Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Predicting Locally Manageable Resource Failures of High Availability Clusters

Premathas Somasekaram* | Radu Calinescu

[1]Department of Computer Science, University of York, York, United Kingdom

Correspondence
*Premathas Somasekaram, Deramore Lane, York YO10 5GH, UK. Email: premathas.somasekaram@york.ac.uk

Present Address
Deramore Lane, York YO10 5GH, UK

Abstract

Critical services from domains as diverse as finance, manufacturing and healthcare are often delivered by complex enterprise applications (EAs). *High-availability clusters* (HACs) are software-managed IT infrastructures that enable these EAs to operate with minimum downtime. This paper presents a novel Bayesian decision network model to improve the failure detection capabilities of the HACs components using a comprehensive set of characteristics for the analysed component. The model then combines these characteristics to predict whether the failure of this component can be managed locally at the failed component level without propagating the failure to upper-level components and causing a complete system failure. By improving the detection capabilities and predicting locally manageable failures, the model improves the decision-making process of HACs, and has the potential to reduce the downtime and improve availability for the applications protected by HACs. The model uses the capabilities of the Bayesian decision networks, which combines Bayesian networks with the utility theory, to assign weights to different characteristics and consolidate the related variables to output the result. The model evaluation in a realistic testbed environment with three servers, an established HAC and a well-known EA shows that the model can improve the area under the Receiver Operating Characteristic (ROC) curve for prediction of locally manageable failures by up to 9.05% compared to the baseline HAC results.

KEYWORDS:
Bayesian networks, dependability, high availability, high availability clusters, reliability

1 | INTRODUCTION

Business-critical enterprise applications (EAs) hosted on cloud and on-premises in domains ranging from healthcare and finance to logistics and manufacturing require continuous availability to ensure that service delivery is undisrupted. If the availability of such applications is disrupted for even a short time, the consequences can be severe, such as significant financial loss [1,2,3,4,5]. The term *high availability* (HA) refers to how availability can be improved by providing continuous availability. The usual way to achieve HA is through software-based solutions called *high-availability clusters* (HACs) [6,7].

HACs are sophisticated autonomous solutions that can ensure continued operations of EAs even in the event of failures of single-point-of-failure (SPOF) components of EAs [6]. A SPOF component is a critical element in a system whose failure can affect the entire system. The HAC includes all components required for EA operation, both software and hardware, in the cluster

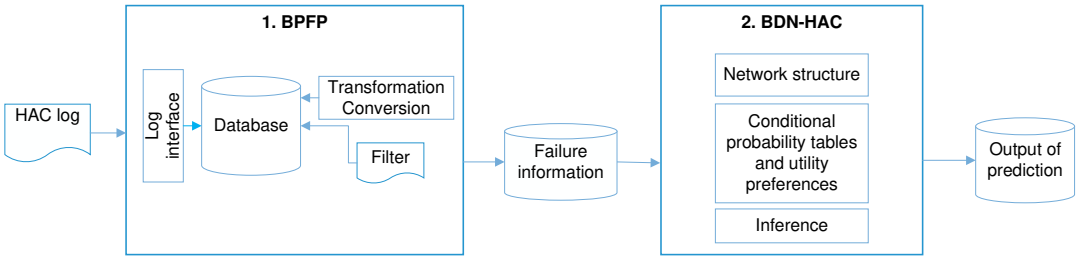


FIGURE 1 The high-level view of the Bayesian decision network model comprises two components. Bayesian prognostic framework preparation (BPFP) pre-processes HAC log data, and the Bayesian decision network (BDN-HAC) model predicts whether a resource failure is locally manageable or not.

to achieve this and ensures that the component states are continuously monitored. When the HAC detects a component failure, multiple modules of the HAC collaborate to ensure that the correct decision is taken to mitigate the failure using a threefold strategy.

1. The HAC attempts to reinitialise the failed component (e.g., by restarting) and its child components.
2. If the reinitialisation is unsuccessful, the failure is propagated to a resource group level. A resource group combines all related components to enable the failover of the entire resource group to another node (server) as one logical entity.
3. Suppose the resource group failover is unsuccessful due to its dependency on other resource groups or a critical failure at the node level. In that case, the HAC initiates a complete system failover to an available node in the HAC.

Thus, HACs can resolve failures or mask them so that end users typically do not notice any failure or only experience insignificant disturbances in most cases.

Although many HAC solutions are on the market (e.g., PowerHA SystemMirror[8] and Solaris Cluster[9]), challenges are still associated with HACs because of the lack of standardisation and restriction in deployment platforms, such as public clouds (e.g., limited availability of shared storage)[10]. Furthermore, HACs underutilise several opportunities available in modern EAs and HACs, including the following:

- The components of an EA are organised hierarchically. However, the effect on other components in the hierarchy is typically not evaluated when a component fails, even though the component failure affects all components under the failed component in the hierarchy.
- The criticality of an EA component is not considered in the decision-making process of HACs, which means the failure of a noncritical component can unnecessarily trigger a complete system failover.
- The component type (e.g., local file system or global CPU) is not considered. However, the failure of different component types can have different degrees of influence on the HAC.
- Modern EAs provide self-healing capabilities to improve availability. However, HACs do not consider these capabilities, and exploiting these can significantly reduce downtime because certain failures could be managed automatically and very efficiently by the EAs.
- The HAC typically records all system events, including failures of individual components and failovers. Still, such historical data are not exploited in the decision-making process for HACs.

We exploit these underutilised opportunities to derive the research problem, and to motivate the use of a Bayesian decision network-based approach that leverages them to improve the availability of EAs protected by HACs by enhancing their failure detection capabilities and enabling the prediction of locally manageable failures.

This paper introduces a novel *Bayesian decision network* (BDN) model that uses a set of characteristics to understand and interpret the failures of HACs using Bayesian decision reasoning capabilities. In addition to the established characteristics typically employed by HAC solutions, we add a group of new characteristics to leverage the underutilised HAC and EA capabilities

and historical data to improve the overall detection capabilities (e.g., the self-healing capability of an application) and predict whether a resource failure can be managed locally (with minimal or no disruption). Consequently, these improve the decision-making capabilities of HACs. Therefore, the model improves the first part of the threefold strategy employed by HACs to deal with individual component failures.

The solution comprises two components as presented in Figure 1: (1) the BDN for HACs (BDN-HAC model) and (2) a Bayesian prognostic framework preparation (BPFP) component. The BPFP is responsible for processing and preparing data to be used by the BDN-HAC model. This component uses a log interface to extract the HAC log data that are transformed and converted before applying a filter. The filter is essentially a functionality applied to prepare the data specifically for the BDN-HAC model. Hence, the resulting preprocessed data are used as input to the BDN-HAC model. The BDN-HAC model represents the identified characteristics as nodes in the network to perform BDN reasoning. In the model, conditional probability adds weight to parent nodes. In contrast, preferences add weight to the top-level child nodes in the utility node. After variable consolidation and dimensionality reduction, the result is a utility value that indicates whether a resource failure can be managed locally.

The solution presented in this paper represents a key part of the end-to-end 'Bayesian prognostic framework for high availability clusters' that we recently proposed in a short position paper[11]. While our preliminary work[11] provides a brief sketch of the solution, its technical details, development, operation and detailed evaluation are described for the first time in this paper. As such, the main contributions of our paper are as follows:

1. We propose a two-stage technique, where the relevant HAC characteristics and related properties are identified in the first stage, and an FMEA-based approach ranks and selects relevant characteristics in the second stage. We also introduce a method for connecting multiple properties to a target property using FMEA based on the effect of the individual properties on the target component failure.
2. We present a technique for mapping properties into variables.
3. We propose a method for calculating relative weights using FMEA when multiple components are connected to a target node.
4. We introduce an approach applying a BDN model to reduce the dimensionality using conditional probabilities and preferences. The approach also introduces the use of unobservable latent chance nodes in the BDN model.
5. We present a novel BDN model to a) improve failure detection capabilities using a comprehensive set of characteristics for the analysed component and b) predict locally manageable failures of HACs.

These contributions can promote the application of BDN models using component characteristics to improve detection capabilities and to predict locally manageable failures of HACs.

The rest of the paper is organised as follows. Section 2 provides an overview of HACs and BDNs. Section 3 presents a formalised and general approach to identifying characteristics, presenting related properties as variables, assigning weights to variables, and reducing the dimensionality before all these are transformed into the BDN model. Section 4 describes the evaluation testbed, created test cases, and related data sets. Moreover, the results from the prediction quality of the model, execution time and runtime overhead are presented. In this section, we also discuss the results, threats of validity and lessons learned. Section 5 discusses related work. Lastly, Section 6 concludes this paper and presents potential future work.

2 | BACKGROUND

2.1 | High Availability Clusters

We start by presenting two key elements of HACs. First, a *resource* represents the smallest component in the HAC (e.g., an IP address used by a database or an application service). There are dependencies between different resources, which are described by a hierarchical map[12,13,6]. Second, resources that are combined to provide a specific functionality form a *resource group*. For example, a database resource group represents all required resources to operate the database service as a single logical entity. An EA comprises multiple resource groups[7,12,13].

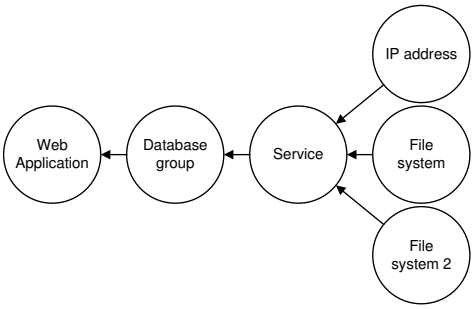


FIGURE 2 Structure of the running example application and its components.

We describe HAC failure handling using the web application from Figure 2.[1] This application, which we will use as a running example throughout the paper, comprises five resources: the *database group*, *service*, *IP*, *File System 1* and *File System 2*. The resource group *database group* is a logical resource that groups all four underlying resources. When a parent *service* with three child resources (*IP*, *File System 1*, and *File System 2*) fails, all child resources also fail. The HAC aims to resolve the failure by restarting the *service* (Strategy 1). However, because this resource depends on two child resources, they must be restarted before restarting the parent. Similarly, when a resource is stopped, all related child resources are stopped before the parent resource is stopped. If a failure is not resolved, it may lead to the failure of the associated resource group or entire application. For example, if *File System 1* fails, and the failure is not resolved, it may bring down the entire resource group. In that case, the HAC stops all resources in a specific order in the primary node, relocates the resources to a secondary node, and starts the resources in a specific order (Strategy 2). The third strategy deals with a complete system failover. In this case, all resource groups and related resources are stopped in the required order, relocated to the secondary node and started orderly.

All three HAC actions reduce the *mean time to repair* ($MTTR$)[6] significantly by avoiding time-consuming manual detection, diagnosis and recovery activities, improving availability.

2.2 | Bayesian Decision Networks

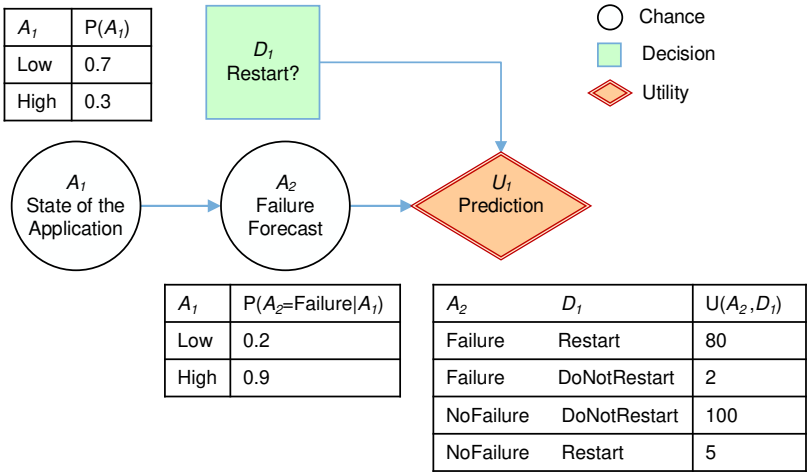
A BDN (or influence diagram) is an extension to a Bayesian network (BN) that combines BNs with decision theory, enabling decision-making under uncertainty[15]. To facilitate this, a BDN introduces two functionalities: utility and decision. A utility functionality (function) uses preferences to define the desirability of a state; hence, the utility can be described as a measure of the quantified preference of a state. A decision function describes the decision options while considering the information available at the time. To support these functionalities, BDNs introduce two new node types: utility and decision[16].

Thus, four types of nodes can exist in a BDN model: chance (or random), decision, deterministic, and utility. A chance node represents a random variable, similar to random nodes in a BN model. A decision node models the choices available for a decision, and the expectation is that such a choice is interactively selected by a decision-maker[15]. The selection of choice influences the entire network. A deterministic node represents a constant or calculated value. A random node deals with uncertainty, whereas a deterministic node deals with certainty because the outcomes are known. A utility node is based on the utility theory and represents the utility function. Thus, a utility node presents preferences associated with all possible outcomes of the parent nodes.

The concept of expected utility (EU) is used to calculate the action that can provide the utility with the most value in a wide range of decision-making problems[17]. Consider a finite set of actions A and a finite set of possible outcomes O of these actions. Suppose that $U\left(o_j \middle| a_i\right)$ represents the utility of achieving outcome $o_j \in O$, having performed action $a_i \in A$. Then, the EU of action a_i is given by the following:

$$EU\left(a_i\right) = \sum_{o_j \in O} P(o_j \left| a_i\right) \times U\left(o_j \middle| a_i\right), \tag{1}$$

[1]The model was created using a technique we developed as part of our research project, and the corresponding software is available on the GitHub repository for the project[14].



A_1	P(A_1)
Low	0.7
High	0.3

A_1	P(A_2=Failure|A_1)
Low	0.2
High	0.9

A_2	D_1	U(A_2,D_1)
Failure	Restart	80
Failure	DoNotRestart	2
NoFailure	DoNotRestart	100
NoFailure	Restart	5

FIGURE 3 Example Bayesian decision network model consists of one decision node, two chance nodes and one utility node. The corresponding conditional probability tables for chance nodes and a utility table for the utility node are shown.

where $P\left(o_j \middle| a_i\right)$ is the probability of achieving outcome o_j through executing action a_i. The action $a^* \in A$ associated with the maximum EU is calculated as follows[16]:

$$a^* = argmax_{a_i \in A} EU(a_i) = argmax_{a_i \in A} \sum_{o_j \in O} \left(P(o_j \,|a_i) \times U\left(o_j \middle| a_i\right) \right). \tag{2}$$

To illustrate the concept of a BDN, we present a simple model for a business application at a fictional company (Figure 3). The model consists of two chance nodes (A_1 and A_2), one decision node (D_1) and a utility node (U_1). The conditional probability tables (CPTs) corresponding to the chance nodes and the utility table of U_1 are also presented. The model's objective is to predict the potential failure of a business application. The application has been experiencing intermittent hardware failures, eventually leading to application failure. However, restarting the servers always helps continue to run the application. New servers have been ordered, but the expected delivery time is more than three months away due to the component shortage. The company created a BDN model to ensure that customer experience is not affected by invoking a desirable action. The feedback from customers is that an unstable system is considered more serious than a system that is briefly unavailable. Hence, the model's preference is set to reflect that if a high probability of failure exists, it should lead to the decision to restart the application.

The data for the BDN model are collected for one hour by setting up monitoring of the key components. The aggregated data are passed to the model, particularly to node, A_1. For example, if $P(A_1 = High)$, then $P(A_2 = Failure|A_1 = High)$. The EU is calculated using the sum of products of probabilities and utility (eq. (1)), and in this case, the EU outputs the preferred value ($U_1 = 80$) for the decision to restart as the prediction, suggesting that an immediate restart can satisfy customers.

3 | A BAYESIAN DECISION NETWORK FOR PREDICTING LOCALLY MANAGEABLE RESOURCE FAILURES

Understanding how HACs behave upon failure is required to predict whether a resource-level failure is locally manageable. To capture that information, the proposed model uses a set of characteristics (or properties) that comprise (i) established characteristics extracted from the research literature and current practice and (ii) new characteristics identified by this project. The properties associated with these characteristics include Boolean-value properties indicating whether certain failure recovery mechanisms are present and integer-value properties specifying the number of times that such a mechanism was activated within a given time window. As such, we organise these characteristics into four groups based on their objectives.

1. The objective of the first group is to understand and interpret the runtime behaviour of HACs for a specific resource failure. As the BDN-HAC model operates as a standalone solution, it must consider the behaviour of the HAC upon failure. Hence, the characteristics indicate whether the HAC failure management modules can automatically reinitialise a resource.

