Dependability Evaluation of a Smart Poultry House: Addressing Availability Issues Through the Edge, Fog, and Cloud Computing

Felipe Oliveira, Paulo Pereira, Jamilson Dantas, Jean Araujo and Paulo Maciel

***Abstract*— Internet of Things (IoT) applications equip rural producers with decision support tools and automated solutions that boost agribusiness productivity, quality, and profit. However, most poultry farmers still use conventional methods of operation in which human workers carry out all routines for monitoring and controlling their farms at the expense of greater productivity. One of these human activities is manual weighing, which can be replaced by non-intrusive methods such as computational vision applications that estimate live poultry's weight using video cameras. Since Internet of Things (IoT) devices may have low computing power limiting the ability to process the data locally, they can transfer it to a fog or cloud data center, where they are processed. This paper aims to conduct a dependability study of a poultry house automated with a computer vision-based system for estimating poultry weight considering hierarchical models (e.g., Markov chain, Reliability Block Diagram, and closed-form equation) to represent the whole system and obtain steady-state availability and annual downtime. In addition, our purpose is to consider and compare different architectural solutions, such as edge and fog computing-based solutions. The proposed solution verified that a cloud-based application with no redundancy has a downtime of 34.14% and 9.176% hours when considering a hot-standby redundancy strategy in the office node of a cloud solution.**

***Index Terms*— Internet of Things, Dependability analysis, Markov Chain**

I. Introduction

IN agribusiness, Internet of Things (IoT) applications equate rural producers with decision support tools and automated solutions that boost productivity, production quality, and property profit [1]. For example, computer solutions based on agribusiness might (i) estimate the appearance of pests in the crop and, consequently, reduce the frequent use of insecticides and fungicides [2]; (ii) keep control of safe temperatures and humidity conditions in production storage facilities [3]; (iii) make available to producers and consumers an up-to-date and secure record of the path taken by food at all stages of the production chain [4]; (iv) manage technical, strategic and operational data for the entire rural property for planning, documentation, and process optimization [5].

Poultry agribusiness is an important industry in emerging economies like Brazil and Malaysia. Brazil is the third-largest producer in the world and the first when it comes to the largest exporters [6].

The poultry industry can be divided into egg and meat production. The system's availability features may be peculiar to each category according to the product type. For example, in meat production, the life cycle of chickens is 50 days maximum. The shed is emptied at the end of this period, and a new cycle begins. During this cycle, when the chickens are going through the growth process, any sudden environmental change can cause an increase in mortality. Therefore, monitoring the environment with minimal disruption is critical to avoid poultry mortality. However, the system may be interrupted when the shed is emptied for maintenance.

In addition, when we analyze egg production, the hens' life cycle is longer. Reaching up to two years, laying hens are kept for longer and require as much care as those applied in meat production. In this scenario, high unavailability can blind the technician to the actual conditions of the sheds, leaving damages from which the producers can not recover.

In the last few years, some computational paradigms may have been used as the architectural environment to implement solutions and services [7]. This paper aims to carry out an infrastructure based on cloud, fog, and edge for monitoring and controlling of ambiance metrics of a poultry farm. An automatized poultry farm comprises sensors that collect metrics essential to maintain the poultry's health, such as temperature, water quality indicators, and illumination sensors. Also, we can use video images to extract information about the broiler's weight or sanity. We create and validate, by executing experiments in a controlled environment, Reliability Block Diagram (RBD) and Markov chain models to represent the system and study the availability issues of this system. Additionally, we perform a sensibility analysis to identify essential components to improve the system's availability. Our analysis verified that fog-based availability with no redundancy has a slight advantage compared with the edge solution. When considering the fog and cloud nodes, the downtime is 38.85 hours and 34.14 hours. We also consider scenarios with redundancies and perform a sensitivity analysis where we can

This work was supported by the FACEPE - Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco under Grant IBPG-1220-1.03/19.

Felipe Oliveira, Paulo Pereira, Jamilson Dantas and Paulo Maciel, are with Universidade Federal de Pernambuco, Recife, Brazil. E-mail: {fdo, prps, jrd, prmm}@cin.ufpe.br.

Jean Araujo is with Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil. Email: jean.teixeira@ufape.edu.br.

identify critical components of the system's availability. The contributions of this work are:

- The proposition of architecture of intelligent sheds considering different architectural patterns;
- The suggestion of Continuous-time Markov Chain (CTMC) and Reliability Block Diagram availability models validated through fault injection experiments;
- An availability study considering different types of redundancy and fault tolerance;
- And an availability analysis study highlighted the essential system availability parameters.

The rest of this paper is structured as follows: Section II introduces the concepts of cloud, fog, and edge computing, as well as dependability metrics. Some related works are presented in Section III. Section IV describes the system architecture. In Section V we introduce the proposed models. In Section VI, we perform a dependability study and analysis, and finally, in Section VII, we make the final remarks.

II. Background

This section briefly explains the main concepts crucial to understanding this work. First, we provide an overview of cloud, fog, and edge computing, explain the modeling and availability analysis, and discuss sensitivity analysis.

A. Cloud, Fog and Edge Computing

For processing data and operating Internet of Things applications, several different technologies are available, including cloud computing, fog computing, and edge computing (IoT). Each of these computing paradigms carries out a particular task to facilitate IoT devices' effective and efficient functioning.

Cloud computing [7] provides high computational power and unlimited storage capacity, scarce resources in IoT devices. So it is intuitive to offload the data to the cloud services, where such data can be processed and stored. However, since the data centers are distributed globally, the distance between the cloud server' and the users can imply high latency rates, which can be unwanted in real-time applications.

A fog-based architecture allows some computational resources closer to the IoT devices than the cloud servers. These advances in the network's edge facilitate data transmission flow, reducing network bandwidth latency and consumption. Fog computing can support applications that need low latency and can be used in scenarios where this is not required. Fog computing can be used for localized analytics and decision-making, much like cloud computing can be used for large-scale analytics and long-term storage.

