Future Generation Computer Systems 160 (2024) 120–130

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Resilience and performance quantification of dynamic reconfiguration

Sarah Alhozaimy [a,*], Daniel A. Menascé [b], Massimiliano Albanese [b]

[a] *Software Engineering Department, College of Computer and Information Sciences, King Saud University, 11495, Riyadh, KSA*
[b] *School of Computing, George Mason University, 4400 University Dr, Fairfax, 22030, VA, USA*

ARTICLE INFO

Keywords:
Resilience quantification
Performance analysis
System reconfiguration
Dynamic reconfiguration
Resilience-performance tradeoff
Optimal reconfiguration frequency

ABSTRACT

Dynamic reconfiguration is an adaptive resilience mechanism that can help address several system design problems. Adaptation through dynamic reconfiguration can improve quality of service, increase fault-tolerance, help recover from failures, and prevent and recover from cyber attacks. This mechanism acts primarily by reconfiguring one or more of a system's resources. While system reconfiguration is advantageous, it may bring disadvantages such as performance and availability degradation during reconfiguration intervals. In this work, we quantify the effectiveness of dynamic reconfiguration as a system resilience mechanism and its impact on performance. We define a failure function that captures the effect of dynamic reconfigurations on a system's resilience to failures and develop metrics that capture the impact of reconfigurations on a system's execution time and probability of failure. We also derive analytic models that predict the effectiveness of dynamic reconfigurations on execution time and resilience to failures. Several theorems regarding the tradeoff between resilience to failures and performance and availability are presented. Finally, we define an optimization problem, formalized with the help of these theorems, to determine the optimal reconfiguration frequency to meet performance-resilience tradeoffs.

1. Introduction

When a computer system runs for a long period of time, the process corresponding to the software in execution starts to age or degrade in performance and in system resource utilization. System failures may be caused by the accumulation of internal error states during a system's execution. For example, a failure may result from the accumulation of incorrect values in the random access memory or leaks resulting in poor use of existing memory resources. The failure rate may increase with the complexity of the software and with the duration of its execution time. In real-world systems, even the best engineered systems will eventually be disrupted by residual defects in the software or hardware. This may cause the system to fail to perform its functions or to meet its quality requirements. It is important that a system continues to carry out its mission and be resilient to failures.

This work uses dynamic reconfiguration as a proactive mechanism to increase a system's resilience to failures. Dynamic reconfiguration is an adaptive resilience mechanism that holds a great promise to solve several problems. This technique can be used to improve Quality of Service (QoS) [1], increase fault-tolerance [2,3], recover from failures [4], prevent cyber attacks through Moving Target Defenses (MTDs) [5–7], and facilitate the detection of and recovery from cyber attacks [8]. Dynamic reconfiguration has several important benefits, but it can also introduce some drawbacks such as degradation of a system's transient performance. To address this problem, this paper focuses on analyzing the impact of using dynamic reconfigurations on system performance and resilience tradeoffs.

Dynamic reconfiguration acts primarily by reconfiguring one or more of a system's resources. Despite the large number of proposed reconfiguration techniques in the literature [3,9], most of them are designed for and evaluated with respect to *specific* reconfiguration mechanisms. Gaps still exist with respect to understanding the impact of dynamic reconfiguration mechanisms on system performance in *general* and in analyzing the tradeoffs between system resilience to failures and performance of any computer system. This paper analyzes the impact of using *dynamic reconfigurations* as a mechanism for reducing the impact of failures on a system, regardless of the reconfiguration technique used.

While dynamic reconfiguration is generally desired, its practical implementation is currently limited, in part due to the difficulty of balancing consistency and disruption of system service. Our approach is based on the use of the version consistency mechanism as a criterion for safe dynamic reconfigurations of component-based distributed systems [10]. A similar approach used in the design and implementation of resilient distributed component-based software systems is described in [4].

* Corresponding author.
E-mail addresses: salhozaimy@ksu.edu.sa (S. Alhozaimy), menasce@gmu.edu (D.A. Menascé), malbanes@gmu.edu (M. Albanese).

https://doi.org/10.1016/j.future.2024.05.040
Received 20 July 2023; Received in revised form 11 March 2024; Accepted 17 May 2024
Available online 24 May 2024

Previous work [11] by two of the authors of this paper dealt with a similar approach but very different problem. In [11], the authors were preoccupied with proactively protecting a system from *cyberattacks* by using dynamic reconfigurations aimed at disrupting an attacker's reconnaissance effort. In the current paper, we are concerned with proactively increasing the resilience of a system against *failures* by using dynamic reconfigurations.

Failures and cyberattacks are disruptive to a computer system and its users. However, they are different in the sense that cyberattacks attempt to breach one or more of the following properties: confidentiality, integrity, and availability. Failures interrupt the operation of part or of an entire system. Thus, it is possible for a system to continue to operate while suffering a cyberattack. It is also possible for a system to fail without suffering a cyberattack.

The main contributions of this paper are: (1) a closed-form analytic model that can be used to compute (a) the probability that a system fails during execution and (b) the system's execution time when dynamic reconfiguration is used (see Eqs. (3) and (4), and Theorem 1); (2) an optimization model for determining the optimal system reconfiguration frequency in a way that takes into account given tradeoffs between performance and resilience (see Theorem 2 and Eq. (17)); and (3) the derivation of the maximum probability that a system does not fail under non-uniform reconfiguration intervals (see Theorem 3).

The remainder of the paper is organized as follows. Section 2 introduces a reconfiguration taxonomy and discusses related work. Section 3 refines the problem statement using an experimental example and presents the notation, assumptions, and basic system model considered in the paper. Section 4 presents our resilience analytic model used to study the tradeoffs between performance and resilience under dynamic reconfigurations. Then, Section 5 presents several numerical examples that illustrate the applicability of the proposed models. Section 6 presents an optimization model that can be used to determine the optimal system reconfiguration frequency subject to given performance constraints. The next section discusses in more detail the differences between the current paper, which is focused on tradeoffs between failures and performance, and the paper in [11] from two of the authors of this paper, that deals with tradeoffs between cybersecurity and performance. Finally, Section 8 presents concluding remarks.

2. Background and related work

This paper focuses on the use of reconfiguration to increase system resilience while a system is in execution. As indicated in Fig. 1, we should consider four aspects of system reconfiguration: *why* to reconfigure, *when* to reconfigure, *what* to reconfigure, and *how* to reconfigure. The red boxes in Fig. 1 illustrate that the main focus of this paper is the improvement of fault tolerance considering QoS tradeoffs through dynamic reconfiguration.

Why to reconfigure. As Fig. 1 depicts, reconfiguration can be used to improve QoS [1], increase fault-tolerance [3], recover from failures [4], prevent cyber attacks [5,6], and recover from cyber attacks [8]. In many circumstances, reconfiguration can help achieve several of these objectives at the same time. But, the focus of the work in this paper is on fault tolerance improvement and failure recovery.

When to reconfigure. It is important to determine the best time to reconfigure a system as well as the reconfiguration frequency. This is a critical problem that can affect a system's performance and its availability. A higher reconfiguration frequency increases a system's resilience but it may also decrease its performance and availability. A lower reconfiguration frequency increases the probability of failures.

The literature has commonly referred to three types of reconfiguration: proactive, reactive, and hybrid [6]. Proactive reconfiguration regularly changes configurations through adaptations executed at an established or random interval. Proactive adaptations help mitigate failure occurrence. Reactive adaptation takes place when a failure-related event generates the need to reconfigure. Reactive and proactive adaptation may be combined in a hybrid reconfiguration scheme [5].

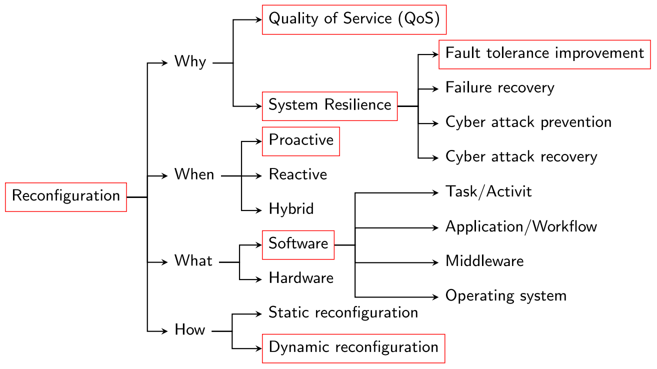


Fig. 1. A reconfiguration taxonomy as in [11] with the difference that the focus of this paper (see red boxes) is different from that of [11].

What to reconfigure refers to which part of a system should be altered, tuned, and reshaped to suit its purposes. It is possible to reconfigure various layers of a system: hardware and software which includes operating system, middleware, application, and task. For example, reconfigurations could be applied at the hardware layer to tolerate faults such as machine crashes and network connectivity errors.

