Computing (2024) 106:2737–2783
https://doi.org/10.1007/s00607-024-01305-x

REGULAR PAPER

Energy-aware dynamic response and efficient consolidation strategies for disaster survivability of cloud microservices architecture

Iure Fé[1] · Tuan Anh Nguyen[2] · Mario Di Mauro[3] · Fabio Postiglione[3] · Alex Ramos[4] · André Soares[5] · Eunmi Choi[6] · Dugki Min[7] · Jae Woo Lee[8] · Francisco Airton Silva[1]

Received: 7 December 2023 / Accepted: 5 June 2024 / Published online: 17 June 2024

Abstract

Computer system resilience refers to the ability of a computer system to continue functioning even in the face of unexpected events or disruptions. These disruptions can be caused by a variety of factors, such as hardware failures, software glitches, cyber attacks, or even natural disasters. Modern computational environments need applications that can recover quickly from major disruptions while also being environmentally sustainable. Balancing system resilience with energy efficiency is challenging, as efforts to improve one can harm the other. This paper presents a method to enhance disaster survivability in microservice architectures, particularly those using Kubernetes in cloud-based environments, focusing on optimizing electrical energy use. Aiming to save energy, our work adopt the consolidation strategy that means grouping multiple microservices on a single host. Our aproach uses a widely adopted analytical model, the Generalized Stochastic Petri Net (GSPN). GSPN are a powerful modeling technique that is widely used in various fields, including engineering, computer science, and operations research. One of the primary advantages of GSPN is its ability to model complex systems with a high degree of accuracy. Additionally, GSPN allows for the modeling of both logical and stochastic behavior, making it ideal for systems that involve a combination of both. Our GSPN models compute a number of metrics such as: recovery time, system availability, reliability, Mean Time to Failure, and the configuration of cloud-based microservices. We compared our approach against others focusing on survivability or efficiency. Our approach aligns with Recovery Time Objectives during sudden disasters and offers the fastest recovery, requiring 9% less warning time to fully recover in cases of disaster with alert when compared to strategies with similar electrical consumption. It also saves about 27% energy compared to low consolidation strategies and 5% against high consolidation under static conditions.

Keywords Availability · Microservices · Petri nets · Reliability · Survivabiliry

Extended author information available on the last page of the article

Mathematics Subject Classification 68U01 · 68M14 · 68M15

1 Introduction

The advent of container-based microservices architecture has garnered considerable interest across various corporate sectors. This paradigm, extensively embracing the containerization concept, has emerged as a preferred modality for service delivery within cloud ecosystems [1, 2]. Cloud-native applications structured around microservices offer multiple advantages, primarily due to their modular architecture. This modularity not only streamlines development processes but also facilitates the distribution of these services across nodes within multiple cloud data centers [2]. Orchestrators, notably Kubernetes, play a pivotal role in managing these distributed microservices, maintaining application operations as delineated by system administrators [3]. The strategic dispersion of microservices enables replication of application components to fulfill dependability and performance criteria. Moreover, deploying these components across various data centers enhances application availability by positioning services in proximity to users [2, 4].

A significant benefit of microservice-based architectures is their potential to curtail energy consumption. The global energy consumption attributed to information and communication technology activities stands at approximately 2%, with projections indicating a surge to over 14% by 2040 barring any regulatory interventions [5–7]. The utilization of containers offers an avenue for energy reduction through the concept of consolidation. Containers consolidation is a technique used in data centers to optimize resource utilization. By consolidating multiple containers onto a single host, data centers can reduce hardware and operational costs. Container orchestration tools like Kubernetes and Docker Swarm make it easy to manage containerized workloads and achieve efficient resource allocation. High consolidation means that the containers are more centralized in one host. Low consolidation means that the containers are more distributed in multiple hosts. This strategy, prevalent in cloud architecture, optimizes data center resources and reduces energy usage by operating fewer physical nodes than the total number of services [8].

However, consolidation can have implications for the fault tolerance and dependability of systems [9]. The availability of multiple servers for instant microservice instantiation decreases disaster recovery time, albeit introducing additional potential failure points in the infrastructure. Disasters may stem from human-induced events like cyber-attacks, terrorism, or sabotage, as well as from natural occurrences such as floods, hurricanes, or fires [10]. Disaster recovery (DR) solutions are imperative for ensuring the continuity of organizational operations amidst such events. Recovery Time (RT) is defined as the duration necessary for the system to resume minimum service levels following a disruption [10–12]. Survivability, in this context, pertains to the system's ability to maintain or swiftly regain operational status within a predefined time frame following an adverse event. Assessing the effectiveness of a system's survivability mechanisms depends on the analysis of this period [12, 13]. Dependability, meanwhile, denotes the system's capacity to deliver a reliable service and encompasses attributes such as availability and reliability. Availability refers to

the system's readiness for service delivery, while reliability measures the continuity of service provision [14].

Nonetheless, many survivability strategies rely on resource-intensive methods, often employing component replication without considering resource constraints [15]. It is essential to quantitatively evaluate the impact of these strategies and the redundancy of components. Insufficient redundancy may be inadequate for addressing disasters like the unavailability of data centers, whereas excessive redundancy might not yield further improvements, leading instead to unnecessary electrical consumption and costs [16, 17]. Another limitation is the singular focus on one system layer, neglecting the interdependent effects of physical and software components [15]. Moreover, disaster recovery solutions should be environmentally considerate in terms of energy usage [15, 18]. Addressing disaster recovery for large applications while considering electrical costs introduces an additional layer of complexity. Achieving a balance between disaster recovery and electrical consumption necessitates managing a broader array of application and infrastructure parameters. This equilibrium must account for the peculiarities of microservices architecture, the behavior of container orchestrators, and resource relocation to maintain consolidation [9]. Resilient computing infrastructures are designed to withstand and recover from unexpected disruptions, such as hardware failures, software bugs, or cyber attacks. Resilience is achieved through redundancy, fault tolerance, and continuous monitoring. Cloud providers like AWS and Azure offer resilient infrastructure services that ensure high availability and disaster recovery. Achieving resilient infrastructures also entails testing various scenarios of sudden component absence, considering different infrastructure and application configurations [13]. Due to the risk of replicating disaster impacts on the application, it is often impossible to observe the behavior of disaster recovery and electrical costs in production stages. Conducting survivability tests in a real environment can be prohibitively expensive in terms of implementation time and electrical consumption. Moreover, developing strategies for specific architectures may yield results that are not generalizable, leading to either under-provisioning or over-provisioning. The work [19] presents a comprehensive availability model for disaster-tolerant data centers using stochastic reward nets, focusing on high-availability configurations, fault and disaster tolerance techniques, subsystem dependencies, and unexpected network failures. It emphasizes the importance of incorporating disaster and fault tolerance into geographically distributed data centers for enhancing cloud-based business availability. The model's analysis provides insights for designing disaster-tolerant systems, balancing availability, downtime cost, and infrastructure construction costs.

Aiming to tackle the problem of survivability optimization, our approach uses a widely adopted analytical model, the Generalized Stochastic Petri Net (GSPN). GSPN are a modeling technique that is widely used in various fields, including engineering, computer science, and operations research. GSPN is an extension of the popular Petri Net modeling technique. Petri Nets are a graphical modeling tool that is used to model systems that involve parallelism, concurrency, synchronization, and resource sharing. GSPN extends the Petri Net modeling technique by introducing stochastic behavior. Stochasticity refers to the randomness that is inherent in many real-world systems. One of the primary advantages of GSPN

is its ability to model complex systems with a high degree of accuracy. GSPN allows for the modeling of both logical and stochastic behavior, making it ideal for systems that involve a combination of both. For example, GSPN can be used to model computer networks, manufacturing systems, transportation systems, and healthcare systems, among others. GSPN also provides a range of analytical tools that can be used to analyze system performance. These tools include steady-state analysis, transient analysis, and sensitivity analysis. Steady-state analysis is used to determine the long-term behavior of a system. Transient analysis is used to analyze the behavior of a system during a transition period, such as when a system is starting up or shutting down [20–22].

This work presents a recovery strategy called Energy-aware Dynamic Response and Efficient Consolidation Strategy (eDRECs) aimed at reducing electrical consumption while adhering to the Recovery Time Objective (RTO) and enhancing the dependability of microservice-based systems in multiple cloud data centers, orchestrated by Kubernetes. GSPN models are appropriate to evaluate recovery strategies, since real system is not needed previously. Thus, GSPNs by simulation, were employed to evaluate three distinct recovery strategies (two from third parties and our proposal). Our proposed strategy involves appropriately sizing the infrastructure based on the desired RTO in the event of a partial infrastructure disaster. The GSPN models consider both physical and software component characteristics and configuration parameters to optimize architectures for desired levels of RT. Moreover, these GSPN models facilitate the examination of the strategies' impact on other dependability metrics and stationary electrical consumption. This analysis ensures that the disaster survivability strategy does not negatively affect the system's availability, reliability, and expected electrical consumption under typical conditions. Thus, this approach offers a novel strategy for rapid recovery, while enabling microservices-based system architects to optimize their infrastructure for lower environmental impact and achieve desired levels of availability, reliability, and RT.

The primary contributions of this study are as follows:

- This approach implements a dynamic response and efficient consolidation strategy that takes energy consumption into account to reduce electrical usage while achieving the desired RTO in cloud microservices infrastructures. Compared to other consolidation strategies, this method has demonstrated improved availability, reliability, and average failure time metrics;
- We present GSPN models that estimate electrical consumption while offering system managers a detailed understanding of the impacts of different configurations on the architecture. These models are capable of modeling survival and dependability, as well as obtaining stationary and transient values;
- The proposed approach is applicable for analyzing system response in two disaster scenarios: unforeseen events (such as cyber-attacks or configuration errors) and predictable disasters (like climate-related events that allow for preemptive action).

This study yields the following findings and implications:

- Using high-consolidation strategies that include additional redundant pods is more energy efficient than strategies that rely on redundant nodes to achieve RTOs. Administrators can considerably reduce electrical consumption by focusing on strategies with additional Pods for quick recovery, as consuming additional nodes brings a more significant increase in electrical consumption than adding pods.
- Strategies based on high consolidation with additional spare Pods exhibit superior reliability and Mean Time to Failure (MTTF) compared to strategies utilizing spare nodes. MTTF stands for Mean Time To Failure and is a measure of reliability used to estimate the expected time a system or component will operate before experiencing a failure [23];
- Strategies with faster instantiation capabilities display improved recovery times as the warning period increases.

The organization of the paper is as follows: Sect. 2 discusses related work. Section 3 describes the system architecture. Section 5 details the developed models and their metrics. Section 6 introduces the use cases. Finally, Sect. 7 offers conclusions and outlines directions for future research.

2 Related work

This section describes research in disaster recovery time reduction and energy consumption minimization within cloud system planning. Andrade et al. [16] introduced a Stochastic Petri Net (SPN) model to evaluate disaster recovery solutions in cloud architectures, focusing on virtual machine-based setups. This model aids in selecting disaster recovery solutions by quantitatively analyzing response time, throughput, cost, and availability metrics, underlining the importance of such assessments in maintaining business continuity during sudden adverse events.

Using CloudSim, Nong et al. [24] proposed a survivability framework for efficient allocation of virtual machines (VMs). This framework addresses rack failures by migrating VMs to operational racks, wherein the VMs are converted into images and reinitiated as containers. Ramasamy et al. [1] developed a redundancy-enhancing method for Kubernetes clusters to bolster high availability amidst service interruptions. This method employs a reserve cluster to facilitate service exchange based on application availability, negating the need for user intervention in cluster restoration post-failure.

In contexts where disasters are predictable, Ma et al. [25] presented an Integer Linear Programming (ILP) problem and an efficient heuristic for pre-disaster service reallocation. This approach aims to maximize protected requests by relocating services to unaffected data centers, with the extent of relocation dependent on the available warning time. Ayoub et al. [26] introduced an ILP model and heuristic for VM migration from potential disaster zones to safe locations, demonstrating reduced downtime and identifying critical timelines to avoid service interruptions.

Colman et al. [27] employed an ILP-based method to assess the impact of both natural and human-induced disasters, like mass cloud destruction attacks, on

distributed data centers. Their approach, focusing on system backup sharing, significantly mitigated disconnection risks due to disasters. Sun et al. [28] proposed a quantitative SPN-based method to evaluate the survivability of distributed systems, including metrics like reliability, maintainability, and availability. Longo et al. [13] developed SPN models to quantify resilience in large-scale systems, particularly focusing on transient behaviors under structural or load changes, such as those expected in disasters.

Trivedi et al. [12] presented a model-based approach utilizing Stochastic Reward Net (SRN) and Continuous Time Markov Chain (CTMC) to assess system recovery capabilities within specified intervals post-disaster. This approach effectively captured performance metrics' behavior in large-scale systems during and after disaster scenarios.

In the realm of resource usage reduction, Hamadah [29] qualitatively discussed rapid recovery possibilities using tools provided by cloud services, with a focus on electrical consumption. Isa et al. [30] proposed an energy-efficient and resilient fog computing infrastructure for health monitoring applications, modeled using Mixed Integer Linear Programming (MILP). This model took into account the energy consumption of both idle and active components, demonstrating the influence of distributed data center deployment on the energy footprint of health monitoring systems. Gandhi et al. [31] introduced a strategy for energy conservation in data centers, emphasizing the need to maintain only the necessary active servers, and exploring the implications of various server deactivation policies (sleep, stand-by, off) on activation time and energy consumption.

There are some papers related to this subject with Petri nets. Pinheiro et al. [32] proposes the modeling of these infrastructures and their auto-scaling mechanisms in a private cloud using stochastic Petri nets (SPNs), the non-dominated sorting genetic algorithm II (NSGA-II), one of the most popular evolutionary algorithms for multiobjective optimization (MOO), and random forest regression (RFR), an ensemble-learning-based method, to identify critical trade-offs between performance and resource consumption considering all deployed MSs. Soylu et al. [33] explores using Petri nets for modelling microservice-based systems. The proposed method utilises abstraction and composition techniques to model complex systems using 1-safe Petri nets. A case study on Banking as a Service (BaaS) demonstrates the suitability of Petri nets for modelling, analysis and verification of microservice-based systems. In our previous paper [34], we have presented a model-driven approach to evaluate the dependability and energy consumption of cloud-fog systems, utilizing Kubernetes, a container application orchestration platform. The developed model considers various determinants affecting system dependability, including hardware and software reliability, resource accessibility, and support personnel availability.

While the literature offers diverse strategies and planning tools for handling various disaster types and considerations related to dependability and energy consumption in data centers, there appears to be a gap in research specifically addressing DR focused on microservice environments in the cloud in relation to electrical costs. This work aims to bridge this gap by presenting a strategy that not only achieves the expected RT levels in both sudden and predictable disaster scenarios but also reduces the overall energy consumption of the system.

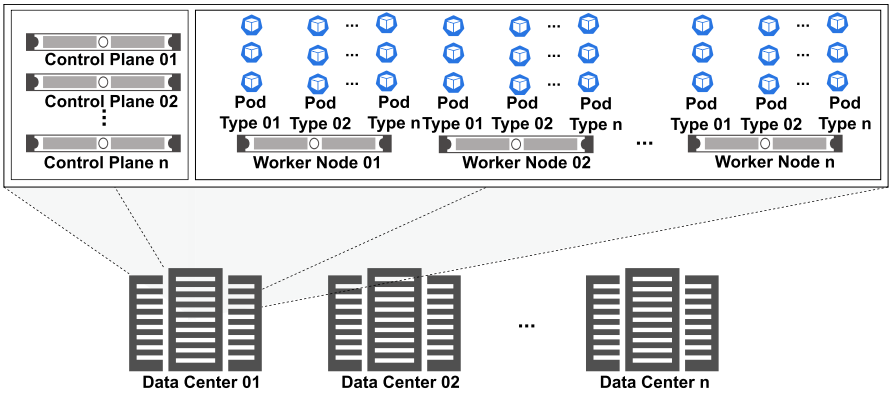


Fig. 1 Kubernetes cluster architecture distributed across n data centers

3 System architecture

This section presents the system architecture, composed of Kubernetes components and their respective functionalities. The sections provide concepts and strategies pertinent to high and low consolidation systems in disaster scenarios.

