Received: 1 March 2023 | Revised: 10 July 2023 | Accepted: 13 July 2023

DOI: 10.1002/qre.3414

RESEARCH ARTICLE

WILEY

Availability analysis of redundant and replicated cloud services with Bayesian networks

Otto Bibartiu[1] | **Frank Dürr**[1] | **Kurt Rothermel**[1] | **Beate Ottenwälder**[2] | **Andreas Grau**[2]

[1]University of Stuttgart, Institute for Parallel and Distributed Systems (IPVS), Stuttgart, Germany

[2]Robert Bosch GmbH, Stuttgart, Germany

Correspondence
Otto Bibartiu, University of Stuttgart, Institute for Parallel and Distributed Systems (IPVS), Universitätsstrasse 38 Stuttgart, Germany.
Email: otto.bibartiu@ipvs.uni-stuttgart.de

Funding information
Robert Bosch GmbH

Abstract

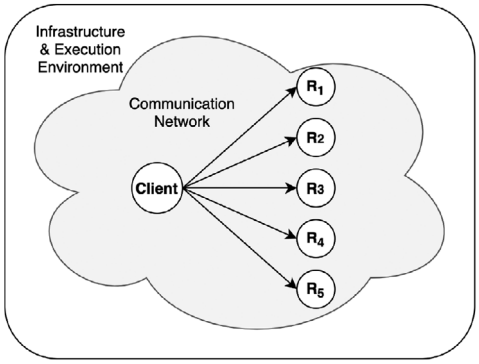
Due to the growing complexity of modern data centers, failures are not uncommon any more. Therefore, fault tolerance mechanisms play a vital role in fulfilling the availability requirements. Multiple availability models have been proposed to assess compute systems, among which Bayesian network models have gained popularity in industry and research due to its powerful modeling formalism. In particular, this work focuses on assessing the availability of redundant and replicated cloud computing services with Bayesian networks. So far, research on availability has only focused on modeling either infrastructure or communication failures in Bayesian networks, but have not considered both simultaneously. This work addresses practical modeling challenges of assessing the availability of large-scale redundant and replicated services with Bayesian networks, including cascading and common-cause failures from the surrounding infrastructure and communication network. In order to ease the modeling task, this paper introduces a high-level modeling formalism to build such a Bayesian network automatically. Performance evaluations demonstrate the feasibility of the presented Bayesian network approach to assess the availability of large-scale redundant and replicated services. This model is not only applicable in the domain of cloud computing it can also be applied for general cases of local and geo-distributed systems.

KEYWORDS

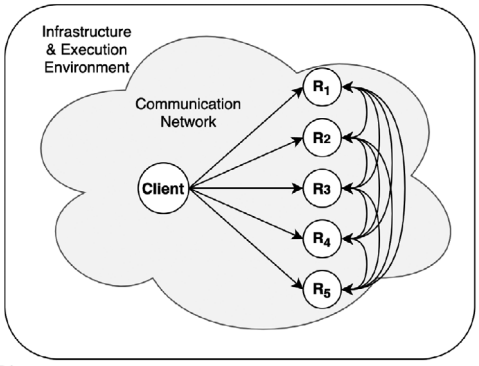
availability analysis, Bayesian networks, fault tolerance, redundancy, replication

1 | INTRODUCTION

Due to the growing complexity of modern data centers, failures are not the exception anymore; they are the norm.[1] For example, the OVHcloud data center incident in 2021 led to the unavailability of multiple online businesses,[2] while the Facebook outage in late 2021, caused by a miss-configuration of the backbone routers,[3] led to an estimated loss of 65 million dollars in revenue.[4] Cloud operation teams and reliability engineers employ fault tolerance techniques



(A) Redundant system represented by an independent set of instances which all ofer the same service.



(B) Stateful replication which necesitates communication between replicas to aggree upon the state of the service.

FIGURE 1 To assess the availability of a redundant or replicated service, one needs to consider the reachability of the instances through the communication network, as well as their fault dependencies with the execution context.

to mask faults through redundancy or replication, by deploying multiple instances of the same service to increase availability. These instances are not fault independent. They normally share common cause failures with the surrounding execution environment and communication network, raising the question if fault tolerance measures meet the availability requirements. To answer this question, this paper proposes a novel Bayesian network modeling approach to assess the availability of redundant and replicated cloud services in presence of network and common-cause failures.

This work distinguishes between the terms redundant and replicated cloud services to address two different modeling semantics with respect to service communication, which can lead to different availability outcomes. In a broader sense, redundancy implies independent service instances (copies) that work in parallel. Redundant services can be stateful or stateless. For example redundant domain-name-system (DNS) servers are stateful, where multiple DNS instances can independently serve client requests. Stateless redundant services are Amazon Web Service (AWS) Lambda and Azure Functions, which are part of the Function-as-a-Service (FaaS) layer. In contrast, replication always involves stateful services that implement a replication protocol to maintain the desired degree of state consistency between the instances. Examples of such systems are replicated databases,[5–7] and distributed locking services.[8] These instances need to communicated with each other at some point in time as supposed to instances of a redundant service.

The ISO/IEC/IEEE International Standard on Systems and Software Engineering defines availability as the "degree to which a system or component is operational and accessible when require".[9] Similarly, we refer to availability as the likelihood of a cloud service to be reachable and operational (up) when required. Figure 1 exemplifies the difference in failure modes when assessing the availability of a redundant or replicated services. Common cause failures and cascading faults in the infrastructure can simultaneously lead to the unavailability of multiple service instances. Network faults might lead to network partitioning, which renders services instances unreachable for client requests or segmenting the instances of a replicated service into groups that cannot agree upon the next states. For example, Figure 1A shows a redundant services. A client application regards the redundant service available as long as it can reach at least one of the instances. In contrast, Figure 1B depicts a replicated service, which has the overhead of inner-replica communication due

the necessity of implementing a replication protocol. So, the replicated service is reachable as long as at least one working instance is reachable by the client, and the instance can communicate with sufficient remaining instances to reach the required quorum size, that is, to correctly implement the replication protocol. As a result, this communication overhead might involve more network components that form an additional source for potential failures, which we need to account for in our availability model.

As Michael R. Lyu noted,[10] it is not sufficient to assess the reliability or availability of a software system in isolation. It is important to also consider the execution (operational) environment, in order to create accurate availability models. However, while researchers acknowledge the significance of infrastructure and communication faults,[11,12] they usually model either the infrastructure,[13–15] or the communication[16–19] part of a system. Moreover, with the advent of cloud computing, reliability engineers face the challenge of modeling the availability of large-scale cloud services. Especially with the introduction of FaaS in cloud computing and NoSQL databases, such as Cassandra, the number of instances per service has grown in the hundreds.[20] Consequently, a high number of components lead to an increase in structural complexity, making many availability models impractical or render them infeasible to model large-scale cloud services.

Consequently, in order assess the availability of today's cloud services, we need holistic availability models that can model large-scale replicated and redundant cloud services while simultaneously accounting for cascading and common-cause failures of the network and infrastructure environment. This paper addresses this problem by proposing a Bayesian network availability model. Bayesian networks have proven helpful in computing the availability of complex systems since they provide a powerful modeling formalism to express complex fault dependencies and uncertainty between components.[21–23] They support a rich set of efficient inference algorithms suitable for fault diagnostic[15] and availability prediction. Moreover, with the introduction of scalable Bayesian network structures,[24,25] we argue that Bayesian networks are a good fit to assess large-scale redundant and replicated cloud services.

This work provides the following contributions.

1) We introduce a high-level modeling formalism to describe complex redundant and replicated cloud services at any preferred level of infrastructure and network granularity, since manually building a Bayesian network availability model of large-scale services can become tiresome, time-consuming, and error-prone (this model gets then translated into the Bayesian network model later).
2) We explain step-by-step how to address the modeling challenges of implementing a Bayesian network model that considers cascading infrastructure and network communication failures.
3) Especially for replicated services, we solve the modeling challenge of addressing network partitioning failures, while also considering a flexible range of fault tolerance semantics like voting and weighted-voting based replication.
4) We also propose a translation procedure that transforms the high-level model into the proposed Bayesian network availability model automatically.
5) Finally, we provide evaluations that demonstrate the feasibility of building and assessing large-scale cloud services models with hundreds of infrastructure components and service instances.

The remainder of this paper is structured as follows: In Section 2, we introduce our system assumptions. Next, in Section 3, we formulate our high level availability model. Afterward, in Section 4, we show how to build the Bayesian network available model. In Section 5, we evaluate the performance of our Bayesian network approach to model large scale services. Next, in Section 6, we discuss the results and suggest future work topics. In Section 7, we present related work on availability modeling of replicated systems. Finally, in Section 8, we conclude this paper.

2 | SYSTEM MODEL

The proposed availability model considers redundant or replicated distributed (cloud service) systems as a set of instances. Instances are assumed to run on virtual or physical hosts, placed within the infrastructure of one or more data centers, and linked by a communication network. The network is assumed to consist of components such as switches, routers, and middleboxes, for example, firewalls, which are placed within the same infrastructure as the hosts themselves.

Specifically, redundant services can be stateless or stateful services, where the stateful service does not replicate its state. Replicated services always refer to stateful services where state is replicated.

A replicated service is available when sufficient replicas are available. Conversely, if too many replicas are unavailable, that is, have crashed or are not reachable, the service is considered unavailable at the time of the request. A quorum is a

certain set of k-out-of-n redundant instances that need to be available to provide a particular service function. Note that different functions such as reading or writing a data object can have different quorum sizes, depending on the replication protocol. Therefore, in this work, service availability implicitly refers to the availability of a specific service function or operation.

The model considers two types of communication patterns. For redundant services, we assume that a client only needs to communicate with one instance to issue its request. For replicated services, it is also sufficient for a client to communicate with one instance to initiate the request. However, that instance needs to be able to communicate with sufficient remaining instances to agree upon the result of the client's request. The exact fault tolerance semantics for redundant and replicated services is flexible and can be defined by the reliability engineer as part of the system description.

The hosts and the communication network are part of the infrastructure, which forms a complex component-based system consisting of *infrastructure components*, such as data centers, racks, power supplies, virtual machines, and network appliances. The model assumes that hard – and software – components, including the service instances, have a crash-recovery model. As soon a component encounters a failure, it crashes and stops, and recovers eventually. Each component in the infrastructure has its probability of failing by its own without external influence.