At the software layer, as indicated in [12], 84% of all causes of software failures are related to memory addressing, lack of responsiveness, and exception handling, which also correlate to the duration of time a flawed piece of code runs.

How to reconfigure refers to determining the best way for changing a system's configuration. This change can happen at run time, in which case it is called dynamic reconfiguration, or can be static, in which case reconfiguration takes place before the execution of a system [13]. Static reconfiguration requires the knowledge of reconfigured resource characteristics beforehand. On the contrary, dynamic reconfiguration requires fewer assumptions about resource characteristics before execution starts.

Related approaches. Table 1 considers approaches that tackle a problem similar to the one dealt with in this paper. The columns of the table include the domain addressed by each approach, the strategy (proactive or reactive) used for reconfiguration, the type of model used for the analysis, the metrics involved in the analysis, whether and how QoS tradeoffs are analyzed, and finally, which validation method is used.

We refer the reader to Table 1 of [14] for a list of related work for 2011 and earlier. That reference does a very good job in terms of categorizing QoS management and optimization of dynamic service-based systems. The study in [15] uses models to distinguish tradeoffs between resilience and different system designs (e.g., default design, design with a GPU removed, design with a camera added, design with a new node) to determine the most resilient version of the system . Differently, from our work, these tradeoffs are not computed through closed-form expressions. As Table 1 demonstrates, the work described in our paper differs from other work in the area of dynamic reconfigurable systems mainly in the sense that (a) it can be applied to any generic architecture; and (b) it provides theorems and closed-form expressions to compute the tradeoff between resilience to failures, defined as the probability of failures, and execution time and availability.

Multiple dynamic reconfiguration techniques have been proposed in the literature to mitigate failures. One of the earliest such techniques is proactive software rejuvenation that counteracts software aging by stopping the execution of a software system, clearing its internal state, and restarting it [2,3].

The work in [21] presents a comparative experimental study of the main software rejuvenation techniques developed to mitigate software aging effects, where the overhead of rejuvenation techniques is related

Table 1
Overview of related approaches.

Ref. No.	Problem domain	Architecture	Reconfiguration strategy	Model	Metrics	QoS tradeoffs	Validation
[16]	Resilience quantification method for large scale systems	Generic	Proactive	Stochastic analytic state space models (monolithic models or interacting sub-models)	Performance (i.e., job rejection rate) and Resilience (i.e., settling time and peak time)	–	Case study for a IaaS cloud
[17]	Prediction of the QoS of dynamic reconfiguration of a single-core CPU and multi-core CPU model with stable and varying workload	Components	Proactive	Analytical model	Throughput and elapsed real time	–	Simulations
[18]	Architecture-level self-adaptive framework that can activate alternative patterns at runtime to cope with increasing demands or recover from failures	Client/server-concurrency architecture patterns (dynamic-thread-creation, half-sync/half-async, and leader/followers)	Reactive	Queuing network	Arrival rate, response time, request drop ratio, system throughput, thread pool utilization and CPU utilization	–	Experiments
[14]	Adaptive service-based systems that integrate several tools into a complete tool suite to model QoS requirements through dynamic adaptation to changes in system state, environment, and workload	Service-based	Proactive	Markov models	Performance and reliability	–	Simulation and experiments based on an adaptive service-based system
[15]	Runtime self-reconfiguration infrastructure mainly focused on failure mitigation in mobile platforms of Cyber–Physical Systems	Layered architecture	Reactive	Colored Petri Net-based (CPN)	Computation time for different variations of system model and reliability	Tradeoff between system resilience and system designs	Case study deployed on a cluster of fractionated satellite
[19]	Coordination support for distributed adaptations in aspect-oriented middleware that support reconfiguration of multiple aspects; and coordination of changes between different nodes	Layered architecture	Proactive	–	Service disruption, total reconfiguration time, and overhead during normal operation	–	Experiment based general example
[20]	Self-architecting software system framework that adjusts an architecture automatically in response to changes in the QoS characteristics of the underlying environment	Layered architecture	Reactive	Analytical model	Availability, execution time, and throughput	–	Experiments
This work	Analysis of the tradeoffs between resilience to failures and the performance of systems that use dynamic reconfiguration	Generic	Proactive	Closed-form analytical models	Execution time, availability, and reliability (i.e., probability of failure)	Tradeoff between execution time and probability of failure	Theorems and experimental proof of concept

to their granularity. The work in [22] analyzes the optimal rejuvenation scheduling to maximize a system's availability and minimize its loss probability. The work in [23] derives analytically the software rejuvenation timing that maximizes the limiting interval reliability or the interval reliability with exponentially distributed operation time. However, the model presented in our work is not limited to software rejuvenation techniques; other dynamic reconfiguration techniques could be used to improve system resilience to failures such as diversification and randomization using MTDs.

The reader is also referred to a survey [9] on dynamically reconfigurable systems that classified 77 papers based on different factors including the hardware and software platforms, methodologies and techniques involved, the domains of application and the result achieved by the studies. The work in [24] proposes Flex-MPI that enables MPI applications to dynamically reconfigure the number of processes to allow the application to complete within a specified time interval under a performance constraint. A computational prediction model also takes into account an efficiency constraint by minimizing the number of dynamically reconfigured processes and a cost constraint by minimizing the operational cost. The work in [25] proposes a taxonomy that clarifies existing concepts for modeling a change and quantifying the change impact on QoS of reconfigurable systems. Most of the papers cited by [25] consider metrics such as performance and cost while our work considers a quantitative analytical approach to model the impact of dynamic reconfiguration and performance predictions to analyze the tradeoff between performance and resilience to failures. Our approach can be applied to any generic system architecture.

3. Refined problem statement

This section refines the problem statement through the use of an experiment to motivate the need for the work. We then provide the notation and assumptions of our proposed analytical model. Finally, we give an overview of the basic reconfiguration system model considered in this paper.

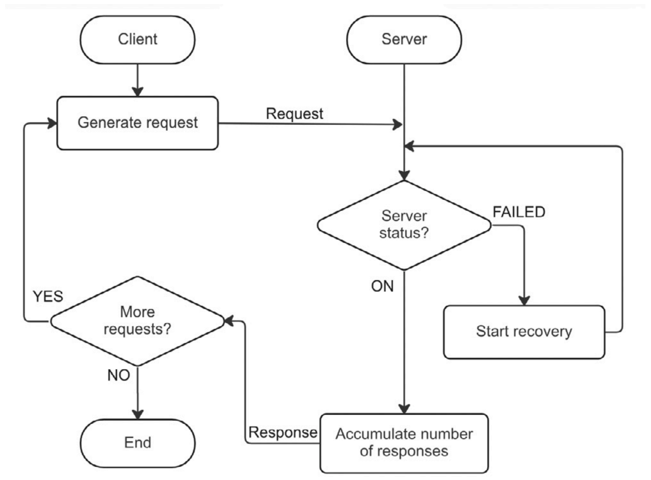


Fig. 2. Flowchart of the motivation experiment.

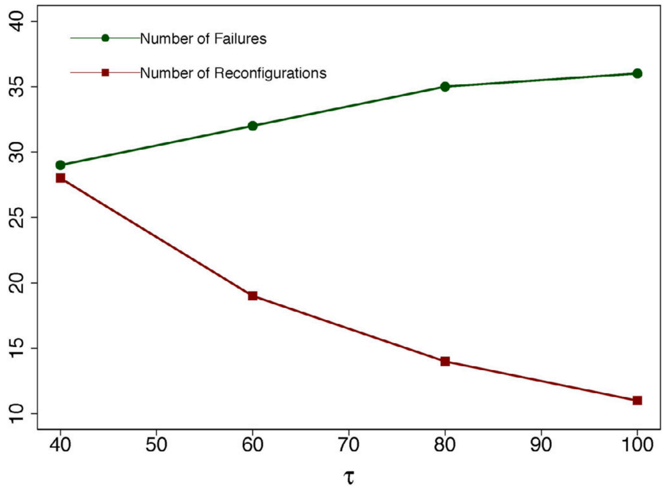


Fig. 3. Client server experiment results. X-axis in sec. $E = 1050$ s; MTBF = 4000 s; $F(t) = 1 - e^{-t/\text{MTBF}}$; reconfiguration time $\rho = 2$ s; reboot time = 4 s.

