Models for Resilience Design Patterns

Mohit Kumar and Christian Engelmann
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

***Abstract*—Resilience plays an important role in supercomputers by providing correct and efficient operation in case of faults, errors, and failures. Resilience design patterns offer blueprints for effectively applying resilience technologies. Prior work focused on developing initial efficiency and performance models for resilience design patterns. This paper extends it by (1) describing performance, reliability, and availability models for all structural resilience design patterns, (2) providing more detailed models that include flowcharts and state diagrams, and (3) introducing the Resilience Design Pattern Modeling (RDPM) tool that calculates and plots the performance, reliability, and availability metrics of individual patterns and pattern combinations.**

***Index Terms*—high-performance computing, resilience, design patterns, models**

I. Introduction

Resilience in extreme-scale high-performance computing (HPC) systems is a critical challenge. With each new supercomputer generation, component counts increase, component reliability decreases (e.g., due to shrinking process technology), and hardware and software complexity increases (e.g., due to heterogeneous computing, complex data, and workflows) [1]–[4]. Recent reports about serious reliability problems also include unexpected issues, such as bad solder, dirty power, and early wear-out [5], [6].

Resilience design patterns [7], [8] offer a structured hard- and software design approach for improving resilience, such that parallel applications running on these supercomputers generate accurate solutions in a timely and efficient manner. Frequently used in computer engineering, design patterns identify problems and provide generalized solutions through reusable templates. The novel resilience design pattern concept identifies and evaluates repeatedly occurring resilience problems and coordinates solutions throughout hardware and software components in supercomputers.

Prior work focused on (1) identifying and formalizing the resilience design patterns in production HPC systems and recent resilience technologies [7]–[9], (2) developing a proof-of-concept prototype for demonstrating the resilience design pattern concept using a fault-tolerant generalized minimal residual method (FT-GMRES) linear solver with portable resilience [10], [11], (3) creating new, outcome-based metrics for HPC resilience [12], and (4) developing initial performance and reliability models for resilience design patterns [13].

This work was sponsored by the U.S. Department of Energy's Office of Advanced Scientific Computing Research. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

This paper extends the previous work in initial performance and reliability models by (1) describing performance, reliability, and availability models for all structural patterns, (2) providing more detailed models with flowcharts and state diagrams, and (3) introducing the RDPM tool to study the characteristics of patterns and pattern combinations.

II. Background

This section summarizes the used terminology and metrics and briefly describes the concept of resilience design patterns.

A. Terminology and Metrics

The taxonomy in this document is largely based on prior work in definitions for dependability in computing systems [3], [8], [14], [15].

A fault is a defect in a system that has the potential to cause an error. It can be dormant. When activated, it leads to an error that causes an illegal system state. A failure occurs when an error reaches the service interface of a system, resulting in system inconsistent behavior with its specification.

Reliability is the probability of a system not experiencing a fault, error, or failure during operation $0 \leq t$ (Eq. 1). The fault, error, or failure distribution is the probability of such an event in the system during $0 \leq t$ (Eq. 2). The probability density function (PDF) $f(t)$ is its relative likelihood. The fault, error, or failure rate λ is the frequency at which a system experiences such an event. The mean-time to error (MTTE) is its expected time to error, while the mean-time to failure (MTTF) is its expected time to failure (Eq. 3).

Given a large population of identical systems, the rate λ for this population displays the "bathtub curve". The initial period has a high but rapidly decreasing rate (wear-in). Then a roughly constant rate can be observed for a prolonged period of time. Finally, the rate begins to increase again (wear-out). A normalized exponential PDF of $\lambda e^{-\lambda t}$ with a constant rate λ is typically assumed for the prolonged center period of the "bathtub curve" (Eqs. 4-6).

$$R(t) = 1 - F(t) = \int_t^\infty f(t)dt \quad (1)$$

$$F(t) = 1 - R(t) = \int_0^t f(t)dt \quad (2)$$

$$MTTF = \int_0^\infty R(t)dt \quad (3)$$

$$R(t) = e^{-\lambda t} \quad (4)$$

$$F(t) = 1 - e^{-\lambda t} \quad (5)$$

$$MTTF = 1/\lambda \quad (6)$$

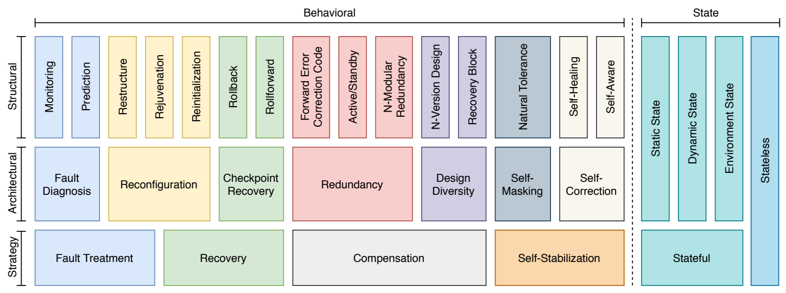


Fig. 1. Classification of resilience design patterns

N systems depending on each other exhibit serial reliability (Eq. 7) and N systems redundant to each other have parallel reliability (Eq. 8). Serial and parallel reliability can be simplified for identical systems (Eqs. 9 and 10). Assuming an exponential PDF for system i of $\lambda e^{-\lambda_i t}$, serial and parallel reliability and identical serial and parallel reliability can be simplified (Eqs. 11-14).

$$R(t)_s = \prod_{n=1}^{N} R_n(t) \quad (7)$$

$$R(t)_p = 1 - \prod_{n=1}^{N} (1 - R_n(t)) \quad (8)$$

$$R(t)_{is} = R(t)^N \quad (9)$$

$$R(t)_{ip} = 1 - (1 - R(t))^N \quad (10)$$

$$R(t)_s = e^{-\lambda_s t}, \quad \lambda_s = \sum_{n=1}^{N} \lambda_n \quad (11)$$

$$R(t)_p = 1 - \prod_{n=1}^{N} (1 - e^{-\lambda_n t}) \quad (12)$$

$$R(t)_{is} = e^{-\lambda t N} \quad (13)$$

$$R(t)_{ip} = 1 - (1 - e^{-\lambda t})^N \quad (14)$$

Availability is the proportion of time a system provides a correct service, with planned uptime (PU) t_{pu}, scheduled downtime (SD) t_{sd}, and unscheduled downtime (UD) t_{ud} (Eq. 15). Performance is the time required to successfully execute a task, including PU, SD, and UD. The mean-time to repair (MTTR) is the expected time to repair a system and can be used with the MTTF to calculate the mean-time between failures (MTBF) (Eq. 16). MTTR, MTTF, and MTBF can also be used to calculate availability (Eq. 17), if there is no SD. Systems depending on each other exhibit serial availability (Eq. 18) and systems redundant to each other have parallel availability (Eq. 19). Serial and parallel availability can be simplified for identical systems (Eqs. 20 and 21).

$$A = \frac{t_{pu}}{t_{pu} + t_{sd} + t_{ud}} \quad (15)$$

$$MTBF = MTTF + MTTR \quad (16)$$

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} \quad (17)$$

$$A_s = \prod_{n=1}^{N} A_n \quad (18)$$

$$A_p = 1 - \prod_{n=1}^{N} (1 - A_n) \quad (19)$$

$$A_{is} = A^N \quad (20)$$

$$A_{ip} = 1 - (1 - A)^N \quad (21)$$

B. Resilience Design Patterns

Design patterns describe generalizable solutions to recurring problems. They are often derived from best practices and contain the essential elements of design problems and their solutions. Design patterns provide designers with a template on how to solve a problem in different situations. They may also describe design alternatives to a specific problem.

