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

GERVÁSIO EUFRAUZINO TEIXEIRA

AVALIAÇÃO DE DESEMPENHO E DISPONIBILIDADE DE UM AMBIENTE DE GESTÃO DE APRENDIZAGEM A PARTIR DE MODELOS COMBINATORIAIS E DE ESPAÇO DE ESTADO

Recife

2025

GERVÁSIO EUFRAUZINO TEIXEIRA

AVALIAÇÃO DE DESEMPENHO E DISPONIBILIDADE DE UM AMBIENTE DE GESTÃO DE APRENDIZAGEM A PARTIR DE MODELOS COMBINATORIAIS E DE ESPAÇO DE ESTADO

Dissertação 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

Orientador (a): Prof. Dr. Jamilson Ramalho Dantas

Recife

2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Teixeira, Gervásio Eufrauzino.
Avaliação de desempenho e disponibilidade de um ambiente de Gestão de Aprendizagem a partir de modelos combinatórios e de espaço de estado / Gervásio Eufrauzino Teixeira. - Recife, 2025.
133f.: il.

Disserttação (Mestrado)- Universidade Federal do Pernambuco, Centro de Informática - CIn, Programa de Pós-Graduação em Ciência da Computação, 2025.
Orientação: Jamilson Ramalho Dantas.

1. Disponibilidade; 2. RBD; 3. CTMC; 4. SPN; 5. LAMP; 6. Moodle. I. Dantas, Jamilson Ramalho. II. Título.

UFPE-Biblioteca Central

Gervasio Eufrauzino Teixeira

“Avaliação de desempenho e disponibilidade de um ambiente de gestão de aprendizagem a partir de modelos combinatórios e de espaço de estado”

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: 29/07/2025.

BANCA EXAMINADORA

_____

Prof. Dr. Andson Marreiros Balieiro
Centro de Informática / UFPE

_____

Prof. Dr. Gustavo Rau de Almeida Callou
Departamento de Computação /UFRPE

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática / UFPE
(orientador)

A todos que lutam, trabalham e estudam... dedico.

AGRADECIMENTOS

A Deus, que no seu mistério da unidade se mostra trino e na sua trindade nos aparece uno.

A minha esposa Diana e minha filha Valentina que sempre me apoiaram e entenderam minhas ausências em suas vidas nos momentos que precisei.

A minha cunhada Dânia pelas energias enviadas.

Aos amigos da 2ª Vara Mista de Cajazeiras Lucivaldo, Chislene e Corrinha pelo apoio no trabalho.

Ao Prof. Paulo Maciel pela generosidade em compartilhar seus conhecimentos.

Ao meu orientador, Prof. Jamilson Dantas, pela paciência e disponibilidade. Agradeço de coração.

Aos colegas do grupo MoDCS, em especial Erick Barros, pela força, pela ajuda e pelas palavras de encorajamento.

Aos colegas da Faculdade Católica da Paraíba, em especial ao NTI – Núcleo de Tecnologia da Informação – e a Coordenação do Curso de Ciência da Computação na pessoa do Coordenador Prof. Renê Gadelha.

"Não se preocupe com suas falhas no que tentou fazer,

mas no que ainda é possível realizar."

– São João Paulo II

RESUMO

A crescente dependência de plataformas de *e-learning*, intensificada por eventos globais (a exemplo da Covid-19) que as transformaram em infraestruturas educacionais de alta criticidade operacional, exige arquiteturas resilientes, com elevada disponibilidade e desempenho compatível com as demandas institucionais. O Moodle, amplamente adotado como solução de código aberto, carece de análises quantitativas que revelem vulnerabilidades em suas arquiteturas de implantação e permitam propor melhorias fundamentadas em evidências analíticas. Este trabalho busca preencher essa lacuna ao propor e validar um *framework* híbrido e hierárquico de modelagem que combina modelos combinatórios e de espaço de estado para avaliar, de forma integrada, a disponibilidade e o desempenho do ambiente Moodle sobre a pilha LAMP (Linux, Apache, MySQL, PHP). A metodologia desenvolvida integra Diagramas de Blocos de Confiabilidade (RBD) para representar dependências estruturais, Cadeias de Markov de Tempo Contínuo (CTMC) para capturar a dinâmica de falhas e reparos, e Redes de Petri Estocásticas (SPN) para modelar a concorrência e as políticas de escalonamento automático em ambientes de nuvem. A validação dos modelos, conduzida por meio de injeção controlada de falhas em componentes críticos e análise estatística com intervalo de confiança de 95%, demonstrou que: (i) a arquitetura básica em hardware físico apresenta disponibilidade de 99,75%, correspondendo a 21,89 horas de inatividade anual; (ii) a introdução de redundância física eleva a disponibilidade para 99,9994%, reduzindo o tempo de inatividade em 99,75%; (iii) a virtualização, embora reduza a disponibilidade isoladamente (38,92 horas de inatividade), quando associada à redundância no nível do *host*, atinge 99,86% de disponibilidade, representando uma redução de 44,82% no tempo de inatividade em relação ao cenário base; e (iv) em nuvem pública, os modelos SPN revelam que políticas de escalonamento reativo podem levar à saturação e à degradação da vazão sob carga elevada. Conclui-se que arquiteturas virtualizadas com redundância e dimensionamento elástico em nuvem são estratégias eficazes para assegurar alta disponibilidade ($>$99,8%) e desempenho consistente. As principais contribuições deste trabalho incluem um *framework* analítico validado para planejamento de capacidade, diretrizes quantitativas para aprimoramento arquitetural e uma metodologia experimental reprodutível para avaliação de plataformas de aprendizagem.

Palavras-chaves: Disponibilidade. RBD. CTMC. SPN. LAMP. Moodle.

ABSTRACT

The growing dependence on e-learning platforms, intensified by global events (such as COVID-19), which have become highly operationally critical educational infrastructures, requires resilient architectures with high availability and performance compatible with institutional demands. Moodle, widely adopted as an open-source solution, supports quantitative analyses that reveal vulnerabilities in its deployment architectures and enable the proposal of improvements based on analytical evidence. This work seeks to fill this gap by proposing and validating a hybrid, hierarchical modeling framework that combines combinatorial and state-space models to comprehensively assess the availability and performance of the Moodle environment on the LAMP stack (Linux, Apache, MySQL, PHP). The developed methodology integrates Reliability Block Diagrams (RBDs) to represent structural dependencies, Continuous-Time Markov Chains (CTMCs) to capture failure and repair dynamics, and Stochastic Petri Nets (SPNs) to model concurrency and autoscaling policies in cloud environments. Model validation, conducted through controlled fault injection in critical components and statistical analysis with a 95% confidence interval, demonstrated that: (i) the basic architecture on physical hardware presents 99.75% availability, corresponding to 21.89 hours of annual downtime; (ii) the introduction of physical redundancy increases availability to 99.9994%, reducing downtime by 99.75%; (iii) virtualization, although reducing availability exclusively (38.92 hours of downtime), when associated with host-level redundancy, achieves 99.86% availability, representing a 44.82% reduction in downtime compared to the baseline scenario; and (iv) in the public cloud, SPN models reveal that reactive scheduling policies lead to saturation and throughput manipulation under high load. We conclude that virtualized architectures with redundancy and elastic scaling in the cloud are effective strategies for ensuring high availability ($>$99.8%) and consistent performance. The main contributions of this work include a validated analytical framework for capacity planning, quantitative guidelines for architectural improvement, and a reproducible experimental methodology for evaluating learning platforms.

Keywords: Availability. RBD. CTMC. SPN. LAMP. Moodle.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
CLMS	Cloud Learning Management System
CTMC	Continuous-Time Markov Chain
EC2	Elastic Compute Cloud
IaaS	Infrastructure as a Service
LAMP	Linux, Apache, MySQL, PHP/Perl/Python
LMS	Learning Management System
MBaaS	Mobile Backend as a Service
Moodle	Modular Object-Oriented Dynamic Learning Environment
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
PN	Petri Net
RBD	Reliability Block Diagram
SaaS	Software as a Service
SLA	Service Level Agreement
SO	Software
SPN	Stochastic Petri Net
TTF	Time to Failure
TTR	Time to Repair
UI	User Interface
VM	Virtual Machine

LISTA DE SÍMBOLOS

α	Taxa de reboot do Hardware
μ	Taxa de reparo
λ	Taxa de falha

SUMÁRIO

1 INTRODUÇÃO

À medida que as tecnologias progrediram e a necessidade por métodos flexíveis e acessíveis de ensino aumentou, os Learning Management System (LMS) tornaram-se elementos essenciais nos ambientes educacionais. Uma das principais soluções nesta área é o Modular Object-Oriented Dynamic Learning Environment (Moodle), um sistema de gerenciamento de aprendizagem de código aberto amplamente utilizado em instituições educacionais devido à sua habilidade de oferecer recursos robustos para desenvolvimento e entrega de cursos, suporte multilíngue e opções de personalização (AL-AJLAN; ZEDAN, 2008). Hoje, é importante manter essas plataformas disponíveis para garantir uma experiência educacional plenamente eficaz. A resiliência de um LMS como o Moodle se refere à capacidade do sistema de resistir às falhas, mantendo suas funcionalidades principais e recuperando-se rapidamente de interrupções além de apresentar desempenho que atenda às necessidades dos seus utilizadores

Esta dissertação propõe, através do uso modelos combinatórios como Diagramas de Blocos de Confiabilidade – RBD –, Cadeias de Markov de Tempo Contínuo – Continuous-Time Markov Chain (CTMC) – e Redes de Petri Estocásticas – Stochastic Petri Net (SPN) – avaliar o desempenho e a disponibilidade da instalação do Moodle dentro de uma infraestrutura Linux, Apache, MySQL, PHP/Perl/Python (LAMP) bem como estudar cenários alternativos que melhorem o seu desempenho. A união desses modelos permite uma avaliação detalhada e abrangente do sistema, levando em consideração as interdependências entre seus componentes, os efeitos de suas falhas/reparos e aqueles menos impactantes no resultado final. A análise sugerida tem como objetivo aumentar a disponibilidade em plataformas LMS, por meio da avaliação de métricas relacionadas ao desempenho e à disponibilidade do sistema, tais como taxa de utilização dos recursos e tempo médio entre falhas e reparos.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

A constante evolução tecnológica, impulsionada por eventos globais como a pandemia de COVID-19, alçou os Ambientes de Gestão de Aprendizagem (LMS) de ferramentas auxiliares a sistemas essenciais à operação e continuidade das atividades educacionais (TESAR, 2020). Esses sistemas se tornaram essenciais para a criação, gestão e distribuição de conteúdo digital, desempenhando papel central na interação entre docentes, discentes e o conteúdo educacional.

Dentre os diversos LMS disponíveis, o Moodle destaca-se como a solução de código aberto mais amplamente adotada (ALTINPULLUK; KESIM, 2021). Desenvolvido em PHP no início dos anos 2000, o Moodle possui instalação relativamente simples, sendo tradicionalmente implantado sobre a pilha LAMP (Linux, Apache, MySQL, PHP) (COMMUNITY, 2024), o que contribui para sua popularidade em instituições de ensino de diferentes portes e regiões (AL-AJLAN; ZEDAN, 2008).

A confiabilidade, a disponibilidade e a resiliência desses ambientes digitais constituem requisitos fundamentais para garantir uma experiência de aprendizado uniforme e de alta qualidade. No contexto atual, a resiliência não é apenas ao tempo de atividade, mas envolve também a habilidade do sistema para resistir a interrupções, manter funções críticas em operação e se recuperar prontamente de falhas. Uma análise minuciosa da disponibilidade é necessária para aprimorar a resiliência das plataformas LMS, garantindo que as atividades acadêmicas sofram o mínimo de interrupção.

A análise de disponibilidade não se limita simplesmente à quantificação do tempo em que o sistema está operacional; envolve uma compreensão detalhada de como o sistema responde a diferentes exigências (MACIEL, 2023b). Nesse contexto, os modelos de estado, que descrevem o comportamento do sistema por meio de estados e transições, são essenciais. Eles oferecem uma base sólida para a avaliação quantitativa da disponibilidade, permitindo a identificação de áreas críticas e a adoção de estratégias de mitigação de risco, o que, por sua vez, aprimora a resiliência do sistema.

A transição do LMS para um ponto central das atividades acadêmicas elevou o atributo de desempenho a um requisito indispensável. Atrasos no tempo de resposta ou uma latência elevada comprometem diretamente a experiência de uso, sendo agravados pela variabilidade da carga de trabalho: operações intensivas no banco de dados, mesmo com poucos usuários, podem impactar negativamente o sistema mais do que acessos simultâneos a conteúdos estáticos. Essa imprevisibilidade operacional torna a identificação de gargalos e falhas um desafio constante.

Adicionalmente, em infraestruturas de nuvem, a escalabilidade constitui um fator decisivo para garantir níveis adequados de desempenho e disponibilidade. A configuração de mecanismos de *auto-scaling* requer um equilíbrio cuidadoso entre custo e desempenho. Políticas reativas podem ser ineficazes sob picos de carga inesperados, enquanto abordagens proativas incorrem em risco de superprovisionamento. Como testar esses comportamentos diretamente em ambientes de produção pode acarretar interrupções indesejadas, a modelagem estocástica

surge como alternativa viável para análise preditiva.

A modelagem formal é, portanto, um instrumento poderoso para essa análise. Modelos de espaço de estado, como as Cadeias de Markov de Tempo Contínuo (CTMC) e Redes de Petri Estocásticas (SPN), permitem representar transições entre estados com base em taxas de falha e de reparo, incorporando elementos de concorrência, sincronização e comportamento probabilístico. Essas abordagens complementam os Diagramas de Blocos de Confiabilidade (RBD), que, embora eficazes na representação da estrutura lógica do sistema, são limitados quanto à modelagem de dinâmicas temporais e de dependência entre recursos.

O uso de simulação pode ser uma alternativa à modelagem formal, contudo, enquanto técnicas de simulação exploram o comportamento do sistema por meio de múltiplas execuções experimentais para fornecer resultados com aproximação estatística, a abordagem via modelagem formal busca a obtenção de soluções analíticas. Para modelos cujo espaço de estados é computacionalmente tratável, essa metodologia permite a obtenção de métricas de desempenho exatas e uma análise de todos os "comportamentos" possíveis. Essa característica oferece uma compreensão abrangente do sistema, eliminando a incerteza estatística inerente à amostragem de cenários simulados e permitindo focar a análise parâmetros fundamentais, como as taxas de falha e reparo, por exemplo.

