPLICATION RECOVERY INTERNAL STATE TRANSITION PROBABILITY MATRIX

	k	$k-1$	...	1	i_{NA}	$i_{\overline{NA}}$	Failure
k	Eq (13)	Eq (12)	0	0	Eq (10)	Eq (14)	0
$k-1$	Eq (13)	0	Eq (12)	0	Eq (10)	Eq (14)	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	Eq (13)	0	0	0	Eq (10)	Eq (14)	Eq (12)
i_{NA}	0	0	0	0	1	0	0
$i_{\overline{NA}}$	0	0	0	0	0	1	0
$Failure$	0	0	0	0	0	0	1

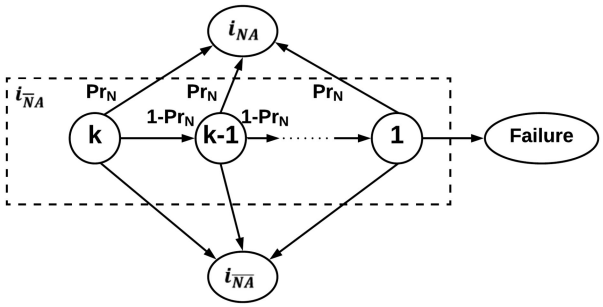


Fig. 5. States internal to network recovery ($i_{\overline{N}A}$).

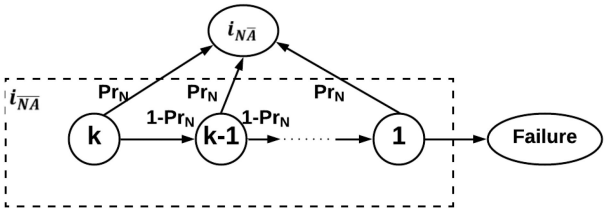


Fig. 6. Internal network and application recovery states ($i_{\overline{NA}}$).

3) Network Recovery: Fig. 5 shows the states internal to network node recovery denoted $i_{\overline{N}A}$ in Fig. 3 as well as its transitions.

Network recovery undergoes a process similar to application recovery described in Fig. 4. The probability of transition from the initial (k) or subsequent retries to the application and network working state i_{NA} is identical to (10) with the exception that p_{r_A} is replaced by p_{r_N}, which denotes the logic of network recovery is successful. Times are also modified from t_{r_A} to t_{r_N}. Unsuccessful retries are also similar to (12) with the substitution of p_{r_A} by p_{r_N}. Unlike application recovery, however, network node failures do not reset the number of retry attempts to k. Occurrence of failure in cabinets, blades, links, or compute nodes leads to network and application recovery. Thus, transition to the network and application recovery state occurs with probability

$$p_{k-j,i_{\overline{NA}}} = 1 - (p_{k-j,i_{NA}} + p_{k-j,k-(j+1)}). \tag{15}$$

4) Network and Application Recovery: Fig. 6 shows the states internal to network and application recovery denoted $i_{\overline{NA}}$ in Fig. 3 as well as its one-step transitions. When the system is in the application and network recovery state, network recovery must occur first because applications cannot proceed without the network. Therefore, like network recovery, the probability of transition from the initial (k) or subsequent retries to the application recovery state $i_{N\overline{A}}$ is identical to (10) with the exception that p_{r_A} is replaced by p_{r_N} and time from t_{r_A} to t_{r_N}. Therefore, transition to the next retry or *Failure* occurs with probability

$$p_{k,(k-1)} = (1 - p_{k,i_{N\overline{A}}}). \tag{16}$$

IV. Hierarchical Model Solution and Performance Analysis

This section explains how to solve the hierarchical model and conduct performance analysis. Section IV-A describes the hierarchical model solution process, explaining the method to calculate the number of visits to each state of the model. Section IV-B presents expressions for the total time spent in each of the states of Fig. 3.

TABLE IV
One-Step Transition Probability Matrix Between Working and Recovery States

	i_{NA}	$i_{N\overline{A}}$	$i_{\overline{N}A}$	$i_{\overline{NA}}$	$(i+1)_{NA}$	Failure
i_{NA}	0	$p_{i_{NA},i_{N\overline{A}}}$	$p_{i_{NA},i_{\overline{N}A}}$	$p_{i_{NA},i_{\overline{NA}}}$	$p_{i_{NA},i_{NA}}$	0
$i_{N\overline{A}}$	$p_{i_{N\overline{A}},i_{NA}}$	0	0	$p_{i_{N\overline{A}},i_{\overline{NA}}}$	0	$p_{i_{N\overline{A}},F}$
$i_{\overline{N}A}$	$p_{i_{\overline{N}A,NA}}$	0	0	$p_{i_{\overline{N}A},i_{\overline{NA}}}$	0	$p_{i_{\overline{N}A},F}$
$i_{\overline{NA}}$	0	$p_{i_{\overline{NA}},i_{N\overline{A}}}$	0	0	0	$p_{i_{\overline{NA}},F}$

A. Hierarchical Model Solution

The model consists of two levels. The upper level comprises the state representation of an application subject to network outages given in Fig. 3, including the working and recovery states, while the lower level includes the recovery state models specified in Figs. 4–6.