3.1 System components

The architecture of a standard Kubernetes cluster, as depicted in Fig. 1, typically spans across multiple data centers in a cloud environment. This infrastructure is composed of two primary node types: the Control Plane (CP) and Worker Nodes (WN).

Applications within this environment are housed in Pods. A Pod is an aggregation of one or more containers that encapsulate applications sharing common network and storage resources. In the context of a microservices architecture, numerous Pods exist, each configurable in varying quantities to accommodate the diverse demands of different application segments. An example of this flexibility is the replication of a Pod containing a particular application type, such as a frontend component, across various data centers. This replication strategy is instrumental in handling requests from a multitude of clients.

Moreover, the allocation of Pods of a specific type is not inherently restricted to any single data center node. This implies that, barring any specific hardware requirements or bespoke configurations, Pods can be dynamically distributed across any node within the data center network [3].

The Control Plane (CP) orchestrates and synchronizes the operations of Worker Nodes (WNs) and Pods within a Kubernetes architecture, adhering to predefined specifications regarding the deployment and scaling of Pods, as well as the allocation of WNs. This includes responsive measures to infrastructure failures [3]. The CP is composed of several critical processes:

- *kube-apiserver*: This serves as the gateway for Kubernetes API, facilitating the administrative interface for Kubernetes management.
- *etcd*: It functions as a repository, maintaining essential system information and state data.
- *Kube-scheduler*: Entrusted with determining the placement of newly instantiated Pods. It evaluates both configuration directives and dynamic factors to assign an appropriate Worker Node for the Pod, typically favoring nodes with ample resources in the absence of constraints.
- *kube-controller-manager*: A conglomerate of processes governing nodes, jobs, and various Kubernetes components.

Additionally, the CP's operational framework includes a computational entity, which may either be a bare metal server or a Virtual Machine (VM), executing these processes [3]. The WN segment comprises nodes that may be either dedicated hardware or a VM, incorporating the following processes:

- *Kubelet*: It oversees container management, receiving Pod specifications and ensuring their optimal functionality on the node.
- *kube-proxy*: This component administers network regulations on the nodes, facilitating both internal and external network communications within the cluster.
- *Container runtimes*: They are tasked with executing containers. Kubernetes extends support to a variety of container runtimes, including Docker, containerd, CRI-O, among others [3].

3.2 Previous survivability strategies

The distributed architecture, as depicted in Fig. 1, enables Pods to be strategically positioned across various data centers in disparate quantities. This spatial diversification affords the application enhanced resilience, allowing it to withstand potential disasters impacting one or multiple data centers [1, 9]. In the event of a data center failure, there is an inevitable decrement in the number of operational Pods. Subsequently, Kubernetes endeavors to compensate for this shortfall by initiating the instantiation of the requisite Pods in the remaining functional data centers. This process of Pod activation, encompassing both container and application initialization, is not instantaneous and is subject to temporal constraints [3]. The duration required for this recovery is contingent upon the extent of the disaster, potentially exceeding the acceptable RTO.

An additional architectural facet warranting consideration is the chosen consolidation methodology, which dictates the distribution and occupancy of nodes within the system. The Low Consolidation Strategy (LCS) includes several approaches identified in literature that prioritize the quick allocation of virtual resources, by keeping physical nodes active continuously [8, 9, 35, 36]. Under LCS, all servers remain operational and ready to accommodate new Pods. In the event of a data center failure, nodes in unaffected data centers are ready to quickly start the instantiation of Pods that fall below a predetermined minimum threshold. Figure 2a

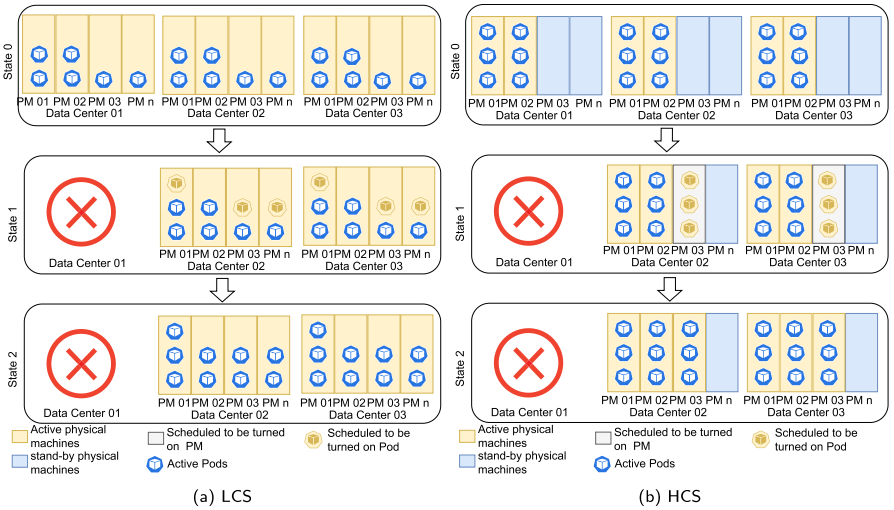


Fig. 2 Behavior of LCS and HCS consolidation strategies in case of disaster

graphically represents the LCS approach. Initially, in state 0, three Data Centers (DCs) each house six Pods. Following a disaster rendering DC 01 inoperative, state 01 illustrates the redistribution and commencement of the six Pods from DC 01 across the remaining DCs. In state 02, the reactivation of the six disrupted Pods is completed. The distributed initialization framework of the LCS strategy is expected to enable fast Pod instantiation. However, this approach may also lead to significant energy consumption as it requires keeping all nodes continuously connected and prepared for immediate Pod instantiation.

In contrast, High Consolidation Strategies (HCS) are characterized by their utilization of the minimal number of active physical nodes necessary for accommodating the required Pods [9, 30, 31]. HCS is capable of achieving the lowest electrical consumption by leveraging only essential resources. Nonetheless, in disaster scenarios, this strategy may necessitate additional server activation time to accommodate Pods displaced from the impacted data center. The activation duration of a node encompasses the time required for power-up, emergence from stand-by, or transition from sleep mode [31]. Furthermore, HCS faces challenges in simultaneously instantiating multiple Pods with a limited number of servers, potentially exacerbating the system's RT [9]. Figure 2b delineates the operational dynamics of this strategy. Initially, in state 0, Pods occupy only the requisite Worker Nodes (WNs) for their operation. State 1 depicts the post-disaster phase, illustrating the process to power up 2 WNs and reestablish 6 Pods displaced by the failure of data center 01. State 2 visualizes the resumption of node and Pod functionality in two data centers.

The system's RT and average electrical consumption are also influenced by other architectural configurations and hardware elements. Hardware selection impacts energy consumption, server connectivity duration, and containerized application performance [9, 15]. Additionally, hardware choices affect system availability and reliability, as indicated by the MTTF. These failures persist throughout the system

and in disaster scenarios, necessitating continuous monitoring of other system dependability metrics. Moreover, the capabilities of the physical nodes impact the system by defining the maximum number of Pods that can be simultaneously instantiated on each node.

Another prevalent strategy involves incorporating spare Pods into the system, adding more Pods than stipulated by the Service Level Agreement (SLA). This K-out-of-N (KooN) strategy is extensively utilized in various availability studies [37–39]. In disaster scenarios, the presence of additional Pods reduces the RT, although it incurs additional electrical costs. The use of redundant elements is a configuration measure that does not require changes to the architecture and can be applied to any of the strategies mentioned.

The effectiveness of survivability may be impacted by disaster forewarning. In the absence of advance warning, system recovery and service restarts may be required in alternate locations, using backups or redundant resources. In contrast, pre-emptive resource reallocation can help mitigate the impact of warned disasters. The effectiveness of this approach and the warning time duration can potentially ensure uninterrupted service delivery during the disaster. However, this preventive measure also faces challenges similar to those discussed earlier regarding the trade-off between recovery time and electrical consumption. In the aftermath of a disaster, there is often a trade-off between recovery time and electrical consumption. While quick recovery is crucial for minimizing damage and restoring critical services, it may require higher energy consumption, which could strain the already limited power supply. Therefore, finding an optimal balance between these two factors is essential to ensure a sustainable and efficient disaster response. This requires careful planning and decision-making, considering the availability of resources, infrastructure, and environmental factors. By leveraging advanced technologies and innovative solutions, it is possible to improve the resilience and responsiveness of disaster management while minimizing the energy footprint and reducing the impact on the environment.

4 Proposed strategy-energy-aware dynamic response and efficient consolidation strategy

This section presents the strategy proposed in this work. It was developed to achieve better recovery times and lower energy consumption when compared to traditional approaches. LCS offers enhanced recovery capacity by maintaining connected nodes ready for immediate Pod instantiation in the event of a disaster. However, this method incurs energy expenditure by keeping unneeded nodes active. On the other hand, HCS is energy-efficient by utilizing only necessary nodes, but faces increased RT during disasters due to the need for activating additional nodes and limited parallel instantiation capacity.

To navigate this trade-off, we introduce the eDRECS strategy that focuses on rapid disaster recovery while also considering energy consumption. Our approach integrates the high responsiveness of a non-consolidated environment with the energy efficiency of a consolidated environment. This strategy is predicated on a

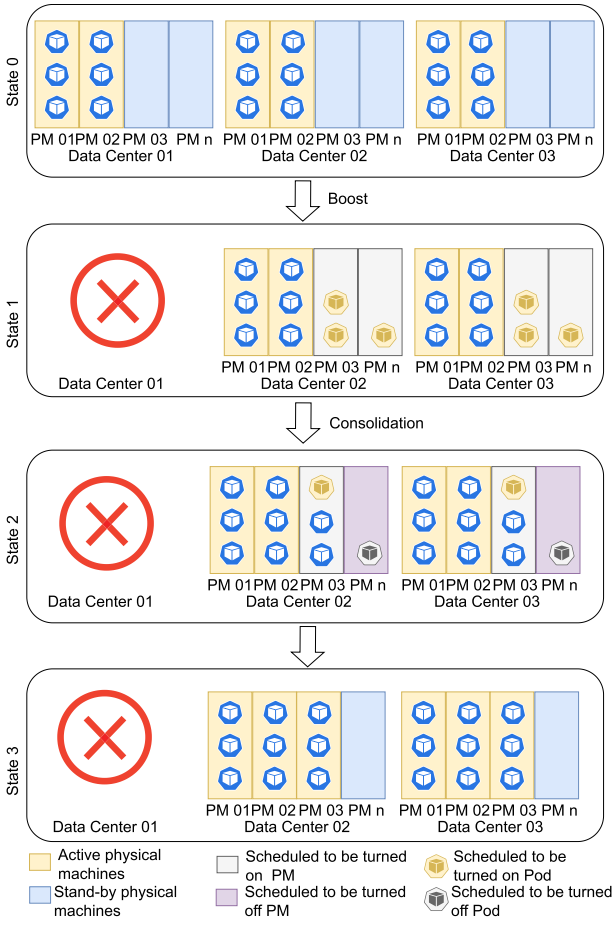


Fig. 3 Behavior of the eDRECS consolidation strategy in case of disaster

high consolidation model with modified behavior during disasters, complemented by the deployment of redundant Pods to expedite recovery and minimize electrical consumption. The operational dynamics of this strategy are illustrated in Fig. 3. State 0 presents the system in its standard configuration, akin to the high consolidation state. State 1 depicts the system's response to failure, initiating the activation of all feasible nodes to enhance the parallelization of Pod creation. In state 1, the strategy makes the system behave like the LCS, offering a greater number of parallel nodes for creating Pods. This additional WN capacity aids in reducing RT. However, this intensified activity is temporary and should be limited to addressing immediate demand. State 2 involves consolidation, scheduling Pod creation in a consolidated manner, followed by the deactivation of surplus Pods and WNs, this is accomplished by relocating the required number of Pods to as few Nodes as possible. State

3 showcases the outcome of this consolidation, maintaining the minimum number of WNs required for the system's Pods.

However, in disaster scenarios, there will be a temporary increase in electrical consumption, as well as continuous power usage of spare Pods within the system. To ensure that the number of redundant Pods is neither too low to achieve the targeted RTO nor too high to exceed acceptable levels of electrical consumption, it is essential to carefully calibrate the number of redundant Pods for each specific situation. This configuration depends on various factors, including application and node activation durations, as well as their availability attributes, as previously discussed in Sect. 3.2.

Details of the specific elements of each element involved in the behavior of this strategy will be discussed in detail in the Sect. 5.3. It contains the model that addresses the time-dependent and configuration aspects of the architecture.

5 Stochastic models of survivability strategies

This section presents the GSPN models for the strategies discussed in Sect. 3. We begin by explaining the LCS model, including its operational dynamics and required variables. We then describe the modifications made to the LCS model to incorporate the High Consolidation Strategy (HCS). Next, we introduce the model for eDRECS. Finally, we describe the metrics used in these models to evaluate availability, reliability, resource utilization, and RT. These models can perform both stationary and transient systems analysis. Stationary analysis provides insights into failures that may arise from various elements of the architecture, such as hardware, software, and configuration. In contrast, transient analysis not only verifies reliability but also assesses the impact of disasters on the infrastructure.

5.1 Model of LCS

Figure 4 exhibits the Generalized Stochastic Petri Net (GSPN) model characterized by LCS. This model represents a scenario where all nodes designated for application deployment across various data centers are active. The transitions of this model are comprehensively detailed in Table 1. Additionally, Tables 2 and 3 provide insights into the guard expressions implemented in the model. The guard expressions are conditions associated with transitions, and evaluating them as true allows the associated transition to fire. Table 2 details guard expressions that can be changed depending on the objective of the system analysis, which may be to identify stationary, transient, or reliability behavior. Table 3 shows the guard expressions that determine whether to add or remove pods based on the quantity and state of WN available in the system. The column with Firing Semantic, indicates the possibility of each transition being of two types: infinite-server (IS) our single-server (SS) semantics. IS means that the tokens can be processed in parallel. SS means that the tokens can be processed sequentially.

