Software Aging Detection and Rejuvenation Assessment in Heterogeneous Virtual Networks

Alberto Avritzer*, Andrea Janes†, Andrea Marin‡, Catia Trubiani§,
Andre van Hoorn¶, Matteo Camilli‖, Daniel S. Menasché**, André B. Bondi††
*EsulabSolutions Inc., Princeton (NJ), USA, Princeton, USA †Free University of Bozen-Bolzano, Italy ‡Università Ca' Foscari di Venezia, Italy§Gran Sasso Science Institute, Italy ¶Universität Hamburg, Germany ‖Politecnico di Milano, Italy **Universidade Federal do Rio de Janeiro, Brazil ††Software Performance and Scalability Consulting LLC, Red Bank (NJ), USA

***Abstract*—In this paper, we report on the application of resiliency enforcement strategies that were applied to a microservices system running on a real-world deployment of a large cluster of heterogeneous Virtual Machines (VMs). We present the evaluation results obtained from measurement and modeling implementations. The measurement infrastructure was composed of 15 large and 15 extra-large VMs. The modeling approach used Markov Decision Processes (MDP). On the measurement testbed, we implemented three different levels of software rejuvenation granularity to achieve software resiliency. We have discovered two threats to resiliency in this environment. The first threat to resiliency was a memory leak that was part of the underlying open-source infrastructure in each VM. The second threat to resiliency was the result of the contention for resources in the physical host, which is dependent on the number and size of VMs deployed to the physical host. In the MDP modeling approach, we evaluated four strategies for assigning tasks to VMs with different configurations and different levels of parallelism. Using the large cluster under study, we compared our approach of using software aging and rejuvenation with the state-of-the-art approach of using a network of VMs deployed to a private cloud without software aging detection and rejuvenation. In summary, we show that in a private cloud with non-elastic resource allocation in the physical hosts, careful performance engineering needs to be performed to optimize the trade-offs between the number of VMs allocated and the total memory allocated to each VM.**

***Index Terms*—software resiliency, software aging, software rejuvenation**

I. Introduction

SOFTWARE aging is the phenomenon of increasing failure rate that long-running software exhibits. Contention or depletion of shared resources are some examples of the causes of software aging. Software aging was first discovered as a result of the application of quantitative approaches to estimate software reliability [1]. The term "software rejuvenation" was also coined in [1]. Software rejuvenation consists of restoring the software to full capacity by cleaning up system resources.

We report on the empirical results of two measurement campaigns. In the first measurement campaign, we evaluated the effectiveness of the resilience approaches introduced in this paper. These results show that the experiments with extra-large VMs (8 gigabytes of computer memory) exhibit significant performance degradation when compared to the strategies that use large (4 gigabytes of computer memory) VMs. Therefore, a second measurement campaign was initiated to help understand the root cause of the performance degradation for the extra-large VM networks. The second measurement campaign results show that the measured performance of each extra-large VM is dependent on the number and size of VMs that are running simultaneously in the physical host. This is a new result that shows that the VM network under study is suffering from two levels of aging: (1) memory leak at the VM level as shown in the detailed measurement plots, and (2) environmental aging as a result of the contention between the VMs for shared resources at the physical host. The Markov Decision Process (MDP) modeling approach presented in Section V-B assumes that each VM operates independently. The MDP model presented in Section V-C was changed to account for the number and type of VMs executing at a physical host. This model improvement was made to account for the empirical results uncovered in our second measurement campaign. The goal of both models is that of systematizing the optimal rejuvenation policy.

In summary, we show that in a private cloud with non-elastic resource allocation in the physical hosts, careful performance engineering needs to be performed to optimize the trade-offs between the number of VMs allocated and the total memory allocated to each VM.

Subsequently, to understand the impact of contention for shared resources at the physical host, we have formulated a bin-packing optimization problem [2] to derive the optimal combination of VMs of large and extra-large sizes. The optimal combination of VM sizes obtained from the solution of the bin-packing optimization problem can be used to avoid the second level of software aging due to contention for shared resources, e.g., memory, when the number extra-large VMs used are the source of memory pressure on the physical host.

Specifically, the key takeaways from the application of the methodology presented in this paper to the large heterogeneous VM cluster under study are the following: (1) avoid competition between rejuvenation and workload execution, (2) consider rejuvenation granularity (the finer, the better), (3) pay attention to contention for physical host resources.

The key contributions of this paper are the following.

(i) An empirical assessment of several **Software Rejuvenation Strategies** executed on a testbed composed of state-of-the-art components.

(ii) An **Optimal Policy Synthesis based on a Markov Decision Process (MDP) analytical model**. The MDP

models the trade-off between preventive maintenance costs, due to rejuvenation, and the risk of incurring reactive costs, due to failures. The MDP was parameterized using failure data from the two measurement campaigns.

(iii) The derivation of the optimal combination of VM sizes as the solution of the bin-packing optimization problem.

This paper is an extended version of the work presented by Avritzer et al. [3]. The blue shaded areas in Figure 1 illustrate the empirical assessment that was presented in [3], while the green shaded areas illustrate this paper's extensions. We extend the work by solving an integer programming problem to derive the optimal combination of VM sizes (Section VII-A) and running a second measurement campaign based on the solution of the integer programming problem (Section VII-B). We have uncovered a second level of software aging due to contention for shared resources. We have also extended the MDP model (Section V) to take into account the Performance Dependency between VMs (Section V-C). In addition, we have extended the related work (Section II) and we have added information about the calibration procedure (Section VI-B).

The outline of this paper is as follows. Section II provides a summary of the related work. Section III introduces the key concepts and the methodology used, while Section IV describes the open-source benchmark used for load generation, and the measurement infrastructure used to derive the empirical results. Section V introduces a parameterized Markov decision process-based model that was used to derive the optimal resiliency policy using measured parameters. Section VI discusses the experimental design and software rejuvenation strategies for the first measurement campaign conducted in this paper. Section VII presents the results of the second measurement campaign and its analysis. Section VII-A presents the integer programming solution used to select the optimal number of VMs to be assigned to a physical host. Section VII-B discusses the experimental design and software rejuvenation strategies for the second measurement campaign, which was based on the optimal configurations derived from the integer programming solution. Section VII-C presents our modeling and measurement results and Section VII-D presents our research key findings, while Section VIII provides our conclusion and recommendations for future research.