Starting from the lower level, the first step is to solve for the probability of transition to each of the terminal states in Fig. 4, namely, i_{NA}, $i_{\overline{NA}}$, and *Failure*. To accomplish this, we use the one-step transition probability matrix $\mathbf{P}$ specified in Table III. Each recovery state model is characterized by an absorbing DTMC. To solve for the average number of visits to each state of a DTMC [15], transitions to states with self-loops (absorbing states) are set to zero in the transition matrix corresponding to Fig. 4, including $p_{i_{NA},i_{NA}}$, $p_{i_{\overline{NA}},i_{\overline{NA}}}$, and failure $p_{F,F}$. This modified matrix is named $\mathbf{Q}$, and $\boldsymbol{\nu} = (\mathbf{I} - \mathbf{Q})^{-1}$ is referred to as the fundamental matrix. The entry $\nu_{l,m}$ indicates the average number of visits to state m given that the process began in state l. Since we assume without loss of generality that a recovery attempt begins in state k, we are only concerned with the first row of the matrix. Thus, we are interested in the average number of visits to states i_{NA}, $i_{\overline{NA}}$, and *Failure*, which can be visited at most once on any individual visit to the application recovery state in Fig. 4. Moreover, states are mutually exclusive and, therefore, constitute a discrete probability distribution that sums to 1.

The sequence of steps, including matrix inversion, is performed analytically with the equations contained in Table III ($\mathbf{P}$) and substituted into the upper level model as the transitions probabilities $p_{i_{N\overline{A}},i_{NA}}$, $p_{i_{N\overline{A}},i_{\overline{NA}}}$, and $p_{i_{N\overline{A}},F}$. These values are shown in the second row of Table IV, which represents transitions between the states in a single column of Fig. 3, the successor state $(i+1)_{NA}$, and the *Failure* state.

A similar procedure is applied to the one-step transition probability matrices for Figs. 5 and 6 according to the steps specified in the discussion above, producing third and fourth rows of Table IV.

The complete transition probability matrix for Fig. 3 contains three blocks on the diagonal possessing the form of Table IV for the two checkpoints with the complete specification shown in Fig. 7, where each block is given by the first four columns in Table IV, transitions to the Failure state are shown in column 13, and $p_{4_{NA},4_{NA}} = 1.0$ represents the successful completion

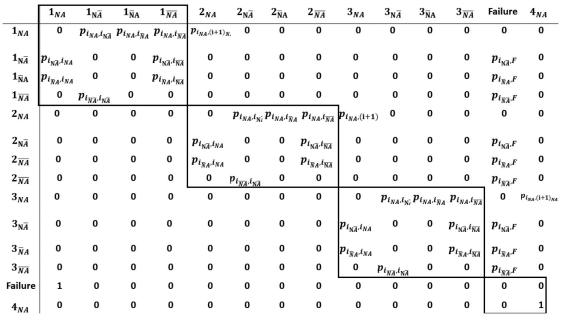


Fig. 7. System state transition probability matrix.

state. The DTMC solution procedure is applied to this upper level model, and the average number of visits to each state commencing from the 1_{NA} state is used in the next section to conduct performance analysis.

The authors acknowledge that while the transition probabilities of the DTMC representation of the system at both levels of the modeling hierarchy can be taken from arbitrary distributions, including those with a nonconstant failure rate, the memorylessness property of the DTMC makes the implicit assumption that the time to failure of components resets each time a new state is entered. Thus, subsequent visits to a recovery or working state do not suffer from increased probability of failure because components have aged. An alternative approaches would require Markov modeling with a large or even infinite number of conditional states to characterize aging or simulation with more complex mathematical expressions characterizing conditional failure rates. However, our results suggest that the model produces reasonable estimates of system utility despite these simplifying assumptions and enables a variety of useful inferences.