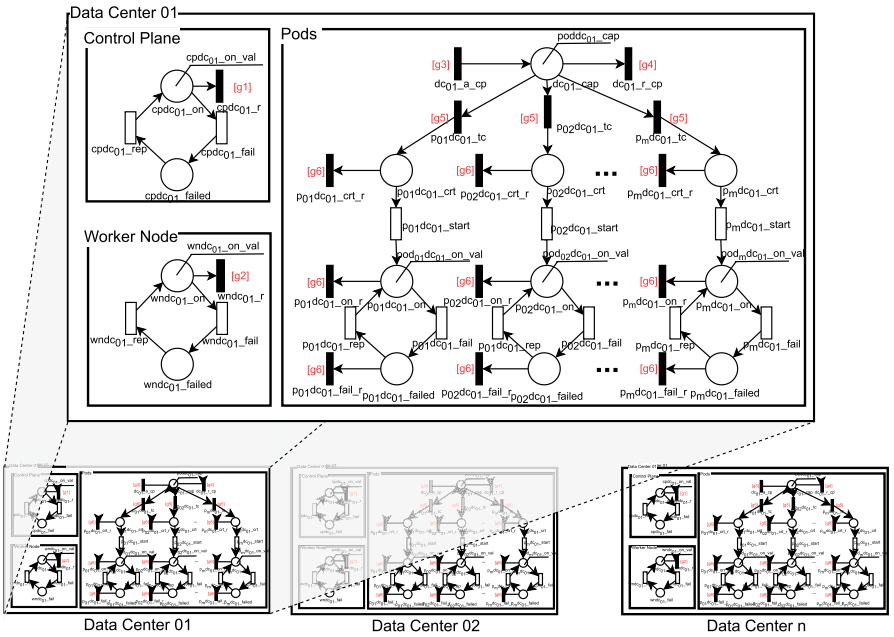


Fig. 4 LCS consolidation strategy—GSPN Model

Table 1 LCS—additional information about model transitions

Transiton	Description	Firing semantic	Weight	Priority
$wndc_n\_rep$	Repair of data center worker node n	SS	–	1
$wndc_n\_fail$	Fail of data center worker node n	IS	–	1
$wndc_n\_r$	Remove data center worker nodes	–	1.0	3
$cpdc_n\_fail$	Repair of data center control plane n	SS	–	1
$cpdc_n\_rep$	Fail of data center control plane n	–	1.0	3
$cpdc_n\_r$	Remove data center control planes	SS	–	1
$dc_n\_a\_cp$	Capacity add in data center n	–	1.0	1
$dc_n\_r\_cp$	Capacity remove in data center n	–	Eq. (12)	2
$p_m dc_n\_tc$	Begin creation of pod type m in data center n	–	1.0	1
$p_m dc_n\_start$	Time to creation of pod type m in data center n	IS	–	1
$p_m dc_n\_fail$	Fail a pod type m in data center n	IS	–	1
$p_m dc_n\_rep$	Repair a pod type m in data center n	SS	–	1
$p_m dc_n\_crt\_r$	Remove creating pods type m in data center n	–	Eq. (13)	2
$p_m dc_n\_on\_r$	Remove healthy pods type m in data center n	–	Eq. (14)	2
$p_m dc_n\_fail\_r$	Remove failed pods type m in data center n	–	Eq. (15)	2
$dc_n\_disaster\_time$	Disaster event	SS	–	1

Table 2 LCS—guard expressions with usage dependent on model application

Model usage	Index	Guard expression	Description
Disaster analysis	[g1]	$\#dc_n\_disaster = 1$	Remove all data center n control planes if a disaster occurs
Disaster analysis	[g2]	$\#dc_n\_disaster = 1$	Remove all data center n worker nodes if a disaster occurs
Stationary metrics	[g1]	False	Transition never fires in this usage
Stationary metrics	[g2]	False	Transition never fires in this usage
System reliability	[g1]	Eq. (1)	Remove all control planes if the system are in a failed state
System reliability	[g2]	Eq. (1)	Remove all worker nodes if the system are in a failed state

Table 3 LCS—guard expressions

Index	Guard expression	Description
[g3]	Eq. (5)	Adds capacity in the data center when there are more active servers
[g4]	Eq. (7)	Remove capacity when has more worker nodes than total pods capacity
[g5]	Eq. (8)	Add Pods when has lower pods than than defined and exist capacity to instantiate pod
[g6]	Eq. (11)	Remove Pods when has more pods than worker node capacity or more pods than defined

The architecture of a multi-data center microservices system is conceptualized as a conglomeration of analogous blocks for each data center. To enhance comprehensibility, the data center block is subdivided into three sub-blocks, the division allows delimiting the representation of the elements in a didactic way. The CP block is composed of the places $cpdc_n\_on$ (Control Plane from data center n on) and $cpdc_n\_failed$,as can be seen in Fig. 4. The tokens within $cpdc_n\_on$ signify the number of operational control plane nodes in the n^{th} functional data center. The initial quantity of available Control Planes in the data center is denoted by the variable $cpdc_n\_on\_val$, while the tokens in $cpdc_n\_failed$ represent the count of non-functional elements within the control plane.

The transition $cpdc_n\_fail$ within the model symbolizes the failure occurrence of a Control Plane component, governed by the failure time distribution specific to that component. Conversely, the transition $cpdc_n\_rep$ is designed to represent the repair duration for components of the Control Plane that have encountered failures. Consistent with prevailing academic literature [16, 40–42], this study presupposes that both failure and repair intervals follow an exponential distribution. However, it is noteworthy that alternate distributions may be employed within Generalized Stochastic Petri Nets (GSPNs), provided that the system operates under simulation, or the transition is substituted by a phase-type distribution as elucidated in [43].

Additionally, the control plane block incorporates the immediate transition $cpdc_n\_r$, which is conditional upon the guard expression [g1]. The specific nature

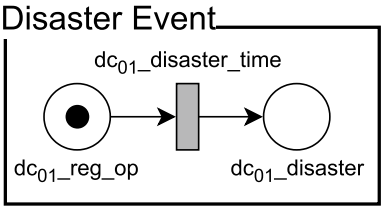


Fig. 5 Transition that ruling the disaster event

of [g1] is contingent upon the metric sought from the model. Table 2 outlines the potential guard expressions. In scenarios aimed at discerning the system's stationary behavior, it becomes necessary to deactivate this transition. It may even be removed from the model altogether, particularly when the focus is exclusively on the failures and repairs of architectural components. In contrast, if the analysis is geared towards ascertaining the system's reliability, the relevant computation is formalized in Eq. (1):

$$\left(\sum_{n\in N} \#cpdc_n\_on < sla\_cp\right) \vee \left(\sum_{n\in T} \#p_1dc_n\_on < sla\_pod_1\right) \vee \left(\sum_{n\in T} \#p_2dc_n\_on < sla\_pod_2\right) \vee ... \left(\sum_{n\in T} \#p_mdc_n\_on < sla\_pod_m\right) \tag{1}$$

where: T = {n | n are the data centers eligible to allocate Pod m}, N = {n | n are the available data centers}.

Equation (1) is formulated to register a system failure in instances of Service Level Agreement (SLA) violations, thereby enabling an assessment of the probability of the system's operational status at any given time. This assessment is done by checking the number of tokens with the SLA associated with the Pods and control plane. Checking the number of tokens is represented by the # character before the place name. The application of Eq. (1) further facilitates the determination of the system's MTTF, aligning with methodologies delineated in existing literature [44]. When the research objective is to analyze the RT or examine the transient behavior of system resources following a disaster, a focused disaster analysis approach is required. This approach utilizes the expression $\#dc_n\_disaster = 1$, triggering in the event of a disaster impacting data center n.

The potential occurrence of a disaster in data center n is contingent upon the presence of a corresponding disaster sub-block, as illustrated in Fig. 5. This sub-block comprises two places: $dc_n\_\_reg\_op$ and $dc_n\_disaster$. Initially, the token in $dc_n\_reg\_op$ indicates the absence of a disaster in data center n, while the presence of a token in $dc_n\_disaster$ signifies a disaster occurrence. The deterministic transition $dc_n\_disaster\_time$ activates at a predetermined moment, signaling a disaster. To assess the disaster's impact and evaluate the efficacy of the selected recovery strategy, transient metrics should be monitored at this juncture. For effective disaster scenario analysis, a transient simulation must be executed using the modeling tool, with pertinent metrics collected during the defined time interval.

It is noteworthy that the model allows for the incorporation of multiple disaster event blocks across various data centers. The employment of several disaster blocks facilitates a comprehensive examination of the repercussions of catastrophic events on the simulated infrastructure. This approach is instrumental in analyzing the system's response and its capability to recuperate from such adversities.

The Worker Node block within the model encapsulates the aggregate of Worker Node entities. The active Worker Nodes are quantitatively represented by the tokens in $wndc_n\_on$, while the tokens in $wndc_n\_failed$ denote the number of failed nodes. System initialization incorporates the parameter $wndc_n\_on\_val$, which specifies the count of worker nodes. This figure crucially determines the feasible number of Pods that can be instantiated within the data center for system operations. The failure of an active Worker Node is modeled by the activation of the transition $wndc_n\_fail$, which reassigns a token from $cpdc_n\_on$ to $cpdc_n\_failed$. Conversely, the restoration of any non-operational node is accomplished through the transition $wndc_n\_rep$, transferring a token back from $cpdc_n\_failed$ to $cpdc_n\_on$.

Additionally, the Worker Node block incorporates the immediate transition $wndc_n\_r$. This transition possesses the capability to eliminate all worker nodes from the infrastructure, contingent upon the specific guard expression [g2] employed. The operational mechanics of $wndc_n\_r$ and the selection of [g2] are dependent on the model's intended application, as outlined by the designer. Consequently, the appropriate choice of guard expression [g2] is determined by the desired verification type, as delineated in Table 2.

The Pods block within the model symbolizes both the Pods and the data center's capacity for Pod instantiation. The instantiation capacity available to the system in a specific data center is indicated by the tokens in dc_n_cap. The initial capacity is assigned through the variable $poddc_n\_cap$. The foundational value of $poddc_n\_cap$ is computed using Eq. (2):

$$poddc_n\_cap = dc_n total\_cap - used\_cap\_dc_n, \tag{2}$$

Where:

$$dc_n total\_cap = ppn\_dc_n \times \#wndc_n\_on, \tag{3}$$

$$used\_cap\_dc_n = \sum_{m \in U} (\#p_m dc_n\_crt + \#p_m dc_n\_on + \#p_m dc_n\_failed), \tag{4}$$

and U = {m | m are the indices of the Pods eligible for data center n allocation}.

Equation (2) accounts for the net allocation capacity of Pods within a data center. This capacity is calculated by deducting the currently allocated Pods (as delineated in Eq. (4)) from the total Pod allocation capacity of the data center (specified in Eq. (3)). It is pertinent to note that the allocation capacity for Pods is contingent upon the product of the number of Pods per Worker Node (ppn_dc_n) and the number of active Worker Nodes ($\#wndc_n\_on$). In the context of this strategy, all Worker Nodes are presumed to be operational.

The modeling of this block must encompass the various types of Pods that can be instantiated within the data center. For each Pod type m that is allocable in

data center n, a corresponding set of places—$p_m dc_n\_on$, $p_m dc_n\_failed$, $p_m dc_n\_crt$—alongside the directly associated transitions, should be established and linked to the data center's capacity (denoted by the place dc_n_cap). The active Pods of type m in data center n are represented by the tokens in $p_m dc_n\_on$. The initial count of these tokens is determined by the variable $p_m dc_n\_on\_val$, which is set based on the initial system configuration. In the event of a Pod failure, the transition $p_m dc_n\_fail$ is triggered, transferring a token from $p_m dc_n\_on$ to $p_m dc_n\_failed$. Subsequent recovery and reactivation of the failed Pod are modeled by the $p_m dc_n\_rep$ transition, which depicts the repair duration.

Furthermore, the model reflects changes in the capabilities of Pods based on the system's state. The introduction of additional capacity for Pod instantiation in the data center is modeled by the activation of the $dc_n\_a\_cp$ transition. This process is governed by the guard expression [g3], which predicates capacity creation on the condition that the sum of the allocated Pods and the available capacity in the data center does not exceed the total capacity provided by the WNs. This relationship is mathematically expressed in Eq. (5):

$$inst\_cap\_dc_n < dc_n total\_cap, \tag{5}$$

where:

$$inst\_cap\_dc_n = used\_cap\_dc_n + \#dc_n\_cap. \tag{6}$$

Therefore, $dc_n\_a\_cp$ can fire when a Worker node is added or retrieved to the system. The removal of available capacity is given by firing the $dc_n\_r\_cp$ transition, enabled by the guard expression [g4] in Eq. (7):

$$inst\_cap\_dc_n > dc_n total\_cap, \tag{7}$$

[g4] enables whenever there is more capacity instantiated than the number of nodes supports. Therefore, it can fire when a Worker Node fails or when a disaster occurs.

Upon the occurrence of a failure, be it at the level of a Worker Node or encompassing an entire data center, Kubernetes orchestrates the instantiation of Pods on alternate nodes. This operational response is represented within the model by the activation of the immediate transition $p_m dc_n\_tc$. This transition is responsible for the allocation and subsequent creation of Pods on one of the available Worker Nodes within a data center deemed suitable for that specific Pod. The activation and regulation of the transition $p_m dc_n\_tc$ are meticulously controlled by the guard expression [g5]. This expression, integral to the model's functionality, operates based on the criteria set forth in Eq. (8). This equation is instrumental in determining the conditions that must be met for the transition to be triggered, thereby enabling an accurate simulation of Kubernetes' resilience mechanisms in response to infrastructure failures.

$$(alocated\_pod\_{\_m} < min\_pod_m) \wedge (crt\_pods\_dc_n < wndc_n\_on), \tag{8}$$

where:

$$alocated\_pod\__m = \sum_{n \in T} (\#p_m dc_n\_crt + \#p_m dc_n\_on + \#p_m dc_n\_failed), \tag{9}$$

and,

$$crt\_pods\_dc_n = \sum_{m \in U} \#p_m dc_n\_crt. \tag{10}$$

Equation (8) conducts an assessment to ascertain whether the current count of Pods of a specific type m is below the system's predefined threshold, and concurrently evaluates the feasibility of initiating new Pods within the active nodes of the data center. The execution of the transition $p_m dc_n\_tc$ results in the reassignment of a token from dc_n_cap to $p_m dc_n\_crt$, symbolically denoting the commencement of Pod creation. The duration of Pod instantiation is determined by the timed transition $p_m dc_n\_start$. The variable $p_m dc_n\_start$ represents a stochastic element, signifying the time required for the Pod and the encompassed microservice to transition to an operational state. It is important to note that the process of Pod instantiation also entails energy consumption, thereby rendering strategies that frequently relocate Pods less favorable due to increased energy demands. Following the activation of $p_m dc_n\_start$, a token is transferred from $p_m dc_n\_crt$ to $p_m dc_n\_on$, thereby indicating the Pod's transition to normal functionality.

Conversely, the model accommodates two scenarios that necessitate the removal of Pods from a data center: the presence of excess Pods or the removal of Worker Nodes. Excessive Pods of a particular type within the system may arise during the recovery phase of a data center. The evaluation of Pod numbers involves a comparison between the existing Pod count and the minimum number specified by the user, denoted as min_pod_m. The removal of Worker Nodes is triggered either due to individual node failures or the collapse of an entire data center. These events are modeled by the activation of transitions $p_m dc_n\_crt\_r$, $p_m dc_n\_on\_r$, and $p_m dc_n\_fail\_r$, which are governed by the guard expression [g6]. This expression, integral to the model's operational logic, is formulated in accordance with Eq. (11):

$$(alocated\_pod\__m > min\_pod_m) \vee (inst\_cap\_dc_n > dc_n total\_cap), \tag{11}$$

We can notice that the transitions $dc_n\_r\_cp$, $p_m dc_n\_crt\_r$, $p_m dc_n\_on\_r$, and $p_m dc_n\_fail\_r$ can compete for firing in situations where the data center has more pods than expected. To define who will have priority for the fire, we use the following Equations:

$$dc_n\_cap\_prob = \frac{\#dc_n\_cap}{used\_cap\_dc_n}, \tag{12}$$

$$remove\_crt\_prob\_p_m dc_n = \frac{\#p_m dc_n\_crt}{used\_cap\_dc_n}, \tag{13}$$

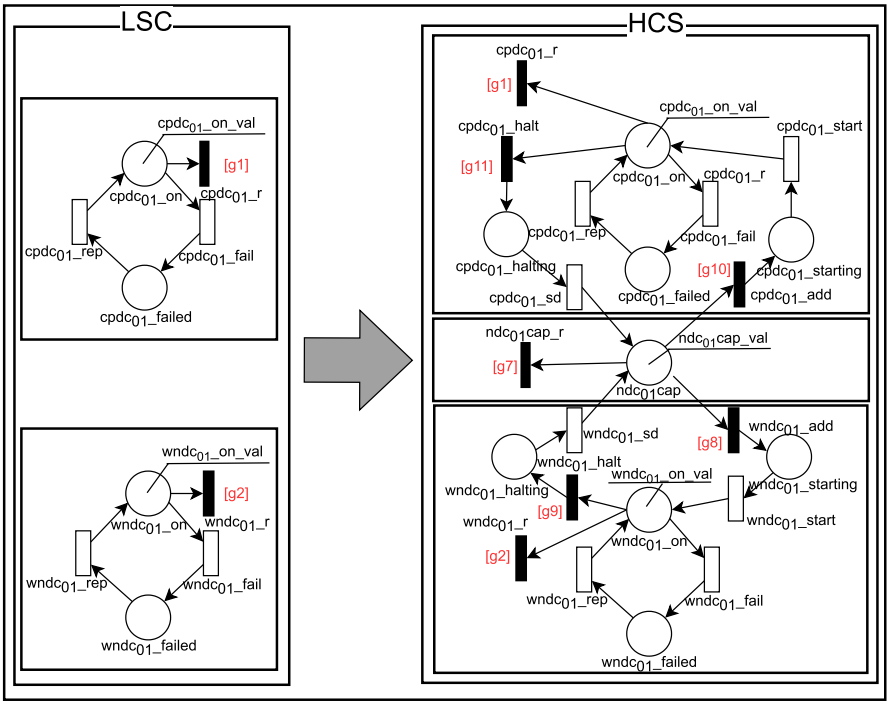


Fig. 6 Change applied to the control plane and worker nodes block of the LSC strategy to represent the HCS strategy

$$remove\_on\_prob\_p_m dc_n = \frac{\#p_m dc_n\_on}{used\_cap\_dc_n}, \tag{14}$$

$$remove\_fail\_prob\_p_m dc_n = \frac{\#p_m dc_n\_fail}{used\_cap\_dc_n}, \tag{15}$$

The allocation weights within the model play a pivotal role in determining from which state a Pod will be removed in the event of a node failure. Specifically, if a server experiences a failure and its utilized capacity is minimal, the model is more likely to decrease the Pod allocation capacity rather than directly removing active Pods from that data center.

