UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

FELIPE DIAS DE OLIVEIRA

Avaliação de Disponibilidade de Sistemas de Ambiência Agroindustriais

Recife

2023

FELIPE DIAS DE OLIVEIRA

Avaliação de Disponibilidade de Sistemas de Ambiência Agroindustriais

Dissertação apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciências da Computação.

Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Orientador: Dr. Paulo Romero Martins Maciel

Coorientador: Dr. Jean Carlos Teixeira de Araujo

Recife

2023

FICHA

Felipe Dias de Oliveira

“Avaliação de Disponibilidade de Controle de Ambiência de Sistemas Agroindustriais”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 03/03/2023

BANCA EXAMINADORA

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática / UFPE

_____

Dr. Danilo Mendonça Oliveira
HighQSoft GmbH, Alemanha

_____

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática / UFPE
(**Orientador**)

Aos meus pais Paulo e Maria.

Aos meus irmãos Lucas, Matheus, Poliana e Fabiana.

À minha amada Elaine.

AGRADECIMENTOS

Gostaria de dedicar esta seção de agradecimentos a todas as pessoas especiais que fizeram parte da minha trajetória até o momento. Seu apoio e incentivo foram fundamentais para meu crescimento pessoal e profissional.

Agradeço aos meus pais, Paulo e Maria, por estarem sempre ao meu lado, oferecendo amor, orientação e suporte incondicional em todas as fases da minha vida. Minha gratidão também vai para minha esposa, Elaine, que tem sido uma companheira incansável, me apoiando em todas as decisões e desafios que enfrentamos juntos. Sem seu suporte certamente teria desistido dessa empreitada. Obrigado por acreditar em mim quando até eu mesmo duvidei. Aos meus irmãos, Poliana, Fabiana, Matheus e Lucas, e aos meus sobrinhos, agradeço pela alegria e carinho que compartilhamos.

Dedico esse trabalho em memória dos tios Geraldo Alves e Ivo Alves, e das minhas queridas avós, Lucinda e Isabel, guardo lembranças afetuosas e sou grato pelo tempo que tive ao lado deles.

Aos amigos Paulinho, Luan e Erick, sou grato por me receberem em suas casas e proporcionarem momentos descontraídos e memoráveis. Vocês são amigos que considero como meus irmãos. Ao meu amigo David, mesmo à distância, seu apoio e incentivo foram essenciais para continuar firme em minha jornada de pesquisa.

Não posso deixar de agradecer ao meu coorientador, Jean, e ao meu orientador, Paulo, pela orientação valiosa e contribuições essenciais em meu trabalho acadêmico. Obrigado pela paciência e empenho nessa jornada. Também expresso minha gratidão aos membros da banca, Danilo e Jamilson, pela avaliação cuidadosa e sugestões valiosas que ajudaram a melhorar meu trabalho.

Por fim, aos integrantes do grupo MoDCs, agradeço pelas experiências compartilhadas, conhecimento compartilhado e enriquecimento mútuo.

Expresso minha mais sincera gratidão a todos vocês por fazerem parte da minha jornada, tornando-a mais significativa e enriquecedora. Cada um de vocês contribuiu de maneira única para o meu crescimento e sucesso, e por isso, sou profundamente grato por tê-los em minha vida. Muito obrigado!

"Tudo seria fácil se não fossem as dificuldades."

—BARÃO DE ITARARÉ (BERNARDO, 2021)

RESUMO

Os aplicativos de *Internet-of-Things* (IoT) equipam os produtores rurais com ferramentas de apoio à decisão e soluções automatizadas que aumentam a produtividade, a qualidade e o lucro do agronegócio. No entanto, a maioria dos avicultores ainda utiliza métodos convencionais de operação em que trabalhadores humanos realizam todas as rotinas de monitoramento e controle de suas granjas em detrimento de uma maior produtividade. Uma dessas atividades humanas é a pesagem manual, que pode ser substituída por métodos não invasivos, como aplicativos de visão computacional que estimam o peso de aves vivas por meio de câmeras de vídeo. Como os dispositivos IoT podem ter baixo poder de computação, limitando a capacidade de processar os dados localmente, eles podem transferi-los para um *data center* em nuvem ou em nuvem, onde são processados. Este trabalho tem como objetivo realizar um estudo de disponibilidade de um aviário automatizado com um sistema baseado em visão computacional para estimar o peso das aves considerando modelos hierárquicos (por exemplo, cadeia de Markov, Reliability Block Diagram (RBD) e equação de forma fechada) para representar todo o sistema e obter disponibilidade de estado estacionário e tempo de inatividade anual. Além disto, o objetivo é considerar e comparar diferentes soluções arquitetônicas, como soluções baseadas em *edge* e *fog computing*. A solução proposta verificou que uma aplicação baseada em nuvem sem redundância apresenta um *downtime*de 34,14 e 9,176 horas quando considerada uma estratégia de redundância *hot-standby* no nó escritório de uma solução em nuvem.

Palavras-chaves: Internet das coisas; Granjas Inteligentes; Análise de Disponibilidade; Análise de Sensibilidade; Cadeias de Markov.

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 dissertation aims to conduct an availability 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 (RBD), 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.

Keywords: Internet of Things; Smart Poultry Farms; Availability analysis; Sensitivity Analysis; Markov Chain.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABPA	Associação Brasileira de Proteína Animal
API	*Application Programming Interface*
AS	Análise de sensibilidade
CTMC	*Continuous-time Markov chain*
DDR3	*Double Data Rate 3*
DTMC	*Discrete Time Markov Chain*
ERP	*Enterprise Resource Planning*
GB	*Gigabyte*
GHz	*Gigahertz*
GPD	Ganho de Peso Diário
HD	*High Definition*
HTTP	*Hypertext Transfer Protoco*
ICA	Índice de Conversão Alimentar
IEA	Índice de Eficiência Alimentar
IEP	Índice de Eficiência Produtiva
IoT	*Internet-of-Things*
IP	*Internet Protocol*
KooN	*K-out-of-N*
MQTT	*Message Queuing Telemetry Transport*
MTTF	*Mean Time to Fail*
MTTR	*Mean Time to Repair*
NA	*Não Aplicável*
NFC	*Near Field Communication*
RAM	*Random-Acess Memory*
RBD	*Reliability Block Diagram*

RF	Fator de Redução
SBC	*Smart Business Center*
SDRAM	*Synchronous Dynamic Random-Access Memory*
SLA	*Service Level Agreement*
SoC	*System on Chip*
SPN	*Stochastic Petri Nets*
SSD	*Solid State Drive*
TTR	Tempo para Restaurar
UFPE	Universidade Federal de Pernambuco

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, rápidos avanços tecnológicos foram incorporados a dispositivos eletrônicos com conexão remota a objetos comuns do dia a dia. Os chamados objetos inteligentes são eletrônicos, veículos ou imóveis com capacidade de comunicação e automação de algumas tarefas que antes eram realizadas mediante intervenção humana. A capacidade computacional presente nesse tipo de objeto permite a criação de uma nova rede interconectada de dispositivos inteligentes. Cada dispositivo permite a criação de objetos capazes de receber, manipular e transmitir informações.

Nesse contexto, o termo *Internet-of-Things* (IoT) refere-se à rede global que é criada pela interconexão de objetos inteligentes (MIORANDI et al., 2012). As aplicações de dispositivos IoT estão presentes em diversos campos do conhecimento humano e novas aplicações de IoT surgem a todo momento em setores como saúde (DOUKAS; MAGLOGIANNIS, 2012), transporte e logística (SICARI; RIZZARDI; COEN-PORISINI, 2019), automação residencial (PARK et al., 2017) e outros.

No agronegócio, os aplicativos de IoT estão equipando os produtores rurais com ferramentas de apoio à decisão e soluções automatizadas que permitem um avanço na produtividade, qualidade da produção e lucratividade da propriedade (ELIJAH et al., 2018). Soluções como essas podem:

(i) Estimar o aparecimento de pragas na cultura e, consequentemente, reduzir o uso frequente de inseticidas e fungicidas (LEE et al., 2017);

(ii) Controlar as condições ideais de temperatura e umidade nas instalações de armazenamento da produção (VENKATESH et al., 2017);

(iii) Disponibilizar aos produtores e consumidores um registro atualizado e seguro da trajetória dos alimentos em todas as etapas da cadeia produtiva (REGATTIERI; GAMBERI; MANZINI, 2007);

(iv) Gerenciar dados técnicos, estratégicos e operacionais de toda a propriedade rural para planejamento, documentação e otimização de processos (KALOXYLOS et al., 2012); e outros.

Do ponto de vista organizacional, a adoção de aplicativos IoT traz uma série de benefícios, como a rápida obtenção de informações do campo, auxiliando na tomada de decisões administrativas ou mesmo na adoção de medidas que melhorem a produção.

No Brasil, o agronegócio avícola é uma indústria muito importante. O país é o terceiro maior produtor globalmente (Tabela 1), terceiro maior consumidor doméstico (Tabela 2) e o primeiro quando se considera os maiores exportadores (Tabela 3).

Tabela 1 – Os 5 países que mais produziram carne de frango em 2022

País	**Produção (toneladas)**
Estados Unidos	20.005.000
China	14.300.000
Brasil	14.250.000
Rússia	4.750.000
México	3.940.000

Fonte: Elaborada pelo autor baseada em PSD (2023).

Tabela 2 – Os 5 países que mais consumiram carne de frango em 2022

País	**Consumo (toneladas)**
Estados Unidos	17.706.000
China	14.415.000
Brasil	9.810.000
México	4.842.000
Rússia	4.750.000

Fonte: Elaborada pelo autor baseada em PSD (2023).

Tabela 3 – Os 5 países que mais exportaram carne de frango em 2022

País	**Consumo (toneladas)**
Brasil	4.445.000
Estados Unidos	3.317.000
Tailândia	1.035.000
Turquia	510.000
China	530.000

Fonte: Elaborada pelo autor baseada em PSD (2023).

Este setor da economia tem uma importância significativa tanto para mercados externos como para o mercado interno, uma vez que no Brasil a produção de produtos dessa indústria vem crescendo nos últimos anos de acordo com a Associação Brasileira de Proteína Animal (ABPA)(ABPA, 2023). Cerca de 69% da produção de proteína animal oriunda de carne de frango é destinada para consumo interno, enquanto 31% é destinado às exportações brasileiras. Vale destacar a importância da produção de ovos para a região nordeste do Brasil, em especial para o estado de Pernambuco. O estado é o oitavo maior produtor de ovos do Brasil em 2019, correspondendo a 5,51% da produção nacional.

O ganho de produtividade desse setor está altamente ligado à aquisição e implementação de novas tecnologias, como na construção de galpões autônomos e mais eficientes com sistemas de alimentação e ventilação automáticos, melhoramentos genéticos com a adoção de raças mais produtivas e utilização de probióticos na alimentação das aves, o que melhora a absorção de nutrientes pelas aves, aumentando a conversão de ração em carne. Todas essas melhorias que vêm sendo adotadas têm contribuído para a ascensão do desempenho da produção ano a ano (FREITAS; BERTOGLIO, 2001).

No entanto, devido à falta de dados sobre o uso de IoT na indústria avícola no Brasil, assume-se que - como na Malásia (MANSOR et al., 2018) - a maioria dos avicultores ainda usa métodos convencionais de operação onde todas as rotinas de monitoramento e controle de suas fazendas são realizadas por trabalhadores humanos. Essas tarefas são totalmente automatizáveis em alguns casos, o que reduziria a intervenção humana e diminuiria o estresse das aves, e por consequência, poderia trazer uma maior produtividade. Uma dessas atividades realizadas por humanos é a pesagem manual de uma amostra aleatória de aves, um processo intrusivo que pode estressar os animais, é trabalhoso, demorado, suscetível a erros de transcrição e sujeito a erro humano (TURNER; GURNEY; BELYAVIN, 1983), (DOYLE; LEESON, 1989).

Por outro lado, são propostos métodos não intrusivos utilizando aplicativos de visão computacional (MOLLAH et al., 2010), (MORTENSEN; LISOUSKI; AHRENDT, 2016), que estimam o peso de aves vivas utilizando câmeras de vídeo. Esse tipo de técnica possui o mínimo de intervenção humana, é de custo relativamente baixo e possui taxas de erro aceitáveis para o acompanhamento de ganho de peso das aves. Outro exemplo de utilização de dispositivos inteligentes na avicultura é a obtenção de dados de ambiência, que são informações a respeito do ambiente onde as aves se encontram. É possível, por meio de sensores, obter informações como temperatura e umidade, detectar a presença de gases nocivos como monóxido ou dióxido de carbono, além do monitoramento da qualidade da água que é servida aos animais.

Isso demonstra que é possível ter sistemas que tornem essa tarefa totalmente automatizada, em que a interação humana se limita ao controle do sistema, ajustando parâmetros de interesse. Adicionalmente, é viável incorporar outros sensores, como os de luminosidade e velocidade do vento, o que aumenta a quantidade de informações sobre os aviários e auxilia os técnicos avícolas na identificação e resolução de problemas. Apesar das vantagens, é importante lembrar que a implantação de sistemas de aviários inteligentes requer investimentos significativos para a aquisição dos equipamentos necessários e mudanças na cultura e rotina do produtor rural. Entretanto, os benefícios em termos de aumento de produtividade e redução de custos operacionais podem ser bastante significativos, além de contribuir para a sustentabilidade e bem-estar animal.

Do ponto de vista da implementação de um sistema de informação que utilize dispositivos inteligentes para a aquisição de dados, pode-se levar em consideração vários tipos de abordagens e soluções arquitetônicas, tais como *cloud*, *edge* e *fog computing*.

Cada um desses paradigmas computacionais tem vantagens e desvantagens. Por exemplo, a computação em nuvem (BUYYA; BROBERG; GOSCINSKI, 2010) geralmente tem armazenamento ilimitado e alta capacidade de processamento, garantias de acordo de nível de serviço (*Service Level Agreement* (SLA)) e centros de dados distribuídos geograficamente que aumentam a disponibilidade. A possibilidade de ajustar a capacidade computacional virtualizada de acordo com as necessidades, sem sobredimensionamento, é um diferencial que impacta diretamente nos custos de manutenção de um sistema desse tipo, uma vez que é comum pagar somente pelo que for utilizado. Ao necessitar de mais recursos na computação em nuvem há a possibilidade de auto escalonamento, o que configura novos recursos de forma automática e os libera quando não são mais necessários. Por outro lado, altas latências podem ser observadas, tornando-o inviável para sistemas críticos sensíveis à latência.

Os paradigmas *edge* e *fog* aproximaram o poder computacional dos usuários, reduzindo os *overheads* da rede. No entanto, nessas abordagens, espera-se menos disponibilidade do que uma nuvem. Na computação em *fog* têm-se que os dados gerados pelos dispositivos são transmitidos para um servidor intermediário que está entre a *edge* e a nuvem o que reduz a latência de rede enquanto é possível garantir algum maior poder computacional. Já na *edge* os dados são transmitidos para um dispositivo com baixa capacidade de processamento, mas que se encontra na mesma rede que os sensores e atuadores que estão gerando os dados. Isso diminui o tempo de transmissão, porém devido à baixa capacidade de processamento e armazenamento desses dispositivos, há limitações que podem inviabilizar esse tipo de arquitetura

dependendo da natureza da aplicação.

Propõe-se neste trabalho um estudo de disponibilidade de um aviário automatizado com sensores de ambiência e um sistema de captura de peso baseado em visão computacional. Para isso, sugere-se modelos de cadeia de Markov de tempo contínuo e de *Reliability Block Diagram* (RBD) para representar o sistema e obter algumas métricas de confiabilidade, como disponibilidade. Foram consideradas diversas soluções arquitetônicas, tais como abordagens fundamentadas em computação em nuvem, *edge* e *fog*, e efetuaram-se comparações. Adicionalmente, realizou-se uma análise de sensibilidade para identificar os componentes sensíveis do sistema, destacando pontos essenciais para a otimização da disponibilidade do sistema.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

A adoção de dispositivos inteligentes na indústria avícola pode ser uma ferramenta importante de suporte à decisão para os técnicos responsáveis pelos aviários. Atualmente, para obter informações sobre as instalações sob sua responsabilidade, o técnico precisa se deslocar para visitar os galpões avícolas, que muitas vezes estão a uma distância considerável uns dos outros. Ao chegar ao galpão, o técnico precisa estar equipado para realizar medições, verificar as instalações para aferir a qualidade da água, observar o comportamento das aves e realizar pesagens. Isso demanda um tempo considerável do técnico, que consegue visitar, no máximo, um ou dois galpões por dia. Como geralmente um técnico é responsável por vários galpões, isso acarreta em uma amostragem pequena de informações, além de poucas visitas ao longo do ciclo de alojamento das aves.

Ao adotar uma solução de sensoriamento inteligente baseada em IoT, os dados seriam enviados de forma automática e diária ao escritório do técnico. Com dados mais detalhados, o técnico pode ter uma visão mais precisa do que está acontecendo nos galpões, obtendo *insights* sobre possíveis problemas e realizando sua visita ao galpão munido de informações importantes para dar suporte à identificação de problemas e às soluções que ele irá adotar. Isso reduz o número de visitas técnicas, tornando-as mais direcionadas e objetivas, além de reduzir as chances de contaminação dos aviários por patógenos que podem ser transportados pela entrada de pessoal contaminado.

Nesse contexto, a adoção de um sistema de informação baseado em IoT é de grande valor para o acompanhamento de aviários. É desejável que tal sistema seja altamente confiável, uma vez que um aviário é um ambiente sensível e qualquer alteração brusca no ambiente pode

ocasionar o adoecimento ou mesmo a morte de animais. Assim, técnicas que indiquem como melhorar e garantir níveis de disponibilidade de tal sistema podem ser de grande valia para profissionais responsáveis por sua implantação e manutenção. Por meio da sugestão de uma arquitetura *baseline* e da criação de modelos matemáticos que serão utilizados para o cálculo da disponibilidade dos componentes da arquitetura, é possível realizar estudos de disponibilidade do sistema, identificando componentes importantes para a maximização da disponibilidade.

O resultado dessa pesquisa permite que o profissional responsável pela implantação do sistema identifique pontos relevantes em termos de disponibilidade, além da utilização dos modelos matemáticos para auxiliar na tomada de decisão sobre qual será a arquitetura utilizada. Além disso, há a possibilidade de aplicação dos resultados obtidos em ambiente real, uma vez que o estado de Pernambuco é um dos maiores produtores de carne e ovos de frango, o que pode acarretar em melhorias de produtividade para os produtores locais.

1.2 TRABALHOS RELACIONADOS

Esta seção apresenta uma seleção de referências bibliográficas dos últimos anos, que são fundamentais para compreender as contribuições reais desta pesquisa.

O trabalho mencionado no item (ARAUJO et al., 2019) modelou o sistema IoT de um edifício inteligente e apresentou uma estratégia de modelagem baseada em modelos hierárquicos usando a *Continuous-time Markov chain* (CTMC). A partir dos modelos propostos, os autores estimaram a disponibilidade em estado estacionário comparando uma infraestrutura local e um sistema de computação em nuvem, considerando uma estratégia de redundância *cold standby*. Além disso, eles realizaram uma Análise de sensibilidade (AS) para verificar quais componentes do sistema mais afetaram a disponibilidade do sistema. No entanto, os autores não consideraram outros tipos de arquitetura, como "*edge*" e "*fog computing*", como feito em neste trabalho.

Em (KHARCHENKO et al., 2016), foram examinados os desafios de confiabilidade e segurança de uma rede de centro de negócios inteligente (*Smart Business Center* (SBC)) baseada em IoT, e um modelo de Markov foi construído para demonstrar um alto nível de proteção contra ataques de hackers.

Em relação à aplicação de IoT em cuidados de saúde, (STRIELKINA; KHARCHENKO; UZUN, 2018) propõe modelos de disponibilidade para um sistema IoT que considera falhas e ataques a componentes. O artigo aborda as principais causas de falha em um sistema de saúde e

apresenta modelos de cadeia de Markov para configurar a infraestrutura de IoT.

Ainda em relação aos cuidados de saúde inteligentes, em (ANDRADE; NOGUEIRA, 2020; MACIEL et al., 2017), é proposta uma abordagem baseada em rede de Petri (*Stochastic Petri Nets* (SPN))para modelagem e análise de soluções de recuperação de desastres para infraestruturas de IoT. Os autores podem medir métricas como disponibilidade do sistema, custos e tempo de recuperação (Tempo para Restaurar (TTR)) de um sistema IoT de saúde com os modelos propostos. Além disso, a análise de sensibilidade também pode ser realizada para avaliar o efeito da variação dos parâmetros do modelo na disponibilidade.

Em (PEREIRA et al., 2021a), modelos de cadeia de Markov são propostos para modelar aplicações sensíveis à latência, como um sistema de reconhecimento facial de segurança em uma estação de trem. Além disso, os autores propõem modelos de disponibilidade considerando arquiteturas *edge* e *fog*, realizando uma análise de disponibilidade orientada a capacidade e uma avaliação de custo.

No trabalho referido no item (TANG; XIE, 2021), os autores utilizam modelos de cadeia de Markov para avaliar um sistema de *IoT* na área da saúde. Eles obtêm algumas métricas, como a disponibilidade, e apresentam um método para melhorar o desempenho da disponibilidade, a fim de diminuir a indisponibilidade do sistema.

No estudo intitulado (LUFYAGILA; MACHUVE; CLEMEN, 2022), os pesquisadores delineiam uma análise da efetividade inerente à adoção de sistemas de monitoramento de instalações avícolas embasados em IoT, em detrimento de abordagens tradicionais. Com tal propósito, é apresentada uma proposição de uma plataforma de monitoramento remoto e seguro das condições ambientais, acessível mediante um portal online. Entretanto, os autores não realizam uma avaliação substancial da eficácia do sistema proposto para aprimorar a produtividade das aves, realizando comparações com métodos convencionais.

