RDPM: An Extensible Tool for Resilience Design Patterns Modeling *

Mohit Kumar and Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{kumarm1,engelmannc}@ornl.gov

Abstract. Resilience to faults, errors, and failures in extreme-scale high-performance computing (HPC) systems is a critical challenge. Resilience design patterns offer a new, structured hardware and software design approach for improving resilience. While prior work focused on developing performance, reliability, and availability models for resilience design patterns, this paper extends it by providing a Resilience Design Patterns Modeling (RDPM) tool which allows (1) exploring performance, reliability, and availability of each resilience design pattern, (2) offering customization of parameters to optimize performance, reliability, and availability, and (3) allowing investigation of trade-off models for combining multiple patterns for practical resilience solutions.

Keywords: high-performance computing, resilience, design patterns, tool

1 Introduction

Resilience ensures successful execution of application running on HPC systems with thousands of nodes prone to several software and hardware failures. Next generation of HPC systems, contending for exaflops speed, will see more of these software and hardware failures, requiring more rigorous resiliency techniques. Recent unexpected issues in HPC systems such as bad solder, dirty power, and early wear-out [10, 17] calls for better resiliency measures.

Resilience design patterns [12, 13] present a structured hard- and software design approach to tackle resilience problems in next generation HPC systems. Prior work focus on (1) identifying and standardizing the resilience design patterns in production high-performance computing (HPC) systems [12, 11, 13], (2)

* 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).

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 [1, 2], (3) describing performance, reliability, and availability models for all structural patterns with flowcharts and state diagrams, and (4) introducing initial Resilience Design Pattern Modeling (RDPM) tool to study the characteristics of patterns [15].

This paper extends the previous work by (1) exploring each resilience design pattern models with parameter values customization, and (2) advancing RDPM tool to study combination of resilience design patterns.

2 Background

This section describes the metrics and resilience design patterns necessary to understand models implemented in RDPM.

2.1 Terminology and Metrics

The glossary in this work is mostly derived from our prior work in computing systems [13, 19, 3, 14].

A fault is a flaw in a system that can result in an error. It may not cause any error when hidden, but once activated it can result in an error that can put a system in an illegal state. Once the error gets to the system service interface, it becomes a failure and makes the system inconsistent.

Reliability of a system is the probability of it not running into a fault, error, or failure $0 \leq t$ (Eq. 1). The fault, error, or failure distribution is the system reliability probability during $0 \leq t$ (Eq. 2). Its relative possibility is probability density function (PDF) $f(t)$. The rate at which a system encounters fault, error, or failure is λ. The mean-time to error (MTTE) is its anticipated time to error, while the mean-time to failure (MTTF) is its anticipated time to failure (Eq. 3).

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

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

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

$$A = \frac{t_{pu}}{t_{pu} + t_{sd} + t_{ud}} \quad (4)$$

$$MTBF = MTTF + MTTR \quad (5)$$

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} \quad (6)$$

Availability is the part of the time a system works correctly, with planned uptime (PU) t_{pu}, scheduled downtime (SD) t_{sd}, and unscheduled downtime (UD) t_{ud} (Eq. 4). Performance is the time in which a task is executed successfully, including PU, SD, and UD. The mean-time to repair (MTTR) is the anticipated time to repair. It can be used with the MTTF to determine the mean-time between failures (MTBF) (Eq. 5). Availability can be determined using MTTR, MTTF, and MTBF (Eq. 6), if there is no SD.

2.2 Resilience Design Patterns

Resilience design patterns [12] specifically tackle the problem of handling faults, errors, and failures in extreme-scale HPC. They help in finding the problem

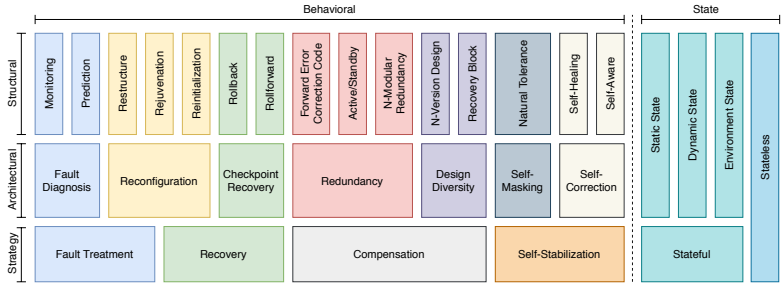


Fig. 1. Classification of resilience design patterns

induce by faults, errors, and failures and provide solutions to resolve them. Architects and developers can use resilience design patterns catalog [13] to create next generation resilient systems. Resilience design patterns allow investigation of design options to study the cost-benefit trade-offs between performance, protection coverage, and power consumption of different resilience solutions.