A integração de RBD, CTMC e SPN constitui uma forma robusta para avaliar tanto a disponibilidade quanto o desempenho do Moodle operando sobre uma infraestrutura LAMP. Essa abordagem multifacetada permite capturar a complexidade do sistema sob diversas perspectivas, identificando gargalos, interdependências e efeitos de propagação de falhas.

Dessa forma, esta dissertação justifica-se por propor um framework analítico que visa: (i) identificar os componentes críticos para a disponibilidade e o desempenho da plataforma Moodle; (ii) avaliar quantitativamente o impacto de alterações arquiteturais, como a introdução de redundância ou a variação de tipos de instância de nuvem; e (iii) oferecer suporte analítico à tomada de decisão, com vistas à otimização da alocação de recursos e ao equilíbrio entre desempenho e disponibilidade.

A contribuição científica reside, portanto, na construção de um modelo estocástico abrangente e integrado que reflita o comportamento de um LMS complexo, fornecendo subsídios concretos para o planejamento proativo de capacidade e para o aprimoramento da resiliência e eficiência dos ambientes de aprendizagem digitais.

1.2 OBJETIVOS

O objetivo central deste trabalho é analisar e avaliar quantitativamente o desempenho e a disponibilidade de um ambiente virtual de gestão de aprendizagem, especificamente o Moodle, a partir do desenvolvimento e aplicação de uma abordagem de modelagem híbrida e hierárquica baseada em RBD, CTMC e SPN. Essa abordagem permite identificar parâmetros sensíveis, analisar cenários críticos e propor melhorias estruturais que contribuam para a aumento da eficiência e da disponibilidade do sistema.

Para alcançar o objetivo geral, os seguintes objetivos específicos foram estabelecidos:

1. Desenvolver e validar um modelo formal hierárquico que integre múltiplos formalismos:
   - **RBD:** para quantificar métricas de disponibilidade da arquitetura, considerando as dependências lógicas entre os componentes.
   - **CTMC e SPN:** para capturar o comportamento dinâmico do sistema, incluindo os processos de falha e reparo, a chegada de requisições de utilizadores e a utilização de recursos.
2. Estimar um conjunto de métricas de interesse a partir dos modelos, incluindo:
   - Métricas de Disponibilidade: Disponibilidade, Indisponibilidade, Número de nove, Tempo Médio de Falha (*Mean Time to Failure (MTTF)*), Tempo Médio de Reparo (*Mean Time to Repair (MTTR)*), *Downtime* e *Uptime*.
   - Métricas de Desempenho: Vazão (*Throughput*), probabilidade de descarte, taxa de descarte e utilização do sistema.
3. Identificar os componentes e parâmetros mais impactantes para o desempenho e disponibilidade do sistema, por meio de análises de sensibilidade.
4. Quantificar métricas de desempenho e disponibilidade do sistema, por meio da modelagem e avaliação de cenários de aprimoramento da plataforma.
5. Apresentar recomendações e estratégias baseados nos resultados obtidos a fim de fornecer suporte à decisão para o planejamento de ambientes de aprendizagem digitais.

1.3 TRABALHOS RELACIONADOS

Para situar esta pesquisa no contexto científico e fundamentar sua contribuição, esta seção apresenta uma revisão dos trabalhos que abordam os pilares centrais da pesquisa. A análise foca em estudos sobre a avaliação de desempenho e disponibilidade de componentes da pilha LAMP, a implementação de sistemas de gestão de aprendizagem em ambientes de nuvem e a aplicação de modelagem estocástica para análise de desempenho e escalonamento automático.

A literatura inicial focou no desempenho de componentes individuais da pilha LAMP. Jogi e Sinha (2016) realizaram estudo sobre o desempenho do banco de dados, incluindo o MySQL, em cenários de alta carga de gravação. Kurien, Mathew e Mana (2022) estudaram a interação do PHP com o MySQL em um Sistema de Gerenciamento de Ativos, com foco na segurança e integridade dos dados. Koch e Hao (2021) apresentaram uma análise do desempenho de componentes individuais em uma pilha LAMP, com ênfase particular no Servidor Web Apache em ambientes de nuvem (AWS).

Paralelamente, outros estudos avaliaram LMS de uma perspectiva mais sistêmica. Wannapiroon, Kaewrattanapat e Premsmith (2019), por sua vez, desenvolveram um Sistema de Gerenciamento de Aprendizagem em Nuvem (*Cloud Learning Management System (CLMS)*), avaliando eficiência e satisfação do usuário, mas sem incluir testes relacionados à disponibilidade.

Mais recentemente, pesquisas começaram a endereçar a avaliação de desempenho e disponibilidade de Ambientes Virtuais de Aprendizagem (AVAs), como o Moodle, em infraestruturas de nuvem. Lima et al. (2021a) e Lima et al. (2021b) realizaram uma avaliação focada no desempenho e no consumo de energia do ambiente Moodle em nuvens privadas. Avançando nesta linha, Gonçalves et al. (2022) propuseram modelos estocásticos para o planejamento de AVAs, comparando o impacto de arquiteturas baseadas em contêineres e máquinas virtuais. Estes trabalhos confirmam a relevância de avaliar o Moodle, mas focam primariamente em desempenho e consumo de energia, ou no planejamento de infraestrutura, sem uma análise aprofundada da disponibilidade validada por falhas nos componentes base.

Embora os trabalhos de Lima et al. (2021a), Lima et al. (2021b) e Gonçalves et al. (2022) abordem o Moodle, o presente trabalho se diferencia e estende a literatura ao focar especificamente na avaliação da plataforma sobre a pilha LAMP tradicional. A contribuição central é uma metodologia híbrida e hierárquica (RBD, CTMC e SPN) para uma análise detalhada da disponibilidade do Moodle. Crucialmente, esta modelagem é validada experimentalmente atra-

vés da injeção de falhas e reparos nos componentes críticos da pilha (Hardware, SO, Apache, MySQL e PHP), provando a precisão e aplicabilidade dos modelos propostos de uma forma não explorada pelos estudos anteriores.

A aplicação de modelagem estocástica para avaliar a disponibilidade de serviços em nuvem é uma área de pesquisa ativa e recente, o que reforça a relevância desta dissertação. Metodologias similares têm sido aplicadas a outros serviços de nuvem análogos ao Moodle. Por exemplo, uma série de estudos recentes modelou a disponibilidade e o desempenho de servidores de arquivos, como o *Nextcloud*, em nuvens privadas e em *Apache Cloudstack*, utilizando SPN. Em linhas semelhantes, Borges et al. (2025) avaliaram um sistema de vigilância por vídeo com armazenamento distribuído e Silva et al. (2024) analisaram a disponibilidade de uma plataforma *Mobile Backend as a Service (MBaaS)*, ambos utilizando modelagem de disponibilidade e análise de sensibilidade. Callou e Vieira (2024) também fornecem uma análise geral da disponibilidade e desempenho de serviços em nuvem.

No contexto específico dos sistemas de *e-learning* e escalonamento automático, trabalhos pioneiros como o Casale e Cremonesi (2006) já abordavam a modelação e análise de desempenho de um sistema de *e-learning* em larga escala, embora em arquiteturas físicas. O estudo seminal de Fe et al. (2017) propôs um modelo SPN para auxiliar no planeamento da nuvem com escalonamento automático, avaliando métricas de desempenho e custo.

Para além da modelação reativa, a literatura explora outras estratégias de escalonamento. Abordagens proativas ou preditivas tentam antecipar futuras necessidades de recursos com base em dados históricos, utilizando técnicas como séries temporais ou *machine learning* como em Padala, Hou e Shin (2009) e Gandhi et al. (2014). Embora potencialmente mais eficientes na gestão de picos de carga previsíveis, estes métodos acarretam uma maior complexidade de implementação e o risco de sobreprovisionamento se as previsões forem imprecisas. Investigações mais recentes também combinam modelos de desempenho com heurísticas de otimização, como o *GRASP*, para automatizar a busca por configurações ótimas que equilibrem custo e desempenho, apontando para uma futura automação do planeamento de capacidade a exemplo de Fé I. et al. (2022). Em um contexto correlato de gerenciamento de recursos, Feitosa et al. (2025) realizaram uma avaliação de desempenho abrangente de estratégias de migração de contêineres, um aspecto vital da elasticidade em ambientes modernos. A nossa escolha de focar no escalonamento reativo é justificada pela sua prevalência nas configurações padrão das plataformas de nuvem comerciais e pela sua relevância direta para os administradores de sistemas que gerem estes ambientes no dia-a-dia.

A Tabela 1 apresenta uma comparação entre trabalhos relacionados e a presente dissertação em termos de principais contribuições.

A abordagem proposta nesta dissertação, portanto, fornece uma análise quantitativa aprofundada de métricas de disponibilidade (*MTTF*, *MTTR*) e desempenho (utilização, probabilidade de descarte, taxa de descarte e vazão efetiva), focada especificamente na implantação do Moodle sobre a pilha LAMP. Enquanto outros estudos recentes analisaram o desempenho do Moodle em nuvens privadas, a validação experimental por injeção de falhas nos componentes específicos do LAMP (Hardware, SO, Apache, MySQL, PHP) preenche uma lacuna metodológica clara. Por fim, os resultados apresentados fornecem diretrizes práticas para gestores e pesquisadores interessados em aprimorar a disponibilidade e o desempenho do Moodle, com especial relevância para instituições educacionais que usam LMSs de alta disponibilidade para garantir a continuidade de suas operações acadêmicas.

Tabela 1 – Tabela comparativa dos trabalhos relacionados

Autores	Tipo de LMS *	Desempenho	Disponibilidade	Redundância	Análise de Sensibilidade	Performabilidade	Virtualização
(JOGI; SINHA, 2016)	N/A (Bancos de Dados)	✓					N/A
(WANNAPIROON; KAEWRATTANAPAT; PREMSMITH, 2019)	Cloud LMS	✓					Nuvem
(KURIEN; MATHEW; MANA, 2022)	N/A (Digital AMS)						N/A
(KOCH; HAO, 2021)	N/A (Pilha LAMP)	✓					Nuvem (AWS)
(CASALE; CREMONESI, 2006)	N/A (E-learning)	✓					Local (Grid)
(FE et al., 2017)	N/A (IaaS)	✓	✓		✓	✓	Local e Nuvem (Auto-scaling)
(PADALA; HOU; SHIN, 2009)	N/A (Recursos Virtuais)	✓					Local
(GANDHI et al., 2014)	N/A (Nuvem)	✓					Nuvem (Provisioning)
(Fé I. et al., 2022)	N/A (Nuvem)	✓	✓	✓		✓	Nuvem Privada
(LIMA et al., 2021a; LIMA et al., 2021b)	Moodle	✓					Nuvem Privada
(GONÇALVES et al., 2022)	Moodle (AVA)	✓					Nuvem (VM/Contêiner)
(SILVA et al., 2023)	N/A (Serv. Arquivos)	✓					Nuvem Privada
(CALLOU; VIEIRA, 2024)	N/A (Genérico)	✓	✓			✓	Nuvem
(LEONARDO; BEZERRA; CALLOU, 2024)	N/A (Nextcloud)	✓	✓			✓	Nuvem Privada
(GOMES; CALLOU, 2024)	N/A (Serv. Arquivos)		✓				Nuvem Privada
(SILVA et al., 2024)	N/A (MBaaS)		✓		✓	✓	Nuvem
(LEONARDO; CALLOU, 2025)	N/A (Nextcloud)		✓				Nuvem Privada
(BORGES et al., 2025)	N/A (Vigilância)		✓	✓			Nuvem (Distribuído)
(FEITOSA et al., 2025)	N/A (Contêineres)	✓			✓		Nuvem Privada
Nosso trabalho	**Moodle**	✓	✓	✓	✓	✓	**Local e Nuvem**

*N/A = "Não Aplicável".

Fonte: Elaborada pelo autor (2025)

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em sete capítulos. Inicialmente, o primeiro capítulo apresenta uma visão geral introdutória sobre o tema estudado bem como a motivação e justificativa para o trabalho, os objetivos pretendidos com o estudo, além de trabalhos relacionados.

Em seguida, no Capítulo 2 é apresentado a fundamentação teórica que dá suporte a este estudo. Nesse capítulo é discutido a pilha LAMP, uma visão geral sobre o LMS Moodle,

bem como conceitos indispensáveis sobre avaliação de desempenho de sistemas e formalismos inerentes ao caso. Nessa seção se discute o conceito sobre injeção de falhas. Por fim, são estudados conceitos importantes sobre virtualização e escalonamento automático.

O Capítulo 3 é dedicado à metodologia e à arquitetura propostas na dissertação. Nesta parte do trabalho detalhamos como a arquitetura básica foi implementada bem como a metodologia de pesquisa necessária para sua validação.

O Capítulo 4 é dedicado aos modelos analíticos propostos com base na arquitetura estudada no capítulo anterior. Aqui mostramos uma arquitetura básica de uma instalação Moodle em máquina física como também arquiteturas derivadas para implementação em sistemas virtualizados. Propomos também arquiteturas redundantes bem como de desempenho da plataforma.

No Capítulo 5, o foco se volta para a validação do modelo analítico da arquitetura básica. É apresentado o processo de validação experimental, que busca confirmar se o modelo teórico representa, com um grau de confiança estatístico, o comportamento do sistema real. Para tal, uma implementação da arquitetura básica é submetida a um experimento de injeção de falhas, onde dados sobre a disponibilidade são coletados e analisados. culminando na comparação do resultado de disponibilidade estimado pelo modelo com o intervalo de confiança obtido a partir dos dados experimentais. O objetivo deste capítulo é, portanto, estabelecer a credibilidade do modelo base, assegurando que ele possa ser utilizado com confiança como um alicerce para as análises comparativas e os estudos de caso que serão conduzidos no Capítulo 6.