Edge nodes refer to all devices that can ensure some computational power or storage capacity, performing closer to the sensors and actuators. This proximity with the resources at the "edge" has benefits, which include less transmission latency on data computing or storage. In addition, edge computing can offer the fast response times required for IoT applications even with limited resources. For instance, data can be filtered and prepared for analysis at the edge before being sent to cloud or fog nodes.

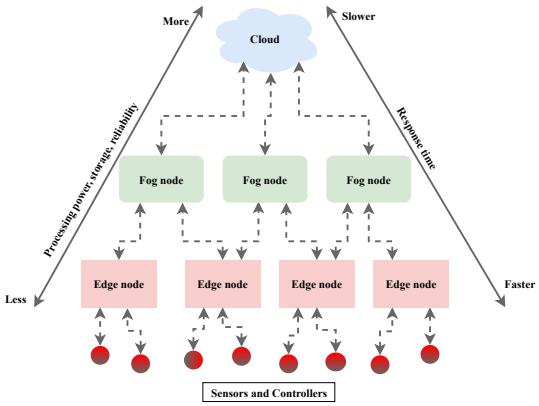


Fig. 1. IoT Stack: Edge to cloud

Figure 1 shows the common stack that represents each paradigm. The edge nodes adjacent to the sensors and actuators' are closer to the pyramid base. The fog is halfway to the cloud, having more computational resources than the edge nodes. Furthermore, the cloud has greater computational resources and storage at the top. It is important to highlight that there is less network overhead closer to the stack base and less latency. On the other hand, more processing power, storage, and reliability are expected closer to the top of the stack.

So when it comes to designing a system, we must consider which would be the desired features.

B. Modeling and Availability analysis: an overview

A system's dependability can be understood by its capacity to do the following: deliver a reliable service, properly fulfill a set of specific functionalities (with high availability, in a safe manner, without adverse effects) [8], and avoid frequent and serious service failures [9].

The dependability concept includes some other metrics, such as availability and reliability. In this paper, we have a particular interest in availability, that is, the probability of a system being in a functioning condition, considering the interchange of operational and non-operating states. Availability (A) can be calculated by the ratio between the Mean Time to Failure (MTTF) and the Mean Time to Repair (MTTR), as shown in Equation 1. Furthermore, the unavailability (U) is calculated using Equation 2.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

$$U = 1 - A \tag{2}$$

The mean time to repair (MTTR) can be estimated using the MTTF, as shown in Equation 3:

$$MTTR = MTTF \times \frac{U}{A} \tag{3}$$

The mean time to failure (MTTF) can be described by integrating the reliability function as shown in Equation 4. The reliability function defines the probability of a system surviving a certain period without failures.

$$MTTF = \int_0^\infty R(t)dt \tag{4}$$

The Reliability Block Diagram (RBD) is a modeling technique that can evaluate a system's reliability through each component's reliability. Each block represents a component of the system and its functioning state, i.e., functional or failed. The system works properly with a path from the start node to the final node.

The Markov chain is an analytical modeling technique that represents a stochastic process represented by a state-space diagram, in which the future state of the chain is determined by the current state and not by the way that the process got to that state [10]. There are three types of Markov chains: Discrete-Time Markov chains (DTMC), Continuous-Time Markov Chains (CTMC) [11] and semi Markov process [12]. This work will use CTMCs when our state space is continuous.

There are several methods to perform a sensitivity analysis. This work utilizes a differential sensitivity analysis technique that uses a sensitivity coefficient calculated by the ratio between a given variable parameter and the output metric. In contrast, the remaining parameters are kept constant. In this way, if we aim to calculate the sensitivity coefficient of a metric Y which depends on a parameter θ, we must use the Equation 5 or Equation 6 for scaled sensitivity [13].

$$S_\theta(Y) = \frac{\partial Y}{\partial \theta} \tag{5}$$

$$SS_\theta(Y) = \frac{\theta}{Y}\frac{\partial Y}{\partial \theta} \tag{6}$$

III. Related Works

This section provides some bibliographic material from the latest years, which is key to understanding this research's real contributions.

The work referred to in item [14] models the IoT system of a smart building and presents a modeling strategy based on hierarchical models using the Continuous Time Markov Chain (CTMC). Using such proposed models, the authors estimate the steady-state availability by comparing a local infrastructure and a cloud computing system, considering a cold-standby strategy. They also perform a sensitivity analysis to verify which components in the system most affected the system's availability. Unfortunately, the authors did not consider other types of architecture, such as edge and fog computing, as we did in our work; the authors verified a cold standby redundancy strategy. However, in our work, we do not conduct such analysis.

Kutyauripo et al. [15] evaluated several modeling techniques applied to the agro-food sectors. The authors describe how to use and the performance of those several models to optimize the production of broiler.

In [16], the reliability and security challenges of an IoT-based smart business center (SBC) network were examined, and a Markov model was built to demonstrate a high level of protection against hacker attacks.

Regarding IoT for smart health care, [17] proposes availability models for an IoT system focused on failures and attacks on components. The paper considers the main causes of failure in a healthcare system and shows Markov chain models to set up the IoT infrastructure.

In [18], Markov chain models are proposed for modeling latency-sensitive applications such as a security face recognition system in a train station. In addition, the authors propose availability models considering edge and fog architectures, performing a capacity-oriented availability analysis and a cost assessment.

In the work referred to [19], the authors use Reinforcement Learning model to offload edge computing networks. The authors took into consideration how the jamming and interference affect the performance of the systems.

Our work proposes the Markov chain and the Reliability Block Diagram (RBD) models for an intelligent poultry house system. It conducts an availability analysis considering the different architectural approaches (such as solutions with edge, fog, and cloud computing) with different redundancy configurations. We also perform a sensitivity analysis to identify the main components affecting the system's availability.