5.2 Model of HCS

The development of the HCS model entailed modifications to the foundational model depicted in Fig. 4, particularly concerning its storage parameters. These alterations are illustrated in Fig. 6, with changes predominantly affecting the Control Plane and Worker Node sub-blocks. Detailed descriptions of the newly incorporated places and transitions are provided in Table 4, while the guard expressions relevant

Table 4 HCS—additional information about model transitions

Transition	Description	Firing semantic	Weight	Priority
$cpdc_n\_add$	Add a new control plane	–	1.0	1
$cpdc_n\_start$	Control plane starting time node	IS	–	1
$cpdc_n\_halt$	Remove a control plane	–	1.0	1
$cpdc_n\_sd$	Shutdown control plane	IS	–	1
$wndc_n\_add$	Add a new worker node	–	1.0	1
$wndc_n\_start$	Worker node starting time	IS	–	1
$wndc_n\_halt$	Remove a worker node	–	1.0	1
$wndc_n\_sd$	Shutdown worker node	IS	–	1
$ndc_n cap\_r$	Remove available cap	–	1.0	1

to the modified model are outlined in Table 6. Therefore, Fig. 6 presents the LCS and HCS versions with the aim of demonstrating the changes necessary to represent the HCS. This section describes the use of the HCS block. As expounded in Sect. 3.2, systems employing a HCS are characterized by the utilization of the minimum necessary number of active nodes. This strategy's adaptability is facilitated through the integration of new places, transitions, and expressions within the model, which are tasked with the activation and deactivation of nodes within a data center. Consequently, the count of connected worker and control nodes in such a system is not fixed but variable, adapting as per the system's ongoing requirements.

In the system model, the quantity of standby nodes is delineated by the number of tokens present in $cpdc_n cap$. Should the extant nodes fall short of fulfilling the configured Pod demand within the system, the immediate transition $wndc_n\_add$ is activated. The criteria for triggering this transition are meticulously governed by the guard expression [g8], which is defined in accordance with Eq. (16). This mechanism ensures a dynamic response to varying system demands, enabling the model to accurately simulate the operational dynamics of a HCS in a Kubernetes environment.

$$total\_var\_cap < \sum_{m \in M} min\_pod_m, \tag{16}$$

where:

$$total\_var\_cap = \sum_{n \in N} ((\#wndc_n\_starting + \#wndc_n\_on) \times ppn\_dc_n \times (\#dc_n\_reg\_op)), \tag{17}$$

and M = {$m \mid m$ are the Pod types}.

Upon the execution of $wndc_n\_add$, a token is reappropriated from the reserve of standby nodes, denoting the initiation of a node's activation process. This token is then relocated to $wndc_n\_starting$, symbolically representing the commencement of the node's power-up sequence. The temporal dynamics associated with the transition $wndc_n\_start$ are defined by its time distribution, which specifies the duration required

to activate a node. The activation of $wndc_n\_start$ results in the transference of a token from $wndc_n\_starting$ to $wndc_n\_on$, thereby signifying that the node is now operational and ready to accommodate Pods. The aggregate of tokens within $wndc_n\_on$ quantitatively represents the number of connected Worker Nodes in the system.

Failures and recoveries of hardware and software components, akin to those in the low-consolidation model, are modeled through the transitions $wndc_n\_fail$, $wndc_n\_rep$, and the place $wndc_n\_failed$. The transition $wndc_n\_halt$ is indicative of a worker node being decommissioned, with the process of deactivation represented by the place $wndc_n\_halting$. The activation of $wndc_n\_halt$ is contingent upon satisfying the guard condition [g9], which is articulated in Eq. (18). This condition ensures the model dynamically reflects the operational changes and requirements of a high-consolidation strategy, particularly in the context of managing the active status of Worker Nodes.

$$\sum_{n \in N} \left(\#wndc_n\_on * ppn\_dc_n \right) > \sum_{m \in M} min\_pod_m, \tag{18}$$

This process involves an evaluation to determine if the system's current capacity for instantiating Pods exceeds the total demand for Pods. Upon satisfying this condition, the transition $wndc_n\_sd$ is triggered, signifying the effective deactivation of a node. The duration required for this deactivation is determined by the time distribution associated with the $wndc_n\_sd$ transition. Once deactivated, the node is subsequently classified as available for future utilization and is accordingly added to $cpdc_n cap$, thereby facilitating its potential reactivation.

A similar operational mechanism has been implemented for Control Plane nodes within the model. In instances where the actual number of Control Planes falls below the system's configured requirement, a new Control Plane is initiated. This initiation is achieved through the activation of $cpdc_n\_add$, which involves the reallocation of a token from the $cpdc_n cap$ capacity to $cpdc_n\_starting$. The execution of $cpdc_n\_add$ is regulated by the guard expression [g10], as formulated in Eq. (19). This guard expression ensures that new Control Planes are only initiated when necessary, aligning with the high-consolidation strategy's emphasis on optimizing resource utilization.

$$\sum_{n \in N} ((\#cpdc_n\_starting + \#cpdc_n\_on) \times (\#dc_n\_reg\_op)) < min\_cp. \tag{19}$$

Upon completion of the initialization phase, the transition $cpdc_n\_start$ is activated, culminating in the placement of a token into $cpdc_n\_on$. This action symbolically signifies the integration of a new Control Plane into the system. The quantity of tokens housed in $cpdc_n\_on$ directly corresponds to the number of connected Control Planes, providing a quantifiable representation of the system's operational Control Plane components. In parallel with the low consolidation model, the occurrences of hardware and software-dependent failures, along with their respective recovery processes, are modeled through the transitions $cpdc_n\_fail$ and $cpdc_n\_rep$, as well as the place $cpdc_n\_failed$. These elements collectively facilitate the simulation of failure and repair dynamics within the Control Plane. Moreover, the procedure for the removal of a Control Plane is executed via the transition $cpdc_n\_halt$. The conditions necessitating this removal are delineated by the guard expression [g11], which is

Table 5 HCS—guard expressions with usage dependent on model application

Model usage	Index	Guard expression	Description
Disaster analysis	[g7]	$\#dc_n\_disaster = 1$	Remove all data center n node capacity if a disaster occurs
Stationary metrics	[g7]	False	Transition never fires in this usage
System reliability	[g7]	Eq. (1)	Remove all n node capacity if the system is in a failed state

Table 6 HCS—guard expression

Index	Guard expression	Description
[g8]	Eq. (16)	Add more worker nodes when current capacity cant fulfil sla requirement
[g9]	Eq. (18)	Remove worker nodes when current alocated capacity is higher then sla requirement
[g10]	Eq. (19)	Add Control plane when current capacity cant fulfil sla requirement
[g11]	Eq. (20)	Remove Control plane when current alocated capacity is higher then sla requirement

methodically defined in Eq. (20). This equation plays a pivotal role in determining the appropriate circumstances under which a Control Plane should be decommissioned, ensuring that the model accurately reflects the operational protocols of a high-consolidation strategy in a Kubernetes environment.

$$\sum_{n \in N} (\#cpdc_n\_on) > min\_cp, \tag{20}$$

which initiates the shutdown, and ends with the firing of $cpdc_n\_sd$, which returns the node to the available nodes.

In the HCS model, the Pods block retains its configuration as established in the low consolidation model. The control over Pod capacity is maintained by monitoring the number of active Worker Nodes, as indicated by the tokens in $wndc_n\_on$. Analogous to the low consolidation model, this high consolidation model is equally adept at facilitating various analyses, including disaster transient analysis, reliability analysis, and the evaluation of stationary metrics. The specific type of analysis to be conducted is determined by the application of guard expressions [g1], [g2], and [g7], with their respective configurations detailed in Tables 5 and 6.

5.3 Model of eDRECS

The model depicted in Fig. 7 represents the eDRECS proposed in this study. This model is essentially an extension of the high consolidation model, enriched with additional places and transitions that pertain to the boost and consolidation phases. The newly incorporated places include $wndc_n\_w\_rp$ and $p_ndc_m\_tsd$. Furthermore, the

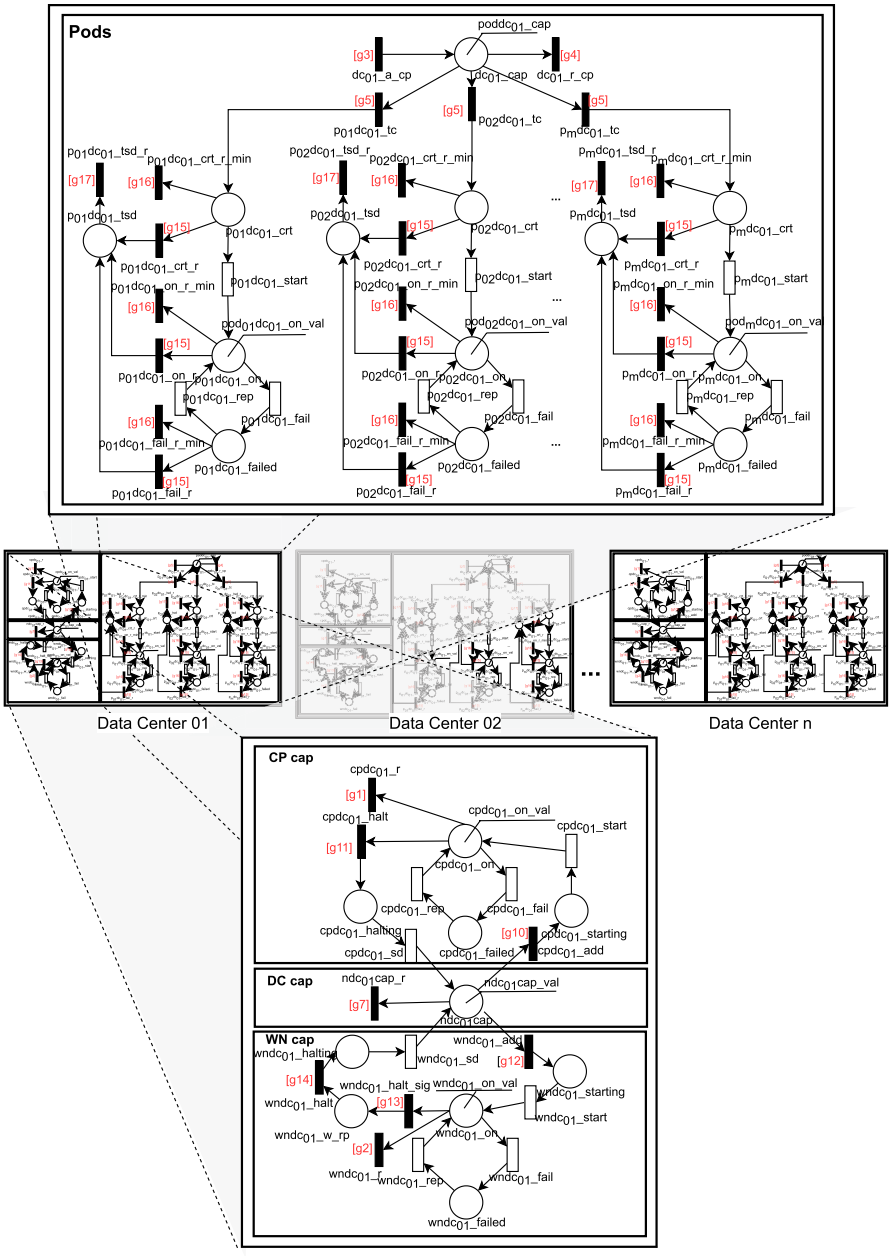


Fig. 7 eDRECS Consolidation Strategy—GSPN Model

augmentations made in terms of transitions and guard expressions are comprehensively outlined in Tables 7 and 8, respectively. These additions are instrumental in characterizing the unique aspects of the eDRECS, enabling the model to simulate the dynamic

Table 7 eDRECS—additional information about model transitions

Transiton	Description	Firing semantic	Weight	Priority
$wndc_n\_halt\_sig$	Start to remove a worker node	–	1.0	1
$p_m dc_n\_crt\_r$	Remove creating pods type m in data center n when has more cap than necessary	–	Eq. (13)	2
$p_m dc_n\_on\_r$	Remove healthy pods type m in data center n when has more cap than necessary	–	Eq. (14)	2
$p_m dc_n\_fail\_r$	Remove failed pods type m in data center n when has more cap than necessary	–	Eq. (15)	2
$p_m dc_n\_crt\_r\_min$	Remove creating pods type m in data center n when pod cap is higher minial required	–	Eq. (13)	2
$p_m dc_n\_on\_r\_min$	Remove healthy pods type m in data center n	–	Eq. (14)	2
$p_m dc_n\_fail\_r\_min$	Remove failed pods type m in data center n	–	Eq. (15)	2
$p_m dc_n\_tsd_r$	Effectively removes excess pods when it is possible to shut down the worker node	–	1	1

Table 8 eDRESCS—guard expression

Index	Guard expression	Description
[g12]	Eq. (21)	Add wd when total allocated capacity is less than minimum amount or there are failed data centers and instantiated capacity is less than minimum
[g13]	Eq. (23)	Request remove a wd when there is more capacity instanced than the minimum amount of pods and the total available capacity is greater than or equal to the minimum amount of pods
[g14]	Eq. (27)	WN can effectively shut down when all its pods are removed
[g15]	Eq. (25)	Request to remove a pod when there is more capacity than the total available after WN shutdown request
[g16]	Eq. (28)	Removing a pod when there are more pods of a given type than the minimum amount for that pod
[g17]	Eq. (26)	Effectively remove a pod when there is more pod capacity attached, or when the instanced capacity in the data center is greater than the amount of nodes to sustain

interplay between rapid response capabilities and energy-efficient consolidation in a Kubernetes environment.