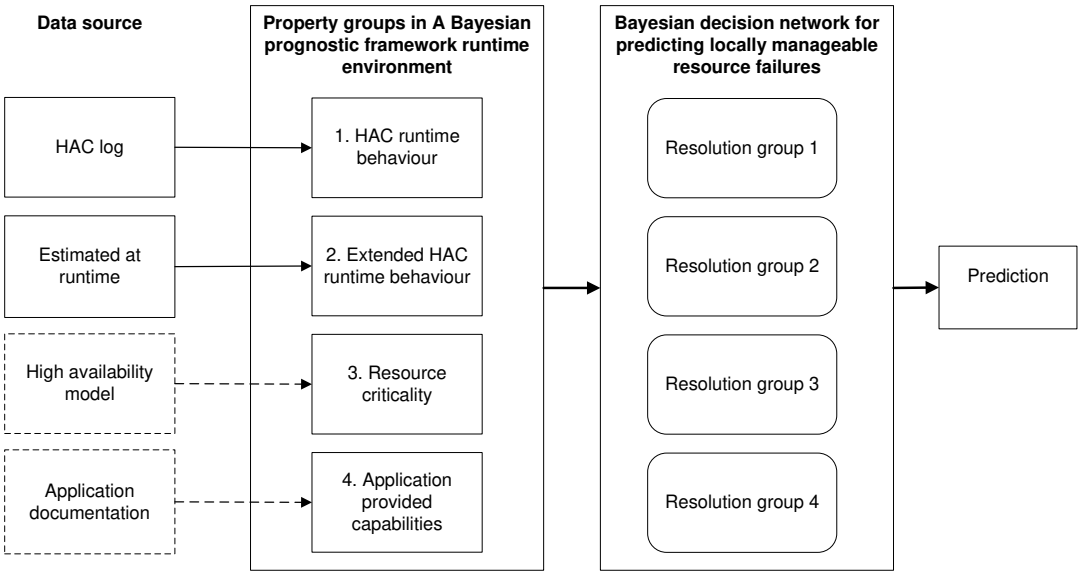


FIGURE 4 Overview of locally manageable resource failure prediction. Dashed boxes and arrows indicate that the data are already available in the runtime environment and are retrieved from sources in the design phase.

2. The second group improves the failure detection by extending the detection scope to include additional characteristics (e.g., the position of a resource in a hierarchy and the resource types). The objective is to improve the detection capabilities and detect failure behaviour at a low level of granularity.

3. The objective of the third group is to assess the criticality of a resource. If a resource is identified as noncritical, the implication could be that the failure does not affect the operation of the EA or lead to the failure of any other interconnected resources. Hence, such a failure can be masked, and the outcome could be treated as a 'manageable' failure.

4. The fourth group is associated with failures resolved by events triggered outside the HAC control (e.g., failures managed by the protected EAs as part of the application-provided self-healing capabilities[18]). An example of this is that if a resource, process or service that has a self-healing capability provided by the application fails, the application initiates the self-healing action (e.g., a restart of the resource) to resolve the problem.

Groups 2, 3 and 4 are used to improve the detection capabilities of the BDN-HAC model. Further, by combining all groups and assessing the relationships between the characteristics, the model predicts whether the failure can be managed locally or not. Hence, the proposed model can improve the decision-making capabilities of HACs. Out of the four groups, current HACs only consider the first. The characteristics of the second group are usually not considered by HACs, but the values can be obtained from HACs at runtime. Groups 3 and 4 comprise new characteristics; therefore, Groups 2, 3 and 4 are introduced in this paper.

As illustrated in Figure 4, the information associated with the four groups of characteristics used by the BDN-HAC model is obtained from multiple sources. First, the data structures required to support the four property groups are set up in the BPFP component during the design phase. These structures are populated with static values obtained from multiple sources (e.g., HA model for the HAC, HAC logs and application documentation). When a HAC resource fails at runtime, the failure data are captured from HAC logs and processed by the BPFP component. Some property values come from the HAC log, whereas others come from the static information available in the runtime environment. Additional property values (e.g., the frequency of resource failures) are calculated at runtime. Therefore, data for all four property groups are prepared and included in the runtime environment of the BPFP component (Figure 4).

Values associated with the four property groups are linked to four resolution groups in the BDN-HAC model. These resolution groups and the relationships between the characteristics are encoded in the BDN-HAC model using conditional probabilities and utility preferences. Therefore, the BDN-HAC model assesses characteristics in all four groups, and the model output is a binary value that indicates whether resource failure can be managed locally.

To work with the characteristics from the four groups in the BDN-HAC model, we introduce techniques that (i) capture the characteristics as variables, (ii) assign relative weights to the variables, and (iii) use these weights to reduce the number of variables (*dimensionality reduction*) in this section.

3.1 | HAC Characteristics for Predicting Locally Manageable Resource Failures

We used a two-stage systematic process to identify HAC characteristics and related properties relevant to detecting and resolving a resource failure and improving detection and decision capabilities.

Stage 1. We first analysed and identified characteristics using research studies, technical manuals, and HAC and EA documentation, HAC logs, as well as the taxonomy and survey that we developed in our research project [10]. We organised the identified characteristics into two categories. The first category consists of key HAC characteristics already known to affect resource failure analysis (established characteristics). For example, the characteristic to enable reinitialisation by the HAC assumes that the HAC understands how the resource works to proceed with the correct initialisation (e.g., remounting a file system). The second group consists of characteristics mentioned in the literature but not related to the HAC resolution of resource failures. Therefore, we deem these to be new characteristics when used alongside HACs for the first time in this paper. These new characteristics can significantly improve detection and resolution capabilities. For example, some EAs have in-built fault-tolerance capabilities to resolve resource failures using self-healing or rejuvenation (e.g., restarting a service or a process) [19]. In such a case, the first attempt to resolve the failure is managed by the application. Therefore, using this characteristic improves failure management. The two categories are defined as follows:

- **Established characteristics (ECs)** are a set of established HAC characteristics that influence how the HAC behaves upon a resource failure.
- **New characteristics (NCs)** are a set of new characteristics that extend the EC category to improve detection and resolution capabilities by capturing more details to perform accurate detection upon a resource failure.

Stage 2. We performed the failure mode and effects analysis (FMEA), a systematic analysis technique to identify potential failures and their effects [20,21]. As FMEA is well suited for systems comprising multiple interconnected components, it can capture the characteristics that significantly influence a resource upon failure. The FMEA was performed in two steps:

(i) each characteristic was treated as a component, and the different states of a characteristic were considered potential failure modes. Characteristics with a high high risk priority number (RPN) to rank system failures were selected. Further, we narrowed the list of characteristics by ranking them using RPNs and their applicability. Applicability implies whether it is possible to obtain information related to a characteristic given the conditions of the testbed environment (Section 4). We identified nine critical properties (four ECs and five NCs) out of the six ECs and 10 NCs initially considered.

(ii) We used three target components to connect to the nine characteristics [22]. For example, error-related characteristics were connected to an error-related target component. Thus, we used these three target components as component failures and the nine characteristics as failure modes to identify the effect of the individual characteristics on target component failures. This approach helped identify the cause-and-effect relationship between the characteristics and target components. The target components are described further as derived or target variables in the next section.

The first part of Table 1 lists the nine retained properties used as inputs for the BDN-HAC model and the descriptions and categories for each. The values column specifies their value ranges, and the resolution group (RG) column lists the related resolution groups. The motivation for the selection of each property and its role in our BDN-HAC model are detailed below.

Failure repetition

Motivation. When the HAC reinitialises a failed resource, it uses a local timeout value for the reinitialisation to complete [23,24,25]. If the reinitialisation does not complete successfully within the time limit imposed by this timeout, the HAC retries the reinitialisation procedure (up to several times). If no reinitialisation attempt succeeds, the HAC eventually reclassifies the resource as not reinitialisable. Therefore, the number of such attempts performed within the last n minutes is an important indicator of how likely the resource will incur a more severe failure. The value for n is estimated from the number of failures allowed in the HAC for a resource type and the resource start time. However, the resource types have different values for the start time, and the number of allowed failures per node could be different per resource. Therefore, an average value is calculated from multiple resource types.

TABLE 1 Transition from properties to variables are shown in the two parts of the table. 1. Properties before mapping: resource properties of high-availability clusters grouped by sets. The related values column lists the potential value interval associated with each property, and the category represents whether the property belongs to established characteristics (EC) or new characteristics (NC) introduced in this paper. The resource group indicator presents the four property groups (Figure 4). 2. Variables after mapping: the values column lists the values associated with the variables after mapping and the related mapping group. The last three rows present the variables derived (target) from other variables

1. Properties before mapping					2. Variables after mapping	
HAC Resource Property	**Description**	**Category**	**Values**	**RG**	**Values**	**Group**
Error-related property set E						
Failure repetition (fr)	Number of failures that occurred in the last $n > 0$ min	EC	{0,...,n}	1	{low, high}	1
Redundancy factor (rf)	Application provides in-built self-healing capabilities	NC	{0,1}	4	{true, false}	2
Aggregated failure count (afc)	Distinct failures of the resource within the last $m > 0$ h	EC	{0,...,n}	1	{low, high}	1
Reinitialisation factor (rc)	Resource reinitialisation possible	EC	{1,2}	1	{true, false}	2
Dependency-related properties set D						
Dependency type (dt)	Type of resource	NC	{1,2,3}	2	{local, shared, global}	3
Dependency levels down (dld)	Number of lower-level resources	NC	{0,...,n}	2	{low, high}	1
Dependency levels up (dlu)	Number of upper-level resources	NC	{0,...,n}	2	{low, high}	1
Criticality-related property set C						
Critical factor (cf)	Indicates the criticality of a resource	NC	{0,1}	3	{true, false}	2
Current status-related property set S						
Current state (cs)	Current status of a resource	EC	{on, off}	1	{online, offline}	3
Error rating (e)	n/a	n/a	n/a	n/a	{failure, no_failure}	4
Dependency factor (d)	n/a	n/a	n/a	n/a	{low, high}	4
Resource state (r)	n/a	n/a	n/a	n/a	{failure, no_failure}	4

Notes: On: online, Off: offline, RG: resolution group, EC: established characteristic, NC: new characteristic, n/a: not applicable

Role. The model considers the number of failures while assessing other properties (e.g., *aggregated failure count* or *reinitialisation factor*) to determine whether the likelihood of unmanageable failure increases.

Redundancy factor

Motivation. An application that the HAC protects may have built-in self-healing capabilities (e.g., software rejuvenation[26,19]) for key resources, enabling the application to automatically initiate the first mitigation action upon a resource failure[27,28,18]. The HACs must recognise these features to avoid initiating any mitigating actions that could conflict with the application actions. However, HACs do not use this property.

Role. This property indicates how the application-provided self-healing capabilities can be used to reinitialise a resource, increasing the likelihood of managing the resource failure locally.

Aggregated failure count

Motivation. The number of failures of a resource during a period is aggregated to indicate a potential persistent failure. It can also show that a global threshold value for the timeout to manage resource failures is reached[23,25,29]. The implication is that the HAC classifies the resource as more error-prone in a specific node and may ban the resource from starting in that node. A high number of resource failures within the last n hours indicates a potential persistent failure. The aggregated failure count is calculated using the global timeout value[23,25,29], number of failures allowed in a node for a resource type, and average start time per resource type. When a resource cannot be started on a node after exceeding the number of allowed starts on the node, this may be because (1) policies are set automatically by the HAC to prevent the resource from starting on the node, (2) policies are set automatically by the HAC to prevent the resource from starting on other nodes, or (3) policies are set not to allow the resource to start on any node. An example of the third case is file system corruption. Suppose the corruption is on the block level. In that case, it affects shared storage or replication, displaying the same failure in all nodes, which sets a policy to prevent the resource from being brought up in any node, affecting all related resources.

Role. This property identifies a persistent failure pattern and whether the mitigation actions have been successful. A high value indicates a more persistent failure and/or that the mitigation actions have been successful, and the HAC may have set one of the

three policies to prevent the resource from starting. A low value indicates a low probability of failure, and when combined with other positive outcomes, the result may indicate a high likelihood of managing the failure locally.

Reinitialisation factor

Motivation. This refers to the ability of the HAC to reinitialise a resource[23,7].[2] The reinitialisation procedures are different for various resources. For example, if the resource type is a service, the procedure is to restart the service. If the resource type is a file system, the mitigation action is to remount the file system. There are multiple steps associated with these procedures, such as checking the resource status and shutting it down before restarting.

Role. This property evaluates whether the HAC can reinitialise a resource. If the property is set to true, the probability of managing failure for the resource increases significantly.

Dependency type

Motivation. There are three types of resource dependencies (local, shared and global), and they have different impact factors[7] when the related resources fail. Hence, each resource is assessed based on the impact factor. A local dependency type can only affect other resources in the same group, whereas a shared resource may affect one or more related resource groups. However, a global resource will likely affect all resource groups and the entire system.

Role. This property evaluates the impact factor associated with each resource type and combines the outcome with the results of evaluating other properties to enable accurate predictions.

Dependency levels down

Motivation. The HAC resources have a hierarchical organisation[6,7], which means the start and stop procedures follow a particular sequence to start or stop all related resources. For example, when a resource fails, an attempt to reinitialise the resource is started, including resources at the lower hierarchical level. If the number of such lower-level resources is high, it may affect the overall start or stop time, which can decrease the likelihood of managing the failure of any of these resources.

Role. This considers the effect of losing lower-level resources when a resource fails or when a resource is reinitialised. A lower value indicates an effect on fewer resources, and the evaluation can be combined with other properties, such as the *critical factor* and *reinitialisation factor*.

Dependency levels up

Motivation. Similar to dependency level down, a high number of upper-level resources decreases the likelihood of managing a failure[6,7]. When a low-level resource fails, it may affect all related upper-level resources within the hierarchy. For example, if a low-level resource "disk" crashes, it may terminate all processes using that disk, causing those resources to fail. These failures can cause the failures of other resources to adhere to the start and stop dependencies. If a sufficient number of resources fail, the situation can be interpreted by the HAC as critical, initiating a failover for the resource group or the entire system.

Role. This property assesses the effect on upper-level resources. A higher value results in a low likelihood of managing failures.

Critical factor

Motivation. If a resource is critical, the probability of causing a resource group failure or system failure is estimated to be high[30]. The objective is to evaluate whether such a resource has an immediate effect on the system operations. For example, if a system can survive without a particular resource for a brief period, it can be rendered noncritical. The failure of such a resource does not need to be propagated to other resources; hence, there is no effect at the resource group or system level.

Role. This considers the critical factor of a resource. If a resource is not critical, the likelihood of the propagated failure is reduced significantly.

Current state

Motivation. This property captures the current status of a resource[30]. If the status is offline, and the property failure repetition is high, which may indicate that the failure resolution was unsuccessful. If the resource is online after recording a momentary failure, it may indicate that the procedures associated with either the reinitialisation factor or redundancy factor may have resolved the problem.

Role. The model considers the current state of a resource. For example, when the state is online, it significantly increases the likelihood of managing failures.

We validated these characteristics using the testbed application (Section 4.1), as discussed later in Section 4.

[2]The ability to reinitialise failed resources or failed components of a resource automatically by the HAC.

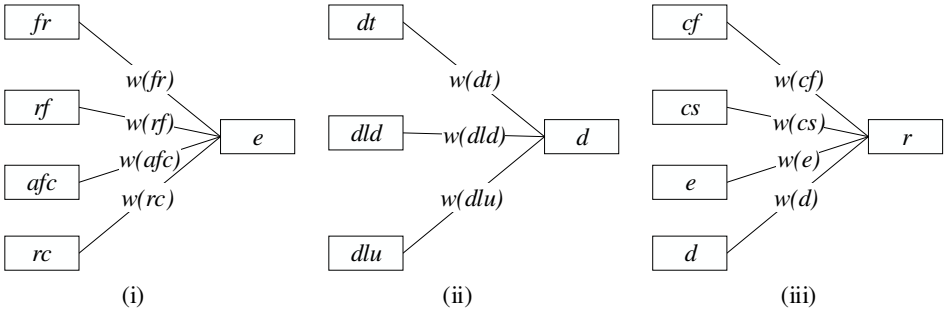


FIGURE 5 Reducing dimensionality and assigning relative weights to variables in (i) the error-related property set E and (ii) dependency-related property set D and (iii) combining the outcomes of (i) and (ii) with the criticality-related property set C and current status-related property set S.

3.2 | General Variable and State Definitions

Before we can use the HAC resource properties from the first part of Table 1 with the BDN model introduced in this section, they must be mapped to basic variables.[3] This mapping is described in the second part of Table 1, which presents the values and groups for these variables.