The current resilience design patterns catalog has 21 behavioral patterns: 4 strategy, 7 architectural, and 15 structural (Fig. 1). It also contains 5 state patterns. This paper extends the prior work [15], by introducing RDPM tool to explore performance, reliability, and availability of each structural resilience design pattern and investigate trade-off models for combining multiple patterns for practical resilience solutions.

3 Related Work

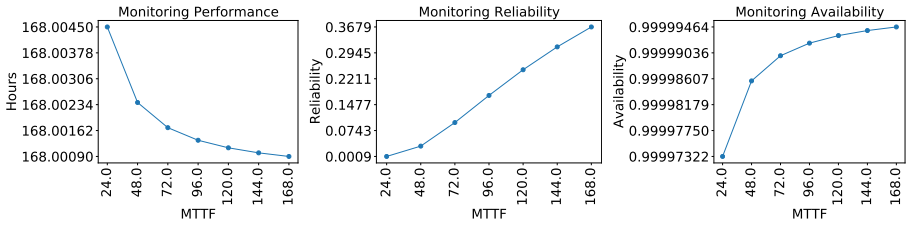
Reliability modeling, analysis and optimization proposes three types of models [18]: structural, state-space, and hierarchical. Structural models use block diagrams, reliability graphs, and fault trees to show the relation between systems. State-space models use Markov chains to show dependency between systems. Hierarchical models combine abstract structural models with Markov models to balance the speed of analysis and model accuracy. Additionally, Trivedi et.al. [21] propose performability analysis to model the interaction between performance and failure recovery behavior.

Rollback pattern represents Checkpoint/restart (C/R), which is one of the main resiliency strategies in HPC. In C/R, most of the reliability and performance models have been about optimum checkpoint interval [22, 4] and its application to systems with a non-constant MTBF [20], different failure distributions [16], and multilevel C/R solutions [5].

In production HPC, modular redundancy is still not in use. Modular redundancy research is mostly concentrated on solutions and models at the Message Passing Interface (MPI) [9, 7]. For the first time, Elliott et. al. combine two different resilience mechanisms, C/R and modular redundancy [6], to explore performance and reliability trade-offs. This paper implements and further investigates the performance, reliability, and availability trade-off models.

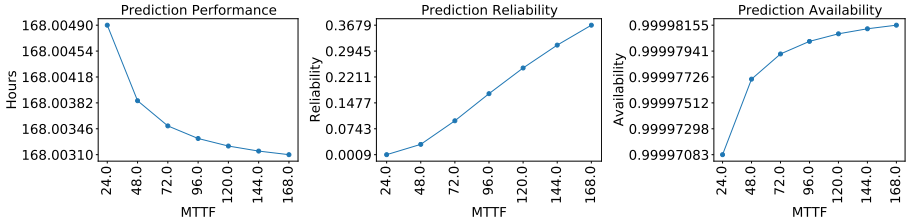
4 RDPM

RDPM tool simplify the modeling of performance, reliability, and availability of patterns and their combination. Each pattern has its own models and parameters, which makes it hard to understand the performance, reliability, and availability for different parameters values under different implementations. Things



(a) Performance (b) Reliability (c) Availability

Fig. 2. Monitoring pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

Fig. 3. Prediction pattern performance, reliability, and availability

get more complex when multiple patterns are combined horizontally or vertically for resiliency. The RDPM tool allows calculation of performance, reliability, and availability with ease for individual or combined patterns.

The Python-based RDPM[1]tool allows calculation, plotting, and storing of performance, reliability, and availability values for patterns and patterns combination. It has five components - RDP, Extract, Plot, CSV, and Patterns. RDP is the main class. It allows extraction of parameters from XML file and calculation, storing, and plotting of performance, reliability, and availability values. Extract allows extraction of individual pattern parameters from XML[2]file. Patterns calculate the performance, reliability, and availability values and pass to Plot to draw line/3D scatter plot. The calculated values are also passed to CSV for storing as CSV files.