O Capítulo 6 dedica-se à análise detalhada das arquiteturas desenvolvidas no Capítulo 4. O escopo principal reside na avaliação de desempenho e na mensuração da disponibilidade do sistema, utilizando métricas escolhidas para cada modelo. Concomitantemente, uma análise de sensibilidade é conduzida para inferir os parâmetros mais determinantes sobre as métricas supracitadas. Em seguida, por meio de uma série de experimentações, examina-se como alterações nos parâmetros de entrada ou de saída do modelo podem influenciar cenários distintos e, por conseguinte, orientar a seleção da instalação mais apropriada do Moodle. Nesse capítulo, também, é realizado estudo de caso sobre instâncias virtuais.

Por fim, o Capítulo 7 apresenta a conclusão da trabalho, indicando as principais contribuições para a sociedade, dificuldades encontradas bem como direções para futuras pesquisas.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais necessários à compreensão deste trabalho. Inicialmente, é apresentada uma visão geral sobre os Sistemas de Gestão de Aprendizagem — LMS. Em seguida, para melhor entender a configuração do LMS discutido nesta dissertação, exploramos a Pilha LAMP. A continuação inclui um estudo sobre Avaliação de Desempenho de Sistemas, com ênfase em dependabilidade por meio de modelos estocásticos associados a modelos combinatórios e de espaço de estado. É igualmente crucial entender a análise de sensibilidade paramétrica, assim como o procedimento de injeção de falhas em sistemas. De igual modo é explorado conceitos cruciais sobre virtualização, escalonamento automático.

2.1 *LEARNING MANAGEMENT SYSTEM – LMS*

O *Learning Management System* (LMS), traduzido como Sistema de Gestão de Aprendizagem, é uma tecnologia baseada na web, plataforma ou software voltado para organizar, gerenciar e avaliar processos de ensino. Essencialmente pensado para o *e-learning*[1], este sistema geralmente inclui duas partes principais: Um servidor responsável pelas funções centrais e uma interface de usuário – *User Interface (UI)* – utilizada por instrutores, estudantes e administradores.

Nos últimos anos, o setor educacional sofreu mudanças significativas devido ao avanço acelerado da tecnologia. Uma das transformações mais evidentes foi a popularização dos LMS, que surgiram como uma alternativa ao modelo tradicional de ensino em sala de aula. O LMS oferece diversos recursos, incluindo salas virtuais, fóruns, avaliações online e ferramentas colaborativas, que não só enriquecem a experiência de aprendizado, mas também facilitam a interação entre estudantes e professores. Este movimento em direção ao aprendizado digital foi motivado pela demanda por flexibilidade, acessibilidade e eficiência no contexto educacional (LIAW; HUANG, 2013).

Embora os conceitos iniciais sobre LMS datem de 1924 (TEAM, 2024) e o desenvolvimento do principal expoente, o Moodle [2], remonte a 2002 (COMMUNITY, 2024), um aumento substancial em sua utilização foi observado durante a pandemia da COVID-19, quando inúmeras

[1] Forma de ensino e aprendizado que utiliza tecnologias digitais e recursos online para facilitar a aquisição de conhecimentos e habilidades.

[2] Disponível em https://moodle.org/

instituições educacionais foram obrigadas a fazer uma rápida transição para o aprendizado remoto.

O site Forbes Advisor (HAAN, 2024) realizou uma avaliação das principais opções de LMS, considerando vários fatores, tais como facilidade de utilização, custo, capacidades de personalização e atendimento ao cliente. O Moodle destaca-se por sua interface intuitiva e uma ampla gama de funcionalidades, incluindo aprendizado móvel, videoconferência, detecção de plágio, apoio para cursos online abertos massivos e integrações. Sua natureza de código aberto garante uma alta personalização, tornando-o adequado para empresas, escolas e universidades.

Moodle foi criado usando a estrutura de código aberto denominada LAMP, que é formada por Linux (sistema operacional), Apache (servidor web), MySQL (banco de dados) e PHP (linguagem de programação). Devido à portabilidade desses elementos e à adaptabilidade da plataforma, ela oferece suporte a uma grande variedade de ambientes. Apesar de o Moodle poder ser facilmente instalado em outras arquiteturas tecnológicas, a configuração LAMP permanece como a escolha preferida pelos administradores do Moodle (Moodle Community, 2024).

As bases pedagógicas do sistema estão fundamentadas no construtivismo social que define que a interação entre aluno e conteúdo é a principal responsável pelo conhecimento adquirido pelo discente (MOODLE, 2020). A plataforma oferece suporte para cursos presenciais, mistos e totalmente online, disponibilizando mais de 20 tipos de atividades, como fóruns, salas de bate-papo, wikis, glossários, quizzes e tarefas (TRINDADE, 2020).

Dois módulos de questionário importantes são o *survey* e o *feedback*. O módulo survey disponibiliza questionários com perguntas pré-definidas, como o COLLES (*Constructivist On-Line Learning Environment Survey*), que aborda a qualidade do ambiente de aprendizado online, e o ATTLS (*Attitudes to Thinking and Learning Survey*), que avalia atitudes em relação ao pensamento crítico e ao ensino (TRINDADE, 2020). Em contrapartida, o módulo *feedback* permite que os professores criem questionários personalizados para coletar opiniões dos alunos (TRINDADE, 2020).

2.2 PILHA LAMP

Dentre os conjuntos de software frequentemente utilizados para servidores web, LAMP – Acrônimo para Linux, Apache, MySQL, PHP – é uma implementação amplamente adotada (AKATSU et al., 2020) e de fácil instalação. A combinação do Sistema Operacional Linux,

do Servidor Web Apache, do banco de dados MySQL e da linguagem de programação PHP (PRANAM, 2018) tornou-se uma escolha comum para diversos sites.

A seguir, apresentamos uma explicação sucinta de cada componente dessa pilha amplamente empregada em sistemas web.

- Linux: Desenvolvido como um sistema operacional de código aberto nos anos 1990 por Linus Torvalds, o Linux surgiu a partir do Unix (SAILELLAH, 2023). Logo conquistou popularidade e, hoje em dia, é implementado em diversas plataformas, desde desktops até servidores. A razão principal do sucesso do Linux está na sua confiabilidade, segurança e capacidade de escalonamento, permitindo personalizações que atendem às necessidades específicas de indivíduos e organizações (NARCISO, 2023). O Linux funciona como a camada inicial da pilha LAMP e oferece suporte aos outros componentes das camadas superiores;

- Apache: O acesso a recursos na web depende do emprego de um software apto a atender às requisições de um cliente, o qual é denominado servidor web. Atualmente, existem várias opções disponíveis, como *IIS*, *Nginx* e *lighttpd*, etc. O *Apache* é um dos servidores web mais frequentemente utilizados. Este servidor oferece um amplo conjunto de funcionalidades, ferramentas e uma extensa comunidade de usuários ativos, o que o torna uma escolha frequente como principal servidor web (KOCH; HAO, 2021). Conhecido também como servidor *HTTP Apache*, foi lançado em 1995, e rapidamente se destacou, tornando-se a solução de servidor web mais utilizada já em 1996 (APACHE, 2024). De acordo com uma pesquisa recente de (NETCRAFT, 2024), constatou-se que até fevereiro de 2024, o *Apache* liderava o mercado juntamente com o *Nginx*.

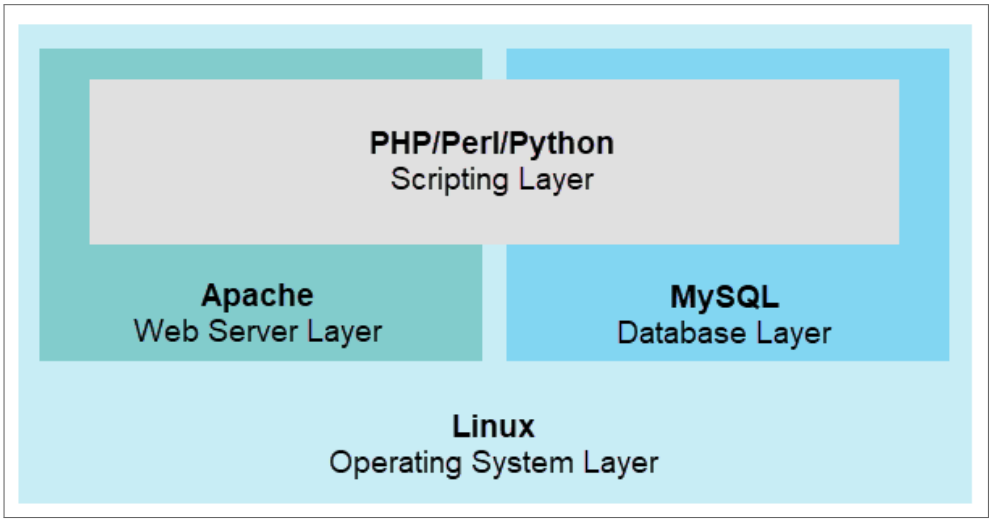
  Devido à sua natureza de código aberto e à colaboração comunitária, o *Apache* é geralmente visto como mais acessível e fácil de aprender em comparação a outros servidores que demandam um maior esforço para aprendizado – por exemplo, *NGINX* (GIRVIN, 2025).

- MySQL: Reconhecido como um dos bancos de dados relacionais mais utilizados, MySQL é celebrado por seu excelente desempenho, confiabilidade e facilidade de operação (MySQL, 2024). Sua vasta aceitação deriva da sua compatibilidade com uma gama de linguagens de programação populares, como PHP, Python, Java, C, entre outras. Frequente na função de banco de dados web, o MySQL possui uma capacidade significativa

de armazenamento, guardando desde dados individuais até inventários abrangentes de itens acessíveis. Ele simplifica a automação da recuperação de dados e proporciona suporte eficiente na criação de aplicativos web em PHP (KURIEN; MATHEW; MANA, 2022). Sendo o banco de dados open-source mais reconhecido, MySQL é frequentemente preferido para diversas aplicações web (MySQL, 2024).

- PHP: Acrônimo recursivo para *PHP: Hypertext Preprocessor*. Esta linguagem de script, de código aberto, é largamente empregada, especialmente para o desenvolvimento web, e pode ser integrada ao HTML (PHP, 2024). As páginas PHP podem conter instruções HTML junto com código PHP, facilitando seu uso por iniciantes na programação. Além disso, possui funcionalidades avançadas que a tornam ideal para projetos web mais complexos, como o Moodle. De acordo com (PYPL, 2024), PHP é a sétima linguagem de programação mais utilizada globalmente, destacando-se como uma das principais opções para desenvolvedores na criação de aplicações web.

Figura 1 – Arquitetura da Pilha LAMP



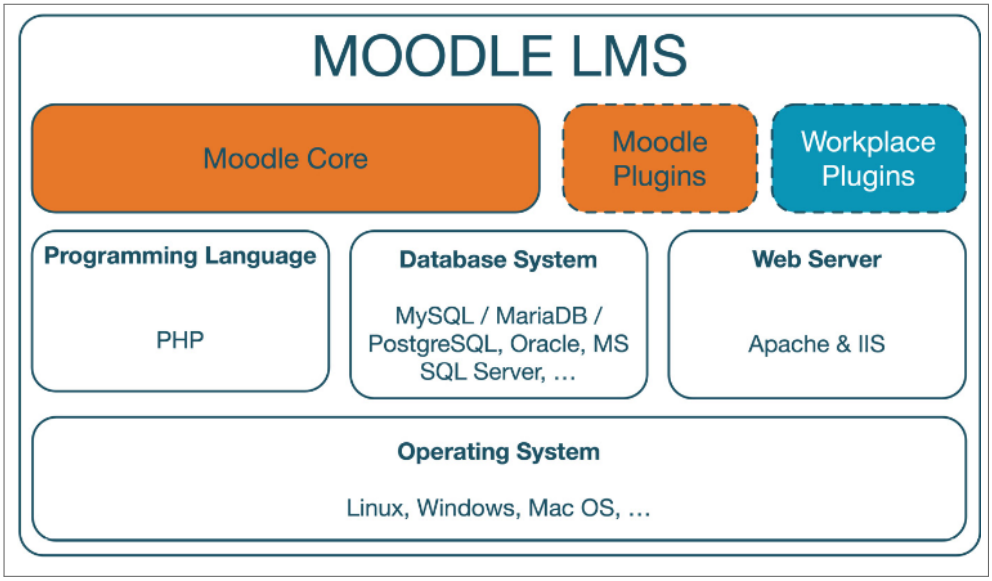
Fonte: SYED (2022)

A pilha LAMP (Figura 1) funciona como um sistema unificado que oferece as capacidades de um aplicativo web. O processo começa quando o cliente faz uma requisição, a qual é atendida pelo servidor Apache. Este, por sua vez, determina se o arquivo solicitado é estático ou dinâmico. Se o arquivo for dinâmico, o Apache encaminha a solicitação para o interpretador de linguagem adequado, como o PHP. Nesta fase, o script pode se comunicar com o banco de dados MySQL para buscar ou armazenar dados para uso futuro.

O PHP, então, manipula os dados e retorna o conteúdo dinâmico em HTML para o servidor Apache, que o renderiza no navegador do usuário. Dessa forma, o ciclo de requisição e resposta é concluído.

Conforme ilustrado na Figura 2, o Moodle foi projetado para ser instalado em uma pilha LAMP, que representa a configuração de instância mais básica disponível.

Figura 2 – Arquitetura Moodle



Fonte: BüCHNER (2022)

2.3 VIRTUALIZAÇÃO

A virtualização consiste no paradigma de separar o hardware do sistema operacional a partir da criação de uma camada de abstração entre eles. Essa tecnologia permite que, em uma mesma máquina física, exista vários máquinas virtualizadas, cada qual com seu sistema operacional e independência entre si apesar de estarem no mesmo servidor. O uso dessa arquitetura é possibilitado pelo hypervisor, um programa que cria e administra as Máquinas Virtuais (VMs) (POPEK; GOLDBERG, 1974). O hypervisor abstrai os recursos do hardware hospedeiro — como processador, memória e espaço de armazenamento — das máquinas virtuais e os distribui conforme a necessidade para cada VM, passando a sensação de que cada máquina tem seu próprio hardware exclusivo.