3.1. Experimental motivation

To motivate this study, we designed an experiment including a client and a server running on two separate machines communicating over TCP as illustrated in Fig. 2. The client goes through several iterations of sending requests to the server, which can be in one of two possible states, On or Failed, according to some probability distribution. If the server is determined to be in a Failed state, a recovery process (a reboot in our case) is started. At the end of the recovery process, the server status is checked again. If the server is On, the server accumulates the number of responses and sends a response to the client. The client then generates another request to the server and a new client–server iteration is started. The time during which the client is in the On state is called τ

Fig. 3 shows the results of these experiments. The top (green) curve shows that the number of failures increases with the duration of the execution segment (τ). The bottom (red) curve shows that the number of reconfigurations decreases with the duration of the execution segment. So, a higher reconfiguration rate (i.e., a lower value of τ) leads to less failures.

3.2. Notation

We use the following notation throughout the paper.

- n: number of execution segments between two consecutive renewal events.
- τ_j: duration of the jth execution segment.
- ρ_j: duration of the reconfiguration interval that follows the jth execution segment.
- $F(t)$: failure function defined as the probability that a system fails at time $t_f \leq t$.
- P^r_{nf}: probability that a failure does not occur while executing a system when reconfiguration is used.
- P^{nr}_{nf} probability that a failure does not occur while executing a system when reconfiguration is not used.
- E: system execution time without reconfiguration.
- E_r: system execution time under reconfiguration.
- A: system availability, i.e., percentage of time the system is executing, which does not include the time it is reconfiguring. Formally, $A = E/E_r$.
- X: expansion factor, i.e., the factor by which the system execution time increases due to reconfiguration. Formally, $X = E_r/E = 1/A$.
- E_r^{max}: constraint on the maximum value of E_r.

3.3. Assumptions

We make the following assumptions:

- A1: A system saves its state, before a reconfiguration starts, so that it does not need to restart after the reconfiguration is complete.
- A2: Failures only occur during execution segments and not during reconfigurations, which can be hardened using mechanisms such as the one presented in [4].
- A3: Failures during execution segment j are assumed to be independent from failures during execution segment i for $i \neq j$. The rationale for this assumption stems from the fact that, when a system is reconfigured, the reconfigured system has different properties (e.g., different software modules, resulting in different potential opportunities for failures).
- A4: The value of the failure function does not change after the system has completed execution. This assumption follows from the fact that failures can only take place while the system is in execution.
- A5: When a system reconfigures, the value of its failure function resets to zero after the reconfiguration is complete.

3.4. Basic reconfiguration system model

The basics of the reconfiguration model considered here are illustrated in Fig. 4. We consider a 24/7 system subject to *renewal events* that typically last for a few hours (e.g., to install security patches and/or new versions of a system for maintenance).

Fig. 4 shows that the time between two consecutive renewal events is when the system execution takes place. This time is called system *execution time*. In Fig. 4(a), the system execution time, E, is not interrupted by reconfigurations. However, in Fig. 4(b), the system execution time is broken into a sequence of *execution segments* of duration τ_j $(j = 1, \dots, n)$ interleaved with *reconfiguration intervals* of duration ρ_j $(j = 1, \dots, n-1)$. Examples of reconfiguration include reboots or any moving target defense [5–7] technique. As discussed before, the purpose of these reconfigurations is to improve fault tolerance. After a reconfiguration is complete, the system's execution resumes from where it was paused. (see Assumption 1 in Section 3.3).

4. Resilience and performance quantification

In this section, we provide our definition of failure function that we use to measure system resilience through dynamic reconfigurations. We then derive closed-form expressions for the metrics defined in Section 3.2. The approach proposed in this paper uses analytic models that

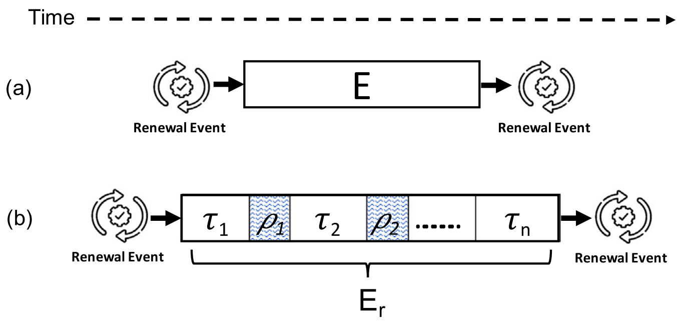


Fig. 4. (a) System execution time E when no reconfiguration is used, and (b) System execution time E_r with $n-1$ reconfigurations and n execution segments. The jth execution segment lasts for τ_j time units and the jth reconfiguration interval lasts for ρ_j time units.

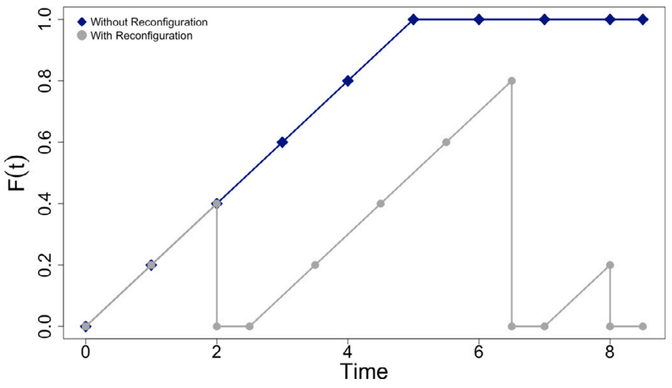


Fig. 5. Blue line: a linear example of the $F(t)$ function without reconfiguration: $F(t) = t/5$ for $t \le 5$ and $F(t) = 1$ for $t > 5$. Gray line: a modified linear failure function with reconfiguration for τ_j time units and reconfiguration duration $\rho = 0.5$.

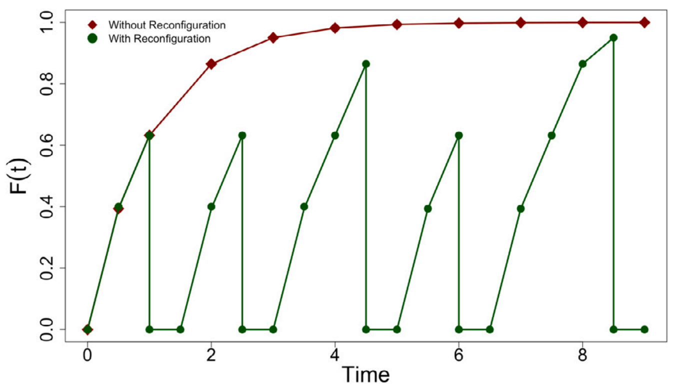


Fig. 6. Exponential examples of the $F(t)$ function: Red line: an exponential function without reconfiguration: $F(t) = 1 - e^{-t}$ for $t \le 5$ and $F(t) = 1$ for $t > 5$. Green line: modified exponential failure functions with reconfiguration for τ_j time units and reconfiguration duration $\rho = 0.5$.

Table 2
Failure probability p_{fail} without reconfiguration.

Case No.	Condition	p_{fail}
I	$E \le t^*$	$F(E)$
II	$E > t^*$	1

Table 3
Failure probability p_{fail} under reconfiguration.

Case No.	Condition	p_{fail}
I	$E \le \min(t^*, \tau_j)$	$F(E)$
II	$\tau_j \le \min(t^*, E)$	$F(\tau_j)$
III	$t^* \le \min(E, \tau_j)$	1

quantify the effectiveness of dynamic reconfigurations on execution time and system resilience to failures.

Let $F(t) \mapsto [0,1]$, called the *failure function*, be the probability that a system fails at time $t_f \le t$.

This function is similar to a cumulative distribution function and by definition is monotonically increasing.

That means that as a system ages, its reliability decreases or stays constant (see e.g., software aging). This is reasonable because failures tend to be cumulative over time.

Let t^* denote the system *maximum lifetime*, i.e., the smallest value of t such that $F(t) = 1$. In other words, t^* is the time elapsed between the completion of a reconfiguration and the next system failure. Note that when a system starts to be reconfigured, the failure function $F(t)$ is reset to zero. We will come back to this at the end of Section 8.

Figs. 5 and 6 show examples of two failure functions: a linear one (in blue) and an exponential one (in red). The linear function represents the case in which the system resilience decreases linearly with time. The exponential function represents a case in which the system resilience decreases exponentially with time.

In both cases, the system fails at time equal to 5 units (i.e., $t^* = 5$). A larger value of t^* translates into a more resilient system.

According to assumption A4 in Section 3.3, the value of the failure function does not change after the system has completed execution. So, assume that the failure functions shown in Figs. 5 and 6 are for systems that complete execution at time $t = 2$. The maximum value of the probability that the system fails while executing is 0.4 for the linear function and 0.865 for the exponential function. Thus, $F(t) \le F(E)$ for $t \le E$.