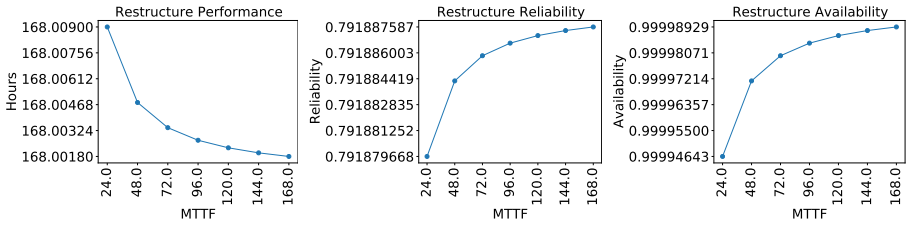
4.1 Structural Patterns

Next, we will define the parameters, calculate performance, reliability, and availability values, and plot it for all the structural patterns. The performance, reliability, and availability models for all the structural patterns can be found in [15].

Monitoring: The monitoring pattern uses a monitoring system to recognize a defects or anomalies. Fig. 2 demonstrates performance, reliability and availability of the Monitoring pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days). t_m,T_a, and T_n is 1 second. Reliability remains low with wrong results as the pattern just monitor the system.

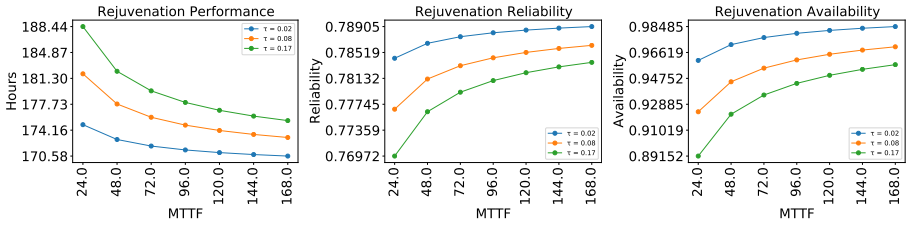
Prediction: The prediction pattern uses a monitoring system to recognize the potential of future defect or anomaly. Fig. 3 demonstrates performance, reliability and availability of the Prediction pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days). t_{mon},t_f, t_r, and

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



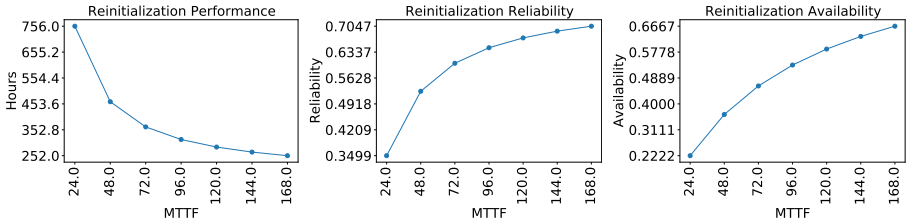
(a) Performance (b) Reliability (c) Availability

Fig. 4. Restructure pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

Fig. 5. Rejuvenation pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

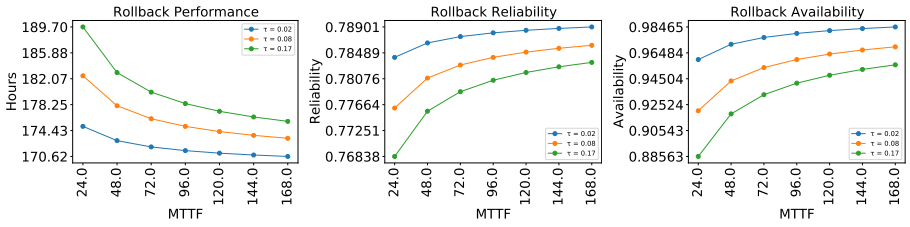
Fig. 6. Reinitialization pattern performance, reliability, and availability

t_{mod} is 2 seconds. T_n is 1 second. Reliability remains low with wrong results as the pattern just monitor the system to predict potential defect or anomaly.

Restructure: The restructure pattern changes the interconnection between the systems to reduce the impact of a fault, error, or failure. Fig. 4 demonstrates performance, reliability and availability of the Restructure pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days). t_d, T_i, and T_r is 2 second. The MTTF M_u of the unprotected part of the system is 720 hours (30 days). Reliability increases as the pattern resolve the fault, error, or failure.