Existem duas formas principais de virtualização. A virtualização completa que simula todo o hardware, permitindo, assim, que os sistemas virtualizados operem isoladamente, contudo essa abordagem gera uma perda de desempenho. Por outro lado a paravirtualização adota uma

abordagem de colaboração, onde o sistema virtualizado interage diretamente com o hypervisor (BARHAM et al., 2003). Essa técnica diminui a sobrecarga e melhora o desempenho do sistema

Através dos anos a virtualização trouxe muitos benefícios, principalmente para aplicações web que agora podem ser desenvolvidas em qualquer local do planeta e implementada em um servidor na nuvem. O isolamento proporcionado pelo servidor assegura que cada VM funcione como um ambiente individual, evitando que uma falha ou problema de segurança em uma VM afete as outras, aumentando a segurança do sistema (WALDSPURGER, 2002). Com o passar dos anos essa tecnologia se tornou tão onisciente que praticamente qualquer usuário pode ter uma máquina virtualizada com sistema operacional de hardware dedicado.

Um dos maiores benefícios da virtualização é a sua flexibilidade, permitindo criar, apagar ou mover rapidamente máquinas virtuais (VMs) entre diferentes servidores físicos, conforme as demandas do ambiente de computação.

Plataformas de aprendizado, a exemplo do Moodle, que passam por picos de demanda em certos momentos – por exemplo, como início de semestre, semanas de avaliação ou encerramento do período letivo –, se beneficiam ao serem implementadas de forma virtualizada. A virtualização oferece a flexibilidade necessária para gerenciar mudanças, permitindo a distribuição de serviços entre várias VMs e garantindo um desempenho constante e de alta disponibilidade (CLARK et al., 2005).

Porém, otimizar a alocação de recursos e prever o desempenho sob diferentes cargas são desafios grandes. A flexibilidade da virtualização cria um novo problema: o gerenciamento de um sistema complexo. Essa complexidade justifica a necessidade de uma forma de modelagem formal, como a proposta aqui, para analisar e otimizar o desempenho do sistema a partir de métricas de interesse.

2.4 ESCALONAMENTO AUTOMÁTICO

A elasticidade é um dos principais atributos da computação em nuvem. Entendida como a capacidade do sistema se ajustar, em tempo de execução, os recursos computacionais para se adequar às variações de carga (HERBST; KOUNEV; REUSSNER, 2013). O escalonamento automático é o método que concretiza essa elasticidade, automatizando o provisionamento de recursos sem interferência humana a fim de aperfeiçoar o equilíbrio do sistema.

A adaptação de recursos normalmente acontecem de duas maneiras: Escalonamento Vertical compreendendo a modificação de recursos (ex: CPU, RAM) de uma VM existente e,

Escalonamento Horizontal que consiste em adicionar ou retirar instâncias de VM de um número de máquinas virtuais anteriormente determinadas.

Uma correta adoção de uma política de escalonamento automático é fator crítico para atender aos usuários e evitar desperdício de recursos computacionais. Contudo, a complexidade dessa configuração – que implica no conhecimento dos limites do sistema – demonstra a necessidade de um modelo estruturado que possibilite a análise de cenários para identificar a melhor forma de implementação do sistema(GANDHI et al., 2014).

O escalonamento automático está diretamente ligado ao conceito de elasticidade, uma característica fundamental da computação em nuvem que a distingue de outros paradigmas de computação distribuída, como a computação em grade (ARAUJO, 2015). A elasticidade é a capacidade que um sistema possui de alocar e desalocar recursos computacionais de forma dinâmica e autônoma, com o objetivo de adaptar-se às variações da carga de trabalho ao longo do tempo (ARAUJO, 2015), (LIMA, 2015). Isso permite que o sistema "estique" ou "encolha" sua capacidade para manter o desempenho acordado em um *Service Level Agreement (SLA)*, otimizando ao mesmo tempo os custos operacionais (ARAUJO, 2015). Ao consumidor, os recursos parecem ser ilimitados e podem ser ajustados sob demanda (MELL; GRANCE, 2011).

A alocação de recursos em uma nuvem do tipo *Infrastructure as a Service (IaaS)* é um desafio central (MANVI; SHYAM, 2014). As estratégias de gerenciamento de elasticidade são cruciais para o ambiente e podem ser classificadas como (ARAUJO, 2015):

- Reativas: As decisões de escalonamento (adicionar ou remover recursos) são tomadas em resposta atençao a limiares pré-definidos em métricas de desempenho, como utilização de CPU ou tempo de resposta. A sua principal desvantagem é o tempo necessário para que um novo recurso seja provisionado e esteja pronto para uso, o que pode levar a um período de degradação do serviço.

- Proativas: Utilizam técnicas de análise de séries temporais e aprendizado de máquina para prever o comportamento futuro da carga de trabalho. Com base nessas predições, os recursos são alocados ou desalocados antecipadamente, buscando evitar violações de SLA antes que ocorram (HERBST; KOUGIOUKOTAS; REMANN, 2013).

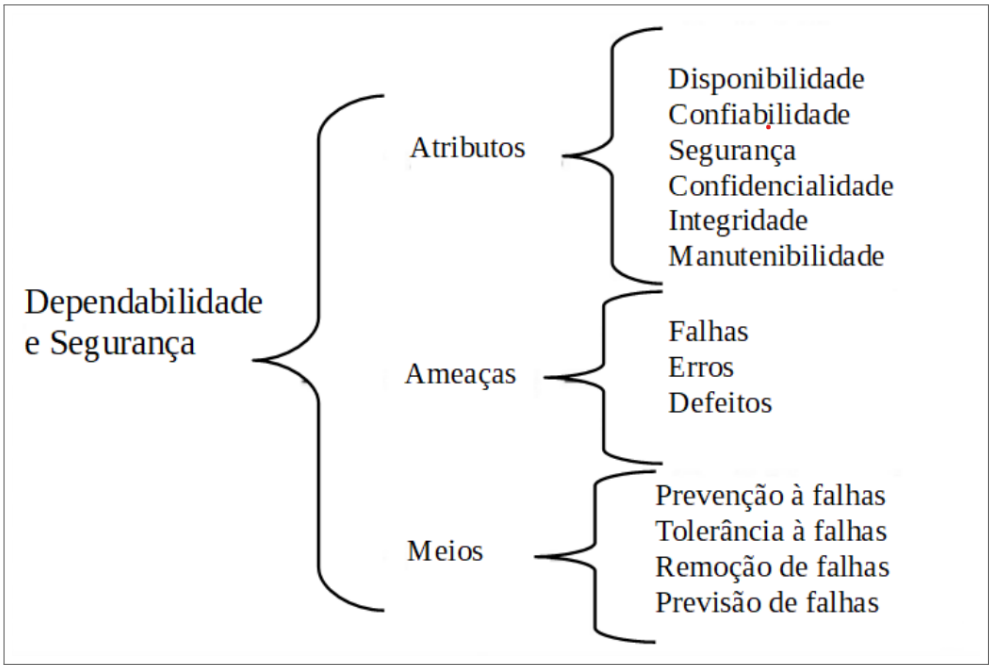
A escalabilidade horizontal, que consiste em alterar o número de instâncias de um recurso (como VMs), é a abordagem mais comum para implementar o escalonamento automático (GUPTA; CHRISTIE; MANJULA, 2017).

2.5 DEPENDABILIDADE

O conceito de dependabilidade remonta à década de 1820, quando Charles Babbage planejou um sistema de cálculo mecânico com o objetivo de eliminar erros humanos (SCHAFFER, 1994). O trabalho de Laprie (1995) lista os primeiros conceitos de dependabilidade e, desde então, tem sido usado na academia. Para Avizienis et al. (2004) a dependabilidade de um sistema é a capacidade de evitar falhas de serviço que são frequentes ou mais graves do que o aceitável.

Tanto que foi necessário criar uma taxonomia para demonstrar o funcionamento de sistemas confiáveis, como mostrado na Figura 3

Figura 3 – Árvore de confiabilidade e segurança



Fonte: Adaptado de AVIZIENIS et al. (2004)

Pode-se observar que a dependabilidade é um conceito que envolve seis requisitos principais (COSTA, 2015)

- Disponibilidade – Capacidade do sistema estar pronto para fornecer o serviço corretamente;
- Confiabilidade – Probabilidade de o sistema fornecer o serviço continuamente, sem erros, até um certo tempo t;
- Segurança – Ausência de consequências catastróficas para o(s) usuário(s) e o ambiente;

- Confidencialidade – Ausência de divulgação não autorizada de informações;
- Integridade – Ausência de alterações impróprias no estado do sistema;
- Manutenibilidade – Capacidade de passar por reparos e modificações;

Um elemento crucial ao desenvolver um modelo de avaliação de desempenho para o LMS é a Disponibilidade, que Maciel (2023a) define como a proporção estimada do tempo em que o sistema está funcional ao longo de seu ciclo de vida. Além disso, é fundamental entender a Disponibilidade Estacionária (A) como a relação entre o tempo de operação esperado e a soma do tempo de operação e falhas esperados. Esta medida pode ser expressa pela seguinte Equação 2.1:

$$A = \frac{E[Uptime]}{E[Uptime] + E[Downtime]} \tag{2.1}$$

Onde:

- A é a disponibilidade estacionária do sistema;
- E[Uptime] é o tempo de atividade esperado do sistema;
- E[Tempo de inatividade] é o tempo esperado de falha do sistema;

A disponibilidade é frequentemente expressa pelo número de noves, que indica a quantidade de dígitos 9 consecutivos na porcentagem de tempo em que o sistema está operacional (MARWAH et al., 2010). Por exemplo, se um sistema está disponível $99,9\%$ do tempo, sua disponibilidade é designada por 3 noves. A quantidade de noves é determinada pela Equação 2.2:

$$N = -\log_{10}(1 - A) \tag{2.2}$$

A indisponibilidade, em contraste com a disponibilidade, refere-se à probabilidade de que o sistema não esteja acessível. Pode ser calculado usando as Equações (2.3) e (2.4)

$$UA = \frac{E[Downtime]}{E[Downtime] + E[Uptime]} \tag{2.3}$$

$$UA = 1 - A \tag{2.4}$$

Através da indisponibilidade, pode-se calcular o período em que o sistema estará inativo dentro de um determinado intervalo de tempo. O tempo de inatividade anual indica o número

esperado de horas em que o sistema não funcionará durante um ano. Esse valor é determinado conforme a Equação 2.5:

$$Downtime_{year} = UA \times 8760h \tag{2.5}$$

Os dados de *uptime* e *downtime* não estão sempre disponíveis; em tais casos, são utilizadas as médias entre eventos de falhas e reparos para que:

- MTTF – Tempo médio para que falhas do sistema ocorram. Ele pode ser calculado usando a expressão:

$$MTTF = \int_0^\infty R(t)\,dt \tag{2.6}$$

  onde $R(t)$ é a função de confiabilidade do sistema, representando a probabilidade de que o sistema opere sem falhas até o tempo t.

- MTTR – Tempo médio para o sistema ficar disponível após uma falha ocorrer. Pode ser calculado usando a expressão:

$$MTTR = \int_0^\infty M(t)\,dt \tag{2.7}$$

onde $M(t)$ é a função de manutenibilidade do sistema, representando a probabilidade de que o reparo do sistema não seja concluído até o tempo t.

De tal forma que a disponibilidade (A) pode ser calculada usando a Equação 2.8:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.8}$$

2.6 AVALIAÇÃO DE DESEMPENHO DE SISTEMAS

A avaliação de desempenho é um processo sistemático que visa analisar e quantificar o comportamento de um sistema computacional em relação a um conjunto de métricas, sob uma determinada carga de trabalho (DANTAS, 2008). Esta avaliação é essencial para o planejamento de capacidade, otimização, depuração e comparação de sistemas (JAIN, 1991).

As métricas de desempenho mais comuns incluem, mas não se limitam a:

- Tempo de Resposta: Intervalo de tempo entre o envio de uma requisição por um usuário e o recebimento da resposta completa do sistema (MENASCé; ALMEIDA; DOWDY, 2002).
- Vazão (*Throughput*): Taxa na qual as requisições podem ser atendidas pelo sistema, geralmente medida em requisições por segundo ou transações por segundo (MENASCé; ALMEIDA; DOWDY, 2002).
- Utilização: Percentual de tempo em que um recurso (por exemplo, CPU, disco) está ocupado processando requisições (DANTAS, 2008).

A avaliação pode ser realizada através de medições em sistemas reais, simulação ou modelagem analítica (DANTAS, 2008). A modelagem analítica, utilizando técnicas como SPN, é particularmente útil para analisar o desempenho de sistemas complexos e distribuídos, como os encontrados em ambientes de nuvem (MELO, 2016a). A avaliação pode ser aplicada a domínios específicos, como bancos de dados em nuvens híbridas como no trabalho de (TEIXEIRA, 2017), aplicações *Software as a Service (SaaS)* a exemplo da tese de Gonzaga (2014), ou analisando o impacto de funcionalidades de segurança no consumo de recursos como tratado em Orozco (2018).

2.7 MODELOS PARA ANÁLISE DE DESEMPENHO E DISPONIBILIDADE

A avaliação de sistemas computacionais, em particular no campo dos Ambientes de Gestão de Aprendizagem demanda uma análise cuidadosa de suas características de desempenho e disponibilidade para assegurar a qualidade do serviço (TRINDADE, 2020). Dada a complexidade desses ambientes, a modelagem matemática surge como uma estratégia essencial para a análise quantitativa do sistema (MELO, 2018) e (DIAS, 2017). Na literatura, os modelos empregados para esse propósito podem ser divididos em duas categorias principais: modelos combinatórios e modelos de espaço de estado. Modelos combinatórios, como os RBDs, são apropriados para expressar a estrutura lógica do sistema e a forma como as falhas dos componentes afetam o sistema em sua totalidade (CLEMENTE, 2022) e (ČEPIN, 2011). Por outro lado, modelos de espaço de estado, como as CTMCs e as SPNs, são utilizados para representar o comportamento dinâmico do sistema, detalhando as transições entre diversos estados operacionais e de falha ao longo do tempo (MARSAN; CONTE; BALBO, 1984). Comumente, essas metodologias são

aplicadas de maneira complementar, em estruturas hierárquicas, para oferecer uma análise completa. A seguir detalharemos cada um desses formalismos.