Resilience design patterns [7] address the issues of dealing with faults, errors, and failures in computing systems, specifically in extreme-scale HPC. They identify the problems caused by such events and the solutions to handle them. The catalog of resilience design patterns [8] may be used by architects and developers as essential building blocks. It permits exploration of design alternatives and optimization of the cost-benefit trade-offs between performance, protection coverage, and power consumption of different resilience solutions.

The latest pattern classification (Fig. 1) features 21 behavioral patterns: 4 strategy, 7 architectural, and 15 structural. It also includes 5 state patterns. This paper extends the previous work [13], which introduced performance and reliability models for 6 resilience design patterns (Fault Diagnosis, Reconfiguration, Redundancy and Design Diversity architectural, Rollback, and Rollforward), to all 15 structural patterns.

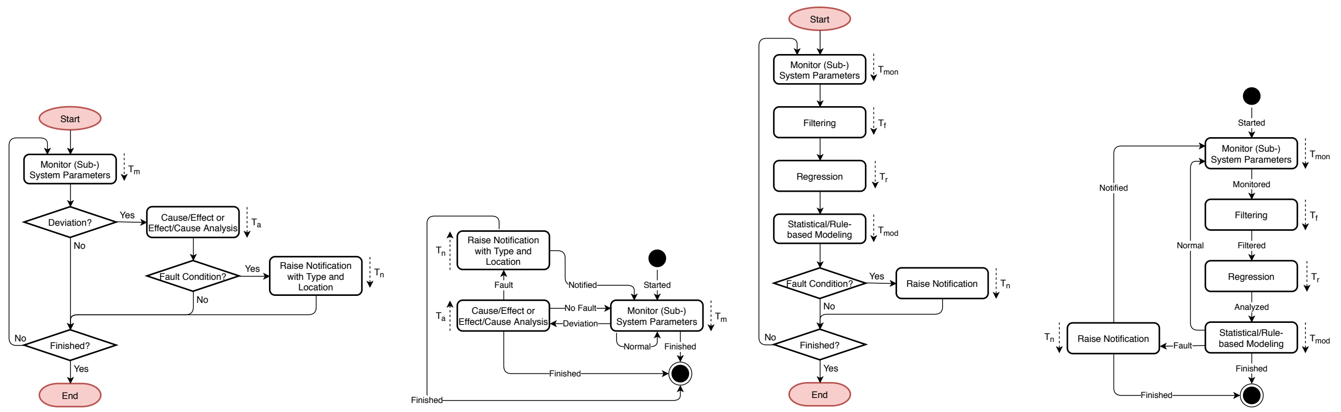
III. Related Work

Reliability and performance modeling, analysis, and optimization can be categorized [16] into: structural, state-space, and hierarchical models. Structural models highlight the relationships between systems using block diagrams, reliability graphs, and fault trees. State-space models describe dependencies between systems using Markov chains. Hierarchical models balance the speed of analysis with its accuracy by combining abstract structural models with detailed Markov models. In addition, performability analysis [17] models the interaction between failure recovery behavior and performance.

Checkpoint/restart (C/R), represented by the Rejuvenation, Rollback, and Rollforward pattern, is the main resilience strategy in HPC. Much of the reliability and performance modeling work in C/R has focused on the optimum checkpoint interval [18], [19] and on applying it in practice to systems with a non-constant MTBF [20], different failure distributions [21], and multilevel C/R solutions [22]. Research in the redundancy area is not used in production HPC and mostly focused on solutions and models for modular redundancy at the Message Passing Interface (MPI) [23], [24].

IV. Resilience Design Pattern Models

This section describes the performance, reliability, and availability models for each of the 15 structural resilience design patterns. They can be used to design and deploy resilient systems by understanding the different trade-offs. We assume an exponential PDF for fault, error, and failure.


Start
Monitor (Sub-) System Parameters
T_m
Deviation?
Yes
No
Cause/Effect or Effect/Cause Analysis
T_a
Fault Condition?
Raise Notification with Type and Location
T_n
Finished?
End
Raise Notification with Type and Location
Fault
Notified
Started
Cause/Effect or Effect/Cause Analysis
No Fault
Deviation
Normal
Finished
Monitor (Sub-) System Parameters
T_{mon}
Filtering
T_f
Regression
T_r
Statistical/Rule-based Modeling
T_{mod}
Raise Notification
Monitored
Filtered
Analyzed
Notified
Fault

(a) Monitoring Flowchart (b) Monitoring State diagram (c) Prediction Flowchart (d) Prediction State diagram

Fig. 2. Monitoring and Prediction pattern flowchart and state diagram

TABLE I
MONITORING PATTERN PARAMETERS

Parameter	Definition
T_m	Time to monitor (sub-) system parameters, including wait and probe times
T_a	Time to perform the cause/effect or effect/cause analysis
T_n	Time to raise notification with type and location

TABLE II
PREDICTION PATTERN PARAMETERS

Parameter	Definition
T_{mon}	Time to monitor (sub-) system parameters, including wait and probe times
T_f	Time to perform the filtering
T_r	Time to perform the regression
T_{mod}	Time to perform the statistical/rule-based modeling
T_n	Time to raise notification

A. Monitoring

The Monitoring pattern supports methods to recognize the presence of a defect or anomaly within a monitored system. The solution requires a monitoring system, which may be a subsystem of the monitored system or an external independent system. The flowchart of the pattern is shown in Fig. 2a, the state diagram in Fig. 2b, and its parameters in Table I.