II. Related Work

The Handbook of Software Aging and Rejuvenation [5] presents a detailed overview of the fundamentals, approaches, models, and applications of software aging and rejuvenation. In the following, we provide a summary of the most closely related approaches in the context of software aging and rejuvenation models, monitoring, and empirical methodologies.

Analytic Models. A survey on software aging and rejuvenation studies is presented in [6], where the authors point out that analytical models were mainly designed for scheduling the rejuvenation time. The Markov chain model introduced in [1] has four states: (i) "Highly robust", (ii) "Failure probable", (iii) "Crash failure", (iv) "Rejuvenation". The costs associated with unscheduled downtime are modeled by the "Crash failure" state. In the "Failure probable" state, a failure will cause a transition to the "Crash failure" state and a software rejuvenation action would cause a transition to the "Rejuvenation" state. In [7], a predictive model is proposed to derive the optimal rejuvenation time while considering thresholds for the resource consumption of the aging-related indicators. In this paper, we introduce a new software rejuvenation model as shown in Figure 4. The model selects the optimal action to take at a given time by maximizing a value function. The model parameters are empirically computed using measurements from the large heterogeneous network of Virtual Machines.

Fig. 1: Illustration of the empirical assessment implemented in the paper. The visualization is based on the BPMN notation [4], where solid lines indicate process flow, dashed lines indicate data input or output.

Markov Regenerative Models. In [8], the authors have introduced a Markov regenerative stochastic Petri net to model the generally distributed timed transitions that are triggered by software rejuvenation. In [9], a Markov regenerative process model is built based on the system condition, i.e., faults are injected and contribute to determining the degradation rate of the application. The goal is to get the optimal inspection time intervals, and experiments show that the unavailability and the overall loss probability of the system were reduced. In [10], a Markov regenerative model approach is adopted to determine the optimal inspection time interval, thus maximizing application service availability. Memory leaks were injected to measure aging-related parameters in a virtualized environment, i.e., built on top of a kernel-based VM. In this paper, we developed a Markov Decision Process model that was empirically instrumented with failure probabilities that were derived from results of our measurement campaigns.

Monitoring. In [1], the authors were the first to introduce the term "software rejuvenation" in the context of software availability. They demonstrated that for client-server architectures,

periodic rejuvenation at idle times (e.g., weekends) increased server availability. In a subsequent related work [11], the best times to execute the software rejuvenation to reduce packet loss were determined by monitoring the software aging rate and real-time traf c demands. The importance of system monitoring was assessed even later in the literature. For instance, in [12], the authors present an evaluation of software rejuvenation policies (focusing on transaction systems). Experiments in [12] show that monitoring the number of completed operations (or transactions), instead of other properties (e.g., the accumulated operation time), is indeed valuable in minimizing the system failure rate.

Empirical Results. In [13], the authors presented empirical data collected at Lucent Bell Labs with the objective of developing a tool for OS resource monitoring. This effort was one of the rst systematic approaches to show the phenomena of software aging. In [14] and [15], a semi-Markov model and tool were developed to dynamically monitor workload states with the goal of predicting the time for resource degradation. All parameters in the model and the tool were empirically derived, including the model states, the residence time distributions, the states' probability of transition, and the state rewards. This work was extended in [16] to develop a tool for software aging detection in a cloud system. In [17], an empirical study is proposed to investigate aging-related bug datasets. There, machine learning techniques are adopted to improve the accuracy of a prediction model. Interestingly, in [18], the authors propose a taxonomy of aging-related bugs extracted by using popular deep-learning libraries, and employing the indicators outlined by [16].

Granularity. Multi-granularity software rejuvenation has been studied extensively, see for example [9], [19], [20]. An overview of software rejuvenation granularity concepts was presented in [21]. In this paper, we evaluate the impact of multi-granularity software rejuvenation on the large heterogeneous network ecosystem that will be presented in Section IV.

III. Key Concepts and Methodology

In this section, we present the de nition of the key concepts and a structured description of the methodology.

A. Key Concepts

Scalability requirement de nition . The approach used to de ne the scalability requirement is based on the speci ed coverage of 99.7% of the normal distribution probability mass. The main assumption used to de ne this scalability requirement is that scalable systems' response will rarely cross the response time threshold de ned by three standard deviations from the mean. Therefore, we use the average workload response time, , and standard deviation, , results measured at the low load calibration point, to compute the scalability requirement, + 3 .

Software aging detection. Violations of the de ned scalability requirement are used to trigger software aging events.

Software rejuvenation approach. Eighteen strategies for software rejuvenation are de ned in the experimental design presented in Section VI-D.

Normalized distance (ND) metric. The normalized distance metric is used to incorporate the scalability requirement into the analysis of each resiliency strategy. The normalized distance metric assesses the system's ability to meet the scalability requirement, R, and is related to the user perception of system performance, T. The normalized distance ranges from 0 2. Therefore, a resilient system will have $T \quad R$, and the normalized distance, ND, will be 1. In contrast, if the measured system performance is in violation of the scalability requirement the normalized distance, ND, will be > 1. The normalized distance is de ned in Equation 1.

Markov Decision Process (MDP). We introduce a tool for the computation of the optimal rejuvenation policy starting from the measurement campaigns that we carried out. Given the age of a machine (measured as the number of served requests), we estimate the probability of failing the normalized distance target. The MDP allows us to decide at any moment (i) which machine to use to serve the requests, (ii) when to start the rejuvenation, taking into account that this process may have a cost and hence should not be done more frequently than necessary.

B. Methodology

The methodology introduced in this paper consists of the following steps and is depicted in Figure 2:

1) De ne the system under test (Section IV).
2) Synthesize the optimal scheduling policy using the parameterized Markov decision process (MDP) based on modeling assumptions (Section V). This step aims to identify the optimal scheduling policy from rst principles. The MDP model is parameterized with results of the rst and second measurement campaigns.
3) Execute calibration experiments to de ne the scalability requirement (Section VI-A) that was used to assess the software aging of each experiment and de ne the nominal load level to be used in the experiments (Section VI-B).
4) Create an experimental design consisting of the rejuvenation strategies and the rejuvenation levels of granularity (Section VI-C and Section VI-D).
5) Execute the rst measurement campaign to evaluate the resiliency effectiveness of the rejuvenation strategies. The empirical results from the rst measurement campaign are presented in Figures 12a-17c.
6) a. Analyze the empirical results obtained from the rst measurement campaign to detect threats to resiliency (Section VII). This analysis effort consisted of investigating the root causes of threats to resiliency identi ed in the rst measurement campaign. We de ne a mixed integer programming problem to obtain the optimal heterogeneous combination of VMs (Section VII-A).
   b. Execute the second measurement campaign based on VM con gurations obtained from the mixed integer programming problem solution.
   c. Analyze the empirical results obtained from the second measurement campaign (Section VII-B).