IV. System Architecture

When it comes to a smart poultry farm, we consider obtaining data from the poultry houses by sensors and using IP cameras and video processing to estimate the poultry weight [20]. We suggest the following architectural solutions: an edge-based solution, a fog-based solution, and a cloud-based solution considering, in all cases, a hot-standby redundancy.

These solutions follow a basic one: to obtain the input data (video through the IP camera and metrics by the sensors) and transmit it to a node with enough power capacity to process these data. In the processing node, images are classified, and the attributes of interest are extracted. The data obtained from the sensor nodes (the temperature, gas, light, and water quality sensors) is also centered on the edge node and transmitted to the office. The technical responsible for the shed has access to this data through a computer in the local office, where the data is stored locally and may undergo further processing. So with the information of interest in hand, he gets an overview of the situation and can intervene and make any changes if needed. In this scenario, the technician holds information about a specific shed and has an overview of all the sheds he manages.

An edge-based architecture is shown in Figure 2. In this approach, the video captured through the IP camera is processed only by the edge node. The technician has access to the processed data in a local office, which is interconnected to the system's components by a network switch or a router. The downside of this solution is that the edge resources are limited, which restrains the video processing capacity.

A fog-based solution can be also viewed in Figure 2. In this architecture, data processing responsibility is transferred to a fog node. This fog node is a server that can be in the same network as the office or the IP camera but can also be a server in another network accessible by the Internet but not so far from the cloud.

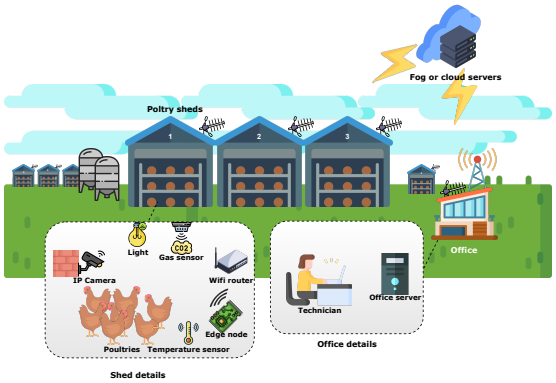


Fig. 2. Environment design

The cloud-based solution can also be viewed in Figure 2. Data usually are transferred to a cloud server that processes it and returns the results to the office application in this approach. In this architecture, network latency is expected, given that the infrastructure administrators access the cloud servers through the Internet.

Figure 3 shows an abstraction made in order to simplify our models. We assume that a physical node is a junction between the hardware and the operating system. This component represents the edge node, the fog node, and the office node. Regarding the cloud, we assume that this component (physical node) represents a virtual machine instance. On top of a physical node, one or more application instances are running. These applications can be the classification software or a virtual machine providing the office's application service.

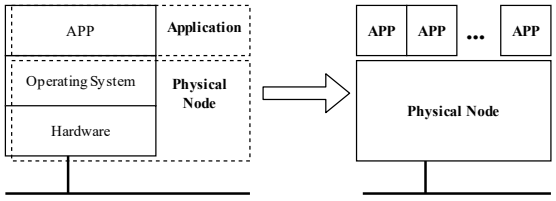


Fig. 3. Abstraction of a general node

V. Proposed Models

This section presents availability models for the environment described in Section IV. We used a Markov chain and an RBD model to represent and establish the link between the components of the system. These formalisms are chosen because they offer closed equations to calculate the availability, which is more convenient than simulation processes and is processed faster.

First, to build our Markov chain model, we need to obtain a component representing the abstraction we made in Figure 3. We must consider that the operating system and the hardware on the stack are a single component called a physical node. This simplification allows the model to be simpler, excluding unnecessary details that cannot refine our model. To do this, we use the RBD model and also extract Equation 7, where we have the availability of physical nodes (A_{pn}), availability of hardware (A_{hw}), and availability of operating system (A_{os}); from which we can obtain the availability of one physical node instance. The RBD model obtained the MTTF and MTTR related to the edge, fog, cloud, and office physical node components by an exact evaluation on Mercury Tool[1] [21].

$$A_{pn} = A_{hw} \times A_{os} \tag{7}$$

We can also consider the sensors as a unique component called a sensor module and use a serial RBD to represent it. The components of the sensor module are an *esp32* microcontroller, a temperature sensor, a gas sensor (that captures the CO2 and Ammonia gases), a light sensor that is for measuring the light intensity, and a water quality sensor, that measures the *ph*, temperature, and turbidity of the water.

Equation 8 refers to the availability equation for one instance of the module sensor extracted from the serial RBD described previously. The availability of the sensor module is represented by the product of the esp32 availability (A_{esp32}), the temperature sensor availability (A_{ts}), the gas sensor availability (A_{gs}), the availability of the light sensor (A_l), and the water quality sensors availability (A_w).

$$A_s = A_{esp32} \times A_{ts} \times A_{gs} \times A_l \times A_w \tag{8}$$

Figure 4 shows the proposed Markov chain model. Our model comprises five components: the camera, the network switch, the sensors' module, the processing node, and the office node.