2.7.1 Diagramas de Bloco de Confiabilidade - RBD

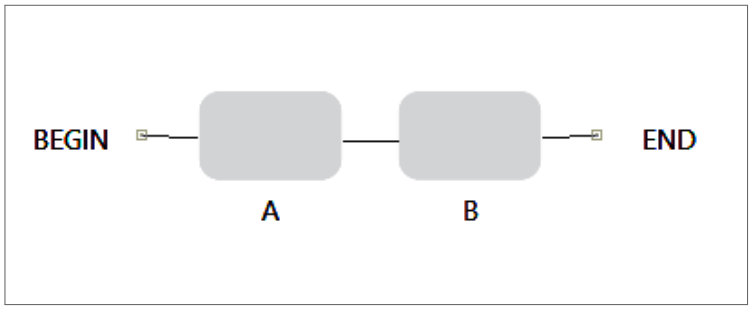
O Diagrama de Bloco de Confiabilidade é uma abordagem gráfica e um modelo combinatório utilizado para a análise de disponibilidade e confiabilidade de sistemas. A abordagem é fundamentada na lógica de sucesso, onde um sistema é considerado operacional se houver pelo menos um caminho contínuo de blocos funcionais conectando o nó de entrada ao nó de saída do diagrama. Este formalismo permite ilustrar de forma intuitiva como a confiabilidade dos componentes individuais (representados por blocos) contribui para a confiabilidade geral do sistema, seja levando ao sucesso ou à falha (CATELANI; CIANI; VENZI, 2015) e (LIU; WU, 2011).

O RBD é ideal para calcular métricas como confiabilidade, disponibilidade e MTTF de um sistema a partir dos dados de seus componentes (ČEPIN, 2011).

Um RBD é constituído por blocos conectados que podem ser organizados em três configurações principais para representar as interdependências lógicas entre os componentes do sistema:

- Configuração em Série: Representa uma lógica "E"(AND), na qual todos os componentes do sistema devem estar operacionais para que o sistema funcione (Figura 4). A falha de um único bloco causa a falha de todo o sistema.

Figura 4 – RBD em série



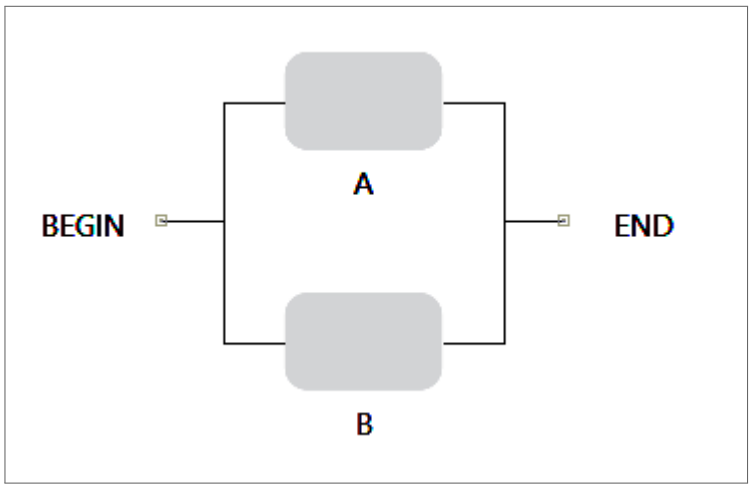
Fonte: Elaborada pelo autor (2025)

A disponibilidade (A_{serie}) de um sistema com n componentes em série é o produto da disponibilidade de seus componentes individuais (Ai), conforme Equação 2.9

$$A_{\text{série}} = \prod_{i=1}^{n} A_i = A_1 \times A_2 \times \cdots \times A_n \tag{2.9}$$

- Configuração em Paralelo: Representa uma lógica "OU"(OR) e é utilizada para modelar sistemas com redundância de componente (Figura 5). O sistema é considerado operacional se pelo menos um de seus n componentes redundantes estiver funcionando. O sistema só falha se todos os seus componentes falharem simultaneamente.

Figura 5 – RBD em paralelo



Fonte: Elaborada pelo autor (2025)

A disponibilidade ($A_{parelelo}$) de um sistema com n componentes em paralelo é calculado conforme Equação 2.11

$$A_{\text{paralelo}} = 1 - \prod_{i=1}^{n}(1 - A_i) \tag{2.10}$$

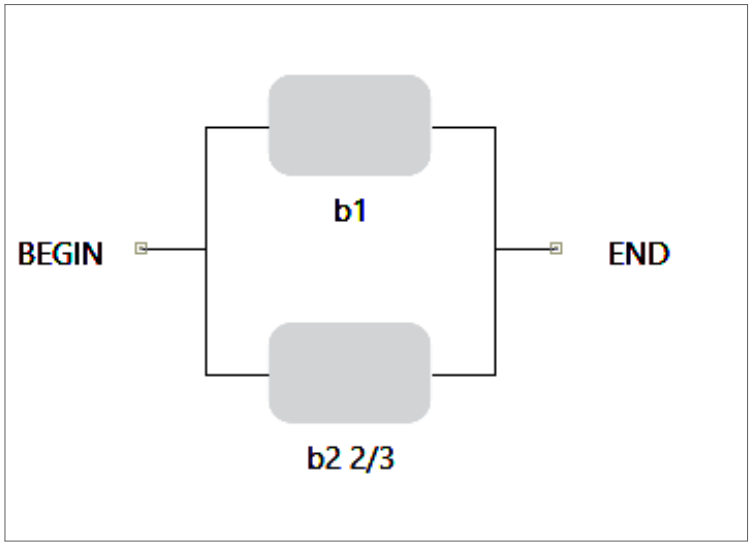
- Configuração K-out-of-N (KooN): Esta é uma configuração mais geral de redundância, na qual um sistema composto por N componentes idênticos é considerado operacional se pelo menos K desses componentes estiverem funcionando (onde $1 \leq K \leq N$) (Figura 6).

A disponibilidade de um sistema KooN (A_{KooN}), assumindo componentes idênticos com disponibilidade A_c é dada pela Equação 2.11

$$A_{\text{KooN}} = \sum_{i=k}^{n} \binom{n}{i} A_c^i \times (1 - A_c)^{n-i} \tag{2.11}$$

A configuração em paralelo é um caso particular de KooN, onde K=1. Esta estrutura é fundamental para modelar sistemas com redundância parcial, como por exemplo clusters de servidores.

Figura 6 – RBD KooN com K=2 e N=3



Fonte: Elaborada pelo autor (2025)

Uma das principais aplicações do RBD é como o primeiro nível em uma modelagem hierárquica (DIAS, 2017). Sistemas ou subsistemas complexos que operam em série (e cujos componentes possuem taxas de falha e reparo exponencialmente distribuídas) podem ser sintetizados por um modelo RBD. O resultado dessa sintetização é um único bloco com valores equivalentes de $MTTF$ e $MTTR$, por exemplo. Esses valores agregados são, então, utilizados como parâmetros para as transições temporizadas de um modelo de espaço de estado de nível superior, como uma SPN, reduzindo a complexidade do modelo final (CLEMENTE, 2022).

Apesar de sua utilidade, o RBD é um modelo estático e possui limitações. Ele tem dificuldades em representar sistemas com dependências complexas, diferentes políticas de reparo, ou sistemas cujo comportamento de falha muda dinamicamente ao longo do tempo. Para capturar essas dinâmicas complexas, modelos de espaço de estado como as SPNs e CTMCs são mais adequados e frequentemente utilizados de forma complementar (MELO, 2018)

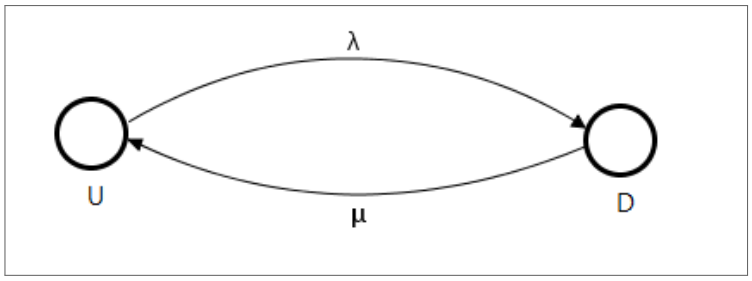
2.7.2 Cadeias de Markov de Tempo Contínuo - CTMC

Uma Cadeia de Markov de Tempo Contínuo (CTMC) é um formalismo matemático baseado em modelos de espaço de estado, fundamental para a análise de sistemas cujo comportamento evolui ao longo do tempo de forma probabilística. Uma CTMC é um processo estocástico caracterizado por possuir um conjunto discreto de estados e um parâmetro de tempo contínuo, o que significa que as mudanças de estado podem ocorrer em qualquer instante (SOUZA, 2009).

A principal característica deste processo é a propriedade de Markov, também conhecida como propriedade de "falta de memória". Esta propriedade estabelece que a evolução futura do sistema, a partir de um estado conhecido, depende exclusivamente de seu estado atual, sendo independente de todos os estados passados que levaram até ele. Para que um processo satisfaça a propriedade de Markov em tempo contínuo, o tempo de permanência em qualquer estado deve ser uma variável aleatória que segue uma distribuição exponencial. A taxa desta distribuição, denotada por λ, determina a frequência com que as transições para outro estado ocorrem.

Dentro de um contexto envolvendo qualquer sistema que possa ser consertado, este pode estar em estado de falha ou em condições operacionais. Quando os parâmetros "tempo até a falha" (*Time to Failure (TTF)*) e "tempo até o reparo" (*Time to Repair (TTR)*) seguem uma distribuição exponencial com taxas λ e μ, respectivamente, a disponibilidade do modelo pode ser ilustrada conforme a Figura 7.

Figura 7 – Modelo genérico em CTMC



Fonte: Elaborada pelo autor (2025)

Nesta configuração, o estado U (ativo) denota a condição operacional, enquanto o estado D (inativo) indica a falha (MACIEL, 2023a). O evento de falha é caracterizado pela transição do estado U para D com taxa λ, enquanto o reparo é descrito pela transição do estado D para U com taxa μ. Ademais, a matriz de taxas, Q, pode ser expressa conforme apresentado pela Equação 2.12.

$$Q = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix} \tag{2.12}$$

Resolvendo o sistema de equações, as probabilidades de estado estacionário são $\pi_U = \frac{\mu}{\lambda+\mu}$ e $\pi_D = \frac{\lambda}{\lambda+\mu}$. Neste contexto, a disponibilidade do sistema é a probabilidade dele estar no estado operacional, ou seja, $A = \pi_U$

Uma das principais restrições da modelagem direta com CTMCs é a explosão combinatória do espaço de estados, que aumenta com a quantidade de componentes concorrentes, dificultando assim a construção da matriz Q e tornando sua solução computacionalmente inviável.

Formalismos de alto nível como Redes de Petri Estocásticas (SPNs) proporcionam uma especificação modular e concisa, fundamentada em lugares, tokens e transições. Muitas ferramentas realizam o mapeamento automático da SPN para a CTMC correspondente, elaboram Q e resolvem o modelo de forma numérica.

2.7.3 Redes de Petri Estocásticas - SPN

A avaliação de sistemas computacionais complexos, que englobam concorrência, sincronização e compartilhamento de recursos, requer a utilização de formalismos de modelagem com grande capacidade expressiva. As Redes de Petri – Petri Net (PN) – tradicionais são um dos formalismos, amplamente empregados para descrever e analisar o comportamento lógico de um sistema, mas sem uma concepção inerente de tempo. Para analisar métricas de desempenho (como vazão, tempo de resposta) e confiabilidade (como disponibilidade, confiabilidade), é essencial incluir a variável temporal no modelo (MELO, 2018).

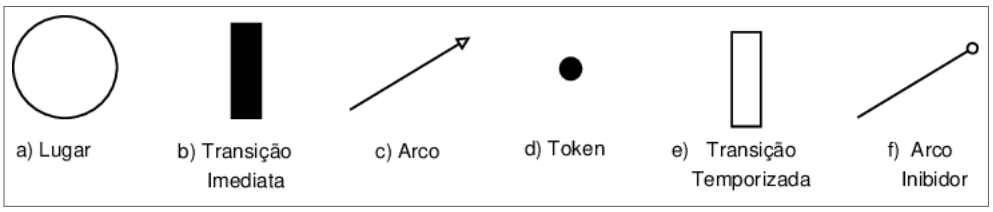
As SPNs surgem como uma extensão do formalismo clássico justamente para suprir essa necessidade, associando um tempo de disparo, que é uma variável aleatória, a cada transição da rede (MOLLOY, 1982). A escolha da distribuição de probabilidade para esses tempos é um aspecto fundamental. Nas SPNs, as durações das atividades são tipicamente modeladas pela distribuição exponencial, que possui a distinta propriedade de "falta de memória" (*memoryless property*). Essa abordagem implica que a probabilidade de um evento futuro ocorrer (como a falha de um componente) depende apenas do estado atual do sistema, e não do tempo que ele já permaneceu nesse estado.

A característica de ausência de memória é o que define formalmente um processo de Markov. Por essa razão, a evolução de uma SPN ao longo do tempo pode ser representada por uma CTMC. Para compreender essa ligação, é necessário estabelecer o espaço de estado do sistema, que representa o conjunto de todas as configurações possíveis que o sistema pode ter. Em uma Rede de Petri, cada configuração é representada por uma marcação, que indica a alocação de tokens nos lugares da rede. Ainda é necessário entender o conceito de gráfico de acessibilidade que é um grafo que inclui todos os estados (marcações) que o sistema pode

atingir a partir de um estado inicial, assim como as transições (disparos) que conectam um estado ao outro. Em razão da propriedade de Markov assegurada pela distribuição exponencial, esse diagrama de alcançabilidade é matematicamente equivalente a uma CTMC, onde os nós representam os estados e os arcos indicam as taxas de transição entre eles (MARSAN; CONTE; BALBO, 1984). A principal vantagem de conseguir uma CTMC equivalente é que existem soluções analíticas e numéricas estabelecidas para "solucioná-la".

Formalmente, uma SPN é representada por elementos distintos, como mostrado na Figura 8: lugares (círculos), que representam estados ou condições; tokens (pontos), que residem nos lugares e definem a marcação atual da rede; transições (retângulos), que denotam eventos ou ações; e arcos (setas), que conectam lugares a transições e vice-versa, definindo o fluxo de tokens. Arcos inibidores – terminados com um pequeno círculo – habilitam uma transição apenas se um lugar específico estiver vazio (CIARDO; GERMAN; LINDEMANN, 1994).