Fig. 2: Illustration of the methodology (based on the BPMN [4] notation), where solid lines indicate process flow, dashed lines indicate data input or output.

IV. Large Heterogeneous Network Ecosystem

The two empirical evaluations of network resiliency reported in this paper were executed using resources allocated to our research project by the Hasso Plattner Institute (HPI) as part of the Future Service-Oriented Computing (SOC) Lab[1], as illustrated in Figure 3.

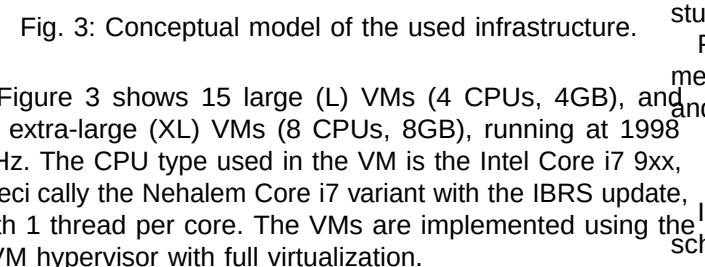


Fig. 3: Conceptual model of the used infrastructure.

Figure 3 shows 15 large (L) VMs (4 CPUs, 4GB), and 15 extra-large (XL) VMs (8 CPUs, 8GB), running at 1998 MHz. The CPU type used in the VM is the Intel Core i7 9xx, specifically the Nehalem Core i7 variant with the IBRS update, with 1 thread per core. The VMs are implemented using the KVM hypervisor with full virtualization.

The system under test was deployed to each VM using the open-source DeathStarBench benchmark [22]. This benchmark was also used to gather the test results from the HPI Large Heterogeneous Network Environment. Each HPI VM was configured by following the guidelines provided by the benchmark.

[1] https://hpide/en/research/infrastructure/future-soc-lab/

TABLE I: MDP notation table

Symbol	Description
$\langle i, j \rangle$	State of a pair of machines, where $i, j \in \{0, \ldots, K, R\}$: i is the state of the L machine, j is the state of the XL machine
K	Maximum number of test runs after which we surely rejuvenate
R	Rejuvenation state
p_i^{XL}	Failure probability of extra-large VMs at state i, $0 \leq i \leq K$ (8 CPUs, 8 GB RAM)
p_i^{L}	Failure probability of large VMs at state i, $0 \leq i \leq K$ (4 CPUs, 4 GB RAM)
$C(t)$	Total cost accumulated up to time t

All tests were run at a constant load using the compose review microservice. This microservice activates the movie review service and accesses a local movie review MongoDB database. Sections VI and VI-B present the empirical results that were obtained by running the benchmark to assess the resiliency of the large heterogeneous network ecosystem under study in this paper.

Python scripts illustrated in Figure 3 comprise the measurement infrastructure we use to execute tests, collect test results, and then analyze the data.

V. Optimal Policy Synthesis

In this section, we introduce a model to derive optimal scheduling policies. We begin by introducing the intuitive aspects of the model, followed by the model description and numerical results derived with MDPToolBox [23].

A. Intuition and Motivation

The proposed model serves to capture the benefits of proactive maintenance. Proactive maintenance is one of the most traditional applications of MDPs, as one needs to capture the mid and long-term impacts of decisions to set the optimal

actions [24]. Indeed, MDPs allow us to assess the impact of decisions over time and identify the optimal actions to be taken. When greedy solutions fall short of being optimal, MDPs provide a valuable decision-making tool by incorporating the consequences of decisions into a value function that spans a given time horizon. By incorporating the long-term effects of decisions into this value function, using dynamic or linear programming techniques, a complex multi-stage problem requiring foresight can be transformed into a more manageable single-stage greedy problem. The optimal decision, in this case, corresponds to choosing the action that maximizes the value function. Before introducing the model, we present the assumptions used to derive it:

Assumption 1: We assume that each machine L is paired with a machine XL, so the goal of the policy is to choose where to run each task at each time slot. Only one machine in each pair can execute a task at any slot. In general, the entire system consists of many pairs.

Assumption 2: The optimal policy determines, for each pair of machines, at each time slot, the optimal action to be taken.

Available actions and outcomes are presented below by following increasing levels of instantaneous costs.

Execute workload, with success: in this case, the cost may depend on whether one uses the L or XL machine.
Rejuvenate a machine, and rely on its paired machine to execute the workload: in this case, there is a rejuvenation cost, as there is a planned unavailability.
Execute workload, with a failure: in this case, the cost is the largest, as there is non-planned unavailability.

Note that the purpose of rejuvenation is to avoid non-planned failures. To this aim, the decision maker incurs an instantaneous rejuvenation cost to avoid long-term failure costs.

The MDP is parameterized using the following elements, where each item below corresponds to the respective item in the above list:

the costs associated with running an L or XL machine were assumed to be the same;
the cost associated with a rejuvenation of an L or XL machine is a tunable parameter in the model. In this work we modeled this cost to be a fraction of the assumed failure cost equal to $1=3$; and
the probability of failure, of an L or XL machine, at each slot, based on the uptime of the machine, as illustrated in Figures 5 (a)-(b).

Model description: The MDP model is illustrated in Figure 4. Let $h_i; j\ i$ be the state of the system where $i; j \in \{0; :::; K; R\}$, $K > 0$ is the maximum number of tests run after which we surely rejuvenate the machines, and R is a label that denotes that the machines are being rejuvenated. This state definition, leveraging a pair of machines, does not take into account the whole system state, as further discussed in Section V-B. Table I summarizes the notation.

The first element of the pair denotes the state of the L machine, and the second element is the state of the XL machine. As shown in Figure 4, each state has 4 actions, which are illustrated in different colors (Figures 6 and 7) and are described in the following.

Fig. 4: MDP Model.

L: execute the experiment on the L machine;
XL: execute the experiment on the XL machine;
L/XL: execute the experiment on the L machine and rejuvenate the XL machine;
XL/L: execute the experiment on the XL machine and rejuvenate the L machine.