Moreover, the model assumes that infrastructure components have fault relations, representing potential common causes of failures. These fault relationships can form a cause-effect chain, where the failure of one component is the cause of failure of another component, essentially propagating the failure through the infrastructure, until it eventually leads to the failure of the cloud service, that is, cascading failure. In order to formalize the relationship between two directly fault dependent components, the model assumes that the dependence can be described by means of a static fault tree.[26]

Client applications and instances can communicate with each other by exchanging messages via the communication network. The network is composed of network components forming a network graph. The end-to-end communication, that is, channels, between instances and clients can be synchronous or asynchronous and implemented by one or more redundant network routes. A channel crashes when there is no route in the network to connect the two endpoints, and a route becomes unavailable when at least one network component along the route crashes. Client applications might be placed outside of the known infrastructure. In this case, the model considers the paths starting from the network appliance that constitutes the entry point of the data center; or, if the client application is within the data center, its host. Moreover, we assume the exists some dedicated network components, for example, firewalls or load balancers, that act as *gateways*, that is, entry points, for clients applications to communicate with the service.

A particular placement of instances to virtual or physical hosts is called a *deployment* and known beforehand. Instances do not migrate. If an instance crashes, it does not recover on a different host. It recovers back at its former host. Hence, if a host crashes, all its instances can recover when the host recovers. The model makes no restrictions on the number of instances per host. Multiple instances can run on the same host. In the case of replication, the model does not assume the concurrency control method or the particular replication protocol. Either at any given point in time there are enough replicas *up* and *reachable* to agree upon the results of a client's request, or too many replicas crashed or are unreachable, such that the remaining replicas cannot form a quorum for any client request, resulting into the unavailability of the service.

3 | HIGH LEVEL MODEL DESCRIPTION

This section will address the modeling challenge of building a Bayesian network model to infer the availability of a cloud service in the presence of cascading infrastructure and network faults. To ease the modeling process, we present a high-level model description first, which we later translate to a Bayesian network. The model contains three basic sub-models: a failure model for the infrastructure, a model for the network, and a model to describe the fault-tolerance semantics of the service. This provides the advantage to choose the component granularity of the system. First, we begin with the basic unit of our model, a component.

Definition 3.1 (Component). A component $C \in \boldsymbol{C}$, from the finite set of all components of the system $\boldsymbol{C} = \{C_1, C_2, \dots\}$, is an indivisible hard or software entity with the states $\{F, T\}$, and a probability distribution $P(C = F) = q_i$ to observe the component as faulty (unavailable) and $P(C = T) = 1 - q_i$ to observe the component as operational or working (up).

The set $\boldsymbol{I} = \{I_1, \dots, I_n\} \subset \boldsymbol{C}$ are instances of the service. The remaining components are infrastructure and network components.

Components might have fault dependencies between themselves. We describe these fault dependencies as a direct acyclic graph (DAG).

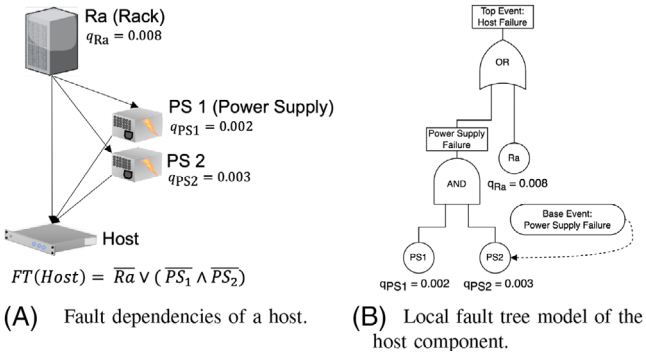


FIGURE 2 Example excerpt of a fault dependency graph of a host and its fault tree representation defined by the function FT(host).

Definition 3.2 (Fault Dependency Graph). Given the set of all components $\boldsymbol{C}$, the model defines the fault dependency graph as a DAG $G_{\text{FD}} = (\boldsymbol{C}, E_{\text{INF}}, FT)$, with edges $E_{\text{INF}} \subseteq \boldsymbol{C} \times \boldsymbol{C}$, and an associated (static) fault tree model (FT) for every component in $\boldsymbol{C}$.

Directed edges are tuples (C_i, C_j), where C_i is said to be a parent component of C_j, and C_j is said to be a child component of C_i. These edges can also define a *contained-in* relation, to signify that one component is contained within another.

In order to express complex component dependencies, $FT(C_i)$ contains the definition of a static fault tree that describes the fault semantics of a component C_i as a function of its parent components. $FT(C_i)$ has as the top event (TE) the failure of component C_i and as base events C_i's parents components. To illustrate how to apply FT, Figure 2A shows the excerpt of a fault dependency graph consisting of a host that depends on its rack, and two redundant power supplies. The fault dependency graph encodes the external conditions when the host fails. In this case, the host fails if the rack fails, or both power supplies stop working. FT(host) encodes this failure relation at the host component, as shown in Figure 2A, leading to the corresponding fault tree representation shown in Figure 2B. This fault tree has the power supplies and the rack as basic input events and the host failure as the TE. The hosts fails when the rack fails, or both power supplies fail, represented by OR gate at the TE and the AND gate at the basic events of the power supplies. Note that the fault dependency model is a DAG, disallowing cyclic fault dependencies since it leads to cycles in the final Bayesian network graph, which is not allowed by definition.

To account for communication faults, the model needs a representation of the network. Network components represent network appliances such as switches, routers, load-balancers, and firewalls. Consequently, the failure of related infrastructure components can influence the failure of a network component, which can lead to communication failures. Unlike the fault dependency graph, the network graph can have cycles.

Definition 3.3 (Network Graph). Given a set of hosts $H \subset \boldsymbol{C}$, a set of network components $N \subset \boldsymbol{C}$, and their union $\boldsymbol{C}_{\text{NET}} = H \cup N$, the network is a graph $G_{\text{NET}} = (\boldsymbol{C}_{\text{NET}}, E_{\text{NET}})$ with unidirectional edges, where the edges $E_{\text{NET}} \subseteq \boldsymbol{C}_{\text{NET}} \times \boldsymbol{C}_{\text{NET}}$ define the communication links between any two network components.

With this graph notion, reliability engineers can decide the granularity of the network model. Suppose they have little or no knowledge of the network. In that case, they can represent the network as 'one switch' connecting all instances, aggregating all potential failure probabilities as one value for one *super* component. However, they can also describe more complex network graphs if they have ample knowledge, which improves the model w.r.t. a more realistic representation of the actual network.

The final system description of the cloud service is the unification of the above model definitions.

Definition 3.4 (High-level System Model). A system

$$S = (\boldsymbol{C}, Q, G_{\text{FD}}, G_{\text{NET}}, D, P, \mathcal{G}, c)$$

is a eight-tuple consisting of the following elements:

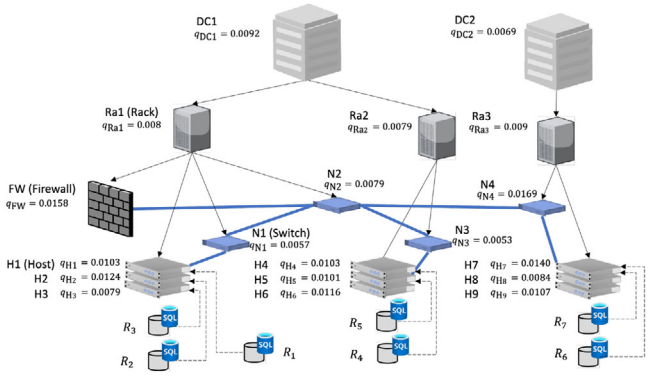


FIGURE 3 Database management system example.

- $\boldsymbol{C}$ The set of all infrastructure, network components and instances.
- Q The fault tolerance model defined as a path set of instances $Q = \{Q_1, \ldots, Q_m\} \subseteq 2^I$.
- G_{FD} The fault dependency graph.
- G_{NET} The network graph.
- D The association of instances to hosts $D : \boldsymbol{I} \rightarrow H$.
- P The function of all fault probabilities of the components in $\boldsymbol{C}$.
- $\mathcal{G}$ The set of network components that act as entry point for client applications to establish a communication channel with the instances of the services. $\mathcal{G} \subseteq C_{\text{NET}}$.
- c A Boolean value $c \in \{false, true\}$ to indicate if the service is redundant or replicated.

The parameter Q defines all instance combinations for which the service is considered in a working state in the presence of instance failures. This generic definition fits redundant as well as replicated service. It implies the enumeration of all valid instance combinations to build Q, building a (minimal) path set of the service instances. For example, let us assume a service has three instances $\boldsymbol{I} = I_1, I_2, I_3$ and the service works as long as two instances are up. As a result, Q is the enumeration of all combinations with at least two instances $Q = \{\{I_1, I_2\}, \{I_1, I_3\}, \{I_2, I_3\}, \{I_1, I_2, I_3\}\}$. This definition provides a flexible way to express a wide range of fault tolerance semantics. However, the enumeration of all instance combinations can become inefficient, especially when considering services with hundreds of instances. To alleviate this burden, we suggest an implicit construction method for k-out-of-n redundancy and voting-based replication models, as well as for the special cases of read-one and write-all replication. For these specific models, we define Q as a tuple (V, t), where $V = (v_1, \ldots, v_n)$ are instance votes and t a threshold value. The availability model will then account for the probability of observing sufficient working instances such that their votes exceed the threshold. For example, we can express the previous examples as $Q = ((1, 1, 1), 2)$ to implement the majority set without enumerating all possible set combinations. If the service has different thresholds, that is, different quorum size requirements, per operation like read-one write-all replication. Read-one would have $t = 1$ for the read operation and write-all $t = n$ for the write operation. The service definition would then refer to one specific operation. Multiple operations can be supported by defining a service model for each operation separately and compute their availability values. At this point, it is up to the reliability engineers how to aggregate the availability of the different operations. They can use the lowest resulting value as a means to assess the probability of the worst-case service model, or they could compute the (weighted) average availability across all operations. Independently of what aggregation method a they chooses, this work shows how to build the availability model accordingly.

Let us exemplify the system model by describing a database management system as shown in Figure 3, which we will then use as a running example for the construction of the Bayesian network model next section.

Figure 3 shows the overall system with its infrastructure and network components that provides the execution environment for the database management system. Although the data center infrastructure might be much larger, we only consider those components which serve the service. The database management system consists of seven replicas I_1 to I_7, placed on hosts within the infrastructure of two data centers. Without loss of generality, the service is available as long as the replicas can form a majority quorum.

Black arrows define fault dependencies between infrastructure components and blue edges represent communication links between network components. Without restrictions, in this example, we assume that a component fails when all its

parent components fail; however, our Bayesian network model will also be capable of modeling more complex component dependencies, such as redundant power supplies. Each component has its own intrinsic fault probability q representing the likelihood of a component failure without external influence. Here, the fault probabilities are sampled from a beta distribution with $\forall i : q_i \sim \mathcal{B}(10, 1000)$.