No âmbito da aplicação de inovações tecnológicas computacionais na produção agroindustrial, referenciada por (SINGH et al., 2020), os autores realizam uma análise detalhada sobre a aplicação da tecnologia contemporânea na monitorização da saúde em fazendas avícolas, visando aumentar a produtividade. O estudo abrange o uso de dispositivos IoT com diversos sensores, processamento de dados visuais e sonoros, e análise das vocalizações das aves. A infraestrutura de comunicação e as operações em nuvem têm um papel crucial no processo, divididas em abordagens centrada em objeto e orientada a IoT. A incorporação de tecnologias modernas, como Inteligência Artificial e IoT, pode melhorar substancialmente a monitorização da saúde das aves, impactando positivamente a eficiência. A proposta da arquitetura RESTful

para o desenvolvimento de ambientes de nuvem IoT é relevante, e a consideração de sensores diversificados e processamento de dados é vital para detectar sinais precoces de doenças. Esta abordagem enfatiza a necessidade de pesquisas adicionais para explorar o potencial da IA e IoT na avicultura.

Em (RODRIGUES et al., 2021), os autores propõem uma arquitetura hospitalar inteligente projetada para transmitir dados de pacientes em tempo real aos profissionais médicos, aprimorando assim a qualidade dos cuidados de saúde. Utilizando modelos SPNs, os autores destacam as vantagens da arquitetura proposta, evidenciadas pela redução dos tempos de espera, maior satisfação dos pacientes e otimização da utilização de recursos.

Este trabalho propõe o uso de cadeias de Markov de tempo contínuo e modelos *Reliability Block Diagram* (RBD) para um sistema de aviário inteligente. Realiza-se uma análise de disponibilidade considerando diferentes abordagens arquitetônicas (como soluções com *edge*, *fog* e *cloud computing*) com diferentes configurações de redundância. Também é efetuada uma análise de sensibilidade para identificar os principais componentes que afetam a disponibilidade do sistema.

A Tabela 4 mostra um comparativo entre os trabalhos aqui mostrados.

Tabela 4 – Comparação dos trabalhos relacionados

Trabalho	**Aplicação**	**Modelos**	**AS**
(ARAUJO et al., 2019)	IoT em edificações	CTMC	Sim
(KHARCHENKO et al., 2016)	SBC IoT	CTMC	Não
(STRIELKINA; KHARCHENKO; UZUN, 2018)	IoT na saúde	CTMC	Não
(ANDRADE; NOGUEIRA, 2020)	IoT na saúde	SPN	Sim
(MACIEL et al., 2017)	IoT na saúde	SPN	Sim
(PEREIRA et al., 2021a)	IoT na segurança	CTMC	Não
(TANG; XIE, 2021)	IoT na saúde	CTMC	Sim
(LUFYAGILA; MACHUVE; CLEMEN, 2022)	IoT na agroindústria	NA	Não
(SINGH et al., 2020)	IoT na agroindústria	NA	Não
(RODRIGUES et al., 2021)	IoT na saúde	SPN	Sim
(KASAREDDY; MUKHOPADHYAY, 2022)	IoT na agroindústria	NA	Não
Este trabalho	IoT na agroindústria	CTMC e RBD	Sim

Fonte: Elaborada pelo autor.

Observa-se que todos os trabalhos abordam a análise de disponibilidade em diferentes tipos de sistemas IoT. As técnicas de modelagem e análise utilizadas também variam, incluindo Rede de Petri e CTMC. As métricas avaliadas incluem disponibilidade, segurança contra ataques,

custos e tempo de recuperação (TTR). Os autores também consideraram diferentes abordagens arquitetônicas, como *edge*, *fog* e *cloud computing*. A análise de sensibilidade é usada em alguns trabalhos para identificar os principais componentes que afetam a disponibilidade do sistema.

1.3 OBJETIVOS

O objetivo deste trabalho é a construção de modelos matemáticos que auxiliem o planejamento de infraestrutura de sistemas de aviários inteligentes com alta disponibilidade. A metodologia utilizada descreve etapas para a construção de um modelo matemático, sua validação e análise de sensibilidade para a identificação da influência dos componentes na disponibilidade. Assim, os objetivos do trabalho são:

- Projetar e construir uma arquitetura básica para sistemas IoT para suporte de aviários inteligentes;
- Gerar modelos utilizando os formalismos RBD e Cadeias de Markov considerando diferentes arquiteturas de implementação;
- Identificar componentes que tenham influência significativa na disponibilidade da arquitetura.

1.4 ORGANIZAÇÃO DO TRABALHO

A organização estrutural deste trabalho é a seguinte: no Capítulo 2, é feita uma introdução aos aviários inteligentes. E mais, são apresentadas técnicas de redundância, internet das coisas e análise de sensibilidade e modelagem de sistemas. No Capítulo 3, é apresentada a arquitetura do sistema de aviário inteligente proposto neste trabalho. No Capítulo 4, é descrita a metodologia utilizada neste trabalho, mostrando as etapas realizadas durante sua condução. Também são apresentados os modelos propostos para a análise de dependabilidade do sistema de aviário inteligente e a metodologia adotada para a validação dos modelos. No Capítulo 5, são realizados estudos de caso, a validação dos modelos propostos e estudos de disponibilidade considerando diferentes cenários de redundância e arquiteturas de sistema (*edge*, *fog* e *cloud computing*). Para além disso é realizada uma análise de sensibilidade para identificar os principais componentes que afetam a disponibilidade do sistema. Por fim, o Capítulo 6 apresenta

as principais conclusões deste trabalho, bem como as limitações e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo faz uma descrição dos principais conceitos para a compreensão deste trabalho. Aqui são apresentados os conceitos de aviários inteligentes, internet das coisas, computação em nuvem (*cloud computing*), computação em névoa (*fog computing*), computação em borda (*edge computing*), dependabilidade, técnicas de redundância, análise de sensibilidade além de formalismos de modelagem de disponibilidade.

2.1 AVIÁRIOS INTELIGENTES

A avicultura é a atividade que se refere a criação de aves comerciais, como frangos, galinhas, patos, entre outras aves. Ela é uma atividade econômica importante em muitos países, pois a carne e os ovos dessas aves são fontes importantes de proteína para a alimentação humana. Além disso, a avicultura também é uma fonte importante de emprego em muitas áreas rurais (OLIVEIRA; FILHO; EVANGELISTA, 2008).

Para que haja sucesso nessa atividade econômica, é necessário que sejam seguidos protocolos rigorosos de manejo sanitário, nutricionais e de bem estar animal, garantindo assim a qualidade dos alimentos produzidos (GARCIA; GOMES, 2019). Isso inclui a monitorização regular da saúde das aves, a utilização de rações nutricionais balanceadas e o controle de fatores de estresse, como a temperatura e a umidade do ambiente de criação (VOGADO et al., 2016).

Nesse contexto, os aviários inteligentes são sistemas tecnológicos avançados que monitoram e controlam as condições ambientais dos galpões avícolas, incluindo temperatura, umidade, luminosidade, entre outros fatores importantes para o bem estar das aves, conhecida na literatura zootécnica como **ambiência**. Ademais, tais sistemas coletam dados sobre a produção, como por exemplo a quantidade de alimento consumida e a taxa de crescimento das aves, o que possibilita uma avaliação mais precisa e confiável das condições de criação.

A utilização de aviários inteligentes oferece diversos benefícios para a saúde e o bem-estar das aves, como a manutenção de temperaturas e umidades ideais, além de um controle mais eficiente de fatores estressantes, como barulho e luz. Quando as condições ambientais são inadequadas, as aves podem ficar estressadas, desenvolver doenças e apresentar baixo desempenho de produção. Por outro lado, quando o ambiente é controlado e otimizado, as aves podem se manter saudáveis e produtivas, o que resulta em uma melhor rentabilidade para

o produtor (JUNIOR, 2019).

Além disso, esses sistemas fornecem informações precisas e atualizadas para os responsáveis técnicos, o que permite uma tomada de decisão mais eficiente e baseada em dados confiáveis. Isso pode incluir informações sobre as condições climáticas, o estado de saúde das aves e a produção, permitindo uma gestão mais eficiente e planejada da produção avícola.

2.2 INTERNET DAS COISAS: UMA VISÃO GERAL

A Internet das Coisas (IoT, na sigla em inglês) é um sistema de dispositivos conectados que permite que eles sejam monitorados e controlados remotamente através de uma rede de computadores (BUYYA; DASTJERDI, 2016). Esses dispositivos incluem desde sensores e atuadores, passando por câmeras, termostatos e outros aparelhos eletrônicos. Eles são equipados com tecnologias de conectividade, como Wi-Fi, *Near Field Communication* (NFC) e outras, para permitir a troca de dados entre si e com a nuvem. O objetivo da IoT é fornecer uma visão integrada e controlada das coisas no mundo físico, tornando-as mais inteligentes e interativas. Os sistemas IoT comum envolve três tecnologias principais: sistemas, middleware e serviços em nuvem,

A computação em nuvem (BUYYA; BROBERG; GOSCINSKI, 2010) fornece alto poder computacional e capacidade de armazenamento ilimitada, recursos escassos em dispositivos IoT. Portanto, é intuitivo descarregar os dados para os serviços em nuvem, onde esses dados podem ser processados e armazenados. Porém, como os *data centers* estão distribuídos globalmente, a distância entre o servidor em nuvem e os usuários pode implicar em altas taxas de latência, o que pode ser indesejado em aplicações de tempo real.

A computação em névoa (*fog computing*) é um paradigma de computação distribuída que amplia a capacidade de processamento e armazenamento próximo às fontes de dados, para além da nuvem central. Ele é uma alternativa para lidar com a crescente quantidade de dados gerados pelos dispositivos IoT e outros tipos de sensores e atuadores distribuídos em uma rede (NAHA et al., 2018).

No modelo de nuvem tradicional, todos os dados coletados são transmitidos para a nuvem central para processamento e análise. No entanto, isso pode resultar em congestão na rede e atraso na transferência de dados. A computação em névoa resolve esse problema ao fornecer capacidade de processamento e armazenamento próximo aos dispositivos de borda, permitindo que eles processem e analisem grande parte dos dados localmente antes de enviar apenas os

dados relevantes para a nuvem (DASTJERDI et al., 2016).

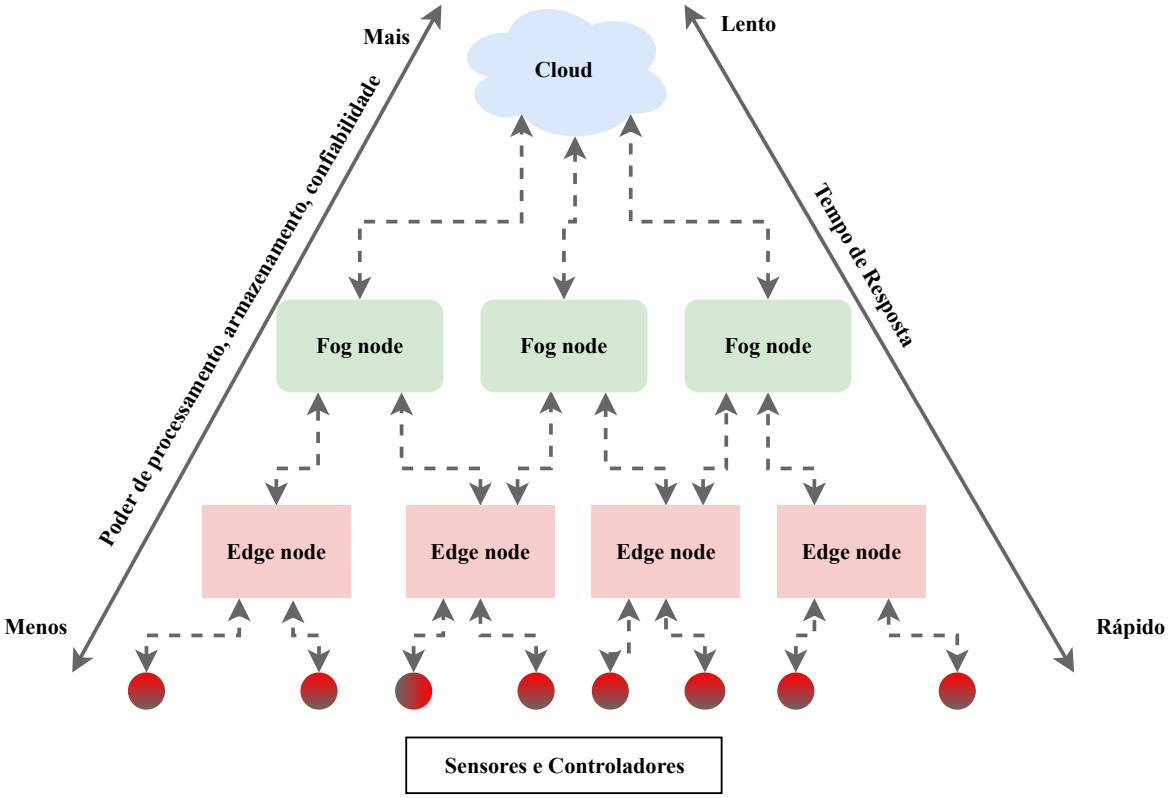
Assim, a computação em névoa permite uma resposta mais rápida e eficiente aos dados coletados, além de uma melhor segurança dos dados, já que parte dos dados são processados e armazenados localmente, sem a necessidade de serem transmitidos para a nuvem central.

Aplicativos baseados em computação em nuvem ou em névoa exigem que todos os dados sejam carregados em um servidor, onde os dados são processados e os resultados são enviados de volta aos dispositivos IoT. Essa situação pode gerar custos de transmissão de dados, como grande latência de rede, o que pode comprometer aplicações sensíveis ao tempo como soluções de cidades inteligentes ou outras (YU et al., 2017).

Na computação em borda, *edge computing*, os nós de borda referem-se a todos os dispositivos que podem garantir algum poder computacional ou capacidade de armazenamento, atuando mais próximo dos sensores e atuadores (SATYANARAYANAN, 2017). Essa proximidade com os recursos na "borda" traz benefícios, que incluem menor latência de transmissão na computação ou armazenamento de dados. Além disso, a computação na borda também pode melhorar a segurança e a privacidade dos dados, já que os dados não precisam ser transmitidos para fora da rede para processamento (LIU et al., 2019; YU et al., 2017).

A Figura 1 mostra a pilha comum que representa cada paradigma. Os nós de processamento em borda adjacentes aos sensores e atuadores estão mais próximos da base da pirâmide. A computação em névoa está a meio caminho da nuvem, tendo mais recursos computacionais do que os nós de borda. Além do mais, a nuvem possui maiores recursos computacionais e armazenamento no topo. É importante destacar que há menos sobrecarga de rede mais perto da base da pilha e menos latência. Por outro lado, mais poder de processamento, armazenamento e confiabilidade são esperados mais perto do topo da pilha.

Figura 1 – IoT *Stack*: Da Borda para a nuvem



Fonte: Elaborada pelo autor baseada em Naha et al. (2018).

2.3 DEPENDABILIDADE

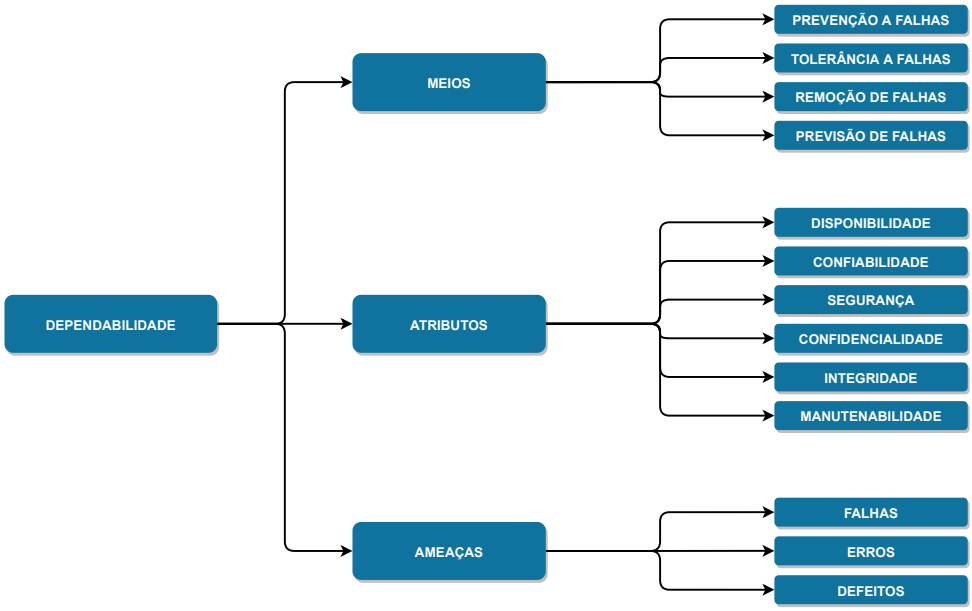
A confiança de um sistema pode ser entendida pela sua capacidade de: entregar um serviço confiável, cumprir adequadamente um conjunto de funcionalidades específicas (MACIEL et al., 2012; FERNANDES et al., 2012) e evitar falhas frequentes e graves de serviço (AVIZIENIS et al., 2004).

O estudo da dependabilidade de um sistema é um processo que relaciona diferentes aspectos que podem comprometer a capacidade de entrega de serviços confiáveis. A Figura 2 mostra uma árvore que esquematiza esses aspectos. Nela vemos vemos os **meios**, **atributos** e **ameaças**.

O conceito de dependabilidade inclui os atributos quantitativos disponibilidade e confiabilidade. Neste trabalho, se tem um interesse particular na **disponibilidade**, ou seja, a probabilidade de um sistema estar em condição de funcionamento, considerando a intermitência de estados operacionais e não operacionais (IEEE, 1990).

Essa probabilidade é representada em termos da razão entre o tempo em que um sistema

Figura 2 – Árvore da dependabilidade



Fonte: Elaborada pelo autor baseada em Avizienis et al. (2004).

está disponível (*uptime*) e o período em que este não está disponível (*downtime*). Assim, a disponibilidade (A) pode ser calculada pela razão entre o *Mean Time to Fail* (MTTF) e o *Mean Time to Repair* (MTTR), conforme mostrado na Equação 2.1. Além disso, a indisponibilidade (U) é calculada usando a Equação 2.2.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.1}$$

$$U = 1 - A \tag{2.2}$$

O MTTR pode ser estimado usando o MTTF, conforme mostrado na Equação 2.3:

$$MTTR = MTTF \times \frac{U}{A} \tag{2.3}$$

O MTTF pode ser descrito integrando a função de confiabilidade conforme mostrado na Equação 2.4. A função de confiabilidade define a probabilidade de um sistema sobreviver a um determinado período sem falhas (MACIEL, 2023a).

$$MTTF = \int_0^\infty R(t)dt \tag{2.4}$$

$$R(t) = e^{-\lambda t}$$

A representação da disponibilidade do sistema pelo número de noves é uma possibilidade (ÖHMANN; SIMSEK; FETTWEIS, 2014), como indicado na Tabela 2.1. Se um sistema tem quatro noves de disponibilidade, por exemplo, pode ser classificado como tolerante a falhas e ter um tempo de inatividade anual de menos de uma hora, enquanto a alta disponibilidade é obtida por meio de sistemas com cinco ou mais noves. O número de noves é calculado pelo expressão $-log_{10}(U)$ (MACIEL, 2023a), onde U é a indisponibilidade do sistema que pode ser calculada conforme a Equação 2.2.

Tabela 5 – Tabela de disponibilidade em número de noves

Número de 9's	Disponibilidade (%)	Downtime Anual
1	90,00	36,5 dias
2	99,00	3,65 dias
3	99,90	8,76 horas
4	99,99	52,56 minutos
5	99,999	5,26 minutos
6	99,9999	31,56 segundos
7	99,99999	3,16 segundos

Fonte: Elaborada pelo autor baseada em Maciel (2023a).

2.4 TÉCNICAS DE REDUNDÂNCIA

As técnicas de redundância são métodos utilizados para garantir a continuidade de serviços e sistemas, mesmo em caso de falhas ou problemas. Essas técnicas são amplamente utilizadas em sistemas críticos, como sistemas de segurança, sistemas financeiros e sistemas de comunicações (JAIN; GUPTA, 2011).

Uma das técnicas mais comuns de redundância é a redundância de hardware. Isso significa que o sistema possui mais de um componente físico para realizar a mesma tarefa. Se um componente falhar, outro pode assumir sua função sem interrupção do serviço. Por exemplo, em sistemas de segurança, pode haver mais de uma câmera para monitorar uma área, ou mais de um dispositivo de armazenamento para salvar informações.

Outra técnica de redundância é a redundância de software. Nesta abordagem, o sistema possui mais de um software para realizar a mesma tarefa. Se um software falhar, outro pode assumir sua função. Esta técnica é comumente usada em sistemas de backup e recuperação de desastres.

Além dessas técnicas, existem outras formas de redundância, como redundância de energia, redundância de conexão e redundância de pessoal, que visam garantir a continuidade dos serviços.

Quando realiza-se um estudo sobre os sistemas de redundância verifica-se basicamente dois tipos: redundância ativa e redundância baseada em espera (*standby*).

Tem-se redundância ativa nos sistemas onde há vários componentes em operação ao mesmo tempo, e um ou mais componentes de backup que estão prontos para assumir o controle em caso de falha de algum componente principal. Em outras palavras, quando há redundância ativa, os componentes de backup estão sempre ativos e prontos para responder, enquanto os componentes principais estão em operação normal. Exemplos de redundância ativa incluem sistemas de backup de energia, sistemas de segurança, sistemas de comunicações, sistemas médicos, entre outros. Nesses sistemas, se um componente principal falhar, o componente de backup é automaticamente acionado sem interrupção, garantindo a continuidade do funcionamento do sistema. A vantagem dos sistemas de redundância ativa é que eles garantem uma alta disponibilidade e confiabilidade do sistema, pois o componente de backup está sempre pronto para responder. Além disso, os sistemas de redundância ativa são projetados para detectar falhas rapidamente e responder rapidamente, minimizando o tempo de inatividade do sistema.