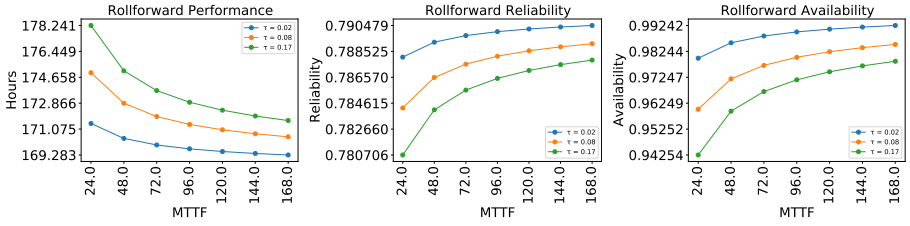
Rejuvenation: The rejuvenation pattern restores the affected system to reduce the impact of a fault, error, or failure. Fig. 5 demonstrates performance, reliability and availability of the Rejuvenation pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days). t_d and $T_l + T_r$ is 2 second. $T_{e,f}$ is 0.5 hour. T_s is 1, 5 and 10 minutes. The MTTF M_u of the unprotected part of the system is 720 hours (30 days). Restoring the affected system results in higher execution time.

Reinitialization: The reinitialization pattern restores the affected system to its initial state to reduce the impact of a fault, error, or failure. Fig. 6 demonstrates performance, reliability and availability of the Reinitialization pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days). t_d,T_i, and T_r is 2 second. The MTTF M_u of the unprotected part of



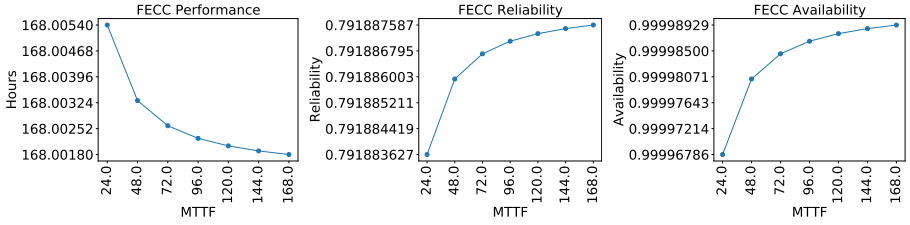
(a) Performance (b) Reliability (c) Availability

Fig. 7. Rollback pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

Fig. 8. Rollforward pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

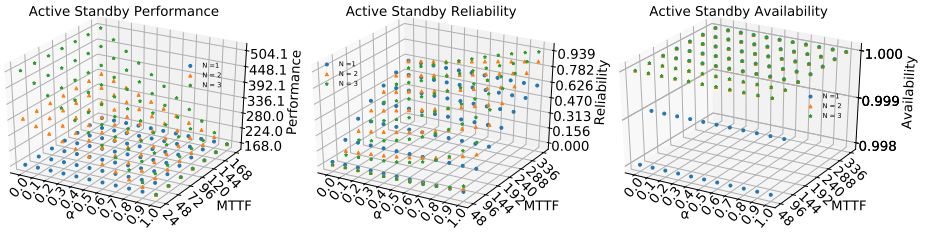
Fig. 9. Forward Error Correction Code pattern performance, reliability, and availability

the system is 720 hours (30 days). Execution time increases significantly as the application executes from the start whenever required.

Rollback: The rollback pattern restores the system to the last checkpoint before a fault, error, or failure. Fig. 7 demonstrates the performance, reliability, and availability of the Rollback pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days), the time to save system state/progress to storage T_s is 1, 5 and 10 minutes, $T_l + T_r$ is 1 seconds, and the MTTF M_u of the unprotected part of the system is 720 hours (30 days). Faster storage results in better performance, reliability, and availability.

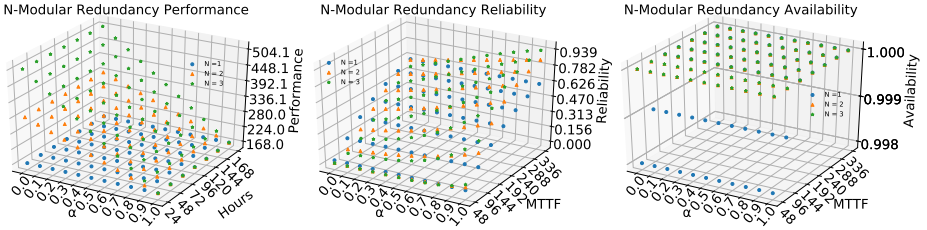
Rollforward: The rollforward pattern restores the system to the time when a fault, error, or failure. Fig. 8 demonstrates the performance, reliability, and availability of the Rollforward pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days), the time to save system state/progress to storage T_s is 1, 5 and 10 minutes, $T_l + T_r$ is 1 seconds, and the MTTF M_u of the unprotected part of the system is 720 hours (30 days). Rollforward results in better performance, reliability, and availability than rollback as the system restores to the point when a fault, error, or failure occur.