Consider now the effect of dynamic reconfiguration on the failure function for different values of τ_j as shown in Figs. 5 and 6 in gray for the linear one and in green for the exponential one. If the reconfiguration period τ_j is less than the execution time E of the system, then the maximum value of the failure function is $F(\tau_j)$. Thus, $F(t) \le F(\tau_j)$ for $t \le \tau_j$. For example, if $\tau_1 = 1$ for the exponential failure function of Fig. 6 and if the reconfiguration time is $\rho_1 = 0.5$, then the maximum value of the failure function at $t = 1$, is $F(1) = 0.63$.

We can then compute the probability p_{fail} that the system fails when there are no reconfigurations as summarized in Table 2. In case I, the system completes execution before a failure occurs. So, p_{fail} is the value of the failure function when the system completes execution. If the system completes execution after the failure threshold t^* (case II in Table 2), the system would have failed before the system completes and $p_{\text{fail}} = 1$.

As summarized in Table 3, there are other cases to address under reconfiguration. In case I, the system completes execution before the first reconfiguration and before the system fails. So, p_{fail} is the value of the failure function when the system completes. In case II, the reconfiguration occurs before the system ends ($\tau_j \le E$) and before the failure function reaches its threshold ($\tau_j \le t^*$). So, p_{fail} is the value of the failure function at time τ_j, because due to assumption A5; the value of failure function associated with the component restarts after the reconfiguration is complete. In case III, the failure function reaches the value 1 before the system completes and before reconfiguration occurs. So, $p_{\text{fail}} = 1$.

4.1. Dynamic reconfiguration analytic model

The analytic model presented in this paper is captured in three theorems, whose proofs are presented in the Appendix. Note that the model presented in Fig. 4 allows the duration of each reconfiguration to be different. This is useful when one needs to model situations in which the reconfiguration process is not deterministic and it cannot start at fixed points in time due to dependencies with other processes. Additionally, the model allows the duration of each execution segment to be different. This is useful when one needs to model situations in which it is necessary to consider additional uncertainty into the reconfiguration process to avoid failures.

We now derive closed-form expressions for the metrics defined in Section 3.2. The execution time without reconfiguration is given by

$$E = \sum_{j=1}^{n} \tau_j. \tag{1}$$

The expression for the execution time with reconfiguration, E_r, can be easily computed by adding the summation of the reconfiguration times ρ_j to the execution time E.

$$E_r = E + \sum_{j=1}^{n-1} \rho_j. \tag{2}$$

We now compute the probability, P^r_{nf}, that a system does not fail under reconfiguration. In order for a failure not to occur under the system reconfiguration scenario, failures cannot occur in any of the execution segments.

The probability that the system does not fail during an execution segment of duration τ_j is equal to one minus the probability that the system fails during that interval, i.e., $1 - F(\tau_j)$. So, the probability that the system does not fail during system execution is the probability that it does not fail during any execution segment. This probability can be computed as the product of the non-failure probabilities for all execution segments assuming independence across all execution segments.

$$P^r_{nf} = \prod_{j=1}^{n} [1 - F(\tau_j)]. \tag{3}$$

The independence assumption is reasonable because the system is assumed to reconfigure after each execution segment.

When the system does not reconfigure, the probability P^{nr}_{nf} that the system does not fail is

$$P^{nr}_{nf} = 1 - F(E). \tag{4}$$

Theorem 1, stated below and proved in the Appendix, shows that the probability that a system does not fail when reconfiguration is used is higher than that probability when reconfiguration is not used.

Theorem 1. *Let P^r_{nf} be given by Eq. (3), P^{nr}_{nf} be given by Eq. (4), and the failure function $F(.) \mapsto [0,1]$ be a monotonically increasing function. Then, $P^r_{nf} \geq P^{nr}_{nf}$ for $j = 1, \ldots, n$.*

It is worth noting that Theorem 1 does not depend on the exact shape of the failure function $F(.)$, but only on its monotonically increasing nature.

5. Numerical results

Tables 4–6 provide several numerical results obtained using our model with eight different sets of model parameters and two types of failure functions (linear and quadratic).

Tables 4 and 5 present the input parameters for the experiments and Table 6 shows the corresponding results. For convenience, row numbers in column 1 of each table indicate how rows from Tables 4 and 5 correspond to a row in Table 6. Table 4 contains the values of input parameters τ_1/E to τ_7/E, t^*, and the computed value of E. Table 5 shows the values of $(\rho_1/E) \times 100$ through $(\rho_6/E) \times 100$. We divide the values of ρ's by E to normalize the units and multiply by 100 in order to obtain a number with a larger absolute value. Note that this is done for display purposes only. The model computations are done with the values of ρ without any normalization.

Consider for example row 3 that indicates that $E = 840$. According to Eq. (2), the execution time of the system under reconfiguration is

Table 4
Input parameters for several numerical examples for non-uniform intervals.

#	τ_1/E	τ_2/E	τ_3/E	τ_4/E	τ_5/E	τ_6/E	τ_7/E	t^*	E
1	0.0168	0.0588	0.1008	0.1429	0.1849	0.2269	0.2689	600	595
2	0.0357	0.0714	0.1071	0.1429	0.1786	0.2143	0.25	800	700
3	0.0357	0.0714	0.1071	0.1429	0.1786	0.2143	0.25	900	840
4	0.0241	0.0723	0.1205	0.1506	0.1807	0.2108	0.241	910	830
5	0.0497	0.0807	0.1118	0.1429	0.1739	0.205	0.236	1000	805
6	0.0681	0.1021	0.1234	0.1447	0.166	0.1872	0.2085	1200	1175
7	0.051	0.1156	0.1327	0.1497	0.1667	0.1837	0.2007	1500	1470
8	0.0408	0.0748	0.1088	0.1429	0.1769	0.2109	0.2449	2300	2205

Table 5
Continuation of input parameters for several numerical examples for non-uniform intervals.

#	$(\rho_1/E) \times 100$	$(\rho_2/E) \times 100$	$(\rho_3/E) \times 100$	$(\rho_4/E) \times 100$	$(\rho_5/E) \times 100$	$(\rho_6/E) \times 100$
1	0.3361	0.3361	0.3361	0.3361	0.3361	0.3361
2	0.5714	0.5714	0.5714	0.5714	0.5714	0.5714
3	0.3571	0.4762	0.5952	0.5952	0.7143	0.7143
4	0.3614	0.7229	1.0843	1.2048	1.2048	1.3253
5	1.8634	1.8634	1.8634	1.8634	1.8634	1.8634
6	0.8511	1.1064	1.5319	1.8723	2.1277	2.5532
7	1.3605	1.6327	1.7687	1.9048	2.0408	2.1769
8	1.3605	1.4059	1.4512	1.5873	1.5873	1.6327

$E_r = 869$, i.e., a 3.5% overhead compared with the no-reconfiguration scenario.

Table 6 shows that the probability that a system does not fail when reconfiguration is used is greater than or equal to the probability that a system does not fail when reconfiguration is not used as indicated by Theorem 1 ($P^r_{nf} \geq P^{nr}_{nf}$) for both linear and quadratic failure functions. In fact, the non-failure probability is much larger when reconfiguration is used than when reconfiguration is not used. For example, for the case in row 3 and a linear failure function, the probability that the system does not fail in any of the 7 execution segments is $P^r_{nf} = 0.3597$, while without reconfiguration that probability is $P^{nr}_{nf} = 0.0667$.

Table 6 also reports the availability and the expansion factor. One can see that the availability ranges from 89.94% to 98.02% as the expansion factor X decreases from 1.1118 to 1.0202. So, it is important to determine an optimal value of τ_j that maximizes the non-failure probability while keeping the execution time penalty due to reconfigurations below a certain limit (see Section 6 for a discussion of this problem).

It is important to note that our model presents closed-form analytical expressions to calculate the non-failure probability, the availability, and the expansion factor for any values of the parameters and for any monotonically increasing failure function.

This section presents several charts to illustrate the metrics derived in the previous section. Fig. 7 shows the variation of the availability A $(= E/E_r)$ vs. the ratio τ/E for $E = 100, t^* = 120$, and $\rho = 2$ (top), $\rho = 6$ (middle), and $\rho = 15$ (bottom). As τ/E approaches 1, meaning no reconfiguration, E_r tends to E and therefore the availability A tends to 1. The figure also illustrates that the availability A decreases as the reconfiguration time ρ increases because E_r grows with ρ.

Fig. 8 is another view of Fig. 7 in which the y-axis is the expansion factor $X = E_r/E$ instead of $A = E/E_r$. The figure shows that as τ tends to E, E_r tends to E. Both figures illustrate a fast variation in E_r when τ is less than 20% of E.