Figura 8 – Elementos de uma SPN



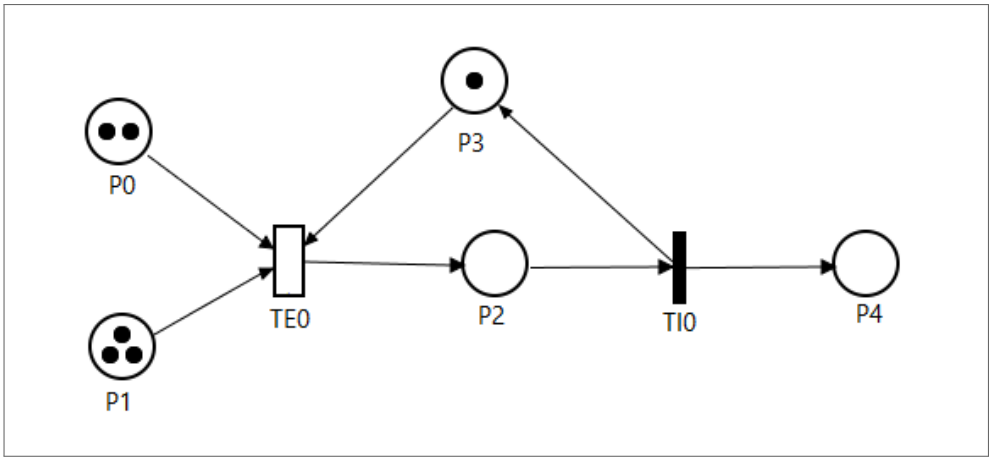
Fonte: (AUSTREGéSILO; CALLOU, 2019)

As SPNs são adequadas para modelar ambientes de nuvem, que são inerentemente concorrentes, assíncronos e estocásticos. O modelo proposto neste artigo utilizará lugares para rastrear o estado do sistema a partir de transições temporizadas para representar o processamento de trabalhos, enquanto transições imediatas capturam a lógica da política de escalonamento automático, modelando a dinâmica complexa do ambiente Moodle.

O comportamento dinâmico de umaSPN é determinado pelas regras de sua semântica de disparo. A Figura 9 mostra um modelo genérico de uma SPN. Uma transição é considerada habilitada quando todos os seus lugares de entrada contêm tokens suficientes e todos os seus lugares conectados por arcos inibidores estão vazios. Uma vez habilitada, a transição pode disparar, consumindo tokens de seus lugares de entrada e depositando tokens em seus lugares de saída, o que altera a marcação da rede para um novo estado. Se transições imediatas e temporizadas estiverem habilitadas simultaneamente, a transição imediata sempre terá prioridade (CLEMENTE, 2022).

Por fim, é necessário destacar que os arcos possuem pesos. Esses pesos indicam quantos

Figura 9 – Modelo Genérico em SPN



Fonte: Elaborada pelo autor (2025)

tokens são consumidos de um lugar ou quantos tokens são inseridos num lugar por uma transição ao disparar. Os pesos são fundamentais para definir a regra de habilitação de uma transição: ela só pode disparar se houver ao menos tantos tokens quanto o peso de cada arco de entrada exige. O peso padrão é 1, contudo, a depender o que se pretende modelar, podem ter valores diferentes ou expressões condicionais que os determinem.

2.8 ANÁLISE DE SENSIBILIDADE

A técnica de análise de sensibilidade é amplamente empregada na classificação para determinar quais fatores exercem maior influência sobre as métricas de um modelo (HAMBY, 1994). Um método direto e eficaz consiste em alterar cada parâmetro individualmente enquanto os demais permanecem constantes. Ao fazer isso, é possível obter uma classificação de sensibilidade ao observar as mudanças na saída do modelo.

Na análise de desempenho, a técnica da análise diferencial é frequentemente utilizada. Esta técnica envolve o cálculo das derivadas parciais das métricas relevantes com respeito a cada parâmetro. Por exemplo, considerando a métrica Y, que é função de um parâmetro λ, a sensibilidade de Y em relação a λ é determinada utilizando a Equação 2.13 ou 2.14, dependendo da escala de sensibilidade escolhida (FRANK, 1978).

$$S_\lambda(Y) = \frac{\partial Y}{\partial \lambda} \tag{2.13}$$

$$S^*_\lambda(Y) = \frac{\partial Y}{\partial \lambda}\left(\frac{\lambda}{Y}\right) \tag{2.14}$$

$S_\lambda(Y)$ e $S^*_\lambda(Y)$ também são chamados de coeficientes de sensibilidade (HAMBY, 1994). Os valores desses coeficientes, quando ordenados, produzem uma classificação que é usada para comparar o grau de influência entre todos os parâmetros.

2.9 INJEÇÃO DE FALHA

A injeção de falhas é uma técnica experimental para avaliação da dependabilidade de um sistema, que consiste em introduzir falhas de forma deliberada e controlada para observar seu comportamento e testar seus mecanismos de tolerância a falhas (LIMA, 2015). O objetivo é avaliar a robustez do sistema e medir métricas como cobertura de falhas, latência de detecção e tempo de recuperação (DANTAS et al., 2011).

Essa técnica é uma alternativa poderosa ao monitoramento passivo de sistemas, especialmente quando as falhas naturais são eventos raros, pois permite acelerar o processo de avaliação e testar cenários de falha específicos que podem não ocorrer naturalmente (CLEMENTE, 2022). A injeção de falhas pode ser realizada em diferentes níveis (LIMA, 2015):

- Nível de Hardware: Através da alteração de componentes ou exposição a ambientes que induzam erros.
- Nível de Software: Inserção de código que simula falhas de software, como corrupção de memória ou desligamentos inesperadas.
- Nível de Simulação/Modelo: Introdução de eventos de falha em modelos analíticos ou de simulação para avaliar o impacto no comportamento geral do sistema (MENDONçA et al., 2018).

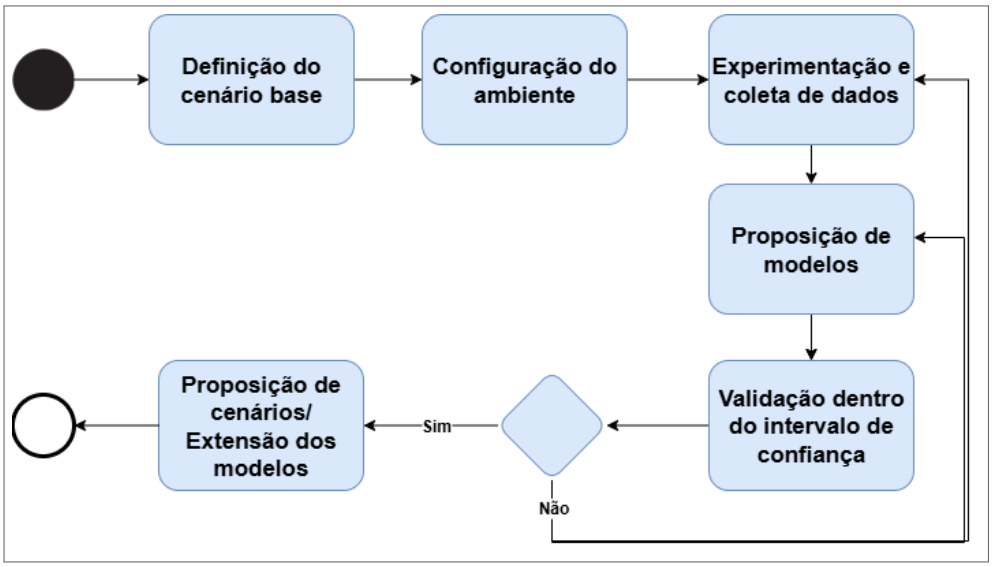
3 METODOLOGIA E ARQUITETURA

Este capítulo apresenta a metodologia e a arquitetura sugerida para avaliar a disponibilidade e desempenho de um LMS instalado sob uma pilha LAMP

3.1 METODOLOGIA

A metodologia sugerida abrange seis fases, cada uma estruturada da melhor forma para assegurar uma análise metódica do LMS implementado na plataforma LAMP. Primeiramente, é fundamental **definir o cenário de base** que servirá como ponto de referência ao longo da pesquisa. Este cenário engloba todas as variáveis e condições iniciais que serão levadas em conta durante o estudo (ARAUJO et al., 2013). Essa fase é essencial para assegurar que os resultados obtidos sejam significativos e aplicáveis ao contexto analisado (MACIEL et al., 2018). Posteriormente, procede-se à **configuração do ambiente** onde a pesquisa será conduzida. Isso envolve preparar todos os recursos, ferramentas e condições necessárias para a execução dos experimentos. Nesta etapa, um servidor é instanciado com a pilha LAMP, já descrita na Seção 2.2. A configuração correta do ambiente é crucial para garantir a validade e confiabilidade dos dados coletados. A Figura 10 apresenta o fluxograma utilizado para ilustrar a metodologia proposta.

Figura 10 – Metodologia Proposta



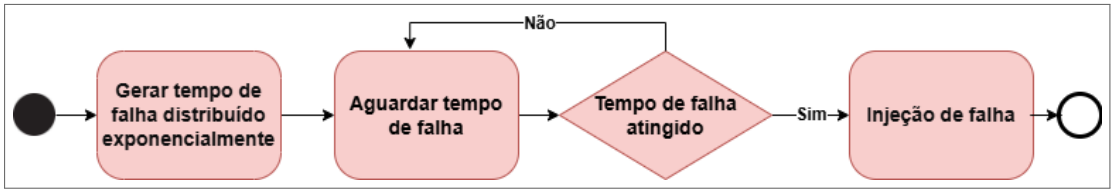
Fonte: Elaborada pelo autor (2025)

Com o ambiente devidamente configurado, inicia-se a fase de **experimentação e coleta**

de dados. Neste ponto, no servidor Moodle (Máquina/Servidor B), falhas são sistematicamente injetadas e reparadas em cada componente do sistema seguindo os códigos disponibilizados nos Apêndices A, B, C, D e E. Este processo é executado usando um script Python hospedado em um servidor secundário (Máquina/Servidor A). Os tempos para injeção e reparo de falhas seguem uma distribuição exponencial para simular imprevisibilidade dos eventos. Esta Máquina/Servidor A também é responsável por verificar a disponibilidade do serviço Moodle a cada cinco segundos, registrando essa informação em um arquivo. Nas etapas subsequentes, os dados deste arquivo de log serão usados para determinar o tempo médio para falha e o tempo médio para reparo real do sistema. Essas informações são importantes para calcular a disponibilidade do sistema.

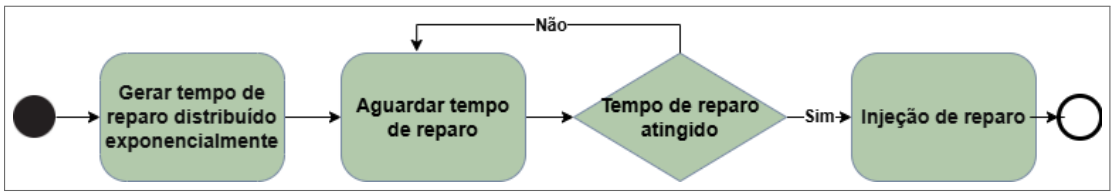
Durante a fase de "Experimentação e coleta de dados", falhas e reparos são introduzidos em cada parte do sistema (*hardware, software, Apache, MySQL e PHP*) em intervalos com tempo exponencial distribuído por meio de um script Python hospedado na Máquina/Servidor A – Figura 11 e Figura 12. O script gera tempo de falha distribuídos exponencialmente para o componente alvo e monitora se o tempo gerado foi alcançado. Caso seja atingido, uma falha será injetada no componente. Após a geração da falha, é necessário também reparar o componente dentro de um tempo distribuído exponencialmente. Assim, o script calcula esse tempo e aguarda até que ele seja atingido. Quando alcançado, o componente é reparado e um novo tempo de falha é determinado.

Figura 11 – Diagrama de Injeção de Falha



Fonte: Elaborada pelo autor (2025)

Figura 12 – Diagrama de Injeção de Reparo



Fonte: Elaborada pelo autor (2025)

Uma vez concluída a coleta de dados, o passo seguinte é **propor modelos** que consigam

representar adequadamente os comportamentos observados. Esses modelos permanecerão provisórios até passarem pelo processo de validação. Validar é uma fase crucial que assegura a precisão e robustez necessárias para que os modelos sejam aplicáveis de forma prática.

Na etapa de **validação**, é crucial verificar se os modelos refletem o cenário original com elevado grau de confiança. Para este propósito, comparamos os resultados do cenário original com os obtidos pelos modelos propostos. É desejável que os resultados dos modelos propostos caiam dentro de um intervalo de confiança de 95%.

Caso os modelos sejam considerados satisfatórios, o processo avança para a etapa de proposição de cenários e extensão dos modelos. Se não, é necessário verificar se houve algum problema na fase de experimentação de dados ou nos modelos propostos. Esta verificação assegura que apenas modelos confiáveis prossigam no processo.

Durante a fase de **proposição de cenários/ extensão dos modelos**, a partir dos modelos validados, são propostos novos cenários de de análise bem como a extensão dos modelos criados

É possível resumir as fases e resultados na Tabela 2:

Tabela 2 – Compilado do processo metodológico

Passo metodológico	Produto/Saída da etapa
Definição do cenário base	Estabelecimento de um ambiente inicial que reflete as condições reais ou ideais do estudo, servindo como referência para análises comparativas.
Configuração do ambiente	Implementação das condições e ferramentas necessárias para a realização do estudo, incluindo a instalação de softwares, definição de recursos físicos e ajustes nas variáveis do sistema.
Experimentação e coleta de dados	Injeção de falhas e reparos com tempo exponencialmente distribuídos nos componentes do servidor Moodle. Coleta do status do servidor (UP ou $DOWN$) e criação de arquivo de Log para posterior análise.
Proposição de modelos	Criação de modelos em RBD, CTMC e/ou SPN que reflita(m) o cenário base.
Validação dentro do intervalo de confiança	Análise dos dados adquiridos na etapa de experimentação e coleta de dados definindo métricas de interesse (como disponibilidade, indisponibilidade, tempo de inatividade anual, etc) comparando-os e validando-as com os dados da literatura e validando-os dentro do intervalo de confiança definido.
Proposição de cenários/Extensão dos modelos	A partir dos modelos previamente validados, serão delineados novos cenários de estudo, bem como promovidas ampliações e ajustes nos modelos existentes, com o objetivo de capturar de forma abrangente diferentes condições operacionais e variações do sistema analisado.