Finally, the database management system has the following service description:

$$S_{\text{Example}} = (\boldsymbol{C}, Q, G_{\text{FD}}, G_{\text{NET}}, D, P, \mathcal{G}, c)$$

- The set of all components is

$$\begin{aligned}\boldsymbol{C} = \{&DC_1, DC_2, Ra_1, Ra_2, Ra_3, FW, N_1, N_2, N_3, N_4,\\ &H_1, H_2, H_3, H_4, H_5, H_6,\\ &H_7, H_8, H_9, I_1, I_2, I_3, I_4, I_5, I_6, I_7\}\end{aligned}$$

- For the majority set, we need to form all combinations of at least four replicas. $Q = \{\{I_1, I_2, I_3, I_4\}, \{I_2, I_3, I_4, I_5\}, \dots\}$. Or we can use the short hand notation $Q = ((1, 1, 1, 1, 1, 1, 1), 4)$.
- The deployment of replicas to hosts is given by the function D.

$$\begin{aligned} D(I_1) = H_1 \quad D(I_2) = H_2 \quad D(I_3) = H_3\\ D(I_4) = H_4 \quad D(I_5) = H_5 \quad D(I_6) = H_7\\ D(I_7) = H_7\end{aligned}$$

- The fault dependency graph has the following definition.

$$G_{\text{FD}} = (\boldsymbol{C}, E_{\text{INF}}, FT)$$

$$\begin{aligned}E_{\text{INF}} = \{&(DC_1, Ra_1), (DC_1, Ra_2), (DC_2, Ra_3),\\ &(Ra_1, FW), \dots, (D(I_4), I_4),\\ &(D(I_5), I_5), (D(I_6), I_6), (D(I_7), I_7)\}\end{aligned}$$

Here, the instances use the deployment function to identify their host within the fault dependency graph.

In this example, a component automatically fails when its parent component fails. So, FT is a simple mapping of the failure event of the parent component of C, denoted as $pa(C)$.

$$\forall C \in \boldsymbol{C} : FT(C) = \bigwedge_{C_i \in pa(C)} (C_i = F)$$

If a component has no parent, for example, DC_1, then pa returns the empty set.

- The network graph has the following form.

$$G_{\text{NET}} = (C_{\text{NET}}, E_{\text{NET}})$$

$$C_{\text{NET}} = \{FW, N_1, N_2, N_3, N_4, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8, H_9\}$$

$$\begin{aligned}E_{\text{NET}} = \{&\{FW, N_2\}, \{N_2, N_1\}, \{N_2, N_3\}, \dots,\\ &\{N_4, H_7\}, \{N_4, H_8\}, \{N_4, H_9\}\}\end{aligned}$$

- The fault probabilities of observing the components as unavailable are:

$$P(DC_1 = F) = 0.0092 \quad P(DC_2 = F) = 0.0069 \ldots$$

$$P(H_8 = F) = 0.0084 \quad P(H_9 = F) = 0.0107$$

  For the sake of readability, we assume that instances do not fail due to intrinsic faults. Hence, they have an availability of one.
- The entry point for client applications is the firewall: $\mathcal{G} = \{FW\}$
- With $c = true$, the model will consider communication between the instances, describing a replicated service.

For example, the final model would address failure modes where rack Ra_1 would fail, which leads to the failure of all its built-in components. This includes its hosts H_1 to H_3, the firewall, and the switches N_1 and N_2 to fail as well. As a result, the replicas I_1 to I_3 would also fail since Ra_1 is a common cause of failure here. The Bayesian network model compactly encodes all combinations of component state and their probabilities, for which the service is considered available, as part of its qualitative representation, without enumerating all potential failure combinations explicitly.

4 | BAYESIAN NETWORK MODEL

The translation of the high-level service model into a Bayesian network consists of three steps. First, it builds a Bayesian network model of the fault dependency graph. Afterward, it extends the initial Bayesian network with the failure model for inner-replica communication when considering replicated services. The third step finalizes the Bayesian network model by including the failure model for the client-to-instance communication. This modeling approach is novel insofar it can address network partitioning failures, which defines the availability of the service as a function of the channels between instances. For instance, in the case of replicated services with voting-based replication, instead of building a model that accounts for at least k-out-of-n working instances, we build a model where we can infer the probability that for any reachable instance, there are at least (k-1)-out-of-(n-1) working channels connected to the remaining working instances.

4.1 | Background

We will use the Bayesian network representation of fault tree gates throughout the modeling process. This section provides the necessary background to understand fault trees and their equivalent Bayesian network notation. Readers familiar with this notation are free to skip this subsection.

There are three basic gate types that have all fault tree variants in common: the AND, OR, and the k-out-of-n voting gate.[27] Bobbio et al.[28] introduced the general approach to represent fault tree gates with the help of Bayesian networks. This work will use the translation concepts as templates to construct the proposed Bayesian network availability model.

A discrete Bayesian network[29] is a DAG $G = (X, E)$ that represents a joint probability distribution $P(X)$ over the set of discrete random variables $X = \{X_1, X_2, \ldots X_n\}$. The term *variable* or *node* are used interchangeably to denote the vertices of the Bayesian network graph. For every edge $(X_i, X_j) \in E$ between the nodes X_i and X_j, X_i is said to be a parent node of X_j, and X_j is a child node of X_i. Each variable has a conditional probability distribution $P(X_i = x_i | \text{pa}(X_i))$ encoded as a conditional probability table (CPT). The CPT contains the probability to observe a certain state $X_i = x_i$ given the observed states of its parent nodes denoted by parent function $\text{pa}(X_i) = \{X_p : \forall (X_p, X_i) \in E\}$. Nodes without parents are called root nodes and have an a prior probability distribution $P(X_i = x_i)$.

A Bayesian network entails a full joint probability distribution compactly as the product of all the nodes' conditional probability distributions:

$$P(X) = \prod_{x \in X} P(x | \text{pa}(x)) \tag{1}$$

With the help of the joint probability distribution, one can use inference to compute the posterior distribution $P(Y|X')$ of some query $Y \subset X$ of uncertain variables from a given subset $X' \subset X \backslash Y$ of observations of the remaining variables.

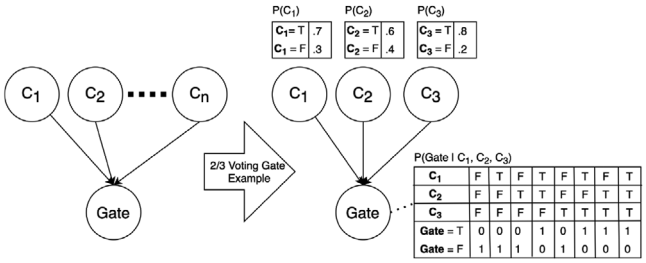


FIGURE 4 Basic Bayesian network to represent the fault tree's AND/OR, or k-out-of-n voting gates (left). Example instance of a Bayesian network k-out-of-n model (right).

Figure 4 (left side) shows the main Bayesian network structure to realize the AND/OR and the k-out-of-n voting gate. The basic structure has n components C_1 to C_n with prior probabilities represented by their eponymous binary random variables with states $\{F, T\}$, observing the component either faulty or available, respectively. The individual semantics of the gate types are encoded within the CPT of the Gate node.

4.1.1 | AND model

For every state combination of the parent nodes, we define Gate $= F$ if all parent nodes are observed to be in state F. Hence, the conditional probability distribution for the Gate node has the following short-hand definition:

$$\begin{aligned} P(\text{Gate} = T | \forall C \in pa(\text{Gate}) : C = F) &= 0 \\ P(\text{Gate} = F | \forall C \in pa(\text{Gate}) : C = F) &= 1 \\ P(\text{Gate} = T | \exists C \in pa(\text{Gate}) : C = T) &= 1 \\ P(\text{Gate} = F | \exists C \in pa(\text{Gate}) : C = T) &= 0 \end{aligned} \tag{2}$$

4.1.2 | OR model

For every state combination of the parent nodes, we will observe Gate $= F$ if at least one parent node is in state F.

$$\begin{aligned} P(\text{Gate} = T | \forall C \in pa(\text{Gate}) : C = T) &= 1 \\ P(\text{Gate} = F | \forall C \in pa(\text{Gate}) : C = T) &= 0 \\ P(\text{Gate} = T | \exists C \in pa(\text{Gate}) : C = F) &= 0 \\ P(\text{Gate} = F | \exists C \in pa(\text{Gate}) : C = F) &= 1 \end{aligned} \tag{3}$$

4.1.3 | k-out-of-n model

For example, Figure 4 (right side) shows an instance of the k-out-of-n model for a two-out-of-three voting gate. The k-out-of-n voting gate triggers a fault event when k or more input events are in a faulty state. Hence, the CPT of the Gate node has to count how many parent nodes are in the state F. This is done for each column. We set the probability to 1 for state T if less than k parent nodes are in the state F, or set the probability of F to 1 if k or more parent nodes are in the state F. Formally, the conditional probability distribution of the k-out-of-n model has the following definition:

$$\forall c_1, \ldots, c_n \in \{F, T\}^n$$

ALGORITHM 1 Generating the service model.

```
1:  procedure CreateServiceModelS
2:      (C, Q, G_FD, G_NET, D, P, G, c) ← S
3:      BN = (X, E) with X = {} and E = {}
4:      X = X ∪ S
5:      CreateFaultGraph(BN, G_FD, D, P)
6:      if c then
7:          ReplicatedService(BN, Q, G_NET, G, D)
8:      else
9:          RedundantService(BN, Q, G_NET, G, D)
10:     end if
11:     return BN
12: end procedure
```

$$P(\text{Gate} = F|c_1, \dots, c_n) = \begin{cases} 1 & \sum_{i=1}^{n} \mathbf{1}_F(c_i) \geq k \\ 0 & \text{otherwise} \end{cases}$$

$$P(\text{Gate} = T|c_1, \dots, c_n) = 1 - P(\text{Gate} = F|c_1, \dots, c_n) \tag{4}$$

where $\mathbf{1}_F(x)$ is an indicator function such that

$$\mathbf{1}_F(x) := \begin{cases} 1 & \text{if } x = F, \\ 0 & \text{otherwise.} \end{cases}$$

4.2 | Transformation overview

Algorithm 1 introduces the pseudo code to build the Bayesian network model based on the high-level service description. Here, the notion $(x, y, z) \leftarrow S$ means that a structure, say S, provides its elements x, y, and z to the outer scope, which is known as pattern matching in the context of functional programming. First we set up a an empty Bayesian network with the node set X and edge set E. Afterward, we add our first node S, which is a binary random variable representing the availability of the service. At the end of the procedure, one can then infer the fault probability, or availability, of the service by computing the marginalization $P(S = F)$, or $P(S = T)$ respectively. The definition of the conditional probability distribution of S follows in the procedures in line 7 or 9.