Na redundância baseada em espera, os componentes de backup estão sempre disponíveis, porém não são ativados a menos que seja necessário. Dito de outro modo, os componentes de backup estão em espera, esperando a falha de algum componente principal para entrar em operação. Nesses sistemas, o componente principal é responsável pelo funcionamento normal do sistema, enquanto o componente de backup está pronto para responder em caso de falha. Se o componente principal falhar, o componente de backup é acionado e assume o controle do sistema, garantindo a continuidade do funcionamento. A vantagem desses sistemas é que eles são mais econômicos e mais simples de manter do que os sistemas de redundância ativa, pois os componentes de backup não estão sempre ativos e consumindo recursos. No entanto, há um tempo de resposta mais lento em caso de falha, pois é necessário um tempo para que o componente de backup seja acionado e assuma o controle do sistema, o que pode implicar em indisponibilidade no sistema. A redundância baseada em espera pode ser dividida em três tipos (JAIN; GUPTA, 2011):

- *Hot standby*: o componente de backup está sempre ativo e pronto para responder ime-

diatamente em caso de falha do componente principal. O componente de backup está sincronizado com o componente principal, de modo que pode assumir o controle sem interrupção.

- *Warm standby*: o componente de backup está em modo de espera, mas pode ser iniciado rapidamente em caso de falha do componente principal. Embora o componente de backup não esteja sincronizado com o componente principal, ele é configurado de modo a poder responder rapidamente a uma falha.

- *Cold standby*: o componente de backup não está ativo e pode levar algum tempo para ser iniciado em caso de falha do componente principal. Esse tipo de sistema de redundância é geralmente utilizado em aplicações onde o tempo de inatividade é tolerável e o custo de manter o componente de backup ativo é alto.

2.5 ANÁLISE DE SENSIBILIDADE

A análise de sensibilidade é uma técnica utilizada para avaliar como pequenas variações em determinados fatores afetam o resultado final de um modelo ou sistema. Ela é frequentemente utilizada em áreas como finanças, engenharia, economia e ciência da computação para avaliar o impacto de mudanças em variáveis chave em projetos ou investimentos. A análise de sensibilidade permite aos analistas identificar quais fatores são mais críticos para o sucesso do projeto ou do investimento e avaliar seu potencial impacto. Isso pode ser útil para tomar decisões informadas e preparar estratégias de contingência para lidar com incertezas (MATOS et al., 2015).

A análise de sensibilidade pode ser realizada por meio de modelos matemáticos ou simulações computacionais. Ela pode ser feita com uma única variável ou com várias variáveis ao mesmo tempo, o que permite aos analistas avaliar o impacto de múltiplas mudanças ao mesmo tempo (HAMBY, 1994).

Este trabalho utiliza uma técnica de análise de sensibilidade diferencial que usa um coeficiente de sensibilidade calculado pela razão entre um parâmetro variável dado e a métrica de saída. Em contraste, os demais parâmetros são mantidos constantes. Dessa forma, se deseja-se calcular o coeficiente de sensibilidade de uma métrica Y que depende de um parâmetro θ, deve-se usar a Equação 2.5 ou Equação 2.6 para sensibilidade escalonada (FRANK; ESLAMI, 1980).

$$S_\theta(Y) = \frac{\partial Y}{\partial \theta} \tag{2.5}$$

$$SS_\theta(Y) = \frac{\theta}{Y}\frac{\partial Y}{\partial \theta} \tag{2.6}$$

2.6 MODELAGEM

Esta seção descreve os formalismos utilizados para a criação dos modelos apresentados neste trabalho. Foram utilizados Diagramas de Blocos de confiabilidade e Cadeias de Markov.

2.6.1 Diagrama de Blocos de Confiabilidade

Os diagramas de blocos de confiabilidade (RBD) consistem em blocos que representam os componentes de um sistema e setas que representam as conexões entre eles. Cada bloco é identificado com uma letra ou um número e é associado a uma taxa de confiabilidade, que representa a probabilidade de que o componente funcione corretamente. As setas representam as dependências entre os componentes e as taxas de confiabilidade dessas conexões podem ser calculadas com base nas taxas de confiabilidade dos componentes (SMITH, 2021).

A avaliação da confiabilidade do sistema como um todo é realizada calculando a taxa de confiabilidade combinada dos componentes e suas conexões. Isso é feito por meio de fórmulas matemáticas que levam em consideração as taxas de confiabilidade dos componentes e as dependências entre eles (JAIN; GUPTA, 2011).O resultado da análise é a taxa de confiabilidade do sistema como um todo, que pode ser utilizado para avaliar o desempenho do sistema e identificar pontos fracos que precisam ser melhorados. As principais configurações de um modelo RBD são mostradas na Tabela 2.6.1, que incluem configurações seriais, paralelas e *K-out-of-N* (KooN) (SAHNER; TRIVEDI; PULIAFITO, 2012).

Além disso, os diagramas de blocos de confiabilidade podem ser utilizados para modelar sistemas complexos de maneira simplificada, tornando-os mais fáceis de entender e analisar. Eles também são úteis para identificar pontos de falha críticos no sistema e para avaliar a eficácia de medidas de melhoria da confiabilidade.

2.6.2 Cadeias De Markov

As cadeias de Markov são modelos matemáticos que descrevem o comportamento de um sistema em termos de estados e transições, que podem representar eventos de falha e reparo em sistemas ou componentes. Esses modelos, também conhecidos como processos estocásticos, são úteis para a descrição de análises estatísticas que possuem valores de tempo em seus parâmetros.

Um processo estocástico é um modelo matemático que descreve a evolução de um sistema aleatório ao longo do tempo. Ele é definido como uma sequência de variáveis aleatórias indexadas pelo tempo, em que cada variável aleatória representa o estado do sistema em um determinado momento.

Em termos matemáticos, um processo estocástico pode ser formalmente definido como uma coleção de variáveis aleatórias $X_t, t \in T$ definidas em um espaço de probabilidade $(\Omega, \mathcal{F}, P)$, onde T é um conjunto de índices que representa o tempo, o Ω representa o espaço amostral e o $\mathcal{F}$ representa o espaço de eventos. A coleção $X_t, t \in T$ é chamada de trajetória do processo estocástico (CASSANDRAS; LAFORTUNE, 2006).

A evolução do processo estocástico é descrita por uma função de distribuição de probabilidade conjunta $P(X_{t_1} \leq x_1, X_{t_2} \leq x_2, \ldots, X_{t_n} \leq x_n)$ que descreve a probabilidade de que o processo esteja em um estado específico em diferentes momentos no tempo.

Uma cadeia de Markov é um processo de Markov, no qual a distribuição condicional de um estado futuro depende apenas do estado atual, e não dos estados anteriores. O processo é descrito por uma sequência de variáveis aleatórias discretas, em que o tempo pode assumir um valor discreto ou contínuo. Dependendo do tipo do valor que o tempo assume (se contínuo ou discreto) faz com que as cadeias de Markov se separem em duas classificações: **cadeia de Markov de tempo contínuo** (CTMC) e **cadeia de Markov de tempo discreto** (*Discrete*

Tabela 6 – Modelos de confiabilidade

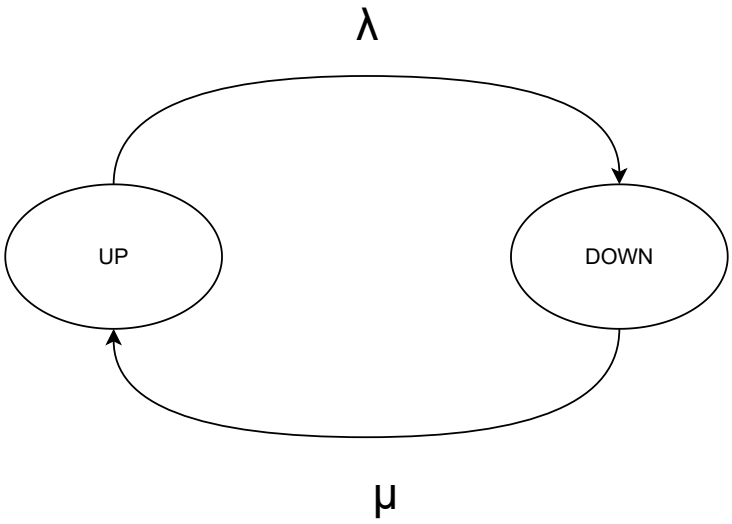
Configuração do Sistema	**Modelo Matemático**
Série	$R_{série} = R_1 \times R_2 \times \cdots \times R_n$
Paralela	$R_{paralela} = 1 - (1 - R_1) \times (1 - R_2) \times \cdots \times (1 - R_n)$
k-out-of-n	$R_{k-out-of-n} = \sum_k^n \binom{n}{k} R^k (1 - R)^{n-k}$

Fonte: Maciel (2023a).

Time Markov Chain (DTMC))

A representação gráfica de uma cadeia de Markov pode ser vista como uma máquina de estados, onde os nós do grafo representam os estados do sistema e as transições entre os estados são representadas pelas arestas. Os modelos podem ser discretos ou contínuos, e a transição entre estados depende exclusivamente do estado atual. A Figura 3 mostra um exemplo básico de cadeia de Markov contendo dois estados UP e DOWN.

Figura 3 – Exemplo de uma cadeia de Markov



Fonte: Elaborada pelo autor baseada em Maciel (2023a).

$$\lambda = \frac{1}{MTTF}$$
$$\mu = \frac{1}{MTTR}$$

As cadeias de Markov são mais convenientes do que os RBDs (Reliability Block Diagrams) quando se deseja descrever propriedades dinâmicas dos sistemas, como a avaliação de desempenho e a análise de dependências entre componentes.

As cadeias de Markov são úteis porque permitem aos analistas prever o comportamento futuro de um sistema com base nas probabilidades atuais. Além disso, uma vez que a modelagem é realizada e as equações fechadas são extraídas, elas são triviais de se utilizar e podem ser utilizadas para modelar sistemas complexos de maneira eficiente.

2.7 CONSIDERAÇÕES FINAIS

Este capítulo mostrou os principais conceitos para a compreensão dessa dissertação. Foi realizada uma visão geral sobre aviários inteligentes e sua importância no monitoramento de dados de ambiência. Logo após alguns conceitos sobre internet das coisas, computação em nuvem, em névoa e na borda foram apresentados. Na sequência, o conceito de dependabilidade foi exposto junto com formas de se calcular a disponibilidade de um sistema, seguido por técnica de redundância. Finalmente, uma breve introdução à análise de sensibilidade foi conduzida, seguida da apresentação das técnicas de modelagem de disponibilidade.

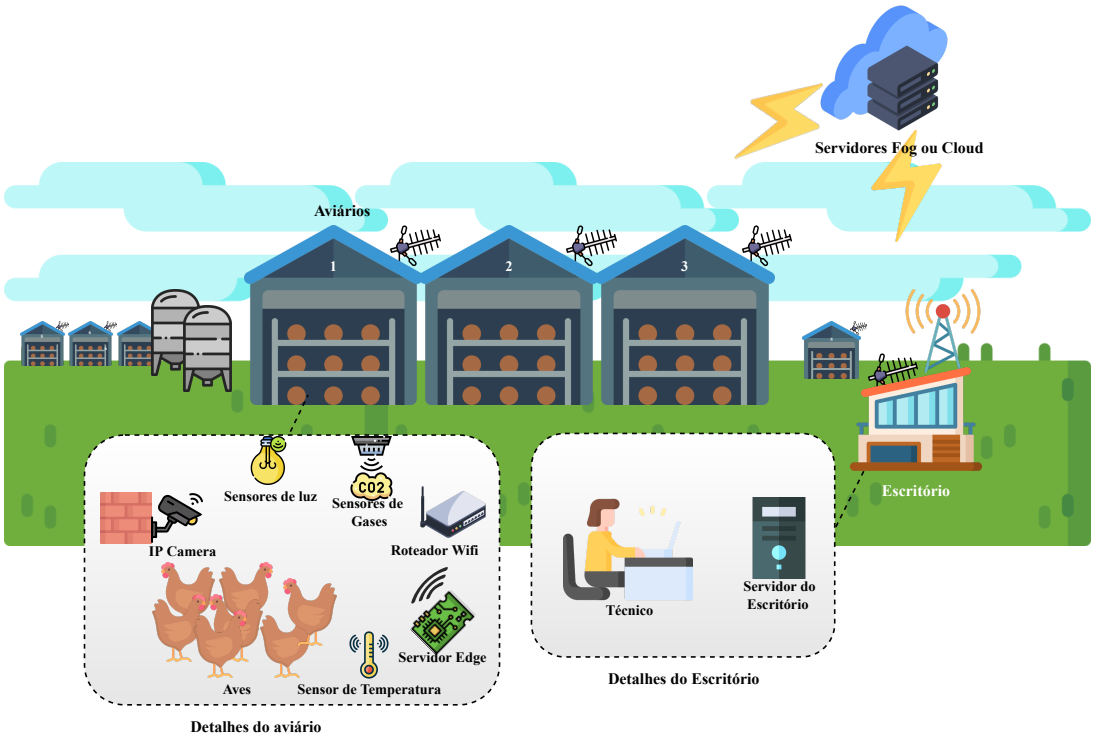
3 ARQUITETURA DO SISTEMA

Quando se trata de um aviário inteligente, se pode obter através de sensores dados de ambiência (como temperatura, luminosidade, qualidade da água, etc.) e usar câmeras IP e processamento de vídeo para estimar o peso das aves (MORTENSEN; LISOUSKI; AHRENDT, 2016). Sugere-se as seguintes soluções arquitetônicas: uma solução baseada em computação na borda, uma solução baseada em computação em névoa e uma solução baseada em computação em nuvem considerando, em todos os casos, uma redundância *hot-standby*. A Figura 4 exibe uma ilustração que demonstra uma visão geral desse sistema.

Essas soluções arquiteturais seguem uma abordagem básica: obter os dados de entrada (vídeo da câmera IP e métricas dos sensores) e transmiti-los a um nó com capacidade computacional suficiente para processar tais dados. No nó de processamento, as imagens são classificadas e os atributos de interesse são extraídos. Os dados obtidos dos sensores (os sensores de temperatura, gás, luz e qualidade da água) também são centralizados no nó de borda e transmitidos ao escritório. O responsável técnico do galpão tem acesso a esses dados por meio de um computador no escritório local. Assim, com as informações de interesse em mãos, ele obtém uma visão geral da situação e pode intervir para fazer as alterações necessárias. Neste cenário, o técnico detém informações sobre um determinado galpão e tem uma visão geral de todos os galpões que gerencia.

O processo de aquisição de dados, transmissão para um servidor de processamento e para o servidor de escritório irá ocorrer de forma automática, sendo fundamental o correto funcionamento de cada componente desse sistema para o trabalho do técnico que está na outra ponta do processo. A Figura 5 mostra como ocorre o processo de comunicação entre os componentes do sistema. Uma vez capturadas as imagens por meio da câmera IP e dos sensores, os dados são transmitidos para o o servidor de processamento por meio dos protocolos HTTP e MQTT. Uma vez processados, esses dados são consumidos por meio de APIs que são consumidas pelo escritório local. O servidor de processamento do escritório local é um importante componente do sistema, pois é através dele que os técnicos tem acesso às informações e conseguem inferir outras como o Ganho de Peso Diário (GPD), Índice de Conversão Alimentar (ICA), Índice de Eficiência Alimentar (IEA) e Índice de Eficiência Produtiva (IEP), além da viabilidade (EMBRAPA, 2023). Outro fator importante que ocorre no escritório local, é a integração com outros sistemas de gestão, como sistemas ERP que gerenciam as demais etapas da produção e da

Figura 4 – Projeto de ambiente: visão geral



Fonte: Elaborada pelo autor.

administração do negócio.

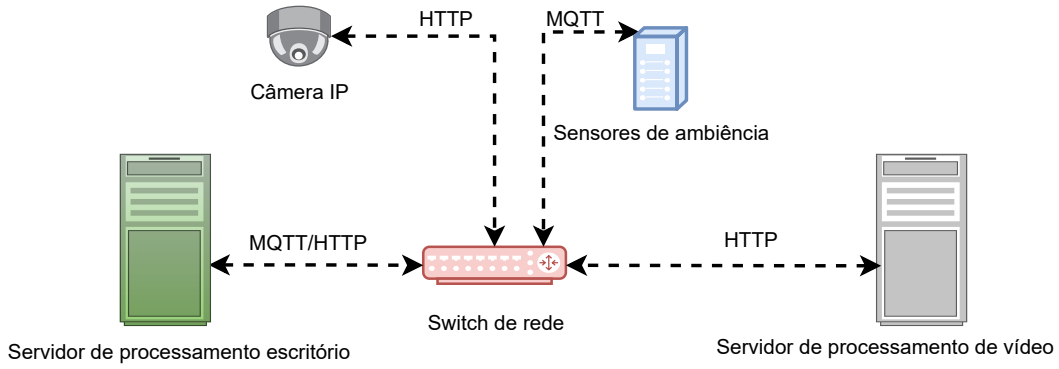
3.1 ARQUITETURA BASEADA EM COMPUTAÇÃO NA BORDA

Uma arquitetura baseada em computação na borda é mostrada na Figura 5. Nesta abordagem, o vídeo capturado pela câmera IP é processado apenas pelo nó de borda. O técnico tem acesso aos dados processados em um escritório local, que é interligado aos componentes do sistema por um *switch* de rede ou roteador. A desvantagem dessa solução é que os recursos de borda são limitados, o que restringe a capacidade de processamento de vídeo.

A arquitetura de um sistema de aviário inteligente baseado em computação na borda apresenta diversas vantagens em relação aos sistemas tradicionais. A capacidade de processamento local permite uma maior agilidade na tomada de decisões, além de reduzir a dependência de conexões de internet estáveis, o que pode ser um grande problema em áreas rurais.

A utilização de sensores distribuídos no aviário possibilita o monitoramento em tempo real de diversos parâmetros como temperatura, umidade, consumo de ração e água, entre outros, gerando um volume de dados significativo. Nesse sentido, a arquitetura apresentada fornece

Figura 5 – Arquitetura com computação em borda: comunicação entre os componentes do sistema



Fonte: Elaborada pelo autor.

uma solução eficiente para lidar com a grande quantidade de dados gerados, possibilitando sua análise e processamento em tempo hábil, uma vez que latências de rede não são aplicáveis já que os dados são processados localmente.

3.2 ARQUITETURA BASEADA EM COMPUTAÇÃO EM NÉVOA

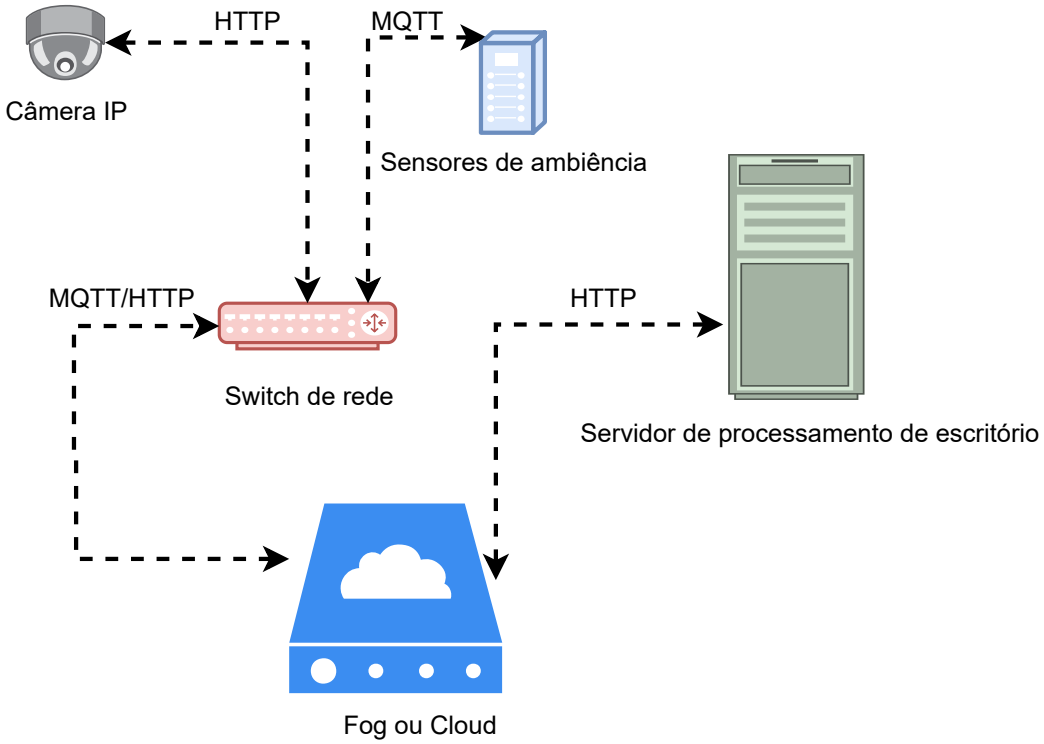
Uma solução baseada em névoa pode ser visualizada na Figura 6. Nesta arquitetura, a responsabilidade do processamento de dados é transferida para um nó na névoa. Este nó de névoa é um servidor que pode estar na mesma rede do escritório ou da câmera IP, mas também pode ser um servidor em outra rede acessível pela Internet, mas não tão longe quanto nuvem.

A vantagem de se utilizar um servidor desse tipo é o ganho em capacidade de processamento, onde é possível processar maiores volumes de dados mantendo os tempos de resposta esperados, já que não é necessário que as informações trafeguem até a nuvem.

A solução baseada em nuvem também pode ser visualizada na Figura 6. Os dados geralmente são transferidos para um servidor em nuvem que os processa e retorna os resultados para o aplicativo de escritório nessa abordagem. Nessa arquitetura, a latência da rede é esperada, visto que os administradores da infraestrutura acessam os servidores em nuvem pela Internet.