Forward Error Correction Code: The Forward Error Correction Code (FECC) pattern applies redundancy to system state or resources to reduce the impact of a fault, error, or failure. Fig. 9 demonstrates performance, reliability and availability of the Forward Error Correction Code pattern. The task's execution time T_E is 168 hours (7 days), MTTF M is 24-168 hours (1-7 days).



(a) Performance (b) Reliability (c) Availability

Fig. 10. Active/Standby pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

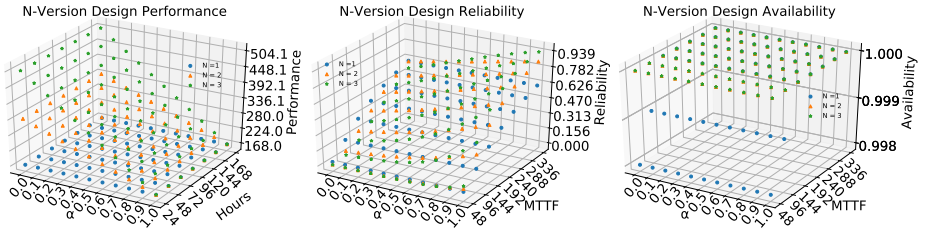
Fig. 11. N-modular Redundancy pattern performance, reliability, and availability

T_a,$t_{en} + t_d$, and T_c is 2 second. The MTTF M_u of the unprotected part of the system is 720 hours (30 days). Redundancy allows better performance, reliability, and availability than other patterns discussed till now. However, reliability is still low as the pattern doesn't employ redundancy fully.

Active/Standby: The Active/Standby pattern applies redundancy in the form of N functionally identical replicas to reduce the impact of a fault, error, or failure. Fig. 10 demonstrates the performance, reliability and availability of the Active/Standby pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, $t_i + t_d + t_r$ is 2 seconds, and T_f is 1 minute. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Reliability increases significantly but redundant systems overhead increases execution time significantly.

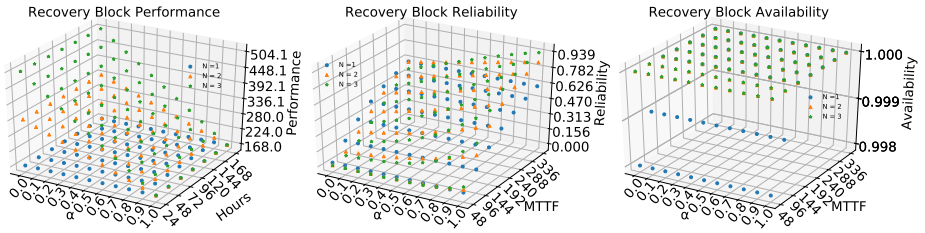
N-modular Redundancy: The N-modular redundacy pattern applies redundancy in the form of N functionally identical replicas to maintain continuous correct operation of a system. Fig. 11 demonstrates the performance, reliability and availability of the N-modular Redundancy pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, $t_i + t_o$ is 1 second, and T_r is 1 minute. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability remain same as the active/standby pattern as the parameters remain almost same.

N-Version Design: The N-version design applies redundancy as N functionally equivalent alternate system implementations to handle a fault, error, or failure. Fig. 12 demonstrates the performance, reliability and availability of the N-Version Design pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space,



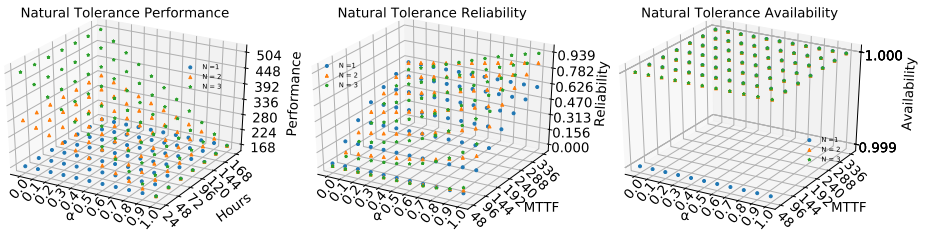
(a) Performance (b) Reliability (c) Availability