Four groups of variables are identified based on the mapping changes they must undergo for inclusion in the model (the second part of Table 1). Group 1 variables require categorising the value of their corresponding property as either low when this value is not larger than the threshold set using transformation conditions or high when this value exceeds the threshold. The transformation conditions for Group 1 variables, *dependency levels down* and *dependency levels up*, are defined using the FMEA to indicate a potential effect on related resources (upper or lower) in the hierarchical representation of the HAC upon the resource failure[31]. The transformation conditions for the other two Group 1 variables, *failure repetition* and *aggregated failure count*, are calculated from local and global timeout values, respectively. Group 2 variables are converted from integer to Boolean, and the variables in the scope are *redundancy factor*, *reinitialisation factor*, and *critical factor*. The third group of variables, *current state* and *dependency type*, does not change but is transferred directly to the model. The fourth group represents the target or *derived variables* (i.e., variables obtained from the set properties in the first part of Table 1). The variables in this group are *error rating*, *dependency factor* and *resource state*. These are described as three target components in Section 3.1.

All variables are binary except for the *dependency type*, which is multivalued to represent the three types of dependency. The decision to use such binary variables comes from our FMEA performed in Section 3.1 (step i), where possible states of each property were included to investigate the different states of resources, potential failures, and effects[20,21].

3.3 | Relative Weight Assignment and Dimensionality Reduction

The variables from Table 1 have different effects on the outcome of the BDN-HAC model. Hence, a weighting factor is encoded in the model to reflect these levels of effect. For example, the variable *reinitialisation factor* (rf) is assigned more weight than *dependency levels down* (dld) because, if the *reinitialisation factor* is set to true, the implication is that the HAC can reinitialise the resource. If that activity succeeds, the resource is no longer considered a failed resource; hence, the BDN-HAC model does not interpret the failure as a failure that must be managed locally. Thus, the *reinitialisation factor* is more important than *dependency levels down*, and more weight is associated with it. In addition, the number of variables must also be reduced (dimensionality reduction) for the BDN-HAC model to process and compute an outcome (prediction). This section presents our approach for applying relative weights to each property to reduce the dimensionality.

Dimensionality reduction and weight assignment are performed in two steps. First, the variables are grouped based on the defined sets in the first part of Table 1. There is a causal relationship between the variables within a set. For example, when the value of a variable changes, this also influences other variables in the same set, as described in step ii of the FMEA (Section 3.1). The relative weights within a set are used to derive a target variable reflecting the effects of the variables from that set. This target variable represents all variables in the set, effectively reducing the dimensionality. Two main variable sets are defined: the

[3]Properties have multiple values, and mapping them to variables allows them to be processed (e.g., by transforming them into categorical variables).

error rating (E) and *dependency factor* (D). The former comprises all variables related to errors in a resource, whereas the latter comprises dependency-related variables. Figure 5 (i) displays the variable set *error rating* (E) and the four variables that are part of this set. The four variables undergo dimensionality reduction where they are consolidated into the target variable e with a relative weight $w(v)$ assigned to each variable $v \in E$. Similarly, Figure 5 (ii) presents the set *dependency factor* (D), which has three variables, and its dimensionality is reduced to one target variable d by assigning relative weights to its elements.

In the second step (Figure 5 (iii)), the dimensionality is reduced further. To this end, both target variables e and d derived from the sets E and D, respectively, and the two variables from sets C and S from Table 1 are consolidated into a target variable r. Thus, 11 variables are reduced to one variable. The value of the target variable r indicates whether the resource failure in question can be managed.

The relative weights were calculated from the RPNs obtained using FMEA (Section 3.1) as follows

$$W_i = \frac{w_i}{\sum_{i=1}^{n} w_i}, \tag{3}$$

where w_i represents the value obtained from the i-th RPN, and n denotes the total number of components connected to a target variable.

In the remainder of this section, we demonstrate the two steps used to perform dimensionality reduction and weight assignment.

All variables are assumed to be random, and therefore a probabilistic approach when adding weights for a given state of a child (target) variable is formalised.

For any combination of values fr_0, rf_0, afc_0 and rc_0 of the four input variables from Figure 5 (i), the conditional probability that error e has a specific value e_0 is given by the following:

$$P\left(e = e_0 \mid fr = fr_0, rf = rf_0, afc = afc_0, rc = rc_0\right) = \sum_{v \in E} \left[w(v) P\left(e = e_0 \mid v = v_0\right)\right], \tag{4}$$

where $w(v) \in (0, 1]$ is the weight associated with the variable $v \in E$ such that $\sum_{v \in E} w(v) = 1$.

Similarly, for any combination of values dt_0, dtd_0 and dlu_0 of the three input variables from Figure 5 (ii), the conditional probability that the dependency d has a specific value d_0 is given as follows:

$$P\left(d = d_0 \mid dt = dt_0, dld = dld_0, dlu = dlu_0\right) = \sum_{v \in D} \left[w(v) P\left(d = d_0 \mid v = v_0\right)\right], \tag{5}$$

where $w(v) \in (0, 1]$ is the weight associated with the variable $v \in D$ such that $\sum_{v \in D} w(v) = 1$.

In the second step, the outcomes of Equations (4) and (5) are combined with the two variables from the property sets C and S in Table 1. For any combination of values cf_0, cs_0, e_0 and d_0 of the input variables from Figure 5 (iii), the conditional probability that the resource state r has a specific value r_0 is given by the following:

$$P\left(r = r_0 \mid cf = cf_0, cs = cs_0, e = e_0, d = d_0\right) = \sum_{v \in R} \left[w(v) P\left(r = r_0 \mid v = v_0\right)\right], \tag{6}$$

where $R = \{cf, cs, e, d\}$, and $w(v) \in (0, 1]$ is the weight associated with the variable $v \in R$ such that $\sum_{v \in R} w(v) = 1$. This equation gives the likelihood of resource failure after the dimensionality reduction.

Our BDN-HAC model uses different variants of eqs. (4)–(6). Weight assignment and dimensionality reduction are performed as part of constructing the model and defining the CPT as described next.

3.4 | BDN-HAC

In this section, we present the BDN-HAC model. The design rationale is to represent the identified characteristics in a probabilistic model while ensuring that the criticality of each characteristic is captured through weight assignment. The characteristics must also be consolidated at multiple levels for the BDN model to output a prediction value. For example, the related characteristics are grouped and eventually represented by a latent (target) variable, and all such variables are further consolidated into the main utility variable to present a final prediction value. Therefore, such a model should also represent target variables as latent nodes.

We explored several alternatives, such as the principal component analysis (PCA) and Bayesian networks[32,33,34,35] and a variation of Bayesian decision networks using the additive-linear utility (ALU)[36]. An ALU allows connecting multiple parent utility nodes to a single ALU node, summing the outcomes linearly into the ALU node. The ALU-based BDN model is the only comparable approach because, to the best of our knowledge, other alternatives and approaches do not support incorporating

TABLE 2 Description of all nodes in the BDN-HAC model

Node ID	Node Name	Node Type	State	Relative Weights
A_1	Error rating	Chance†	{failure, no_failure}	{≈ 0.15}
A_2	Failure repetition	Chance	{low, high}	{≈ 0.1}
A_3	Redundancy factor	Chance	{true, false}	{≈ 0.4}
A_4	Aggregated failure count	Chance	{low, high}	{≈ 0.1}
A_5	Reinitialisation factor	Chance	{true, false}	{≈ 0.4}
B_1	Dependency factor	Chance†	{low, high}	{≈ 0.15}
B_2	Dependency type	Chance	{local, shared, global}	{≈ 0.3}
B_3	Dependency levels down	Chance	{low, high}	{≈ 0.4}
B_4	Dependency levels up	Chance	{low, high}	{≈ 0.3}
C_1	Critical factor	Chance	{true, false}	{≈ 0.35}
D_1	Current state	Decision	{online, offline}	{≈ 0.35}
U_1	Resource state	Utility	$\{-100 < U_1 < 100\}$	Target

†Indicates a latent node representing a target variable

relative weights (calculated using FMEA), dimensionality reduction in two steps, and using these steps to predict an outcome. Therefore, we experimented with the BDN-HAC model presented in this paper and the BDN model based on ALU[37]. The experimental results demonstrate that the ALU model tends to be more deterministic and limits the available options (e.g., the flexibility to define conditional probabilities and to work with more parameters)[37]. Therefore, we opted to use only the BDN-HAC model presented in this section. The model uses properties/variables from Section 3.2, but the types are changed to work with a BDN model.

3.4.1 | Variable and State Definitions

The BDN-HAC model includes one utility node and one decision node. Table 2 lists the node IDs, nodes representing variables (node name), node type, state and relative weights associated with each node. The *node names* are the same as in Section 3.1. The *node type* refers to the type of node in the model and demonstrates how the variable types in Section 3.2 are mapped onto the node types in the model. The column *state* describes the states associated with each node, and the column *relative weights* lists the relative weights associated with each node. The relative weights are obtained from the second step of the FMEA and by using the corresponding RPNs (Section 3.1). For example, RPN for a single variable/property is divided by the sum of the RPNs in the related set to obtain the relative weight for the variable. The model mainly uses the variables defined in Table 1. However, two variables are changed to reflect the BDN nature of the model. Node D_1 (variable *current state*) is changed to a decision node with two states, *online* and *offline*, whereas node U_1 represents the variable *resource state*. Node U_1 is a utility node responsible for assembling the outcome; hence, it does not follow a probabilistic approach but instead uses a utility function. The node uses a scale between -100 and 100, where 0 is the boundary to interpret the outcome. The output of the utility node is continuous; thus, it is defined as a continuous node, whereas all other nodes are created as discrete.

The relative weights were obtained using eq. (3). First, we calculated the weights for each variable connected to a target variable, and an example is provided for the target variable A_1 and connected variable A_2 below. Assuming that the variable A_2 has the RPN of 100, we obtain the following

$$A_2 = \frac{100}{1000} = 0.1, \tag{7}$$

where 100 represents the RPN obtained from the FMEA, and 1000 denotes the sum of all RPNs for the connected variables A_2, A_3, A_4 and A_5.

Second, we calculated the relative weights of the nodes A_2, D_1, B_1 and C_1 using eq. (3), and the resulting relative weights were added to the utility table of U_1. Further, if the outcome of a target node is negative, a negative weight is added, and when the outcome of a target variable is positive, a positive weight is used, which can be expressed as follows

$$W_i = \begin{cases} -w_i, & \text{if } c_i = \text{negative outcome} \\ w_i, & \text{otherwise,} \end{cases} \tag{8}$$

where c_i represents a negative outcome in a child node, such as that obtained for A_1 = failure, B_1 = low, C_1 = true and D_1 = offline.

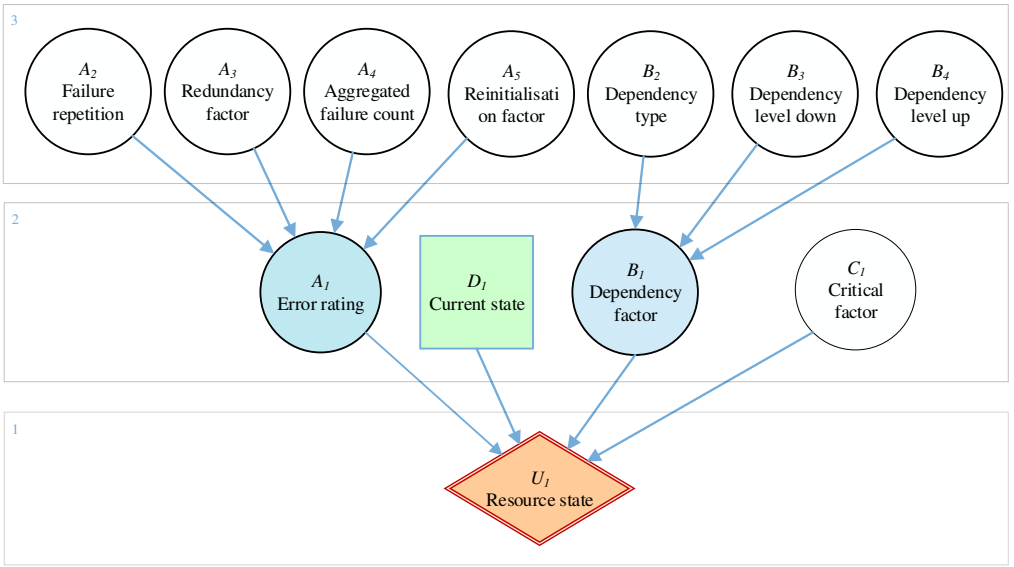


FIGURE 6 Bayesian decision network model: BDN-HAC. Random nodes are depicted with a white background. Blue shading indicates latent nodes. Green shading indicates the decision node, and the utility node is shaded orange.

The calculation of utility values using both weights and conditional probabilities is described in Section 3.4.2. We validated the BDN models and the corresponding numbers using two example applications constructed from HAC log files obtained from a company. Further validations were done using the running example (Section 2.1) and the testbed application (Section 4.1).

3.4.2 | Transformation into the Bayesian Decision Network

As illustrated in Figure 6, the BDN-HAC model is organised into three layers and 12 nodes. The third layer consists of nodes that represent most of the HAC properties. The causality is maintained by ensuring that the nodes belonging to the same set build a causal relationship. Subsequently, conditional probabilities are used to quantify causal relationships.

Two nodes (A_1 and B_1) in the second layer are responsible for reducing the parent node dimensionality from the third layer and adding relative weights. This method is possible due to conditional probabilities in the two child nodes. These two nodes are *latent nodes* in the BDN; hence, they are unobservable. In addition, two nodes (D_1 and C_1) in the second layer capture the crucial properties of HACs, which can significantly influence the overall outcome. All four nodes in the second layer converge in the first layer.

The only node in the first layer is a utility node (U_1) responsible for assigning weights and further reducing dimensionality by associating each parent node with a preference. Both chance nodes and the decision node in the second layer influence the utility node, whereas the nodes in the third layer indirectly influence the utility node. Hence, D_1 has a significant influence on U_1 because U_1 aims to maximise the EU of the decisions by D_1. This relationship is represented by a functional edge from D_1 to U_1, indicating functional dependency, which is established between other nodes and a utility node. The decisions made by D_1 also influence the rest of the network. However, the model considers all nodes, which means that the decision from D_1 may not reflect the actual outcome. For example, an unfavourable decision by D_1 may not lead to a negative outcome by the model.

Following the two-step approach defined in Section 3.3, and using Equations (4), (5) and (6), we use the following equations to represent the model. First, given any combination of values $A_{2,0}$, $A_{3,0}$, $A_{4,0}$ and $A_{5,0}$ for nodes A_2 to A_5, the conditional probability that the *error rating* node A_1 has a specific value $A_{1,0}$ is given as follows:

$$P\left(A_1 = A_{1,0} \mid A_2 = A_{2,0}, A_3 = A_{3,0}, A_4 = A_{4,0}, A_5 = A_{5,0}\right) = \sum_{v \in \{A_2, A_3, A_4, A_5\}} \left[w(v) P\left(A_1 = A_{1,0} \mid v = v_0\right)\right], \tag{9}$$

where $w(v) \in (0, 1]$ is a weight associated with $v \in \{A_2, A_3, A_4, A_5\}$ such that $\sum_{v \in \{A_2, A_3, A_4, A_5\}} w(v) = 1$. Similarly, given any combination of values $B_{2,0}$, $B_{3,0}$ and $B_{4,0}$ for nodes B_2, B_3 and B_4, respectively, the conditional probability that the *dependency*

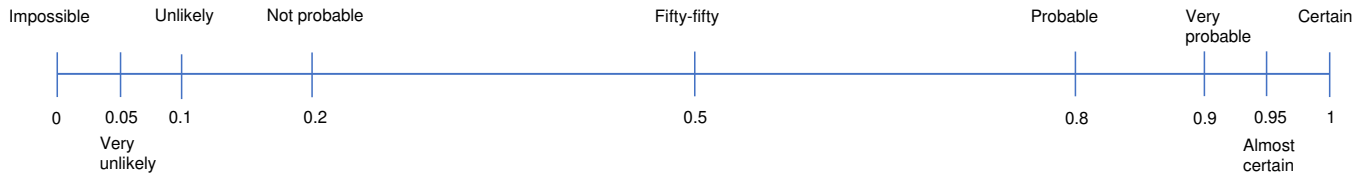


FIGURE 7 Probability scale showing quantitative probability numbers and corresponding statements.

factor node B_1 has a specific value $B_{1,0}$ is given as follows:

$$P\left(B_1 = B_{1,0} \mid B_2 = B_{2,0}, B_3 = B_{3,0}, B_4 = B_{4,0}\right) = \sum_{v \in \{B_2, B_3, B_4\}} \left[w(v) P\left(B_1 = B_{1,0} \mid v = v_0\right)\right], \tag{10}$$

where $w(v) \in (0, 1]$ is a weight associated with $v \in \{B_2, B_3, B_4\}$ such that $\sum_{v \in \{B_2, B_3, B_4\}} w(v)\} = 1$.