B. Performance Modeling

This section presents expressions for the total time spent in each of the states shown in Fig. 3. The expression for the total time spent in a working state is given by

$$\begin{aligned}\rho_{i_{NA}} = \tau + (\nu_{i_{NA}} - 1) \\ \times [\left(\frac{p_{i_{NA},i_{N\overline{A}}}(\tau)}{p_{i_{NA},i_{N\overline{A}}}(\tau) + p_{i_{NA},i_{\overline{N}A}}(\tau) + p_{i_{NA},i_{\overline{NA}}}(\tau)} \right) \\ E[H_A(\tau)] \\ + \left(\frac{p_{i_{NA},i_{\overline{N}A}}(\tau)}{p_{i_{NA},i_{N\overline{A}}}(\tau) + p_{i_{NA},i_{\overline{N}A}}(\tau) + p_{i_{NA},i_{\overline{NA}}}(\tau)} \right) \\ E[H_{(N-n)}(\tau)] \\ + \left(\frac{p_{i_{NA},i_{\overline{NA}}}(\tau)}{p_{i_{NA},i_{N\overline{A}}}(\tau) + p_{i_{NA},i_{\overline{N}A}}(\tau) + p_{i_{NA},i_{\overline{NA}}}(\tau)} \right) \\ E[H_n(\tau)]] .\end{aligned} \tag{17}$$

The first term (τ) corresponds to the time to complete the computation prior to the checkpoint, while the number of the visits to the states $N\overline{A}$, $\overline{N}A$, and $\overline{NA}$ is normalized and multiplied by the remaining number of visits to the working state ($\nu_{i_{NA}} - 1$), and $E[H_a(\tau)]$ corresponds to the mean time spent in the working state prior to failure of an application's compute nodes, $E[H_{(N-n)}(\tau)]$ network resources not assigned to the application, and $E[H_n(\tau)]$ network resources assigned to the application, where

$$E[H(\tau)] = \int_0^\tau R(t)dt. \tag{18}$$

An upper bound on the total time spent on transitions between working states performing checkpointing is

$$\rho_c = \sum_{i=2}^{\ell+1} \nu_{i_{NA}} \times t_c \tag{19}$$

where the index i begins at 2 because the first checkpointing operation is performed between 1_{NA} and 2_{NA} and t_c is the time to perform a checkpoint. Equation (19) provides an upper bound because $\nu_{i_{NA}}$ also includes visits that originate from the various recovery states. We explicitly evaluate the pessimism introduced by this bound in the illustrations and confirm that the impact does not significantly affect the inferences enabled by the model.

The expression for the total time spent in the ith application recovery state is

$$\begin{aligned}\rho_{i_{N\overline{A}}} = \sum_{j=0}^{k-1} \nu_{i_{N\overline{A}_{k-j}}} (p_{k-j,i_{NA}} \times t_{r_A} + p_{k-j,k-(j+1)} \times t_{r_A} \\ + \; p_{k-j,k} \times E[H_A(t_{r_A})] + p_{k-j,i_{\overline{NA}}} \times E[H_{(N-n)}(t_{r_A})])\end{aligned} \tag{20}$$

which is the sum of the number of visits to the $(k-j)$th retry of the ith application recovery state given in Fig. 4 multiplied by the sum of the probability of transitioning to one of four destination states multiplied by the time spent prior to transitioning to that state. Specifically, the probability of the application recovering ($p_{k-j,i_{NA}}$) and an unsuccessful retry attempt ($p_{k-j,k-(j+1)}$) both take deterministic time t_{r_a}, while the probability of restarting the number of retries to k ($p_{k-j,k}$) takes average time $E[H_A(t_{r_A})]$ corresponding to the mean holding time prior to the failure of an application's compute nodes within the duration of an application recovery attempt (t_{r_A}), and the probability of entering application and network recovery $p_{k-j,i_{\overline{NA}}}$ takes time $E[H_{(N-n)}(t_{r_A})]$ prior to the failure of an element that induces network recovery. Similar to (20), the expression for the total time spent in the ith network recovery state is given by

$$\begin{aligned}\rho_{i_{\overline{N}A}} = \sum_{j=0}^{k-1} \nu_{i_{\overline{N}A_{k-j}}} (p_{k-j,i_{NA}} \times t_{r_N} + p_{k-j,k-(j+1)} \times \\ t_{r_N} + p_{k-j,i_{\overline{NA}}} \times E[H_{NA}(t_{r_N})])\end{aligned} \tag{21}$$

where the only change to the first two terms is to replace the time to perform an application recovery (t_{r_A}) to the time to perform network recovery (t_{r_N}). The third term of (21) captures the contribution of failures that contribute to the probability of transitioning to network and application recovery ($p_{k-j,i_{\overline{NA}}}$)

multiplied by the time spent prior to failure of any network element or compute node ($E[H_{NA}(t_{r_N})]$) that induces network recovery prior to the end of the present network recovery attempt (t_{r_N}).