For any given service model S, we build the Bayesian network availability model of the fault dependency graph with the method CreateFaultGraph in line 5, in order to account for cascading and common cause failures, and then include the concrete service type according to c. If c is true, we include the replicated service model with the method ReplicatedService in line 7, otherwise the procedure builds the redundant service model in line 9. The remainder of this section will introduce each of the three sub-procedures in detail.

4.3 | Fault dependency graph

Given a system model S, the first step in the translation procedure is to build the Bayesian network representation of the fault dependency graph. Perhaps it is not apparent why the fault dependency graph forms the beginning. However, due to the cause-effect semantics of Bayesian networks, it is essential to start with root causes first and then successively attach the effects, which themselves are failure causes for other components. Hence, infrastructure failures form the initial causes of failures.

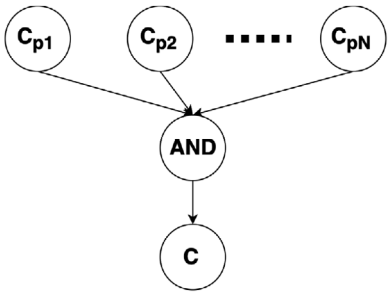


FIGURE 5 AND fault relation between infrastructure components.

4.3.1 | Failure model of a component

A component $C \in \boldsymbol{C}$ fails either because of an intrinsic failure or because of an external fault caused by its parent components. First, we define the general Bayesian network structure of a single component. This structure will then be used as a building block for the upcoming Bayesian network representation of the fault dependency graph.

First, the procedure creates a binary random variable for every component in C with the states $\{F, T\}$, where each variable defines the probability of observing the eponymous component as faulty or available. The procedure applies to each component C the Bayesian network transformation of $FT(C)$ according to,[28] where the fault of C is the TE, and C's parent components are the base events. For example, Figure 5 shows the Bayesian network representation of a component C that expresses its dependability to its parent components C_{p_1} to C_{p_N} as a fault tree with one AND gate. Hence, the CPT uses the previously introduced AND model from Equation (2). A component C can also fail by its intrinsic fault with probability q, which is part of C's CPT definition. The conditional probability distribution of C represents a *noisy*-AND model. Hence, the CPT of C from Figure 5 has the following definition.

$$\begin{aligned} P(C = T|\boldsymbol{AND} = F) &= 0 \\ P(C = F|\boldsymbol{AND} = F) &= 1 \\ P(C = T|\boldsymbol{AND} = T) &= 1 - q \\ P(C = F|\boldsymbol{AND} = T) &= q \end{aligned} \tag{5}$$

4.3.2 | Translating the fault dependency graph

Algorithm 2 repeats the approach mentioned above for each component. It transforms a given fault dependency graph G_{FD} into a Bayesian network. First, the procedure creates a node for every component (line 3). Then, it creates their corresponding Bayesian network fault tree representation defined in $FT(C)$ (line 7), using the building formalism introduced by Bobbio et al. in ref. [28], and then connecting the parent components as base events to the resulting structure at line 9. Finally, we also connect the node of the component that represents the TE with the corresponding component node (line 11). Afterward, it adds the node representation of the instances to the host nodes according to a predefined deployment D (line 15).

Applying Algorithm 2 to the example S_{Example} leads to the preliminary Bayesian network shown in Figure 6. Here, without loss of generality and for the sake of readability, the AND fault relation between all infrastructure components can be simply combined to one node with the noisy AND model of the component. With this simplification, the Bayesian network corresponds in its shape to the fault dependency graph, as illustrated in Figure 3. Moreover, to visually assist the translation procedure, the nodes in Figure 6 are rearranged. All network components are on the left, and all hosts with their processes are on the right side (gray dashed box).

ALGORITHM 2 Building the Bayesian network infrastructure model.

procedure CREATEFAULTGRAPHBN, G_{FD}, D, P
  $(\boldsymbol{C}, E_{\text{INF}}, FT) \leftarrow G_{\text{FD}}$
  for $C \in \boldsymbol{C}$ **do**
    $X = X \cup C$ ▷ Create node C with state $\{F, T\}$
  end for
  for $C \in \boldsymbol{C} \setminus \boldsymbol{I}$ **do**
    $BN_C = FT(C)$ ▷ Create Bayesian network model of $FT(C)$ according to[28]
    for $C_{pj} \in \text{pa}(C)$ **do**
      $E = E \cup (C_{pj}, BN_{C,j})$
    end for
    $E = E \cup (TE(BN_C), C)$
    add CPT to C using P and Equation (5)
    with $q = P(I_i = F)$
  end for
  for $I_i \in I$ **do**
    $E = E \cup (D(I_i), I_i)$
    add CPT to I_i using P and Equation (5)
    with $q = P(I_i = F)$
  end for
  return BN
end procedure

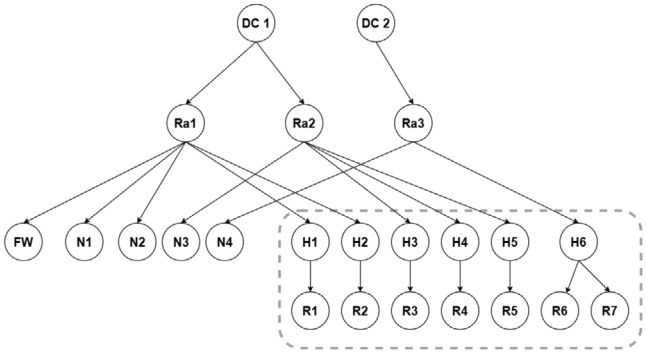


FIGURE 6 Bayesian network infrastructure model of the data management system example.

4.3.3 | Channel model

In order to model service reachability in the presence of network partitioning failures, we need to discuss how to model the probability of observing communication failures with Bayesian networks. Instances and client applications communicate over channels, which is realized as a route along the network graph. The goal of a channel is to assess the accessibility between two instances in the presence of possible network faults. From an availability perspective, when a route fails, because some network component had failed along the route, then a channel can be established along a different if one still exists. Therefore, a channel is considered unavailable, when all potential routes have failed. A channel subsumes the fault probability of observing all routes between the two endpoints as interrupted.

Figure 7 shows the Bayesian network structure that contains the node $C_{I_i-I_j}$, representing the probability of a communication failure between two instances I_i and I_j. For readability, this section refers to $C_{I_i-I_j}$ simply as a *channel node*. A channel node is conditionally dependent on three nodes: an AND node and two nodes for the endpoints of the channel.

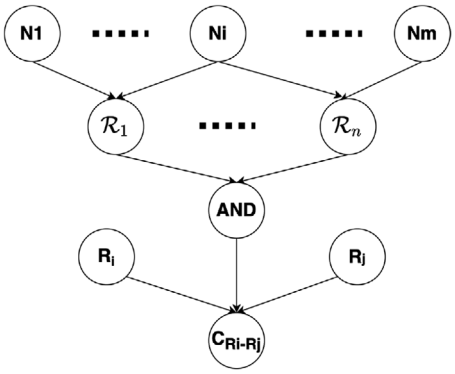


FIGURE 7 Bayesian network representation of a single communication channel.

The AND node represents the failure probability that no route exists, whereas the endpoint nodes represent the failure probability of the corresponding instances. The CPT of the channel node entails an OR model, defining the probability of observing a channel failure when one of the endpoints fails, or no working route exists.

The nodes that define the failure of the endpoints, that is, I_i and I_j, are the node representations of the service instances. However, they could also represent different failure causes that indirectly affect the channel, which could be a client application, for example, the host of the client, or a common endpoint of a second channel. The latter is essential for the replicated service model, to model inner-replica communication.

Finally, nodes R_1 to R_n define the failure probabilities of routes. These route nodes use an OR model for their CPTs and are conditionally dependent on the network components N_1 to N_m that are part of the corresponding route in the network graph. This model also considers correlating route failures when a route shares the same network components. For example, if N_i fails, route R_1 are R_n are interrupted. The same applies when multiple channels share the same routes, respectively.

Algorithm 3 formalizes the construction of a channel as a procedure. Necessary inputs are source C_{src} and destination C_{dst} component and a pair of Bayesian network nodes X_{src} and X_{dst}, which represent the failure causes of the channel's endpoints. As discussed briefly, the model distinguishes between the components for which it computes the routes and the parent nodes that provide the failure causes at the channel's endpoints. The node $AND_{src-dst}$ indicates that the AND node belongs to the channel $C_{src-dst}$, in order to distinguish the AND nodes between multiple channels. First, the procedure computes all routers in the network graph at line 3. Afterward, line 4 to 8 initializes the channel nodes with its parent nodes. Line 9 iterates over the list of routes and determines if the route has existed as a node in the Bayesian network graph or not. If yes, then the corresponding route node is directly added to the channel as shown in line 17. If not, the procedure creates the new route node and connects it with its corresponding network components (lines 10 to 13). The remainder of the procedure finalizes the CPT of the channel node and returns it as a reference.

Without a doubt, the number of routes can get intractably large. In this case, one might resort to simplifying the network graph. That can be done either by aggregating multiple network components, or by considering a limited number of routes – or both. However, while this simplification increases performance, it comes to the expense of model fidelity.

4.4 | Redundant service model

Given the channel model, we can build the model of a redundant service first. Successful communication exists when clients can access sufficient working instances directly. In real-life, a client application will most likely try to connect to one instance, whereas the Bayesian network represents the probability of connecting to any of those instances. Due to the high user-load assumption, we need to account for the likelihood of observing sufficient working instances, even if we need one instance to handle the request.

Algorithm 4 describes how to extend the previously created Bayesian network model of the infrastructure with the redundant service model. We stated in the system model, that a client application can access a service through one or more dedicated network component that act as entry points, that is, gateways, in the network. Therefore, we introduce a new set of binary random variables $K = \{K_i\}_{i=1}^{m}$, with $m = |\mathcal{G}|$, which represents the probability of accessing sufficient instances through the i-th entry point defined in $\mathcal{G}$.