A adoção de arquiteturas baseadas em computação em nuvem ou na névoa, apesar de trazer maiores latências de rede, tem a vantagem de permitir um provisionamento mais rápido e escalável do que soluções baseadas em computação na borda. Isso ocorre porque os recursos disponibilizados através da internet são virtualizados, o que permite que sejam rapidamente alocados quando há necessidade de aumentar a capacidade computacional. Por outro lado, quando esses recursos não são mais necessários, podem ser rapidamente liberados, otimizando

Figura 6 – Arquitetura com computação em névoa ou nuvem: comunicação entre os componentes do sistema



Fonte: Elaborada pelo autor.

os custos da operação. Na computação em borda, esse processo de alocação de recursos é mais lento, e uma vez que os recursos não são mais necessários, eles ficam ociosos.

3.3 CONSIDERAÇÕES FINAIS

Este capitulo apresentou uma visão geral da solução modelada. Na sequência, uma arquitetura baseada em computação em borda foi considerada e, por fim, uma arquitetura baseada em computação em névoa ou nuvem.

4 METODOLOGIA E MODELOS

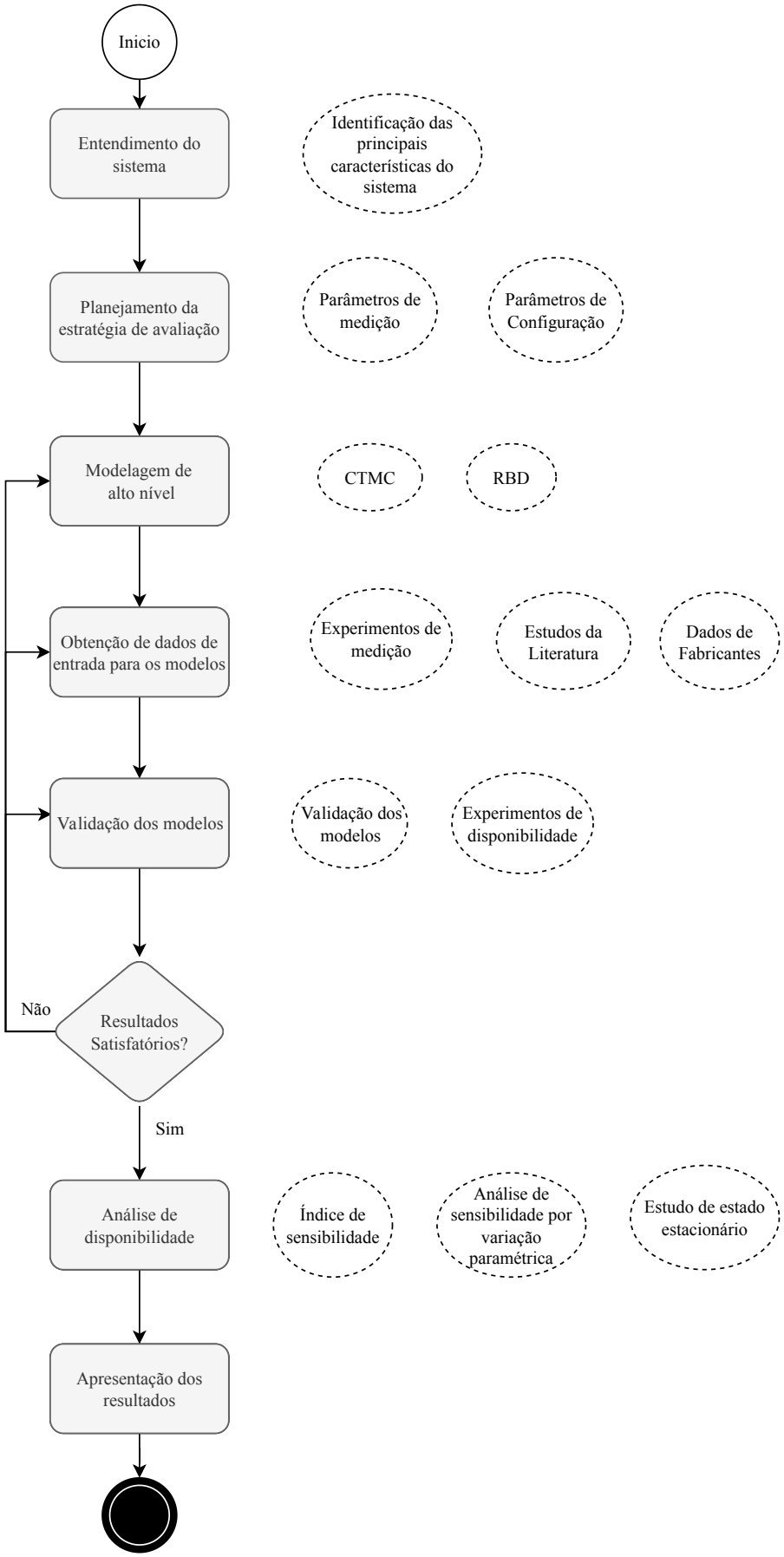
Este capítulo apresenta a metodologia utilizada para a execução desta dissertação e os modelos utilizados. Ele está dividido nas seguintes seções: Metodologia Geral, Modelos e Metodologia de Validação de Modelos. Por fim, são realizadas as considerações finais do capítulo.

4.1 METODOLOGIA GERAL

A metodologia adotada foi dividida em sete atividades: entendimento do sistema, planejamento da estratégia de avaliação, proposição de modelos de alto nível, obtenção de dados de entrada para os modelos, validação dos modelos, avaliação de disponibilidade e apresentação dos resultados.

A Figura 7 apresenta um fluxograma da metodologia adotada neste trabalho. Cada retângulo representa uma etapa e foi seguida obedecendo à ordem indicada pelas setas. Convencionou-se que somente após a finalização de uma etapa, o avaliador passa para a etapa seguinte. O losango representa uma pergunta que pode resultar em caminhos distintos dependendo da resposta obtida. Os círculos tracejados representam possíveis atividades a serem realizadas em cada etapa (ou seja, ações, análises, estratégias, etc.). As setas tracejadas indicam as possíveis instanciações que poderiam ser adotadas para o cumprimento da etapa. É importante ressaltar que todas as instanciações de cada etapa foram executadas ao longo do desenvolvimento do trabalho. Cada atividade da metodologia é descrita com detalhes adiante.

Figura 7 – Metodologia para avaliação de disponibilidade



Fonte: Elaborada pelo autor.

Entendimento do sistema: Para avaliar a disponibilidade de sistemas IoT na avicultura, é necessário compreender como o sistema funciona, identificando todos os seus componentes e realizando um levantamento sobre soluções existentes, aplicações e funcionalidades, com o objetivo de delimitar uma configuração para a infraestrutura que se pretende modelar. Essa atividade demanda atenção e cuidado por parte do avaliador, a fim de evitar erros de interpretação que possam comprometer as demais etapas da metodologia. Essa etapa é fundamental para a correta execução das demais etapas, uma vez que a partir dela é possível obter conhecimento sobre as arquiteturas existentes, bem como sobre técnicas de modelagem e validação.

Planejamento da estratégia de avaliação: em sequência, é necessário estabelecer qual estratégia de avaliação será adotada para a execução do trabalho. Por estratégia de avaliação, entende-se todas as ações que serão tomadas e decididas, como, por exemplo, configurações do ambiente base, parâmetros e métricas de interesse, técnicas e tecnologias, entre outros. Isso terá impacto direto sobre quais modelos serão adotados e como serão validados. Neste ponto, o avaliador deve levar em consideração aspectos arquiteturais da aplicação para estabelecer quais componentes possuem influência nos níveis de qualidade de serviço.

Modelagem de alto nível: uma vez definidos os parâmetros e configurações do ambiente, além da definição do foco da avaliação, é possível conceber modelos que sejam representações do sistema. Esses modelos são representações de alto nível da infraestrutura computacional adotada e podem ser formulados em diversos formalismos, como RBD ou CTMC. A partir do formalismo adotado, é possível extrair equações fechadas que permitem a verificação de diferentes parâmetros. Para a criação de modelos, são utilizadas ferramentas de modelagem que permitem a edição visual e o cálculo de métricas de interesse. Um exemplo de ferramenta de modelagem utilizada é o *Mercury* (SILVA et al., 2015; BASHIR; LUŠTREK, 2021), que permite a criação de modelos e realização de análises. Como estratégia para a construção de modelos, neste trabalho, alguns componentes dos modelos de alto nível são representados como sub-modelos, permitindo observar métricas de componentes específicos. Detalhes adicionais serão fornecidos na Seção 4.2.

Obtenção de dados de entrada para os modelos: nesta etapa, os dados necessários para a validação do modelo são obtidos. São realizadas ações para coleta de dados por meio de experimentos de medição e extração de dados na literatura. As fontes de dados podem ser diversas, desde dados fornecidos pelos fabricantes dos equipamentos até livros e artigos científicos.

Validação do modelo: na etapa de de validação do modelo são realizadas as ações

necessárias para verificar se há evidências suficientes para que o modelo que foi proposto nas etapas anteriores seja rejeitado por não ser representativo do sistema que se está estudando. Para isso, é necessário realizar um experimento de observação do ambiente onde o sistema está em operação, de preferência em um ambiente de testes controlados. O objetivo desse experimento é verificar a ocorrência de falhas nos componentes do sistema e como eles são restaurados. Para isso, realiza-se um experimento de injeção de falhas onde são gerados tempos de falha aleatórios exponencialmente distribuídos com média MTTF. Passado o tempo que foi gerado, uma falha é injetada em um dos componentes do sistema e um contador de tempo para o reparo é iniciado. Esse contador irá aguardar até que o tempo de reparo seja atingido, tempo esse que também é gerado por meio de uma distribuição exponencial de média MTTR. Ao mesmo tempo em que falhas são injetadas, o ambiente é monitorado em ordem de obter os tempos de falha e reparo observados para cada componente. Com os tempos de falha e reparo e as quantidades de falhas e reparos, são realizados tratamentos estatísticos para obter os tempos médios de falha e reparo e com isso calcular o intervalo de confiança da disponibilidade. Caso a disponibilidade do sistema obtida ao inserir os MTTFs e MTTRs no modelo proposto esteja dentro do intervalo de confiança calculado, considera-se o modelo validado sem evidências para o rejeitar. Mais detalhes serão fornecidos na Seção 4.3.

Avaliação de disponibilidade: na etapa de avaliação de disponibilidade são realizadas ações para calcular métricas de interesse a fim de atingir níveis mínimos de disponibilidade. Essa disponibilidade visa atender às expectativas dos usuários e administradores dos sistemas, garantindo a qualidade dos serviços computacionais. Para isso, é fundamental que os modelos propostos sejam representativos da infraestrutura computacional do sistema. Após a obtenção das métricas de interesse para a análise da disponibilidade, como o *uptime* e *downtime* anual, é possível verificar se os resultados obtidos são satisfatórios ou não. Além disso, técnicas de análise de sensibilidade são aplicadas nesse ponto do trabalho visando identificar quais componentes do sistema possuem maior influência nas métricas de disponibilidade. Diferentes estratégias de avaliação de disponibilidade podem ser adotadas, como, por exemplo, variações paramétricas. Com os resultados das análises de disponibilidade e sensibilidade, é possível propor mudanças no sistema que maximizem a disponibilidade.

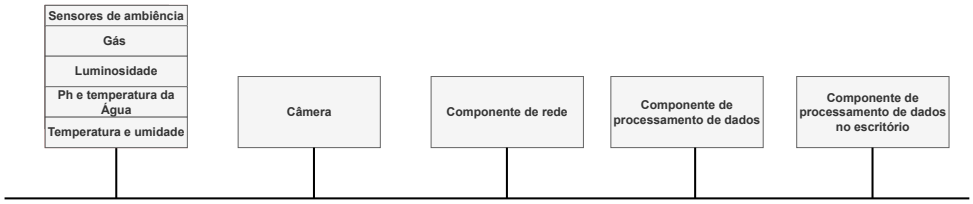
Apresentação dos resultados: a última etapa, apresentação dos resultados, é onde os resultados obtidos são agrupados e organizados em gráficos e tabelas, o que permite uma melhor compreensão das contribuições obtidas e a realização de recomendações.

4.2 MODELAGEM

Esta seção apresenta modelos de disponibilidade para o ambiente descrito no Capítulo 3. Para isso foram utilizadas cadeias de Markov e modelos RBD para representar e estabelecer a ligação entre os componentes do sistema. Esses formalismos são escolhidos porque oferecem equações fechadas para calcular a disponibilidade, que são mais úteis e fáceis de aplicar, e são processadas mais rapidamente (DANTAS et al., 2016).

Para realizar a modelagem do sistema de forma a captar as características inerentes do sistema real, se faz necessário elaborar um diagrama que simplifique os componentes do ambiente e estabeleça a relação entre eles, conforme mostrado na Figura 8. A partir desse modelo genérico é possível pensar em como modelar o sistema utilizando um formalismo adequado.

Figura 8 – Modelo genérico do sistema

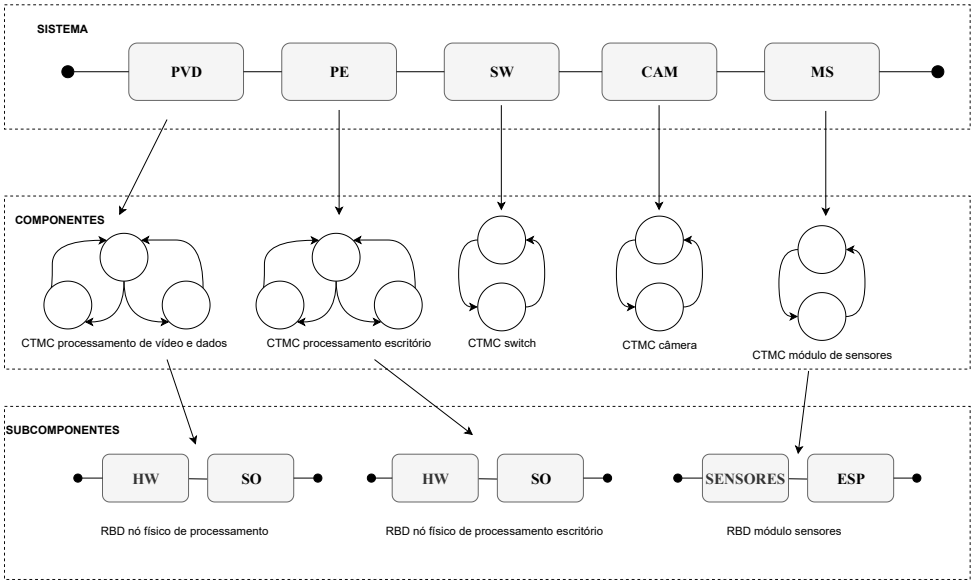


Fonte: Elaborada pelo autor.

A Figura 9 mostra o modelo hierárquico heterogêneo que é o principal resultado dessa pesquisa. Esse modelo é hierárquico pois existe uma hierarquia entre os componentes do sistema, e heterogêneo pois cada componente do sistema é representado por um formalismo diferente, fazendo com que haja uma relação entre o modelo do sistema, os componentes do sistema e os subcomponentes destes componentes. O RBD no topo da Figura 9 representa o sistema de aviário inteligente mostrado na Figura 5 e simplificado na Figura 8.

A partir do RBD se extrai a equação de disponibilidade do sistema. Desta forma, se pode calcular a disponibilidade para todo o sistema, utilizando-se a Equação 4.1. Por se tratar de um RBD em série, onde o correto funcionamento do sistema depende que todos os componentes que compõem esse sistema funcionem corretamente, temos que a disponibilidade do sistema A_{sys} é calculada pelo produto das disponibilidades dos demais componentes do sistema. Por tanto para se obter a disponibilidade do sistema pelo modelo geral, deve-se primeiro obter a disponibilidade dos componentes. A disponibilidade desses componentes estão representadas como A_{PDV} sendo a disponibilidade do nó de processamento de vídeo e dados, A_{PE} a disponibilidade do nó de processamento do escritório, A_{SW} sendo a disponibilidade

Figura 9 – Modelo hierárquico heterogêneo



Fonte: Elaborada pelo autor.

do *switch* de rede, A_{CAM} a disponibilidade da câmera e, por fim, A_{MS} a disponibilidade do módulo de sensores.

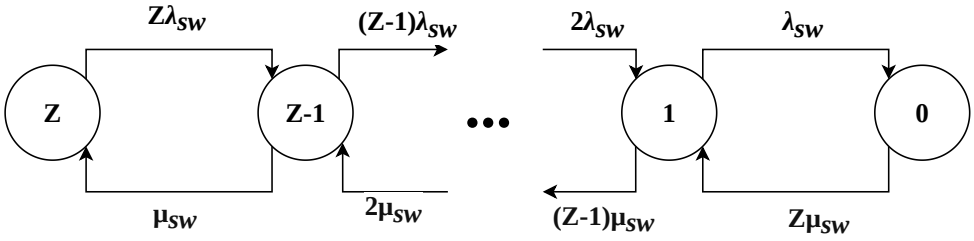
Vê-se na Figura 9 que o próximo nível descreve os modelos CTMC que representam os componentes do sistema. As CTMCs da figura são apenas para fins ilustrativos da hierarquia adotada, uma vez que os modelos reais serão mostrados na sequência.

$$A_{sys} = A_{PDV} \times A_{PE} \times A_{SW} \times A_{CAM} \times A_{MS} \tag{4.1}$$

As Figuras abaixo mostram os componentes do modelo de cadeia de Markov proposto. O modelo compreende cinco componentes: a câmera, o *switch* de rede, o módulo dos sensores, o nó de processamento e o nó do escritório.

Em um modelo *baseline*, foi considerado um cenário funcional mínimo composto por um *switch*, um módulo sensor, uma câmera, um computador de escritório executando um aplicativo e um nó de processamento executando uma instância do software responsável pelo processamento do vídeo. Essa estratégia tem por objetivo simplificar o cenário *baseline*, mas o modelo considera ter um ou mais de cada componente, ou seja, é possível que em outros cenários se tenham redundâncias de cada elemento que compõe a arquitetura. Então, por exemplo, o modelo pode ter um, dois ou Z *switches* de rede, conforme a Figura 10. Na mesma direção, pode-se ter de 1 até W câmeras (Figura 11); de 1 à M nós de processamento, onde cada nó pode hospedar de 1 à L aplicativos em execução (Figura 13).

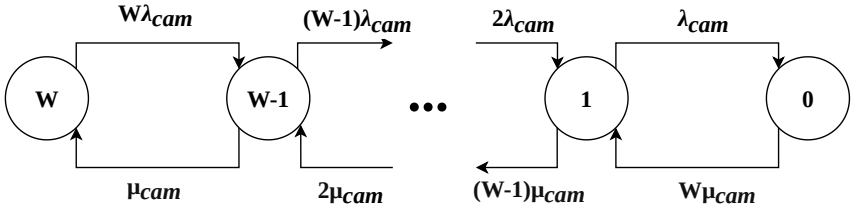
Figura 10 – Modelo CTMC do *switch* de rede



Fonte: Elaborada pelo autor.

$$A_{SW} = \sum_{j=1}^{Z} \frac{Z!}{j!\,(Z-j)!} \times \frac{\lambda_{sw}^{Z-j}\mu_{sw}^{j}}{(\lambda_{sw}+\mu_{sw})^{Z}} \tag{4.2}$$

Figura 11 – Modelo CTMC da câmera

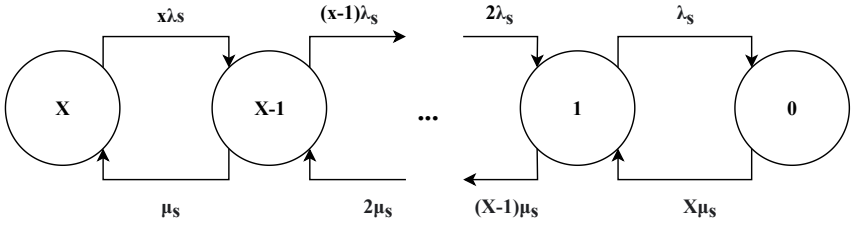


Fonte: Elaborada pelo autor.

$$A_{CAM} = \sum_{k=1}^{W} \frac{W!}{k!\,(W-k)!} \times \frac{\lambda_{cam}^{W-k}\mu_{cam}^{k}}{(\lambda_{cam}+\mu_{cam})^{W}} \tag{4.3}$$

De maneira similar ao que foi apresentado no modelo quando mostrado o modelo para os componentes câmera e *switch*, o modelo do componente módulo de sensores mostrado na Figura 12 possui estrutura semelhante. Conforme pode ser visualizado na Figura 12, o modelo prevê a possibilidade de 1 único módulo até X réplicas.