Finally, the expression for the total time spent in the ith network and application recovery state is simply

$$\rho_{i_{\overline{NA}}} = \sum_{j=0}^{k-1} \nu_{i_{\overline{NA}_{k-j}}} (p_{k-j,i_{N\overline{A}}} \times t_{r_N} + p_{k-j,k-(j+1)} \times t_{r_N}). \tag{22}$$

Therefore, the expression for the total time is

$$\begin{aligned} \rho = & \sum_{i=1}^{\ell+1} \rho_{i_{NA}} + \rho_c + \sum_{i=1}^{\ell+1} \rho_{i_{N\overline{A}}} + \sum_{i=1}^{\ell+1} \rho_{i_{\overline{N}A}} + \sum_{i=1}^{\ell+1} \rho_{i_{\overline{NA}}} \\ & + \rho_F = \sum_{x \in X} \sum_{i=1}^{\ell+1} \rho_{i_x} + \rho_c + \rho_F \end{aligned} \tag{23}$$

where $\rho_F = \nu_F \times t_{r_F}$ and ν_F is the number of visits to the *Failure* state, t_{r_F} is the time spent restarting from failure, and X is the set of working and recovery states $\{NA, N\overline{A}, \overline{N}A, \overline{NA}\}$.

Thus, the fraction of the time spent performing computations that lead to successful completion is

$$\mathscr{U} = \frac{t_n}{\rho}. \tag{24}$$

In the ideal case, where there are no failures or checkpointing, $\mathscr{U} = 1.0$.

Performing sensitivity analysis on the parameters of the model such as the failure rate parameters can be used to identify where bottlenecks lower efficiency, thereby suggesting where to invest efforts to lower failure rates of the various system components.

V. Results

This section presents a series of examples to illustrate the analysis and optimizations enabled by the model and is based on the empirical parameters reported in Table I. The failure rates given in Table I are failure rate parameters of the exponential distribution. However, the model specification is general and can accommodate any life distribution $R(t)$ for the elements of the system. We conclude the section with a discussion of implications for future design challenges.

A. Point Calculation of Utility

This example explains how to calculate the fraction of the time spent performing computations ($\mathscr{U}$). For the sake of illustration, the total computation time is $t_n = 6$ h when distributed over $n_d = 1000$ compute nodes. There are $\ell = 2$ checkpoints. Thus, (1) indicates that the length of the intervals is $\tau = 2$.

Table V summarizes the transition probabilities from the working state (i_{NA}) as well as the results of the DTMC analysis for the application, network, and network and application recovery states described in Figs. 4–6, thereby providing the transition probabilities for the system-level state diagram specified in Fig. 3.

TABLE V
Numerical Values for One-Step Transition Probability Matrix Between Working and Recovery States

	i_{NA}	$i_{N\overline{A}}$	$i_{\overline{N}A}$	$i_{\overline{NA}}$	$(i+1)_{NA}$	Failure
i_{NA}	0	0.0101	0.1686	0.0093	0.8120	0
$i_{N\overline{A}}$	0.4576	0	0	0.0812	0	0.4611
$i_{\overline{N}A}$	0.2480	0	0	0.1180	0	0.6340
$i_{\overline{NA}}$	0	0.2599	0	0	0	0.7400

TABLE VI
Numerical Values for Number of Visits to Each State of Application Subject to Network Outages and Failure

State	Visits (ν)	State	Visits (ν)
1_{NA}	1.6852	$1_{\overline{N}A}$	0.2841
2_{NA}	1.4406	$2_{\overline{N}A}$	0.2428
3_{NA}	1.2315	$3_{\overline{N}A}$	0.2076
$1_{N\overline{A}}$	0.0305	$1_{\overline{NA}}$	0.0516
$2_{N\overline{A}}$	0.0261	$2_{\overline{NA}}$	0.0441
$3_{N\overline{A}}$	0.0223	$3_{\overline{NA}}$	0.0377
4_{NA}	1	Failure	0.6008

TABLE VII
Time Spent in Each State and Total Time

	$\rho_{i_{NA}}$	$\rho_{i_{N\overline{A}}}$	$\rho_{i_{\overline{N}A}}$	$\rho_{i_{\overline{NA}}}$
1	3.2607	0.0076	0.0946	0.0172
2	2.8106	0.0065	0.0809	0.0147
3	2.4259	0.0056	0.0691	0.0126
Total	8.4973	0.0197	0.2446	0.0445