Fig. 12. N-Version Design pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

Fig. 13. Recovery Block pattern performance, reliability, and availability



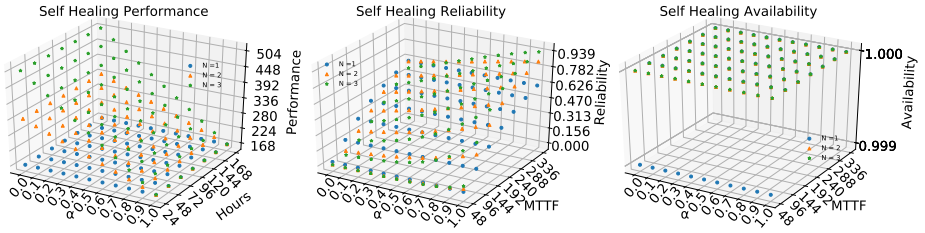
(a) Performance (b) Reliability (c) Availability

Fig. 14. Natural Tolerance pattern performance, reliability, and availability

α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, $t_i + t_o$ is 1 second, and T_r is 1 minute. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability are same as the active/standby pattern as the parameters are almost same.

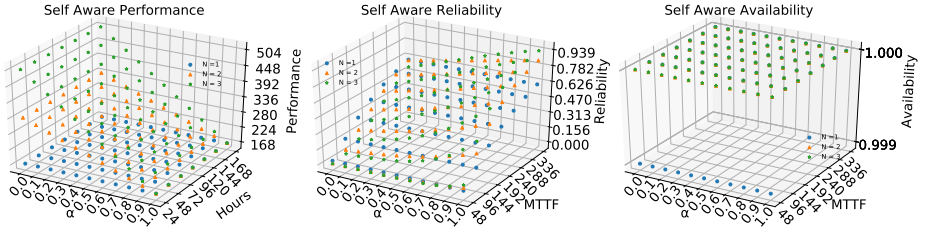
Recovery Block: The recovery block pattern applies redundancy as a functionally equivalent alternate system implementation encapsulated in a recovery block. Fig. 13 demonstrates the performance, reliability and availability of the Recovery Block pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, $t_i + t_o$ is 1 second, and T_r is 1 minute. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability are same as the active/standby pattern as the parameters are almost same.

Natural Tolerance: The natural tolerance pattern uses implicit error/failure detection and self-masking to reach a correct system state from an illegal system state. Fig. 14 demonstrates the performance, reliability and availability of the Natural Tolerance pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, t_d



(a) Performance (b) Reliability (c) Availability

Fig. 15. Self-Healing pattern performance, reliability, and availability



(a) Performance (b) Reliability (c) Availability

Fig. 16. Self-Aware pattern performance, reliability, and availability

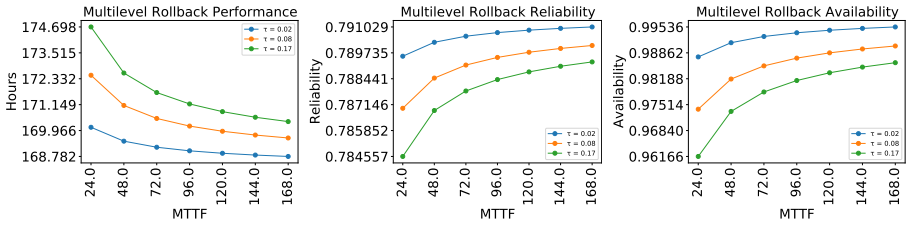
is half second, and T_m is 30 seconds. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability improve a little from the active/standby pattern as the parameters T_m improve by 30 seconds as compared to T_f.

Self-Healing: The self-healing pattern uses explicit error/failure detection and self-correction to reach a correct system state from an illegal system state. Fig. 15 demonstrates the performance, reliability and availability of the Self-Healing pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). T_a is 1 second, t_d is half second, and T_c is 30 seconds. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability are same as the natural tolerance pattern as the parameters remain almost same.

Self-Aware: The self-aware pattern uses explicit error/failure detection and self-correction to reach a correct system state from an illegal system state. Fig. 16 demonstrates the performance, reliability and availability of the Self-Aware pattern. The task's execution time T_E is 168 hours (7 days). To demonstrate performance, redundancy N is 1, 2 or 3 and in time and space, α between 0 and 1, and MTTF M is 24-168 hours (1-7 days). t_m, T_a, and T_o is 1 second. T_c is 30 seconds. To demonstrate reliability and availability, redundancy N is 1, 2 or 3 and in space with $\alpha = 1$, the MTTF M is 48-336 hours (2-14 days in 1 day increments). Performance, reliability, and availability remain same as the natural tolerance pattern as the parameters remain almost same.