When the monitoring system is a part of monitored system, the failure-free performance $T_{f=0}$ of the monitoring pattern is defined by the task's total execution time without any resilience strategy T_E and the time to monitor sub-system parameters, including wait and probe times T_m with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$, plus the time T_a to perform the cause/effect or effect/cause analysis and the time T_n to raise notification with type and location, where the total time to perform the cause/effect or effect/cause analysis and to raise notification with type and location is number of faults time T_a and T_n. The number of faults can be calculated by dividing T_E by MTTF (M). Assuming constant times T_m (t_m for P input-execute-output cycles), T_a, and T_n, T can be defined by Eq. 22. As the Monitoring pattern is not impacted by error or failure, the reliability remains the same as per Eq. 4. The availability of the Monitoring pattern can be calculated using the task's total execution time without the Monitoring pattern T_E and the performance with the Monitoring pattern T (Eq. 15). T_E is PU and T is PU , SD and UD.

$$T = T_E + P(t_m) + \frac{T_E}{M}\left(T_a + T_n\right) \qquad (22)$$

B. Prediction

Prediction supports methods to recognize the potential of a future defect or anomaly within a monitored system. The solution requires a monitoring system, which may be a subsystem of the monitored system or an external independent system. The flowchart of the pattern is shown in Fig. 2c, the state diagram in Fig. 2d, and its parameters in Table II.

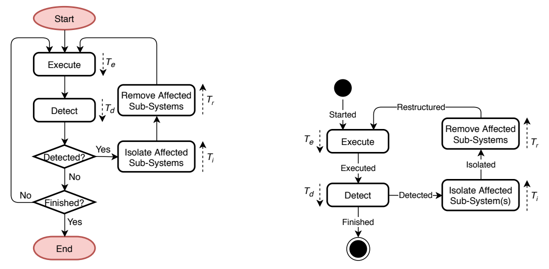
When the monitoring system is a part of the monitored system, the failure-free performance $T_{f=0}$ of the prediction pattern is defined by the task's total execution time without any resilience strategy T_E, the time to monitor sub-system parameters, including wait and probe times T_{mon}, the time to perform the filtering T_f, the time to perform the regression T_r, and the time to perform the statistical/rule-based modeling T_{mod} with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$, plus the time T_n to raise notification with type and location, where the total time to raise notification with type and location is number of faults time T_n. Assuming constant times T_{mon} (t_{mon}), T_f (t_f), T_r (t_r), T_{mod} (t_{mod}), and T_n, T can be defined by Eq. 23. Like the Monitoring pattern, the reliability remains the same and availability is defined by Eq. 15.

$$T = T_E + P(t_{mon} + t_f + t_r + t_{mod}) + \frac{T_E}{M}\left(T_n\right) \qquad (23)$$

C. Restructure

Restructure alleviates the impact of a fault, error, or failure on system operation by changing the interconnection between the subsystems in the overall system. It has a detection component that is similar to the Monitoring or Prediction pattern and an additional containment and mitigation component that acts upon the notification from the detection component. The flowchart of the pattern is shown in Fig. 3a, the state diagram in Fig. 3b, and its parameters in Table III.

In case when monitoring system is a part of monitored system, the failure-free performance $T_{f=0}$ of the restructure



(a) Flowchart (b) State diagram

Fig. 3. Restructure pattern flowchart and state diagram

TABLE III
RESTRUCTURE PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute system progress
T_d	Time to detect or predict a fault, error or failure
T_i	Time to isolate the affected subsystem(s)
T_r	Time to remove the affected subsystem(s)

pattern is defined by the task's total execution time without any resilience strategy T_E and the time to detect or predict a fault, error or failure T_d with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$, plus the time T_i to isolate the affected subsystem(s) and the time T_r to remove the affected subsystem(s), where the total time to isolate the affected subsystem(s) and to remove the affected subsystem(s) is number of faults, errors, or failures time T_i and T_r. Assuming constant times T_d (t_d), T_i, and T_r, T can be defined by Eq. 24.

Given that the Restructure pattern enables the resumption of correct operation after an error or failure, the reliability of a system employing it is defined by errors and failures that are not handled by the pattern, such as failures of the persistent storage system. The reliability after applying the Restructure pattern $R(t)$ can be obtained using the performance under failure T and the failure rate λ_u (or MTTF M_u) of the unprotected part of the system (Eq. 25). The availability is defined by Eq. 15.

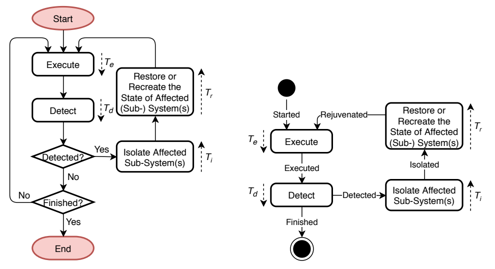
$$T = T_E + P(t_d) + \frac{T_E}{M}\left(T_i + T_r\right) \tag{24}$$

$$R(t) = e^{-\lambda_u T} = e^{-T/M_u} \tag{25}$$

D. Rejuvenation

Rejuvenation alleviates the impact of a fault, error, or failure on system operation by restoring the affected subsystem or system to a known correct state. It has a detection component that is similar to the Monitoring or Prediction structural patterns and an additional containment and mitigation component that acts upon the notification from the detection component and is similar to the Rollback or Rollforward structural patterns. The flowchart of the pattern is shown in Fig. 4a, the state diagram in Fig. 4b, and its parameters in Table IV.

Rejuvenation pattern detection component is same as the Monitoring pattern (Eq. 22). The containment and mitigation component impact the task total execution time same as in Rollback or Rollforward pattern (described later). We define performance using the Rollback pattern. We calculate performance under failure T by adding the time to detect or predict


Start
Execute
T_e
Detect
T_d
Restore or Recreate the State of Affected (Sub-) System(s)
T_r
Detected?
Yes
Isolate Affected Sub-System(s)
T_i
No
No
Finished?
Yes
End
Started
Rejuvenated
Restore or Recreate the State of Affected (Sub-) System(s)
T_r
T_e
Execute
Executed
Isolated
T_d
Detect
Detected
Isolate Affected Sub-System(s)
T_i
Finished

(a) Flowchart (b) State diagram

Fig. 4. Rejuvenation pattern flowchart and state diagram

TABLE IV
REJUVENATION PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute system progress
T_d	Time to detect or predict a fault, error, or failure
T_i	Time to isolate the affected subsystem(s)
T_r	Time to restore or replace the state of the affected subsystem(s)

a fault, error, or failure T_d with the total number of input-execute-output cycles P in Eq. 30. T_l, T_r, and T_s represent T_i time to isolate the affected subsystem(s) and T_r time to restore or replace the state of the affected subsystem(s). Assuming constant times T_d (t_d), T_l, T_r, and T_s, T can be defined by Eq. 26. Reliability and availability are defined by Eq. 25 and Eq. 15, respectively.

$$T = T_E + P(t_d) + \left(\frac{T_E}{\tau} - 1\right) T_s + \frac{T_E}{M} T_{e,f}(\tau + T_s) + \frac{T_E}{M}(T_l + T_r) \tag{26}$$