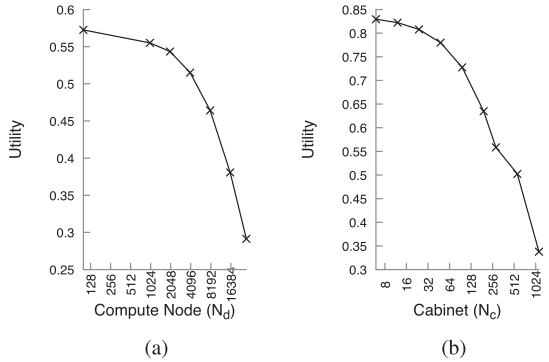
Solving the system-level model specified in Fig. 7, the expected number of visits to each state of Fig. 3 is given in Table VI.

Moreover, the mean holding time spent in the working state prior to failure of an application's compute nodes is $E[H_A(\tau)] = 1.987650$, while the holding times prior to failure of network resources assigned to the application and network resources not assigned to the application are $E[H_n(\tau)] = 1.992760$ and $E[H_{(N-n)}(\tau)] = 1.822700$, respectively.

Substituting the values from Table VI into (17) and (20)–(22), the times spent in the working and recovery states of the system-level model are given in Table VII. Furthermore, the times spent checkpointing and recovering from failure are $\rho_c = 1.336020$ and $\rho_F = 0.600819$, respectively. Therefore, the fraction of the time spent performing computations is $\mathscr{U} = 0.558506$. While this value may seem low, it should be noted that in the case where there are no failures, but two half hour checkpointing operations are performed, the value of $\mathscr{U}$ is $6/7 = 0.857143$. Moreover, to assess the impact of the upper bound in (19), we consider the best case, where each checkpoint is visited just once $\rho_c = 1.0$, which produces a value of $\mathscr{U} = 0.576539$, meaning that the pessimism introduced ($0.576539 - 0.558506 = 0.018033$) is less than 2% for the numerical values considered.

B. Sensitivity to Number of Compute Nodes (n_d)

To quantify the impact of the number of compute nodes on utility, Fig. 8(a) plots $\mathscr{U}$ for a range of values up to all nodes within the system on a logarithmic scale, holding all other model parameters constant at the values used in the first example.



(a) (b)

Fig. 8. Impact on utility with increasing (a) compute node and (b) cabinets. (a) Impact of job size $\mathscr{U}$. (b) Impact of the number of cabinets on $\mathscr{U}$.

TABLE VIII
SENSITIVITY ANALYSIS

Rate/probability/time	Parameter	Improvement	Rank
Compute node failure	$\lambda_d/2$	0.0068	7
Network node failure	$\lambda_n/2$	0.1049	1
Link failure	$\lambda_l/2$	0.0005	11
Blade failure	$\lambda_b/2$	0.0141	5
Cabinet failure	$\lambda_c/2$	0.0011	9
Application recovery probability	$p_{r_A} * 2$	0.0041	8
Network recovery probability	$p_{r_N} * 2$	0.0300	3
Application recovery time	$t_{r_A}/2$	0.0007	10
Network recovery time	$t_{r_N}/2$	0.0088	6
Failure recovery time	$t_{r_F}/2$	0.0161	4
Checkpointing time	$t_c/2$	0.0370	2

Fig. 8(b) indicates that utility decreases from 0.572590 to 0.291273 as the number of compute nodes increases. This is anticipated, because expressions such as (4) for the probability of not entering application or network recovery decreases as n_d increases. Thus, the model quantifies the degraded resource utilization to be expected when additional compute nodes are assigned to complete a computation more quickly without additional checkpoints.

C. Sensitivity to Total Number of Cabinets (N_c)

Similarly, to quantify the impact of the number of cabinets on utility, Fig. 8(b) plots $\mathscr{U}$ for a range of values up to 1136, which is four times larger than the Blue Waters system, from which the empirical parameters were determined. All other model parameters are held constant at the values used in the first example. Fig. 8(b) indicates that increasing the number of cabinets decreases the utility from 0.829505 to 0.337818, since this increases the number of elements that contribute to network outages regardless of the size of the number of compute nodes assigned to the application running within the HPC environment. This analysis suggests that simply increasing the scale of the computing resources will degrade utility, and that methods to support further scaling will be necessary.

D. Comparative Sensitivity Analysis

To illustrate how the model can be used to determine the relative benefit of lowering failure rates, increasing recovery probabilities, and decreasing recovery times, Table VIII provides a sensitivity analysis, ranking these potential improvements to the empirical model parameters.