The rejuvenation of a machine has a cost of 10. Upon the execution of a test on a machine, there can be a forced rejuvenation of that machine because of the deterioration of the response time. In this case, the cost is 10 for the test and 30 for the failure to meet the desired quality of service. Whenever the policy can choose either an L or an XL machine, the policy will opt for the latter. Let $C(t)$ be the total cost accumulated in the time interval $[0; t)$ according to the rules that we have just described. The goal of the MDP is finding the optimal rejuvenation strategy that minimizes $\lim_{t!1} C(t)=t$.

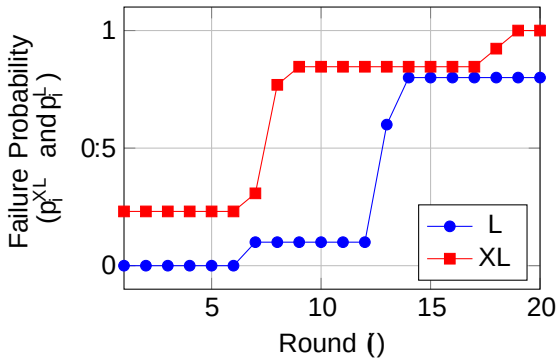
B. MDP with VM Independence Assumption

We now consider the MDP model with the following assumption:

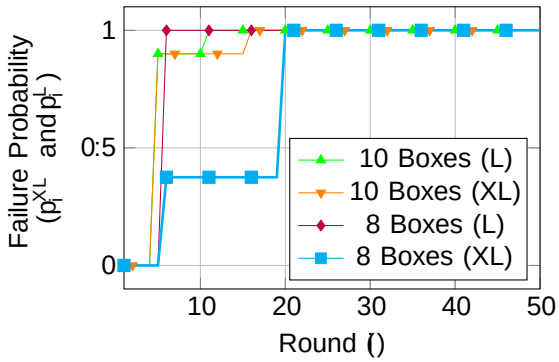
Assumption 3: All pairs of machines are independent and decoupled from each other, i.e., their rejuvenation probability is independent of the number of pairs that are operating at each time epoch.

The probability of failing to meet the desired metric on the response time after the i-th experiment, namely p_i, is determined experimentally. Figure 5 shows the failure probability over time, measured in rounds. In Figure 5(a) we consider 15 boxes, and an experiment running for 20 rounds. The failure probability of large machines (L) is dominated by that of extra-large machines (XL) because the latter consumes more resources causing more frequent failures.

Figure 6 shows the optimal policy, given all possible configurations of the L and XL machines. The entries in Figure 6 are the optimal actions chosen for each $s(i; j)$, as described in Figure 4. The optimal policy suggested by the MDP consists of using the XL machine as the main machine and the L as the backup. This corresponds to Strategy -1 in our experimental evaluation, i.e., leftmost points in Figure 18. As discussed in Section VII-D 3, whereas our experiments



(a) 15 boxes



(b) 8 and 10 boxes

Fig. 5: Failure probability increases over time, measured in rounds: (a) 15 VMs; (b) 8 VMs and 10 VMs

suggest that resource contention at the physical host is the root cause of performance degradation for Strategy -1, the optimal policy derived from the MDP assuming independence between VMs, captures VMs CPU and memory bottlenecks, which are more stringent for the L machines, motivating the use of those machines as backup, and XL as core. This finding motivated us to parameterize the MDP model to account for performance dependency between VMs in the MDP model. This new parameterization is presented in Section V-C.

C. MDP with Performance Dependency between VMs

We now assume that the VMs are functionally independent but contend for physical host resources. To address the contention for resources on the host machine, we reduced the number of VMs used in our experiments. Figure 5(b) illustrates that this adjustment resulted in the failure probability of XL machines being lower than that of L machines. This occurs because now the contention at the host machine is reduced, allowing for XL machines to perform better than L.

More formally, the probability of QoS violation after the service ofn requests is dependent on the number of VMs that are concurrently running (e.g., as shown in Table III according to our measurements).

Assumption 4:All pairs of machines are functionally independent but coupled to each other depending on the total number of VMs deployed to the physical host. This dependency is taken into account thanks to the second measurement campaign. Henceforth, we are using Assumptions 1, 2, and 4.

Figure 7 shows the optimal policy, given all possible con gurations of the L and XL machines, for two levels of VMs used: 8 machines (left) and 10 machines (right). The entries in Figure 7 are the optimal actions chosen for each state(i, j), as described in Figure 4.

The most important message that becomes evident from Figure 7 is that the optimal policy for minimizing the operating costs of the systems (in terms of QoS violations and number of rejuvenations) depends on the entire state of the cloud system and not just on the properties of the single VMs. This is because the functional independence of VMs does not imply performance independence.

The optimal policy illustrated in Figure 7, as suggested by the MDP, consists of using the XL machine as the main machine and the L machine as a backup only up to a certain combinations of states(i, j), which is dependent on the total number of VMs deployed to the physical host.

VI. Empirical Results from First Measurement Campaign

In this section, we present the steps in the methodology related to the rst measurement campaign: (1) the de nition of the scalability requirement used as a response time threshold to detect software aging (SectionVI-A), (2) the calibration procedure employed to de ne the nominal load level used (SectionVI-B), and (3) the experimental design (SectionVI-C) used to de ne the eighteen software rejuvenation strategies evaluated (Section VI-D).

The context for the development of the experimental design (SectionVI-C) of these eighteen software rejuvenation strategies was based on the initial empirical results obtained from measurements on the large heterogeneous network ecosystem under study. As we realized the trade-offs incurred in the rejuvenation granularity and the execution of immediate rejuvenation, we derived three levels of granularity and six rejuvenation strategies that evaluated the resiliency impact of adding a pause to the experiment (immediate x pause) during rejuvenation events, which machines would be rejuvenated (extra-large x large), the rejuvenation granularity (i, dc, dp), and adding a delay before triggering the rejuvenation event and synchronizing the triggered rejuvenation events (bulk rejuvenation). The detailed resiliency impact for each experiment run of these strategies are illustrated in Figures 12a- 17c and analyzed in SectionVII-C. In addition, Figures 18 and 19 illustrate the aggregated resiliency impact of each rejuvenation strategy by using the average normalized distance for each rejuvenation strategy.