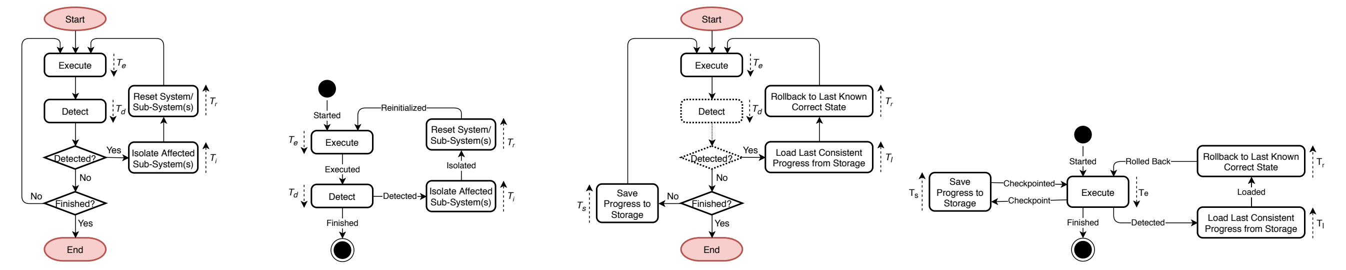
E. Reinitialization

Reinitialization alleviates the impact of a fault, error, or failure on system operation by restoring the affected subsystem or system to its initial state. It has a detection component that is similar to the Monitoring or Prediction structural patterns and an additional containment and mitigation component that acts upon the notification from the detection component. The flowchart of the pattern is shown in Fig. 5a, the state diagram in Fig. 5b, and its parameters in Table V.

TABLE V
REINITIALIZATION PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute system progress
T_d	Time to detect or predict a fault, error, or failure
T_i	Time to isolate the affected subsystem(s)
T_r	Time to reset the entire system or affected subsystem(s)

Reinitialization pattern failure-free performance $T_{f=0}$ is defined by the task's total execution time without any resilience strategy T_E and the time to detect or predict a fault, error, or failure T_d with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$, plus the time T_i to isolate the affected subsystem(s), the time T_r to remove the affected subsystem(s), and the time for work lost (which is assumed to be half of T_E), where the total time to isolate the affected subsystem(s), to remove the affected subsystem(s), and the time for work lost is number of faults, errors, or failures time T_i, T_r, and half of T_E. Assuming



(a) Reinitialization Flowchart (b) Reinitialization State diagram (c) Rollback Flowchart (d) Rollback State diagram

Fig. 5. Reinitialization and Rollback pattern flowchart and state diagram

constant times T_d (t_d), T_i, and, T_r, T can be defined by Eq. 27. Reliability and availability are defined by Eq. 25 and Eq. 15, respectively.

$$T = T_E + P(t_d) + \frac{T_E}{M}\left(T_i + T_r + T_E * 0.5\right) \tag{27}$$

F. Rollback

Rollback supports resilient operation by restoring the system to the time when the last checkpoint occurred in the event of an error or failure. The flowchart of the pattern is shown in Fig. 5c, the state diagram in Fig. 5d, and its parameters in Table VI.

TABLE VI
ROLLBACK PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute system progress
T_d	Time to detect an error/failure (not part of this pattern)
T_l	Time to load consistent system state and progress from storage
T_r	Time to rollback to the last known correct state
T_s	Time to save system state and progress to storage

The failure-free performance $T_{f=0}$ of the Rollback pattern is defined by the task's total execution time without any resilience strategy T_E and the time spent on saving system state and progress to storage T_s during task execution with a total number of checkpoints N. Assuming a constant checkpoint interval τ, the total number of checkpoints N is defined by the task's total execution time without any resilience strategy T_E divided by τ. T_d, time to detect an error/failure, is not part of this pattern.

The performance under failure T is defined by the failure-free performance $T_{f=0}$, plus the total lost time to execute system progress T_{EL} and the total time to load consistent system state and progress from storage and to rollback to the last known correct state T_R (Eq. 28). Assuming constant times T_s, T_l, and T_r, the performance under failure T can be further simplified with a total number of failures (Eq. 29). T can be calculated [19] using a first-order (Eq. 30) and a higher-order (Eq. 31) approximation for an optimal checkpoint interval τ. Reliability and availability are defined by Eq. 25 and Eq. 15, respectively.

$$T = T_E + T_S + T_{EL} + T_R \tag{28}$$

$$T = T_E + NT_s + T_{EL} + \frac{T_E}{M}(T_l + T_r) \tag{29}$$

$$T = T_E + \left(\frac{T_E}{\tau} - 1\right) T_s + \frac{T_E}{M} T_{e,f}(\tau + T_s) + \frac{T_E}{M}(T_l + T_r),$$
$$\tau = \sqrt{2MT_s} \tag{30}$$

$$T = Me^{(T_l+T_r)/M}\left(e^{(\tau+T_s)/M} - 1\right)\frac{T_E}{\tau},$$
$$\tau = \sqrt{2MT_s}\left[1 + \frac{1}{3}\left(\frac{T_s}{2M}\right)^{1/2} + \frac{1}{9}\left(\frac{T_s}{2M}\right)\right] - T_s \tag{31}$$

G. Rollforward

Rollforward supports resilient operation by restoring the system to the time when the error/failure event occurred in the event of an error or failure. The flowchart of the pattern is shown in Fig. 6a, the state diagram in Fig. 6b and its parameters in Table VII.

TABLE VII
ROLLFORWARD PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute (sub-) system progress
T_d	Time to detect an error/failure (not part of this pattern, but shown for completeness)
T_l	Time to load consistent (sub-) system state and progress from storage
T_r	Time to rollforward to the correct state before the event
T_s	Time to save (sub-) system state and progress to storage

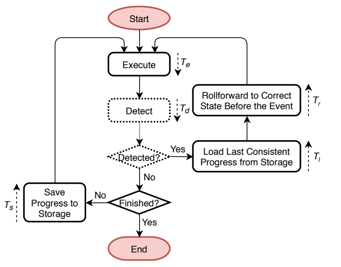
The Rollforward pattern avoids losing any work as it recovers the system to stable state immediately before the error or failure event. Assuming constant times T_s, T_l, and T_r, the performance T can be calculated by getting rid of lost work T_{EL} in Eq. 30 (Eq. 32). Reliability and availability are defined by Eq. 25 and Eq. 15, respectively.

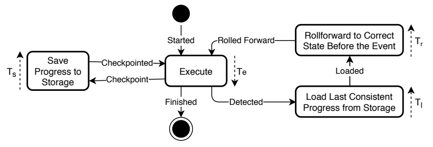
$$T = T_E + \left(\frac{T_E}{\tau} - 1\right) T_s + \frac{T_E}{M}(T_l + T_r),\ \tau = \sqrt{2MT_s} \tag{32}$$

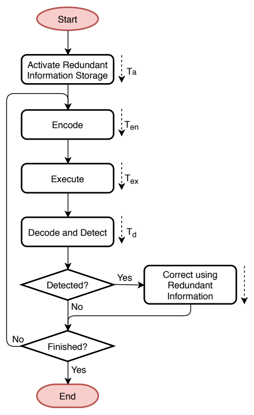
H. Forward Error Correction Code

Forward Error Correction Code (FECC) supports resilient operation by applying redundancy to system state and optionally to system resources in the form of encoded system state. Input is encoded, processed redundantly in an encoded fashion by the system, and the output is then decoded. The decoding corrects an error or failure. The flowchart of the pattern is shown in Fig. 6c, the state diagram in Fig. 6d, and its parameters in Table VIII.