Table VIII indicates that reducing the network node failure rate by a factor of 2 would increase utility by over 10%. This makes intuitive sense because network nodes are the most plentiful type of component that induces network failures and has the lowest mean time to failure, as specified in Section II-D. This type of sensitivity analysis coupled with historical industry trends and the corresponding economic investments required to achieve these improvements can be used to objectively allocate resources to improve system-level measures such as utility in order to cost effectively enhance future systems, directly supporting scalability for next-generation systems.

E. Discussion

The results obtained from the model suggest potential for improvements in the future HPC system. Key takeaways include the following.

1) Fig. 8(a) confirms that increasing the size of the application, in terms of number of nodes, lowers the overall system utility. Thus, to increase the overall utility, application developers must attempt to decompose a large-scale application into multiple instances of a small-scale application. Furthermore, the industry trend is to build faster and powerful compute nodes, which is becoming harder to improve successive generations of processor chips, due to the limitations of Moore's Law [16], [17]. Increasing the number of cores, scaling the frequency of cores, and speeding up the memory subsystem should require fewer nodes to complete the same computation. However, this poses two additional challenges: a) this places the burden on the application developer to take advantage of the powerful nodes and b) it is difficult to achieve higher performance per node for large-scale applications in practice, even after refactoring the code due to either inherent bottlenecks in the application code or jitter in the OS [18].
2) Fig. 8(b) shows that decreasing the number of cabinets increases the utility of the system. Decreasing the number of cabinets decreases not only cabinets, but also blades, network nodes, compute nodes, and links that contribute additional failure points. From Table VIII, it can be seen that λ_n has a significantly greater impact on the overall utility than λ_d. Thus, there are two possible ways to build an equivalent HPC system (in terms of floating point operations per seconds) with fewer cabinets: a) build more powerful compute nodes or b) build more powerful network nodes that can house two to four times more compute nodes at the same levels of reliability (λ_n). As discussed above, building powerful compute nodes comes with its own caveats. However, increasing the number of compute nodes per network node will decrease the number of required cabinets, blades, and network nodes in the system. The good news is that the HPC industry is already following this trend. For example, the network node in Cray Aries-based systems [19] houses four compute nodes. Our model suggests that there is further potential for improvement to build powerful network nodes and

cabinets, if such a system design is technologically and economically feasible.

3) Table VIII shows that checkpointing time (rank 2), failure recovery time (rank 4), and network recovery time (rank 6) can collectively play an important role to improve system utility. Checkpointing time can be decreased by using burst-buffer technology (NVRAM or SSD-based) [20] to significantly increase the write speed for checkpoints. Current network recovery techniques spend a significant amount of time for a) collecting heartbeats from every component and b) exploring the alternate routing paths in the network. The time spent in these steps can be decreased by parallelizing these methodologies.

VI. Related Work

A. Failure Characterization of HPC Systems

Many HPC technical studies [21]–[23] have suggested that the key challenges to achieve efficiency at exascale are energy, memory, concurrency, and resilience. Faults and errors occur frequently in large-scale HPC systems, requiring a range of recovery techniques to provide error resilience and prevent failures. Unsuccessful failure avoidance can lead to outage [24]. Hence, the vast majority of work seeks to characterize and understand the cause of failures such as fault propagation in HPC systems at different levels of the system, including hardware [25]–[29], software [25], [30], complete HPC systems [25], [31], [32], and application failures [33]. These studies have demonstrated the importance and potential widespread impact of failures in supercomputers.

B. Checkpointing and Recovery

Not all system-level failures and recoveries can be transparently masked from user applications. Propagation of errors to application leads to application failures. Previous work [33] showed that 1.53% of applications fail due to system problems and contribute to 9% of the production hour, which is equivalent to a loss of $421 878 in energy bills alone. To minimize the loss of compute node hours, applications employ checkpointing and recovery techniques.

Young [4] proposed one of the earliest models to identify optimal checkpoint intervals. This model assumes that the mean time between failure (MTBF) of the system is very large compared to the checkpoint and recovery time and, hence, did not consider failures during checkpointing and recovery. Daly [34] subsequently presented a modification of Young's model for large-scale systems, which accounts for failures during checkpointing and recovery as well as multiple failures in a single computation interval. However, it does not model the coordination overhead of the checkpointing protocol itself. Other models [3], [35] consider the effects of correlated failures on checkpointing interval and failures during application checkpointing. Their model does not consider the recovery protocols of the system components.