We evaluate several strategies to achieve resiliency in a large heterogeneous network of VMs. We extend the policies presented in Section V, where the state was a pair of VMs, to account for a network of VMs. We have used off-the-shelf VMs and a well-assessed benchmark. We have executed experiments for the calibration of the load. As the objective is to detect performance degradation due to software aging, we employed a low-level workload that did not stress memory resources, when the system is operating at full capacity.

Fig. 6: MDP with independence assumption: optimal policy.

Fig. 7: Optimal policy obtained with MDP for 8 VMs (left) and 10 VMs (right).

We used two means to achieve resiliency in our experiments: (1) software rejuvenation and (2) load balancing to backup VMs. In addition, we can choose to run the workload in large VMs or extra-large VMs.

A. Scalability Requirement: Response Time Threshold

The calibration of the response time threshold was done as follows. We implemented an approach for calibration, where the workload is executed at a rate of 10 requests/sec for 5 minutes, at the extra-large machines. The average workload response time, , and standard deviation, results measured at this low load, are used to compute the scalability requirements for each automated experiment. We use the largest obtained measure of + 3 result as the scalability requirement. For this measurement, we obtained = 81:536 ms, and = 32:37 ms. Therefore, the response time threshold (scalability requirement) used to trigger software aging detection in our experiments is + 3 = 178:65 ms.

The figures below adopt the notation illustrated in Figure 8. For each experiment, the obtained mean response time (in ms) is shown with a cross as a marker. If the obtained response time is below the requirement, the result is shown in green, in orange otherwise. The free memory is represented with the y-axis using a blue dot as a marker. Since SOC we also want to represent load-balancing, i.e., the handover

Fig. 8: Example visualization with explanation.

to another machine, this is depicted with and the machine name at the moment the handover takes place. In Fig. 8, this handover occurs in experiment 3. From there on, also the average response times of the backup machine, as well as its free main memory, are depicted in the visualization using a gray color. As soon as the load-balancing switches back to the original machine, response times and available memory are depicted again in color (experiment 5 in Figure 8).

B. Empirical Data Used for Calibration: Single large and extra-large VMs

We executed the “compose review” microservice in the HPI environment with load requests of 1/sec, 10/sec, 20/sec, 30/sec, 40/sec, and 50/sec. Figure 9 shows the single-VM

large measurement results, while Figure 10 shows the single VM extra-large measurement results used for calibration. We used single-VM machine tests to obtain estimations of load, low and high load performance because we wanted to avoid multiple VM interactions. In these figures, we show in green shade the area of good performance, in blue dots the available memory, and in green crosses the response time measurements. Please observe that in these calibration tests rejuvenation is not triggered. We can see from Figure 9 that the loads of 40/sec and 50/sec bring the environment to overload conditions, as can be seen from the increased number of response time spikes at these loads when compared to the loads of 10/sec and 20/sec. We can also observe the increased memory consumption rate at the 40/sec and 50/sec loads in both Figures 9 and 10. Therefore, we selected 30/sec as the nominal load and used it for all the experiments of this paper.

C. Experimental Design

TABLE II: Rejuvenation Strategies (-1, 0, 1, 2, 3, 4) at three levels of granularity (Images and DB Install (i), Docker Component (dc), Docker Platform (dp)

No.	Rejuvenate immediately	Pause experiment during rejuvenation	Rejuvenated machines	Rejuvenation granularity	Bulk rejuvenation	Figure
-1i		5 [a]	extra-large	i	5 [b]	12a
0i	5		large	i		13a
1i	5	5 [a]	large	i		14a
2i			large	i	5 [b]	15a
3i	5	5 [a]	large & extra-large	i		16a
4i			large & extra-large	i	5 [b]	17a
-1dc		5 [a]	extra-large	dc	5 [b]	12b
0dc	5		large	dc		13b
1dc	5	5 [a]	large	dc		14b
2dc			large	dc	5 [b]	15b
3dc	5	5 [a]	large & extra-large	dc		16b
4dc			large & extra-large	dc	5 [b]	17b
-1dp		5 [a]	extra-large	dp	5 [b]	12c
0dp	5		large	dp		13c
1dp	5	5 [a]	large	dp		14c
2dp			large	dp	5 [b]	15c
3dp	5	5 [a]	large & extra-large	dp		16c
4dp			large & extra-large	dp	5 [b]	17c

[a] for 10 minutes
[b] every 50 minutes (10 runs)

Table II contains the experimental design that was used to evaluate different strategies for resiliency, for each rejuvenation granularity that was evaluated. We considered three levels of rejuvenation granularity, ranging from coarser to finer: (i) cleaning and reinstalling Docker images and volumes, (dp) restarting the Docker platform, and (dc) restarting the specific Docker component responsible for the memory leak. For each row in Table II we report the experiment index and the rejuvenation strategy used. We also provide for each table row a pointer to the figure illustrating the associated empirical results. The threat to resiliency was a memory leak, as shown in Figure 11, where the resident set size for a specific component increases with every experiment. This finding justified the creation of the Docker component (dc) rejuvenation strategy, which consists of restarting the process containing the memory leak.

D. Software Rejuvenation Strategies

Table II presents the software rejuvenation strategies evaluated in the first measurement campaign. The shaded areas in the Table represent the three levels of granularity used. Each strategy id number contains one of three granularity levels used: immediate (i), Docker component (dc), or Docker platform (dp). For example, Strategy -1 uses the parameters in the rows labeled -1i, -1dc, and -1dp in Table II, while Strategy 3 uses the parameters in the rows labeled 3i, 3dc, and 3dp.

In Strategy -1, the extra-large VMs are used to execute the test run. In this strategy, an extra-large VM is replaced by a large VM when rejuvenation is triggered. The test run is paused while rejuvenation is taking place. The bulk rejuvenation approach is employed as follows. The system gathers all rejuvenation events in a window of time (10 runs) and rejuvenates all these hosts simultaneously. Figure 12 illustrates the associated empirical results.

In Strategy 0-4 the large VMs are assigned to execute the test run. Strategy 0, 1, and 3 activate the software rejuvenation action as soon as a threshold violation is detected. However, for Strategy 1 and 3, a pause of the test run execution is triggered while software rejuvenation is executing. The test run pause was added to the software rejuvenation strategy to avoid the cost of rejuvenation overhead. Moreover, for Strategy 2 and 4, bulk rejuvenation was implemented instead of test run pause. For Strategy 3 and 4 both large and extra-large (core and backup) VMs were rejuvenated once threshold violation was detected, while for the other strategies, only the core VMs involved in the test run execution were rejuvenated. Figure 13-17 illustrate the associated empirical results for Strategy 0-4.