When a disaster occurs, the boost phase begins, starting with the instantiation of Worker nodes and Pods. Node instantiation is performed by the transition $wndc_n\_add$ with enablement dependent on the guard expression [g12], in the following Equation:

$$\left(total\_var\_cap < \sum_{m \in M} min\_pod_m\right) \vee \left(\left(\left(\#dc_1\_reg\_op = 0\right) \vee \left(\#dc_2\_reg\_op = 0\right) \vee ...\left(\#dc_n\_reg\_op = 0\right)\right) \wedge \left(av\_pod < \sum_{m \in M} min\_pod_m\right)\right), \tag{21}$$

where:

$$av\_pod = \sum_{n \in N} \sum_{m \in M} ((\#p_m dc_n\_crt + \#p_m dc_n\_on + \#p_m dc_n\_failed) \times (\#dc_n\_reg\_op)). \tag{22}$$

The transition $wndc_n\_add$ is fired whenever there is less capacity than necessary or when there is a disaster that has not yet been recovered. Therefore, nodes can be added even if the future capacity is greater than necessary.

After reaching the expected minimum number of Pods in the system, it is necessary to reconsolidate the system, relocating the Pods to the nodes that should remain active. The removal of nodes begins with the firing of $wndc_n\_halt\_sig$. This transition is enabled through the guard condition [g13], expressed in Eq. (23):

$$(av\_pod \geq \sum_{m \in M} min\_pod_m) \wedge (av\_cap > \sum_{m \in M} min\_pod_m), \tag{23}$$

where:

$$av\_cap = \sum_{n \in N} ((\#wndc_n\_on \times ppn\_dc_n) \times (\#dc_n\_reg\_op)). \tag{24}$$

The guard condition [g13] within the model performs a critical assessment to ascertain if the current number of instantiated Pods meets or exceeds the system's requirements, and concurrently evaluates whether the existing number of nodes surpasses the operational necessity of the system. In scenarios where the system is identified as possessing an excess of nodes, a protocol for node shutdown is initiated. This process is modeled by the introduction of a token into the place $wndc_n\_w\_rp$, symbolically indicating a request for node deactivation. Subsequent to the initiation of a Worker Node shutdown, it becomes imperative to migrate the Pods from the deactivating node to a node within the consolidated environment. The process of Pod removal is facilitated by the transitions $p_m dc_n\_on\_r$, $p_m dc_n\_fail\_r$, $p_m dc_n\_crt\_r$, and $dc_n\_r\_cp$. These transitions are instrumental in executing the consolidation removal process, which is governed by the guard condition [g15]. The specifics of

this condition are mathematically formulated in Eq. (25), ensuring that the model accurately reflects the optimized consolidation behavior as part of the overall strategy in managing Kubernetes environments.

$$used\_cap\_dc_n > \#wndc_n\_on * ppn\_dc_n. \tag{25}$$

If there are tokens in all these places, the choice is given by the weights of the transitions, using the Eqs. (12), (13), (14), and (15). Firing any of these transitions deposits a token into $p_n dc_m\_tsd$, causing Pods to be marked for shutdown.

Pods designated for deactivation continue to operate until the system has sufficient replacement Pods in place. These Pods are only effectively deactivated once the system confirms an adequate number of operational Pods. In the interim, while awaiting the addition of new Pods, these marked Pods remain active. The process of adding new Pods is initiated by the activation of the transition $p_m dc_n\_tc$. This transition is specifically designed to trigger under conditions where the places $p_m dc_n\_on$, $p_m dc_n\_failed$, and $p_m dc_n\_crt$ are devoid of Pods. Once the system ascertains that it possesses a sufficient number of operational Pods, the transition $p_m dc_n\_tsd\_r$ is then activated to execute the removal of the Pods earmarked for deactivation. The operational dynamics of the $p_m dc_n\_tsd\_r$ transition are meticulously governed by Eq. (26), which checks for each pod type m, whether there are more additional Pods created, adding up all active Pods and comparing with the minimum quantity configured for type m (min_pod_m). This equation plays a pivotal role in determining the appropriate juncture for Pod removal, thereby ensuring that the model accurately simulates the process of Pod deactivation and replacement in line with the strategy's emphasis on maintaining system efficiency and resource optimization.

$$\begin{aligned}&\left(\sum_{m\in U}((\#p_m dc_n\_on) \times (\#dc_n\_reg\_op)) \geq min\_pod_m\right)\\ &\quad \vee \left(\left(\sum_{m\in U}(\#p_m dc_n\_crt + \#p_m dc_n\_on + \#p_m dc_n\_failed) + \#dc_n\_cap\right)\right.\\ &\quad \left.> \left(ppn\_dc_n \times (\#wndc_n\_on + \#wndc_n\_w\_rp)\right)\right) \vee (\#wndc_n\_on = 0)\end{aligned} \tag{26}$$

Finally, removing these Pods allows the $wndc_n\_halt$ transition to fire by enabling guard expression [14], in Eq. (27):

$$\sum_{m\in M} \#p_m dc_n\_tsd = 0 \tag{27}$$

which effectively initiates the shutdown of the nodes, depositing them into $wndc_n\_halting$.

Tokens residing in $wndc_n\_halting$ are subject to a waiting period, contingent upon the node shutdown duration as specified by the $wndc_n\_sd$ transition. This interim phase, preceding the deactivation of Pods, potentially results in a transient escalation in the number of Pods within the system. However, such a temporary increase is strategically implemented to safeguard against any Service Level Agreement (SLA)

violations that might otherwise transpire during the consolidation process. Additionally, the model accommodates scenarios where Pod removal is necessitated due to the presence of excess Pods within the system. This removal process is meticulously orchestrated by transitions governed by the guard expression [g16]. The specifics of this expression, which plays a critical role in determining the conditions under which excess Pods are removed, are delineated in the following equation:

$$used\_cap\_dc_n > \sum_{m \in M} min\_pod_m \tag{28}$$

5.4 System metrics

In order to appraise the efficacy of each approach delineated within the model, a suite of metrics is employed. These metrics encompass the count of active Pods, the number of active Worker Nodes, as well as key performance indicators such as availability, reliability, MTTF, and power consumption. Furthermore, in conjunction with the parameters specified in the models, it is imperative to also consider a set of system parameters. These parameters are integral to the comprehensive evaluation of the model's performance and are enumerated in Table 9. The calculation and consideration of these parameters are essential in deriving a holistic understanding of the efficiency and effectiveness of each approach within the modeled scenarios. The formulation of these metrics and their respective computational aspects are encapsulated in Eq. (29) that measures the expected number of active Pods:

$$total\_exp\_pod = \sum_{m \in M} E(\#p_m dc_n\_on), \tag{29}$$

the expression $E(\#p_m dc_n\_on)$ defines the expected number of tokens in place of $\#p_m dc_n\_on$, this expression is presented in Eq. (30):

$$E(\#p_m dc_n\_on) = \sum_{i=1}^{n} P(\#p_m dc_n\_on = i) * i \tag{30}$$

where $P(\#p_m dc_n\_on = i)$ represents the probability of there being i tokens in place $p_m dc_n\_on$ [45].

This approach meticulously focuses on Pods that are functioning in alignment with their designated operational specifications, thereby excluding any Pods that are experiencing failures from the analysis. The count of operational Pods serves as a pivotal metric for system administrators, enabling them to determine the system's capability to manage the anticipated workload efficiently. Crucially, this metric needs to satisfy a minimum threshold value as stipulated in the Service Level Agreement (SLA). This requirement takes into account the potential for inherent system failures and the necessity for timely recovery within the RTO, particularly in scenarios involving disaster occurrences. Employing transient simulation for the analysis of the number of operational Pods metric provides an invaluable perspective on the system's average performance over time. This

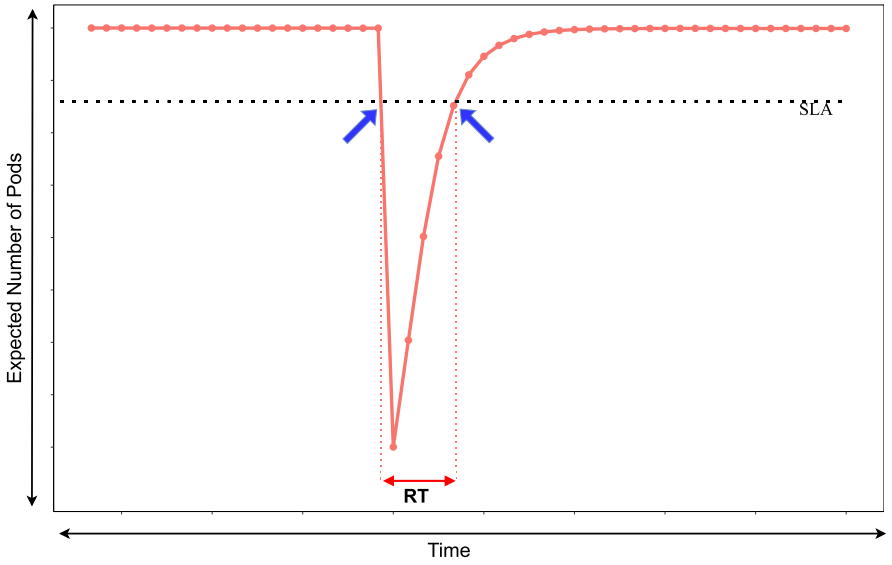


Fig. 8 RT identification through transient model analysis

Table 9 Meaning of additional model metrics parameters

Index	Guard expression
ρ_n	Average power used by the control plane in data center n
φ_n	Average power used by the worker node in data center n
ω_n	Average power used by the down node in data center n
θ_n	Average power consumed to turn on the control plane in data center n
υ_n	Average time to turn on the control plane in data center n
ζ_n	Average time to turn on the worker node in data center n
$\alpha_{n,m}$	Average power consumed by pod m in data center n
$\beta_{n,m}$	Average power consumed to turn on pod m in data center n
γ_n	Average power consumed to turn on the worker node in data center n
$\tau_{n,m}$	Average time to turn on pod m in data center n
π_n	Average time to turn off the control plane in data center n
ϱ_n	Average power consumed to turn off the control plane in data center n
ϕ_n	Average time to shut down worker node in data center n
ψ_n	Average power consumed to turn off the worker node in data center n

method allows for a comprehensive understanding of how the system behaves under varying conditions, including its response to and recovery from disruptions. Figure 8 showcases the results derived from a transient analysis focusing on the expected count of operational Pods within the system during a disaster event. Within this figure, two distinct blue arrows are utilized to delineate the onset of the disaster and the subsequent point of recovery. The temporal span between

these two demarcated points effectively represents the system's RT, offering a quantifiable measure of the system's ability to respond to and recuperate from disaster scenarios.

The Eq. (31) measures the system's expected number of functional worker nodes:

$$total\_exp\_wn = \sum_{n \in N} E(\#wndc_n\_on), \tag{31}$$

this metric evaluates the expected capacity available for allocating Pods, it allows you to check the use of system capacity. The availability assessment is presented in Eq. (32):

$$\begin{aligned} A = P\{ & \Big(\sum_{n \in N} (\#cpdc_n\_on) \geq sla\_cp \Big) \wedge \Big(\sum_{n \in T} (\#p_1 dc_n\_on) \geq sla\_pod_1 \Big) \\ & \wedge \Big(\sum_{n \in T} (\#p_2 dc_n\_on) \geq sla\_pod_2 \Big) \wedge ... \Big(\sum_{n \in T} (\#p_m dc_n\_on) \geq sla\_pod_m \Big) \}, \end{aligned} \tag{32}$$

this metric allows for a stationary assessment of the system's availability.

The availability of the system is defined based on specific criteria: the system is deemed available when it maintains at least *sla_cp* operational control planes and a minimum of sla_pods_m for each Pod type. This criterion ensures that the microservices within the system are not only operational but also manageable and accessible, thereby aligning with the system's intended functional parameters. As detailed in Sect. 5, the model's ability to compute the reliability of the system hinges on the manipulation of conditions [g1], [g2], and [g7]. If these conditions are changed, the model's focus is shifted to reliability, as the aforementioned changes change the model to an absorbent version, causing the model to absorb the tokens at each system failure state. With this change, Eq. (32) results in the system reliability value [46, 47]. Consequently, this allows for the calculation of the average absorption time, which provides insights into the average duration between failures within the system. Moreover, the model incorporates three distinct equations, each corresponding to one of the consolidation strategies, for calculating the system's energy consumption. These equations are tailored to reflect the unique energy dynamics associated with the LCS, HCS, and the eDRECS, thereby enabling a precise and strategy-specific assessment of energy utilization.

Equation (33) presents the electrical consumption of the system with less consolidation:

$$\begin{aligned} P = & \sum_{n \in N} \Big(E(\#cpdc_n\_on) * \rho_n + E(\#wndc_n\_on) * \varphi_n \Big) \\ & + \sum_{n \in N} \sum_{m \in M} \Big(E(\#p_m dc_n\_on) * \alpha_{n,m} + \frac{E(\#p_m dc_n\_crt) * \beta_{n,m}}{\tau_{n,m}} \Big), \end{aligned} \tag{33}$$

it takes into account the electrical consumption of active nodes and Pods in the system, as well as the consumption of instantiating new Pods in the architecture. If we use the highest consolidation strategy, we must use Eq. (34):

$$
\begin{aligned}
P = \sum_{n \in N} \Bigg(& E(\#cpdc_{n}\_on) * \rho_n) + E(\#wndc_{n}\_on) * \varphi_n + E(\#dc_{n}\_cap) * \omega_n \\
& + \frac{E(\#cpdc_{n}\_starting) * \theta_n}{\upsilon_n} + \frac{E(\#wndc_{n}\_starting) * \gamma_n}{\zeta_n} \\
& + \frac{E(\#cpdc_{n}\_halting) * \pi_n}{\varrho_n} + \frac{E(\#wndc_{n}\_halting) * \psi_n}{\phi_n} \Bigg) \\
& + \sum_{n \in N} \sum_{m \in M} \left(E(\#p_m dc_{n}\_on) * \alpha_{n,m} + \frac{E(\#p_m dc_{n}\_crt) * \beta_{n,m}}{\tau_{n,m}} \right).
\end{aligned}
\tag{34}
$$

the HCS strategy also requires adding the cost of activating the nodes and keeping the node in stand-by mode, which can be turned off, waiting for a signal to turn on, or in standby mode. The Equation used in the strategy with optimized boosting can be seen in Equation:

$$
\begin{aligned}
P = \sum_{n \in N} \Bigg(& E(\#cpdc_{n}\_on) * \rho_n + E(\#dc_{n}\_cap) * \omega_n + E(\#wndc_{n}\_w\_rp) * \varphi_n \\
& + (E(\#wndc_{n}\_on) + \frac{E(\#cpdc_{n}\_starting) * \theta_n}{\upsilon_n} + \frac{E(\#wndc_{n}\_starting) * \gamma_n}{\zeta_n} \\
& + \frac{E(\#cpdc_{n}\_halting) * \varrho_n}{\pi_n} + \frac{E(\#wndc_{n}\_halting) * \psi_n}{\phi_n} \Bigg) \\
& + \sum_{n \in N} \sum_{m \in M} \left((E(\#p_m dc_{n}\_on) + E(\#p_m dc_{n}\_tsd)) * \alpha_{n,m} + \frac{E(\#p_m dc_{n}\_crt) * \beta_{n,m}}{\tau_{n,m}} \right),
\end{aligned}
\tag{35}
$$

where the costs of maintaining Pods and nodes in consolidation were added concerning the strategy with greater consolidation. The average power presented in this work only considers the consumption of nodes and servers. The consumption of other network, cooling, and operational equipment is outside the scope of this work. The electrical consumption values used in the metrics can be obtained from measurements, technical specifications from manufacturers, and consumption models already existing in the literature [31, 48–50].