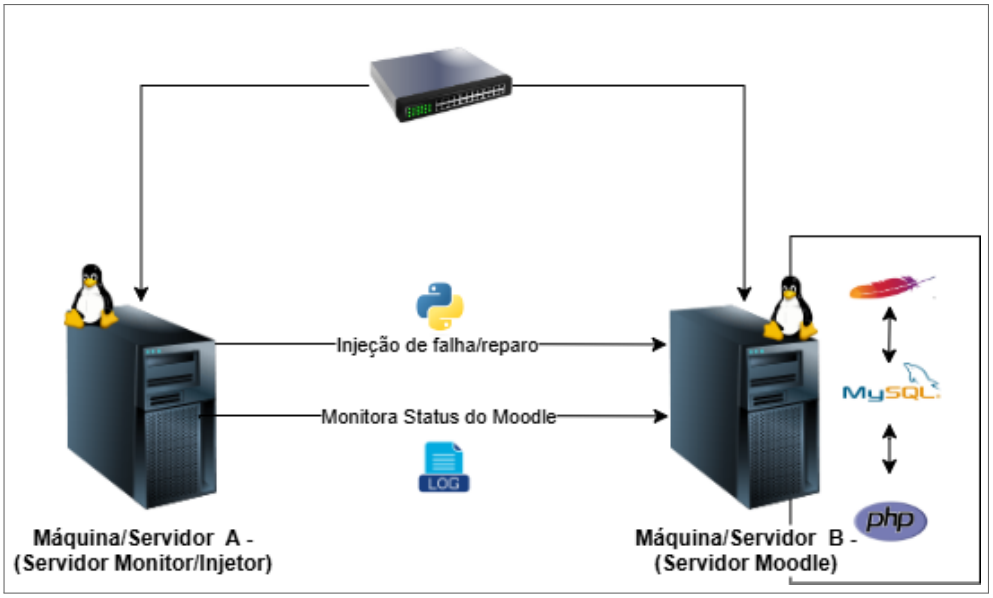
Fonte: Elaborada pelo autor (2025)

3.2 ARQUITETURA BÁSICA

Nesta subseção, é descrita a arquitetura básica empregada para a elaboração deste estudo. A Figura 13 ilustra a arquitetura proposta. Ela consiste em dois servidores conectados por meio de um *switch* que possibilita a comunicação entre as máquinas.

A Máquina/Servidor A é um Servidor Linux 22.04 LTS encarregado de conduzir injeções de falhas e reparos com tempos distribuídos exponencialmente nos componentes da Máquina/-Servidor B. Esses injetores são scripts Python e, além de sua execução, a Máquina/Servidor

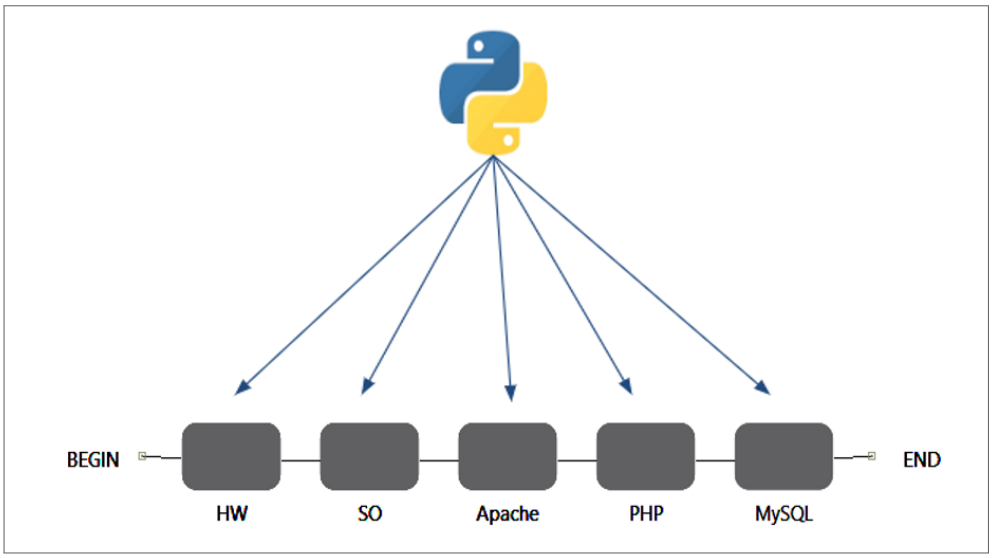
Figura 13 – Arquitetura proposta



Fonte: Elaborada pelo autor (2025)

A monitora o status (disponibilidade ou indisponibilidade) do Moodle que está hospedado na Máquina/Servidor B como mostrado na Figura 14

Figura 14 – Injetor de falhas/reparo



Fonte: Elaborada pelo autor (2025)

Os scripts de injeção de falhas têm a função de introduzir falhas e efetuar reparos nos componentes da Máquina/Servidor B, incluindo Hardware, Sistema Operacional, Apache, MySQL e PHP. A implementação de cinco scripts diferentes garante que cada operação de falha e reparo seja executada de forma autônoma e segura. Esses scripts estão disponíveis nos Apêndices A, B, C, D e E desta dissertação.

A Máquina/Servidor B, que é um Servidor Linux 22.04 LTS, atua como a estrutura principal do sistema, executando uma pilha LAMP juntamente com o Moodle LMS na versão 4.3. A operação ininterrupta dos componentes essenciais do sistema é crucial para sua disponibilidade: hardware, sistema operacional, Apache, MySQL e PHP devem estar sempre funcionais para que o Moodle esteja online. A falha em qualquer um deles resulta em uma indisponibilidade temporária do sistema, o que ressalta a importância de cada componente para a manutenção da funcionalidade global do sistema.

Neste ambiente Moodle, falhas e reparos são aplicados a cada parte do sistema em momentos que seguem uma distribuição exponencial, usando os scripts Python que executam na Máquina/Servidor A. Esta máquina também tem a função de verificar a disponibilidade do Moodle a cada cinco segundos e armazenar as informações em um arquivo de log ilustrado na Figura 13 através do código disponibilizado no Apêndice F. Quando todos os componentes estão operacionais, o servidor registra a data e hora e o status "UP" no log. Se algum componente estiver indisponível, ele registra a data e hora e o status "DOWN". Esses registros serão posteriormente utilizados para calcular o MTTF e o MTTR, dados cruciais para avaliar a disponibilidade do sistema.

O arquivo de log terá entradas como no exemplo abaixo:

```
2024-02-19 21:00:09:047 - System - UP
2024-02-19 21:00:14:813 - System - UP
2024-02-19 21:00:21:344 - System - UP
2024-02-19 21:00:27:107 - System - UP
2024-02-19 21:00:32:906 - System - UP
2024-02-19 21:00:38:696 - System - UP
2024-02-19 21:00:44:496 - System - UP
2024-02-19 21:00:50:309 - System - Down
2024-02-19 21:00:56:065 - System - Down
2024-02-19 21:01:01:846 - System - Down
2024-02-19 21:01:07:648 - System - Down
2024-02-19 21:01:13:466 - System - Down
2024-02-19 21:01:19:269 - System - Down
```

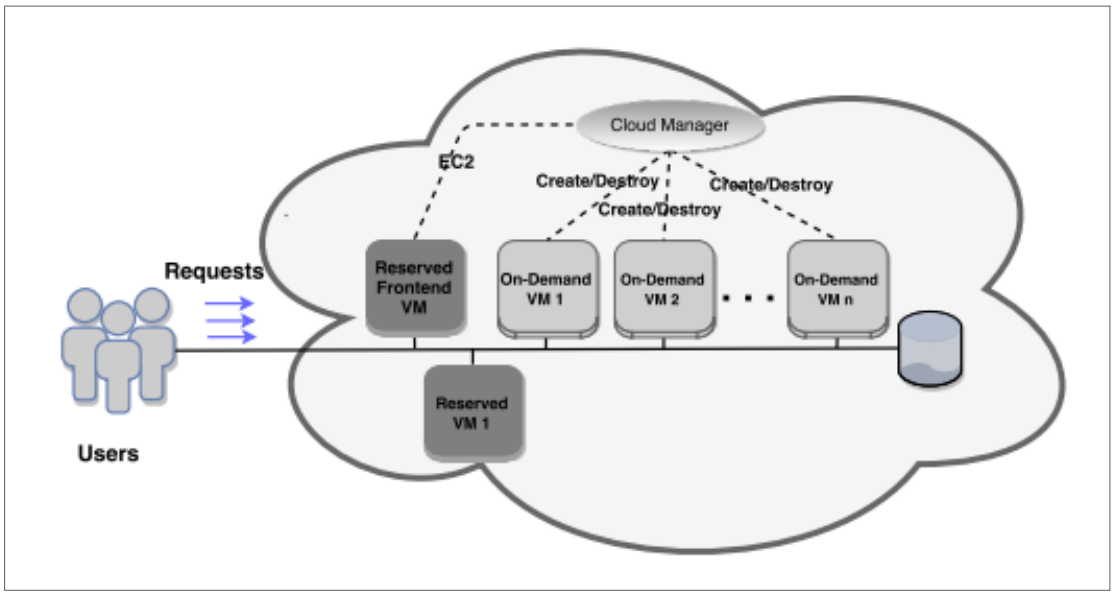
Esta abordagem permite rastrear por quanto tempo a Máquina/Servidor B esteve disponível ou indisponível, o que permite a aplicação dos modelos descritos no Capítulo 4.

3.3 ARQUITETURA BASEADA EM NUVEM PÚBLICA

Além da arquitetura básica, propomos analisar uma instalação do Moodle implantada em uma infraestrutura de nuvem pública, tomando por base o sistema AWS. Esta arquitetura foi concebida para ser flexível a fim de suportar tanto recursos fixos quanto variáveis que possam acomodar a carga de trabalho exigente de um ambiente educacional (ASSUNÇÃO et al., 2016). Escolhemos a IaaS que fornece controle de servidor e a migração do Moodle para a nuvem (BISWAS et al., 2015).

Os pedidos dos usuários são enviados para um *front-end* que, nesta arquitetura, é implementado pelo serviço *Elastic Load Balancing* da AWS. Este componente atua como um balanceador de carga, distribuindo as requisições uniformemente sobre um conjunto de VMs, cada qual executando um servidor web Apache para processar as solicitações do Moodle. Para alcançar seu desempenho máximo, a arquitetura emprega duas categorias de instâncias de VM: um número fixo de VMs reservadas e um número dinâmico de instâncias de VM sob demanda acionadas para lidar com picos de tráfego (BISWAS et al., 2015).

Figura 15 – Topologia de nuvem com escalonamento automático



Fonte: Adaptado de (FE et al., 2017)

A Figura 15 fornece uma visão geral da arquitetura de sistemas elásticos na nuvem. As instâncias reservadas são mostradas em blocos cinza escuro e as instâncias sob demanda em cinza claro. O administrador do sistema reserva um certo número de VMs, enquanto as VMs sob demanda mudam dinamicamente com base em parâmetros definidos pelo usuário

para auto-escalonamento (valores mínimos e máximos, tipo de instância e carga de trabalho observada).

No caso descrito, o *front-end* solicita a criação de novas VMs sob demanda por meio de chamadas *Elastic Compute Cloud (EC2)* ao gerenciador de nuvem, mas esse ajuste também poderia ser feito no painel da plataforma (HUNG; HU; LI, 2012). Isso afeta as métricas de desempenho do sistema, por exemplo, taxa de transferência, tempo médio de resposta ou utilização do sistema.

O pedido é enviado de maneira semelhante a uma fila. Se não houver VM disponível para lidar com novos pedidos eles serão enfileirados na fila de espera, onde aguardarão as VMs (tanto reservadas quanto sob demanda) estarem desocupadas. Uma vez que uma VM é finalizada, ela verifica a fila de VMs para o primeiro pedido a ser satisfeito (a regra "primeiro a entrar, primeiro a sair" ou *FIFO*). Este conceito de fila é projetado para evitar a perda de um pedido durante os picos de carga.

O sistema adota uma política de escalonamento automático reativo, que monitora constantemente a carga de trabalho da VM para orientar a decisão de escalonamento. Quando a quantidade de solicitações na fila atinge um limiar, uma ação de escalonamento é iniciada instanciando sob demanda novas VMs que atendem aos requisitos das solicitações recebidas. Da mesma forma, quando a carga de trabalho é baixa e o uso de recursos excede a necessidade, um limite de redução leva à destruição das VMs sob demanda, visando à redução de custos operacionais desnecessários.

A seleção de um tipo de VM é uma opção de configuração importante, pois tem um forte impacto no desempenho do sistema. Fornecedores de nuvem, como a AWS, dispõem inúmeras opções de VMs capacidades de processamento (*vCPU*), memória e variações de preço distintas, que precisam ser selecionados cuidadosamente à luz dos requisitos da aplicação.

Além disso, é crucial determinar quantas requisições cada instância da VM pode atender em sua configuração mínima, sem redundâncias ou configurações na máquina para aumentar seu desempenho como mecanismos de cache, por exemplo. Para tanto, recorreu-se à documentação oficial de instalação do Moodle (Moodle Community, 2024) a fim de saber a configuração mínima para uma instalação bem como estimar a capacidade de usuários atendidos. É crucial, também, entender que esses números são aproximações para usuários ativos simultaneamente com um perfil de uso moderado. A documentação Moodle recomenda como parâmetro mínimo de memória 1GB de RAM e como parâmetro ideal em um servidor de produção 8G.

A Tabela 3 ilustra exemplos de tipos de instância do Amazon EC2 com seus respectivos

recursos e a estimativa da quantidade de alunos únicos que podem ser atendidos. VMs com mais *vCPUs* e memória podem processar requisições mais rapidamente, mas implicam um custo maior.

Tabela 3 – Estimativa de capacidade de instâncias AWS para Moodle

Instância	**vCPUs**	**Memória (GiB)**	**Usuários Simultaneamente**
t3.small/t4g.small	2	2	5 – 15
t3.medium/t4g.medium	2	4	15 – 40
t3.large/t4g.large	2	8	40 – 80
t3.xlarge/t4g.xlarge	4	16	80 – 180
t3.2xlarge/t4g.2xlarge	8	32	180 – 400

Fonte: adaptada de (FE et al., 2017)),

A arquitetura detalhada nesta seção, com seus componentes essenciais como o balanceador de carga, o uso combinado de instâncias reservadas e sob demanda, e a política de escalonamento reativo, estabelece a base para a análise subsequente. A complexa interação entre a carga de trabalho variável, as capacidades de processamento das diferentes instâncias das VM e os limiares de escalonamento torna a avaliação puramente descritiva insuficiente para prever o comportamento do sistema sob diferentes condições. Portanto, para capturar essa dinâmica e avaliar quantitativamente as métricas de interesse, faz-se necessária a construção de um modelo formal.