ALGORITHM 3 Routine to create Bayesian network sub-graph for channels.

procedure CREATECHANNEL(BN, G_{NET}, $C_{src} \in C_{\text{NET}}$, $C_{dst} \in C_{\text{NET}}$, $X_{src} \in X$, $X_{dst} \in X$)
  $(X, E) \leftarrow BN$
  routes := compute all paths from C_{src} to C_{dst} in G_{NET}
  $X = X \cup C_{src-dst}$
  $X = X \cup AND_{src-dst}$
  $E = E \cup (AND_{src-dst}, C_{src-dst})$
  $E = E \cup (X_{src}, C_{src-dst})$
  $E = E \cup (X_{dst}, C_{src-dst})$
  for $\mathcal{R}$ in *routes* **do**
    if $\mathcal{R} \notin X$ **then**
      $X = X \cup \mathcal{R}$
      for $C \in \mathcal{R}.components$ **do**
        $E = E \cup (C, \mathcal{R})$
      end for
      add OR model to CPT of $\mathcal{R}$
    end if
    $E = E \cup (\mathcal{R}, AND_{src-dst})$
  end for
  add OR model to CPT of $C_{src-dst}$
  add AND model to CPT of $AND_{src-dst}$
  return $C_{src-dst}$
end procedure

ALGORITHM 4 Implementation of the redundant service model.

procedure REDUNDANTSERVICE$BN, Q, G_{\text{NET}}, \mathcal{G}, D$
  $(X, E) \leftarrow BN$
  $m = |\mathcal{G}|$
  for $i \in [1, m]$ **do**
    $X = X \cup K_i$
  end for
  for $G_i \in \mathcal{G}$ **do**
    for $I_i \in \boldsymbol{I}$ **do**
      $C_{G_i - I_i}$:=CREATECHANNEL(BN, G_{NET}, $G_i, D(I_i), G_i, I_i$)
      $E = E \cup (C_{G_i - I_i}, K_i)$
    end for
    $E = E \cup (K_i, S)$
    add CPT of K_i according to Q.
  end for
  add AND model to CPT of S
end procedure

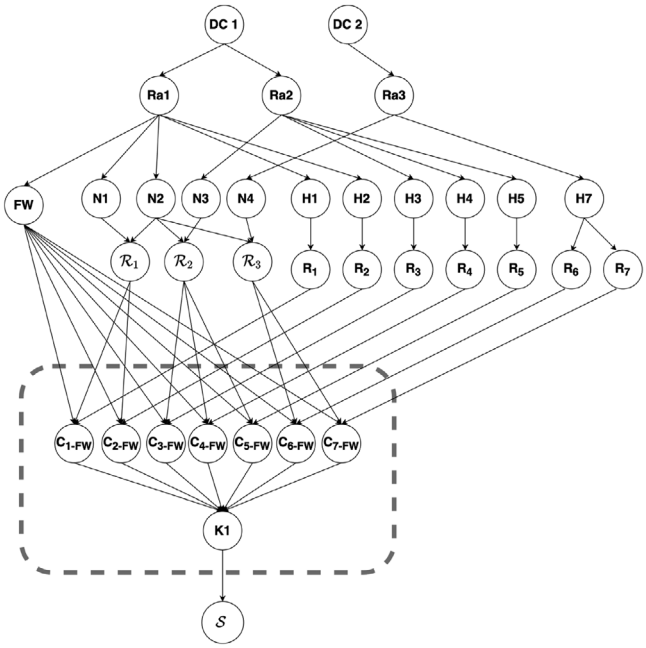


FIGURE 8 The Bayesian network of a redundant service example.

At line 10, the procedure creates the channel nodes for each entry point in the set $\mathcal{G}$ to each instance. The channel creation procedure takes as input the network component that acts as an entry point, the host of the instances as defined by their deployments, and the two nodes that represent the failure of the channel's endpoints. In a follow up step (line 11), all channel nodes that are related to the i-th entry point component connect to one node K_i, which implements the reachability requirement of accessing sufficient instances form that entry point as part of its CPT. For example, if one instance is sufficient for a working service, then each K_i would implement an AND model at line 14, representing the fault probability that the i-th client cannot communicate with any instance at all. A detailed discussion on how to integrate general requirements for K_i at line 14 can be found at the end of this section.

Finally, Algorithm 4 finishes by introducing the final service node $\mathcal{S}$. This node accounts for the probability that no client at any entry point has sufficient working channels to communicate with the instances. Hence, we can compute the probability of a single service failure as the marginal $P(\mathcal{S} = F)$ or its availability $P(\mathcal{S} = T)$ using Bayesian inference.

For instance, Figure 8 shows the Bayesian network model of the example service $\mathcal{S}$ from Section 3, assuming a redundant service. In this example, all clients communicate with the instances via the firewall (represented by node `FW`). There are three routes $\mathcal{R}_1$ to $\mathcal{R}_3$, which are shared by all seven channels, emphasized by the dashed box. Each channel is connected to the firewall node, representing the client. Since there is only one entry point, the set $K = \{K_1\}$ contains one node. For example, assuming the service can tolerate three instance failures, node K_1 implements a four-out-of-seven model (see Equation 4).

4.5 | Replicated service model

For replicated services, we said that clients first send their request to one instance, which then communicates with the remaining instances. This communication pattern subsumes and implements the likelihood of accessing at least one instance that can communicate with sufficient remaining instances. Hence, we will show how to use this communication pattern to encode all possible states in which the instances, or cannot, reach the desired number of votes, for example, quorum size, as defined by the fault tolerance model in Q. Consequently, the final Bayesian network will encode the probability of observing the service in a working state, giving potential infrastructure and communication faults.

Algorithm 5 begins first by modeling the communication channels between instances. It introduces again the set of binary random variables $K = \{K_i\}_{i=1}^{n}$ where $n = |R|$, which represent the failure probability of communicating with an insufficient number of instances when the i-th instance initiates the replication protocol. Hence, every K_i is a child node of $n - 1$ channel nodes (line 12 and 13), since the fault probability of instance R_i is already part of one of the endpoints

ALGORITHM 5 Implementation of the replicated service model.

```
1:  procedure ReplicatedService(BN, Q, G_NET, T, D)
2:      (X, E) ← BN
3:      X = X ∪ S ▷ Create service node S
4:      n = |R|
5:      for i ∈ [1, n] do
6:          X = X ∪ K_i
7:      end for
8:      for (R_i, R_j) in R × R do
9:          if C_{R_i−R_j} ∉ X and C_{R_j−R_i} ∉ X then
10:             C_{R_i−R_j} := CreateChannel(BN, G_NET,
11:                                 D(R_i), D(R_j), R_i, R_j)
12:             E = E ∪ (C_{R_i−R_j}, K_i)
13:             E = E ∪ (C_{R_i−R_j}, K_j)
14:         end if
15:     end for
16:     add CPT for all K_i ∈ K according to Q.
17:     for G_i ∈ G do ▷ Second Step
18:         X = X ∪ G_i
19:         for j=1; j < n; j++ do
20:             C_{G_i−P_j} := CreateChannel(BN, G_NET,
21:                                 G_i, D(R_j), G_i, K_j)
22:             E = E ∪ (C_{G_i−R_i}, S)
23:         end for
24:     end for
25:     add AND model to CPT of S
26: end procedure
```

of the channels. Next, the procedure builds a channel node for every entry point G_i to every instance R_i by using K_i as failure cause(line 21). Instead of directly addressing the failure probability of an instance, the model uses K_i to represent the instance R_i. In the case of a network partitioning, K_i would contain the probability that R_i can still access sufficient processes in its partition.

Finally, node S accounts for the failure probability that no client can access the service through any gateway(line 25). Hence, one can now infer the fault probability, or availability, of the service by computing the marginalization $P(S = F)$, or $P(S = T)$ respectively.

For example, Figure 9 shows the Bayesian network of the database service example, based on the assumption that client applications access the service via the firewall. The left box shows the channel nodes representing the fault probabilities for the communication between clients and service instances. The right box shows the channels of each instance to every other instance. A node K_i has as parent nodes the channel nodes of the i-th instance. Hence, to implement the majority set requirement, one can use a three-out-of-six model for K_i to encode the probability of observing at least three working channels, which implies that the i-th instance is also working.

Next, we discuss in detail how to implement the CPTs of the nodes in K as hinted at line 16.

4.5.1 | Read-one/write-all

Read-one/write-all is a special case in replication since every operation has its particular quorum requirements. We already had a brief introduction on read-one/write-all in the last section. There, we discussed how to implement the service requirements for read $Q_{ro} = 2^R/\emptyset$, and for write quorums $Q_{wa} = \{R\}$. Consequently, each operation needs its own

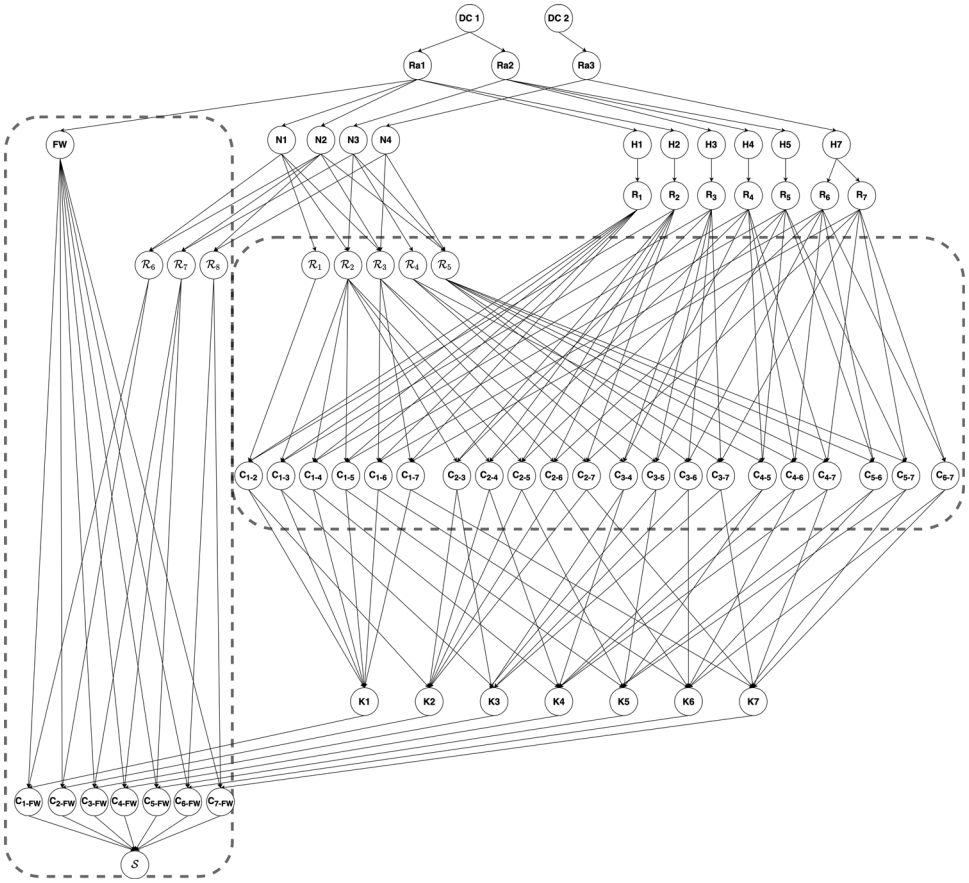


FIGURE 9 The Bayesian network of the indirect communication pattern for the database example.