5.5 Validation

This subsection presents the validation of the model used to identify infrastructure recovery times for microservices orchestrated by Kubernetes distributed across multiple data centers. The objective of this validation is to compare the recovery time value calculated by the model with the recovery time experimented in real systems. The modified real scenario was composed of a Kubernetes-based microservices infrastructure deployed in different data centers on Amazon AWS. Six t2.medium instances were used to install the Worker Nodes (WNs) and one t2.medium to install the Control Plane [51]. The infrastructure was distributed across three different availability zones, which represent different data centers in a region [51]. Each distinct data center has received two WN, in order to distribute the infrastructure as

Table 10 Model and real system RT results

Environment	Sample size	Mean	StDev	SE mean
Real system	30	0.1161	0.0577	0.011
Model	30	0.1041	9.08e−7	1.7e−7

presented in the model. We use a container type executing a nginx application. The minimum number of Pods was configured in Kubernetes (twelve Pods), and we have adopted eleven as our SLA.

To perform the validation, we used the failure and repair times from the literature [31, 49, 52–55], while the time for detection and disaster recovery of pods must be obtained for each specific case. In our case the workload for starting a new pod was defined as updating and building the internal codes for presentation in nginx. The individual startup times of a Pod were then measured. We repeated the measurement thirty times to obtain a representative sample of the real system. The data obtained from the measurement were subjected to statistical analysis to identify the distribution and its parameters for insertion into the model. We used the normal distribution, as it was one of those that fitted with 95% confidence, and it was also present in Mercury to represent timed transitions. The values for failure detection and initialization times by the cluster were 0.09374 for the mean and 0.00253 for the standard deviation. Then we have fed the model with the values from this distribution, as well as using the MTTFs found in the literature for VMs and containers.

The next step was to simulate the disaster of a DC in AWS. Such simulation was performed using AWS CLI commands to shut down all nodes on a specific DC. From this point onwards, the instantiation times of new Pods by Kubernetes were measured to reach the minimum value configured in the cluster. This procedure was also replicated 30 times and then compared with the results obtained by the model. Table 10 presents the statistical results of the real system and the model. The *P*-value result of the Two-Sample T-Test with 95% confidence of the results is 0.268, as this value is greater than 0.05 we cannot refute the null hypothesis. Therefore, the result generated by the model is statistically equivalent to the real system. For the purpose of ensuring that the results obtained during the validation process are replicable and to provide assistance to model users, we have made the code used for experimentation available.[1] It is worth noting that the infrastructure was developed using the AWS Cloud Development Kit (CDK), which is a framework used for provisioning infrastructure as AWS code. Therefore, it is possible to implement an infrastructure similar to what was used in this work in a simplified manner. Additionally, scripts were created to install Kubernetes CPs and WNs and these were added to the VM launcher. The project also includes scripts for obtaining initialization times and RTs of the real infrastructure.

[1] https://gitlab.com/iuresf/disaster_validator.

Table 11 Parameters applied in case studies

Parameter	Value
cp_1_mttf, $wn\_d_1\_mttf$, cp_2_mttf, $wn\_d_2\_mttf$, cp_3_mttf, $wn\_d_3\_mttf$	8760 h
pod_{01}_mttf, $pod\_{}_{03}\_mttf$	1258.0 h
pod_{02}_mttf	768 hw
cp_{01}_mttr, $wn\_d_{01}\_mttr$	1 h
pod_{01}_mttr,pod_{02}_mttr, $pod\_{}_{03}\_mttr$	0.238 h
$\tau_{1,1}$, $\tau_{1,2}$, $\tau_{1,3}$, $\tau_{2,1}$, $\tau_{2,2}$, $\tau_{2,3}$, $\tau_{3,1}$, $\tau_{3,2}$, $\tau_{3,3}$	0.1 h
υ_1, ζ_1, υ_2, ζ_2,υ_3, ζ_3	0.2 h
π_1, ϕ_1, π_2, ϕ_2,π_3, ϕ_3	0.0833 h
$\alpha_{1,1}$, $\alpha_{1,2}$, $\alpha_{1,3}$, $\alpha_{2,1}$, $\alpha_{2,2}$, $\alpha_{2,3}$, $\alpha_{3,1}$, $\alpha_{3,2}$, $\alpha_{3,3}$	23.0911 W
ρ_1, φ_1,ρ_2, φ_2,ρ_3, φ_3	170 W
$\beta_{1,1}$, $\beta_{1,3}$, $\beta_{2,1}$, $\beta_{2,3}$,$\beta_{3,1}$, $\beta_{3,3}$	0.01 W
$\beta_{1,2}$, $\beta_{2,2}$, $\beta_{2,3}$	0.02 W
θ_1, γ_1, θ_2, γ_2, θ_3, γ_3	0.05 W
ω_1, ω_2, ω_3	0.001 W
o_1, o_1, o_2, ψ_1, ψ_2, ψ_3	0.04 W
ppn_dc_1, ppn_dc_2, ppn_dc_3	9

6 Case studies

This section delineates the outcomes derived from the analysis of the models proposed in Sect. 5. The ensuing subsections will systematically explore the availability and reliability of all models under consideration. These results were ascertained through a combination of stationary analysis for availability and transient analysis for reliability. The Mercury Tool[2] [56] was employed for modeling and generating these results. The Mercury modeling tool is a powerful software tool used for modeling and analyzing complex systems. It is specifically designed for modeling stochastic Petri nets, including Generalized Stochastic Petri Nets (GSPN). The tool provides a user-friendly interface for creating models and supports a range of analysis techniques, including steady-state analysis, transient analysis, and sensitivity analysis. The Mercury tool also includes a range of optimization algorithms that can be used to optimize system performance. Overall, the Mercury modeling tool is a valuable tool for researchers and engineers working in fields such as computer science, operations research, and engineering. In our work we have executed the tool in a local machine, with 11th Gen Intel(R) Core(TM) i9-11900K processor (8 cores-3.50GHz), 126 GB DDR4 RAM, 512 nvme, and Ubuntu 20.04 operating system.

Table 11 enumerates the values assigned to each system component, sourced from existing literature that previously utilized these values [57, 58]. Additional parameters, such as the time interval between server changes and failure detection duration, were established empirically, with the values falling within a range of minutes. Two

[2] https://www.modcs.org/

distinct use cases were developed to demonstrate the practicality of the approach advocated in this work, presenting a comparative analysis of the eDRECS with other strategies. These cases involved an analysis of the transient behavior of the number of Pods, Worker Nodes, and the average electrical power consumption during a disaster. Additionally, the performance of these metrics in each strategy, in the context of a disaster-free system, was also scrutinized.

In these use cases, the system is conceptualized as comprising three data centers, each equipped with 14 nodes dedicated to the cluster. Out of these nodes, 3 are allocated as Control Planes, with the remaining serving as Worker Nodes. The system operates with three types of microservices, which can be deployed in any of the data centers. The system's Service Level Agreements (SLAs) were defined to ensure 150 active Pods, with 50 of each type, and an RTO of 0.25 h. For the purpose of this analysis, the system configuration adheres to parameters as outlined in Table 11, derived from a range of scholarly and technical sources [31, 49, 52–55]. It is, however, important to acknowledge that such values may vary significantly across different organizations, contingent on variables such as hardware specifications, application requirements, configuration settings, desired SLA levels, and available support infrastructure. Owing to the adaptable nature of the methodology employed in this work, the analysis can be tailored to specific case studies, requiring only the customization of configuration parameters to align with each unique scenario. The simulation outcomes are presented along with their 95% confidence intervals.

6.1 Scenario one: analysis of sudden data center failure

This use case investigates the repercussions of an unanticipated failure in a single data center, set to transpire 1.5 h into the scenario. The scenario contemplates a catastrophic event that renders a data center entirely non-operational. The primary objective is to evaluate and compare various strategies in terms of their recovery time and electrical consumption in response to this disaster. The specific parameters utilized in the models are delineated in Table 11.

For the system to be deemed operational under this scenario, an SLA of 150 Pods is established, with an expectation of recovery within 0.25 h (RTO) post-disaster. Concurrently, strategies that minimize electrical consumption are also sought. The critical factor in meeting these requirements is identifying the minimum number of Pods k necessary to achieve the RTO, as illustrated in Fig. 8. This parameter k is determined through simulations across various network scenarios, each featuring different quantities of redundant Pods k. The optimal value of k is ascertained as the lowest number that yields a RT equal to or less than the RTO. Distinct recovery strategies necessitate varying values of k.

The analysis will focus on the lower k values identified for each strategy, enabling a detailed examination of the transient behavior of Pods recovery metrics, Worker Nodes (WN), electrical consumption, and reliability. These k values will also inform the calculation of stationary values related to the system's dependability and electrical consumption metrics across different strategies.

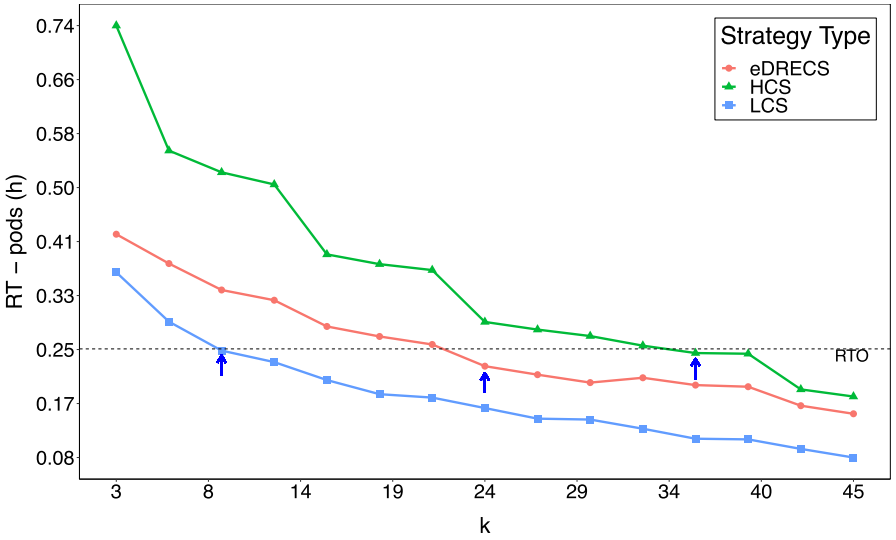


Fig. 9 Scenario one—spare pods variation

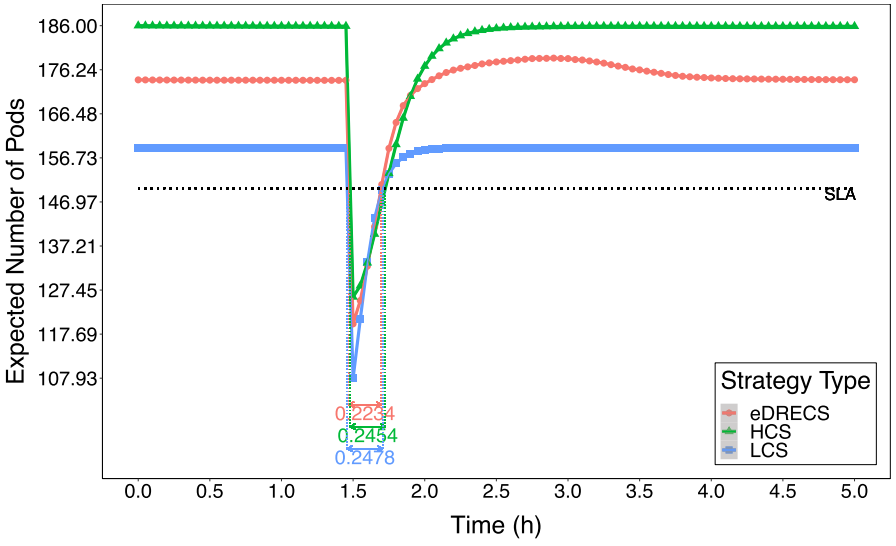


Fig. 10 Scenario one—expected number of pods

Figure 9 presents the impact of varying the number of redundant Pods (k) in the system under different strategies. It is observed that to meet the minimum RTO in the HCS, the system requires the addition of 36 redundant Pods. In contrast, the eDRECS necessitates a minimum of 24 redundant Pods, while the LCS requires only 9 Pods to achieve the RTO. These optimal k values for each strategy are highlighted in Fig. 9.

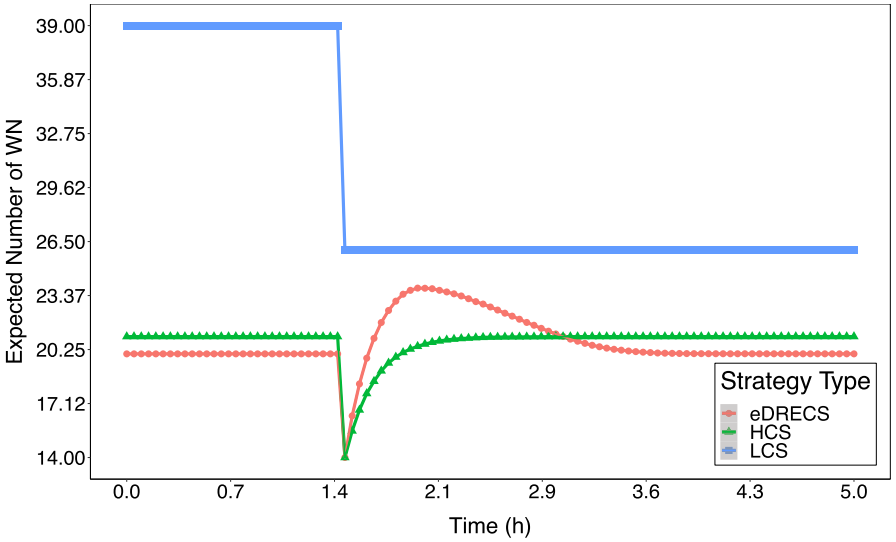


Fig. 11 Scenario one—expected number of worker nodes

These findings will be utilized to evaluate the Recovery Time, Worker Nodes, electrical consumption, and reliability metrics, as depicted in Figs. 9, 10, 11, and 13, respectively. Furthermore, each strategy will entail varying levels of energy consumption, contingent upon the number of Pods required to adhere to the RTO. Strategies that exhibit greater consolidation at the disaster's onset will likely necessitate additional Pods to offset the latency associated with activating computing nodes.

Post determination of the requisite number of redundant Pods for each strategy, an analysis was conducted on the other metrics associated with the K configuration employed in each respective strategy. Figure 10 delineates the expected number of Pods within the system over time. The black dashed horizontal line in the figure represents the Service Level Agreement (SLA) for Pods. A noticeable decline in the available number of Pods is observed across all strategies. The green line, depicting the strategy with the least consolidation, illustrates the lowest initial configuration of Pods.

The Recovery Times (RTs) marked in Fig. 9 are also presented in Fig. 10. The LCS demonstrates a recovery duration of 0.2474 h, the HCS recovers in 0.2454 h, while the eDRECS records the fastest recovery time of 0.2234 h to reach the SLA. Post-achievement of the SLA, the strategies endeavor to attain their configured minimum Pod count. Upon reaching this threshold, both LCS and HCS stabilize, whereas eDRECS continues to increment the Pod count, attributed to its reconsolidation process, which necessitates the creation of new Pods to facilitate the shutdown of surplus Worker Nodes (WNs). As denoted in Fig. 9, all strategies successfully meet the stipulated RTO. However, the selected strategy significantly impacts the required number of Pods to achieve this objective. Furthermore, as explicated in Eqs. (33), (34), and (35), the addition of Pods influences the system's electrical consumption.