4.2 Pattern Combinations

Multi-level Rollback: Recent work [8] detailed prior solutions and proposed a new approach for offering a separate resilience strategy for computation offloaded to a general-purpose computing graphics processing unit (GPGPU) accelerator.



(a) Performance (b) Reliability (c) Availability

Fig. 17. Multi-level Rollback performance, reliability, and availability

While the application itself is employing the Rollback pattern (level $l = 0$), an additional Rollback pattern is employed for the offloaded computation (level $l = 1$) to contain and mitigate GPGPU errors and failures using a more efficient strategy. The GPGPU computation is rolled back to a locally stored checkpoint upon an error or failure. The performance, reliability, and availability are calculated based on the parameters for each pattern, making the GPGPU resilience pattern a subsystem of the application resilience pattern.

While the application is waiting for the offloaded computation to finish, it is assumed that no other computation takes place and there is no need to save system state and progress to storage at level 0. Therefore, the application's failure free performance $T_{f=0}$ and performance under failure T are composed of the corresponding performances at level 0 and 1 (Eqs. 7 and 8). The reliability $R(t)$ 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. 9). The availability A can be calculated using the task's execution time without any resilience strategy T_E and the performance under failure T (Eq. 10).

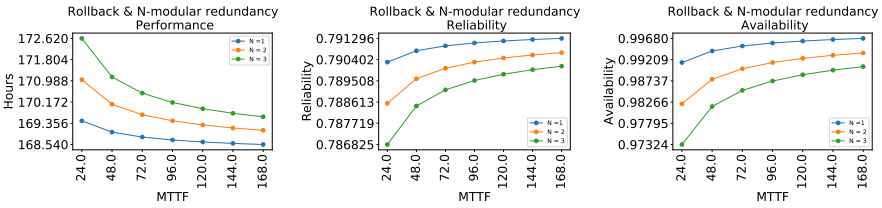
$$T_{f=0} = T_{f=0,l=0} + T_{f=0,l=1} \quad (7) \qquad R(t) = e^{-\lambda_u T} = e^{-T/M_u} \quad (9)$$

$$T = T_{l=0} + T_{l=1} \quad (8) \qquad A = \frac{T_E}{T} = \frac{T_E}{T_{l=0} + T_{l=1}} \quad (10)$$

Fig. 17 shows the performance, reliability and availability of 2-level Rollback using the parameters from in Fig. 7 with 80% of the task's execution time T_E offloaded to a GPGPU, the time to save GPGPU state/progress to node-local storage $T_{s,l=1}$ of 1 second and the time to load it and to roll it back the same. Multi-level rollback provides better performance, reliability, and availability than normal rollback pattern.

Rollback and N-modular Redundancy: The recent work OpenMP target offload resilience [8] also considered employing the N-modular Redundancy pattern. In this case, GPGPU errors and failures are detected and potentially corrected using redundancy. The performance, reliability, and availability are calculated similarly to the multi-level Rollback based on the parameters for each pattern (Eqs. 7-10).

Fig. 18 shows the performance, reliability, and availability of this solution using the parameters from Fig. 7, where 80% of T_E offloaded to a GPGPU. GPGPU redundancy N is 1, 2, or 3 and in time ($\alpha = 1$), the times to replicate the input T_i and to compare the outputs T_o are 0. The time to reboot a GPGPU and use it again for redundancy T_r and the MTTR R are 1 minute. Inclusion of redundancy further improves performance, reliability, and availability than rollback pattern.



(a) Performance (b) Reliability (c) Availability

Fig. 18. Rollback and N-modular Redundancy performance, reliability, and availability

5 Conclusion

We introduced the RDPM tool, which allows exploring the design space for resilience solutions in HPC systems. It applies the resilience design pattern concept and models the performance, reliability and availability of resilience solutions. The parameterized resilience patterns can be employed horizontally, i.e., covering different parts of the system, or vertically, i.e., covering subsets of each other. The tool is easily extensible to new patterns and provides results in plots and CSV files. Future work involves extending the RDPM tool with power consumption models. The ultimate goal of this longer-term effort is to enable hardware/software codesign for performance, resilience and power consumption.