In the second step, the outcomes of Equations (9) and (10) are combined with nodes C_1 and D_1. For any combination of values $A_{1,0}$, $B_{1,0}$, $C_{1,0}$ and $D_{1,0}$ of nodes A_1, B_1, C_1 and D_1, respectively, the preference that the utility node *resource state* $U_1 \in [-100, 100]$ has a specific value $U_{1,0}$ is given by the following:

$$U\left(U_1 = U_{1,0} \mid A_1 = A_{1,0}, B_1 = B_{1,0}, C_1 = C_{1,0}, D_1 = D_{1,0}\right) = \sum_{v \in \{A_1, B_1, C_1, D_1\}} \left[w(v) U\left(U_1 = U_{1,0} \mid v = v_0\right)\right], \tag{11}$$

where $w(v) \in (0, 1]$ is a weight associated with $v \in \{A_1, B_1, C_1, D_1\}$ such that $\sum_{v \in \{A_1, B_1, C_1, D_1\}} w(v) = 1$. The outcome of the utility node U_1 (and, thus, of the proposed BDN-HAC model) is interpreted as:

$$P(locally\_manageable\_resource\_failure) = \begin{cases} low, & \text{if } U_1 < 0 \\ high, & \text{otherwise} \end{cases} \tag{12}$$

Hence, a utility $U_1 = 0$ functions as the cutoff value for BDN-HAC decision-making.

3.4.3 | Conditional Probability Tables

All or most nonutility and nonlatent nodes must be instantiated to predict the outcomes as accurately as possible. When nodes are instantiated, only the posterior distributions are evaluated. However, if a situation with incomplete data arises, the prior distribution is also evaluated. Hence, the model's objective is to instantiate as many nodes as possible to improve the prediction quality.

This section describes the conditional probabilities associated with each node and the justifications for setting their prior distributions. The probability distributions were estimated using a method of elicitation[38,16]. The method uses a numerical probability scale[39] as shown in Figure 7, and it represents both quantitative probability numbers and corresponding statements. The elicitation was performed at a single probability level[16], and as input, we used extensive literature reviews and expert knowledge developed in this research. However, it was challenging to obtain statistics concerning HAC failures, particularly failures at the resource level. Therefore, we used studies of several non-HAC systems and configurations (e.g., high-performance computing - HPC[40,41]) to estimate the probability distributions.

Table 3 lists the probability distributions obtained using the above method for all relevant nodes in the BDN-HAC model. While chance nodes have CPTs, there are no probability distributions associated with the utility (U_1) and decision (D_1) nodes. The decision table D_1 lists two possible states, *online* and *offline*. The utility table U_1 represents weight assignments and dimensionality reductions as a numerical measure of preferences over the entire network. Hence, there are 10 CPTs, one decision table, and one utility table associated with the BDN-HAC model. The CPTs for all nodes in the third layer and the second-layer node, C_1, have local distributions. The CPTs for A_1 and B_1 are specified as conditional probabilities over their parent nodes. The weight assignments and dimensionality reductions are encoded in these conditional probabilities and presented in the CPTs as described in Section 3.4.1. The distribution of A_2 reflects the fact that the probability of repeated failures is relatively low since the HAC has the responsibility of mitigating the failures promptly, and a failover is triggered otherwise[29]. Thus, the probability distributions are assigned to $P(low) = .75$ and $P(high = .25)$. The distribution of A_3 represents the built-in redundancy factor provided by the applications. An EA may include self-healing capabilities on a resource level, such as quickly restarting a crucial process[42,2,43]. However, such EAs may only deal with issues associated with specific components (e.g., processes); and these capabilities cannot be extended to include a platform or infrastructure-related components because the applications do not

TABLE 3 Probability distributions for all nodes in the BDN-HAC model and their states

Node	Probability Distribution	Node	Probability Distribution
A_1	$P(A_1 \mid A_2, A_3, A_4, A_5)$	B_2	local = .8 shared = .15 global = .05
A_2	low = .75 high = .25	B_3	low = .5 high = .5
A_3	true = .3 false = .7	B_4	low = .3 high = .7
A_4	low = .9 high = .1	C_1	true = .8 false = .2
A_5	true = .75 false = .25	D_1	online offline
B_1	$P(B1 \mid B2, B3, B4)$	U_1	utility node

manage them. When applications provide these capabilities, the HAC must still evaluate the effect on other linked resources. Thus, the probability distributions are set to $P(true) = .3$ and $P(false = .7)$ considering these factors. The distribution for A_4 is set to $P(low) = .9$ and $P(high = .1)$ to reflect the fact that the HAC can mitigate repeated errors in most cases[31,2]. The value for A_5 is derived from the fact that most HACs provide a mechanism to reinitialise a resource, and it is set to $P(true) = .75$ and $P(false = .25)$. Hence, A_5 is considered a typical property of HACs. However, some resources cannot be reinitialised by HACs, e.g. errors related to the CPU, memory and operating system[44].

In addition, B_2 captures the type of dependency on a resource[31,29]. The assumption is that the failure of a resource with local dependency is restricted to the resource group. In contrast, the failure of a shared resource may affect multiple resource groups. Similarly, a global dependency can have an effect at the system level. Therefore, the distribution is set to $P(local) = .8$, $P(shared) = .15$ and $P(global) = .05$ based on these distributions in typical HAC configurations. The distribution of B_3 describes the effect on all lower-level nodes[31], and is set to $P(low) = .5$ and $P(high = .5)$. Similarly, B_4 presents the distribution based on the effect on all upper-level resources, and is set to $P(low) = .3$ and $P(high = .7)$. Finally, the distribution of C_1 reflects the criticality of a resource ($P(low) = .2$ and $P(high = .8)$). Most resources are considered critical[31]. However, some resources can be classified as noncritical because they do not pose an immediate threat to the application operation.

3.5 | Causality and Decision Network

The inference of the BDN-HAC model assumes that one or more nodes are instantiated. The decision node is usually set to *offline* because "failure" is a keyword used to extract failure events from log data in the BPFP component (i.e., the BDN-HAC model is used when a resource failure is detected). Therefore, the model is set to consider an automated decision in this case. Nodes that are not instantiated rely on both prior and posterior probability distributions. Figure 8 illustrates the model inference. The running example in Figure 2 is depicted as the "high availability cluster model". This model represents the composition of the underlying HAC resources. As described earlier, the web application comprises *service*, *IP*, *File System 1*, and *File System 2*, where the resource *database group* is a logical resource that groups all underlying resources, as depicted in Figure 2. When the parent *service* with three child resources (*IP*, *File System 1*, and *File System 1*) fails, all child components also fail, indicated by the "high availability cluster model". We use the running example and the related HAC setup to present the complete flow and subsequently describe the reasoning.

Example 1. Suppose that the first failure event occurred in resource "File System 2". This event is depicted by the arrow labelled 1 in Figure 8, where the dotted circle around the failed resource indicates that it is a noncritical resource. When Event 1 occurs in the resource, the failure information is extracted by the BPFP component, which adds all values obtained from the runtime environment of the BPFP. Thus, data for all four property groups are added, then passed to the BDN-HAC model. The BDN-HAC model is initiated and considers the resource failure while incorporating the properties and their causal relationships in two steps. The error-related and dependency-related nodes are consolidated in two latent nodes (blue circles). In the second step, the outcomes from the latent nodes and the two nodes (C_1 and D_1) are consolidated into the utility node (U_1). The utility node U_1

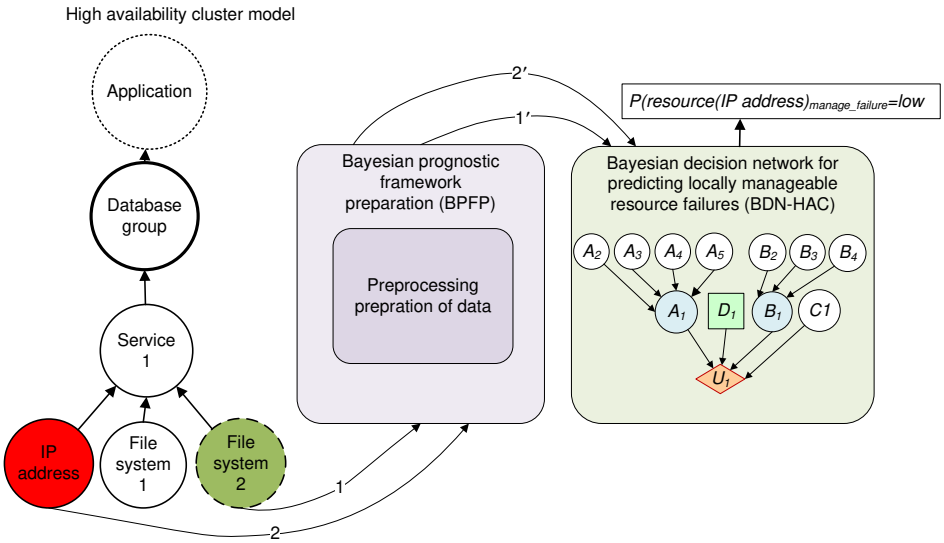


FIGURE 8 Illustrative example of the BDN-HAC model inference for the running example.

TABLE 4 Effect of incomplete data on prediction outcomes for model BDN-HAC, where NI indicates nodes that are not instantiated

No	A_2	A_3	A_4	A_5	B_2	B_3	B_4	C_1	D_1	U_1
1	low	true	low	true	local	low	low	true	offline	54.62
2	NI	true	low	true	local	NI	NI	true	offline	31.94
3	low	NI	low	NI	local	low	low	NI	offline	18.58
4	low	NI	low	NI	local	low	low	NI	online	87.4

outputs a value that indicates a low failure probability for the resource related to $1'$; hence, the failure can be managed locally. Although the model considers all values, the deciding factor is that the RG, 3, promotes masking a noncritical failure.

Suppose now that a new failure even, Event 2, occurs in the resource 'IP address', which is depicted as a red circle in Figure 8. The BDN-HAC model is initiated and considers the resource failure while incorporating the properties and their causal relationships in two steps. Consequently, the utility node U_1 outputs a value that indicates a high probability that the failure of the resource related to $2'$ cannot be managed locally.

Reasoning with Incomplete Data. Incomplete data influence the prediction outcome. If different subsets of nodes are instantiated for the same resource failure, the outcome may differ each time. Further, if data are missing for these nodes that add significant weight to the outcome, it will also significantly affect the BDN-HAC utility calculation and may even increase the likelihood of mispredicting whether a resource failure can be handled locally. Similarly, in some cases, nodes with lesser weights can take precedence over nodes with significant weights.

Example 2. To demonstrate these behaviours, four sets of data (including a complete set and three incomplete sets) are used to initiate our BDN-HAC-1 model, and the outcomes are listed in Table 4. Latent nodes (A_1 and B_1) are not listed because they do not receive any data. However, the conditional probability distributions of the latent nodes are updated automatically when the probability distributions of the parent nodes are updated. The weights, as presented in Section 3.4.1, are implicitly managed because they are part of the conditional probability construction. The sample data are constructed to emulate a nonfailure outcome and are performed for the same resource to provide a consistent view.

The first data set in Table 4 is fully instantiated, while the second data set has some incomplete data. The outcome shows that in both scenarios the BDN-HAC model yields an optimistic prediction that the resource failure will be locally manageable (since $U_1 > 0$, see eq. (12)). When some nodes are not instantiated, both prior and posterior (instantiated nodes) probability distributions are computed, impacting the outcome. There are also differences between nodes with significant weights and those

with lesser weights. For example, two of the critical nodes (C_1 and A_5) are not instantiated in Data Set 3, and the prediction (i.e., the utility U_1) differs significantly compared to Data Sets 1 and 2.

The prediction for Data Set 3 is still optimistic, which means that the likelihood of the resource failure not being locally manageable is low. However, the outcome has a lower utility value than for the other scenarios. This is because A_5 has a favourable prior probability distribution, which assumes that the resource can be reinitialised and decreases the likelihood of failure. Hence, the prior probability distribution of node A_5 has precedence over the prior probability of node C_1 even though C_1 has a high weight factor. Suppose A_5 is instantiated with a *false* value. In that case, the prior probability of C_1 takes precedence because it favours the state *true* (most of the HAC resources are considered critical), increasing the likelihood of failure.

Influence of the Decision Node 'Current State'. The decision node D_1 significantly influences the overall outcome. When no other node is instantiated, the default decision is *online*, and the expected utilities for the policies *online* and *offline* are obtained from the model as 83.7 for online and 2.45 for offline. Thus, the combined prior and posterior probability distributions, weights and preferences favour the decision policy *online*, and the decision outcome only changes when other nodes are considered.

Example 3. D_1 is set to *offline* in Table 4 for Data Sets 1–3, but when other nodes are considered, the final prediction in all cases is that the resource failure in question will be manageable locally. If the decision changes to *online* (Data Set 4 in Table 4), it changes the prediction significantly. The reason for this is that, when the decision is set to *online*, it adds significant weight to the overall outcome. Therefore, the prediction indicates that the failure of the resource is manageable locally.

4 | EVALUATION

We carried out extensive experiments to answer the following research questions:

RQ1. **What is the prediction quality of the proposed model, and how much improvement can be achieved compared with the HAC solution?** This research question corresponds directly to the main motivation for constructing a BDN model to address unutilised opportunities and, by doing so, improve the detection and prediction capabilities.

RQ2. **What effect do incomplete data have on prediction quality?** A situation with incomplete data may arise when one or more nodes do not receive data, and it could be the reason a particular HAC may not support some characteristics. It is important to determine the effects of incomplete data on prediction quality in such cases.

RQ3. **How do incomplete data in critical nodes vs. noncritical nodes influence prediction quality?** Critical nodes have high relative weights, implying a significant effect on prediction quality.

RQ4. **How do incomplete data in nodes representing established characteristics vs. new characteristics affect prediction quality?** The two sets of characteristics presented in this paper are compared when data are supplied to these nodes to ensure that the addition of the new characteristics can improve prediction quality. Both these sets of characteristics can also include critical and noncritical nodes.

RQ5. **What is the execution time of the BDN solution?** With this research question, we aim to determine the execution time of the solution (the BDN-HAC model and BPFP component) to ensure no negative effects. The HACs protect critical business systems, and as such, the response time must be fast to take mitigation actions based on predictions.

RQ6. **What is the runtime overhead of the BDN solution?** We aim to measure the runtime overhead to ensure that the operations of the proposed solution do not influence the production system in scope.

4.1 | Testbed

The testbed was established in the public cloud,[4] and the open-source HAC we deployed in the testbed was the ClusterLabs stack[5] (Pacemaker/Corosync)[24,45]. Subsequently, we deployed an enterprise resource planning (ERP) solution and included it in the HAC. The testbed was running continuously for more than two years and three months between February 2019 and May

[4]https://contabo.com/en/
[5]https://www.clusterlabs.org/

TABLE 5 Virtual machines used to enable high availability in the testbed, where VCPU indicates a virtual CPU

	Server 1	Server 2	Server 3
CPU (vCPU cores)	8 vCPU cores 2.20 GHz	8 vCPU cores 2.20 GHz	6 vCPU cores 2.40 GHz
Memory (GB)	30	30	20
Operating system (64-bit)	openSUSE Leap 15.0	openSUSE Leap 15.0	openSUSE Leap 15.0
Role	Primary node	Secondary node	Storage server
IP address	IP address 1	IP address 2	IP address 3
network	Network 1	Network 2	Network 3

TABLE 6 High availability cluster (HAC) configuration listing all resources of the ERP solution in the testbed, showing resource names, types, and groups (RGs)

Resource Name	Resource Type	RG	Resource Name	Resource Type	RG
Message and lock instance group	Resource group	N/A	FS main instance	File system	MI
Message and lock service	Service	MLI	VIP	VIP	MI
FS message and lock instance	File system	MLI	Backup lock server group	Resource group	N/A
FS trans	File system	MLI	lock system	Service	BLS
FS interface	File system	MLI	VIP	VIP	BLS
VIP	VIP	MLI	DLM group	Resource group	N/A
Database group	Resource group	N/A	DLM	DLM	DLM
Database	Service	DB	FS DLM	File system	DLM
FS database	File system	DB	CPU monitor	Monitor	N/A
VIP	VIP	DB	NIC monitor	Monitor	N/A
Main instance group	Resource group	N/A	SBD	SBD	N/A
Main instance	Service	MI			

Notes: RG: resource group, N/A: not applicable because it is a resource group, MLI: message and lock instance group, DB: database, MI: main instance group, BLS: backup lock server group, DLM: distributed lock manager, FS: file system, SBD: STONITH Block Device, VIP: virtual IP, NIC: network interface card

2021 to facilitate the development, testing, optimisation and evaluation of the BDN-HAC model. The following sections also describe the selected ERP and how the ERP components were integrated into the HAC.

Virtual Machines. We configured three virtual machines (VMs) for the testbed environment, and the details are presented in Table 5. Servers 1 and 2 functioned as the primary and secondary nodes of the HAC, respectively. Server 3 was employed as the shared storage provider.