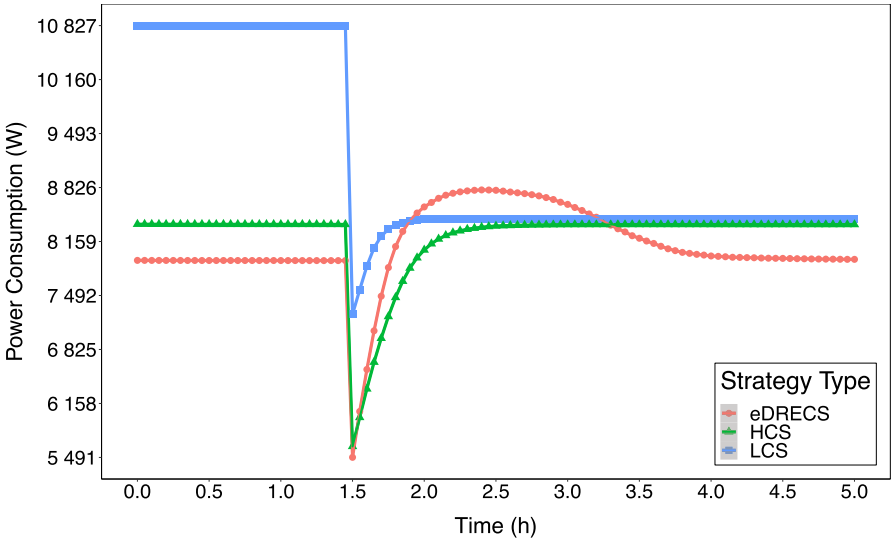


Fig. 12 Scenario one—power usage

Figure 11 illustrates the variation in the number of active Worker Nodes over time. In the LCS, the number of these elements alters only during the disaster, owing to node removals from the data center. Conversely, in HCS, the count of Worker Nodes gradually increases post-failure, corresponding to the activation time of the nodes, until it stabilizes at the quantity needed to satisfy Pod demand. For eDRECS, a substantial increase in Worker Nodes is noted following the disaster, surpassing those in HCS, due to the boost behavior. Once the Pods SLA is attained and consolidation is implemented, the number of Worker Nodes in eDRECS reverts to a count lower than that in HCS, reflecting the strategy's dynamic response to changing operational requirements.

Figure 12 provides an insight into the average electrical consumption across different strategies. Notably, the LCS exhibits the highest average electrical consumption within the system. This increased consumption in the LCS strategy is attributed to the energy demands of the continuously connected nodes. Despite a reduction in node numbers due to the disaster, the energy usage of nodes in LCS remains higher compared to the additional Pods used in other strategies.

During the disaster, the HCS experiences a decrease in consumption, correlating with the reduced number of Pods and Worker Nodes (WNs). However, as the recovery process progresses with the reconnection of new WNs and the restoration of Pods, the consumption escalates, eventually returning to pre-disaster average levels. Notably, the introduction of spare Pods in HCS results in consumption levels comparable to those of the LCS strategy. In scenarios where spare Pods have relatively high consumption, it is anticipated that their energy usage might exceed that of LCS post-disaster.

Conversely, the eDRECS demonstrates lower electrical consumption than the other strategies both before the disaster and after achieving stabilization.

Table 12 Scenario one—stationary results

Metric	Strategy	Mean
A (%)	HCS	99.9751
	LCS	99.9344
	eDRECS	99.9930
Pods number	HCS	185.8650
	LCS	158.8892
	eDRECS	173.8836
Worker nodes number	HCS	21.0020
	LCS	39.0
	eDRECS	20.0010
Power usage (W)	HCS	8369.7279
	LCS	10,824.2364
	eDRECS	7923.4119

Nevertheless, in the interim phase—post-disaster and pre-stabilization—eDRECS temporarily exhibits increased consumption. This spike is a consequence of the boost and consolidation phase, where the number of WNs and Pods is augmented to expedite recovery, as previously observed in the Pods and Worker Nodes graphs.

Subsequent to attaining the minimum required number of Pods in the system, the eDRECS strategy initiates a consolidation phase. During this phase, Pods are methodically disconnected and reallocated to a minimal number of WNs. This strategic reallocation results in eDRECS achieving lower electrical consumption values compared to other strategies, thereby highlighting the efficiency of the eDRECS strategy in balancing rapid recovery with energy conservation.

The analysis of transient behavior during a disaster scenario is crucial, but it is equally important to ensure that the system's performance during normal operating conditions—when it is not under the influence of a disaster—does not yield any unintended consequences. To this end, a stationary evaluation of the system is conducted, focusing on periods where the system is expected to function without the impact of disaster-related disruptions. The outcomes of these evaluations pertaining to various metrics are compiled in Table 12.

The highest availabilities (A) are observed in the HCS and the eDRECS. In contrast, the least consolidated strategy, typically characterized by fewer redundant and essential nodes, exhibits the lowest availability. This pattern aligns with the operational dynamics of Kubernetes, which distributes Pods across the available nodes. Consequently, any failure in a node has a direct and proportional impact on the system's overall availability. From an availability standpoint, having a greater number of essential yet non-redundant nodes inherently increases the probability of node failure.

To mitigate this issue in a less consolidated system, one potential approach would be to incorporate a higher number of redundant Pods. However, this solution comes with a trade-off, as it would lead to a significant escalation in electrical consumption. Therefore, while addressing the availability concerns in a less consolidated system,

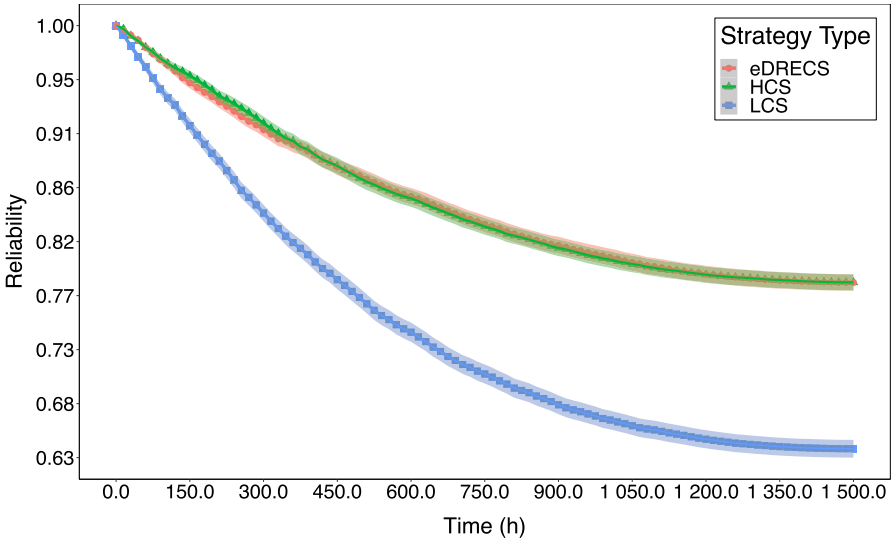


Fig. 13 Scenario one—reliability

it is imperative to carefully weigh the increased energy demands that accompany such a mitigation strategy.

Table 12 provides a comprehensive overview of the expected stationary number of Pods within the system. It's possible to see that the number of Pods that are currently available is less than the number that was originally defined in the cluster. For instance, in the case of HCS, we have only 185.8650 Pods available, whereas the initial configuration had 186 Pods, which is the minimum quantity plus the value of k. This behavior reflects the inherent vulnerabilities of the Pods and the infrastructural limitations underpinning them.

Additionally, Table 12 offers insights into the expected count of active Worker Nodes (WN) in the system. Consistent with expectations, the LCS maintains the highest number of active servers. In comparison, the HCS and the eDRECS have approximately 21 and 20 WNs, respectively. This indicates that, in the absence of disasters, the HCS strategy necessitates an additional WN to support the augmented Pod capacity.

The average stationary electrical consumption for each strategy is also delineated in Table 12. It is evident that the LCS, with its larger number of connected WNs, incurs the highest electrical consumption. Meanwhile, the HCS exhibits intermediate consumption levels, attributable to its higher Pod and WN counts compared to the eDRECS strategy. The eDRECS, due to its efficient utilization of resources, demonstrates a more economical energy profile. These findings underscore the critical role of consolidation strategies in influencing not only the operational capacity of Kubernetes environments but also their energy efficiency.

Figure 13 exhibits the reliability of the various strategies within the system. The LCS demonstrates the lowest reliability values, attributable to its higher likelihood of unavailability. This reduced reliability is a consequence of the LCS having the

Table 13 Scenario one—MTTF

Strategy	Mean	Min CI	Max CI
HCS	2881.520	2826.0179	2937.0226
LCS	1643.7390	1609.9806	1677.4974
eDRECS	2885.9356	2829.1083	2942.7629

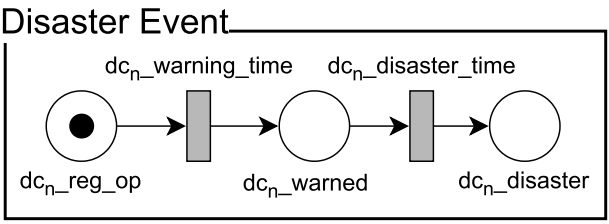


Fig. 14 GSPN warned disaster event

least number of redundant Pods and the most nodes, thereby increasing the risk of system failure due to simultaneous Pod failures or the failure of a Worker Node (WN) with multiple Pods. In contrast, both the HCS and the eDRECS exhibit comparable reliability values, with their means falling within each other's confidence intervals [59]. The reliability function provides the MTTF of the system, as shown in Eq. (36). Table 13 reports the MTTF for each strategy, indicating similar MTTFs for HCS and eDRECS, both significantly higher than LCS.

$$MTTF = \int_0^\infty R(t)dt, \tag{36}$$

The analyses—transient, stationary, and reliability—confirm that eDRECS can recover within the expected RTO with lower energy consumption compared to other strategies. Specifically, eDRECS achieved savings of 5.33% and 26.79% in stationary electrical consumption compared to HCS and LCS, respectively. Additionally, the Recovery Time for eDRECS was 8.96% and 9.7% lower than HCS and LCS, respectively. Furthermore, eDRECS exhibited MTTF values similar to HCS and superior to LCS, alongside better availability levels.

6.2 Scenario two: anticipating and mitigating warned failures

Some disasters, like natural calamities, often have identifiable precursors, allowing for preemptive actions. Anticipatory measures enable scaling of resources in unaffected data centers ahead of a disaster. This analysis alters the disaster block to the configuration depicted in Fig. 14, facilitating the evaluation of system behavior across varying warning times and determining the minimum warning duration necessary to avert downtime. In the modified disaster block model, the transition *dc$_n$_warnig_time* and the place *dc$_n$_warned* are incorporated. Initially, the system

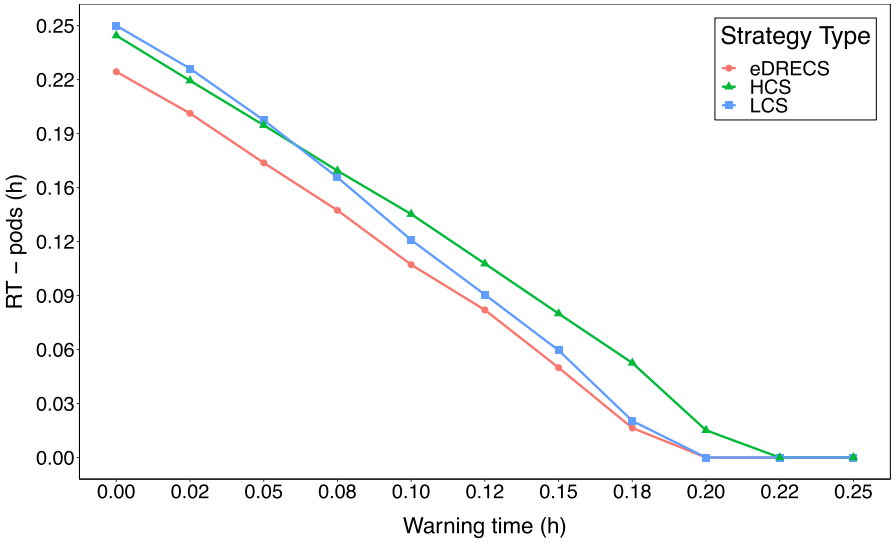


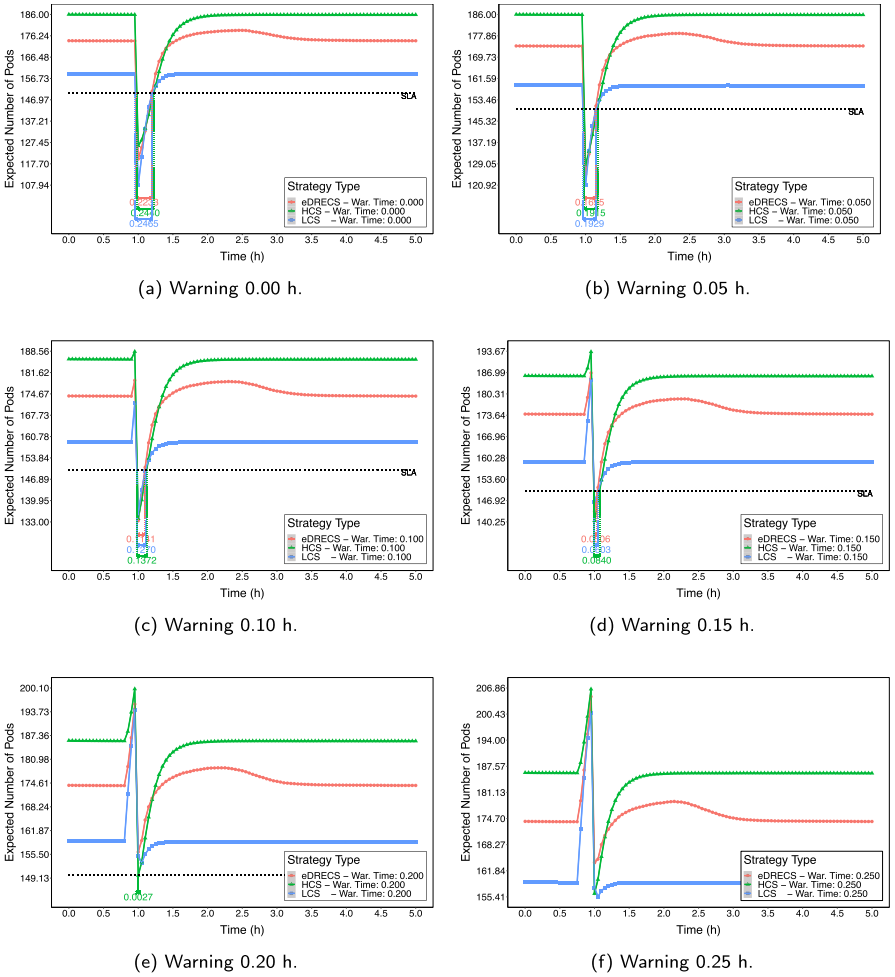
Fig. 15 Scenario two—warning time variation

is in its normal state (dc_n_reg_op). Upon activation of dc_n_warnig_time, the system transitions to a warned state (dc_n_warned), indicating imminent disaster and enabling preemptive measures. The activation of dc_n_disaster_time signifies the actual occurrence of the disaster.

When warned of an impending disaster in a data center, the architecture can presume that the Pods and Nodes in the affected center are down. This triggers the instantiation of new Pods and the connection of necessary WNs and Control Planes to compensate for the impending loss. This process is governed by the variable (#dc_n_reg_op) in guard conditions [g5], [g8], [g10], and [g12].