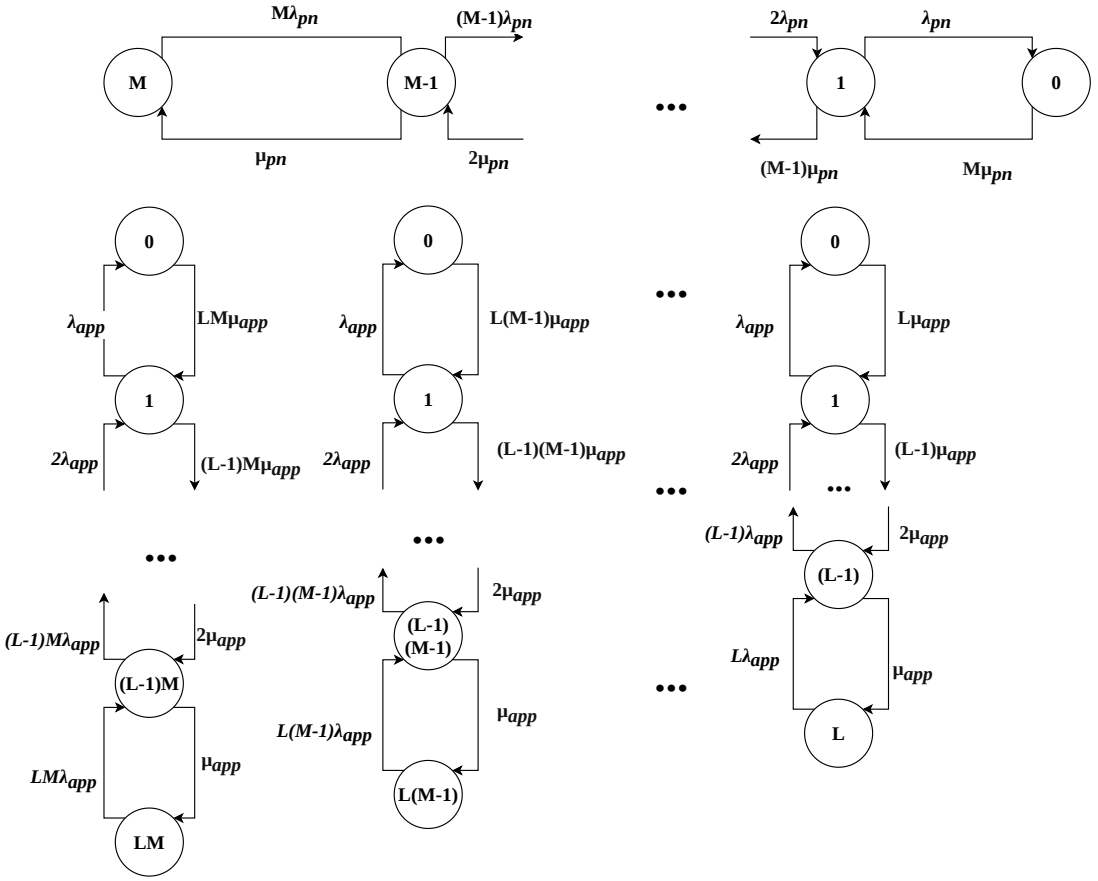
Figura 12 – Modelo CTMC do módulo de sensores



Fonte: Elaborada pelo autor.

$$A_{MS} = \sum_{q=1}^{X} \frac{X!}{q!\,(X-q)!} \times \frac{\lambda_{s}^{X-q}\mu_{s}^{q}}{(\lambda_{s}+\mu_{s})^{X}} \tag{4.4}$$

Figura 13 – Modelo CTMC do nó de processamento



Fonte: Elaborada pelo autor.

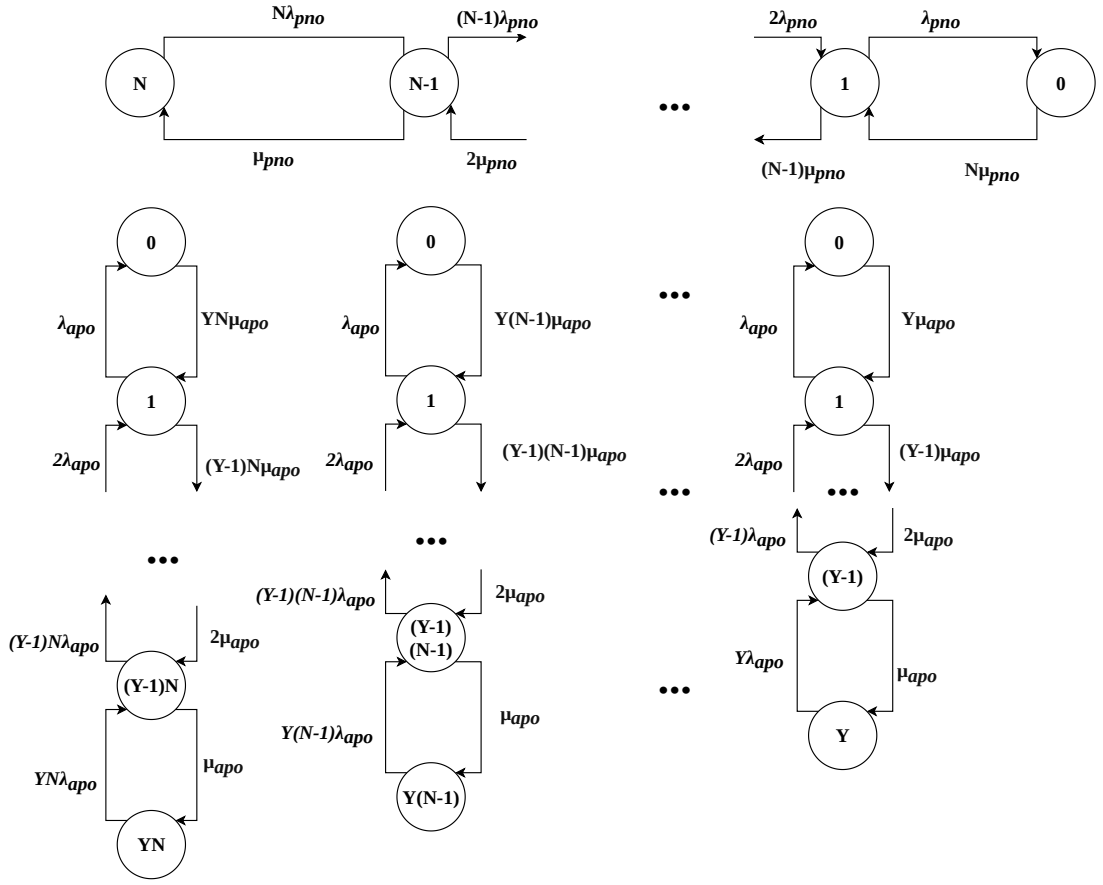
Um ou vários aplicativos podem ser executados no modelo de nós de processamento e escritório. Esta aplicação pode ser o software que classifica as imagens ou o software de análise de escritório. Por exemplo, se houver um nó de processamento ativo, se pode ter até L aplicativos rodando nesse nó. Por outro lado, há até $2L$ aplicativos disponíveis se houver dois nós de processamento ativos. Portanto, se for considerada uma determinada quantidade de nós de processamento M, se pode ter ML aplicativos em execução.

$$A_{PVD} = \sum_{i=1}^{M} \left(1 - \frac{\lambda_{app}^{iL}}{(\lambda_{app} + \mu_{app})^{iL}}\right) \times \frac{M!}{i!\,(M-i)!} \times \frac{\lambda_{pn}^{M-i}\mu_{pn}^{i}}{(\lambda_{pn} + \mu_{pn})^{M}} \tag{4.5}$$

O modelo de nó de processamento de escritório tem uma função semelhante. Cada nó de escritório pode ter um ou até Y aplicativos em execução em qualquer nó. Como o modelo suporta até N nós de processamento de escritório, pode se ter, se houver o máximo de nós de processamento de escritório, até NY aplicativos de escritório em execução.

$$A_{PE} = \sum_{p=1}^{N} \left(1 - \frac{\lambda_{apo}^{pY}}{(\lambda_{apo} + \mu_{apo})^{pY}}\right) \times \frac{N!}{p!\,(N-p)!} \times \frac{\lambda_{pno}^{N-p}\mu_{pno}^{p}}{(\lambda_{pno} + \mu_{pno})^{N}} \tag{4.6}$$

Figura 14 – Modelo CTMC do nó de processamento escritório



Fonte: Elaborada pelo autor.

Cada *switch* pode ter uma falha e uma taxa de reparo (representadas respectivamente por λ_{sw} e μ_{sw}), bem como cada câmera (λ_{cam} e μ_{cam}). O nó de processamento tem uma taxa de falha λ_{pn} e uma taxa de reparo μ_{pn}. Cada aplicativo em execução no nó de processamento também possui taxas de falha e reparo (λ_{app}, μ_{app}). Finalmente, a taxa de falha do nó do escritório é representada por λ_{pno} e a taxa de reparo por μ_{pno}. Por sua vez, o aplicativo de escritório também possui taxas de falha e reparo (λ_{appo}, μ_{appo}). O modelo aqui apresentado assume que cada componente e réplica são idênticos e que o sistema possui equipes de reparo suficientes e independentes.

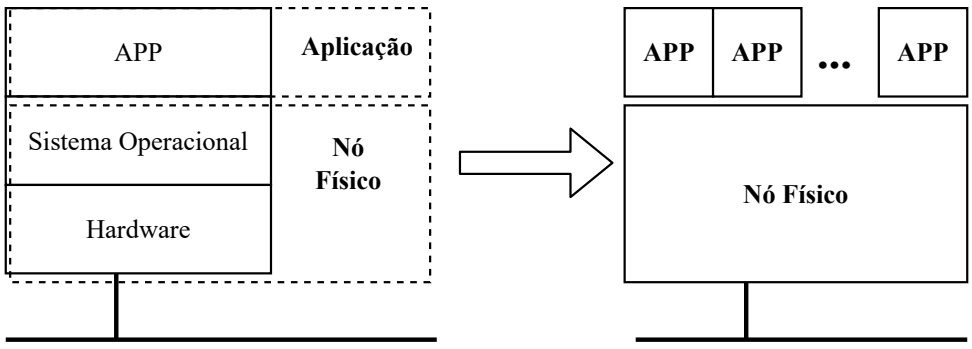
Da cadeia de Markov se pode extrair as equações de disponibilidade mostradas adiante. Essas equações são importantes pois com elas é possível testar mudanças na infraestrutura sem que haja a implementação em uma infraestrutura real de fato. Isso significa que é mais econômico do que comprar e implementar todos os componentes de hardware e software que compõe a infraestrutura. Primeiro, a equação de disponibilidade foi obtida para cada componente: Equação 4.2 para o *switch*, Equação 4.3, Equação 4.4 para o módulo sensor,

Equação 4.5 para os nós do servidor de processamento e Equação 4.6 para os nós do servidor de escritório.

O terceiro nível do modelo hierárquico, mostrado na Figura 9, mostra os modelos utilizados para representar os subcomponentes que são utilizados nos modelos CTMC. Esses subcomponentes são o nó físico e o módulo de sensores que são representados nesta camada como modelos RBD. O modelo RBD dos nós de processamento é fiel ao à abstração realizada para o componente de processamento, porém o RBD do módulo de sensores mostrado na Figura 9 está simplificado e é mostrado com detalhes abaixo.

Primeiro, para construir o modelo de cadeia de Markov, foi necessário obter um componente que representa a abstração que foi realizada na Figura 15. Se optou por considerar que o sistema operacional e o hardware na pilha são um único componente que foi chamado de nó físico. Essa simplificação permite que o modelo seja mais simples, excluindo detalhes desnecessários que não podem refinar o modelo. Para isso, foi utilizado um o modelo RBD e também realizada a extração da Equação 4.7, da qual se pode obter a disponibilidade de uma instância de nó físico. O modelo RBD obteve o MTTF e MTTR relacionados aos componentes *edge*, *fog*, *cloud* e *office node* físicos por uma avaliação exata no *Mercury Tool* (SILVA et al., 2015). Com este modelo RBD, foi realizada uma análise de confiabilidade usando a ferramenta de modelagem Mercury.

Figura 15 – Abstração de um nó geral

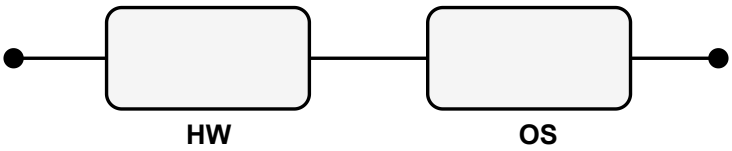


Fonte: Elaborada pelo autor baseada em Pereira et al. (2021b).

O MTTF e MTTR de cada bloco físico são mostrados na Tabela 7 e todos esses valores são extraídos da literatura (PEREIRA et al., 2021b).

A partir do RDB mostrado na Figura 16, calcula-se o MTTF do componente pela Equação 4.7. Já a função de confiabilidade é calculada de acordo com a Equação 4.8.

Figura 16 – Modelo RBD de nó físico



Fonte: Elaborada pelo autor baseada em Pereira et al. (2021b).

$$MTTF_{NF} = \int_0^\infty R_{NF}(t)dt \tag{4.7}$$

$$R_{NF}(t) = R_{HW}(t) \times R_{OS}(t) = e^{-\lambda_{HW}t} \times e^{-\lambda_{OS}t} \tag{4.8}$$

Com a informação do se pode calcular a taxa de falha do nó físico onde ele for aplicado, representado por $\lambda_{NF} = \frac{1}{MTTF_{NF}}$. Essa informação é utilizada como parâmetro de entrada das CTMC mostradas nas Figuras 13 e 14.

Também se pode considerar o conjunto de sensores como um componente único chamado módulo sensor e usar um RBD em série para representá-lo. Os componentes do módulo sensor são um microcontrolador *ESP32*, um sensor de temperatura, um sensor de gás (que capta os gases tóxicos, como CO2 e amônia), um sensor de luminosidade que serve para medir a intensidade da luz e um sensor de qualidade da água, que mede *ph*, temperatura e turbidez da água.

Figura 17 – Modelo RBD do módulo de sensores

Fonte: Elaborada pelo autor.

Equação 4.14 se refere à equação de confiabilidade para uma instância do módulo sensor extraída do RBD em série descrito anteriormente. A confiabilidade do módulo sensor é representada pelo produto da confiabilidade do componente esp32 (R_{ESP32}, mostrada na Equação 4.9), a confiabilidade do sensor de temperatura (R_{ST}, Equação 4.10), a confiabilidade do sensor de gás (R_{SG}, Equação 4.11), a confiabilidade do sensor de luz (R_{SL}, Equação 4.12), e a confiabilidade dos sensores de qualidade da água (A_{SQA}, Equação 4.13).

$$R_{ESP32} = e^{-\lambda_{ESP32}t} \tag{4.9}$$

$$R_{ST} = e^{-\lambda_{ST}t} \tag{4.10}$$

$$R_{SG} = e^{-\lambda_{SG}t} \tag{4.11}$$

$$R_{SL} = e^{-\lambda_{SL}t} \tag{4.12}$$

$$R_{SQA} = e^{-\lambda_{SQA}t} \tag{4.13}$$

$$R_{MS}(t) = R_{ESP32}(t) \times R_{ST}(t) \times R_{SG}(t) \times R_{SL}(t) \times R_{SQA}(t) \tag{4.14}$$

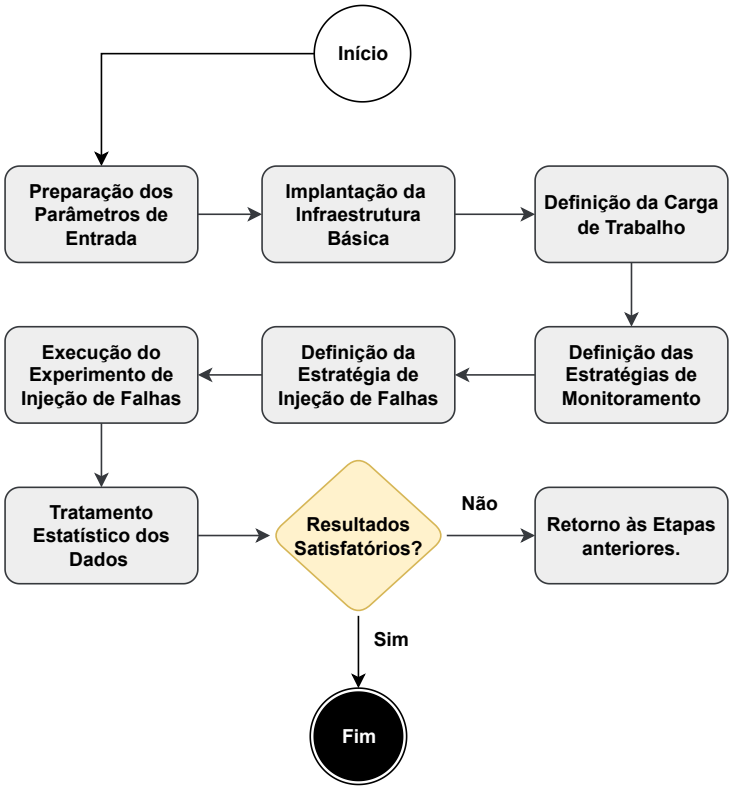
$$MTTF_{MS} = \int_0^\infty R_{MS}(t)dt \tag{4.15}$$

Com a equação de confiabilidade do módulo de sensores é possível calcular o MTTF do módulo de sensores (Equação 4.15) e com isso obter a taxa de falha do módulo de sensores $\lambda_{MS} = \frac{1}{MTTF_{MS}}$, que servirá de entrada para o modelo mostrado na Figura 12.

4.3 ESTRATÉGIA DE VALIDAÇÃO DE MODELOS

A estratégia adotada no processo de validação do modelo é mostrada na Figura 18. Essa metodologia é composta de sete atividades (preparação dos parâmetros de entrada, implantação da infraestrutura básica, definição da carga de trabalho, execução do experimento de injeção de falhas, definição da estratégia de injeção de falhas, definição das estratégias de monitoramento e tratamento estatístico dos dados) e uma pergunta. Cada uma dessas atividades é descrita em detalhes adiante.

Figura 18 – Metodologia utilizada na condução da validação dos modelos



Fonte: Elaborada pelo autor.

Preparação dos Parâmetros de Entrada: uma falha pode demorar muito para ocorrer em um sistema real operacional. De forma a contornar essa limitação, as ocorrências de falhas são aceleradas no experimento. Para isso é considerado um Fator de Redução (RF), que é uma estratégia para acelerar a ocorrência de falhas. Para aplicar essa estratégia divide-se MTTF original pelo RF e o resultado é utilizado como entrada no experimento de injeção de falhas (COSTA et al., 2016). Adotou-se um RF de 876. Esse fator de redução faz com que um ano do sistema real (1 ano tem 8760 horas) seja representado em dez horas. Esse fator de redução é aplicado apenas nos tempos de falha. Se for aplicado ao tempo de reparo, será prejudicial a análise, uma vez que o tempo de reparo será tão pequeno que provavelmente não será detectado no monitoramento.

Considera-se que o MTTF e o MTTR, aplicados para injetar as falhas e simular seus reparos, conforme Tabela 7, são distribuídos exponencialmente. A coluna "MTTF RF" da mesma tabela mostra o valor do MTTF após a aplicação do fator de redução. Esses são os valores que foram aplicados como parâmetros no injetor de falhas. Os tempos de falha e recuperação foram utilizados em um experimento de injeção de falhas implementado em uma infraestrutura de arquitetura base.

Implantação da Infraestrutura Básica: para a realização de um experimento de injeção de falhas, é fundamental ter um ambiente controlado para a realização das ações necessárias. Portanto, não é recomendado utilizar ambientes de produção nessa etapa, visto que o objetivo é gerar indisponibilidade no sistema. A infraestrutura básica para a realização do experimento de injeção de falha é mostrado na Figura 19. A configuração do ambiente foi composta por: um servidor de processamento de vídeo representado por um Orange pi WinPlus com processador *AllWinner* A64 SoC, 2GB DDR3 SDRAM e 32GB de armazenamento. O sistema operacional utilizado neste componente foi o Raspbian Server 10, e o aplicativo de processamento de vídeo foi um programa escrito em Python 3. O servidor de processamento no escritório foi representado por um computador pessoal; sua configuração era um processador AMD C-60, 4GB de RAM e 500GB de capacidade de armazenamento, com Debian 10. O injetor de falhas estava sendo executado em outro computador, com processador Intel i7-8565U 1,8GHz, 8GB de RAM, SSD de 256GB e Linux Debian 10. O computador monitor, responsável por realizar o monitoramento do sistema, possui um processador AMD E-300 1.3GHz, com uma memória 4GB DDR3 e 320GB de armazenamento com Debian 10. Como câmera foi utilizada uma webcam Full HD 1080p. O módulo de sensores foi composto por um microcontrolador ESP32[1], junto com um sensor de ph e temperatura da água, um sensor de temperatura e umidade

[1] http://esp32.net/

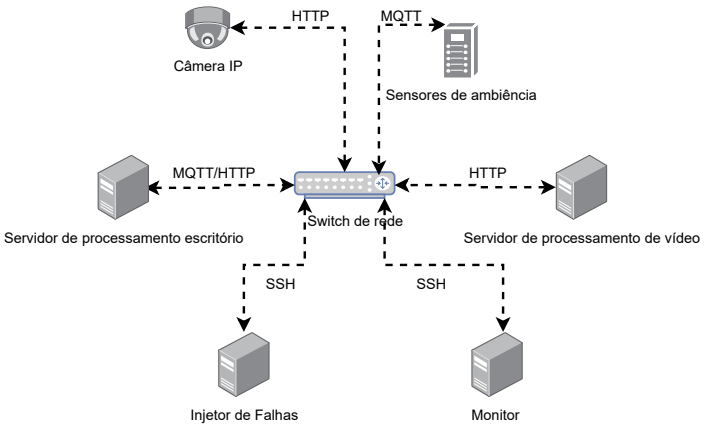
Tabela 7 – Tempos médios de falha e reparo por componente

Componente	MTTF (h)	MTTF RF (h)	MTTR (h)
Câmera	25000	28,53	2
Orange pi	25000	28,53	12
Raspbian OS	8000	9,13	6
Aplicação	8000	9,13	0,1
Switch	25000	28,53	6
Hardware Nó escritório	40000	45,66	48
Sistema operacional do nó de escritório	400	0,46	0,5
Aplicação do nó de escritório	1200	1,37	0,5
Módulo Esp32	22602	25,80	2
Sensor de Temperatura	13140	15	2
Sensor de gases	13140	15	2
Sensor de qualidade da água	13140	15	2
Sensor de Luz	13140	15	2

Fonte: Elaborada pelo autor com dados da literatura.

DHT22, um sensor de gases MQ-135 e sensor de luminosidade LM393 com sensibilidade ajustável, interligados em um único circuito. A infraestrutura foi interconectada por meio de um *switch* de rede Gigabit de 8 portas.

Figura 19 – Ambiente do experimento de injeção de falhas



Fonte: Elaborada pelo autor.

Definição da Carga de Trabalho: utilizou-se de sensores e uma câmera para gerar os dados que foram transmitidos na rede. Esses dados alimentaram aplicações que estavam executando tanto no no de processamento de vídeo como no nó de escritório. A carga de trabalho foi pensada para ser intensiva de modo a estressar os componentes do sistema. Assim definiu-se que a transmissão do vídeo ocorreria de forma ininterrupta e os dados dos sensores seriam transmitidos por amostras a cada 5 segundos.