The key takeaway from these empirical results is that Strategy -1, which employs extra-large VMs as core, exhibits significant performance degradation. In contrast, some strategies that employ the large VMs as the core can achieve resiliency as defined in Section VII-C. In that section, the detailed analysis of the first measurement campaign is presented.

VII. Analysis

The first measurement campaign employed as core VM either extra-large or large VMs. However, the optimal policy results and the empirical results from the first measurement campaign indicate that a heterogeneous combination of VM sizes might create a more resilient network. These results motivated us to pose a Mixed Integer Programming problem that is presented in Section VII-A. The solution to this optimization problem is used as the experimental design for the second measurement campaign. The empirical results from the second measurement campaign are presented in Section VII-B. The key modeling and measurement results are presented in Section VII-C and takeaways are reported in Section VII-D.

Fig. 9: Machine 23, load 10, 20, 30, 40, and 50 req/sec

Fig. 10: Machine 48, load 10, 20, 30, 40, and 50 req/sec

Fig. 11: Maximum memory usage (in all machines) for each experiment of processes that grew by more than 10% overall.

A. Mixed Integer Programming Problem

The Mixed Integer Programming problem to find the optimal number of large and extra-large VMs to configure can be posed as a bin-packing optimization problem, where we have a single bin that represents the actual physical memory capacity in the host machine. Each VM memory is represented by a box. Therefore, we have a set of boxes of two different sizes, corresponding to each VM's memory size: four and eight gigabytes.

Bin Packing Algorithm. The bin packing algorithm addresses the challenge of efficiently packing a set of items (boxes) of various sizes into a limited number of bins (containers) while optimizing certain objectives. This formulation aims to minimize the total delay during the processing of these items. The goal is to assign a specific number of boxes to the bin while optimizing for average response time and satisfying specific constraints that were derived from the empirical data.

Decision Variables. Let x_i be a binary decision variable representing whether item i is assigned to the bin:

$$x_i = \begin{cases} 1, & \text{if item } i \text{ is assigned to the bin} \\ 0, & \text{otherwise} \end{cases}$$

Parameters.

B: Bin size (capacity),
n: Total number of items,
s_i: Size of item i,
k: Number of items to assign to the bin (a fixed number).

Objective: Minimize Total Delay.

The objective is to minimize the total delay, which is determined by the sizes of the assigned boxes, x_i. The delay coefficients for each box size are as follows:

$$\text{Delay Coefficients:} \begin{cases} \text{Box size } 4: & D_4 = 0.04 \text{ seconds}, \\ \text{Box size } 8: & D_8 = 0.02 \text{ seconds} \end{cases}$$

The delay coefficients for box types 4 and 8, denoted as D_4 and D_8, respectively, are defined to account for the different processing times. s_i represents the size of box i. For instance, if $s_i = 4$, the box is of size 4 GB, and if $s_i = 8$, the box is of size 8 GB. The objective function represents the total delay T_d experienced in processing all the boxes. It is computed as follows:

$$T_d = \sum_i x_i \left(D_4 \left(1 - \frac{s_i}{8} \right) + D_8 \frac{s_i}{8} \right).$$

This function combines the delays of each box size weighted by the respective coefficients D_4 and D_8. It accounts for the ratio of the individual box size to the larger box size (8) as a factor influencing the delay experienced for each box.

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 12: Mean response times and available memory, Strategy -1, machine 036

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 13: Mean response times and available memory, Strategy 0, machine 022

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 14: Mean response times and available memory, Strategy 1, machine 022

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 15: Mean response times and available memory, Strategy 2, machine 022

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 16: Mean response times and available memory, Strategy 3, machine 022

(a) Images and DB reinstall (i) rejuvenation (b) Docker component (dc) rejuvenation (c) Docker Platform (dp) rejuvenation

Fig. 17: Mean response times and available memory, Strategy 4, machine 022

Fig. 18: Arithmetic mean of the normalized distance over all iterations of the first campaign, for each experiment. Lines connect the same machine.

This formulation allows the algorithm to find an optimal solution that minimizes the total delay by adjusting the assignment of boxes based on their sizes and the delay coefficients associated with each size.

Constraints.

1) Number of Assigned Items: ensure that exactly k items are assigned to the bin: $\sum_{i=1}^{n} x_i = k$
2) Bin Capacity Constraint: ensure that the total size of assigned items does not exceed the bin capacity B: $\sum_{i=1}^{n} s_i \cdot x_i \leq B$

The algorithm solves the problem seeking an optimal solution that minimizes the overall delay.

B. Empirical results from second measurement campaign

The optimal solution of the integer programming problem that minimizes the overall delay defines the experimental design for the second measurement campaign. We present in Table III the optimal solutions that were obtained when the total number of boxes to assign to a bin was varied from 2 to 16. Table III shows the number of boxes, the optimal configuration, the modeled response time assuming VM independence, and the empirical results obtained by running the experiments at the testbed.

Tables III, IV and Figure 19 summarize the empirical results.

Table III — (1) Modeled and measured performance for optimal solutions for different numbers of boxes in the bin packing problem without rejuvenation, with an increasing number of boxes, from 2 to 16; (2) Empirical results for the optimal solutions with rejuvenation, Strategy 3dc, with an increasing number of boxes, from 2 to 16.

Table IV — Number of outliers, average, and median response times for no rejuvenation experiments

Figure 19 — Arithmetic mean of the normalized distance of the second campaign, without rejuvenation, for large (L) and extra-large (XL) VMs showing normalized distance degradation as a function of the number of VMs

We can observe the one order of magnitude improvement obtained when running the approach rejuvenation, Strategy 3dc with 14 boxes, 0.1086 seconds, as compared to the 14 boxes and no rejuvenation approach reported in Table III, 1.0981 seconds. In addition, we can observe that the rejuvenation, Strategy 3dc can significantly improve performance for the optimal configurations with 12, and 10 boxes. The 16 boxes cases produce an extremely overloaded configuration. In that case, we observe a degradation in performance when using the rejuvenation, Strategy 3dc For the smaller configurations with extra-large VMs, the contention at the physical hosts is diminished, and the no rejuvenation approach is slightly better, because the extra-large VMs don't require frequent rejuvenation in this case.

TABLE III: Comparison of optimal solutions with and without rejuvenation. Measurement in seconds.