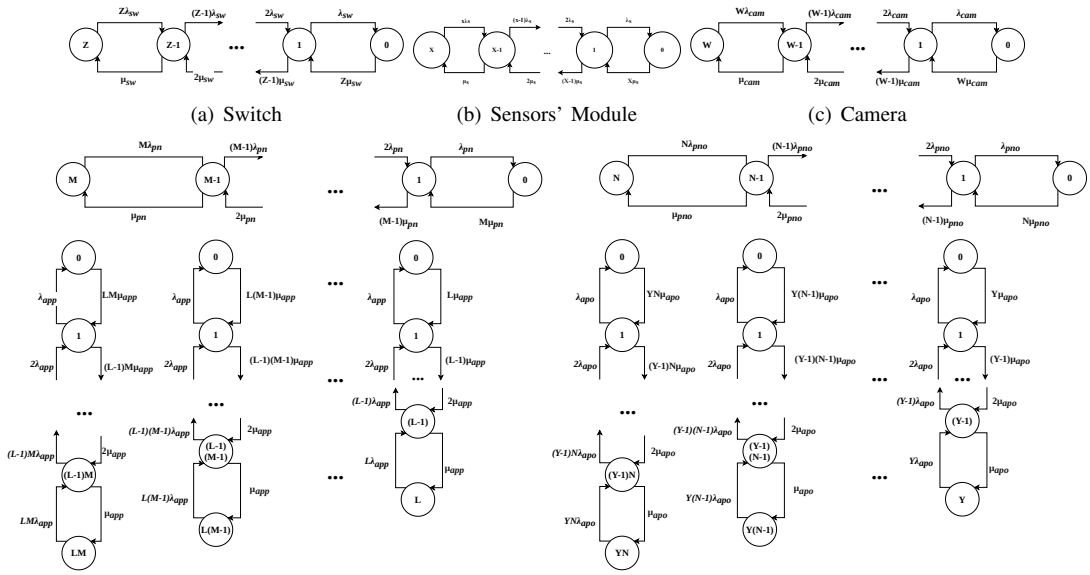
In a baseline model, we consider a minimal functional scenario composed of a switch, a sensor module, a camera, an office computer running an application, and a processing node executing an instance of the Software responsible for processing the video. This strategy is to simplify our baseline, but our model considers having one or more of each component, i.e., we can have redundancies of each element that forms part of the architecture. So, for example, our model can have one, two, or Z switches. In the same direction, we can have up to W cameras, M processing nodes with L applications running, and N office nodes running applications.

One or several applications can run in the processing and office nodes model. This application can be either the Software that classifies the images or the office analysis software. For example, if we have one active processing node, we can have up to L applications running in that node. On the other hand, we have up to $2L$ applications available if we have two active processing nodes. So, if we have M processing nodes, we can have ML applications running.

The office processing node model has a similar function. Each office node can have one or up to Y applications running in any node. Since the model supports up to N office processing nodes, we can have, if we have the maximum of office processing nodes, up to NY office applications running.

Each switch can have both a failure and a repair rate (represented respectively by λ_{sw} and μ_{sw}), as well as a camera (λ_{cam} and μ_{cam}). The processing node has a failure rate λ_{pn} and a repair rate μ_{pn}. Each application running in the processing node also has failure and repair rates (λ_{app}, μ_{app}).

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



(a) Switch (b) Sensors' Module (c) Camera

(d) Processing node (e) Office node

Fig. 4. Markov chain models

Finally, the office node failure rate is represented by λ_{pno} and the repair rate by μ_{pno}. In turn, this office application also has failure and repair rates (λ_{appo}, μ_{appo}). Our model assumes that each component and replica are identical and that the system has sufficient and independent repair teams.

From the Markov chain, we can extract the availability equations shown forward. These equations are important because, with them, we can test infrastructure changes without implementing them duly. This means it is more cost-efficient than buying and implementing the entire infrastructure. First, we obtain the availability equation for each component: Equation 9 for the switch, Equation 10 for the sensor module, Equation 11, Equation 12 for the processing server nodes, and Equation 13 for the office server nodes. In this way, we can calculate the availability for the entire system, shown in Equation 14.

$$A_{sw} = \sum_{j=1}^{Z} \frac{Z!}{j!\,(Z-j)!} \times \frac{\lambda_{sw}^{Z-j}\mu_{sw}^{j}}{(\lambda_{sw}+\mu_{sw})^{Z}} \tag{9}$$

$$A_{s} = \sum_{q=1}^{X} \frac{X!}{q!\,(X-q)!} \times \frac{\lambda_{s}^{X-q}\mu_{s}^{q}}{(\lambda_{s}+\mu_{s})^{X}} \tag{10}$$

$$A_{cam} = \sum_{k=1}^{W} \frac{W!}{k!\,(W-k)!} \times \frac{\lambda_{cam}^{W-k}\mu_{cam}^{k}}{(\lambda_{cam}+\mu_{cam})^{W}} \tag{11}$$

$$A_{p} = \sum_{i=1}^{M} \left(1 - \frac{\lambda_{app}^{iL}}{(\lambda_{app}+\mu_{app})^{iL}}\right) \times \frac{M!}{i!\,(M-i)!} \times \frac{\lambda_{pn}^{M-i}\mu_{pn}^{i}}{(\lambda_{pn}+\mu_{pn})^{M}} \tag{12}$$

$$A_{o} = \sum_{p=1}^{N} \left(1 - \frac{\lambda_{apo}^{pY}}{(\lambda_{apo}+\mu_{apo})^{pY}}\right) \times \frac{N!}{p!\,(N-p)!} \times \frac{\lambda_{pno}^{N-p}\mu_{pno}^{p}}{(\lambda_{pno}+\mu_{pno})^{N}} \tag{13}$$

$$A_{sys} = A_p \times A_o \times A_{sw} \times A_{cam} \times A_s \tag{14}$$

It is worth pointing out that we use the Mercury[2] Modeling Tool [21] to find our closed-form solutions. The Mercury Tool allows us to draw Markov chains and calculate the probability of each state. This tool also allows us to export our Markov chain model to the Wolfram Mathematica[3] Tool [22], where we can find the probability equations of each state. From there, we perform algebraic manipulations and can develop the generalization of our analytical models.