Definição das Estratégias de Monitoramento: a estratégia de monitoramento adotada foi a de utilização de *scripts bash* para realizar a consulta periódica ao componentes, checando o *status* de cada um. Vale destacar que ao verificar a disponibilidade de um componente se deve observar não só a característica de o componente estar ativo ou não (por exemplo, ligado ou desligado), mas é importante estabelecer meios de consultar se o componente está realizando as tarefas para a qual foi designado. Por exemplo, na aplicação de processamento de vídeo não basta apenas verificar se o processo da aplicação existe e está em execução, mas também checar se os vídeos estão sendo processados. Isso é importante pois o processo pode entrar em estado de erro aparentando estar em um estado saudável.

Paralelamente a execução do experimento, um *script* bash verificou a cada 10 segundos se os componentes do sistema estavam funcionando corretamente, armazenando o estado do respectivo componente. O estado é um indicador considerado ao verificar se o componente está disponível (*up*) ou indisponível (*down*), que é salvo em um arquivo de texto. A função

desse *script*, conforme podemos ver nas épocas E2 e E4 da Figura 20 e na Figura 21, é observar os componentes que fazem parte do sistema e identificar variações na disponibilidade dos mesmos, capturando quando as falhas ocorrem e o tempo necessário para a realização do reparo.

Definição da Estratégia de Injeção de Falhas: para injetar as falhas no sistema, optou-se por ter um componente separado da infraestrutura básica do sistema realizando essa ação. Esse componente é um computador que executou um programa escrito na linguagem de programação Python para gerar os tempos de falha e realizar a simulação de defeitos nos componentes do sistema. O funcionamento básico dessa aplicação de injeção de falhas é descrito a seguir e mostrado pelo Algoritmo 1. Os tempos de falha foram gerados utilizando-se como base os tempos médio de falha e reparo passados como parâmetro para a aplicação. Como esses tempos são exponencialmente distribuídos, a biblioteca *numpy* foi utilizada para gerar os tempos utilizando os parâmetros MTTR e MTTF mostrados na Tabela 7 como semente a cada interação do experimento, i.e., a cada rodada de falha e reparo. Em alguns casos também foi utilizado *scripts bash* para auxiliar no processo de injeção da falha e na posterior restauração do componente.

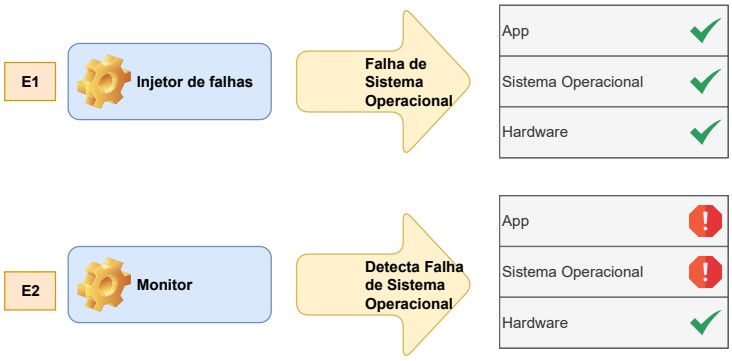
Devido à existência de dependência entre alguns componentes do sistema, a falha em um componente pode afetar o correto funcionamento de outros componentes que dependem dele. Por exemplo, a falha em no hardware de um servidor afetar o funcionamento do sistema operacional que dele depende. Bem como a falha do sistema operacional deve implicar na indisponibilidade a aplicação que dele depende para funcionar corretamente. Em outras palavras, a falha em um componente crítico pode propagar para outros componentes, causando impactos em cascata no sistema como um todo. Esse comportamento é mostrado na Figura 20.

Quando o injetor de falhas realiza a injeção em um componente, como por exemplo no sistema operacional, esse por sua vez faz com que a aplicação também falhe (época E1 da Figura 20). Por sua vez, aguardado o tempo de reparo, o sistema é restaurado e o injetor de falhas envia o comando para que o sistema seja restabelecido. Nessa etapa de restauração, há um mecanismo para que os componentes que têm dependência com o componente restabelecido também o sejam. A Figura 21 mostra esse processo.

Algorithm 1 Injetor de Falhas

```
while true do
    tempo_ate_falha = gerar_tempo_ate_falha(MTTF);
    tempo_de_reparo = gerar_tempo_de_reparo(MTTR);
    esperar(tempo_ate_falha);
    injetar_falha();
    esperar(tempo_de_reparo);
    restaurar_sistema();
end
```

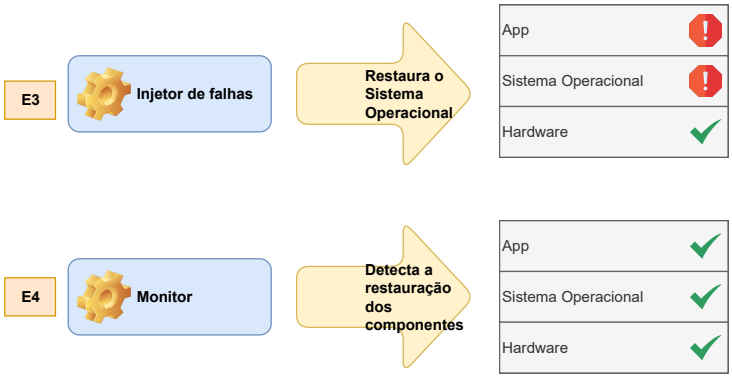
Figura 20 – Injeção e detecção da falha



Fonte: Elaborada pelo autor.

É importante destacar que esse comportamento de cascata também é implementado no injetor de falhas. Uma vez a falha no sistema operacional ocorre, por exemplo, o processo que estava responsável por contar o tempo para a falha da aplicação é encerrado. Quando o sistema operacional é reestabelecido, é criado um novo processo para contabilizar o tempo até a falha da aplicação.

Figura 21 – Restauração dos componentes em falha após período de reparo



Fonte: Elaborada pelo autor.

Execução do Experimento de Injeção de Falhas: a etapa de execução do experimento consiste em aplicar as cargas de trabalho ao sistema ao mesmo tempo em que o computador injetor de falhas é ativado para injetar as falhas no ambiente. Paralelamente a todas essas atividades um outro computador monitora todos os componentes do sistema, salvando *logs* em disco que foram utilizados mais tarde para a obtenção das métricas que serviram para a realização da análise estatística dos resultados. O experimento teve duração de 100 horas, e foram observadas um total de 501 falhas e reparos em todos os componentes do sistema. Esta informação será útil para calcular o intervalo de confiança na próxima etapa.

Tratamento Estatístico dos Dados: finalizado o experimento, com os dados que foram medidos pode-se calcular os tempos de falha de cada componente, calculando posteriormente o tempo médio de falha de cada um deles. Com os tempos que foram medidos, e com a informação de número de falhas e reparos observados, agora é possível calcular a disponibilidade observada do sistema, que é $0,99474$. Tem que se destacar que o valor de disponibilidade calculado é alto pois foi realizada a conversão dos valores obtidos no experimento pelo fator de redução. Com isso têm-se valores aproximados aos que se espera encontrar em cenários sem fator de redução. Também foi aplicado um método (KEESEE, 1965) para calcular a disponibilidade do intervalo de confiança de 95%. Com isso, é possível determinar se há evidências para rejeitar ou aceitar o modelo proposto com o intervalo de confiança. Foi obtido que a média está entre $0,99374 < X < 0,99558$.

Caso ao fim do tratamento estatístico não se obtivessem dados satisfatórios para a validação do modelo, se pode voltar a etapas anteriores da metodologia adotada no trabalho, de modo a ajustar o modelo, obter outros dados ou mesmo a alteração do ambiente de experimentação para corrigir distorções.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo mostrou a metodologia adotada no processo de desenvolvimento dessa dissertação. Ele mostra as etapas que foram seguidas, desde o entendimento do sistema até a proposição de melhorias para o sistema. Através dela é possível realizar a reprodução do que foi proposto ao longo do trabalho, além de ser um importante instrumento para a compreensão das contribuições encontradas neste documento.

Este capítulo apresentou os modelos de disponibilidade produzidos nesse trabalho, baseados em RBDs e em cadeias de Markov de tempo contínuo. Também foram mostradas as equações

fechadas extraídas desses modelos. Esses modelos são úteis para realizar experimentos de disponibilidade, tema do próximo capítulo.

5 ESTUDOS DE CASO

Nesta capítulo, são descritos os três estudos de casos que foram conduzidos. A primeira seção mostra o primeiro estudo de caso: a validação dos modelos. Para isso foi conduzido um experimento de injeção de falhas considerando o sistema baseado em computação em borda. A segunda seção descreve o segundo estudo de caso: disponibilidade em estado estacionário com base em um experimento na ferramenta de modelagem *Mercury* (Maciel et al., 2017) considerando diferentes estratégias de arquitetura. Por fim, o terceiro estudo de caso, descrito na terceira seção, é a realização de uma análise de sensibilidade para a verificação de componentes críticos para a disponibilidade do sistema.

5.1 VALIDAÇÃO DO MODELO BASEADO EM COMPUTAÇÃO NA BORDA

O objetivo é criar um modelo que represente um sistema real; uma vez criado, é necessário validar o modelo para garantir que ele possa ser utilizado sem que haja a necessidade de implantação e medição do sistema. Como linha de base para o estudo, é considerado um ambiente com um nó de borda executando o aplicativo de processamento de vídeo. Foi realizado um experimento de injeção de falhas para obter os dados necessários para tal validação. Portanto, no sistema analisado, todas as falhas ocorreram por meio de um experimento de injeção de falhas.

Ao calcular a disponibilidade com base no modelo de Cadeia de Markov proposto (Figura 9), foi obtida uma disponibilidade de $0,99557$, que está dentro do intervalo de confiança calculado na seção 4.3. Portanto, não há evidências para rejeitar o modelo como sendo um instrumento representativo do sistema.

5.2 ESTUDO DE DISPONIBILIDADE DE ESTADO ESTACIONÁRIO

Inicialmente, foi realizada a análise de confiabilidade de estado estacionário do modelo correspondente a Equação 4.8, utilizando a ferramenta Mercury e os tempos de cada componentes extraídos da literatura mostrados na seção 7. Os resultados desta análise podem ser encontrados na Tabela 8. As taxas de falha e reparo utilizadas foram as disponíveis em (DANTAS et al., 2016) para os números da nuvem. Foi considerado que o MTTF e o MTTR

para a nuvem são de 2880 e $0,03$ horas, respectivamente. Os demais MTTFs mostrados na Tabela 7, Névoa, borda e Escritório, foram calculados com a Equação 4.7. Com base nisso, foi possível determinar que o MTTF e o MTTR para o componente do nó de névoa são, respectivamente, de $2167,4226$ e $1,1659$ horas. Já o nó físico da borda tem um MTTF de $6060,6056$ horas e um MTTR de $7,4567$ horas. Por fim, o MTTF e o MTTR para o nó físico do escritório correspondem a $396,0396$ e $0,9709$ horas.

Tabela 8 – Resultados do modelo RBD do nó físico

Métrica	**Nuvem**	**Névoa**	**Borda**	**Escritório**
MTTF (h)	2880	2167,4226	6060,6056	396,0396
MTTR (h)	0,03	1,16591	7,45672	0,97089
Disponibilidade (%)	0,99999	0,9994624	0,9987711	0,9975
# 9's	4,98227	3,2695	2,9105	2,6116
Downtime (h)	0,09125	4,7128	10,7718	21,4368

Fonte: Elaborada pelo autor baseada em Dantas et al. (2016).

Aplicando os números apresentados na Tabela 7 ao modelo representado pela Equação **??**, obtêm-se os valores de MTTF e MTTR para o componente do módulo sensor. Os tempos de falha e reparo obtidos foram de $2868,14107$ e $2,00055$ horas, respectivamente. Esses resultados podem ser visualizados na Tabela 9. Agora, é possível definir os componentes do nó de processamento e do nó de escritório nas cadeias de Markov mostradas na Figura 13. Ao realizar uma análise de estado estacionário usando os modelos com os tempos apresentados nas Tabelas 7 e 8, foi possível construir as Tabelas 10 e 11, que mostram os resultados com redundância e sem redundância *hot-standby*.

Tabela 9 – Resultados para o modelo RBD do módulo de sensores

Métrica	**Módulo de Sensores**
MTTF (h)	2868,14107
MTTR (h)	2
Disponibilidade (%)	0,99930
# 9's	3,15675
Downtime (h)	6,10996

Fonte: Elaborada pelo autor.

A Tabela 10 mostra os resultados sem que se considere estratégias de redundância. O modelo foi configurado com um *switch*, uma câmera, um módulo de sensor, um dispositivo de

borda, névoa ou nuvem com um aplicativo de processamento e um nó de escritório. É possível ver na Tabela 10 que o tempo de inatividade da arquitetura baseada em névoa é menor do que a solução de borda. Um tempo de inatividade de $38,85$ horas para a arquitetura de névoa significa que o sistema ficaria inativo por aproximadamente 39 horas em um ano, com uma disponibilidade de $0,99556$, que contém $2,35$ noves. No entanto, se considerarmos um nó de ponta, a disponibilidade será de $0,99488$, com $2,29$ noves representando um tempo de inatividade anual de $44,77$ horas e um uptime de $8715,22$ horas. O tempo de inatividade da solução baseada em nuvem é o menor das três arquiteturas testadas, sendo de aproximadamente 34 horas por ano. Assim, foi obtida uma disponibilidade de $0,99610$, com noves de $2,40$.

Tabela 10 – Resultados sem redundância

Métrica	**Baseada na borda**	**Baseada na névoa**	**Baseada na nuvem**
Disponibilidade (%)	$0,99489$	$0,99556$	$0,99610$
Indisponibilidade (%)	$0,00511$	$0,00443$	$0,00389$
# 9's	$2,29148$	$2,35310$	$2,40922$
Uptime (h/y)	$8715,22661$	$8721,14905$	$8725,85858$
Downtime (h/y)	$44,77339$	$38,85095$	$34,14142$

Fonte: Elaborada pelo autor.

Quando se considera cenários com redundância no nó de processamento, ou seja, com dois nós de processamento em *hot-standby*, que podem ser vistos na Tabela 11, todas as arquiteturas - na névoa, na nuvem e na borda - apresentam resultados semelhantes. Os resultados com redundância podem ser vistos na Tabela 11. O tempo de inatividade do sistema com redundância para este componente é de aproximadamente 34 horas. A disponibilidade é de $0,996125$, o que significa um tempo de atividade de $8,726$ horas por ano.

Tabela 11 – Resultados com redundância *hot standby*

Componente redundante	**Disponibilidade**	**Indisponibilidade**	**#9**	***Uptime* (h/y)**	**Downtime (h/y)**
Nó físico na Edge	$0,99612$	$0,00388$	$2,41160$	$8726,04510$	$33,95489$
Nó físico na Fog	$0,99612$	$0,00387$	$2,41174$	$8726,05578$	$33,94421$
Nó físico na Cloud	$0,99612$	$0,00387$	$2,41177$	$8726,05854$	$33,94146$
Nó de escritório na Edge	$0,99773$	$0,00226$	$2,64498$	$8740,16069$	$19,83931$
Nó de escritório na Fog	$0,99841$	$0,00159$	$2,79949$	$8746,10008$	$13,89992$
Nó de escritório na Cloud	$0,99895$	$0,00105$	$2,97980$	$8750,82308$	$9,17692$

Fonte: Elaborada pelo autor.

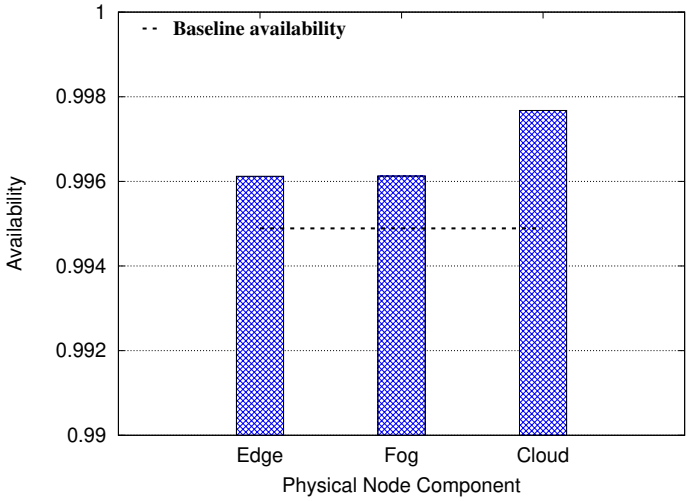
Pode-se observar uma melhoria nos resultados ao considerar a redundância do nó de escritório. Na arquitetura baseada na borda, a disponibilidade obtida foi de $0,99773$, correspondendo a uma disponibilidade de $2,64$ noves e um *uptime* de 8740 horas. Na solução na névoa, o resultado do *uptime* foi de 8746 horas, o que significa uma disponibilidade de $0,99841$ com $2,79949$ noves. Esses resultados representam um ganho de 6 horas na solução baseada na névoa em um ano. Na solução baseada em nuvem, a disponibilidade foi de $0,99895$, o que equivale a aproximadamente um tempo de *uptime* de 8751 horas e um tempo de inatividade de $9,17692$ horas em um ano. A solução baseada em nuvem apresentou um ganho de mais de 10 horas em relação à solução baseada na borda.

Embora a abordagem redundante possa trazer melhores resultados em termos de disponibilidade, é importante destacar que ela também implica em custos adicionais. Esses custos podem estar associados à aquisição de equipamentos e ao consumo de energia, refrigeração e manutenção contínuos, já que o computador sobressalente está sempre ligado. No entanto, dependendo da solução adotada, é possível reduzir esses custos adotando políticas que liberem os recursos quando não estão sendo utilizados. A computação em nuvem, por exemplo, permite o uso de planos baseados em pagamento por uso, o que pode ajudar a reduzir os custos em relação a outras soluções.

Em uma configuração em que existem três réplicas em nós físicos e nós de escritório, e em um cenário em que é possível tolerar a falha de até duas dessas réplicas, então, para que o sistema esteja operacional, considera-se que pelo menos um nó deve ser funcional (2oo3). A Figura 22 mostra essa situação por tipo de arquitetura. Pode-se observar que a solução em nuvem apresenta uma melhor disponibilidade, $0,99767$, com um tempo de inatividade de $20,35$ horas.

Uma estratégia menos dispendiosa é o *cold-standby*, que mantém o nó sobressalente desligado até ocorrer uma falha. Nessa situação, foi avaliada a arquitetura de *cold-standby* considerando o nó em névoa, que apresentou um tempo de inatividade de $39,435$ horas, disponibilidade de $0,99549$ e tempo de atividade de 8720 horas em um ano. Como pode ser visto, a diferença entre as estratégias de *hot-standby* e *cold-standby* resulta em aproximadamente 1 hora a menos de tempo de atividade anual. Portanto, embora não seja uma aplicação crítica, a solução *cold-standby* na névoa pode ser adotada sem perdas significativas em termos de disponibilidade. Essa solução reduz o tempo de inatividade anual em $4,3$ horas em comparação com uma arquitetura de névoa não redundante e em $12,6$ horas em comparação com a solução de borda não redundante.

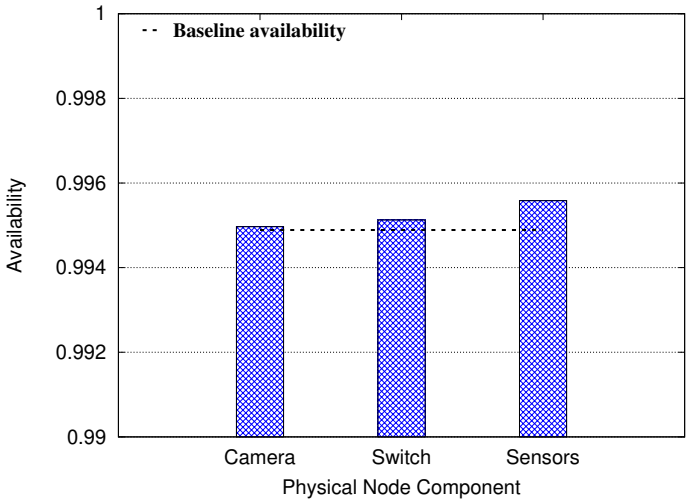
Figura 22 – Gráfico 2oo3 no componente nó físico



Fonte: Elaborada pelo autor.

Considerando uma arquitetura 2oo3 na qual cada componente da solução base (considerando o processamento em borda) é variado, a Figura 23 é apresentada. Nessa figura, a arquitetura 2oo3 é aplicada a cada componente simultaneamente. É possível observar como a disponibilidade do sistema é alterada, tomando como referência a linha de base. O módulo do sensor apresenta o melhor impacto na disponibilidade do sistema, aumentando a disponibilidade da linha de base para $0,99558$, o que representa um tempo de inatividade anual de $38,71$. Isto significa que é mais interessante, quando se quer maximizar a disponibilidade do sistema, adotar redundância no módulo de sensores do que os outros componentes analisados.

Figura 23 – 2oo3 por componente



Fonte: Elaborada pelo autor.

Foi considerado também um cenário em que todos os componentes do sistema possuam três redundâncias e possam tolerar duas falhas em suas réplicas. Nesse caso, a disponibilidade do sistema considerando a solução de ponta será de $0,99997$, com um tempo de inatividade de $0,21$ horas, representando aproximadamente 12 minutos de indisponibilidade em um ano.

A decisão sobre qual estratégia adotar para garantir a disponibilidade desejada deve considerar a disponibilidade que se deseja atingir e a viabilidade financeira de implementação da solução. Embora colocar o maior número possível de réplicas para cada componente seja uma solução viável para maximizar a disponibilidade, os custos envolvidos na operação podem ser superiores aos ganhos obtidos.