No. of boxes	Assigned boxes	Modeled response time (s)	Measured response time, no rejuvenation (s)	Measured response time, with rejuvenation (s)
16		0.035	8.31650	17.429000
14		0.031	1.09810	0.180600
12		0.027	0.20702	0.104000
10		0.020	0.09187	0.047600
8		0.020	0.03739	0.041300
6		0.020	0.03043	0.032630
4		0.020	0.02619	0.027260
2		0.020	0.02167	0.024364

TABLE IV: Number of outliers, average, and median response times for no rejuvenation experiments

Experiment	Number of outliers	Average (ms)	Median (ms)
-2-30-2XL	9	1137.381	21.678
-2-30-4XL	30	1856.283	26.198
-2-30-6XL	69	548.743	30.437
-2-30-8XL	88	1268.839	37.391
-2-30-10XL	83	10975.401	91.879
-2-30-2L	19	239.694	30.320
-2-30-4L	29	929.630	31.159
-2-30-6L	57	240.331	34.197
-2-30-8L	91	6345.441	38.812
-2-30-10L	28	30193.215	56.430
-2-30-4XL12L	31	14595.671	8316.595
-2-30-6XL8L	89	6324.139	1098.101
-2-30-8XL4L	96	7239.404	207.026

C. Modeling and Measurement Results

We summarize the key results from the models and measurement campaigns presented. Section VII-B describes the insights obtained from the normalized distance metric between the observed and baseline performance. Specifically, Figures 18 and 19 illustrate the normalized distance plots for the first and second measurement campaigns, respectively. We presented selected plots in Figures 12–17 for specific hosts. Our

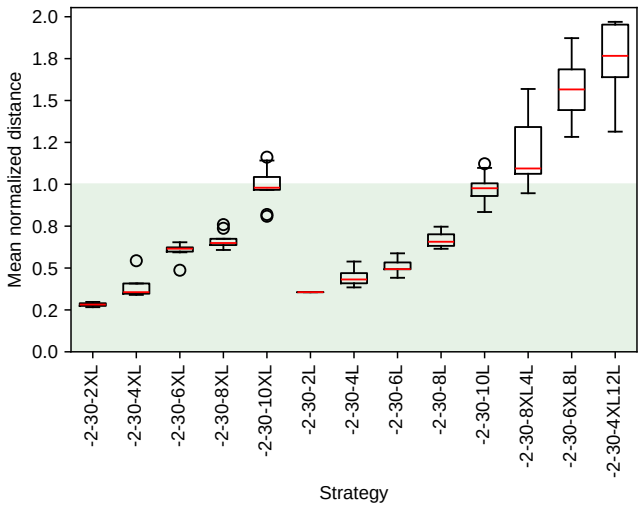


Fig. 19: Arithmetic mean of the normalized distance of the second campaign, **without rejuvenation**, for large and extra-large VMs showing normalized distance degradation as a function of the number of VMs.

replication package[2] contains all the collected empirical data. A summary of the key takeaways is presented in Section VII-D.

Insights From Metrics of Interest: The metric of interest used in this paper is the normalized distance from the scalability requirement as described in Equation 1 below, where the scalability requirement, and measured average response times are given by R and T, respectively:

$$ND = 2\ T/(T + R): \tag{1}$$

Figure 18 and Figure 19 presents the average normalized distance over all runs, for each strategy and granularity level for the first measurement campaign and the selected data from the second measurement campaign without rejuvenation. The x-axis of Figure 18 shows each strategy as described in Table II, while the x-axis for Figure 19 shows each data point in Table III.

In the leftmost side of Figure 18, corresponding to Strategy -1, we consider the case with extra-large hosts as core machines, and large hosts as backup. Most of the test runs in Strategy -1 showed an average normalized distance larger than 1. For strategies 0–4, we consider the case with large hosts as core machines and extra-large hosts as backup. We can see from Figure 18 that for the immediate rejuvenation strategy without pause or bulk, Strategy 0(i), the normalized distance is greater than 1 for all test runs.

When rejuvenation granularity and time of rejuvenation were added, the best performance was achieved for the Docker components granularity for strategies 0, 3, and 4. For strategies 1 and 2, Docker component granularity showed good performance but less improvement than in cases 0, 3, and 4.

D. Key Takeaways

In this subsection, we present the key takeaways from the application of the methodology presented in this paper to the large heterogeneous VM cluster.

1) Avoiding competition between rejuvenation and workload execution: Among the different approaches adopted to reduce the competition for resources between rejuvenation and workload execution, we considered bulk rejuvenation and deferral of workload execution while rejuvenation is taking place. Our experiments indicate that the latter is the most effective. Indeed, we observed that rejuvenation overhead was a significant differential in the overall performance. The best strategy, i.e., Strategy 3, implemented a workload pause during rejuvenation events. Such pause is key to avoiding competition for resources between rejuvenation and the execution of the workload.

2) Rejuvenation granularity (the finer, the better): Recall that we have three rejuvenation granularities: (i) coarse, corresponding to full installation, (dp) corresponding to Docker platform rejuvenation, and (dc) corresponding to Docker container rejuvenation. Note that (dp) is an intermediate level of granularity, between (i) and (dc). For the particular case of Strategy 3, which was the best-performing experiment, we observe that (dc) yields the best results. This is in agreement with the fine level of granularity being key to reducing rejuvenation overhead.

3) Contention for physical host resources: as discussed in Section VI, we observed that using large machines as core machines produced the best performance when compared to the case where extra-large machines are core and large machines are backup. This insight motivated us to derive optimal configurations using integer programming. The overall performance was dependent on the contention for physical host resources, as shown in Table III.

4) Insights from MDP: The results obtained from the MDP, and illustrated in Figures 6 and 7, show the best times to assign jobs to the large and extra-large VMs. Recall that the MDP is parameterized using the experimentally derived probability of failure, p_i, and therefore closely resembles the real behavior of the machines. The goal of the MDP is not that of *explaining* the internal dynamics that bring to the need for rejuvenation, but that of finding the optimal rejuvenation policy given the experimental measurements obtained by the first and second campaigns. Indeed, after experimentally deriving vectors p_i, for L and XL machines, we found, as expected, that p_i is an increasing function of the age i. We further discovered that the larger the contention for physical resources, the steepest the increase of p_i. In particular, when the total number of machines increased from 8 to 10, the MDP suggests that a switch from XL to L should be deployed earlier, i.e., for lower aging levels (see Figure 7), leading to lower utilization and more frequent rejuvenation.