The failure free performance $T_{f=0}$ of the FECC pattern is defined by the task total execution time without any resilience strategy T_E, the total time to activate the redundant information storage T_a, the time to encode T_{en}, and the time to decode


(a) Rollforward Flowchart


(b) Rollforward State diagram


(c) FECC Flowchart

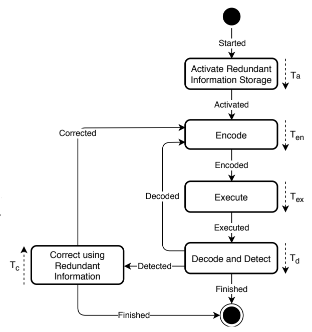

(d) FECC State diagram

Fig. 6. Rollforward and Forward Error Correction Code pattern flowchart and state diagram

TABLE VIII
FECC PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate the redundant information storage
T_{en}	Time to encode the input for the (sub-) system
T_{ex}	Time to execute (sub-) system progress
T_d	Time to decode the output from the (sub-) system and detect
T_c	Time to correct using redundant information

and detect T_d with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_c to correct using redundant information, where total time to correct using redundant information is number of error or failure times T_c. Assuming constant times T_a, T_{en} (t_{en}), T_d (t_d), and T_c, T can be defined by Eq. 33. All the above equations define total execution time while redundancy is in time. Reliability and availability are defined by Eq. 25 and Eq. 15, respectively.

$$T = T_E + T_a + P(t_{en} + t_d) + \frac{T_E}{M}(T_c) \tag{33}$$

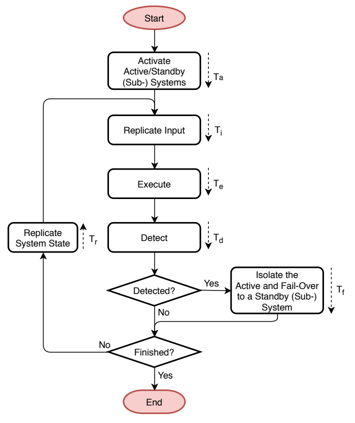
I. Active/Standby

Active/Standby supports resilient operation by applying redundancy in the form of N functionally identical replicas, using redundancy in space and potentially in time. The flowchart of the pattern is shown in Fig. 7a, the state diagram in Fig. 7b, and its parameters in Table IX.

TABLE IX
ACTIVE/STANDBY PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate the active and standby (sub-) systems
T_i	Time to replicate the input to the active and standby (sub-) systems
T_e	Time to execute progress on the active (sub-) system
T_d	Time to detect an error in or failure of the active (sub-) system
T_f	Time to isolate the active (sub-) system and fail-over to a standby (sub-) system
T_r	Time to replicate system state from the active (sub-) system to the standby (sub-) systems

The failure-free performance $T_{f=0}$ of the Active/Standby pattern is defined by the task total execution time without any resilience strategy T_E, the total time to activate the active and


(a) Flowchart

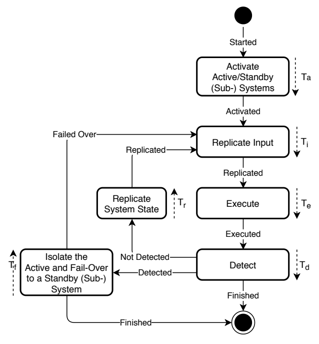

(b) State diagram

Fig. 7. Active/Standby pattern flowchart and state diagram

(sub-) standby systems T_a, the time to replicate the input to the active and standby (sub-) systems T_i, the time to detect an error in or failure of the active (sub-) system T_d, and the time to replicate system state from the active (sub-) system to the standby (sub-) systems T_r with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_f to isolate the active (sub-) system and fail-over to a standby (sub-) system, where total time to isolate is number of error or failure times T_f. Assuming constant times T_a, T_i (t_i), T_d (t_d), T_r (t_r), and T_f, T can be defined by Eq. 34. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq. 35) can be reformulated.

$$T = T_E + T_a + P(t_i + t_d + t_r) + \frac{T_E}{M}(T_f) \tag{34}$$

$$T = \alpha T_E + (1-\alpha)NT_E + T_a + P(t_i + t_d + t_r) + \frac{T_E}{M}(T_f) \tag{35}$$

Reliability is defined by the parallel reliability of the N-redundant execution and the performance under failure T (Eq. 36). It can be simplified for redundancy of identical systems (Eq. 37).

$$R(t) = 1 - \prod_{n=1}^{N}(1 - e^{-\lambda_n T}) \tag{36}$$

$$R_i(t) = 1 - (1 - e^{-\lambda T})^N \tag{37}$$

The availability A of the Active/Standby pattern is defined by N-parallel availability and the performance under failure T (Eq. 38). It can be simplified for redundancy of identical systems (Eq. 39). If T_a, T_i, T_d, T_r, and T_f are small enough, non-identical and identical availability can be simplified further (Eqs. 40 and 41), where M_n (or M) is the MTTF and R_n (or R) is the MTTR of each individual system (T_f).

$$A = 1 - \prod_{n=1}^{N}(1 - A_n) = 1 - \prod_{n=1}^{N}\left(1 - \frac{T_{E,n}}{T_n}\right) \quad (38)$$

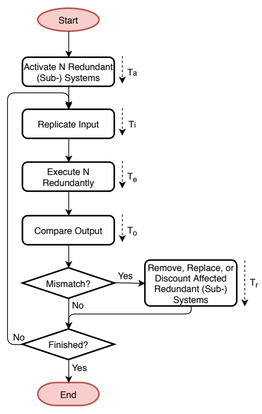
$$A_i = 1 - (1 - A)^N = 1 - \left(1 - \frac{T_E}{T}\right)^N \quad (39)$$

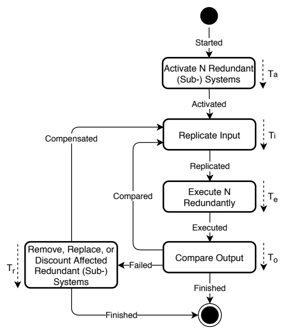
$$A = 1 - \prod_{n=1}^{N}\left(1 - \frac{M_n}{M_n + R_n}\right) \quad (40)$$

$$A_i = 1 - \left(1 - \frac{M}{M + R}\right)^N \quad (41)$$

J. N-modular Redundancy

N-modular Redundancy enables the continuous correct operation of a system by applying redundancy in the form of N functionally identical replicas. Redundancy in time uses the same resources to execute replicas. Redundancy in space uses redundant resources. A mix between both is possible, where there are more replicas than redundant resources. The flowchart of the pattern is shown in Fig. 8a, the state diagram in Fig. 8b, and its parameters in Table X.