5.3 ANÁLISE DE SENSIBILIDADE

Realizou-se uma análise de sensibilidade para estudar como os parâmetros do modelo impactam a disponibilidade do sistema. Essa análise visa mostrar qual parâmetro influencia significativamente a disponibilidade do sistema quando ele atinge o estado estacionário. A Tabela 12 apresenta o *ranking* de sensibilidade calculado com base na derivada parcial da Equação 4.1. Aqui, os parâmetros do índice de sensibilidade são mostrados em ordem decrescente.

O topo da classificação inclui os parâmetros que influenciam diretamente a disponibilidade do sistema. Esses parâmetros estão relacionados ao número de nós de escritório, ao número de nós de processamento, ao número de aplicativos de escritório por nó e à taxa de falha e reparo do nó de escritório. Por exemplo, os parâmetros (N, M, Y, λ_{pno} e μ_{pno}) têm impacto no aumento da probabilidade de falha do componente e, consequentemente, do sistema como um todo. Por outro lado, as taxas de falha e reparo da câmera e dos aplicativos de processamento têm menos influência na disponibilidade do sistema.

Com base no *ranking* é possível estudar por meio de experimentos como pequenas alterações nos parâmetros do sistema irão afetar a disponibilidade geral. Esse estudo de variação paramétrica é interessante no processo de planejamento de disponibilidade, pois permite a uma tomada de decisão mais acurada quanto aos equipamentos que serão utilizados na implantação de um sistema desse tipo, suas quantidade para atender os níveis de disponibilidade almejados, bem como sobre a adesão a diferentes tipos de contratos de manutenção, visando atender diferentes acordos de níveis de serviço (*Service Level Agreement*s (SLAs)).

As Figuras 24, 25, 26, 27, 28 e 29 são usadas para verificar a sensibilidade da disponibilidade do sistema quando cada parâmetro é variado individualmente. Por exemplo, para variar N, M

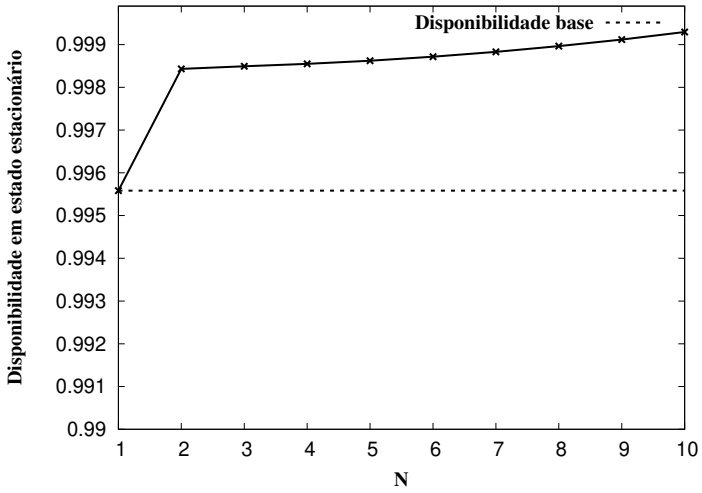
Tabela 12 – Classificação de sensibilidade baseada em derivadas parciais

Parâmetro	$\|SS(A)\|$
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}	$0,00041656$
λ_{apo}	$0,00041656$
λ_{sw}	$0,000239981$
μ_{sw}	$0,000239981$
L	$0,000141145$
λ_{cam}	$0,0000800064$
μ_{cam}	$0,0000800064$
λ_{app}	$0,0000125018$
μ_{app}	$0,0000125018$

Fonte: Elaborada pelo autor.

e Y, mudamos os números em etapas únicas (ou seja, 1, 2, 3, ...), já que eles estão no domínio inteiro. Esse tipo de gráfico é utilizado para verificar como a variação de um parâmetro impacta na disponibilidade do sistema. Podemos observar nas figuras que alguns parâmetros, como o número de nós físicos ativos (servidores) ou o número de *switches* ativos, causam uma pequena variação na disponibilidade após a segunda redundância. Assim, é possível avaliar se os custos necessários para adicionar a réplica de um novo componente valem a pena, uma vez que esses componentes melhoram ligeiramente a disponibilidade. Além disso, os parâmetros que mais afetam diretamente a disponibilidade do sistema são o número de nós de escritório, o número de nós de processamento, o número de aplicativos de escritório por nó e a taxa de falha e reparo do nó de escritório (N, M, Y, λ_{pno} e μ_{pno}). A taxa de falha e reparo da câmera e dos aplicativos de processamento, por outro lado, têm menos influência na disponibilidade do sistema.

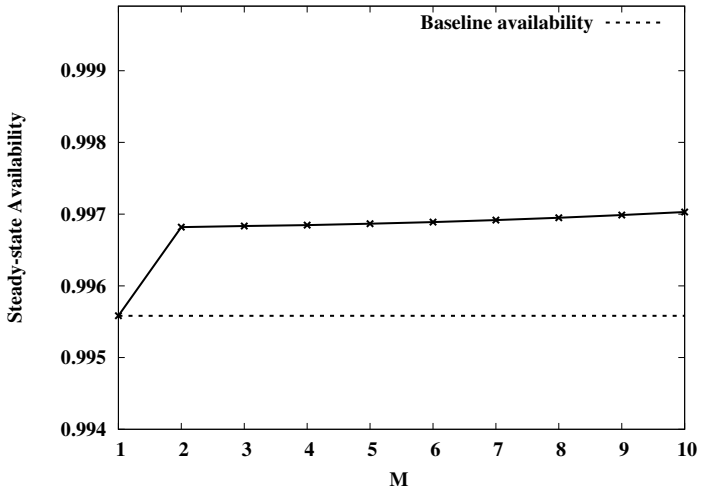
Figura 24 – Número de nós físicos de escritório ativos



Fonte: Elaborada pelo autor.

O número de nós físicos de escritórios ativos, exibido na Figura 24 mostra que ao adicionar uma instancia em *hot standby* na arquitetura melhora a disponibilidade do sistema em $0,3\%$. Contudo a curva da disponibilidade suaviza a medida que novos nós são adicionados, não havendo melhoras relevantes na disponibilidade após o segundo nó.

Figura 25 – Número de nós físicos do servidor ativos

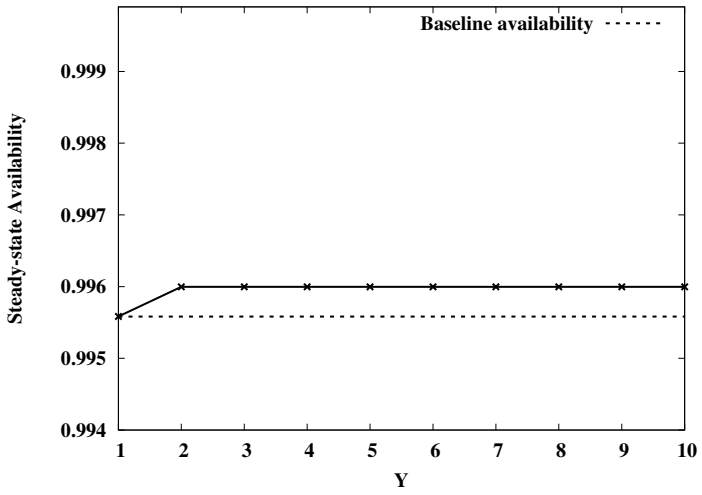


Fonte: Elaborada pelo autor.

Na Figura 25 é mostrado como o número de nós físicos de servidor ativos (representado pelo parâmetro M) interfere na disponibilidade geral. Vê-se que com o ingresso do segundo nó redundante no sistema a melhora de disponibilidade em $0,14\%$. A adição de novas redun-

dâncias causam pequenas variações na disponibilidade do sistema, porém são pequenas e não significativas.

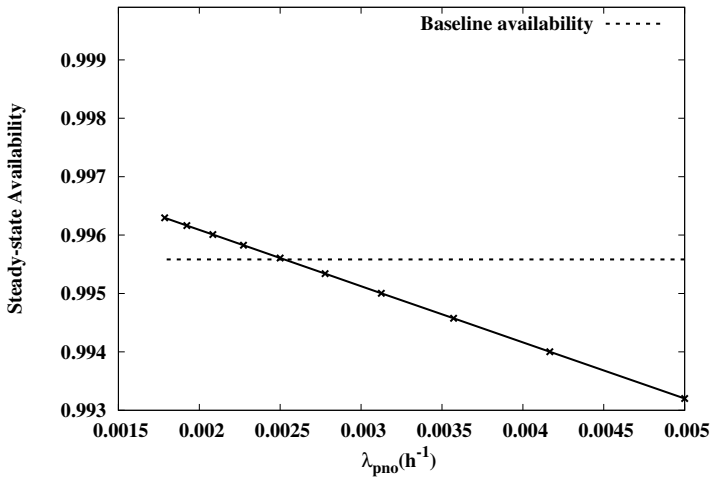
Figura 26 – Número de aplicativos em nós físicos do escritório



Fonte: Elaborada pelo autor.

Pela Figura 26 verifica-se que aumentar significativamente o número de aplicações executando no nó de escritório não tem correspondência com o aumento da disponibilidade. Após a adição da segunda aplicação, o sistema passa a ter uma disponibilidade de $0,9965$, o que representa uma melhora de $0,1\%$ na disponibilidade. Aplicações adicionais não melhoram a disponibilidade, contudo, pode ser que outros fatores sejam melhorados como a performance do sistema (MACIEL, 2023b).

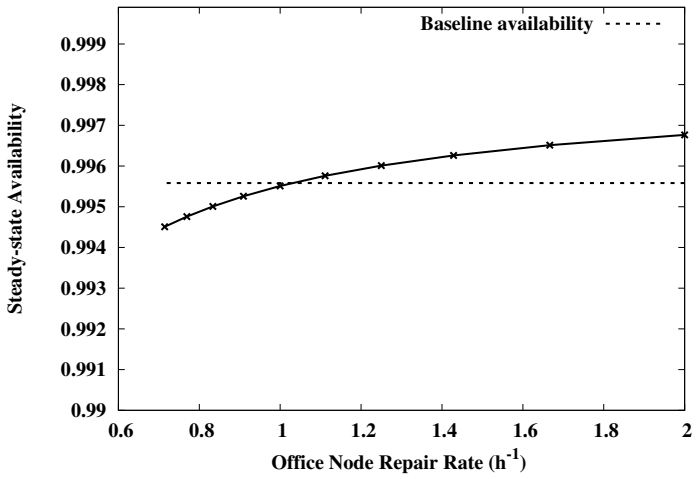
Figura 27 – Taxa de falha de nó do escritório



Fonte: Elaborada pelo autor.

Percebe-se na Figura 27 que caso a taxa de falha aumente em 50%, i.e., se o MTTF diminuir, a disponibilidade do sistema cai em $0,23\%$, mas que se o contrário ocorrer, ou seja, a taxa de falha melhorar em 50%, o ganho de disponibilidade é de apenas $0,09\%$. A taxa de falha está fortemente conectada com a qualidade do equipamento adquirido e a qualidade dos softwares que executam nesse componente.

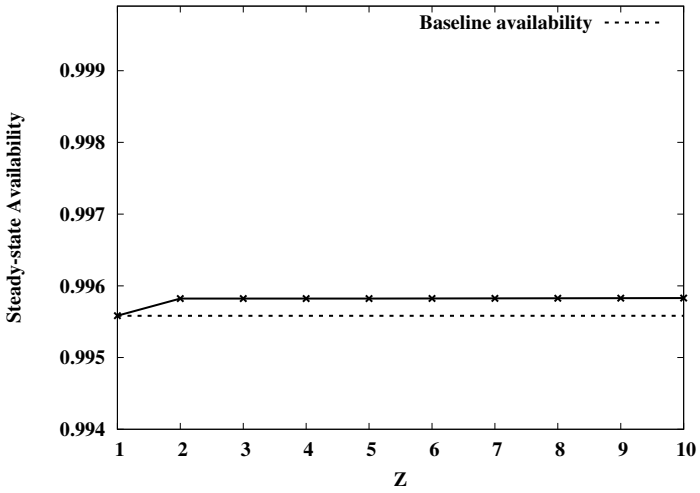
Figura 28 – Taxa de reparo de nós do escritório



Fonte: Elaborada pelo autor.

Já a taxa de reparo, uma vez variada 50% acima do valor considerado no experimento *baseline*, a melhora esperada é da ordem de $0,19\%$. É importante destacar que melhoras no tempo de reparo podem ser obtidas por meio de alguns fatores como, por exemplo, diminuindo o tempo para detecção da falha, permitindo que a equipe responsável pela manutenção do equipamento possa atuar de pronto. Outro fator que influencia nessa métrica é a quantidade de equipes de reparo disponíveis.

Figura 29 – Número de *switches* ativos



Fonte: Elaborada pelo autor.

Por outro lado, alterar o número de nós de escritório de um para dois e obter uma melhor taxa de reparo para os nós de escritório aumenta a disponibilidade geral. É possível decidir qual componente será redundante na infraestrutura ou qual será a melhor taxa de reparo levando em consideração os custos operacionais. Uma melhor taxa de reparo dependerá do contrato adotado e do número de equipes de reparo disponíveis. É fundamental destacar que os gráficos ajudam a encontrar o melhor ajuste para maximizar a disponibilidade da infraestrutura.

Por exemplo, se for pressuposto que o MTTF do nó do escritório é equivalente a 560 horas e que o MTTR corresponde a $0,5$ horas. Pode-se obter esse MTTF adquirindo melhores hardwares e sistemas operacionais. Além disso, pode-se aumentar o MTTR com mais equipes de reparo. Supõe-se que dois nós de escritório serão utilizados em *hot-standby* com base nos tempos mostrados na Tabela 8 e que a arquitetura adotada também será baseada em computação na borda, considerando que o MTTF e MTTR são, respectivamente, 8400 e $3,5$ horas. Considere também que se está utilizando uma câmera, um *switch* e um módulo sensor com os tempos descritos nas Tabelas 7 e 9. Com base nessa configuração, a disponibilidade será de $0,99855$, representando um tempo de inatividade anual de $12,67$ horas. É uma melhoria de aproximadamente 7 horas no uptime, reduzindo o tempo de inatividade em $36,12\%$.

5.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram mostrados os resultados dos estudos de casos realizados nessa pesquisa. Foi realizada a validação do modelo considerando uma arquitetura básica utilizando o

processamento na borda. Uma vez validado o modelo, se avançou para a etapa de estudo de estado estacionário dos modelos, considerando cenários com e sem redundância. Por fim, uma análise de sensibilidade foi realizada, mostrando componentes sensíveis do sistema e como a adoção de réplicas nesses componentes alteram a disponibilidade do sistema.

6 CONCLUSÕES

Este estudo teve como objetivo propor modelos de confiabilidade para realizar estudos de disponibilidade em um aviário inteligente com sensores para medir os indicadores ambientais, como temperatura, e a utilização de câmeras e sistema de classificação baseado em visão computacional para estimar o peso das aves. Foram considerados três sistemas arquitetônicos: baseados na nuvem, baseados na borda e baseados na névoa. Também foi considerada uma estratégia de redundância para medir o impacto na melhoria da disponibilidade, considerando os nós *hot-standby*. Como resultado, foram apresentados três estudos de caso: primeiro, uma validação de um modelo baseado na borda com um experimento de injetor de falhas; em segundo lugar, foi realizado um estudo de disponibilidade em estado estacionário; então, por fim, foi realizada uma análise de sensibilidade.

A análise de disponibilidade verificou que a solução baseada em névoa sem redundância tem uma pequena vantagem em comparação com a solução de borda. Quando considerado o nó de névoa, a disponibilidade é $0,99556$. Uma disponibilidade semelhante foi observada quando foi realizada a comparação entre às soluções de névoa e borda quando replicas em *hot-standby* foram adicionadas. No entanto, a solução baseada em névoa tem melhor capacidade de processamento e armazenamento.

A solução baseada em nuvem obteve melhores resultados quando analisada sem redundância. O cenário que permitia redundância e configuração 2oo3 obteve uma disponibilidade de $0,99767$, representando uma melhor disponibilidade quando comparado aos demais casos. Ao considerar a redundância no nó do escritório, a solução baseada em nuvem se saiu melhor, alcançando um tempo de inatividade de $9,18$ horas. No entanto, é importante ressaltar que, apesar de ser uma boa opção em termos de disponibilidade, a solução baseada em nuvem pode originar atrasos na rede, comprometendo o bom funcionamento da solução preferida.

Garantir a disponibilidade de um sistema é uma preocupação central para garantir a qualidade do serviço oferecido. A fim de alcançar um nível desejável de disponibilidade, é importante escolher a estratégia adequada levando em conta a viabilidade financeira da implementação da solução. Embora seja possível maximizar a disponibilidade aumentando o número de réplicas de cada componente, é importante lembrar que os custos envolvidos na operação podem superar os ganhos obtidos.

6.1 CONTRIBUIÇÕES

Abaixo são listadas algumas das contribuições apresentadas por esta dissertação:

- A proposição de arquitetura de galpões inteligentes considerando diferentes padrões arquitetônicos;
- A sugestão de modelos de disponibilidade de *Continuous-time Markov Chain* (CTMC) e *Reliability Block Diagrams* (RBDs) validados através de experimentos de injeção de falhas;
- Um estudo de disponibilidade considerando diferentes tipos de redundância e tolerância a falhas;
- Um estudo de análise de disponibilidade destacou os parâmetros essenciais de disponibilidade do sistema.

  Além das contribuições mencionadas, um artigo foi produzido e está em processo de revisão na revista *IEEE Transactions on Industrial Informatics* sob o título *"Dependability Evaluation of a Smart Poultry House: Addressing Availability Issues Through the Edge, Fog, and Cloud Computing"*.

6.2 DIFICULDADES E LIMITAÇÕES

Apesar de esta dissertação ter atingidos os objetivos que foram estabelecidos no início da pesquisa, há de se reconhecer algumas dificuldades enfrentadas podem afetar a validade dos resultados além de algumas limitações que foram identificadas no decorrer da execução dos trabalhos.

A primeira dificuldade enfrentada foi que devido ao período pandêmico que iniciou-se em março de 2020 houve a necessidade de realização de isolamento social, o que implicou em restrições de acesso à Universidade Federal de Pernambuco (UFPE). Essa restrição impediu o acesso aos laboratório de trabalho e consequentemente aos recursos nele presente. Como medida paliativa, um mini laboratório foi implementado na minha residência. Devido à limitações no espaço disponível e a impossibilidade de realizar os experimentos em um local controlado, não é possível afirmar que não houveram interferências externas que possam ter influenciado nos resultados obtidos.

Outro ponto que vale destacar foi a dificuldade de localizar trabalhos relacionados à avaliação de disponibilidade de sistemas aplicados à indústria agropecuária. Isso pode se dar por alguns fatores, que se pode especular: a pouca experiência em pesquisa; como a principal ferramenta de busca para obter material relacionado foi o buscador Google, pode ter ocorrido falhas no processo de definição das *strings* de busca o que limitou as respostas da ferramenta; ou ainda podemos considerar a possibilidade de ainda não haver tantos trabalhos que estudem dependabilidade nesse contexto.

Como limitação do trabalho aqui desenvolvido, destaco a não consideração, nas análises realizadas, dos custos de se adotar estratégias de redundância. O custo de se adotar determinada estratégia de redundância pode ser um fator determinante para a implantação desse tipo de sistema em ambientes de produção.

Uma restrição adicional diz respeito aos modelos e técnicas empregados nesta tese. De modo a simplificar o modelo uma série de decisões foram tomadas para reduzir a quantidade de estados do modelo CTMC. Essas decisões implicam em abstrações que podem levar a perdas de detalhes importantes para a disponibilidade do sistema que não foram possíveis de serem observadas previamente e que podem afetar os resultados obtidos. Outro ponto é que usuário necessita de conhecimentos básicos para a parametrização das variáveis do sistema. Seria interessante que o usuário tivesse a possibilidade de realizar essa parametrização por meio de um sistema.

6.3 TRABALHOS FUTUROS

Como trabalhos futuros pretende-se estender essa pesquisa realizado outros tipos de análises como por exemplo:

- **Análise de desempenho do sistema proposto**: a avaliação do desempenho pode ajudar a identificar gargalos e pontos fracos do sistema, permitindo que sejam feitas melhorias para aumentar a sua eficiência.Isso pode resultar em redução de custos, aumento da produtividade e melhoria da qualidade do serviço, o que reflete na experiência de uso dos técnicos usuários do sistema.

- **Estudo de análise de disponibilidade orientada a custos**: ao avaliar o custo-benefício de diferentes soluções, esse tipo de análise irá ajudar a garantir que os recursos financeiros sejam alocados de forma eficiente e eficaz para obter a disponibilidade dese-

jada. Além disso, a análise de disponibilidade orientada a custos também é importante para ajudar a avaliar a relação custo-benefício de uma solução de disponibilidade. Nem sempre uma solução mais cara é a melhor solução para maximizar a disponibilidade de um sistema, especialmente quando outras soluções mais baratas podem fornecer benefícios similares.

- **Criação de um *framework* para realização de estudos de dependabilidade em sistemas agroindustriais**: outro ponto que pode ser explorado em trabalhos futuros é a criação de um sistema que permita modelar e obter métricas de interesse de outros conceitos de dependabilidade não somente no setor de criação e manejo das aves, mas também em outros pontos da cadeia produtiva como no preparo de rações, manejo de medicações e suplementos, abate de aves e logística de transporte da produção.

REFERÊNCIAS

ABPA. *Mercados Arquivos*. 2023. Acessado em 18 de fevereiro de 2023. Disponível em: <https://abpa-br.org/mercados/>.

ANDRADE, E.; NOGUEIRA, B. Dependability evaluation of a disaster recovery solution for iot infrastructures. *The Journal of Supercomputing*, Springer, v. 76, n. 3, p. 1828–1849, 2020.

ARAUJO, E.; DANTAS, J.; MATOS, R.; PEREIRA, P.; MACIEL, P. Dependability evaluation of an iot system: A hierarchical modelling approach. In: IEEE. *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. [S.l.], 2019. p. 2121–2126.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, v. 1, n. 1, p. 11–33, 2004.