We now show in Fig. 9 how the non-failure probabilities P^r_{nf} and P^{nr}_{nf}, with and without reconfiguration, respectively, vary as a function of τ/E. The figure depicts two pairs of lines: one for a linear failure function $F(t)$ function and another for a quadratic failure function $F(t)$. The linear $F(t)$ function is

$$F(t) = \begin{cases} t/t^* & t \leq t^* \\ 1 & t > t^* \end{cases} \tag{5}$$

Table 6
Results for the numerical examples of Tables 4 and 5.

#	P^{nr}_{nf} linear $F(.)$	P^{r}_{nf} linear $F(.)$	P^{nr}_{nf} quadratic $F(.)$	P^{r}_{nf} quadratic $F(.)$	Availability $\times 100$ $A = E/E_r \times 100$	Expansion factor $(X = E_r/E)$
1	0.0083	0.3320	0.0166	0.8237	98.02	1.0202
2	0.1250	0.3857	0.2344	0.8702	96.69	1.0343
3	0.0667	0.3597	0.1289	0.8534	96.66	1.0345
4	0.0879	0.3692	0.1681	0.8600	94.43	1.0590
5	0.195	0.4205	0.3520	0.8945	89.94	1.1118
6	0.0208	0.3450	0.0412	0.8581	90.87	1.1004
7	0.020	0.3447	0.0396	0.8576	90.18	1.1088
8	0.0413	0.3495	0.0809	0.8486	91.76	1.0898

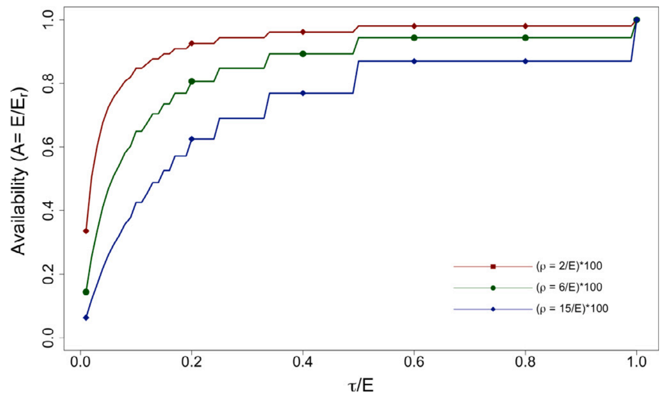


Fig. 7. Availability ($A = E/E_r$) vs. τ/E for $E = 100, t^* = 120$, and ($\rho = 2/E$) $*$ 100 (top), ($\rho = 6/E$) $*$ 100 (middle), and ($\rho = 15/E$) $*$ 100 (bottom).

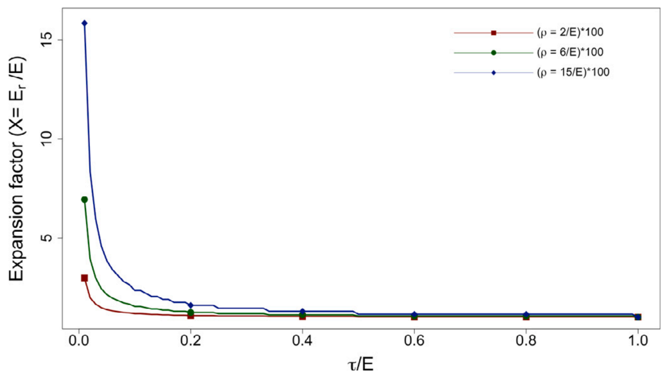


Fig. 8. Expansion factor ($X = E_r/E$) vs. τ/E for $E = 100, t^* = 120$, and $\rho = (2/E) \times 100$ (top), $\rho = (6/E) \times 100$ (middle), and $\rho = (15/E) \times 100$ (bottom).

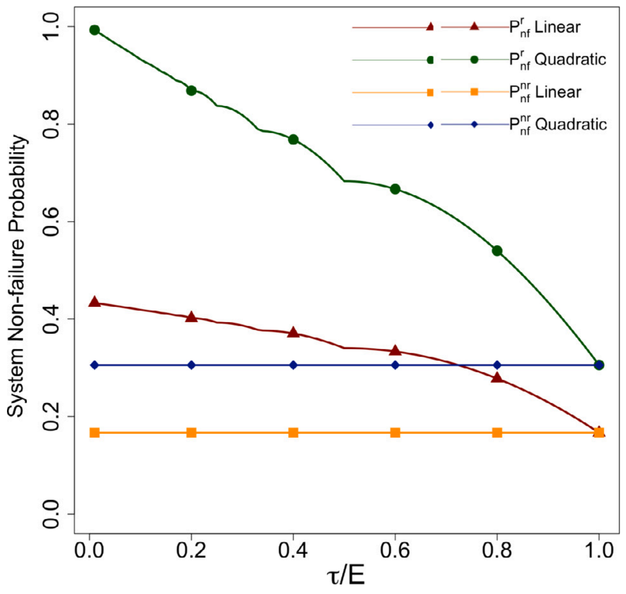


Fig. 9. P^r_{nf} and P^{nr}_{nf} vs. τ/E for $E = 100$, $t^* = 120$, $\rho = 2$ for $F(t) = t/t^*$ (linear) and $F(t) = (t/t^*)^2$ (quadratic) for $t \le t^*$ and $F(t) = 1$ for $t > t^*$.

and the quadratic $F(t)$ function is

$$F(t) = \begin{cases} (t/t^*)^2 & t \le t^* \\ 1 & t > t^*. \end{cases} \quad (6)$$

Fig. 9 shows the following:

- $P^r_{nf} \ge P^{nr}_{nf}$ for both linear and quadratic failure functions. These results are consistent with Theorem 1. This can be seen by (i) comparing the top decreasing curve (P^r_{nf} for the quadratic $F(t)$ function) with the top horizontal line (P^{nr}_{nf} for the quadratic $F(t)$) and by (ii) comparing the other decreasing curve (P^r_{nf} for the linear $F(t)$) with the bottom horizontal line (P^r_{nf} for the linear $F(t)$).
- The probability that the system does not fail while executing is higher for the quadratic failure function $F(t)$.
- P^r_{nf} is a monotonically decreasing function of τ. This is in fact demonstrated in Theorem 2 in Section 6 for any monotonically increasing $F(t)$ function.

While reconfigurations increase the resilience of a system against failures, these reconfigurations increase a system's execution time due to the overhead of reconfigurations. Fig. 10 illustrates the tradeoff between resilience to failures (indicated by a higher probability P^r_{nf} that the system does not fail while executing) and a maximum acceptable performance degradation. As the figure illustrates, as P^r_{nf} tends to 1 (i.e., the system does not fail while executing) the performance degradation, given by E_r/τ, grows very fast because reconfigurations need to occur at a high rate. For example, E_r/τ goes from 1 to 2.8 to 22 to 298 as P^r_{nf} goes from 0.3 to 0.9 to 0.9975 to 0.9999 (see Fig. 10).

That means that the y-axis indicates the overhead measured in number of execution segments. The next section formally describes the solution of the optimization problem that handles this tradeoff.

6. Optimal reconfiguration frequency

We study in this section the optimal reconfiguration frequency and start with the *uniform* case, i.e., the case in which all execution segments have the same duration (i.e., $\tau_j = \tau$, $j = 1, \dots, n$) and all reconfiguration intervals have the same duration (i.e., $\rho_j = \rho$, $j = 1, \dots, n-1$) and the system execution time E is fixed.

As we observed in previous sections, as τ increases, there are fewer reconfigurations, which in turn decreases the probability P^r_{nf} that the system does not fail while executing. Thus, a decrease in τ is detrimental to users of the system in terms of performance but is beneficial in terms of reliability. On the other hand, as τ decreases, the system's execution time of a system increases due to additional reconfigurations.

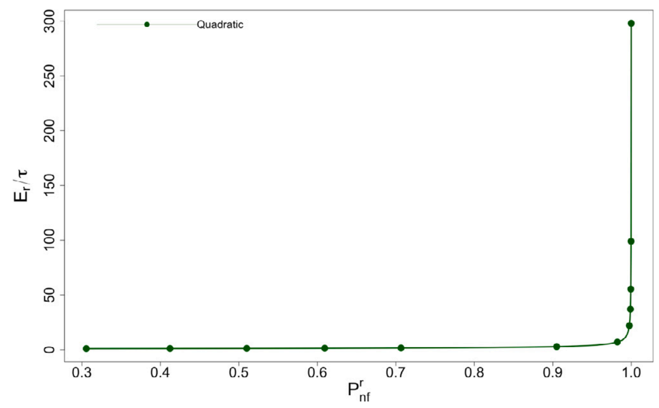


Fig. 10. E_r/τ vs. P^r_{nf} for $E = 100$, $t^* = 120$, $\rho = 2$ and $F(t) = (t/t^*)^2$ for $t \le 1$ and $F(t) = 1$ for $t > 1$.