HAC Solution. The ClusterLabs stack HAC (Pacemaker/Corosync) we selected is a comprehensive solution that supports a broad range of features[10]. Although the ClusterLab is an open-source HAC, it has been commercialised and included in commercial HACs (SUSE Linux Enterprise High Availability Extension, Red Hat High Availability Add-On, etc.). We selected this HAC for three reasons: (1) It is based on an open-source licence and is free of charge. (2) It supports many EAs, including our selected ERP solution. (3) It could be deployed in the public cloud where we established the testbed. Thus, the HAC provides ERP-specific agents to manage ERP resources and understands the internals of the ERP solution.

Enterprise Application. We used the above infrastructure to deploy, configure, and run a fully-fledged commercial ERP application widely used in industry to manage business functions, such as accounting, procurement, and logistics. The licence to use this application was secured with the help of a project collaborator. The actual ERP application and the details for the collaborator cannot be listed for confidentiality reasons. The selected ERP solution is a multilayered and multitiered solution that can be implemented across several servers. To demonstrate how various layers are addressed in the HA setup, we present the layers as resource groups in Table 6: the main instance, message and lock instance, backup lock instance, and database instance. The resource group distributed lock manager (DLM) is not part of the application layer, but it is included because it is required to enable HA for the EA. All four layers and their components are treated as SPOF in this implementation.

HAC Configuration. The complete configuration of the HAC is listed in Table 6, which presents the resource names, types, and corresponding groups. The five resource groups provide complete protection for the ERP application. Except for the resource group DLM, all resource groups had dedicated virtual IPs (VIPs) to enable relocation. In contrast, the resource group 'DLM' hosted a cluster file system. Therefore, a VIP was not required because the services were available on both nodes simultaneously, managed by the DLM service.

TABLE 7 Overview of the data sets

Data Set	#Nodes	#Instances	#Failures	TC	Scope	S†
1	12	72	48	T1-T9	Ts	N
2	12	68	48	T1-T9	Ts	Y
3	12	36	26	T10-T19	Pd	N
4	12	37	28	T10-T19	Pd	Y

Notes: #Nodes: number of nodes in the model, #Instances: total number of obtained records, #Failures: number of failures for further analysis, TC: test case (T1–T9: part of the test, T10–T19: part of the production data), in Scope: Ts: Test, Pd: Production data, in † S: stickiness: Y: used, N: not used.

4.2 | Evaluation Methodology

As described earlier in Section 4.1, the HAC-protected ERP application from the testbed was running for approximately two years. During the first nearly 14 months, the testbed was monitored to collect information to develop the BDN model. Afterwards, the ERP application was subjected to a wide range of injected failures for the final eight months (6 July 2020 to 12 March 2021). The logs generated by the HAC were used to create multiple data sets to test and evaluate the BDN model. All relevant data sets, log files, test protocols, calculations and graphs are available on our GitHub repository.[6] The BDN model was not integrated with the HAC, so its evaluation and the end-to-end model evaluation were conducted by passing inputs from these data sets to the BDN model, enabling us to establish the effect of using the BDN model as part of a future HAC.

4.3 | Test Cases

HACs experience failure on multiple levels, and these failures were handled using the threefold strategy detailed earlier in the paper. As the size and complexity of the HAC increases, so does the combination of failures that can occur. Testing all conceivable kinds of failures is generally not possible. Therefore, we chose to induce critical failures using fault injection methods[46,47]. Moreover, to ensure that we covered most of critical combinations of HAC configuration parameters, the test cases were repeated under two conditions: with and without the stickiness set, meaning that the standard behaviour of the HAC under analysis could be observed. The stickiness parameter prefers the node where the resources are currently running.

Fault Injection Methodology. The objective of using fault injection was to induce many real-life failures related to EAs and HACs. Hence, we identified the corresponding failures by analysing the research literature, case studies, and documentation[23,48,49,50]. Moreover, we also identified several fault injections recommended by vendors of both the EA and HAC that we implemented in the testbed[51,52,50]. A typical failure in the HAC is at a single resource level, but multiple failures can also occur across two or more resource groups. Further, we created additional fault injections to capture failures related to the characteristics introduced in Section 3 (e.g., application-provided self-healing capabilities or a weak (noncritical) resource). Hence, the following types of failures are included in the evaluation of the BDN model: (1) a critical resource fails, (2) a resource with application-provided self-healing capabilities fails, (3) a noncritical resource fails, (4) two resources in the same resource group fail, (5) two resources in two different resource groups fail, (6) a resource repeatedly fails, and (7) a shared-dependency resource fails.

Test Cases. Two groups of test cases were created, comprising nine and 10 test cases, respectively. The data sets generated by the first group were used to test the model, and the data sets from the second group were used to supply online testing data to the model for evaluation experiments. Each test case consists of one or more fault injections. Usually, only one fault injection is associated with a test case. However, when two faults are injected on two different resources, the test case consists of two fault injections. The resources in such a case could come from the same resource group or two resource groups.

4.4 | Data Sets

We recorded the following information during the execution of our test cases: name of the failed resource, event timestamp, related resource group and characteristics introduced in Section 3. We created four data sets and organised them into two groups: (1) test and (2) production. The complete list of data sets produced by the experiments is presented in Table 7, and we detail each data set as follows:

[6]https://github.com/ps234/BDN-project

TABLE 8 Metrics derived from the basic metrics, respective symbols, formulas and descriptions

Metric	Symbol	Formula	Description
Recall or sensitivity or true positive rate (TPR)	r, tpr	$\frac{TP}{(TP+FN)}$	Correctly predicted failures/all true failures
False positive rate (FPR)	fpr	$\frac{FP}{(FP+TN)}$	Incorrectly predicted failures/all nonfailures
Accuracy	ac	$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	Correctly predicted failures/all predictions
Precision	pr	$\frac{TP}{(TP+FP)}$	Correctly predicted failures/all predicted failures
Specificity	$1-fpr$	$\frac{TN}{(TN+FP)}$	Correctly predicted nonfailures/all nonfailures
F-measure	fm	$\frac{2\times pr\times tpr}{pr+tpr} \in [0,1]$	Weighted harmonic mean of precision and recall
MCC†	mcc	$\frac{(TP\times TN - FP\times FN)}{\sqrt{(TP+FP)\ (TP+FN)(TN+FP)(TN+FN)}}$	Provides a balanced measure to measure the quality of binary classifications

†MCC: Matthews correlation coefficient

1. Data Set 1 was obtained by executing test cases T1–T9 twice in the HAC with the stickiness policy disabled. The resulting HAC log file was extracted, processed and prepared by the BPFP because the primary objective of this data set was to provide data to the BDN-HAC model. The HAC failures identified and prepared by the BPFP were duplicated to test the prediction quality of the BDN model.

2. Data Set 2 was acquired by executing test cases T1–T9 twice in the HAC with the stickiness policy enabled. The resulting HAC log file was extracted, processed and prepared by the BPFP to be input into the BDN-HAC model. We duplicated the output from the BPFP to validate the prediction quality of the BDN-HAC.

3. Data Set 3 was acquired by executing test cases T10–T19 twice in the HAC with the stickiness policy disabled. The resulting HAC log file was extracted, processed and prepared by the BPFP to deliver the data to the BDN-HAC model.

4. Data Set 4 was obtained by executing test cases T10–T19 twice in the HAC with the stickiness policy enabled. The resulting HAC log file was extracted, processed and prepared by the BPFP to be input into the BDN-HAC model.

4.5 | Evaluation Metrics

Several metrics were used to evaluate the BDN-HAC model. When measuring the prediction quality, four outcomes are possible, and we refer to them as *basic metrics*, which are used to define *derived metrics*[32]. We provide a concise description of each outcome as follows: true positive (TP): both the prediction and actual result are positive; false positive (FP): the prediction is positive, whereas the actual result is negative; false negative (FN): the prediction is negative, whereas the actual result is positive; and true negative (TN): both the prediction and actual result are negative. The derived metrics used in the evaluation and defined in terms of the basic metrics are presented in Table 8.

Receiver operating characteristics (ROC) curve. The ROC analysis and resulting ROC curves are used to evaluate the prediction quality of a binary model or compare the prediction quality of multiple binary models[53]. The ROC curve depicts the trade-offs between the true positive rate (TPR) or sensitivity and false positive rate (FPR) or 1– specificity[32]. Hence, the curve presents the TPR on the vertical axis and the FPR on the horizontal axis, and the corresponding area under the curve (AUC) represents the area underneath the ROC curve[32]. The AUC ranks randomly chosen positive predictions higher than randomly chosen negative predictions. If $TPR \approx 1$ and $FPR \approx 0$, it indicates the model performs well[54]. The following paragraph describes the experimental design for ROC analysis to measure the prediction quality of the BDN-HAC model.

We assumed a 95% confidence interval for measuring the prediction quality using the ROC analysis. Further, we used a state variable to indicate the binary outcome and used 0 to indicate failure and 1 for no failure using the expected outcome (Section 4.6). Thus, the binary value was given as the "value of the state variable" when plotting the ROC curves, which means 0 represents both resource group and system failures. Subsequently, we mapped the outcomes from the HAC and BDN models using a mapping table derived from the threefold strategy that HACs use to deal with failures. These values were entered as test variables for HAC and BDN-HAC. The BDN-HAC outcomes are mapped to either 1 or 3 to indicate failure and no failure, as it deals only with binary outcomes. Similarly, the corresponding HAC outcomes are also mapped to a binary value to indicate

failure or no failure. For example, if the HAC performs a system failover and the BDN-HAC predicts no failure, these values are mapped to 1 and 3, respectively. Moreover, the state variable is set according to the expected outcome. Using this procedure, we used the actual outcomes to map and plot the ROC curves using the software package Statistical Product and Service Solutions (SPSS Statistics).

4.6 | Expected Outcome

Expected outcomes are results expected from the HAC when a particular type of resource fails. For example, the expected result may suggest that a resource group failover is sufficient to resolve the problem when a specific resource type fails. To assess a typical HAC outcome upon failure, we identified the set of expected outcomes. The actual outcomes of the HAC and BDN-HAC were checked against the expected outcomes to evaluate the quality of the actions taken by the HAC and the quality of the predictions by the model.

To obtain the expected outcomes, we reviewed the results and test cases that vendors and service providers published for the EA under analysis[51,48,52,49,55,56,50]. Furthermore, we validated these outcomes with further experiments on the HAC in the testbed. Hence, the empirical evidence was obtained using the following three approaches:

1. We analysed the HAC log files from different HAC solutions for the same type of EA architecture.
2. We performed multiple iterations of experiments in the testbed to study the results. For example, we performed failovers manually to ensure that the resource groups could failover independently, and we verified that the HAC in question could handle such failures and take suitable mitigation actions.
3. We analysed the data produced by the experiments and derived conclusions. For example, the HAC initiated a system failover when a shared resource failed. However, in the subsequent experiments, the HAC reinitialised the resource and achieved no failure.

4.7 | Evaluation of the Locally Manageable Resource Failure Prediction

The evaluation of the BDN-HAC model is presented in six steps. First, we evaluated the prediction quality of the model (RQ1). Next, we assessed the model's ability to deal with incomplete data (RQ2). Third, we determined the difference in prediction quality when only critical nodes receive data compared with when noncritical nodes receive data (RQ3). Fourth, we investigated prediction quality when nodes representing established characteristics (ECs) receive data compared with nodes representing new characteristics (NCs) receive data (RQ4). Fifth, we evaluated the execution time of the BDN solution (RQ5). Finally, we investigated the runtime overhead (RQ6).

The total number of nodes in the implemented BDN-HAC model are 12, out of which ten chance nodes, two latent nodes, one utility node and one decision node. The model has in total 83 parameters and 23 states. Using the model, we employed the data sets prepared in Section 4.4 to perform the evaluation. All latent nodes were excluded when the model was inferred to ensure they did not obtain any data. The decision node "*current_state*" was set to offline because the BDN-HAC model was constructed for use after failures of individual HAC resources. Furthermore, we employed policy evaluation as the primary algorithm for BDN inference.

We used the GeNIe modeller v3.0 as the primary tool[57] to construct and test the BDN model, and multiple software solutions, such as SPSS Statistics, Power BI and Excel, to analyse and visualise the results.

4.7.1 | Prediction Quality (RQ1)

Using the experimental design for ROC analysis from Section 4.5, we plotted the ROC curves for Data Sets 1, 2, 3 and 4 (i.e., for all data sets processed by the BDN-HAC model) as depicted in Figures 9, 10, 11 and 12. The HAC outcome had 8% FNs in Data Set 1 (Figure 9), indicating that the HAC performed a system failover when a service, such as a database, failed. However, the expected outcome was 'manageable (no failure)' because the HAC was expected to reinitialise the resource without triggering a complete system failover.

In contrast, the BDN-HAC model detected the failure and predicted it as a locally manageable failure, and the FN became a TP. Another example is that the HAC performed a resource group failover when the resource 'main instance service' was terminated as part of the fault injection in Data Set 2 (Figure 10). This event occurred only once, and in the subsequent events,

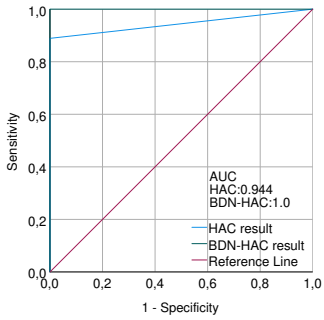


FIGURE 9 Receiver operating characteristic (ROC) curve indicating prediction quality for Data Set 1.

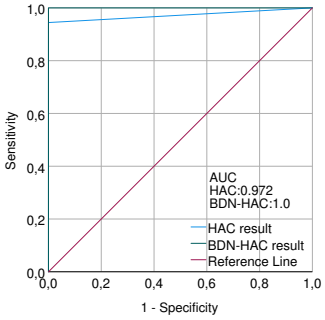


FIGURE 10 Receiver operating characteristic (ROC) curve indicating prediction quality for Data Set 2.

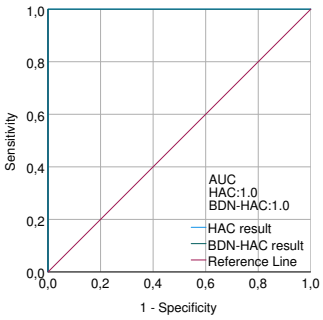


FIGURE 11 Receiver operating characteristic (ROC) curve indicating prediction quality for Data Set 3.

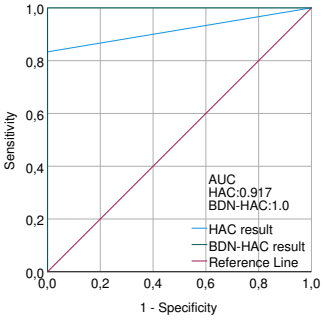


FIGURE 12 Receiver operating characteristic (ROC) curve indicating prediction quality for Data Set 4.

TABLE 9 Summary of the receiver operating characteristic (ROC) analysis grouped by data sets displaying the results for the area under the curve (AUC). An asymptotic significance (AS) with a 0.5 null hypothesis true area is also presented. The AS determines the statistical significance of the relationship between the variables, and $p < 0.05$ indicates a statistically significant relationship—further, the standard error (SE) under the nonparametric assumption is also presented

Test Result	AUC	AS	SE	Confidence Interval	
				Lower Bound	Upper Bound
Data Set 1					
HAC result	0.944	0.016	0.000	0.913	0.976
Data Set 1	1.000	0.000	0.000	1.000	1.000
Data Set 2					
HAC result	0.972	0.011	0.000	0.950	0.995
Data Set 2	1.000	0.000	0.000	1.000	1.000
Data Set 3					
HAC result	1.000	0.000	0.000	1.000	1.000
Data Set 3	1.000	0.000	0.000	1.000	1.000
Data Set 4					
HAC result	0.917	0.020	0.000	0.878	0.955
BDN-HAC Data Set 4	1.000	0.000	0.000	1.000	1.000

the HAC restarted the same service successfully. The BDN-HAC model correctly predicted that the HAC could manage the failure locally (i.e., restart the service). In Data Set 3, both the HAC and BDN model performed identically, reaching a higher AUC (Figure 11). In Data Set 4, the HAC initiated unnecessary system failures for a shared resource failure (cluster file system), whereas the BDN-HAC treated the failure as locally manageable (no failure) (Figure 12).

We calculated the results from Data Sets 1, 2, 3 and 4, and the BDN-HAC model achieved a mean AUC that was 4.85% better (i.e., higher) than that achieved by the HAC. Furthermore, the BDN-HAC model using data sets without stickiness (Data Sets 1 and 3) performed better than those with stickiness (Data Sets 2 and 4). For example, the BDN-HAC model using Data Set 3 with stickiness achieved the same result as the HAC. The results are summarised in Table 9. The result shows that assessing a set of characteristics can improve the prediction quality. This means there is a potential to improve the prediction quality even further by adding more relevant characteristics. However, such characteristics must be common for a major part of the HAC solutions because our objective is to create a HAC-neutral solution.