We proposed analytical models instead of simulations or measurements because they are costly. Simulations need to perform an entire run to get a sample point of how the system behaves; consequently, simulations perform several runs to build a sample space [23]. On the other hand, measurements are only feasible if there is already a system or prototype to compare. The biggest drawback of this method is that it must have at least a prototype to measure, and monitoring activities are very susceptible to errors [23]. Therefore, we proposed analytical models to help analysts to plan and evaluate faster and easier conceptual systems.

VI. Case Studies

In this section, we describe the cases that we have conducted. The first step is a fault injection experiment to obtain data to validate the edge-based model. We also study steady-state availability based on an experiment in the Mercury tool [21].

[2]https://www.modcs.org/
[3]https://www.wolfram.com/mathematica/

A. Edge-based Model Validation

We aim to create a model representing a real system; once created, we need to validate this model to ensure that it suits the real scenario. As a baseline for our study, we consider an environment with one edge node running the video processing application. We perform a fault injection experiment to obtain the data required for such validation. This technique is used to understand the system's behavior when a failure occurs. Hence, a controlled environment is created and monitored while the failures are injected. A failure can take a long time to occur in an operating system. Before that, we accelerate the occurrences of failures in our experiment. We consider a reduction factor (RF) of 876 for each MTTF, i.e., turning one year into ten hours. Thus, in our system, failure occurred by a fault injection experiment; 10 hours are equivalent to a period of 1 year in a real system.

We consider that the MTTF and MTTR, applied to inject the failures and simulate their repairs, as shown in Table I, are exponentially distributed. The column "MTTF RF" of the same table shows the value of the MTTF after applying the reduction factor. These are the values that are set in the fault injector.

These times for failure and recovery have been used in a fault injection experiment implemented in an infrastructure of the base architecture.

Component	MTTF (h)	MTTF RF (h)	MTTR (h)
Camera	25,000	28.53	2
Orange pi	25,000	28.53	12
Raspbian OS	8,000	9.13	6
Application	8,000	9.13	0.1
Switch	25,000	28.53	6
Office Hardware	40,000	45.66	48
Office OS	400	0.46	0.5
Office application	1,200	1.37	0.5
Esp32 module	22,602	25.80	2
Temperature sensor	13,140	15	2
Gas sensor	13,140	15	2
Water quality sensor	13,140	15	2
Light sensor	13,140	15	2

TABLE I
MTTF AND MTTR BY COMPONENT

The environmental setup was composed of an Orange pi WinPlus model with an AllWinner A64 SoC processor, 2GB DDR3 SDRAM, and 32 GB of storage. The operating system was the Raspbian Server 10, and the video processing application was a program in Python 3. The office computer was personal; its configuration was an AMD C-60 processor, 4GB RAM, and 500 GB storage capacity, with Debian 10. The fault injector was running in another personal computer, with an Intel i7-8565U 1.8 GHz, 8GB RAM, 256GB SSD, and Linux Debian 10. Interconnecting all this was an 8-port Gigabit network switch. The fault injector was a Python 3 program that creates sub-processes to inject failures in each system component, following the MTTR and MTTF shown in Table I. The fault injector generates a random value based on the exponential distribution with rates from the literature. We use a Python library to generate these random values.

Alongside this execution, a bash script verifies every 10 seconds if those system components are working properly, saving the referred status (if it is up or down) in a text file. The experiment lasted 100 hours, and we got a total of 501 failures and repairs in all system components, i.e., the degree of freedom was 501. This information will be useful for calculating the confidence interval in the next step.

Now, we can calculate the system's availability with the measured information, which is 0.99474. We can also apply a method [24] to calculate the 95% confidence interval's availability. We can now determine if we have evidence to reject the proposed model or to accept it with the confidence interval. We get that the mean is between $0.99374 < X < 0.99558$.

By calculating the availability based on the proposed Markov Chain model, we obtain availability of 0.99557, which is within the confidence interval. So, we have no evidence to reject the model representing the system. By proposing analytical models instead of simulation or measurements, we are able to analyze several distinct scenarios faster than if we used simulation or measurements. The motivation behind this proposal is to make the evaluation of settings easier for the system analyst.

B. Steady-State Availability Study

First, we perform steady-state reliability analysis of the model corresponding to Equation 7, using the Mercury tool and the numbers shown in I. The results of this analysis can be found in Table II. We consider that the MTTF and MTTR for the cloud are 2880 and 0.03, respectively. Based on this, we can extract that the MTTF and MTTR for the fog physical node component are 2167.4226 and 1.1659, respectively. The edge's physical node has an MTTF of 6060.6056 hours and an MTTR of 7.4567 hours. Finally, the MTTF and MTTR for the office's physical node correspond to 396.0396 and 0.9709 hours.

Metric	Cloud	Fog	Edge	Office
MTTF (h)	2880	2167.4226	6060.6056	396.0396
MTTR (h)	0. 03	1.16591	7.45672	0.97089
Availability (%)	0.99999	0.9994624	0.9987711	0.9975
Availability (# 9's)	4.98227	3.2695	2.9105	2.6116
Downtime (h)	0.09125	4.7128	10.7718	21.4368

TABLE II
RESULTS FROM PHYSICAL NODE RBD MODEL

Applying the numbers displayed in Table I to the model represented by Equation 8, we obtain the numbers of MTTF and MTTR for the sensor module component. The obtained failure and repair times are 2868.14107 and 2.00055 hours. We can view these results in Table III. Now we can set the components of the processing node and the office node in the Markov chains shown in Figure 4(d). By conducting a steady-state evaluation using the models with the times shown in Tables I and II, we are able to build Tables IV and V.