Therefore, one can determine an optimal value of τ that maximizes the probability P^r_{nf} that the system does not fail during the execution of the system while keeping the execution time penalty due to reconfigurations below a limit provided by the system's stakeholders.

The expression for P^r_{nf} was given in Eq. (3) and is repeated below for convenience.

$$P^r_{nf} = \prod_{j=1}^{n}[1 - F(\tau_j)]. \tag{7}$$

For the uniform case, Eq. (7) becomes

$$P^r_{nf} = [1 - F(\tau)]^n. \tag{8}$$

The number of execution segments n is given as

$$n = \lceil E/\tau \rceil. \tag{9}$$

The total execution time E_r is equal to the execution time E plus the number of reconfiguration intervals, $n - 1$, multiplied by the duration of a reconfiguration interval (ρ). Thus,

$$E_r = E + \left(\left\lceil \frac{E}{\tau} \right\rceil - 1\right) \times \rho. \tag{10}$$

We can then rewrite the optimization problem as follows:
Maximize $P^r_{nf} = [1 - F(\tau)]^{\lceil E/\tau \rceil}$ $\tau \in [\rho, E]$

$$s.t.\quad E_r = E + \left(\left\lceil \frac{E}{\tau} \right\rceil - 1\right) \times \rho \le E_r^{\max} \tag{11}$$

It is worth observing that this optimization problem is non-linear and that the function $F(.)$ has to be known in order to be able to solve the problem. The solution can be simplified by observing that (i) P^r_{nf} is monotonically decreasing with τ as demonstrated by Theorem 2, which is proved in the Appendix; and that (ii) E_r is monotonically decreasing with τ as can be readily inferred from Eq. (10).

Theorem 2. *Let P^r_{nf} be given by*

$$P^r_{nf} = [1 - F(\tau)]^{\lceil E/\tau \rceil}, \tag{12}$$

$\tau \in [\rho, E]$ and $F(.) \mapsto [0, 1]$ be a monotonically increasing function. Then, P^r_{nf} is a monotonically decreasing function of $\tau \in [\rho, E]$.

Similar to Theorems 1, 2 does not depend on the exact shape of the failure function $F(.)$, but only on its monotonically increasing nature. Thus, in order to maximize P^r_{nf}, which is the objective of the optimization problem above, one should select the smallest possible value of τ according to Theorem 2. But as τ decreases, the execution time E_r under reconfiguration, given by Eq. (10), increases and may violate the constraint that $E_r \le E_r^{max}$. Thus, in order to solve the optimization problem we should select the smallest possible value of

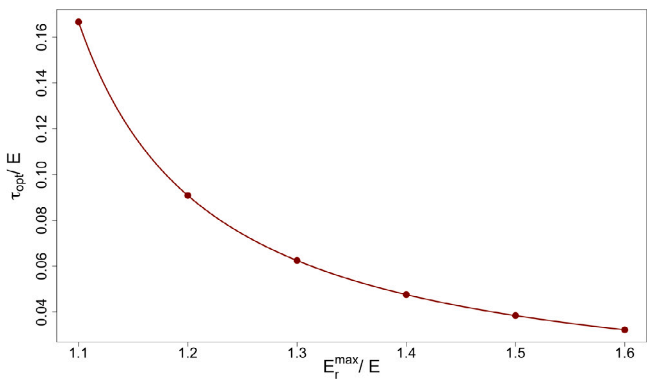


Fig. 11. τ_{opt}/E vs. $E_r^{\max}/E$ for $E = 500$ and $\rho = 10$.

τ such that $E_r = E_r^{max}$. Using Eq. (10), we have to find the value τ that satisfies the equation

$$\lceil E/\tau \rceil = 1 + \frac{E_r^{max} - E}{\rho}. \tag{13}$$

Let

$$\kappa = 1 + \frac{E_r^{max} - E}{\rho}. \tag{14}$$

Thus,

$$\lceil E/\tau \rceil = \kappa \tag{15}$$

and it follows that

$$\kappa - 1 < E/\tau \le \kappa. \tag{16}$$

So, any value of τ such that $E/\tau \in (\kappa - 1, \kappa]$ satisfies the $E_r \le E_r^{max}$ constraint. However, we need to use the smallest possible value τ to maximize the probability of no failure P^r_{nf}. Thus, according to Eq. (16) we need to use $E/\tau = \kappa$. Then, it follows that the optimal value of τ is

$$\tau_{opt} = E/\kappa = \frac{E}{1 + \frac{E_r^{max} - E}{\rho}}. \tag{17}$$

Eq. (17) shows that τ_{opt} decreases in an inversely proportional way to the slack $E_r^{max} - E$. That means that reconfigurations can become more frequent as the maximum value of E_r increases. Eq. (17) also indicates that $\tau_{opt} = E$ when $E_r^{max} = E$, which means that no reconfigurations should happen when the system execution time with reconfigurations, E_r, should not exceed its original value E.

Fig. 11 shows the variation of τ_{opt}/E as $E_r^{\max}/E$ varies from 1.1 (i.e., 10% above E) to 1.6 (i.e., 60% above E) for $E = 500$ and $\rho = 10$. The ratio τ_{opt}/E decreases from a little above 0.16 to about 0.02.

An interesting problem is to identify the values of $\tau_j, j = 1, \dots, n$ that maximize P^r_{nf}. We address this problem in the following theorem, which is proved in the Appendix.

Theorem 3. *Let P^r_{nf} be given by Eq. (3), $E = \sum_{j=1}^{n} \tau_j$, and $F(.) \mapsto [0, 1]$ be a monotonically increasing function. Then the maximum value of P^r_{nf} is $[1 - F(E/n)]^n$ and $\tau_j = \tau = E/n$ for $j = 1, \dots, n$.*

Note that, as we will show below, the maximum value of P^r_{nf} can be obtained in some cases when the values of τ_j are not all equal to E/n as long as their sum is equal to E. Consider the following examples in which $E = 50$, $t^* = 120$, and $n = 5$.

Consider first the case of a linear $F(t) = t/t^*$ function. According to Theorem 3, the maximum value of the probability that the system does not fail is $(1 - 10/120)^5 = 0.65$. While this value can be obtained by setting $\tau_j = 10$ for all five execution segments, it can also be obtained in the case of a linear failure function by setting the values of τ_j in such a way that their sum is equal to E. For example, the values

Table 7
Comparison between [11] and this model.

#	[11]	This work
Aim	Prevent cyberattacks	Increase fault tolerance
Function	Reconnaissance function	Failure function
Metrics	Execution time, availability, and probability of cyberattack	Execution time, availability, and probability of failure
QoS tradeoffs	Security and performance	Reliability and performance

$\tau_1 = 8, \tau_2 = 10, \tau_3 = 12, \tau_4 = 13, \tau_5 = 7$ also provide a value of P^r_{nf} equal to 0.65.

Consider now the case of a quadratic failure function $F(t) = (t/t^*)^2$. We can use a Generalized Reduced Gradient (GRG) nonlinear solver to find the values of τ_j that maximize P^r_{nf}. The maximum value of P^r_{nf} in this case is 0.97. However, contrary to the case of a linear failure function, the maximum value of P^r_{nf} only occurs when $\tau_j = E/n$ for $j = 1, \ldots, n$ and not when $\sum_{j=1}^{n} \tau_j = E/n$.

7. Discussion

Modern computer systems have become more complex over time and traditional resilience mechanisms built around static configurations may no longer adequately protect them against cyberattacks and failures. Failures and cyberattacks are very different events and are disruptive to computer systems and their users. The former typically generate from inside the system and interrupt the operation of part or an entire system while the latter are generated from outside the system being attacked.

Table 7 shows the differences between our previous work [11] – which used dynamic reconfiguration to proactively protect a system from cyberattacks aimed at disrupting an attacker's reconnaissance effort – and the current work. In this paper, we are concerned with proactively increasing the resilience of a system against failures by using dynamic reconfigurations. The analytical model presented in [11] relies on the *reconnaissance function* and this work relies on a different function, *the failure function*. Each work could be extended in different ways, investigating how reconnaissance functions can be learned in practice and failure functions from system execution logs. This model could be extended to analyze the tradeoff between the loss of availability due to reconfigurations and the loss of availability due to system failures or re-configured services. The aim of this work and the work in [11] is to provide comprehensive models that analyze the impact of dynamic reconfiguration on system resilience to failures and cyber attacks.