4.7.2 | Effect of Incomplete Data on Prediction Quality (RQ2)

We analysed the effect of incomplete data on the prediction quality by providing data to only a subset of the BDN-HAC nodes (and using the default probability distributions for other nodes). In the first step, we started with node A_2. In the second step,

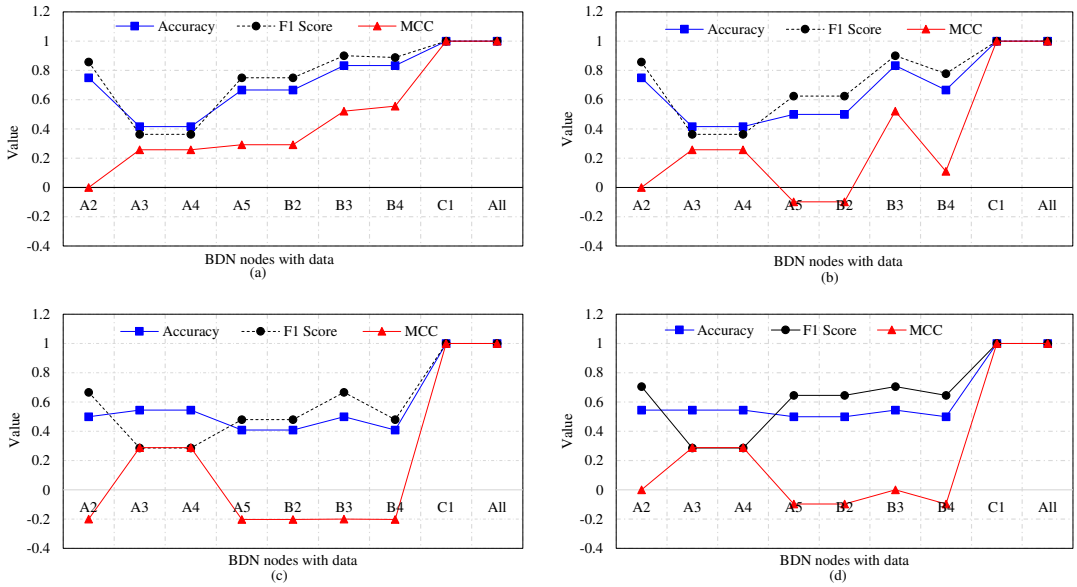


FIGURE 13 Changes in the prediction outcome based on nodes receiving data for (a) Data Set 1, (b) Data Set 2, (c) Data Set 3 and (d) Data Set 4. The labels on the horizontal axis show the last node supplied with data (e.g., the values for 'B_2' were obtained when the nodes A_2, A_3, A_4, A_5 and B_2 were supplied with data, and all of the other BDN nodes were not).

we delivered data to both A_2 and A_3. In the third step, we provided data to three nodes, A_2, A_3 and A_4, and we continued until all nodes were supplied with data in the final step.[7] Figure 13 illustrates the model output as each node was provided with data, and it is presented using the accuracy, F1-score, and Matthews correlation coefficient (MCC)[58,59] evaluation metrics. Including the MCC overcomes imbalanced classes, as we noted that the data sets are imbalanced because they contain many FPs and FNs. However, the BDN-HAC model aims to eliminate FPs and FNs because the purpose of the model is to reduce the downtime for business-critical systems. Therefore, a balanced measure is required when measuring the model quality. Hence, the primary metric for measuring prediction quality with incomplete data is MCC.

A high FP percentage was observed when only A_2 obtained data. For example, the percentage was 27% in Data Set 1 and 45% in Data Sets 3 and 4. This resulted in relatively high accuracy (0.4167, 0.75 and 0.5455, respectively) and F-score (0.3636, 0.8571 and 0.6667, respectively) but a low value for MCC. However, when the second node, A_3, also received data, a shift from FPs to FNs occurred. We observed 58% FNs in Data Set 1 and 45% FNs in Data Sets 3 and 4. The curve shows this shift by reducing the accuracy to 0.4167 in Data Set 1. However, no change in accuracy or the F1-score occurred in Data Sets 3 and 4, but a significant change in MCC occurred. In Data Set 4, 45% of the FPs shifted to 45% FNs in A_3.

4.7.3 | Influence of Critical and Noncritical Nodes (RQ3)

Figure 14 displays a comparison between the prediction quality achieved when only the critical nodes were supplied with data, and that achieved when only the noncritical nodes were supplied with data. The number of noncritical nodes is five (A_2, A_4, B_2, B_3 and B_4), whereas the number of critical nodes is three (A_3, A_5 and C_1). However, the three critical nodes have a higher influence than the five noncritical nodes due to weight assignment. In Data Set 1, the data to critical nodes results in 27% FPs, whereas for noncritical notes, the outcome was 9% FNs and 18% FPs. This results, in general, in higher values for critical nodes across all the evaluation metrics, as depicted in Figure 14. A similar pattern was observed in all data sets. For example, we observed 9% FNs for critical nodes in Data Set 4, but the figures were 9% FNs and 40% FPs for noncritical nodes, which results in a negative value for MCC for noncritical nodes. Therefore, when data are supplied only to noncritical nodes, it tends to create more FPs, while when data are supplied only to critical nodes, it creates FNs. These results indicate that incomplete data, primarily when very few noncritical nodes are supplied with data, provide poor predictions. Moreover, the results demonstrate that whenever the next node that receives data is critical, it shifts the FPs to FNs in most cases (i.e., A_3 and A_5), which results in

[7]The BDN uses the prior probability distribution for the nodes not provided with data, except for the latent and utility nodes.

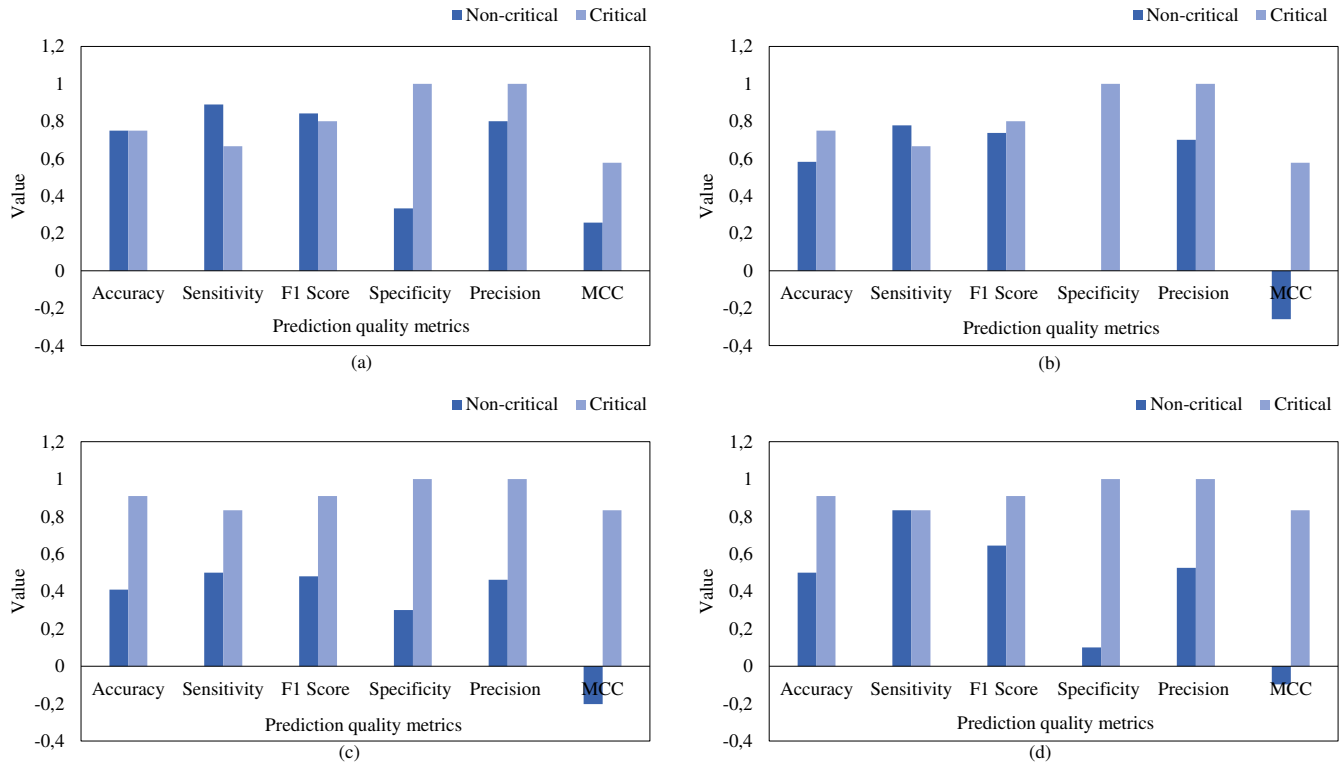


FIGURE 14 Comparison of prediction quality for experiments in which only the critical or only the noncritical nodes were supplied with data for: (a) Data Set 1, (b) Data Set 2, (c) Data Set 3, and (d) Data Set 4.

negative values for MCC. Further, the results confirm that critical nodes play a crucial role in providing prediction capabilities for the BDN-HAC model.

4.7.4 | Established vs New Characteristics (RQ4)

As described in Section 3, the BDN nodes take into account two types of HAC characteristics, namely characteristics also used outside of our project (which we termed 'established characteristics' or ECs) and characteristics proposed by this project (which we term 'new characteristics' or NCs). There are three EC nodes (A_2, A_4 and A_5), and the fourth node is a decision node for which a negative decision to indicate failure is assumed. The NCs consists of five nodes (C_1, A_3, B_2, B_3 and B_4). We investigated the BDN-HAC prediction quality when only EC nodes receive data and compared the result to the scenario when only the NC nodes receive data. Figure 15 depicts the results from the evaluation. There are 17% FPs in ECs compared to 25% FNs in NCs in Data Set 1, and a similar pattern can be observed in Data Set 2. As shown in Figure 15 (a) and (b), accuracy is reduced in NCs as a result. However, significant changes exist in Data Set 3, which indicates that ECs have 45% FPs but 9% FNs, resulting in lower values for ECs across all evaluation metrics except for sensitivity.

Similarly, 45% FPs could be observed in ECs, whereas 9% FPs and 9% FNs in NCs could be observed in Data Set 4. However, ECs have only one critical node (A_5), whereas NCs have two (C_1 and A_3), and this could influence the outcome considerably. Overall, the results reveal that NCs improve the prediction quality. For example, if the accuracy is 0.69 with EC nodes, the quality is improved by 32% when NCs are introduced (Data Set 3).

4.7.5 | Execution Time (RQ5)

Measuring the execution time of the complete BDN solution requires that both components, the BDN-HAC model and the BPFP component, are considered. We measured both components independently first, and then we combined the results to present the overall execution time.

Three steps are associated with using the BDN-HAC model to obtain predictions: BDN input, BDN inference and BDN output. The BDN input transfers data to the BDN model to be processed, and the BDN inference performs the actual model

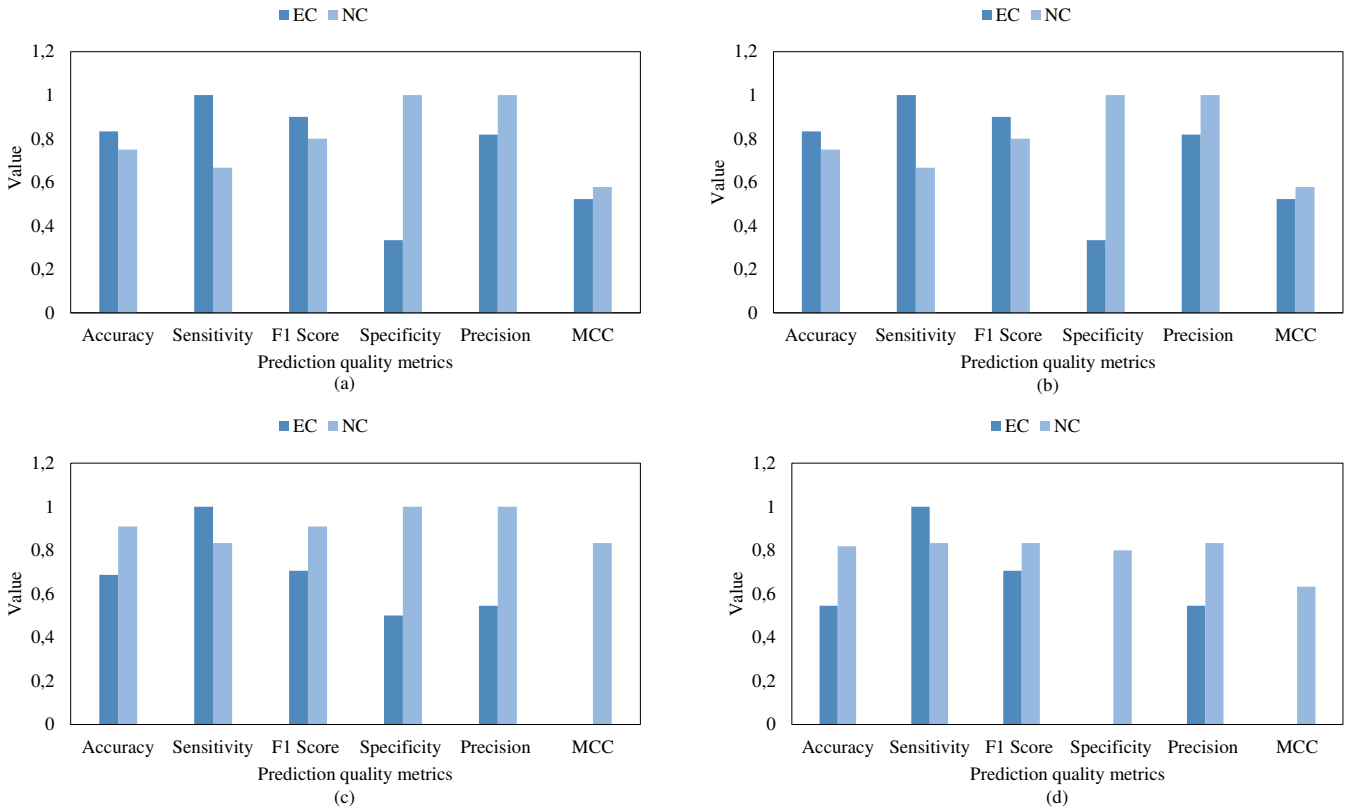


FIGURE 15 Comparison of prediction quality between established characteristics (ECs) and new characteristics (NCs) for (a) Data Set 1, (b) Data Set 2, (c) Data Set 3 and (d) Data Set 4.

execution using data from the previous step. Subsequently, the outcome is interpreted as either a locally manageable failure or not and outputted to take appropriate action. The mean execution time when all steps and data sets are considered is 84ms, which were obtained from running the experiments on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10. The BDN-HAC model requires, on average, 1ms to complete the inference as the model is small (with only 12 nodes) and employs a utility node with predefined preferences, which reduces the computational complexity. The BDN input and output execution times are higher because they correspond to steps that convert the input/output values of the BDN.

The evaluation result indicates that the BDN-HAC module's execution time is negligible. Furthermore, a significant part of the execution times associated with 'BDN input' and 'BDN output' would be eliminated if the BDN-HAC was integrated into a future HAC solution, as in that case, the reading and processing of the logs would be replaced by using events received by the HAC.

Similar to the BDN-HAC, the BPFP component consists of several steps, and the mean execution time for the different steps of the BPFP is 4.5 s. The polling functionality, to use the log interface to poll and extract data from HAC logs, stands out and consumes a sizeable portion of the total runtime of the BPFP component with an interval between 2s and 10s. In contrast, the maximum execution time for each of the other BPFP elements is less than 0.07s. Although the polling interval is set to 10s, extracting the failure information can occur earlier because a failure can occur close to the polling time.

When considering all the execution steps in all the related data sets, we obtain 1.84% for the BDN-HAC model and 98.16% for the BPFP component. The execution time of the BPFP contributes to a significant portion of the overall execution time for the BDN solution. The total time required by the BPFP component is a concern, particularly the polling time. However, the polling time can be increased, which should reduce the overall time required by the solution. Moreover, if the BDN-HAC solution is integrated with HACs in the future, it will eliminate the polling time because the HAC will pass the failure information directly to the BDN-HAC in that case.

4.7.6 | Runtime Overhead (RQ6)

Similar to measuring the execution time, we also measured the runtime overhead associated with the complete solution comprising the BDN-HAC model and the BPFP component. We used GeNIe modeller to perform inference on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10. The two steps, BDN input and BDN output, were executed using Linux scripts in the testbed. We measured the CPU utilisation and memory by monitoring the utilisation at the process level and the CPU utilisation was experienced for a period of $<$ 85ms. The median values for the BDN input are 0.20%, 0.28%, 0.27% and 0.26% for Data Sets 1, 2, 3 and 4, respectively. The maximum values are 0.32%, 0.42%, 0.42% and 0.41%. The BDN inference also has little CPU utilisation, and the maximum values are 0.40%, 0.50%, 0.42% and 0.42%. The reason could be that the BDN model has only 89 parameters, and thus it does not require complex runtime calculations. The BDN output has the lowest utilisation of all three, and the maximum values are 0.20%, 0.36%, 0.35% and 0.38%. The evaluation of execution time demonstrates that it rarely exceeds 100ms, which does not add significant time to the overall execution time of the BDN-HAC solution.