BASHIR, E.; LUŠTREK, M. The mercury environment: a modeling tool for performance and dependability evaluation. In: IOS PRESS. *Intelligent Environments 2021: Workshop Proceedings of the 17th International Conference on Intelligent Environments*. [S.l.], 2021. v. 29, p. 16.

BERNARDO, A. *Barão de Itararé: a vida trágica e o humor anárquico de um ícone do jornalismo*. 2021. <https://www.bbc.com/portuguese/brasil-59689669>. Acesso em: 16 de fev. de 2023.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. *Cloud computing: Principles and paradigms*. [S.l.]: John Wiley & Sons, 2010. v. 87.

BUYYA, R.; DASTJERDI, A. V. *Internet of Things: Principles and paradigms*. [S.l.]: Elsevier, 2016.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to Discrete Event Systems*. Secaucus: Springer-Verlag New York, Inc., 2006. ISBN 0387333320.

COSTA, I.; ARAUJO, J.; DANTAS, J.; CAMPOS, E.; SILVA, F. A.; MACIEL, P. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, v. 32, n. 7, p. 2191–2205, 2016. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.1927>.

DANTAS, J.; MATOS, R.; ARAUJO, J.; OLIVEIRA, D.; OLIVEIRA, A.; MACIEL, P. Hierarchical model and sensitivity analysis for a cloud-based vod streaming service. In: IEEE. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. [S.l.], 2016. p. 10–16.

DASTJERDI, A. V.; GUPTA, H.; CALHEIROS, R. N.; GHOSH, S. K.; BUYYA, R. Fog computing: Principles, architectures, and applications. In: *Internet of things*. [S.l.]: Elsevier, 2016. p. 61–75.

DOUKAS, C.; MAGLOGIANNIS, I. Bringing iot and cloud computing towards pervasive healthcare. In: IEEE. *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. [S.l.], 2012. p. 922–926.

DOYLE, I.; LEESON, S. Automatic weighing of poultry reared on a litter floor. *Canadian Journal of Animal Science*, NRC Research Press Ottawa, Canada, v. 69, n. 4, p. 1075–1081, 1989.

ELIJAH, O.; RAHMAN, T. A.; ORIKUMHI, I.; LEOW, C. Y.; HINDIA, M. N. An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of Things Journal*, IEEE, v. 5, n. 5, p. 3758–3773, 2018.

EMBRAPA. *Desempenho zootécnico do frango de corte*. 2023. <https://www.embrapa.br/agencia-de-informacao-tecnologica/criacoes/frango-de-corte/producao/gestao-unidade-produtora/desempenho-zootecnico>. Acessado em 19 de fevereiro de 2023.

FERNANDES, S.; TAVARES, E.; SANTOS, M.; LIRA, V.; MACIEL, P. Dependability assessment of virtualized networks. In: *2012 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2012. p. 2711–2716.

FRANK, P. M.; ESLAMI, M. Introduction to system sensitivity theory. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 10, n. 6, p. 337–338, 1980.

FREITAS, L. A. R. d.; BERTOGLIO, O. A evolução da avicultura de corte brasileira após 1980. *Economia e Desenvolvimento*, n. 13, jun. 2001. Disponível em: <https://periodicos.ufsm.br/eed/article/view/3426>.

GARCIA, D. A.; GOMES, D. E. A avicultura brasileira e os avanços nutricionais. *Revista Científica*, v. 1, n. 1, 2019.

HAMBY, D. M. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental monitoring and assessment*, Springer, v. 32, n. 2, p. 135–154, 1994.

IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, p. 1–84, 1990.

JAIN, M.; GUPTA, R. Redundancy issues in software and hardware systems: an overview. *International Journal of Reliability, Quality and Safety Engineering*, World Scientific, v. 18, n. 01, p. 61–98, 2011.

JUNIOR, C. G. d. S. T. Consumo energético e ambiência de galpões avícolas fechados e potencial de implementação de sistema híbrido de acondicionamento térmico. Universidade Federal de Viçosa, 2019.

KALOXYLOS, A.; EIGENMANN, R.; TEYE, F.; POLITOPOULOU, Z.; WOLFERT, S.; SHRANK, C.; DILLINGER, M.; LAMPROPOULOU, I.; ANTONIOU, E.; PESONEN, L. et al. Farm management systems and the future internet era. *Computers and electronics in agriculture*, Elsevier, v. 89, p. 130–144, 2012.

KASAREDDY, R.; MUKHOPADHYAY, A. Fpms: A fog based poultry monitoring system. In: *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*. [S.l.: s.n.], 2022. p. 1–6.

KEESEE, W. *A method of determining a confidence interval for availability*. [S.l.], 1965.

KHARCHENKO, V.; KOLISNYK, M.; PISKACHOVA, I.; BARDIS, N. Reliability and security issues for iot-based smart business center: architecture and markov model. In: IEEE. *2016 Third International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*. [S.l.], 2016. p. 313–318.

LEE, H.; MOON, A.; MOON, K.; LEE, Y. Disease and pest prediction iot system in orchard: A preliminary study. In: IEEE. *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. [S.l.], 2017. p. 525–527.

LIU, F.; TANG, G.; LI, Y.; CAI, Z.; ZHANG, X.; ZHOU, T. A survey on edge computing systems and tools. *Proceedings of the IEEE*, IEEE, v. 107, n. 8, p. 1537–1562, 2019.

LUFYAGILA, B.; MACHUVE, D.; CLEMEN, T. Iot-powered system for environmental conditions monitoring in poultry house: A case of tanzania. *African Journal of Science, Technology, Innovation and Development*, Taylor & Francis, v. 14, n. 4, p. 1020–1031, 2022.

MACIEL, P.; MATOS, R.; SILVA, B.; FIGUEIREDO, J.; OLIVEIRA, D.; Fé, I.; MACIEL, R.; DANTAS, J. Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. [S.l.: s.n.], 2017. p. 50–57.

Maciel, P.; Matos, R.; Silva, B.; Figueiredo, J.; Oliveira, D.; Fé, I.; Maciel, R.; Dantas, J. Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. [S.l.: s.n.], 2017. p. 50–57.

MACIEL, P. R.; TRIVEDI, K. S.; MATIAS, R.; KIM, D. S. Dependability modeling. In: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. [S.l.]: IGI Global, 2012. p. 53–97.

MACIEL, P. R. d. M. *Performance, Reliability, and Availability Evaluation of Computational Systems, Volume 2: Reliability, Availability Modeling, Measuring, and Data Analysis*. 1st. ed. [S.l.]: Chapman and Hall/CRC, 2023.

MACIEL, P. R. d. M. *Performance, Reliability, and Availability Evaluation of Computational Systems, Volume I: Performance and Background*. 1st. ed. [S.l.]: Chapman and Hall/CRC, 2023.

MANSOR, H.; AZLIN, A. N.; GUNAWAN, T. S.; KAMAL, M. M.; HASHIM, A. Z. Development of smart chicken poultry farm. *Indonesian Journal of Electrical Engineering and Computer Science*, Institute of Advanced Engineering and Science, v. 10, n. 2, p. 498–505, 2018.

MATOS, R.; ARAUJO, J.; OLIVEIRA, D.; MACIEL, P.; TRIVEDI, K. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, Elsevier, v. 50, p. 151–164, 2015.

MIORANDI, D.; SICARI, S.; PELLEGRINI, F. D.; CHLAMTAC, I. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, Elsevier, v. 10, n. 7, p. 1497–1516, 2012.

MOLLAH, M. B. R.; HASAN, M. A.; SALAM, M. A.; ALI, M. A. Digital image analysis to estimate the live weight of broiler. *Computers and Electronics in Agriculture*, Elsevier, v. 72, n. 1, p. 48–52, 2010.

MORTENSEN, A. K.; LISOUSKI, P.; AHRENDT, P. Weight prediction of broiler chickens using 3d computer vision. *Computers and Electronics in Agriculture*, Elsevier, v. 123, p. 319–326, 2016.

NAHA, R. K.; GARG, S.; GEORGAKOPOULOS, D.; JAYARAMAN, P. P.; GAO, L.; XIANG, Y.; RANJAN, R. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, IEEE, v. 6, p. 47980–48009, 2018.

ÖHMANN, D.; SIMSEK, M.; FETTWEIS, G. P. Achieving high availability in wireless networks by an optimal number of rayleigh-fading links. In: IEEE. *2014 IEEE Globecom Workshops (GC Wkshps)*. [S.l.], 2014. p. 1402–1407.

OLIVEIRA, A. A. P.; FILHO, A. N.; EVANGELISTA, F. R. *A avicultura industrial no Nordeste: aspectos econômicos e organizacionais*. [S.l.]: Banco do Nordeste do Brasil, 2008.

PARK, E.; CHO, Y.; HAN, J.; KWON, S. J. Comprehensive approaches to user acceptance of internet of things in a smart home environment. *IEEE Internet of Things Journal*, IEEE, v. 4, n. 6, p. 2342–2350, 2017.

PEREIRA, P.; ARAUJO, J.; MELO, C.; SANTOS, V.; MACIEL, P. Analytical models for availability evaluation of edge and fog computing nodes. *The Journal of Supercomputing*, Springer, v. 77, n. 9, p. 9905–9933, 2021.

PEREIRA, P.; ARAUJO, J.; MELO, C.; SANTOS, V.; MACIEL, P. Analytical models for availability evaluation of edge and fog computing nodes. *The Journal of Supercomputing*, Springer, p. 1–29, 2021.

PSD. *Foreign Agricultural Service: Market and Trade Data*. United States Department of Agriculture, 2023. Acessado em 18 de fevereiro de 2023. Disponível em: <https://apps.fas.usda.gov/psdonline/app/index.html#/app/advQuery>.

REGATTIERI, A.; GAMBERI, M.; MANZINI, R. Traceability of food products: General framework and experimental evidence. *Journal of food engineering*, Elsevier, v. 81, n. 2, p. 347–356, 2007.

RODRIGUES, L.; GONÇALVES, I.; FÉ, I.; ENDO, P. T.; SILVA, F. A. Performance and availability evaluation of an smart hospital architecture. *Computing*, Springer, v. 103, p. 2401–2435, 2021.

SAHNER, R. A.; TRIVEDI, K.; PULIAFITO, A. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. [S.l.]: Springer Science & Business Media, 2012.

SATYANARAYANAN, M. The emergence of edge computing. *Computer*, IEEE, v. 50, n. 1, p. 30–39, 2017.

SICARI, S.; RIZZARDI, A.; COEN-PORISINI, A. Smart transport and logistics: A node-red implementation. *Internet Technology Letters*, Wiley Online Library, v. 2, n. 2, p. e88, 2019.

SILVA, B.; MATOS, R.; CALLOU, G.; FIGUEIREDO, J.; OLIVEIRA, D.; FERREIRA, J.; DANTAS, J.; LOBO, A.; ALVES, V.; MACIEL, P. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference, DSN*. [S.l.: s.n.], 2015.

SINGH, M.; KUMAR, R.; TANDON, D.; SOOD, P.; SHARMA, M. Artificial intelligence and iot based monitoring of poultry health: A review. In: *2020 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*. [S.l.: s.n.], 2020. p. 50–54.

SMITH, D. J. *Reliability, maintainability and risk: practical methods for engineers*. [S.l.]: Butterworth-Heinemann, 2021.

STRIELKINA, A.; KHARCHENKO, V.; UZUN, D. Availability models of the healthcare internet of things system taking into account countermeasures selection. In: SPRINGER. *International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications*. [S.l.], 2018. p. 220–242.

TANG, S.; XIE, Y. Availability modeling and performance improving of a healthcare internet of things (iot) system. *IoT*, Multidisciplinary Digital Publishing Institute, v. 2, n. 2, p. 310–325, 2021.

TURNER, M.; GURNEY, P.; BELYAVIN, C. Automatic weighing of layer-replacement pullets housed on litter or in cages. *British Poultry Science*, Taylor & Francis, v. 24, n. 1, p. 33–45, 1983.

VENKATESH, D.; TATTI, M.; HARDIKAR, P.; AHMED, S.; SHARAVANA, K. Cold storage management system for farmers based on iot. *Int J Recent Trends Eng Res. ISSN*, p. 2455–1457, 2017.

VOGADO, G. M. S.; VOGADO, K. T. S.; FONSECA, W. J. L.; FONSECA, W. L.; OLIVEIRA, A. M.; VOGADO, W. F.; LUZ, C. S. M. Evolução da avicultura brasileira. *Nucleus Animalium*, Fundação Educacional Ituverava, v. 8, n. 1, p. 49–58, 2016.

YU, W.; LIANG, F.; HE, X.; HATCHER, W. G.; LU, C.; LIN, J.; YANG, X. A survey on the edge computing for the internet of things. *IEEE access*, IEEE, v. 6, p. 6900–6919, 2017.

APÊNDICE A – *SCRIPTS* DE MONITORAMENTO

A.1 MONITOR BÁSICO

```
#!/bin/bash


data_time=''
current_date=''
current_time=''



# get current date and time AAAA-MM-DD HH:MM:SS
function get_date_time()
{
    date_time=`date --rfc-3339=seconds`
}

function get_current_date()
{
    current_date=`echo $date_time | cut -d\  -f1`
}

function get_current_time()
{
    current_time=`echo $date_time | cut -d\  -f2 | awk 'BEGIN{FS
        ="-"}{print $1}'`
}
function update_date_time()
{
    get_date_time
    get_current_date
    get_current_time
}

```

```
while [ True ]
do
update_date_time
stats=$(sshpass -p agnes ssh -f -o StrictHostKeyChecking=no
    192.168.0.200 -l felipe "/home/felipe/./check.sh")
echo "$current_date;$current_time;$stats" >> local_response1.txt
sleep 10
done
```

APÊNDICE B – *SCRIPTS* DE GERAÇÃO DE GRÁFICOS NO GNUPLOT

```
reset
set terminal postscript eps solid color enhanced font "Times-Bold"
    20
set output "result_s_param_M.eps"
set xlabel "{/Times:Bold Number of active physical nodes (servers)}
    "
set ylabel "{/Times:Bold Steady-state Availability}"
# set style line 10 pointtype 5
set yrange [0.9900:0.9999]
# set xrange [0:400]
plot "data_M.txt" using 1:2 with linespoints title "" linestyle 2
    lw 4 lc rgb "black", \
     "data_M.txt" u 1:3 title "{/Times:Bold Baseline availability}"
        with lines dt 3 lw 4 lc rgb "black"
```

APÊNDICE C – *SCRIPTS* DE ANÁLISE DE SENSIBILIDADE

C.1 EXPERIMENTO DE VARIAÇÃO PARAMÉTRICA

```
#!/usr/bin/python3
import math


lambda_pn = 1/6060.60564693312
mu_pn = 1/7.45672676375987

lambda_app = 1/8000.0
mu_app = 1/0.1

lambda_pno = 1/396.039603960393
mu_pno = 1/0.970891089108925

lambda_apo = 1/1200.0
mu_apo = 1/0.5

lambda_sw = 1/25000.0
mu_sw = 1/6.0

lambda_cam = 1/25000.0
mu_cam = 1/2.0


M = 1
N = 1
L = 1
Y = 1
Z = 1
W = 1



```

```
def combinatorial(a, b):
    return (math.factorial(a)/math.factorial(b)*math.factorial(a-b)
        )

def availability_pn(l_pn, m_pn, l_app, m_app, M, L):
    return availability_pno(l_pn, m_pn, l_app, m_app, M, L)

def availability_pno(l_pno, m_pno, l_apo, m_apo, N, Y):
    i = 1
    sum = 0
    while (i <= N):

        sum += (1-(math.pow(l_apo, i*Y)/math.pow(l_apo + m_apo, i*Y
            ))) * combinatorial(N, i) * math.pow(l_pno, N-i) * math.
            pow(m_pno, i)/math.pow(l_pno + m_pno, N)

        i = i+1
    return sum
def availability_sw(l_sw, m_sw, Z):
    return availability(l_sw, m_sw, Z)

def availability_cam(l_cam, m_cam, W):
    return availability(l_cam, m_cam, W)

def availability(l, m, W):
    # print(l, m, W)
    i = 1
    sum = 0
    while (i <= W):

        sum += combinatorial(W, i) * math.pow(l, W-i) * math.pow(m,
            i)/math.pow(l+m, W)

        i = i+1
```

```
    return sum

def incremental_list(min, max, increment):
    a = min
    l = []
    while (a < max):
        l.append(a)
        a += increment
    return l

def experiment_metric_N():
    N_list = incremental_list(1, 11, 1)
    print("N A")
    for n in N_list:
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, n,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%d %f" % (n, a_sys))

def experiment_metric_M():
    N_list = incremental_list(1, 21, 2)
    print("M A")
    for m in N_list:

        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , m,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
```

```
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%d %f" % (m, a_sys))


def experiment_metric_Y():
    Y_list = incremental_list(1, 11, 1)
    print("Y A")
    for y in Y_list:

        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%d %f" % (y, a_sys))

def experiment_metric_Z():
    Z_list = incremental_list(1, 11, 1)
    print("Z A")
    for z in Z_list:

        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

```

```
            print("%d %f" % (z, a_sys))

def experiment_metric_lambda_pno(min_mttf, max_mttf, samples):
    increment = (max_mttf - min_mttf)/float(samples)
    pno_list = incremental_list(min_mttf, max_mttf, increment)
    print("lambda_pno A")
    for mttf in pno_list:
        lambda_pno = 1.0/mttf
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttf, a_sys))

def experiment_metric_mu_pno(min_mttr, max_mttr, samples):
    increment = (max_mttr - min_mttr)/float(samples)
    pno_list = incremental_list(min_mttr, max_mttr, increment)
    print("mu_pno A")
    for mttr in pno_list:
        mu_pno = 1.0/mttr
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttr, a_sys))

```

```
def experiment_metric_mu_pn(min_mttr, max_mttr, samples):
    increment = (max_mttr - min_mttr)/float(samples)
    pn_list = incremental_list(min_mttr, max_mttr, increment)
    print("mu_pn A")
    for mttr in pn_list:
        mu_pn = 1.0/mttr
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttr, a_sys))

def experiment_metric_lambda_pn(min_mttf, max_mttf, samples):
    increment = (max_mttf - min_mttf)/float(samples)
    pn_list = incremental_list(min_mttf, max_mttf, increment)
    print("lambda_pn A")
    for mttf in pn_list:
        lambda_pn = 1.0/mttf
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttf, a_sys))

def experiment_metric_W():
    W_list = incremental_list(1, 11, 1)
```

```
    print("W A")
182 for w in W_list:

184     a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
186     a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, w)
188     a_sys = a_pno*a_pn*a_sw*a_cam

190     print("%d %f" % (w, a_sys))

192 def experiment_metric_mu_apo(min_mttr, max_mttr, samples):
    increment = (max_mttr - min_mttr)/float(samples)
194 pn_list = incremental_list(min_mttr, max_mttr, increment)
    print("mu_apo A")
196 for mttr in pn_list:
        mu_apo = 1.0/mttr
198     a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
200     a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
202     a_sys = a_pno*a_pn*a_sw*a_cam

204     print("%f %f" % (mttr, a_sys))

206 def experiment_metric_lambda_apo(min_mttf, max_mttf, samples):
    increment = (max_mttf - min_mttf)/float(samples)
208 pn_list = incremental_list(min_mttf, max_mttf, increment)
    print("lambda_apo A")
210 for mttf in pn_list:
```

```
        lambda_apo = 1.0/mttf
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttf, a_sys))


def experiment_metric_lambda_sw(min_mttf, max_mttf, samples):
    increment = (max_mttf - min_mttf)/float(samples)
    pn_list = incremental_list(min_mttf, max_mttf, increment)
    print("lambda_sw A")
    for mttf in pn_list:
        lambda_sw = 1.0/mttf
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M,L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N,Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttf, a_sys))

def experiment_metric_mu_sw(min_mttr, max_mttr, samples):
    increment = (max_mttr - min_mttr)/float(samples)
    pn_list = incremental_list(min_mttr, max_mttr, increment)
    print("mu_sw A")
    for mttr in pn_list:
        mu_sw = 1.0/mttr
```

```
        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M, L)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N, Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%f %f" % (mttr, a_sys))

def experiment_metric_L():
    W_list = incremental_list(1, 11, 1)
    print("L A")
    for l in W_list:

        a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app
            , M, l)
        a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo,
            mu_apo, N, Y)
        a_sw = availability_sw(lambda_sw, mu_sw, Z)
        a_cam = availability_cam(lambda_cam, mu_cam, W)
        a_sys = a_pno*a_pn*a_sw*a_cam

        print("%d %f" % (l, a_sys))

# experiment_metric_Y()
# experiment_metric_Z()
# experiment_metric_lambda_pno(200, 600, 10)
# experiment_metric_mu_pno(0.1, 2, 10)
# experiment_metric_mu_pn(3, 11, 10)

# experiment_metric_lambda_pn(4000, 8000, 10)
# experiment_metric_W()
# experiment_metric_mu_apo(0.1, 1, 10)
```

```
# experiment_metric_lambda_apo(600, 1800, 10)
# experiment_metric_lambda_sw(15000, 30000, 10)
# experiment_metric_mu_sw(3, 9, 10)
experiment_metric_L()


a_pn = availability_pn(lambda_pn, mu_pn, lambda_app, mu_app, M,L)
a_pno = availability_pno(lambda_pno, mu_pno, lambda_apo, mu_apo, N,
    Y)
a_sw = availability_sw(lambda_sw, mu_sw, Z)
a_cam = availability_cam(lambda_cam, mu_cam, W)
a_sys = a_pno*a_pn*a_sw*a_cam
print("Availability processing node: ", a_pn)
print("Availability office node: ", a_pno)
print("Availability switch: ", a_sw )
print("Availability cam: ", a_cam)
print("########################################")
print("Availability sys: ", a_pn*a_pno*a_sw*a_cam)
```