Acknowledgements

This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program managers Robinson Pino and Lucy Nowell. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

References

1. Ashraf, R., Hukerikar, S., Engelmann, C.: Pattern-based modeling of multiresilience solutions for high-performance computing. In: ACM/SPEC International Conference on Performance Engineering. pp. 80–87 (2018). https://doi.org/10.1145/3184407.3184421
2. Ashraf, R., Hukerikar, S., Engelmann, C.: Shrink or substitute: Handling process failures in HPC systems using in-situ recovery. In: Euromicro International Conference on Parallel, Distributed, and network-based Processing. pp. 178–185 (2018). https://doi.org/10.1109/PDP2018.2018.00032
3. Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (Jan 2004). https://doi.org/10.1109/TDSC.2004.2
4. Daly, J.T.: A higher order estimate of the optimum checkpoint interval for restart dumps. Future Generation Computer Systems **22**(3), 303–312 (2006). https://doi.org/10.1016/j.future.2004.11.016
5. Di, S., Bautista-Gomez, L., Cappello, F.: Optimization of a multilevel checkpoint model with uncertain execution scales. In: IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 907–918 (2014). https://doi.org/10.1109/SC.2014.79
6. Elliott, J., Kharbas, K., Fiala, D., Mueller, F., Ferreira, K., Engelmann, C.: Combining partial redundancy and checkpointing for HPC. In: International Conference on Distributed Computing Systems. pp. 615–626 (2012). https://doi.org/10.1109/ICDCS.2012.56

7. Engelmann, C., Ong, H.H., Scott, S.L.: The case for modular redundancy in large-scale high performance computing systems. In: IASTED International Conference on Parallel and Distributed Computing and Networks. pp. 189–194 (2009)
8. Engelmann, C., Vallée, G.R., Pophale, S.: Concepts for OpenMP target offload resilience. In: International Workshop on OpenMP. pp. 78–93 (2019). https://doi.org/10.1007/978-3-030-28596-8_6
9. Fiala, D., Mueller, F., Engelmann, C., Ferreira, K., Brightwell, R., Riesen, R.: 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. pp. 78:1–78:12 (2012). https://doi.org/10.1109/SC.2012.49
10. Geist, A.: How to kill a supercomputer: Dirty power, cosmic rays, and bad solder. IEEE Spectrum **10**, 2–3 (2016)
11. Hukerikar, S., Engelmann, C.: A pattern language for high-performance computing resilience. In: European Conference on Pattern Languages of Programs. pp. 12:1–12:16 (2017). https://doi.org/10.1145/3147704.3147718
12. Hukerikar, S., Engelmann, C.: Resilience design patterns: A structured approach to resilience at extreme scale. Journal of Supercomputing Frontiers and Innovations **4**(3), 4–42 (Oct 2017). https://doi.org/10.14529/jsfi170301
13. Hukerikar, S., Engelmann, C.: Resilience design patterns: A structured approach to resilience at extreme scale (version 1.2). Tech. Rep. ORNL/TM-2017/745, Oak Ridge National Laboratory (Aug 2017). https://doi.org/10.2172/1436045
14. Koren, I., Krishna, C.M.: Fault-Tolerant Systems. Morgan Kaufmann (Jul 2007)
15. Kumar, M., Engelmann, C.: Models for resilience design patterns. In: 2020 IEEE/ACM 10th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS). pp. 21–30. IEEE (2020)
16. Levy, S., Ferreira, K.B.: An examination of the impact of failure distribution on coordinated checkpoint/restart. In: Workshop on Fault-Tolerance for HPC at Extreme Scale. p. 35–42 (2016). https://doi.org/10.1145/2909428.2909430
17. Ostrouchov, G., Maxwell, D., Ashraf, R., Engelmann, C., Shankar, M., Rogers, J.: 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)
18. Pham, H.: Reliability Modeling, Analysis and Optimization. World Scientific (2006)
19. Snir, M., et al.: Addressing failures in exascale computing. International Journal of High Performance Computing Applications **28**(2), 127–171 (May 2014). https://doi.org/10.1177/1094342014522573
20. Tiwari, D., Gupta, S., Vazhkudai, S.S.: 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. pp. 25–36 (2014). https://doi.org/10.1109/DSN.2014.101
21. Trivedi, K.S., Malhotra, M.: Reliability and Performability Techniques and Tools: A Survey, pp. 27–48. Springer (1993). https://doi.org/10.1007/978-3-642-78495-8_3
22. Young, J.W.: A first order approximation to the optimum checkpoint interval. Communications of the ACM **17**(9), 530–531 (Sep 1974). https://doi.org/10.1145/361147.361115