The evaluation of the memory utilisation (measured in KB) shows that the BDN inference has the highest utilisation, indicated by the maximum values (68, 63, 68 and 68 KB) in the four data sets. Comparatively, the other two steps used less memory because they dealt with fewer variables. The BDN input processes data for only those variables that the BDN model requires. Moreover, the BDN output has only the utility value from the BDN-HAC model. Even if the maximum memory utilisation for all three steps is combined, it is still less than 180 KB. However, considering that each step is executed sequentially, the computational overhead associated with the CPU and memory is only linked to one step at a time. Therefore, the conclusion is that the overhead associated with executing all three steps of the BDN-HAC model is negligible.

We evaluated the runtime overhead of the BPFP component by measuring the individual elements. The CPU utilisation was experienced for a period of $<$ 5.27s for the duration of the BPFP execution times. The polling component stands out from other components because it uses consistently more CPU resources. The maximum utilisation is 0.9% across all data sets, and the median is 0.7%. Overall, the results indicate negligible CPU utilisation for all components except for polling. However, because polling was scheduled to run every 10s, sometimes a polling process may create a new process while the previous one is still running, explaining the standout behaviour.

The memory utilisation of the BPFP (measured in KB) shows a mean value of 120 KB for all the steps associated with the BPFP component and all data sets. The median value for polling is between 66 and 77 KB. The results reveal that the memory utilisation by the BPFP is very low.

The combined view of resource utilisation demonstrates that the BDN-HAC and the BPFP add little overhead. The components of these two are executed sequentially; therefore, CPU utilisation is not cumulative. The memory utilisation tends to be sequential, as the next component is initiated only when the previous component is finished. Therefore, even the combined memory utilisation is considered low.

4.8 | Threats to Validity

We identified several construct, internal and external threats that could affect the validity of the evaluation presented in this section:

- **Construct validity –** The experiments were conducted using one primary HAC solution. The different HAC solutions available have distinct characteristics that can influence the prediction outcome. To reduce this threat, we used logs from multiple HAC solutions to develop the BDN-HAC model. Furthermore, we evaluated our BDN-HAC model using a HAC solution with different configurations of HAC, such as stickiness, to ensure that the evaluation captures the behaviour of multiple types of HACs.

- **Internal validity –** While the empirical evaluation of the BDN-HAC was performed in a production-like environment, with a complete ERP solution, only data from one EA were used in the evaluation. To mitigate this threat, we used data from multiple HAC solutions to develop the solution.

- **External validity –** The testbed does not have all the possible components, redundancy setups and related configurations required to fully mirror an environment for hosting a business-critical system. For example, a business-critical setup typically requires a separate network to be set up to allow quorum communication, but we could not provide that in the public cloud. Another example is that a business-critical system may use a quorum with a redundant setup using external

devices. However, we could only deploy a quorum on the two available nodes. Therefore, HACs may perform better than in the evaluation results presented. To mitigate this threat, we set up the testbed as close as possible to an environment used to support a business-critical system. Moreover, our objective is to demonstrate this new approach using probabilistic reasoning and improved detection and prediction capabilities. Therefore, the overall solutions are expected to perform better when the BDN-HAC model is deployed on HACs.

4.9 | Discussion

We carried out an extensive evaluation of our BDN-HAC solution by assessing its response to over 200 failures (#Instances in Table 7) injected across 19 distinct test cases (Section 4.3). Most test cases have a single failure injection but some test cases include two failure injections to evaluate the effect of two resources simultaneously failing. When the first failure resulted in either a resource group failover or system failover, the result of the second failure injection became unobservable. Thus, those failure injections affected by this were not considered. Out of the remaining failures, the HAC reinitialised 65 resources and carried out one resource group failover and 36 system failovers. The BDN-HAC model correctly predicted that 70 failures could be managed locally and 32 were unmanageable locally. The BDN-HAC produced more true positives (TPs) and fewer true negatives (TNs) compared to the results of the HAC because it considered the additional properties introduced in this paper as follows:

- The HAC performed four system failovers for a critical resource (database service) that could be reinitialised. The BDN-HAC produced TPs for all of them, considering the properties in Groups 1 and 2, as described in Section 3.
- Two failures associated with noncritical resources led to a resource group failover by the HAC. In contrast, the BDN-HAC produced TPs for all, considering property Group 3.
- Two test cases were used to inject failures consecutively five times on a resource with self-healing capabilities (the redundancy factor in Table 1). The application reinitialised the resource rapidly for all 60 of these failures. The first three fault injections in the same test case were quickly remedied within under 100 ms, but the application required a longer time for the last two. The HAC was not aware of any of the failures or self-healing capability provided by the application. However, when the delay in reinitialising the resource by the application in the later part of the fault injections coincided with the HAC's monitoring interval for that resource, the HAC could identify the failures and record them in the log file. The HAC then reinitialised the resource, although the application had already reinitialised it.

The conclusion is that the BDN-HAC solution exploits the capabilities provided by HAC, EA and other properties to significantly reduce downtime compared with the HAC in the testbed.

4.10 | Lessons Learned

In this section, we present the key lessons learned from developing and evaluating the proposed BDN-HAC model:

- A model that works with HAC failures must understand how the HAC behaves when a particular failure occurs because it otherwise leads to incorrect predictions. We studied the behaviour of HACs extensively to develop a way to capture the behaviour using a set of characteristics. Similar studies are required when extending the model or developing a new BDN model to work in other areas.
- The use of FMEA to identify the effect of individual characteristics on latent (target) variables and translate the outcome into related weights provides the flexibility to add more characteristics. However, relative weights must be identified correctly because, otherwise, it may increase the risk of an incorrect prediction. Similarly, connecting multiple variables to a latent variable requires an analysis of the relationship between the variables and the latent variable, and FMEA can effectively achieve this.
- Although the model is developed to handle incomplete data, the expectation is that complete data are delivered to the model; thus, there is no problem with incomplete data. For this to work, the corresponding environment using BPFP must be set up so that the runtime can prepare the data associated with most data.

- The BDN-HAC model is a white box model that allows following the visual decision paths, and we used this information to understand the reasoning and improve the model.
- Accurate reasoning with BDN requires that the relative weights and preferences are calculated correctly and assigned to the nodes, ensuring that latent nodes do not receive data but represent only conditional probabilities in chance nodes.

5 | RELATED WORK

Failure detection and prediction at the component level are widely researched in such fields as cloud, grid, and high-performance computing (HPC) using various techniques, such as artificial intelligence (AI), machine learning (ML), and rule-based and probabilistic models[32,60,61]. The common attributes of such environments are that they often consist of many server nodes and manage and process critical data[62]. Thus, the failure of the SPOF components may have a severe cost, such as the loss of revenue when a corresponding application is unavailable[1]. Therefore, studies have focused on predicting component-level failures for the SPOF to take appropriate actions before a component failure occurs.

The SPOF components in data centres include network devices, disks and server components, such as CPU and memory. Failure prediction for these components can significantly improve the availability of the systems using the components. Shenglin et al.[63] proposed a tool, PreFix, which predicts potential failures in network switches. The tool uses a random forest algorithm to learn from historical system logs (syslogs) and subsequently uses it to match entries from the real-time syslog to provide a prediction. The tool was compared with two other models employing a spectrum-kernel support vector machine and hidden semi-Markov model, outperforming both in accuracy. The disk is considered a component of servers or systems and is also an essential component in data centres. Xiao et al.[64] proposed an online failure prediction model for disks that uses an online random forest algorithm. The online model was updated incrementally to ensure that recent data were captured, and the online approach provided better failure prediction than the off-line random forest approach. The team conducted experiments, revealing that the online model could achieve a failure detection rate of 93% to 99% with reasonably low false alarm rates. Watanabe et al.[65] experimented with a method to predict failures that learn message patterns as failure signs by classifying messages in virtual environments in a cloud data centre. The method predicted failures with 80% precision and 90% recall.

Both HPC and grid computing are typically large scale and require that components are protected at several levels. A head node in an HPC system is considered an SPOF component, whereas the individual nodes are not. However, failure prediction considering both aspects can potentially improve the availability of such systems. Ashraf et al.[66] employed an ML technique to derive application fault propagation models so that the number of corrupted memory locations at runtime can be estimated in HPC applications. Moreover, Platini et al.[67] proposed an ML model to predict CPU overheating in HPC systems using a gradient boosting tree algorithm. The model employs temperature variation trends and predicted 76% of the overheating events 5 min in advance with a precision of 76%. Anwesha et al.[68] created the technique called Desh to analyse HPC logs to predict failures. The solution uses a three-phase deep learning approach to train, re-train and predict the failure of nodes. The experimental evaluation results demonstrated that the solution could provide a lead time of 3 min with 85% recall and 83% accuracy. The lead time can be used to take proactive actions on the failing nodes.

While grid computing and HPC are indicators for large-scale computing, failure prediction models appear efficient even with small but critical solutions, such as safety-critical systems. For instance, Baldoni et al.[69] tested a model using a hidden Markov model (HMM) to predict failures in a safety-critical system (air traffic control system). The results revealed that failures could be accurately predicted within a few hundred seconds before the failure occurred. Dangut et al.[70] developed a technique based on deep reinforcement learning to predict extremely rare failures in complex aircraft systems. The technique used aircraft central maintenance system logs to predict failures. Experiments using data sets from aircraft central maintenance systems indicated that the technique provides superior performance with a 20.3% improvement in the geometric mean and a 97% reduction in the false-positive rate. Wang et al.[71] developed a BN model to predict weather-related failures in railway turnout systems, and the model demonstrated high accuracy. The BNs were successfully employed to predict system failures of numerous interconnected components because BNs can represent both components and the system[72]. In addition, Sucar et al.[73] created a methodology for modelling the reliability of complex systems based on BNs, using probability propagation techniques to obtain the system reliability. Pitakrat et al.[74] proposed a model for predicting the likelihood of a component failure, and both hardware and software components were evaluated. The result is passed to another model to propagate the failure and predict the component failure effects on the system. Thus, the component failure prediction model was part of an architecture-aware online failure prediction

approach called Hora, and the complete approach improved the area under the ROC curve by 9.9% compared with the outcome of a monolithic approach.

However, there has been limited study of failure prediction at the resource level in the context of HACs. The High Availability Open Source Cluster Application Resource (HA-OSCAR) research program aimed to deliver HA services to SPOF components of HPC and investigated the prediction of hardware components using a hardware platform interface (HPI)[75]. This interface captures events associated with hardware; thus, the initiative collects such information, analyses it and predicts the potential failures of hardware components. Using the HA-OSACR HAC, Lee et al.[76] proposed a stochastic model to predict the SPOF components of HPC (head node). Although the primary focus has been predicting potential failures of SPOF components of HACs, studies have also been conducted to identify potential failures of nodes in HACs. In addition, Cheng et al.[77] proposed a model to detect sick nodes and predict the time-to-failure of nodes in a clustered solution.

BDNs are frequently used to optimise the decision-making process under uncertainty in disciplines, such as engineering[78,79,80,81], medicine[82,83,84,85], biology[86,87], and information technology[88,89,90]. BDNs are also ideal for reducing risk in the decision-making process because risk is associated with uncertainty. For instance, risk can be expressed as a product of likelihood and consequence, which BDNs can represent.

In economics, BDNs are used to reduce risk (e.g., invest in a new product with minimum risk)[91]. Similarly, reducing risks in projects and other implementation initiatives allows one to choose the option that reduces the risk significantly[92,93]. The BDNs are also widely studied to support complex decision-making processes where a large volume of interconnected data must be evaluated[94]. For instance, Seixas et al.[95] proposed a BDN model to support diagnosing Alzheimer's disease (AD) and mild cognitive impairment (MCI). The models exhibit better results for diagnosing MCI when compared with many of the well-known classifiers and competitive results for dementia and AD. Similarly, Neapolitan et al.[96] developed a BDN model to determine whether kidney transplants are beneficial for a particular group of patients by evaluating the likelihood of treatment success.

BDNs are also used in combination with other techniques. For instance, Seixas et al.[97] combined a BDN-based clinical guideline-based system with a rule-based system. Another discipline that uses BDNs to address complex decision-making processes is engineering. For example, Rashid et al.[98] used a BDN model to investigate oil system failure analysis in helicopters, focusing on random failure probabilities. In an example from IT, Christoforou et al.[99] successfully used a BDN model to investigate and determine cloud adaptability for IT services.

To summarise, many studies have explored BDN to support the complex decision-making process under uncertainty. However, no HAC solutions use any form of BDNs[10]. Moreover, we could not find any HAC-related research initiatives that employ BDNs or other stochastic decision models. Hence, to the best of our knowledge, the approach proposed in this paper is the first to construct a BDN model to improve detection capabilities of HAC resource failures and predict whether resource-level HAC failures are manageable based on evaluations of the key characteristics of the HAC resources.

6 | CONCLUSION AND FUTURE WORK

Multiple challenges stand in the way of ensuring the availability of modern EAs. Some of these come from the significant changes in EA deployment patterns, such as moving applications to public clouds. Other challenges arise from changes in the architecture composition of EAs, for instance, from the transition to microservice-based solutions and the integration of new fault-tolerance capabilities (e.g., self-healing) so that the application itself can manage some SPOF components. As such, HACs must continue to evolve to address these challenges while improving the availability of protected applications.

The HAC solution-independent BDN-HAC model introduced in this paper is intended to enable future HACs in this endeavour in three ways. First, the model more accurately detects failures. Second, it enables the prediction of locally manageable resource failures. Third, combining the first and second capabilities, the model improves the decision-making capabilities of HACs.

We evaluated the prediction quality of the BDN-HAC model using the testbed environment and showed that the prediction provided by the improved the availability of the EA when compared with an established HAC solution used as a baseline. Moreover, we also investigated the impact of incomplete data and demonstrated that it significantly affected the prediction quality. We also evaluated the scenario when critical nodes (nodes with a high weight factor) received data and compared this to the scenario when noncritical nodes were supplied with data. The experimental results confirmed that the critical nodes play a vital role in the BDN-HAC model. Similarly, we tested the differences between the existing characteristics and new characteristics of the BDN-HAC model, and showed that the new characteristics improved the prediction quality more than the existing characteristics. Furthermore, we measured the computational overhead and execution time associated with the model.

We found very little overhead (CPU and memory utilisation), and the execution time is also on the order of milliseconds, which is acceptable for the intended use of this model.

In the future, we aim to integrate the BDN-HAC model with HAC solutions that can simplify the decision-making process of HACs, resulting in faster failure mitigation times and reduced overhead. In this paper, we demonstrated that a BDN utility output could represent the probability of managing failure locally at two levels, manageable or unmanageable. This shows that the utility output can be correlated to the different levels of failures. Therefore, it is worth exploring and interpreting the utility outcomes at a granular level, which can then be connected to multiple failure states of a resource. This has the potential to improve the prediction quality in the BDN-HAC model.

Moreover, considering additional HAC characteristics in the BDN-HAC model could improve the detection and prediction qualities. For example, the self-protection ability provided by modern EAs to block cyberattacks and avoid potential downtime caused by such attacks [100,101] can also be added as a new characteristic.

ACKNOWLEDGEMENTS

The second author's work on this project was funded by the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience' and the Assuring Autonomy International Programme.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Wang SS, Franke U. Enterprise IT service downtime cost and risk transfer in a supply chain. *Operations Management Research* 2020: 1–15.
2. Oppenheimer D, Ganapathi A, Patterson DA. Why do Internet services fail, and what can be done about it?. In: . 67 of *USENIX symposium on internet technologies and systems*. Seattle, WA. ; 2003.
3. Nelson R, Staggers N. *Health informatics-e-book: an interprofessional approach*. Elsevier Health Sciences . 2016.
4. Calinescu R, Di Giandomenico F. Special issue on resilient software and software-controlled systems. *Computing* 2021; 103(4): 533-534.
5. Faraji Shoyari M, Ataie E, Entezari-Maleki R, Movaghar A. Availability modeling in redundant OpenStack private clouds. *Software: Practice and Experience* 2021; 51(6): 1218-1241. doi: https://doi.org/10.1002/spe.2953
6. Marcus E, Stern H. *Blueprints for high availability*. Indianapolis, Indiana: John Wiley & Sons . 2003.
7. Schmidt K. *High availability and disaster recovery: concepts, design, implementation*. 22. Springer Science & Business Media . 2006.
8. Quintero D, Balappa V, Bodily S, et al. *IBM PowerHA SystemMirror 7.1.2 Enterprise Edition for AIX*. IBM Redbooks . 2013.
9. Oracle Corporation . Oracle Solaris Cluster System Administration Guide. https://docs.oracle.com/cd/E39579_01/pdf/E39645.pdf; 2015.
10. Somasekaram P, Calinescu R, Buyya R. High-availability clusters: A taxonomy, survey, and future directions. *Journal of Systems and Software* 2022; 187: 111208.
11. Somasekaram P, Calinescu R. Towards a Bayesian prognostic framework for high-availability clusters. In: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion. ACM. ; 2021: 1–8.