VIII. Conclusion

We have extended our previous work in several ways. We have solved an integer programming problem to derive the optimal combination of VM sizes, and we have executed a second measurement campaign based on the solution of the integer programming problem. We have uncovered a second level of software aging due to contention for shared resources.

[2]Publicly available at https://zenodo.org/doi/10.5281/zenodo.10658060.

We have also extended the MDP model to take into account the Performance Dependency between VMs.

We have presented models, measurements, and metrics that can be used to support software rejuvenation and load-balancing actions to achieve resiliency in a large heterogeneous network environment. This large heterogeneous network environment uses Docker and an open-source benchmark that employs a complex media review system. This ecosystem contained two layers of aging: (1) a memory leak at the VM layer, which required the application of software rejuvenation and load balancing to achieve resiliency, (2) aging related to the contention for physical host resources, which impacted the optimal allocation of VM sizes and the choice of rejuvenation strategies. We have implemented two modeling approaches: MDP and Mixed Integer Programming. The first approach was used to derive optimal scheduling for a pair of VM nodes, while the second approach was used to derive the optimal mix of VM sizes to build a VM network.

To evaluate the effectiveness of software rejuvenation and load-balancing techniques, we have also implemented two measurement campaigns. In the first campaign, we evaluated six rejuvenation strategy approaches using three different levels of rejuvenation granularity. We applied the normalized distance of the response time to the derived scalability requirement to assess the VM network resiliency. We have found that software rejuvenation and load balancing are able to achieve resiliency in this ecosystem. However, care must be employed to assess the number and size of VMs to use, the software rejuvenation overhead, and the granularity level, when implementing rejuvenation and load-balancing strategies in these state-of-the-art environments. Our empirical results show a significant difference in the average normalized distance for the eighteen experiments. These findings motivated us to pose an Integer Programming Problem to compute the optimal mix of VM sizes. The optimal solution to this problem for a varying number of VMs was implemented in the second measurement campaign.

We introduced an MDP model to derive the optimal scheduling policy for VM pairs. The MDP optimal policy was evaluated based on the measured p_i, i.e., the probability of forced rejuvenation after the i-th experiment in two cases: (1) assuming independence between VMs, and (2) taking into account the dependency between VMs. In the first case, the optimal policy was to assign the extra-large machines to execute the experiment and use large machines as backup. In the second case, the optimal policy was heterogeneous (Figure 7).

The practical implications of our results are as follows.

These models and measurement results demonstrate that software aging monitoring and rejuvenation can be used effectively to achieve resiliency in large state-of-the-art heterogeneous VM networks deployed to private clouds. These results also show that to be successful, rejuvenation policies need to be carefully engineered to account for the physical environment parameters, such as physical memory, networking, and operational conditions.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable feedback. This work has been partially funded by the MUR-PRIN project 20228FT78M DREAM, MUR Department of Excellence 2023 - 2027 for GSSI, and PNRR ECS00000041 VITALITY.

References

[1] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Twenty-fifth international symposium on fault-tolerant computing.*, pp. 381–390, IEEE, 1995.

[2] J. F. Gonçalves and M. G. Resende, "A biased random key genetic algorithm for 2d and 3d bin packing problems," *International Journal of Production Economics*, vol. 145, no. 2, pp. 500–510, 2013.

[3] A. Avritzer, A. Janes, A. Marin, A. van Hoorn, M. Camilli, C. Trubiani, and D. S. Menasché, "Assessment of aging and rejuvenation for resiliency in heterogeneous network clusters," in *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023 - Workshops, Florence, Italy, October 9-12, 2023*, pp. 198–205, IEEE, 2023.

[4] OMG, "Business Process Model and Notation (BPMN), Version 2.0," January 2011.

[5] T. Dohi, K. S. Trivedi, and A. Avritzer, *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*. World scientific, 2020.

[6] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 1, pp. 1–34, 2014.

[7] K. Jia, X. Yu, C. Zhang, W. Hu, D. Zhao, and J. Xiang, "Software aging prediction for cloud services using a gate recurrent unit neural network model based on time series decomposition," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, pp. 580–593, jul 2023.

[8] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," in *Sixth International Symposium on Software Reliability Engineering*, pp. 180–187, IEEE Computer Society, oct 1995.

[9] G. Ning, J. Zhao, Y. Lou, J. Alonso, R. Matias, K. S. Trivedi, B.-B. Yin, and K.-Y. Cai, "Optimization of Two-Granularity Software Rejuvenation Policy Based on the Markov Regenerative Process," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1630–1646, 2016.

[10] J. Bai, X. Chang, G. Ning, Z. Zhang, and K. S. Trivedi, "Service availability analysis in a virtualized system: A markov regenerative model approach," *IEEE Transactions on Cloud Computing*, vol. 10, pp. 2118–2130, jul 2022.

[11] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability," *Empirical Software Engineering*, vol. 2, pp. 59–77, 1997.

[12] J. Zheng, H. Okamura, L. Li, and T. Dohi, "A comprehensive evaluation of software rejuvenation policies for transaction systems with markovian arrivals," *IEEE Trans. on Reliability*, vol. 66, no. 4, pp. 1157–1177, 2017.

[13] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Ninth International Symposium on Software Reliability Engineering*, pp. 283–292, IEEE, 1998.

[14] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in *10th International Symposium on Software Reliability Engineering*, pp. 84–93, IEEE, 1999.

[15] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, 2005.

[16] H. Sukhwani, R. Matias, K. S. Trivedi, and A. Rindos, "Monitoring and mitigating software aging on ibm cloud controller system," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 266–272, IEEE, 2017.

[17] S. S. Chouhan and S. S. Rathore, "Generative adversarial networks-based imbalance learning in software aging-related bug prediction," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 626–642, 2021.

[18] Z. Liu, Y. Zheng, X. Du, Z. Hu, W. Ding, Y. Miao, and Z. Zheng, "Taxonomy of aging-related bugs in deep learning libraries," in *33rd International Symposium on Software Reliability Engineering*, pp. 423–434, IEEE, 2022.

[19] G. Ning, K. S. Trivedi, H. Hu, and K.-Y. Cai, "Multi-granularity software rejuvenation policy based on continuous time markov chain," in *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*, pp. 32–37, IEEE, 2011.