Start
Activate N Redundant (Sub-) Systems
Ta
Replicate Input
Ti
Execute N Redundantly
Te
Compare Output
To
Mismatch?
Yes
Remove, Replace, or Discount Affected Redundant (Sub-) Systems
Tr
No
Finished?
Yes
End
(a) Flowchart



(b) State diagram

Fig. 8. N-modular Redundancy pattern flowchart and state diagram

TABLE X
N-MODULAR REDUNDANCY PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate N replicas of the system
T_i	Time to replicate the input to the N replicas
T_e	Time to execute system progress in the N replicas
T_o	Time to compare the outputs from the N replicas
T_r	Time to remove, replace, or discount the affected replica(s)

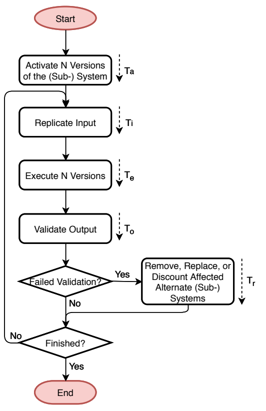
The failure-free performance $T_{f=0}$ of the N-modular Redundancy pattern is defined by the task's total execution time without any resilience strategy T_E, the total time to activate N replicas of the system T_a, the time to replicate the input T_i and the time to compare the outputs T_o with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$, plus the total time T_r to remove, replace, or discount the replica(s) where total time to remove, replace, or discount is number of error or failure times T_f. Assuming constant times T_a, T_i (t_i), T_o (t_o), and T_r, T can be simplified (Eq. 42). Using a ratio for replication in space vs. in time α, T (Eq. 43) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using R (T_r) by Eq. 41.

$$T = T_E + T_a + P(t_i + t_o) + \frac{T_E}{M}(T_r) \quad (42)$$

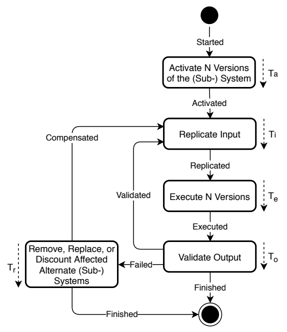
$$T = \alpha T_E + (1 - \alpha)NT_E + T_a + P(t_i + t_o) + \frac{T_E}{M}(T_r) \quad (43)$$

K. N-Version Design

N-Version Design supports resilient operation by applying redundancy in the form of N functionally equivalent alternate system implementations. The flowchart of the pattern is shown in Fig. 9a, the state diagram in Fig. 9b, and its parameters in Table XI.



(a) Flowchart



(b) State diagram

Fig. 9. N-version Design pattern flowchart and state diagram

TABLE XI
N-VERSION DESIGN PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate N versions of the (sub-) system
T_i	Time to replicate the input to the N versions of the (sub-) system
T_e	Time to execute (sub-) system progress in the N versions of the (sub-) system
T_o	Time to validate the output from the N versions of the (sub-) system
T_r	Time to remove, replace, or discount the affected redundant (sub-) system version(s)

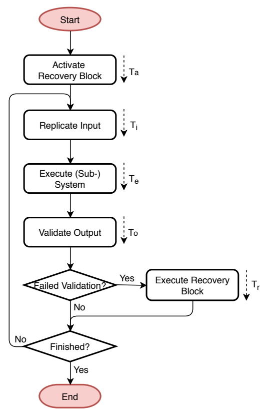
The failure-free performance $T_{f=0}$ of the N_Version is defined by the task total execution time without any resilience strategy T_E (the worst case execution time of N versions of the (sub-) system), the total time to activate N versions of the (sub-) system T_a, the time to replicate the input to the N versions of the (sub-) system T_i, and the time to validate the output from the N versions of the (sub-) system T_o with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_r to remove, replace, or discount the affected redundant (sub-) system version(s), where total time to remove, replace, or discount is number of error or failure times T_r. Assuming constant times T_a, T_i (t_i), T_o (t_o), and T_r, T can be defined by Eq. 44. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq. 45) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using $R\ (T_r)$ by Eq. 41.

$$T = T_E + T_a + P(t_i + t_o) + \frac{T_E}{M}\ (T_r) \tag{44}$$

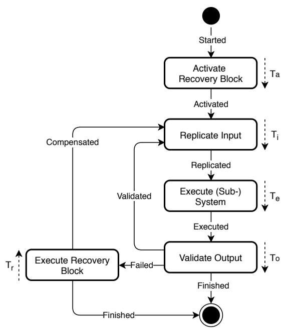
$$T = \alpha T_E + (1-\alpha)NT_E + T_a + P(t_i + t_o) + \frac{T_E}{M}\ (T_r) \tag{45}$$

L. Recovery Block

Recovery Block supports resilient operation by applying redundancy in the form of a functionally equivalent alternate system implementation encapsulated in a recovery block. The flowchart of the pattern is shown in Fig. 10a, the state diagram in Fig. 10b, and its parameters in Table XII.



(a) Flowchart



(b) State diagram

Fig. 10. Recovery Block pattern flowchart and state diagram

TABLE XII
RECOVERY BLOCK PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate the recovery block of the (sub-) system
T_i	Time to replicate the input to the (sub-) system and the recovery block of the (sub-) system
T_e	Time to execute (sub-) system progress
T_o	Time to validate the output from the (sub-) system
T_r	Time to execute the recovery block of the (sub-) system

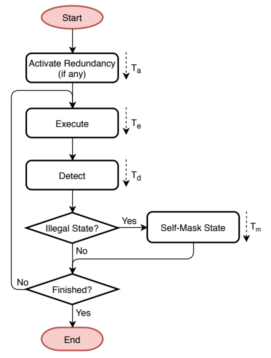
$$T = T_E + T_a + P(t_i + t_o) + \frac{T_E}{M}\ (T_r) \tag{46}$$

$$T = \alpha T_E + (1-\alpha)NT_E + T_a + P(t_i + t_o) + \frac{T_E}{M}\ (T_r) \tag{47}$$

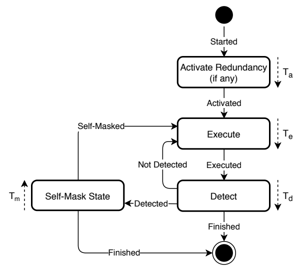
The failure-free performance $T_{f=0}$ of the Recovery Block pattern is defined by the task total execution time without any resilience strategy T_E, the total time to activate the recovery block of the (sub-) system T_a, the time to replicate the input to the (sub-) system and the recovery block of the (sub-) system T_i, and the time to validate the output from the (sub-) system T_o with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_r to execute the recovery block of the (sub-) system, where total time to execute the recovery block of the (sub-) system is number of error or failure times T_r. Assuming constant times T_a, T_i (t_i), T_o (t_o), and T_r, T can be defined by Eq. 46. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq. 47) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using R (T_r) by Eq. 41.