Table IV shows the results with no redundancy strategies. We set up the model with one switch, one camera, one sensor module, one edge, fog, or cloud device with one processing application and one office node. We can see in Table IV

Metric	Sensor modules
MTTF (h)	2868.14107
MTTR (h)	2
Availability (%)	0.99930
Availability (# 9's)	3.15675
Downtime (h)	6.10996

TABLE III
RESULTS FROM SENSORS' MODULE RBD MODELS

that the downtime of the fog-based architecture is lower than the edge solution. A downtime of 38.85 hours for the fog architecture signifies that the system would be offline for approximately 39 hours in a year, with an availability of 0.99556, which contains 2.35 nines. However, if we consider an edge node, the availability will be 0.99488, with 2.29 nines representing an annual downtime of 44.77 hours and an uptime of 8715.22 hours. The cloud-based solution's downtime is the lowest of the three architectures tested, being approximately 34 hours in a year. This way, we got an availability of 0.99610, with 2.40 nines.

Metric	Edge-based	Fog-based	Cloud-based
Availability (%)	0.99489	0.99556	0.99610
Unavailability (%)	0.00511	0.00443	0.00389
Availability (# 9's)	2.29148	2.35310	2.40922
Uptime (h/y)	8715.22661	8721.14905	8725.85858
Downtime (h/y)	44.77339	38.85095	34.14142

TABLE IV
RESULTS WITH NO REDUNDANCY

When we consider scenarios with redundancy in the processing node, i.e., with two processing nodes in hot-standby, which can be seen in Table V, all architectures - fog, cloud, and edge - have similar results. We can see the results with redundancy in Table V. The system's downtime with redundancy for this component is approximately 34 hours. The availability is 0.996125, which means an uptime of 8726 hours a year.

We can see better results when considering the office node redundancy. In the edge-based architecture, the availability obtained was 0.99773, which means 2.64 nines and uptime of 8740 hours. In the fog, the uptime result was 8746 hours, which means availability of 0.99841 with 2.79949 nines. These results represent a gain of 6 hours to the fog solution in a year. In the cloud-based solution, availability was 0.99895 representing approximately an uptime of 8751 hours and downtime of 9.17692 hours in a year. The cloud solution has a gain of more than 10 hours compared to the edge solution.

However, it is noteworthy to highlight that this redundant approach implies costs. These costs can be related to equipment acquisition or other aspects like power consumption, refrigeration, and maintainability since the spare computer is always online. Depending on the solution adopted, we can reduce these costs by embracing the policy of freeing up the resources when they are not being used.

The decision regarding which strategy to adopt to ensure desirable availability must consider the availability that wants to be achieved and the financial feasibility of implementing the solution. Although placing as many replicas as possible for each component is a viable solution to maximize availability, the costs involved in the operation may be higher than the gains obtained.

C. Sensitivity Analysis

We have performed a sensitivity analysis to study how the model's parameters impact the system's availability. This analysis aims to show us which parameter significantly influences the system availability when it reaches the steady-state. Table VI presents the sensitivity ranking calculated based on the partial derivative from Equation 14. Here are the sensitivity index parameters shown in decreasing order:

The top of the rank includes those that directly influence the system's availability. These parameters are related to the number of office nodes, the number of processing nodes, the number of office applications per node, and the failure and repair rate of the office node. For example, the parameters (N, M, Y, λ_{pno} and μ_{pno}) have an impact on the increase of the component's failure probability and, thereby, all system fails. On the other hand, the failure and repair rates of the camera and processing applications have less influence on the system's availability.

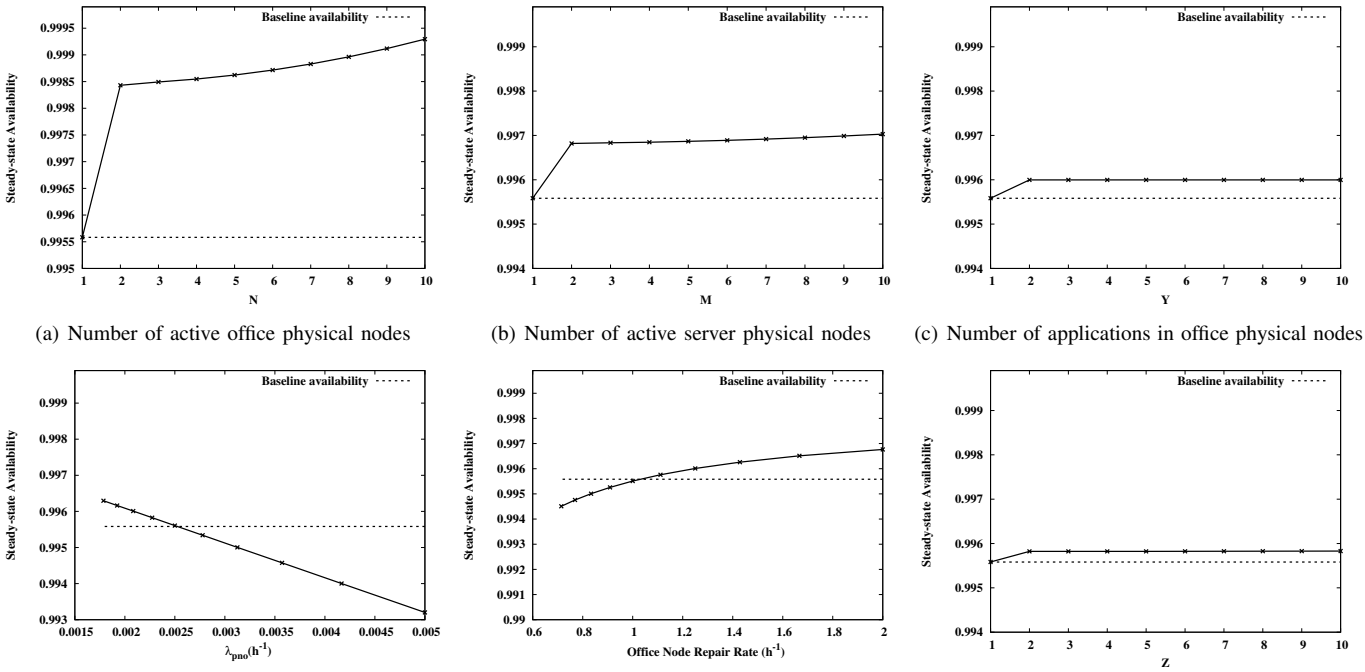
Figure 5 is used to verify the sensitivity of the system's availability when we variate each parameter at a time. For example, once they were in the integer domain, we changed the numbers referring to N, M, and Y in single steps (i.e., 1, 2, 3, ...). This kind of graph is used to verify how a parameter's variation impacts the system's availability. For example, we can see in Figure 5 that some parameters, like the number of active physical nodes (servers) or the number of active switches, cause a small variation in the availability after the second redundancy. So we can assess whether the costs required for adding a new component's replica are worth it once these components slightly improve availability.