Using the same configuration as in Use Case 6.1, the recovery time from the disaster is analyzed. The model anticipates a disaster at 1 h with varying warning times from 0 to 0.25 h. As depicted in Fig. 15, the effectiveness of anticipation is contingent upon the warning time and the chosen consolidation strategy. Longer warning times universally reduce Recovery Times, and sufficient anticipation can completely mitigate disaster effects. The reduction in RT results from additional time to scale infrastructure, activate WNs, and create Pods. In cases with a 0.2-hour warning, both eDRECS and LCS strategies can nullify the effects of a complete data center disaster by instantiating necessary Pods in time. With warning times up to 0.05 h, HCS achieves lower RTs than LCS, after which LCS outperforms due to its greater WN activation capacity. eDRECS shows the best RTs during varying warning times, benefiting from its higher redundant Pod count and boosting strategy.

Beyond the consolidated results depicted in Fig. 15, it is insightful to examine the transient behavior of each strategy across different warning times. The graphs in Fig. 16 illustrate that as the warning time is extended, the quantity of Pods prior to the disaster correspondingly increases. In Fig. 16a and b, minimal variation in preemptive actions is noticeable, with the former representing a scenario



(a) Warning 0.00 h. (b) Warning 0.05 h.

(c) Warning 0.10 h. (d) Warning 0.15 h.

(e) Warning 0.20 h. (f) Warning 0.25 h.

Fig. 16 Scenario two—transient

without warning and the latter indicating insufficient time for substantial Pod creation pre-disaster. Nonetheless, during recovery in these scenarios, the commencement of Pod instantiation contributes to reduced Recovery Times for the strategies.

The results of 0.10 and 0.15 h (Fig. 16c and d) are better than the previous 0.00 and 0.05 warning times. The results are better because the Recovery Time decreases. It is important to note, as well, that there is an increase in the number of Pods before the disaster. Finally, observe the results of 0.20 and 0.25 h (Fig. 16e and f) With a 0.2-hour warning, only the HCS does not maintain the SLA for the minimum number of Pods because the number of pods goes lower than 150 Pods. The LCS and eDRECS, with a 0.2-hour warning, attend the SLA. A 0.25-hour warning ensures no disruptions across all strategies, and the SLA horizontal line is not even presented.

The LCS and eDRECS strategies demonstrate the shortest warning times required to prevent system disruptions during a warned disaster, with 0.2 h being sufficient. Since these strategies utilize the same configuration as in Sect. 6.1, their stationary analysis results remain applicable. Consequently, the eDRECS strategy maintains its recovery efficiency with 26% lower energy consumption and 75% higher MTTF values than LCS.

7 Conclusion

This research introduces a survivability strategy for microservice-based systems in multiple data centers, focusing on meeting Recovery Time Objectives (RTOs) and reducing electrical consumption. Using Generalized Stochastic Petri Nets (GSPN), the strategy was compared with two other approaches, providing metrics like availability, reliability, MTTF, electrical consumption, repair times, and the count of Pods and nodes for disaster recovery planning. Two use cases demonstrated the strategy's effectiveness in unexpected and warned disaster scenarios. It achieved the RTOs with 5.32% and 26.79% less energy use than high and low consolidation strategies, respectively, while maintaining system availability. In warned disasters, the eDRECS and LCS strategies needed only 0.2 h to avoid downtime, compared to 0.25 h for HCS. eDRECS showed similar results to LCS but with more energy savings and higher MTTF. These models show potential for aiding systems and disaster recovery planning, with future research possibly including data storage in models, improving performance metrics, and using optimization algorithms to further cut electrical consumption.

Funding This research was partially supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. 2020R1A6A1A03046811). This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2021R1A2C2094943)

Availability of data and materials Data sharing not applicable.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical approval Not applicable.

References

1. Ramasamy B, Na Y, Kim W, Chea K, Kim J (2022) Hacm: high availability control method in container-based microservice applications over multiple clusters. IEEE Access 11:3461–3471
2. Detti A (2023) Microservices from cloud to edge: an analytical discussion on risks, opportunities and enablers. IEEE Access
3. Kubernetes Production-Grade Container Orchestration. https://kubernetes.io/. Accessed: 2023-08-21

4. Blinowski G, Ojdowska A, Przybyłek A (2022) Monolithic vs. microservice architecture: a performance and scalability evaluation. IEEE Access 10:20357–20374
5. Charfeddine L, Umlai M (2023) Ict sector, digitization and environmental sustainability: a systematic review of the literature from 2000 to 2022. Renew Sustain Energy Rev 184:113482
6. Wang JC (2022) Understanding the energy consumption of information and communications equipment: a case study of schools in taiwan. Energy 249:123701
7. Belkhir L, Elmeligi A (2018) Assessing ict global emissions footprint: trends to 2040 & recommendations. J Clean Prod 177:448–463
8. Tchana A, De Palma N, Safieddine I, Hagimont D (2016) Software consolidation as an efficient energy and cost saving solution. Future Gener Comput Syst 58:1–12
9. Helali L, Omri MN (2021) A survey of data center consolidation in cloud computing systems. Comput Sci Rev 39:100366
10. Abualkishik AZ, Alwan AA, Gulzar Y (2020) Disaster recovery in cloud computing systems: An overview. Int J Adv Comput Sci Appl 11(9)
11. Silvaa B, Maciela PRM, Zimmermannb A, Brilhantea J (2014) Survivability evaluation of disaster tolerant cloud computing systems. In: Proc. Probabilistic Safety Assessment & Management Conference, p 12
12. Trivedi KS, Xia R (2015) Quantification of system survivability. Telecommun Syst 60:451–470
13. Longo F, Ghosh R, Naik VK, Rindos AJ, Trivedi KS (2017) An approach for resiliency quantification of large scale systems. ACM Sigmetr Perform Eval Rev 44(4):37–48
14. Avizienis A, Laprie J-C, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secur Comput 1(1):11–33
15. Welsh T, Benkhelifa E (2020) On resilience in cloud computing: a survey of techniques across the cloud domain. ACM Comput Surv (CSUR) 53(3):1–36
16. Andrade E, Nogueira B (2019) Performability evaluation of a cloud-based disaster recovery solution for it environments. J Grid Comput 17:603–621
17. Di Mauro M, Galatro G, Longo M, Postiglione F, Tambasco M (2022) Performability analysis of containerized ims through queueing networks and stochastic models. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, pp 1–8. IEEE
18. Gorbenko A, Karpenko A, Tarasyuk O (2020) Analysis of trade-offs in fault-tolerant distributed computing and replicated databases. In: 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), pp 1–6. IEEE
19. Nguyen TA, Kim DS, Park JS (2016) Availability modeling and analysis of a data center for disaster tolerance. Future Gener Comput Syst 56:27–50. https://doi.org/10.1016/j.future.2015.08.017
20. Hu H, Yu J, Li Z, Chen J, Hu H (2020) Modeling and analysis of cyber-physical system based on object-oriente generalized stochastic petri net. IEEE Trans Relia 70(3):1271–1285
21. Nourredine O, Menouar B, Campo E, Bossche A (2023) A new generalized stochastic petri net modeling for energy-harvesting-wireless sensor network assessment. Int J Commun Syst 36(11):5505
22. Sun X, Yu Z, Gao H, Li X (2023) Trustworthiness analysis and evaluation for command and control cyber-physical systems using generalized stochastic petri nets. Inf Sci 638:118942
23. Trivedi KS, Kim D-S, Ghosh R (2013) System availability assessment using stochastic models. Appl Stoch Models Bus Ind 29(2):94–109
24. Nong M, Huang L, Liu M (2022) Allocation of resources for cloud survivability in smart manufacturing. ACM Trans Manag Inf Syst (TMIS) 13(4):1–11
25. Ma L, Su W, Wu B, Yang B, Jiang X (2020) Early warning disaster-aware service protection in geo-distributed data centers. Comput Netw 180:107419
26. Ayoub O, De Sousa A, Mendieta S, Musumeci F, Tornatore M (2021) Online virtual machine evacuation for disaster resilience in inter-data center networks. IEEE Trans Netw Serv Manag 18(2):1990–2001
27. Colman-Meixner C, Dikbiyik F, Habib MF, Tornatore M, Chuah C-N, Mukherjee B (2014) Disaster-survivable cloud-network mapping. Photonic Netw Commun 27:141–153
28. Sun X, Lin C, Liu W, Xiao Y (2009) Survivability evaluation of distributed service using stochastic petri net. In: 2009 Fourth International Conference on Communications and Networking in China, pp 1–5. IEEE
29. Hamadah S, Aqel D (2019) A proposed virtual private cloud-based disaster recovery strategy. In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), pp 469–473. IEEE

30. Isa ISM, Musa MO, El-Gorashi TE, Elmirghani JM (2019) Energy efficient and resilient infrastructure for fog computing health monitoring applications. In: 2019 21st International Conference on Transparent Optical Networks (ICTON), pp 1–5. IEEE
31. Gandhi A, Gupta V, Harchol-Balter M, Kozuch MA (2010) Optimality analysis of energy-performance trade-off for server farm management. Perform Eval 67(11):1155–1171
32. Silva Pinheiro TF, Pereira P, Silva B, Maciel P (2023) A performance modeling framework for microservices-based cloud infrastructures. The J Supercomput 79(7):7762–7803
33. Soylu GK, Demirörs O (2023) An exploratory case study: using petri nets for modelling microservice-based systems. In: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp 254–261. IEEE
34. Fé I, Nguyen TA, Soares A, Son S, Choi E, Min D, Lee J-W, Silva FA (2023) Model-driven dependability and power consumption quantification of kubernetes based cloud-fog continuum. IEEE Access
35. Kaur S, Bawa S (2016) A review on energy aware vm placement and consolidation techniques. In: 2016 International Conference on Inventive Computation Technologies (ICICT), vol. 3, pp 1–7. IEEE
36. Sharma O, Saini H (2016) Vm consolidation for cloud data center using median based threshold approach. Proced Comput Sci 89:27–33
37. Pereira P, Melo C, Araujo J, Dantas J, Santos V, Maciel P (2022) Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. The J Supercomput, 1–28
38. Clemente D, Pereira P, Dantas J, Maciel P (2022) Availability evaluation of system service hosted in private cloud computing through hierarchical modeling process. The J Supercomput 78(7):9985–10024
39. Bendechache M, Silva I, Santos GL, Guedes LA, Svorobej S, Mario MN, Ares ME, Byrne J, Endo PT, Lynn T (2019) Analysing dependability and performance of a real-world elastic search application. In: 2019 9th Latin-American Symposium on Dependable Computing (LADC), pp 1–8. IEEE
40. Silva FA, Brito C, Araújo G, Fé I, Tyan M, Lee J-W, Nguyen TA, Maciel PRM (2022) Model-driven impact quantification of energy resource redundancy and server rejuvenation on the dependability of medical sensor networks in smart hospitals. Sensors 22(4):1595
41. Melo C, Araujo J, Dantas J, Pereira P, Maciel P (2022) A model-based approach for planning blockchain service provisioning. Computing 104(2):315–337
42. Araujo E, Pereira P, Dantas J, Maciel P (2020) Dependability impact in the smart solar power systems: An analysis of smart buildings. Energies 14(1):124
43. Tuffin B, Choudhary P, Hirel C, Trivedi K (2007) Simulation versus analytic-numeric methods: a petri net example. In: Proc. of the 2nd VALUETOOLS Conference
44. Ungsunan PD, Lin C, Wang Y, Gai Y (2009) Network processing performability evaluation on heterogeneous reliability multicore processors using srn model. In: 2009 IEEE International Symposium on Parallel & Distributed Processing, pp 1–6. IEEE
45. Maciel PRM (2023) Performance, Reliability, and Availability Evaluation of Computational Systems, Volume I: Performance and Background. CRC Press, New York
46. Sheldon FT, Greiner S, Benzinger M (2000) Specification, safety and reliability analysis using stochastic petri net models. In: Tenth International Workshop on Software Specification and Design. IWSSD-10 2000, pp 123–132. IEEE
47. Trivedi KS, Ciardo G, Malhotra M, Garg S (2005) Dependability and performability analysis using stochastic petri nets. In: 11th International Conference on Analysis and Optimization of Systems Discrete Event Systems: Sophia-Antipolis, June 15–16–17, 1994, pp 144–157. Springer
48. Jin C, Bai X, Yang C, Mao W, Xu X (2020) A review of power consumption models of servers in data centers. Appl Energy 265:114806
49. Lin W, Shi F, Wu W, Li K, Wu G, Mohammed A-A (2020) A taxonomy and survey of power models and power modeling for cloud servers. ACM Comput Surv (CSUR) 53(5):1–41
50. Tadesse SS, Malandrino F, Chiasserini C-F (2017) Energy consumption measurements in docker. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp 272–273. IEEE
51. AWS Instance types. https://aws.amazon.com/pt/ec2/instance-types/. Accessed: 2025-05-02
52. Gomes C, Tavares E, Junior MNDO, Nogueira B (2022) Cloud storage availability and performance assessment: a study based on nosql dbms. The J Supercomput 78(2):2819–2839

53. Kharchenko V, Ponochovnyi Y, Ivanchenko O, Fesenko H, Illiashenko O (2022) Combining markov and semi-markov modelling for assessing availability and cybersecurity of cloud and iot systems. Cryptography 6(3):44
54. Sebastio S, Ghosh R, Mukherjee T (2018) An availability analysis approach for deployment configurations of containers. IEEE Trans Serv Comput 14(1):16–29
55. Morabito R (2015) Power consumption of virtualization technologies: an empirical investigation. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp 522–527. IEEE
56. Maciel P, Matos R, Silva B, Figueiredo J, Oliveira D, Fé I, Maciel R, Dantas J (2017) Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), pp 50–57. IEEE
57. Melo C, Dantas J, Oliveira A, Oliveira D, Fé I, Araujo J, Matos R, Maciel P (2018) Availability models for hyper-converged cloud computing infrastructures. In: 2018 Annual IEEE International Systems Conference (SysCon), pp 1–7. IEEE
58. Gonçalves I, Rodrigues L, Silva FA, Nguyen TA, Min D, Lee J-W (2021) Surveillance system in smart cities: a dependability evaluation based on stochastic models. Electronics 10(8):876
59. Jain R (1991) The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, vol 1. Wiley, New York

Authors and Affiliations

Iure Fé[1] · Tuan Anh Nguyen[2] · Mario Di Mauro[3] · Fabio Postiglione[3] · Alex Ramos[4] · André Soares[5] · Eunmi Choi[6] · Dugki Min[7] · Jae Woo Lee[8] · Francisco Airton Silva[1]

✉ Iure Fé
iure.fe@ufpi.edu.br

✉ Tuan Anh Nguyen
anhnt2407@konkuk.ac.kr

Mario Di Mauro
mdimauro@unisa.it

Fabio Postiglione
fpostiglione@unisa.it

Alex Ramos
alex.ramos@edu.ifce.br

André Soares
andre.soares@ufpi.edu.br

Eunmi Choi
emchoi@kookmin.ac.kr

Dugki Min
dkmin@konkuk.ac.kr

Jae Woo Lee
jwlee@konkuk.ac.kr

Francisco Airton Silva
faps@ufpi.edu.br

[1] Laboratory of Applied Research to Distributed Systems (PASID), Federal University of Piauí (UFPI), Picos, Piauí 64607-670, Brasil

[2] Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul 05029, South Korea

[3] Department of Information Engineering, Electrical Engineering and Applied Mathematics (DIEM), University of Salerno, 84084 Salerno, Italy

[4] Computer Science Department, Federal Institute of Education, Science and Technology of Ceará (IFCE), Ceará, Brasil

[5] Distributed Systems and Network Computer Laboratory (DisNeL), Federal University of Piauí (UFPI), Teresina 64049-550, Brasil

[6] School of Software, College of Computer Science, Kookmin University, Seoul 02707, South Korea

[7] Department of Computer Science and Engineering, College of Engineering, Konkuk University, Seoul 05029, South Korea

[8] Department of Aerospace Information Engineering, Konkuk University, Seoul, South Korea