Zheng *et al.* [36] developed an in-memory checkpointing model when there are no extra compute nodes (or processors) in the system. Like Plank and Thomason [37], we assume availability of spare nodes to provide redundancy in the system to handle permanent failures. However, unlike Plank and Thomason [37], we do consider the overhead of coordination in their model or the effect of scaling the model to a large number of nodes. Egwutuoha *et al.* [38] conducted a survey of reliability and MTBF expressions for HPC systems and discussed rollback-recovery techniques developed over the years for HPC systems.

The goal of the models described above were to identify an optimal checkpoint interval in order to enhance a measure of system utility in the presence of failures, but did not consider the hierarchical organization of the components within a system, the distinct failure modes induced when resources allocated to an application or the broader system fail, and the specific recovery mechanisms undertaken in response to these distinct failure modes. Therefore, the contribution of this article is a performance (utility) model of an HPC system in terms of its architecture, operating logic, and component failure rates. The user-oriented approach avoids the state-space explosion encountered in HPC models that attempt to simultaneously model many concurrent applications. The model can serve as the basis of sensitivity studies to identify factors that impact an application's utility as well as guide component upgrades and related optimization problems.

VII. Conclusion

This article developed a hierarchical model of an application that implements checkpointing running in an HPC environment subject to application, network, and system-wide outages. Failures of hardware at each level of the HPC infrastructure as well as the probability and time to recover from application, network, and system failures were considered. Performance analysis quantified the fraction of the time spent computing, referred to as utility, as well as time spent checkpointing and recovering from failures. Point calculation and a variety of sensitivity analyses on the model parameters were conducted with empirical parameters determined from Blue Waters. Our results indicated that increasing the failure rate of network nodes could most appreciably enhance application utility. However, investment in this and the other alternative improvements must be tempered by the historical trends and cost to achieve such improvements. Additional studies on the number of compute nodes and cabinets (system size) quantified how utilization is expected to degrade as applications use more resources within existing systems as well as within attempts to build exascale HPC systems. While the modeling was performed for the Blue Waters system, its approach is general and should also be applicable to similar systems.

Future research will explore architectural generalization to the models to facilitate analysis of a broad class of similar systems. Many additional sources of failure and performance can be considered to enhance the realism and completeness of the model such as failure within checkpoints, failure of storage elements, correlated failures, additional forms of fault tolerance, application-specific software reliability, and optimal checkpointing, including the situations where aging of system components occurs between checkpoints and delays associated with queueing/scheduling. In addition to seeking objective methods to enhance the utility of computation, we will also seek to

develop richer notions of green and resilient computing, which contrasts with traditional methods that are often restricted to minimizing cost or maximizing throughput.

REFERENCES

[1] D. D. Deavours *et al.*, "The Mobius framework and its implementation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 956–969, Oct. 2002.

[2] A. Pfeffer, "Figaro: An object-oriented probabilistic programming language," Charles River Analytics, Cambridge, MA, USA, Tech. Rep. 137, 2009, p. 96.

[3] L. Wang *et al.*, "Modeling coordinated checkpointing for large-scale supercomputers," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2005, pp. 812–821.

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

[5] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini system interconnect," in *Proc. IEEE Symp. High Perform. Interconnects*, 2010, pp. 83–87.

[6] K. Antypas, N. Wright, N. P. Cardo, A. Andrews, and M. Cordery, "Cori: A cray XC pre-exascale system for NERSC," in *Cray User Group Proc.*, 2014.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, 1996.

[8] L. V. Kale and S. Krishnan, "CHARM++: A portable concurrent object oriented system based on C++," *ACM Sigplan Notices*, vol. 28, pp. 91–108, 1993.

[9] G. Almasi, "PGAS (Partitioned Global Address Space) languages," in *Encyclopedia of Parallel Computing*. New York, NY, USA: Springer, 2011, pp. 1539–1545.

[10] G. Staples, "Torque resource manager," in *Proc. ACM/IEEE Conf. Supercomput.*, 2006, p. 8.

[11] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing, "The application level placement scheduler," in *Proc. Cray User Group*, 2008.

[12] S. Sankaran *et al.*, "The Lam/Mpi checkpoint/restart framework: System-initiated checkpointing," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 4, pp. 479–493, 2005.

[13] Y. Feng, T. Di-Matteo, R. A. Croft, S. Bird, N. Battaglia, and S. Wilkins, "The BlueTides simulation: First galaxies and reionization," *Monthly Notices Roy. Astronom. Soc.*, vol. 455, no. 3, pp. 2778–2791, 2015.