8. Conclusions and future work

We considered dynamic reconfiguration as a mechanism to improve a system's resilience by proactively reconfiguring it so that the probability that it fails is reduced. As an example, Fig. 12 depicts a framework for self-reconfiguration based on the results of the previous sections. The failure function $F(t)$ could be learned from observations and monitoring of the computer system behavior or from the history of a similar system. This function, along with the maximum execution time $E_r^{\max}$ of the system and its execution time E without reconfiguration is used to select a reconfiguration policy Π from a reconfiguration policy database. A reconfiguration policy includes: (a) the reconfiguration frequency $1/\tau_{opt}$ and (b) the mechanism used to reconfigure a system. There may be many different possible reconfiguration mechanisms as described in Section 2. The selected policy Π is passed along to the reconfiguration engine that reconfigures the computer system according to the reconfiguration mechanism and with the optimal frequency $1/\tau_{opt}$ (see Eq. (17)).

This paper demonstrated several theorems regarding the use of dynamic reconfiguration to reduce the incidence of failures. All the results presented rely on the fact that the failure function $F(.) \mapsto [0, 1]$

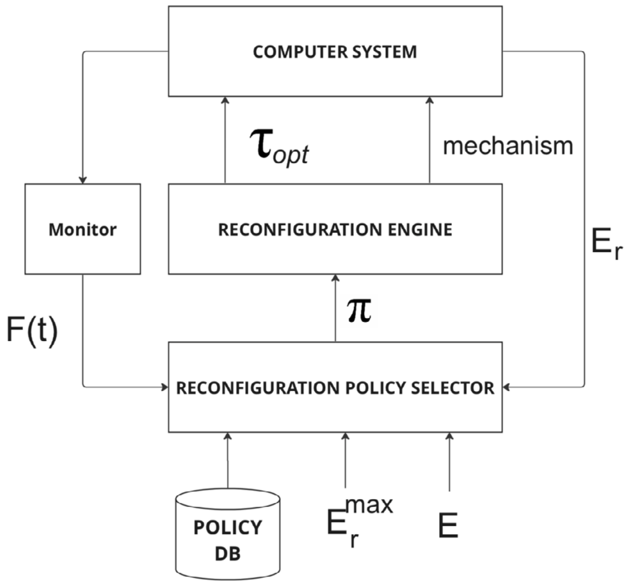


Fig. 12. Self-reconfiguration approach.

is monotonically increasing. The specific shape of $F(.)$ is not relevant for the results we presented.

We derived closed form expressions for the probability P^r_{nf} that the system does not fail while executing if reconfiguration is used and for the probability P^{nr}_{nf} that the system does not fail while executing if reconfiguration is not used. Theorem 1 proved that the probability P^r_{nf} that the system does not fail with reconfiguration is greater than or equal to the probability when no reconfiguration is done.

According to Theorem 2, the probability P^r_{nf} that the system does not fail when reconfiguration is used is a monotonically decreasing function of the duration of the execution segment. Based on Theorems 1 and 2, we derived a model for optimizing the system reconfiguration frequency that takes into account given resilience-performance tradeoffs. We also derived an expression to characterize the duration of an execution segment that maximizes the probability that the system does not fail when reconfiguration is used subject to a given execution time constraint (Theorem 3).

As future work, one could consider that the failure function $F(t)$ is not reset to zero when a reconfiguration occurs. A partial reconfiguration could be modeled by $F(t)$ being reset to some value above zero.

CRediT authorship contribution statement

Sarah Alhozaimy: Conceptualization, Formal analysis, Methodology, Project administration, Software, Validation, Writing – original draft. **Daniel A. Menascé:** Conceptualization, Formal analysis, Methodology, Supervision, Validation, Writing – original draft. **Massimiliano Albanese:** Conceptualization, Supervision, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix. Proof of Theorems

Theorem A.1. *Let P^r_{nf} be given by Eq. (A.1), P^{nr}_{nf} be given by Eq. (A.2), and the failure function $F(.) \mapsto [0,1]$ be a monotonically increasing function. Then, $P^r_{nf} \geq P^{nr}_{nf}$ for $j = 1, \ldots, n$.*

$$P^r_{nf} = \prod_{j=1}^{n}[1 - F(\tau_j)]. \tag{A.1}$$

$$P^{nr}_{nf} = 1 - F(E). \tag{A.2}$$

Proof. Let us compare P^r_{nf} with P^{nr}_{nf}, in order to establish the conditions under which reconfiguration increases the probability that the system does not fail while executing. Because both P^r_{nf} and $P^{nr}_{nf} \in [0.1]$, $\ln P^{nr}_{nf} \leq 0$ and $\ln P^r_{nf} \leq 0$. Since $\ln(.)$ is a monotonically increasing function, $P^r_{nf} \geq P^{nr}_{nf} \iff \ln P^r_{nf} \geq \ln P^{nr}_{nf}$.

According to our previous discussion, we know that $\tau_j \leq E$ and $F(\tau_j) \leq F(E)$. Thus,

$$1 - F(\tau_j) \geq 1 - F(E). \tag{A.3}$$

From Eq. (1) in Section 4.1, we know that

$$E = \sum_{j=1}^{n} \tau_j. \tag{A.4}$$

Hence, $\tau_j \leq E$. Because $F(.)$ is a monotonically increasing function, $F(\tau_j) \leq F(E)$ for $j = 1, \ldots, n$. Therefore,

$$[1 - F(\tau_j)] \geq [1 - F(E)] \ \forall \ j \ = 1, \ldots, n \tag{A.5}$$

Because $\ln(.)$ is a monotonically increasing function, it follows that

$$\ln[1 - F(\tau_j)] \geq \ln[1 - F(E)]. \qquad \text{for } j = 1, \ldots, n \tag{A.6}$$

From Eq. (A.1), it follows that

$$\ln P^r_{nf} = \sum_{j=1}^{n} \ln[1 - F(\tau_j)]. \tag{A.7}$$

From Eq. (A.6), it follows that each term of the summation in Eq. (A.7) is $\geq \ln[1 - F(E)]$. Hence, the summation in Eq. (A.7) is $\geq \ln[1 - F(E)]$, which is equal to $\ln P^{nr}_{nf}$. Thus, it follows that

$$\ln P^r_{nf} \geq \ln P^{nr}_{nf} \iff P^r_{nf} \geq P^{nr}_{nf} \quad \square \tag{A.8}$$

Theorem A.2. *Let P^r_{nf} be given by*

$$P^r_{nf} = [1 - F(\tau)]^{\lceil E/\tau \rceil}, \tag{A.9}$$

$\tau \in [\rho, E]$ and $F(.) \mapsto [0,1]$ be a monotonically increasing function. Then, P^r_{nf} is a monotonically decreasing function of $\tau \in [\rho, E]$.

Proof. Consider $P^r_{nf}(\tau) = [1 - F(\tau)]^{\lceil E/\tau \rceil}$ and $P^r_{nf}(\tau + \Delta\tau) = [1 - F(\tau + \Delta\tau)]^{\lceil E/(\tau+\Delta\tau) \rceil}$ where $0 < \Delta\tau \ll \tau$. Then,

$$\ln P^r_{nf}(\tau) = \lceil E/\tau \rceil \ln[1 - F(\tau)] \tag{A.10}$$

and

$$\ln P^r_{nf}(\tau + \Delta\tau) = \lceil E/(\tau + \Delta\tau) \rceil \times \ln[1 - F(\tau + \Delta\tau)] \tag{A.11}$$

But,

$$\lceil E/\tau \rceil \geq \lceil E/(\tau + \Delta\tau) \rceil. \tag{A.12}$$

Because $F(.)$ and $\ln(.)$ are monotonically increasing functions,

$$\ln[1 - F(\tau)] \geq \ln[1 - F(\tau + \Delta\tau)]. \tag{A.13}$$

Combining Eqs. (A.11), (A.12), and (A.13) we get that

$$\ln P^r_{nf}(\tau) \geq \ln P^r_{nf}(\tau + \Delta\tau) \tag{A.14}$$

which implies that

$$P^r_{nf}(\tau) \geq P^r_{nf}(\tau + \Delta\tau). \tag{A.15}$$

In other words, as τ increases, P^r_{nf} decreases or stays constant. This demonstrates that $P^r_{nf}(.)$ is monotonically decreasing with τ. $\square$

Theorem A.3. *Let P^r_{nf} be given by Eq. (A.1), $E = \sum_{j=1}^{n} \tau_j$, and $F(.) \mapsto [0,1]$ be a monotonically increasing function. Therefore, the maximum value of P^r_{nf} is $[1 - F(E/n)]^n$ and $\tau_j = \tau = E/n$ for $j = 1, \ldots, n$.*

Proof. As τ_j decreases, $F(\tau_j)$ decreases because $F(.)$ is a monotonically increasing function. Then, $[1 - F(\tau_j)]$ increases and P^r_{nf}, given by Eq. (3), increases. Therefore, the maximum value of P^r_{nf} would be obtained by further decreasing all the values of τ_j. However, since $E = \sum_{j=1}^{n} \tau_j$, we can obtain the maximum value of P^r_{nf} by setting the values of all τ_j's as E/n. Therefore, the maximum value of P^r_{nf} is $[1 - F(E/n)]^n$. $\square$

References

[1] D.A. Menascé, H. Gomaa, S. Malek, J. Sousa, SASSY: A framework for self-architecting service-oriented systems, IEEE Softw. 28 (6) (2011) 78–85.