On the other hand, changing the number of office nodes from one to two and getting a better repair rate for the office nodes increases the general availability. We can decide which component will be redundant in the infrastructure or what will be the best repair rates taking into account the operating costs. A better repair rate will depend on the contract adopted and on the number of repair teams available. It is essential to highlight that the graphs help us find the best fit to maximize the infrastructure's availability.

For example, we are considering that now the MTTF of the office node is equivalent to 560 hours and that the MTTR corresponds to 0.5 hours. We can get this MTTF by acquiring better hardware and operating systems. Also, we can increase the MTTR with more repair teams. Let us assume that two office nodes will be used in hot standby based on the times shown in Table II and that the architecture adopted will also be based on edge computing, considering that the MTTF and MTTR, respectively 8400 and 3.5 hours. We will be using a camera, a switch, and a sensor module with the times described in Tables I and III. Based on this configuration, the availability will be 0.99855, representing a yearly downtime of 12.67 hours. It is an improvement of approximately 7 hours in the uptime, reducing the downtime by 36.12%.

Redundant component	Availability	Unavailability	#9	Uptime (h/y)	Downtime (h/y)
Physical node on Edge	0.99612	0.00388	2.41160	8726.04510	33.95489
Physical node on Fog	0.99612	0.00387	2.41174	8726.05578	33.94421
Physical node on Cloud	0.99612	0.00387	2.41177	8726.05854	33.94146
Office node on Edge	0.99773	0.00226	2.64498	8740.16069	19.83931
Office node on Fog	0.99841	0.00159	2.79949	8746.10008	13.89992
Office node on Cloud	0.99895	0.00105	2.97980	8750.82308	9.17692

TABLE V
RESULTS WITH HOT STANDBY REDUNDANCY



(a) Number of active office physical nodes (b) Number of active server physical nodes (c) Number of applications in office physical nodes

(d) Office Node Failure Rate op (e) Office Node Repair Rate (f) Number of active switches

Fig. 5. One at a time sensitivity analysis of the top eight parameters

Parameter	$\lvert SS(A)\rvert$
N	0.0147034
M	0.00838095
Y	0.0032437
λ_{pno}	0.00243776
μ_{pno}	0.00243776
Z	0.00200075
μ_{pn}	0.00125255
λ_{pn}	0.00125255
W	0.000754806
μ_{apo}	0.00041656

TABLE VI
SENSITIVITY RANKING BASED ON PARTIAL DERIVATIVES

VII. FINAL REMARKS

This study aimed to propose dependability models to perform availability studies on a smart poultry house with sensors to measure the environmental indicators, such as temperature, and a computer for estimating poultry weight. We have considered three architectural systems: cloud-based, edge-based, and fog-based. We have also considered a redundancy strategy to measure the impact on improving availability, considering the hot-standby nodes. As a result, we have presented three case studies: first, a validation of an edge-based model with a fault injector experiment; second, we have performed a steady-state availability study; then, we have conducted a sensitivity analysis.

Our availability analysis verified that fog-based availability with no redundancy has a slight advantage compared to the edge solution. When we consider the fog node, the availability is 0.99556. We can see similar availability when we put a hot-standby for the fog and edge solutions side by side. However, the fog-based solution has better processing and storage capacity.

The cloud-based solution achieved better results when analyzed with no redundancy. The scenario that allowed redundancy and a 2oo3 configuration obtained an availability of 0.99767, representing a better availability when compared to the other cases. When considering the redundancy in the office node, the cloud-based solution did better, achieving a downtime of 9.18 hours. However, we also emphasize that despite being a good option in terms of availability, the cloud-based solution can lead to network delays, compromising the proper functioning of the preferred solution.

REFERENCES

[1] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3758–3773, 2018.

[2] H. Lee, A. Moon, K. Moon, and Y. Lee, "Disease and pest prediction iot system in orchard: A preliminary study," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2017, pp. 525–527.

[3] D. Venkatesh, M. Tatti, P. Hardikar, S. Ahmed, and K. Sharavana, "Cold storage management system for farmers based on iot," *Int J Recent Trends Eng Res. ISSN*, pp. 2455–1457, 2017.

[4] A. Regattieri, M. Gamberi, and R. Manzini, "Traceability of food products: General framework and experimental evidence," *Journal of food engineering*, vol. 81, no. 2, pp. 347–356, 2007.

[5] A. Kaloxylos, R. Eigenmann, F. Teye, Z. Politopoulou, S. Wolfert, C. Shrank, M. Dillinger, I. Lampropoulou, E. Antoniou, L. Pesonen *et al.*, "Farm management systems and the future internet era," *Computers and electronics in agriculture*, vol. 89, pp. 130–144, 2012.

[6] [Online]. Available: https://apps.fas.usda.gov/psdonline/app/index.html

[7] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud computing: Principles and paradigms*. John Wiley & Sons, 2010, vol. 87.

[8] R. kamal Kaur, B. Pandey, and L. K. Singh, "Dependability analysis of safety critical systems: Issues and challenges," *Annals of Nuclear Energy*, vol. 120, pp. 127–154, 2018.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[10] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, 1982.