12. Ranade DM. *Shared Data Clusters: Scaleable, Manageable, and Highly Available Systems (Veritas Series)*. 9. John Wiley & Sons . 2003.

13. Vogels W, Dumitriu D, Birman K, et al. The design and architecture of the Microsoft Cluster Service-a practical approach to high-availability and scalability. In: Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on. IEEE. ; 1998: 422–431.

14. Somasekaram P. Holistic Modelling Technique for High Availability Software. https://github.com/ps234/HMTHA/; 2021.

15. Nielsen TD, Jensen FV. *Bayesian networks and decision graphs*. Springer Science & Business Media . 2009.

16. Kjaerulff UB, Madsen AL. Bayesian networks and influence diagrams. *Springer Science+ Business Media* 2008; 200: 114.

17. Korb KB, Nicholson AE. *Bayesian artificial intelligence*. CRC press . 2010.

18. Huang Y, Kintala C, Kolettis N, Fulton ND. Software rejuvenation: Analysis, module and applications. In: wenty-Fifth International Symposium on Fault-Tolerant Computing. IEEE. ; 1995: 381–390.

19. Ghosh D, Sharman R, Rao HR, Upadhyaya S. Self-healing systems—survey and synthesis. *Decision Support Systems* 2007; 42(4): 2164–2185.

20. Pillay A, Wang J. Modified failure mode and effects analysis using approximate reasoning. *Reliability Engineering & System Safety* 2003; 79(1): 69–85.

21. Liu HC, Liu L, Liu N. Risk evaluation approaches in failure mode and effects analysis: A literature review. *Expert systems with applications* 2013; 40(2): 828–838.

22. ALVARO GARCíA EDUARDO GILABERT . Mapping FMEA into Bayesian Networks. *International Journal of Performability Engineering* 2011; 7(6): 525.

23. Benz K, Bohnert TM. Impact of Pacemaker failover configuration on mean time to recovery for small cloud clusters. In: 2014 IEEE 7th International Conference on Cloud Computing. IEEE. ; 2014: 384–391.

24. Khan M, Toeroe M, Khendek F. Comparing Pacemaker with OpenSAF for Availability Management in the Cloud. In: Edge Computing (EDGE), 2017 IEEE International Conference on. IEEE. ; 2017: 106–111.

25. Mayerl C, Huner KM, Gaspar JU, Momm C, Abeck S. Definition of metric dependencies for monitoring the impact of quality of services on quality of processes. In: 2007 2nd IEEE/IFIP International Workshop on Business-Driven IT Management. IEEE. ; 2007: 1–10.

26. Xie W, Hong Y, Trivedi K. Analysis of a two-level software rejuvenation policy. *Reliability Engineering & System Safety* 2005; 87(1): 13–22.

27. Hanmer RS. *Patterns for Fault Tolerant Software*. John Wiley & Sons . 2013.

28. Vaidyanathan K, Harper RE, Hunter SW, Trivedi KS. Analysis and Implementation of Software Rejuvenation in Cluster Systems. *SIGMETRICS Perform. Eval. Rev.* 2001; 29(1): 62–71.

29. Veritas Technologies LLC . Cluster Server 7.3 Administrator's Guide - Linux. https://origin-download.veritas.com/resources/content/live/DOCUMENTATION/SFDC/000126860/en_US/vcs_admin_73_lin.pdf; 2017.

30. Critchley T. *High availability IT services*. Auerbach Publications . 2014.

31. Ranade DM. *Shared Data Clusters: Scaleable, Manageable, and Highly Available Systems (VERITAS Series)*. New York: John Wiley & Sons, Ltd . 2002.

32. Salfner F, Lenk M, Malek M. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* 2010; 42(3): 1–42.

33. Artoni F, Delorme A, Makeig S. Applying dimension reduction to EEG data by Principal Component Analysis reduces the quality of its subsequent Independent Component decomposition. *NeuroImage* 2018; 175: 176–187.

34. Reddy GT, Reddy MPK, Lakshmanna K, et al. Analysis of dimensionality reduction techniques on big data. *IEEE Access* 2020; 8: 54776–54788.

35. Sen SD, Adams JA. An influence diagram based multi-criteria decision making framework for multirobot coalition formation. *Autonomous Agents and Multi-Agent Systems* 2015; 29(6): 1061–1090.

36. Fishburn PC. *The Foundations of Expected Utility*. 31. Springer Science & Business Media . 2013.

37. Somasekaram P. *Bayesian Prognostic Framework for High-Availability Clusters*. PhD thesis. University of York, UK; 2021.

38. Renooij S. Probability elicitation for belief networks: issues to consider. *The Knowledge Engineering Review* 2001; 16(3): 255.

39. Gaag v. dLC, Renooij S, Witteman CLM, Aleman BMP, Taal BG. How to Elicit Many Probabilities. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc. ; 1999; San Francisco, CA, USA: 647–654.

40. Schroeder B, Gibson GA. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* 2009; 7(4): 337–350.

41. Schroeder B, Gibson GA. Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?. *ACM Transactions on Storage (TOS)* 2007; 3(3): 8–es.

42. Li Y, Lan Z. Exploit failure prediction for adaptive fault-tolerance in cluster computing. In: . 1 of *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. IEEE. ; 2006: 8–pp.

43. Di Sanzo P, Avresky DR, Pellegrini A. Autonomic rejuvenation of cloud applications as a countermeasure to software anomalies. *Software: Practice and Experience* 2021; 51(1): 46-71.

44. Jammal M, Kanso A, Heidari P, Shami A. Availability Analysis of Cloud Deployed Applications. In: 2016 IEEE International Conference on Cloud Engineering (IC2E). IEEE. ; 2016: 234–235.

45. Beekhof A. Pacemaker 1.1 Configuration Explained An A-Z guide to Pacemaker's Configuration Options. http://clusterlabs.org/pacemaker/doc/en-US/Pacemaker/1.1/pdf/Pacemaker_Explained/Pacemaker-1.1-Pacemaker_Explained-en-US.pdf; 2017.

46. Hsueh MC, Tsai TK, Iyer RK. Fault injection techniques and tools. *Computer* 1997; 30(4): 75–82.

47. Roux J, Beroulle V, Morin-Allory K, et al. High-level fault injection to assess FMEA on critical systems. *Microelectronics Reliability* 2021; 122: 114135.

48. IBM Corp. . Tivoli System Automation for Multiplatforms V4.1: Failover scenarios. 2017.

49. IBM Redbooks. . Building High Availability with SteelEye LifeKeeper for SAP NetWeaver on SUSE Linux Enterprise Server. 2008.

50. SUSE LLC . Enqueue Replication - SAP NetWeaver High Availability on SUSE Linux Enterprise (12). 2016.

51. Novell, Inc. . SAP Applications Made High Available on SUSE Linux Enterprise Server 10. https://www.b1-systems.de/fileadmin/content/whitepaper/Technical_Guide_SLES_HA_for_SAP.pdf; 2014.

52. Dell Technologies Inc. . EMC Mission-Critical Business Continuity for SAP. 2012.

53. Marcot BG. Metrics for evaluating performance and uncertainty of Bayesian network models. *Ecological modelling* 2012; 230: 50–62.

54. Fawcett T. An introduction to ROC analysis. *Pattern recognition letters* 2006; 27(8): 861–874.

55. Microsoft Corporation . High availability for SAP NetWeaver on Azure VMs on SUSE Linux Enterprise Server for SAP applications. 2020.

56. SUSE LLC . SAP on SUSE Linux Enterprise. 2012.

57. BayesFusion L. GeNIe Modeler. *User Manual. Available online: https://support.bayesfusion.com/docs/(accessed on 19 January 2019)* 2019.

58. Chicco D, Jurman G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 2020; 21(1): 1–13.

59. Luque A, Carrasco A, Martín A, Las Heras dA. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition* 2019; 91: 216–231.

60. Gao J, Wang H, Shen H. Task failure prediction in cloud data centers using deep learning. *IEEE transactions on services computing* 2020.

61. Frank A, Yang D, Brinkmann A, Schulz M, Süss T. Reducing false node failure predictions in HPC. In: 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC). IEEE. ; 2019: 323–332.

62. Astekin M, Zengin H, Sözer H. DILAF: A framework for distributed analysis of large-scale system logs for anomaly detection. *Software: Practice and Experience* 2019; 49(2): 153-170.

63. Zhang S, Liu Y, Meng W, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2018; 2(1): 1–29.

64. Xiao J, Xiong Z, Wu S, Yi Y, Jin H, Hu K. Disk failure prediction in data centers via online learning. In: Proceedings of the 47th International Conference on Parallel Processing. ACM. ; 2018: 1–10.

65. Watanabe Y, Otsuka H, Sonoda M, Kikuchi S, Matsumoto Y. Online failure prediction in cloud datacenters by real-time message pattern learning. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. IEEE. ; 2012: 504–511.

66. Ashraf RA, Gioiosa R, Kestor G, DeMara RF, Cher CY, Bose P. Understanding the propagation of transient errors in HPC applications. In: SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE. ; 2015: 1–12.

67. Platini M, Ropars T, Pelletier B, De Palma N. CPU overheating prediction in HPC systems. *Concurrency and Computation: Practice and Experience* 2021; 33(13): e6231.

68. Das A, Mueller F, Siegel C, Vishnu A. Desh: deep learning for system health prediction of lead times to failure in HPC. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. ACM. ; 2018: 40–51.

69. Baldoni R, Montanari L, Rizzuto M. On-line failure prediction in safety-critical systems. *Future Generation Computer Systems* 2015; 45: 123–132.

70. Dangut MD, Jennions IK, King S, Skaf Z. Application of deep reinforcement learning for extremely rare failure prediction in aircraft maintenance. *Mechanical Systems and Signal Processing* 2022; 171: 108873.

71. Wang G, Xu T, Tang T, Yuan T, Wang H. A Bayesian network model for prediction of weather-related failures in railway turnout systems. *Expert systems with applications* 2017; 69: 247–256.

72. Bottone S, Lee D, O'Sullivan M, Spivack M. Failure prediction and diagnosis for satellite monitoring systems using Bayesian networks. In: MILCOM 2008-2008 IEEE Military Communications Conference. IEEE. ; 2008: 1–7.

73. Torres-Toledano JG, Sucar LE. Bayesian networks for reliability analysis of complex systems. In: Ibero-American Conference on Artificial Intelligence. Springer. ; 1998: 195–206.

74. Pitakrat T, Okanović D, Hoorn vA, Grunske L. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* 2018; 137: 669–685.

75. Leangsuksun C, Liu T, Rao T, Scott S, Libby R. A Failure Predictive and Policy-Based High Availability Strategy for Linux High Performance Computing Cluster. In: The 5th LCI International Conference on Linux Clusters: The HPC Revolution. Citeseer. ; 2004: 18–20.

76. Lee YJ, Kim HY, Lee CH. A Stochastic Availability Prediction Model for Head Nodes in the HA Cluster. In: 22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008). IEEE. ; 2008: 157–161.

77. Cheng FT, Wu SL, Tsai PY, Chung YT, Yang HC. Application cluster service scheme for near-zero-downtime services. *Proceedings - IEEE International Conference on Robotics and Automation* 2005; 2005(April): 4062–4067.

78. Leander J, Honfi D, Ivanov OL, Björnsson Í. A decision support framework for fatigue assessment of steel bridges. *Engineering Failure Analysis* 2018; 91: 306–314.

79. Weber P, Suhner MC. An application of Bayesian Networks to the Performance Analysis of a Process. In: In European Conference on System Dependability and Safety (ESRA 2002/lambda-Mu13). Lyon, France. ESRA-ISdF. ; 2002: 266–273.

80. Nielsen JJ, Sørensen JD. Bayesian networks as a decision tool for O&M of offshore wind turbines. In: ASRANet: Integrating Structural Analysis, Risk & Reliability: 5th International ASRANet Conference, Edinburgh, UK, 14-16 June 2010. ASRANet Ltd. ; 2010.

81. Khakzad N. Optimal firefighting to prevent domino effects: Methodologies based on dynamic influence diagram and mathematical programming. *Reliability Engineering & System Safety* 2021; 212: 107577.

82. Fernandez J, Martinez-Selles M, Arredondo M. Bayesian networks and influence diagrams as valid decision support tools in systolic heart failure management. In: Computers in Cardiology, 2004. IEEE. ; 2004: 181–184.

83. Kao HY. Diagnostic reasoning and medical decision-making with fuzzy influence diagrams. *Computer Methods and Programs in Biomedicine* 2008; 90(1): 9–16.

84. Sethi T, Mittal A, Maheshwari S, Chugh S. Learning to Address Health Inequality in the United States with a Bayesian Decision Network. In: AAAI'19/IAAI'19/EAAI'19. AAAI Press. ; 2019.

85. Constantinou AC, Fenton N, Marsh W, Radlinski L. From complex questionnaire and interviewing data to intelligent Bayesian network models for medical decision support. *Artificial intelligence in medicine* 2016; 67: 75–93.

86. Luoma E, Nevalainen L, Altarriba E, Helle I, Lehikoinen A. Developing a conceptual influence diagram for socio-eco-technical systems analysis of biofouling management in shipping–A Baltic Sea case study. *Marine Pollution Bulletin* 2021; 170: 112614.

87. Carriger JF, Parker RA. Conceptual Bayesian networks for contaminated site ecological risk assessment and remediation support. *Journal of Environmental Management* 2021; 278: 111478.

88. Rios Insua D, Couce-Vieira A, Rubio JA, Pieters W, Labunets K, G. Rasines D. An adversarial risk analysis framework for cybersecurity. *Risk Analysis* 2021; 41(1): 16–36.

89. Oonk S, Maldonado FJ. Automated maintenance path generation with Bayesian networks, influence diagrams, and timed failure propagation graphs. In: 2016 IEEE AUTOTESTCON. IEEE. ; 2016: 1–9.

90. Johnson P, Lagerström R, Närman P, Simonsson M. Enterprise architecture analysis with extended influence diagrams. *Information Systems Frontiers* 2007; 9(2): 163–180.

91. Howard RA. Decision analysis: practice and promise. *Management science* 1988; 34(6): 679–695.

92. Liu Y, Shen Y, Chen Y, Gao F. The integrated process of project risk management based on influence diagrams. In: . 2 of *2004 IEEE International Engineering Management Conference (IEEE Cat. No. 04CH37574)*. IEEE. ; 2004: 746–750.

93. Weflen E, MacKenzie CA, Rivero IV. An Influence Diagram Approach to Automating Lead Time Estimation in Agile Kanban Project Management. *Expert Systems with Applications* 2021: 115866.

94. Schurink C, Lucas P, Hoepelman I, Bonten M. Computer-assisted decision support for the diagnosis and treatment of infectious diseases in intensive care units. *The Lancet infectious diseases* 2005; 5(5): 305–312.

95. Seixas FL, Zadrozny B, Laks J, Conci A, Saade DCM. A Bayesian network decision model for supporting the diagnosis of dementia, Alzheimer's disease and mild cognitive impairment. *Computers in Biology and Medicine* 2014; 51: 140–158.

96. Neapolitan R, Jiang X, Ladner DP, Kaplan B. A primer on Bayesian decision analysis with an application to a personalized kidney transplant decision. *Transplantation* 2016; 100(3): 489.

97. Carvalho CM, Christina D, Saade M, Conci A, Seixas FL, Laks J. A clinical decision support system for aiding diagnosis of Alzheimer's disease and related disorders in mobile devices. In: 2017 IEEE International Conference on Communications (ICC). IEEE. ; 2017: 1–6.

98. Rashid H, Place C, Mba D, et al. Helicopter MGB oil system failure analysis using influence diagrams and random failure probabilities. *Engineering Failure Analysis* 2015; 50: 7–19.

99. Christoforou A, Andreou AS. A cloud adoption decision support model using influence diagrams. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer. ; 2013: 151–160.

100. Claudel B, De Palma N, Lachaize R, Hagimont D. Self-protection for distributed component-based applications. In: Symposium on self-stabilizing systems. Springer. ; 2006: 184–198.

101. De Palma N, Hagimont D, Boyer F, Broto L. Self-protection in a clustered distributed system. *IEEE Transactions on Parallel and Distributed Systems* 2011; 23(2): 330–336.

How to cite this article: Somasekaram P., and Calinescu R. (2022), Predicting locally manageable resource failures of high availability clusters.