Bayesian network model to assess its availability individually. Read-one can be modeled by using the redundant service model. Hence, the model uses an AND model for all CPTs of the nodes in K to account for the fault probability that no channel works. In contrast, for write-all, it depends on the system design. One can use either the redundant or the replicated service model. Both models use the OR model for the CPTs of nodes in K, accounting for the fault probability that there is at least one channel faulty to an instance.

4.5.2 | k-out-of-n voting

In voting-based replication, instances have one vote to decide on an incoming operation request. The system is available when it can reach k-out-of-n votes for some request, for example, majority sets require $k = \lfloor \frac{n}{2} \rfloor + 1$ votes. For replicated services that use the indirect communication pattern, the i-th replica is part of the voting process, where it must acquire at least $k - 1$ votes from the remaining $n - 1$ replicas to consider the service as available. Thus, the CPT of K_i implements an $n - k + 1$-out-of-$n - 1$ mode as defined in Equation (4), that is, considering the inverse on how many channel failures can be tolerated.

For redundant services that use the direct communication pattern with n instances, where k instances are sufficient to signify that the service does not fail due to overload, the model implements the CPT of K_i by using an $(n - k)$-out-of-n model. Thus, the system fails if there are more than $n - k$ channels faulty.

4.5.3 | Weighted voting

In weighted voting, individual replicas can have multiple votes. This forms the general case of the normal voting-based approach from above. To reach a potential quorum, the total number of votes that are available by working instances needs

to exceed a given threshold t. As a result, this work extends the k-out-of-n model from Equation (4) to account for the individual vote counts of the replicas. We use the tuple notation for $Q = (V, t)$, where $V = (v_1, \dots, v_n)$ are instance votes and t the threshold value. Given that K_i refers to the i-th instance, the models use v_j to denote the number of votes of the instance at the opposing endpoint of the j-th channel for a given state combination $c_{i-1}, \dots, c_{i-m}$ of the channel nodes connected at K_i. Here, since the i-th instance initiated the replication protocol, we automatically assume that its votes v_i contribute to the request. Hence we reduce the threshold by its votes.

$$\forall c_{i-1}, \dots, c_{i-m} \in \{F, T\}^m$$

$$P(K_i = T | c_{i-1}, \dots, c_{i-m}) = \begin{cases} 1 & \sum_{j=1}^{m} \mathbf{1}_T(c_{i-j}) v_j \geq t - v_i \\ 0 & \text{otherwise} \end{cases}$$

$$P(K_i = F | c_{i-1}, \dots, c_{i-m}) = 1 - P(K_i = T | c_{i-1}, \dots, c_{i-m})$$

For every state combination $c_{i-1}, \dots, c_{i-m}$, the model builds the weighted sum of those channels that are available and checks if the result is above the threshold.

4.6 | Scalability

Bayesian networks are subject to an exponential growth of memory with regard to their CPTs.[30] The CPT of a node has to implement a conditional probability distribution for each state combination of its parent nodes. If the parent nodes are binary, then the number of CPT entries is $O(2^n)$. Hence, all CPTs of K will exhibit an exponential memory growth in the number of instances. We have a similar situation for nodes that represent the availability of routes. Those nodes implement an OR model, which can have multiple network components that represent a route. Assuming a CPT entry is just several bytes large, it is not hard to see that a node with 30 parent nodes will have a CPT with several gigabytes of memory. Therefore, this Bayesian network approach is suitable only for services with up to 30 instances and short network routes; afterward, the memory becomes the limiting factor.

However, this problem can be mitigated for the AND/OR, and k-out-of-n model. Heckerman[25] provides an equivalent AND/OR model that reduces the space complexity to linear, while Bibartiu et al.[24] provide an equivalent (scalable) k-out-of-n model with polynomial complexity. Having these scalable models, we can substitute the existing AND/OR, and k-out-of-n models in the Bayesian network model with their scalable counterparts. Hence, we can overcome the memory limitations for redundant services and voting-based replication models for large services.

5 | EVALUATION

This section provides an in-depth analysis of the performance and modeling feasibility of the presented Bayesian network availability model. The evaluation will analyze the availability, build, and inference performance for redundant and replicated services for an increasing number of instances. All experiments were performed on a 64-bit machine with 64 Intel(R) Xeon(R) CPU E7-4850 v4 at 2.10 GHz and 1 TB of main memory, running Arch Linux 5.13.12 with GCC 11.1.0, Python 3.9.6, and with pgmpy 0.1.7 (the Bayesian network modeling package) and Numpy 1.20.3. Bayesian network inference is performed with approximate and exact inference whenever possible. For approximate inference, we use the forward sampling method, and for exact inference, we used the Lauritzen-Spiegelhalter Algorithm method[31] from the gRain 1.3.2 package.[32,33] Furthermore, we used in all experiments the scalable Bayesian network representations for AND/OR and voting gates by Heckerman[25] and Bibartiu et al.[24] The implementation of the algorithms and evaluation methods for the presented Bayesian network model are available as open source[1].

Moreover, all experiments will consider two different data center infrastructures. The first infrastructure corresponds to the example used in Section 3, which consists of 19 components. The evaluation will refer to this example as the *small*

[1] https://github.com/openclams/bn-availability-model

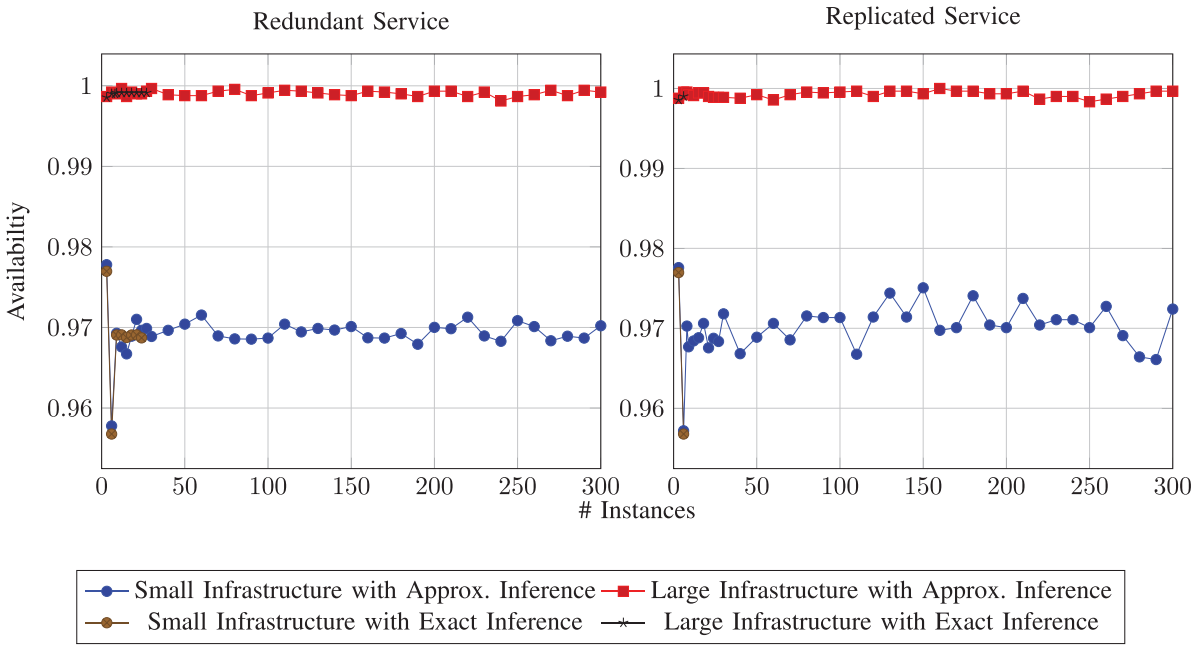


FIGURE 10 The availability results of a service for increasing the number of instances, using approximate and exact inference.

infrastructure. Consequently, the second infrastructure will be called the *large* infrastructure. The large infrastructure consists of three data centers with 40 hosts each, using a random topology of 20 network components to connect hosts and data centers. Moreover, each data center has 100 additional infrastructure components that influence the hosts and the network components. The large infrastructure has in total 440 components. All components in the large infrastructure have an availability value sampled from a beta distribution with $C \sim \text{Beta}(10,000, 1)$, resulting in an average downtime of 1 h during a mission time of 10,000 h. Without loss of generality, we will require that the majority of instances are needed for both service types to be considered available. Other k-out-of-n schemes are also possible, but a different k changes only the content of the corresponding nodes and not the structure of the Bayesian network.

The plot in Figure 10 shows the expected availability for both service types for an increasing number of instances, using the small and large infrastructure, applying approximate and exact inference. Instances were placed in round-robin. We computed the availability for services with up to 300 instances using approximate inference. Exact inference was only possible for up to 27 instances for the redundant service experiments and for up to six instances for the replicated service experiments, independently of the infrastructure size. Approximate inference might vary by nature with every execution. So we compared the results of the exact and approximate inference methods by repeating them 40 times to compute their confidence intervals. As a result, it can be stated with 95% confidence that there is no significant difference in the inference results between the exact and approximate inference methods here.

The availability results between the redundant and the replicated service are similar. The availability decreases up until six instances for the small infrastructure experiments. This is mainly because all instances are placed in the first data center. The follow-up placements also consider the second data center in the small infrastructure for services with seven or more instances. The more instances, the less common-cause failures are shared. However, adding more instances does not lead to higher availability. The higher the distribution of instances, the higher the risk of communication failures since more network components are involved. This limits the availability to a point where the influence of the shared infrastructure outweighs the benefits of replication. Even in the large infrastructure example, where we assume a low average downtime per component, the availability does not converge arbitrarily near to 1.

The plot in Figure 11 shows the mean inference time to compute the presented availabilities. Here we can observe the exponential time increase (linear function in a semi-log plot) of the exact inference method, which contrasts the polynomial time increase (log function in a semi-log plot) of the approximate inference method. There are two main observations. First, the inference time between the redundant and replicated services have different polynomial complexities, and second, the inference time converges independently of the infrastructure size. Clearly, due to the twenty-fold increase of components in the large infrastructure compared to the small infrastructure, the former is slower than the latter for small numbers of instances. However, the number of channels nodes increases with the number of instances.