M. Natural Tolerance

Natural Tolerance relies on the capability of reaching a correct system state from an illegal system state after a finite number of execution steps using implicit error/failure detection and self-masking. Self-masking may be as simple as approximation of a correct state. The Redundancy pattern is often employed to aid in the process of self-masking and to extend the pattern's protection domain. The flowchart of the pattern is shown in Fig. 11a, the state diagram in Fig. 11b, and its parameters in Table XIII.



(a) Flowchart



(b) State diagram

Fig. 11. Natural Tolerance pattern flowchart and state diagram

TABLE XIII
NATURAL TOLERANCE PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate redundancy (if any)
T_e	Time to execute system progress
T_d	Time to detect illegal system state
T_m	Time to self-mask illegal system state

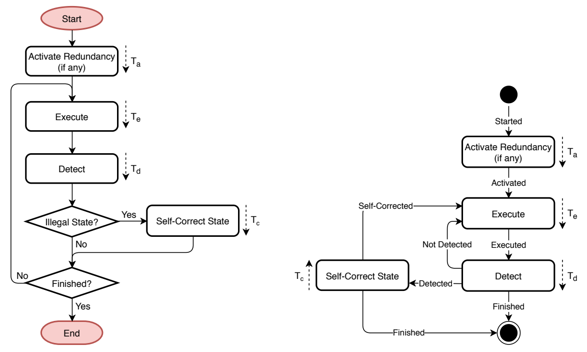
The failure-free performance $T_{f=0}$ of the Natural Tolerance pattern is defined by the task total execution time without any resilience strategy T_E, the total time to activate redundancy T_a, and the time to detect illegal system state T_d with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_m to self-mask illegal system state, where total time to self-mask illegal system state is number of error or failure times T_m. Assuming constant times T_a, T_d (t_d) and T_m, T can be defined by Eq. 48. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq. 49) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using R (T_m) by Eq. 41.

$$T = T_E + T_a + P(t_d) + \frac{T_E}{M}\ (T_m) \tag{48}$$

$$T = \alpha T_E + (1-\alpha)NT_E + T_a + P(t_d) + \frac{T_E}{M}\ (T_m) \tag{49}$$

N. Self-Healing

Self-Healing relies on the capability of reaching a correct system state from an illegal system state after a finite number of execution steps using explicit error/failure detection and self-correction. Self-correction may be as simple as discarding,



(a) Flowchart (b) State diagram

Fig. 12. Self-Healing pattern flowchart and state diagram

TABLE XIV
SELF-HEALING PATTERN PARAMETERS

Parameter	Definition
T_a	Time to activate redundancy (if any)
T_e	Time to execute system progress
T_d	Time to detect illegal system state
T_c	Time to self-correct illegal system state

recomputing, or estimating a wrong value in the system or a wrong or missing output from a subsystem. The flowchart of the pattern is shown in Fig. 12a, the state diagram in Fig. 12b, and its parameters in Table XIV.

The failure-free performance $T_{f=0}$ of the Self-Healing pattern is defined by the task total execution time without any resilience strategy T_E, the total time to activate redundancy T_a, and the total time to detect illegal system state T_d with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time T_c to self-correct illegal system state, where total time to self-correct illegal system state is number of error or failure times T_c. Assuming constant times T_a, T_d (t_d) and T_c, T can be defined by Eq. 50. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq. 51) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using R (T_c) by Eq. 41.

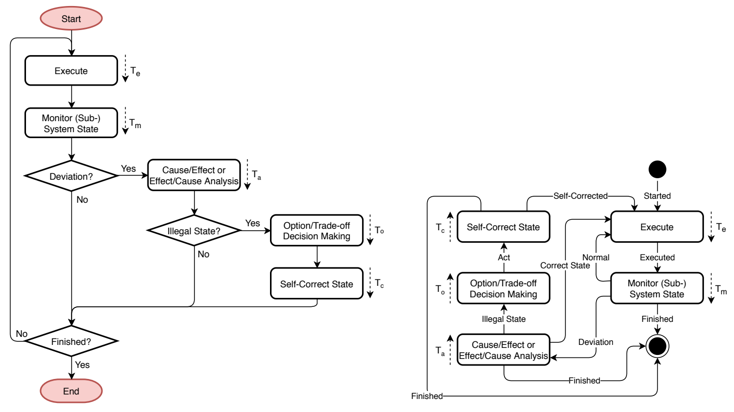
$$T = T_E + T_a + P(t_d) + \frac{T_E}{M}(T_c) \tag{50}$$

$$T = \alpha T_E + (1-\alpha)NT_E + T_a + P(t_d) + \frac{T_E}{M}(T_c) \tag{51}$$

O. Self-Aware

Self-Aware relies on the capability of reaching a correct system state from an illegal system state after a finite number of execution steps using explicit error/failure detection and self-correction. It additionally employs the Fault Diagnosis architectural pattern and an observe, orient, decide, and act (OODA) loop control for error/failure detection and self-correction. The flowchart of the pattern is shown in Fig. 13a, the state diagram in Fig. 13b, and its parameters in Table XV.

The failure-free performance $T_{f=0}$ of the Self-Aware pattern is defined by the task total execution time without any resilience strategy T_E and the time to monitor (sub-) system parameters, including wait and probe times T_m with the total number of input-execute-output cycles P. The performance under failure T is defined by $T_{f=0}$ plus the time to perform



(a) Flowchart (b) State diagram

Fig. 13. Self-Aware pattern flowchart and state diagram

TABLE XV
SELF-AWARE PATTERN PARAMETERS

Parameter	Definition
T_e	Time to execute system progress
T_m	Time to monitor (sub-) system parameters, including wait and probe times
T_a	Time to perform the cause/effect or effect/cause analysis
T_o	Time to perform the option/trade-off decision making
T_c	Time to self-correct illegal system state

the cause/effect or effect/cause analysis T_a, the time T_o to perform the option/trade-off decision making and the time T_c to self-correct illegal system state, where total time to perform the cause/effect or effect/cause analysis, to perform the option/trade-off decision making and to self-correct illegal system state is number of error or failure times T_a, T_o, and T_c. Assuming constant times T_m (t_m), T_a, T_o, and T_c, T can be defined by Eq. 52. When the redundancy is in space, using a ratio for replication in space vs. in time α, T (Eq.53) can be reformulated. Reliability is defined by Eq. 37. Availability can be calculated using R ($T_a + T_o + T_c$) by Eq. 41.

$$T = T_E + P(t_m) + \frac{T_E}{M}(T_a + T_o + T_c) \tag{52}$$

$$T = \alpha T_E + (1-\alpha)NT_E + P(t_m) + \frac{T_E}{M}(T_a + T_o + T_c) \tag{53}$$

V. RESILIENCE DESIGN PATTERN MODELING TOOL

The RDPM tool was developed to simplify studying the characteristics of patterns and pattern combinations. Each pattern has its own models and parameters, which can make it a rather tedious task to understand the performance, reliability, and availability trade-offs of different patterns and pattern implementations. This is even more complicated by horizontal and vertical combinations of patterns to complement each other. The RDPM tool alleviates these difficulties by modeling patterns and pattern combinations and providing performance, reliability, and availability plots for each scenario. This permits design space exploration that navigates the space of patterns as well as individual pattern parameter spaces.