[14] *SPP-2017 Benchmark Codes and Inputs*, National Center for Supercomputing Applications, Urbana, IL, USA, 2017. [Online]. Available: https://bluewaters.ncsa.illinois.edu/spp-benchmarks

[15] K. S. Trivedi, *Probability & Statistics With Reliability, Queuing and Computer Science Applications*. Hoboken, NJ, USA: Wiley, 2008.

[16] R. Colwell, "The chip design game at the end of Moore's law," in *Proc. Hot Chips 25 Symp.*, 2013, pp. 1–16.

[17] T. N. Theis and H.-S. P. Wong, "The end of Moore's law: A new beginning for information technology," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 41–50, 2017.

[18] D. Goodell *et al.*, "Minimizing MPI resource contention in multithreaded multicore environments," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2010, pp. 1–8.

[19] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC series network," Cray Inc., Seattle, WA, USA, White Paper WP-Aries01-1112, 2012.

[20] W. Bhimji *et al.*, "Accelerating science with the NERSC burst buffer early user program," Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, Tech. Rep., 2016.

[21] J. Dongarra *et al.*, "The international exascale software project roadmap," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, pp. 3–60, 2011.

[22] A. Geist and R. Lucas, "Major computer science challenges at exascale," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 427–436, 2009.

[23] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," Information Processing Techniques Office, Defense Advanced Research Projects Agency, Arlington County, VA, USA, Tech. Rep., vol. 13, 2008.

[24] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Mar. 2004.

[25] C. Di Martino, F. Baccanico, J. Fullop, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2014, pp. 610–621.

[26] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Proc. Perform. Anal. Syst. Softw.*, 2014, pp. 221–230.

[27] D. Tiwari *et al.*, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 331–342.

[28] K. S. Yim, C. Pham, M. Saleheen, Z. Kalbarczyk, and R. Iyer, "Hauberk: Lightweight silent data corruption error detector for GPGPU," in *Proc. Parallel Distrib. Process. Symp.*, 2011, pp. 287–300.

[29] S. Jha *et al.*, "Resiliency of HPC interconnects: A case study of interconnect failures and recovery in blue waters," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 915–930, Nov./Dec. 2018.

[30] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability: A study of field failures in operating systems," in *Proc. 21st Int. Symp. Digest Papers Fault-Tolerant Comput.*, 1991, vol. 21, pp. 2–9.

[31] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 3, pp. 212–226, 2009.

[32] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Dec. 2010.

[33] C. Di Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2015, pp. 25–36.

[34] J. Daly, "A model for predicting the optimum checkpoint interval for restart dumps," in *Proc. Int. Conf. Comput. Sci.*, 2003, pp. 3–12.

[35] N. H. Vaidya, "On checkpoint latency," Dept. Comput. Sci., Texas A&M Univ., College Station, TX, USA, Tech. Rep. 95-015, 1995.

[36] G. Zheng, L. Shi, and L. V. Kalé, "FTC-Charm: An in-memory checkpoint-based fault tolerant runtime for charm and MPI," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2004, pp. 93–103.

[37] J. S. Plank and M. G. Thomason, "The average availability of parallel checkpointing systems and its importance in selecting runtime parameters," in *Proc. IEEE 29th Annu. Int. Symp. Fault-Tolerant Comput.*, 1999, pp. 250–257.

[38] I. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *J. Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

Bentolhoda Jafary received the M.S. and Ph.D. degrees in reliability engineering from the Department of Electrical and Computer Engineering, University of Massachusetts Dartmouth, Dartmouth, MA, USA, in 2015 and 2019, respectively.

She is currently a Reliability Engineer with AeroVironment, Simi Valley, CA, USA. Her research interests include reliability modeling and maintainability.

Saurabh Jha received the bachelor's degree in computer science and engineering from the Vellore Institute of Technology, Vellore, India, in 2014, and the master's degree in computer science in 2016 from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, where he is currently working toward the Ph.D. degree with the Department of Computer Science.

His research interests include fault tolerance and reliability of mission-critical systems.

Lance Fiondella (Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Connecticut, Storrs, CT, USA, in 2012.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Massachusetts Dartmouth, Dartmouth, MA, USA, where he is also the Director of the University of Massachusetts Dartmouth Cybersecurity Center.

Ravishankar K. Iyer (Fellow, IEEE) received the Ph.D. degree in electrical engineering from The University of Queensland, Brisbane, Australia, in 1977. He is the George and Ann Fisher Distinguished Professor of Engineering with the University of Illinois at Urbana–Champaign, Champaign, IL, USA, where he holds appointments with the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Department of Computer Science.

Professor Iyer is a Chief Scientist of the Information Trust Institute and an Affiliate Faculty of the National Center for Supercomputing Applications.