[11] W. Ahmad, O. Hasan, and S. Tahar, "Formal dependability modeling and analysis: A survey," in *Intelligent Computer Mathematics: 9th International Conference, CICM 2016, Bialystok, Poland, July 25-29, 2016, Proceedings 9*. Springer, 2016, pp. 132–147.

[12] S. L. Dhulipala and M. M. Flint, "Series of semi-markov processes to model infrastructure resilience under multihazards," *Reliability Engineering & System Safety*, vol. 193, p. 106659, 2020.

[13] P. M. Frank and M. Eslami, "Introduction to system sensitivity theory," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, no. 6, pp. 337–338, 1980.

[14] E. Araujo, J. Dantas, R. Matos, P. Pereira, and P. Maciel, "Dependability evaluation of an iot system: A hierarchical modelling approach," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 2121–2126.

[15] I. Kutyauripo, M. Rushambwa, and L. Chiwazi, "Artificial intelligence applications in the agrifood sectors," *Journal of Agriculture and Food Research*, p. 100502, 2023.

[16] V. Kharchenko, M. Kolisnyk, I. Piskachova, and N. Bardis, "Reliability and security issues for iot-based smart business center: architecture and markov model," in *2016 Third International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*. IEEE, 2016, pp. 313–318.

[17] A. Strielkina, V. Kharchenko, and D. Uzun, "Availability models of the healthcare internet of things system taking into account countermeasures selection," in *International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications*. Springer, 2018, pp. 220–242.

[18] P. Pereira, J. Araujo, C. Melo, V. Santos, and P. Maciel, "Analytical models for availability evaluation of edge and fog computing nodes," *The Journal of Supercomputing*, vol. 77, no. 9, pp. 9905–9933, 2021.

[19] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, and Y. Zhang, "Reinforcement learning-based mobile offloading for edge computing against jamming and interference," *IEEE Transactions on Communications*, vol. 68, no. 10, pp. 6114–6126, 2020.

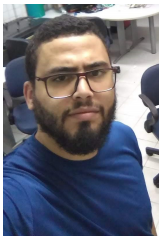
[20] A. K. Mortensen, P. Lisouski, and P. Ahrendt, "Weight prediction of broiler chickens using 3d computer vision," *Computers and Electronics in Agriculture*, vol. 123, pp. 319–326, 2016.

[21] E. Bashir and M. Luštrek, "The mercury environment: A modeling tool for performance and dependability evaluation," in *Intelligent Environments 2021: Workshop Proceedings of the 17th International Conference on Intelligent Environments*, 2021, vol. 29, p. 16.

[22] S. Wolfram, *Mathematica: a system for doing mathematics by computer*. Addison Wesley Longman Publishing Co., Inc., 1991.

[23] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[24] W. Keesee, "A method of determining a confidence interval for availability," NAVAL MISSILE CENTER POINT MUGU CA, Tech. Rep., 1965.



Felipe Oliveira graduated in Computer Science from the Universidade Federal Rural de Pernambuco in 2019. He achieved his master's degree (2023) in Computer Science from the Universidade Federal de Pernambuco, Brazil. His research interests include dependability, high-performance computing, and capacity planning. It also includes stochastic models covering applied areas such as cloud computing and the Internet of Things. He is a member of the Brazilian Computer Society and IEEE.

Paulo Pereira graduated in Computer Science from the Universidade Federal Rural de Pernambuco in 2017. He achieved his master's degree (2019) in Computer Science, and currently, he is a doctoral candidate from Universidade Federal de Pernambuco under supervision of Professor Paulo Maciel. His research interests include performance, reliability, availability, and capacity planning. It also includes stochastic models covering applied areas such as edge, fog, and cloud computing.

Jamilson Dantas received his B.S. degree in Information Systems from the Centro universitário do Rio São Francisco – UNIRIOS, Brazil, in 2009. They received his M.Sc. degree and PhD in Computer Science from Universidade Federal de Pernambuco – UFPE, Brazil in 2013-2018, where he is currently an Assistant Professor. His research interests include performance and dependability evaluation, Markov chains, Petri nets, and other formal models for analysing and simulating computer and communication systems. He works on research projects encompassing video streaming services, cloud computing systems, and network traffic modeling.

Jean Araujo graduated in Information Systems in 2008 from the Faculdade Sete de Setembro, with a master's and a doctorate in Computer Science from the Universidade Federal de Pernambuco, Brazil, in 2012 and 2017, respectively. He was a professor at the Universidade Federal Rural de Pernambuco from 2013 to 2019. Since 2019 he has been a professor at the Universidade Federal do Agreste de Pernambuco, and since 2022 he has been a researcher at the Computer Science post-graduation program at the Universidade Federal de Sergipe. His research interests include software aging and rejuvenation, performance, availability, reliability, and stochastic modelling, which covers applied areas such as cloud computing and the Internet of Things. He has published more than 30 papers in peer-reviewed journals and more than 50 works at international conferences on computer science topics.

Paulo Maciel graduated in Electronic Engineering in 1987 from the University of Pernambuco, with a master's and a doctorate in Electronic Engineering and Computer Science from the Federal University of Pernambuco. From 1996 to 1997, he was at Eberhard-Karls-Universität Tübingen, Germany, doing a "sandwich internship" during his doctorate. He was a professor at the Department of Electrical Engineering at the University of Pernambuco from 1989 to 2003. Since 2001 he has been a member of the Computer Center at the Federal University of Pernambuco, where he is a Full Professor. In 2011, during a sabbatical year, he was at the Department of Electrical and Computer Engineering at the Edmund T. Pratt School of Engineering at Duke University, USA. His research interests include performance, reliability, availability, reliability and capacity planning, and stochastic models covering applied areas such as cloud computing, sustainable data centers, manufacturing, integration, and communication systems.