The Python-based RDPM[1]tool implements performance, reliability, and availability models for each of the 15 structural resilience design pattern as individual classes. Patterns objects

[1]https://code.ornl.gov/6hk/rdpm

can be created and configured using an XML file[2], describing a systems resilience design patterns. Each class offers a method to calculate and plot its performance, reliability, and availability metrics. A full description of the RDPM tool and a demonstration of its capabilities is outside the scope of this paper. However, the interested reader is encouraged to to follow up on some of our early experiments[3] that can not be presented here due to space concerns.

In these experiments, we set T_E to 168 hours. For the Monitoring to FECC patterns, we set MTTF M to 24-168 hours (1-7 days). The MTTF M_u of the unprotected part of the system is 720 hours (30 days). For patterns from Active/Standby to Self-Aware, for performance we set MTTF M to 192 hours (8), for reliability and availability we set MTTF M (M_{RA} in the XML file) to 48-336 hours (2-14 days). The ratio for replication in space vs. in time, α, is in between 0 and 1. All other parameters are measured in hours and range from 1 second to 2 minutes.

VI. Conclusion

This paper extends previous work in initial performance and reliability models for resilience design patterns by (1) describing performance, reliability, and availability models for all 15 structural patterns, (2) providing more detailed models with flowcharts and state diagrams that illustrate pattern behavior, and (3) introducing the RDPM tool to study the characteristics of patterns and pattern combinations, including performance, reliability, and availability models between different patterns and pattern implementations.

Future efforts will focus on models for power consumption and energy for each structural pattern to allow for performance, resilience power/energy HPC system design space exploration using modeling and simulation tools and for runtime performance, resilience power/energy trade-offs by autonomous runtime systems. Future work will also focus on validating the models and verifying the RDPM tool. This paper intends to provide the theoretical foundations for resilience design pattern models and an initial design space exploration tool. It will take real performance, reliability, and availability data to validate the models and to verify the RDPM tool. This was outside the scope of this paper.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Early Career Program, with program managers Robinson Pino and Lucy Nowell.

References

[1] J. T. Daly *et al.*, "Inter-agency workshop on HPC resilience at extreme scale," 2012. [Online]. Available: http://institute.lanl.gov/resilience/docs/Inter-AgencyResilienceReport.pdf

[2] A. Geist *et al.*, "U.S. Department of Energy fault management workshop," Workshop report, 2012. [Online]. Available: https://science.osti.gov/-/media/ascr/pdf/program-documents/docs/FaultManagement-wrkshpRpt-v4-final.pdf

[3] M. Snir *et al.*, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 127–171, May 2014.

[4] P. Radojkovic *et al.*, "Towards resilient EU HPC systems: A blueprint," European HPC resilience initiative, 2020. [Online]. Available: https://resilienthpc.eu/results

[5] A. Geist, "How to kill a supercomputer: Dirty power, cosmic rays, and bad solder," *IEEE Spectrum*, vol. 10, pp. 2–3, 2016.

[6] G. Ostrouchov, D. Maxwell, R. Ashraf, C. Engelmann, M. Shankar, and J. Rogers, "GPU lifetimes on Titan supercomputer: Survival analysis and reliability," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2020*, Nov. 15-20, 2020.

[7] S. Hukerikar and C. Engelmann, "Resilience design patterns: A structured approach to resilience at extreme scale," *Journal of Supercomputing Frontiers and Innovations*, vol. 4, no. 3, pp. 4–42, Oct. 2017.

[8] ——, "Resilience design patterns: A structured approach to resilience at extreme scale (version 1.2)," Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2017/745, Aug. 2017.

[9] ——, "A pattern language for high-performance computing resilience," in *European Conference on Pattern Languages of Programs*, 2017, pp. 12:1–12:16.

[10] R. Ashraf, S. Hukerikar, and C. Engelmann, "Pattern-based modeling of multiresilience solutions for high-performance computing," in *ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 80–87.

[11] ——, "Shrink or substitute: Handling process failures in HPC systems using in-situ recovery," in *Euromicro International Conference on Parallel, Distributed, and network-based Processing*, 2018, pp. 178–185.

[12] S. Hukerikar, R. Ashraf, and C. Engelmann, "Towards new metrics for high-performance computing resilience," in *Workshop on Fault Tolerance for HPC at eXtreme Scale*, 2017, pp. 23–30.

[13] S. Hukerikar and C. Engelmann, "Pattern-based modeling of high-performance computing resilience," in *Lecture Notes in Computer Science: Workshop on Resiliency in High Performance Computing in Clusters, Clouds, and Grids*, vol. 10659, 2017, pp. 557–568.

[14] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[15] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, Jul. 2007.

[16] H. Pham, *Reliability Modeling, Analysis and Optimization*. World Scientific, 2006.

[17] K. S. Trivedi and M. Malhotra, *Reliability and Performability Techniques and Tools: A Survey*. Springer, 1993, pp. 27–48.

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

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

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

[21] S. Levy and K. B. Ferreira, "An examination of the impact of failure distribution on coordinated checkpoint/restart," in *Workshop on Fault-Tolerance for HPC at Extreme Scale*, 2016, p. 35–42.

[22] S. Di, L. Bautista-Gomez, and F. Cappello, "Optimization of a multi-level checkpoint model with uncertain execution scales," in *IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 907–918.

[23] D. Fiala, F. Mueller, C. Engelmann, K. Ferreira, R. Brightwell, and R. Riesen, "Detection and correction of silent data corruption for large-scale high-performance computing," in *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 78:1–78:12.

[24] C. Engelmann, H. H. Ong, and S. L. Scott, "The case for modular redundancy in large-scale high performance computing systems," in *IASTED International Conference on Parallel and Distributed Computing and Networks*, 2009, pp. 189–194.

[2] https://code.ornl.gov/6hk/rdpm/-/blob/master/xml/patterns.xml

[3] https://code.ornl.gov/6hk/rdpm/-/tree/master/images