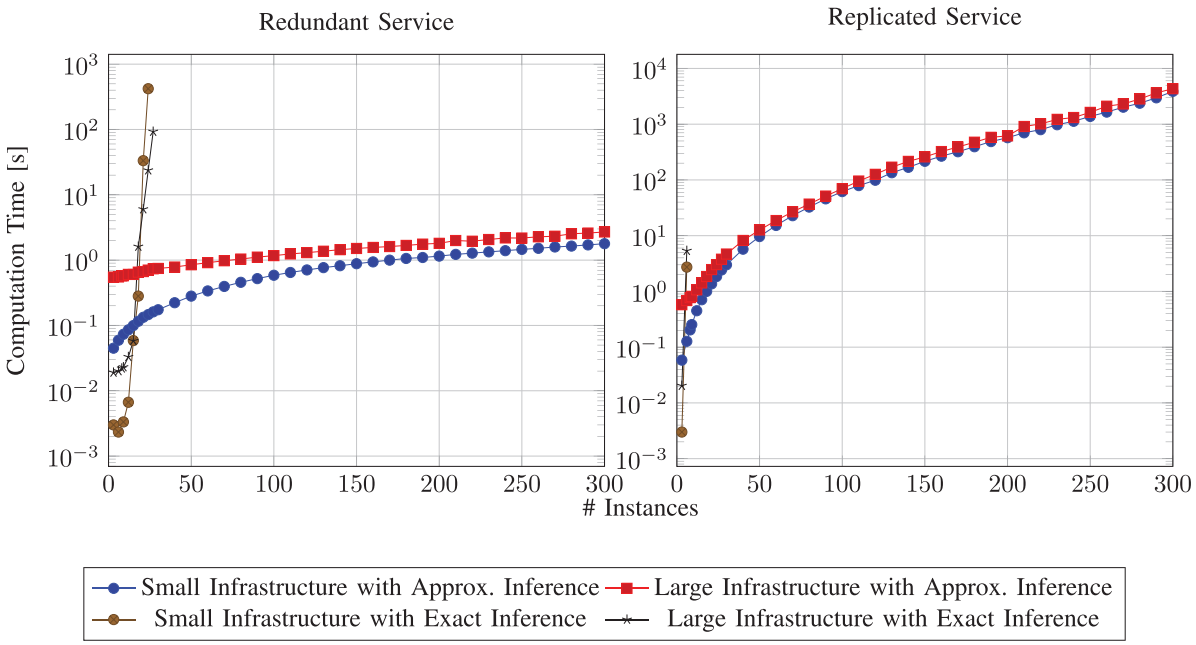


FIGURE 11 The inference time to compute the availability of a service with increasing number of instances for the small and large infrastructure example, using approximate and exact inference.

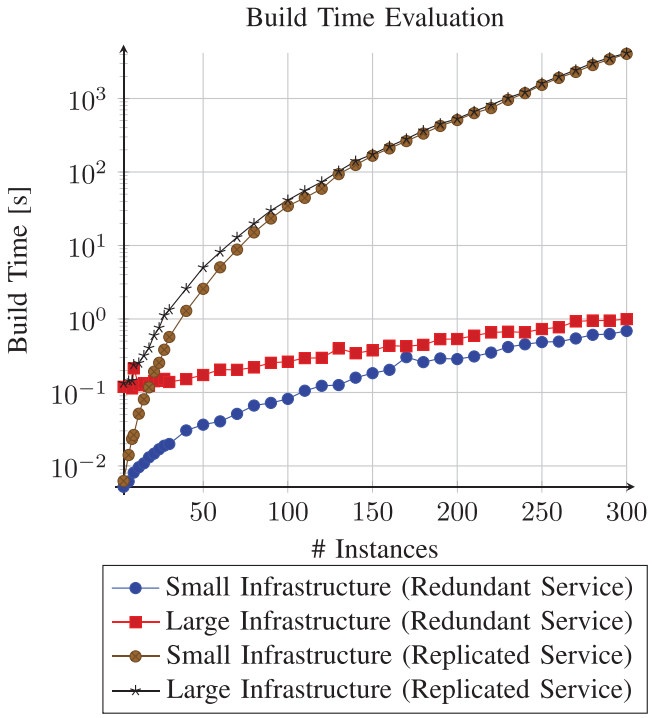


FIGURE 12 Comparing the time to build the Bayesian network model for increasing number of instances.

Hence, the more instances, the more channel nodes. The number of channel nodes outweighs the number of infrastructure components until they become the influencing factor in the computation. The model of the redundant service has a linear increase of channels, whereas the replicated service has a quadratic increase of channel nodes due to the indirect communication pattern.

Finally, Figure 12 introduces the build time to construct the Bayesian network. Clearly, the build time shows a significant difference between the large and small infrastructure examples for small numbers of instances with n less than 30 w.r.t.

service type. However, with increasing numbers instances the time difference diminishes. Afterwards, the sole factor that determines the build time is the service type. For large numbers of instances, the infrastructure has almost not significant influence on the build time anymore. Again the number of channel nodes that grow in proportion to the number of instances outweighs the component nodes of the infrastructure.

6 | DISCUSSION

The evaluation demonstrated the feasibility of the Bayesian network approach to model large-scale and replicated systems. Build and inference time is within manageable time frames for reliability engineers to make informed decisions on the service. Overall, for small service sizes with three to seven replicas as commonly used for transaction-oriented database systems, the reliability engineer can even use exact inference to assess the availability in order to compute deterministic results. We discussed that the number of channels has the most influence with regard to the build and inference time. Replicated services lead to a quadratic growth of channel nodes in the number of instances. Also, the build procedure needs to compute all possible routes that constitute a channel. Finding all possible routes in a graph can become a performance impediment, which is why we suggest considering only a subset of essential routes if performance is of higher priority. The largest model with 300 instances took about 1 h to build. But once the model is built, inference can be performed independently often. Even updating individual beliefs of component failures can be done directly to the respective nodes if needed, without rebuilding the whole Bayesian network.

A particular modeling challenge is the potential lack of accurate availability data (failure probabilities). Acquiring accurate failure data is a non-trivial task for rare events, which require a large number of observations to conclude statistical significance. However, this issue can be addressed in several ways. First, many vendors already provide mean time to failure (MTTF) information for their software or hardware components. Secondly, cloud providers host larger numbers of hardware components in their data centers, which are constantly monitored, providing significant amounts of data also for rare events.[34] Thirdly, for yet unobserved failures of highly available components, one can use rare event analysis (an active research area) in conjunction with expert knowledge acquisition to incorporate prior beliefs first and later refine the estimate with observation during mission time.

Moreover, our model does not consider the effects of long-running requests and the implications of component failures and recoveries during a longer execution time. This would require a dynamic Bayesian network approach[35,36] to model the time dimension, bringing new challenges w.r.t. model assumptions, which might require additional implementation details of the particular replication protocol, increasing the model complexity. Therefore, we consider this challenge as future work.

7 | RELATED WORK

Modeling complex infrastructures is subject to various areas of reliability engineering.[13,18,37,38] Jammal et al.[14] provide a hierarchical infrastructure model for cloud services with Petri nets as an evaluation framework. They consider fault propagation within a hierarchical infrastructure model supporting redundant cloud services with a one-out-of-n fault tolerance semantic. However, they do not consider network communication.

Ghosh et al.,[16] and Narayanan et al.[38] consider a k-out-of-n redundancy model for their instances; however, their infrastructure model only considers fault-independent compute nodes or data centers, respectively.

The Palladio Component model[39,40] provides a holistic modeling approach to evaluate the performance and availability of complex software systems unifying hard- and software into one model. However, the Palladio availability model supports only a one-out-of-n redundancy model and cannot model quorum requirements.

There are several methods to evaluate the availability of a system, among which Bayesian networks have gained large acceptance within the industry and research.[41–44]

Bobbio et al.[28,45] demonstrated the applicability and superiority of Bayesian networks in modeling and evaluating equivalent fault trees.[26] Moreover, Boudali and Dugan[35,36] showed how to use dynamic Bayesian networks to model dynamic fault trees as well, effectively proving that the Bayesian network formalism is powerful enough to cover all non-state space models.

Bennacer et al.[15] use Bayesian networks for network diagnostics by introducing a case-based reasoning inference approach to increase diagnostic performance for large-scale Bayesian network models. While they only focus on

network communication, they provide a tailored inference technique for efficient diagnostics of root causes, which can also be combined with our Bayesian network model when diagnostics is of interest.

Pitakrat et al.[46] use Bayesian networks for online failure predictions of microservice applications. The Bayesian network represents the interconnection between the microservice instances and updates the fault probabilities of the services based on the online monitoring of performance metrics. They consider fault propagation between services; however, replication is not considered.

In summary, a Bayesian network modeling approach, covering a wide range of redundant and replicated services that also includes cascading and correlated faults caused by dependent infrastructure and network communication, was missing.

8 | CONCLUSION

This work introduced a Bayesian network availability model for redundant and replicated services. The Bayesian network model unifies the fault aspects defined within a high-level model description of the service. The high-level model consists of three sub-models: a fault dependency graph to express the failure relation between components of the infrastructure and execution environment, a network model to address communication and network partitioning failures, and a model to define fault-tolerance requirements of the service. We show how to translate the high-level model into one Bayesian network to compute the expected availability. Finally, evaluations demonstrate the feasibility of the Bayesian network approach to represent and assess the availability of large-scale service with hundreds of fault influences and service instances.

ACKNOWLEDGMENTS

This work was supported by the Robert Bosch GmbH.

Open access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available in the supplementary material of this article.