[2] D. Cotroneo, R. Natella, R. Pietrantuono, S. Russo, A survey of software aging and rejuvenation studies, J. Emerg. Technol. Comput. Syst. 10 (1) (2014) http://dx.doi.org/10.1145/2539117.

[3] T. Dohi, A. Avritzer, K. Trivedi, Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions, World Scientific Publishing Company Pte Limited, 2020.

[4] E. Albassam, J. Porter, H. Gomaa, D.A. Menascé, DARE: A distributed adaptation and failure recovery framework for software systems, in: IEEE (Ed.), 2017 IEEE Intl. Conf. Autonomic Computing, ICAC, 2017, pp. 203–208.

[5] B.C. Ward, S.R. Gomez, R.W. Skowyra, J.N. Martin, J.W. Landry, Survey of Cyber Moving Targets Second Edition, Tech. Rep. January, MIT, 2018.

[6] J.-H. Cho, D.P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T.J. Moore, D.S. Kim, H. Lim, F.F. Nelson, Toward proactive, adaptive defense: A survey on moving target defense, Commun. Surv. Tutor. 22 (1) (2020) 709–745, http://dx.doi.org/10.1109/COMST.2019.2963791.

[7] W. Connell, D.A. Menascé, M. Albanese, Performance modeling of moving target defenses with reconfiguration limits, IEEE Trans. Dependable Secure Comput. 18 (01) (2021) 205–219.

[8] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N.O. Tippenhauer, H. Sandberg, R. Candell, A survey of physics-based attack detection in cyber-physical systems, ACM Comput. Surv. 51 (4) (2018) http://dx.doi.org/10.1145/3203245.

[9] G. Fornari, V.A. Santiago Junior, Dynamically reconfigurable systems: A systematic literature review, J. Intell. Robot. Syst. 95 (3–4) (2019) 829–849, Springer.

[10] X. Ma, L. Baresi, C. Ghezzi, V. Panzica La Manna, J. Lu, Version-consistent dynamic reconfiguration of component-based distributed systems, in: ACM (Ed.), Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, New York, NY, USA, 2011, pp. 245–255, http://dx.doi.org/10.1145/2025113.2025148.

[11] S. Alhozaimy, D.A. Menascé, A formal analysis of performance-security tradeoffs under frequent task reconfigurations, Future Gener. Comput. Syst. 127 (2022) 252–262, http://dx.doi.org/10.1016/j.future.2021.09.005.

[12] C.A.R. Dos Santos, R. Matias, K.S. Trivedi, A multisite characterization study on failure causes in system and applications software, in: IEEE (Ed.), 2021 XI Brazilian Symposium on Computing Systems Engineering, SBESC, 2021, pp. 1–8, http://dx.doi.org/10.1109/SBESC53686.2021.9628276.

[13] J.C. Lyke, C.G. Christodoulou, G.A. Vera, A.H. Edwards, An introduction to reconfigurable systems, Proc. IEEE 103 (3) (2015) 291–317, http://dx.doi.org/10.1109/JPROC.2015.2397832.

[14] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, G. Tamburrelli, Dynamic QoS management and optimization in service-based systems, IEEE Trans. Softw. Eng. 37 (3) (2011) 387–409, http://dx.doi.org/10.1109/TSE.2010.92.

[15] S. Pradhan, A. Dubey, T. Levendovszky, P.S. Kumar, W.A. Emfinger, D. Balasubramanian, W. Otte, G. Karsai, Achieving resilience in distributed software systems via self-reconfiguration, J. Syst. Softw. 122 (2016) 344–363, http://dx.doi.org/10.1016/j.jss.2016.05.038.

[16] F. Longo, R. Ghosh, V.K. Naik, A.J. Rindos, K.S. Trivedi, An approach for resiliency quantification of large scale systems, SIGMETRICS Perform. Eval. Rev. 44 (4) (2017) 37–48, http://dx.doi.org/10.1145/3092819.3092825.

[17] W. Li, W. Guo, QoS prediction for dynamic reconfiguration of component based software systems, J. Syst. Softw. 102 (2015) 12–34, http://dx.doi.org/10.1016/j.jss.2014.12.001.

[18] C.-H. Lung, X. Zhang, P. Rajeswaran, Improving software performance and reliability in a distributed and concurrent environment with an architecture-based self-adaptive framework, J. Syst. Softw. 121 (2016) 311–328, http://dx.doi.org/10.1016/j.jss.2016.06.102.

[19] E. Truyen, N. Janssens, F. Sanen, W. Joosen, Support for distributed adaptations in aspect-oriented middleware, in: Proceedings of the 7th International Conference on Aspect-Oriented Software Development, AOSD '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 12–131, http://dx.doi.org/10.1145/1353482.1353497.

[20] D.A. Menascé, J.M. Ewing, H. Gomaa, S. Malek, J.P. Sousa, A framework for utility-based service oriented design in SASSY, in: Proc. First Joint WOSP/SIPEW Intl. Conf. Performance Engineering, in: WOSP/SIPEW '10, ACM, 2010, pp. 27–36.

[21] J. Alonso, R. Matias, E. Vicente, A. Maria, K. Trivedi, A comparative experimental study of software rejuvenation overhead, Perform. Eval. 70 (3) (2013) 231–250, http://dx.doi.org/10.1016/j.peva.2012.09.002, Special Issue on Software Aging and Rejuvenation.

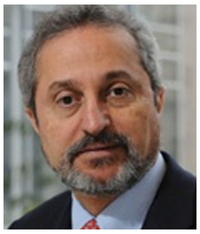
[22] G. Ning, J. Zhao, Y. Lou, J. Alonso, R. Matias, K.S. Trivedi, B.-B. Yin, K.-Y. Cai, Optimization of two-granularity software rejuvenation policy based on the Markov regenerative process, IEEE Trans. Reliab. 65 (4) (2016) 1630–1646, http://dx.doi.org/10.1109/TR.2016.2570539.

[23] T. Dohi, J. Zheng, H. Okamura, K.S. Trivedi, Optimal periodic software rejuvenation policies based on interval reliability criteria, Reliab. Eng. Syst. Saf. 180 (2018) 463–475, http://dx.doi.org/10.1016/j.ress.2018.08.009.

[24] G. Martin, D.E. Singh, M.-C. Marinescu, J. Carretero, Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration, Parallel Comput. 46 (2015) 60–77, http://dx.doi.org/10.1016/j.parco.2015.04.003.

[25] A. Hakamian, F. Klinaku, A. van Hoorn, S. Becker, Resilience, survivability, and elasticity: A taxonomy for change impact quantification of reconfigurable systems, in: 2020 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, 2020, pp. 267–274, http://dx.doi.org/10.1109/ISSREW51248.2020.00084.

Sarah Alhozaimy is an Assistant Professor at King Saud University's Department of Software Engineering. She received a Ph.D. degree in Computer Science from George Mason University. She received her M.Sc. in Advanced Computer Science from the University of Manchester. Her research interests include autonomic computing, moving target defense, security performance tradeoffs, and software engineering.



Daniel A. Menascé is a University Professor Emeritus of Computer Science at George Mason University. He received a Ph.D. degree in Computer Science from UCLA in 1978 and is a Fellow of the IEEE and the ACM. Menascé received the prestigious 2021 David J. King Teaching Award from Mason and the 2017 statewide Outstanding Faculty Award by the State Council of Higher Education for Virginia. The Computer Measurement Group awarded him the 2001 lifetime A.A. Michelson Award. Menascé authored over 280 technical papers and seven books dealing with web technologies, secure e-commerce, computer networks, queuing models, and capacity planning.

Massimiliano Albanese is an Associate Professor at George Mason University's Department of Information Sciences and Technology, and the Department's Associate Chair for Research. He is an Associate Director of the Center for Secure Information Systems. Dr. Albanese obtained his Ph.D. in Computer Science and Engineering from the University of Naples Federico II in 2005 and worked as a Postdoctoral Researcher at the University of Maryland before joining George Mason University in 2011. His research focuses on Information and Network Security, with particular interest in Cyber Attack Modeling and Detection, Cyber Situational Awareness, Moving Target Defense, and Vulnerability Metrics.