ORCID

Otto Bibartiu https://orcid.org/0000-0003-1867-1681

REFERENCES

1. Cotroneo D, Simone LD, Liguori P, Natella R, Bidokhti N. Enhancing failure propagation analysis in cloud computing systems. In: *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE; 2019:139-150. doi:10.1109/issre.2019.00023. ISSN 1071-9458
2. Rosemain M, Satter R. Millions of websites offline after fire at French cloud services firm. https://www.reuters.com/article/us-france-ovh-fire-idUSKBN2B20NU, Mar. 2021, [Online; accessed 12-Oct-2021].
3. Janardhan S. Update about the october 4th outage. https://engineering.fb.com/2021/10/04/networking-traffic/outage/, Oct. 2021, [Online; accessed 12-Oct-2021].
4. Brown A. Facebook Lost About $65 Million During Hours-Long Outage. https://www.forbes.com/sites/abrambrown/2021/10/05/facebook-outage-lost-revenue/. Oct. 2021, [Online; accessed 12-Oct-2021].
5. Lakshman A, Malik P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.* 2010;44(2):35-40. doi:10.1145/1773912.1773922
6. Schiper N, Sutra P, Pedone F. P-store: genuine partial replication in wide area networks. In: *2010 29th IEEE Symposium on Reliable Distributed Systems*. IEEE; 2010:214-224. doi:10.1109/srds.2010.32
7. Alpos O, Cachin C. Consensus beyond thresholds: generalized byzantine quorums made live. In: *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE; 2020:21-30. doi:10.1109/srds51746.2020.00010
8. M Burrows. The chubby lock service for loosely-coupled distributed systems. In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association; 2006:335-350.
9. ISO/IEC/IEEE international standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*. IEEE; 2017:1-541. doi:10.1109/IEEESTD.2017.8016712
10. Lyu MR. Software reliability engineering: a roadmap. In: *Future of Software Engineering (FOSE '07)*. IEEE; 2007:153-170. doi:10.1109/fose.2007.24

11. Garraghan P, Yang R, Wen Z, et al. Emergent failures: rethinking cloud reliability at scale. *IEEE Cloud Comput*. 2018;5(5):12-21. doi:10.1109/mcc.2018.053711662
12. Gunawi HS, Hao M, Leesatapornwongsa T, et al. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM; 2014:1-14. doi:10.1145/2670979.2670986
13. Kim MC. Reliability block diagram with general gates and its application to system reliability analysis. *Ann Nucl Energy*. 2011;38(11):2456-2461. doi:10.1016/j.anucene.2011.07.013
14. Jammal M, Kanso A, Heidari P, Shami A. A formal model for the availability analysis of cloud deployed multi-tiered applications. In: *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. IEEE; 2016:82-87. doi:10.1109/ic2ew.2016.21
15. Bennacer L, Amirat Y, Chibani A, Mellouk A, Ciavaglia L. Self-diagnosis technique for virtual private networks combining Bayesian networks and case-based reasoning. *IEEE Trans Autom Sci Eng*. 2015;12(1):354-366. doi:10.1109/tase.2014.2321011
16. Ghosh R, Longo F, Frattini F, Russo S, Trivedi KS. Scalable analytics for IaaS cloud availability. *IEEE Trans Cloud Comput*. 2014;2(1):57-70. doi:10.1109/tcc.2014.2310737
17. Epstein A, Kolodner EK, Sotnikov D. Network aware reliability analysis for distributed storage systems. In: *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE; 2016:249-258. doi:10.1109/srds.2016.042
18. Ford D, Labelle F, Popovici FI, et al. Availability in globally distributed storage systems. In: *Osdi*. 2010;10:1-7.
19. Chiang MC, Huang CY, Wu CY, Tsai CY. Analysis of a fault-tolerant framework for reliability prediction of service-oriented architecture systems. *IEEE Trans Reliab*. 2021;70(1):13-48. doi:10.1109/tr.2020.2968884
20. Cockcroft A, Sheahan D. Benchmarking cassandra scalability on AWS - over a million writes per second. https://netflixtechblog.com/benchmarking-cassandra-scalability-on-aws-over-a-million-writes-per-second-39f45f066c9e. [Online; accessed 22-Feb-2022].
21. Langseth H, Portinale L. Bayesian networks in reliability. *Reliab Eng Syst Saf*. 2007;92(1):92-108. doi:10.1016/j.ress.2005.11.037
22. Duan R, Zhou H. A new fault diagnosis method based on fault tree and Bayesian networks. *Energy Procedia*. 2012;17:1376-1382. doi:10.1016/j.egypro.2012.02.255
23. Pan R, Yontay P. Reliability assessment of hierarchical systems with incomplete mixed data. *IEEE Trans Reliab*. 2017;66(4):1036-1047. doi:10.1109/tr.2017.2760802
24. Bibartiu O, Dürr F, Rothermel K, Ottenwälder B, Grau A. Scalable k-out-of-n models for dependability analysis with Bayesian networks. *Reliab Eng Syst Saf*. 2021;210:107533. doi:10.1016/j.ress.2021.107533
25. Heckerman D. Causal independence for knowledge acquisition and inference. In: *Uncertainty in Artificial Intelligence*. Elsevier; 1993:122-127.
26. Ruijters E, Stoelinga M. Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. *Comput Sci Rev*. 2015;15-16:29-62. doi:10.1016/j.cosrev.2015.03.001
27. Stamatelatos M, Vesely W, Dugan J, Fragola J, Minarick J, Railsback J. Fault tree handbook with aerospace applications. *Office of safety and mission assurance NASA headquarters*. Washington DC 20546, 2002.
28. Bobbio A, Portinale L, Minichino M, Ciancamerla E. Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliab Eng Syst Saf*. 2001;71(3):249-260. doi:10.1016/s0951-8320(00)00077-6
29. Pearl J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Reasoning*. Morgan Kaufmann Publishers; 1988.
30. Koller D, Friedman N. *Probabilistic Graphical Models: Principles and Techniques*. MIT press; 2009.
31. Lauritzen SL, Spiegelhalter DJ. Local computations with probabilities on graphical structures and their application to expert systems. *J R Stat Soc, Series B (Stat Methodol)*. 1988;50(2):157-194. doi:10.1111/j.2517-6161.1988.tb01721.x
32. Højsgaard S. Bayesian networks in R with the gRain package. *J Stat Softw*. 2012;46(10):1-26.
33. Højsgaard S. Graphical independence networks with the gRain Package for R. *J Stat Softw*. 2012;46(10):1-26. doi:10.18637/jss.v046.i10
34. Hochschild PH, Turner P, Mogul JC, et al. Cores that don't count. In: *Proceedings of the Workshop on Hot Topics in Operating Systems, ser. HotOS '21*. ACM; 2021:9-16. doi:10.1145/3458336.3465297. ISBN 9781450384384.
35. Boudali H, Dugan J. A discrete-time Bayesian network reliability modeling and analysis framework. *Reliab Eng Syst Saf*. 2005;87(3):337-349. doi:10.1016/j.ress.2004.06.004
36. Boudali H, Dugan JB. A new Bayesian network approach to solve dynamic fault trees. In: *Annual Reliability and Maintainability Symposium, 2005. Proceedings*. IEEE; 2005:451-456. doi:10.1109/rams.2005.1408404. ISSN 0149-144X.
37. Kim DS, Machida F, Trivedi KS. Availability modeling and analysis of a virtualized system. In: *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE; 2009:365-371. doi:10.1109/prdc.2009.64
38. Narayanan I, Kansal A, Sivasubramaniam A. Right-sizing geo-distributed data centers for availability and latency. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE; 2017:230-240. doi:10.1109/icdcs.2017.118
39. Brosch F, Koziolek H, Buhnova B, Reussner R. Architecture-based reliability prediction with the palladio component model. *IEEE Trans Softw Eng*. 2012;38(6):1319-1339. doi:10.1109/tse.2011.94
40. Becker S, Koziolek H, Reussner R. The palladio component model for model-driven performance prediction. *J Syst Softw*. 2009;82(1):3-22. doi:10.1016/j.jss.2008.03.066
41. Torres-Toledano JG, Sucar LE. Bayesian networks for reliability analysis of complex systems. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg; 1998:195-206.
42. Ye T, Zhou Y, Chen A, Liu L, Liu S. Extend GO methodology support common-cause failures modeling explicitly by means of Bayesian networks. *IEEE Trans Reliab*. 2020;69(2):471-483. doi:10.1109/tr.2019.2917752

43. Cai B, Kong X, Liu Y, et al. Application of Bayesian networks in reliability evaluation. *IEEE Trans Industr Inform*. 2019;15(4):2146-2157. doi:10.1109/tii.2018.2858281
44. Kammouh O, Gardoni P, Cimellaro GP. Probabilistic framework to evaluate the resilience of engineering systems using Bayesian and dynamic Bayesian networks. *Reliab Eng Syst Saf*. 2020;198:106813. doi:10.1016/j.ress.2020.106813
45. Bobbio A, Portinale L, Minichino M, Ciancamerla E. Comparing fault trees and bayesian networks for dependability analysis. In: *Computer Safety, Reliability and Security*. Springer Berlin Heidelberg; 1999:310-322.
46. Pitakrat T, Okanović D, van Hoorn A, Grunske L. Hora: architecture-aware online failure prediction. *J Syst Softw*. 2018;137:669-685. doi:10.1016/j.jss.2017.02.041

How to cite this article: Bibartiu O, Dürr F, Rothermel K, Ottenwälder B, Grau A. Availability analysis of redundant and replicated cloud services with Bayesian networks. *Qual Reliab Eng Int*. 2024;40:561–584. https://doi.org/10.1002/qre.3414

AUTHOR BIOGRAPHIES

Otto Bibartiu is a PhD candidate at the Distributed Systems Department, Institute of Parallel and Distributed Systems, University of Stuttgart, Germany. He received his Bachelor's and Master's degrees in computer science at ETH Zurich. His research interests include cloud computing availability and probabilistic graph models.

Frank Dürr received the Doctoral and Diploma degrees in computer science from the University of Stuttgart. He is a Senior Researcher and a Lecturer with the Distributed Systems Department, Institute of Parallel and Distributed Systems, University of Stuttgart, Germany, where he is currently leading the the software-defined networking/time-sensitive networking groups of the Distributed Systems Department. His research interests include deterministic real-time communication in wired and wireless networks as well as mobile and pervasive computing.

Kurt Rothermel received the Doctoral degree in computer science from the University of Stuttgart in 1985. From 1986 to 1987, he was Post-Doctoral Fellow with IBM Almaden Research Center, San Jose, USA, and then joined IBM's European Networking Center, Heidelberg. Since 1990, he has been a Professor of computer science with the University of Stuttgart. From 2003 to 2011, he was the Head of the Collaborative Research Center Nexus (SFB 627), conducting research in the area of mobile context-aware systems. He was the Director of the Institute of Parallel and Distributed Systems. His current research interests are in the field of distributed systems, computer networks, and mobile systems.

Beate Ottenwälder received the Doctoral and Diploma degree in computer science from the University of Stuttgart. She was a researcher at the Distributed Systems Department, Institute of Parallel and Distributed Systems, University of Stuttgart, Germany, focusing on complex event processing systems. Currently, she is a product owner for the cloud management engine in the private cloud at the Robert Bosch GmbH, Germany.

Andreas Grau received the Doctoral and Diploma degree in computer science from the University of Stuttgart. He was a researcher at the Distributed Systems Department, Institute of Parallel and Distributed Systems, University of Stuttgart, Germany, focusing on the scalability of network emulation. Currently, he is a senior manager for private cloud infrastructure at the Robert Bosch GmbH, Germany.