4 MODELOS PROPOSTOS

Com base na arquitetura do LMS Moodle, detalhada no capítulo anterior, este capítulo dedica-se à concepção e formalização dos modelos propostos para a análise comparativa de seu desempenho e disponibilidade em dois cenários de implantação distintos: uma implementação local, em infraestrutura física própria, e uma implementação em nuvem pública, utilizando os serviços da AWS. A utilização de modelos matemáticos permite realizar uma avaliação do comportamento do sistema em ambos os contextos, viabilizando a tomada de decisão pelos administradores do sistema.

Tendo em vista a complexidade do sistema, que envolve tanto a confiabilidade estrutural dos componentes quanto a dinâmica de carga de trabalho e provisionamento de recursos, optou-se por uma estratégia de modelagem hierárquica. Esta abordagem permite decompor a o sistema, modelando diferentes aspectos com o formalismo mais adequado e, posteriormente, integrando os submodelos para uma análise completa.

Nesta abordagem hierárquica, cada formalismo é empregado para capturar aspectos distintos do sistema. O RBD é utilizado para a análise da disponibilidade estrutural, oferecendo uma representação intuitiva de como os componentes se interligam em série e paralelo e como suas falhas individuais impactam a funcionalidade do sistema como um todo. Para modelar o comportamento dinâmico de falha e reparo de componentes individuais ou subsistemas, as CTMCs são aplicadas, permitindo o cálculo de métricas como o MTTF e MTTR. Por fim, para analisar os aspectos de desempenho e a dinâmica complexa de carga de trabalho, filas e alocação de recursos, especialmente os mecanismos de escalonamento automático no cenário em nuvem, as SPNs são o formalismo mais adequado devido à sua capacidade de modelar concorrência, sincronização e contenção por recursos.

No contexto da modelagem hierárquica, os RBDs são empregados no primeiro nível para analisar o sistema a partir de suas dependências funcionais mais simples, como componentes em série. O resultado dessa análise é a sintetização de um sistema em um único bloco com a determinação de valores MTTF, MTTR, Disponibilidade, Uptime e Downtime, que servirão de insumo para os modelos de espaço de estado subsequentes.

A evolução natural do processo de modelagem consiste na utilização de CTMC, uma vez que este formalismo permite mapear de forma explícita as dependências entre os componentes do sistema, superando as limitações do RBD, que assume a independência estrutural entre os

blocos. Dessa forma, é possível capturar de modo mais realista, por exemplo, o impacto do tempo de falha e de reparo de cada componente sobre as métricas de desempenho analisadas, conferindo maior aderência às condições operacionais reais.

Pensamento semelhante se aplica à SPN, uma vez que o mapeamento usando RBD não captura a dinâmica complexa da arquitetura Moodle na AWS, que inclui o enfileiramento de requisições e as políticas de auto escalonamento faz-se necessário o uso de uma abordagem mais robusta.

4.1 ARQUITETURA BÁSICA

Nesta seção apresentaremos os modelos básicos da arquitetura Moodle de maneira que servirão para o entendimento da plataforma bem como base para validação e expansão para modelos mais complexos.

4.1.1 Modelo Básico em RBD

A partir da arquitetura delineada na Seção 3.2, foi desenvolvido uma representação RBD para os componentes do sistema. O modelo consiste em cinco blocos dispostos em série, conforme descrito a seguir:

- HW (Hardware): Refere-se ao equipamento físico necessário para executar o Software (SO) e serviços necessários para o Moodle;
- SO (Sistema Operacional): Software base que gerencia o hardware e oferece serviços essenciais para os outros componentes;
- Apache: Servidor web que manipula solicitações HTTP e entrega conteúdo aos usuários;
- PHP: Linguagem script do lado do servidor usada para executar e redenrizar páginas dinâmicas do Moodle;
- MySQL: Sistema de gerenciamento de banco de dados relacional que armazena e gerencia dados do Moodle.

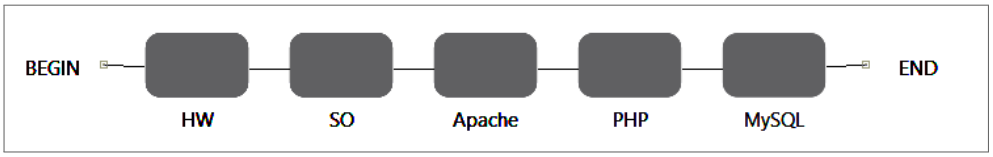
RBDs são comumente usados para representar e analisar a disponibilidade de sistemas. Em um RBD, o sistema é decomposto em componentes individuais, cada um representado por um

bloco, e as conexões entre os blocos indicam a dependência funcional desses componentes. O arranjo desses blocos, que podem ser configurados em série, paralelo ou uma combinação de ambos, ajuda a avaliar a confiabilidade geral do sistema. A simplicidade visual dos RBDs auxilia na compreensão e comunicação da estrutura do sistema, facilitando a identificação de componentes críticos e potenciais pontos de falha.

Em arquiteturas onde é essencial que todos os componentes funcionem corretamente para que o sistema tenha êxito, uma configuração em série é ideal. Aqui, o mau funcionamento de apenas um componente provoca a falha do sistema inteiro. Este tipo de configuração é representativo de cenários onde a continuidade operacional depende de cada elemento na cadeia. Assim, ela ilustra situações onde o sucesso do sistema depende do desempenho eficaz de cada componente, delineando a importância de assegurar a operação confiável de cada parte.

Neste estudo, é assumido que o sistema só estará operacional quando todos os seus componentes estiverem ativos ao mesmo tempo. Consequentemente, uma representação em série dos componentes é exigida, como demonstrado na Figura 16.

Figura 16 – RBD Proposto



Fonte: Elaborada pelo autor (2025)

A disposição em série ressalta a dependência vital que cada elemento tem na estrutura do Moodle. Uma falha no Hardware (HW) impede, por exemplo, o funcionamento do Sistema Operacional (SO). Se o Apache não estiver operando, as requisições HTTP não serão processadas. A falha do PHP impede que o Moodle execute sua lógica e, sem o MySQL, o Moodle não se consegue acessar ou armazenar informações. Por isso, a integridade de cada componente é essencial para a operação total do sistema.

Em continuidade, adota-se uma representação mais abstrata do modelo, consolidando os três últimos blocos (Apache, PHP e MySQL) em um único elemento representativo da aplicação Moodle chamado APP, como ilustrado na Figura 17. Essa simplificação se justifica pela necessidade de reduzir a complexidade do diagrama, mantendo, entretanto, a essência da confiabilidade do sistema. Tal abordagem permite analisar o comportamento do Moodle como uma unidade funcional integrada, sem perda significativa de precisão no contexto das métricas

de disponibilidade e confiabilidade, pois os componentes internos da aplicação podem ser considerados interdependentes e de similar criticidade operacional. Assim, a análise permanece focada na avaliação de falhas em níveis mais amplos, atendendo aos objetivos deste estudo.

Figura 17 – RBD Moodle

Fonte: Elaborada pelo autor (2025)

A configuração apresentada representa o mínimo necessário (cenário base) para o funcionamento do Moodle. Cada componente representado no RBD é essencial, e qualquer falha em um desses componentes resulta em indisponibilidade do sistema. Garantir a operação adequada de cada componente é vital para manter a disponibilidade e a confiabilidade do sistema.

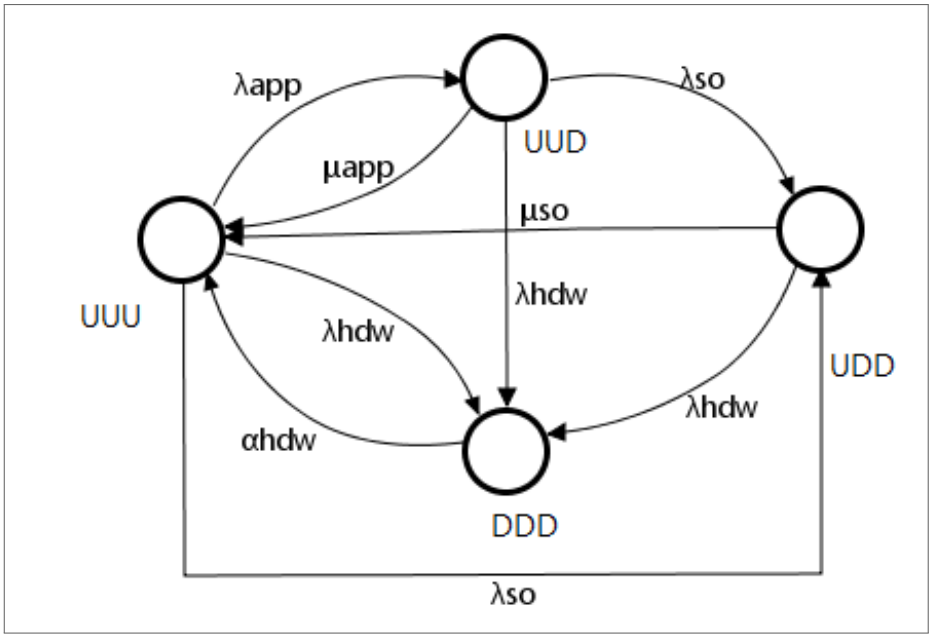
4.1.2 Modelo Básico Virtualizado em CTMC

As CTMCs modelam sistemas onde o comportamento estocástico e as transições de estado ocorrem ao longo do tempo. No formalismo CTMC, o Moodle é representado por um conjunto de estados e taxas de transição entre esses estados, que são distribuídos exponencialmente. As CTMCs permitem uma análise detalhada de processos dinâmicos, como falhas e reparos de componentes, fornecendo uma visão precisa do comportamento temporal do sistema e permitindo cálculos rigorosos de métricas de desempenho, como disponibilidade.

CTMCs são particularmente adequados para sistemas onde a dinâmica temporal e o comportamento estocástico são críticos. Por exemplo, em sistemas de manutenção e reparo, as CTMCs podem modelar eventos de falha, processos de recuperação e suas respectivas durações. Essa abordagem fornece uma compreensão detalhada de como os tempos de falha e reparo influenciam a disponibilidade do sistema. As Cadeias de Markov de Tempo Contínuo oferecem previsões precisas e *insights* sobre o desempenho do sistema, capturando a natureza probabilística das transições de estado.

Para garantir a alta confiabilidade das métricas calculadas, um modelo baseado em CTMC foi desenvolvido a partir do RBD da Figura 17. A Figura 18 ilustra este modelo, mostrando os componentes do sistema e suas interconexões.

Figura 18 – CTMC Moodle



Fonte: Elaborada pelo autor (2025)

Neste formalismo, a pilha LAMP é tratada como um único componente, incorporando os parâmetros de falha e reparo de cada elemento. O status dos componentes define cada estado, conforme mostrado na Tabela 4. A combinação desses componentes unificados é chamada de APP.

Tabela 4 – CTMC – Descrição dos estados

Estado	Descrição	Sistema
UUU	Hardware, SO e APP disponíveis	Disponível
UUD	Hardware e SO disponíveis; APP indisponível	Indisponível
UDD	Hardware disponível; SO e APP indisponíveis	Indisponível
DDD	Hardware, SO e APP indisponíveis	Indisponível

Fonte: Elaborada pelo autor (2025)

No modelo visualizado na Figura 18, λ representa a taxa de falha e μ a taxa de reparo dos componentes do sistema. Suas descrições são apresentados na Tabela 5. Os estados seguem uma sequência em que U representa o estado funcional (UP) e D representa um componente em falha (DOWN), respectivametne para *Hardware*, *SO* e *APP*.

A probabilidade do sistema estar funcionalmente disponível, denotada por $A_s = \pi_{(UUU)}$, corresponde à probabilidade de estado estacionário estar no estado UUU. Neste estado, o

Tabela 5 – CTMC – Descrição das taxas

Taxa	Descrição
λ_{hdw}	Taxa de falha da VM
λ_{os}	Taxa de falha do software
λ_{app}	Taxa de falha da aplicação
α_{hdw}	Taxa de instanciação da VM
μ_{os}	Taxa de reboot do software
μ_{app}	Taxa de reparo da aplicação

Fonte: Elaborada pelo autor (2025)

sistema está operacional. Caso um componente LAMP falhe a uma taxa de λ_{app}, o sistema se desloca para o estado UUD. Deste estado, ele pode ser reparado a uma taxa de μ_{app}, retornando ao estado UUU. Estando no estado UUD pode ocorrer uma falha no sistema operacional com uma taxa de λ_{os} levando o sistema para o estado UDD, podendo ser reparado a uma taxa de μ_{os}, Isso leva o sistema ao estado inicial UUU. Ainda no estado UUU pode ocorrer uma falha no sistema operacional com uma taxa λ_{os}. Nesse caso o sistema migra para o estado UDD. Falhas de hardware ocorrem a uma taxa de λ_{hdw} a partir estados UUD, UDD ou UUU transferindo o sistema para o estado DDD, que só pode ser recuperado por meio de reparo de hardware a uma taxa μ_{hdw}.

O modelo de disponibilidade do sistema pode ser avaliado usando a fórmula fechada 4.1, concebida usando a ferramenta Mathematica em conjunto com a ferramenta Mercury (SILVA et al., 2015).

$$A = \frac{(\lambda_{\text{hdw}} + \lambda_{\text{os}} + \mu_{\text{app}})\,\alpha_{\text{hdw}}\,(\lambda_{\text{hdw}} + \mu_{\text{os}})}{(\lambda_{\text{app}} + \lambda_{\text{hdw}} + \lambda_{\text{os}} + \mu_{\text{app}})\,(\lambda_{\text{hdw}} + \alpha_{\text{hdw}})\,(\lambda_{\text{hdw}} + \lambda_{\text{os}} + \mu_{\text{os}})} \tag{4.1}$$

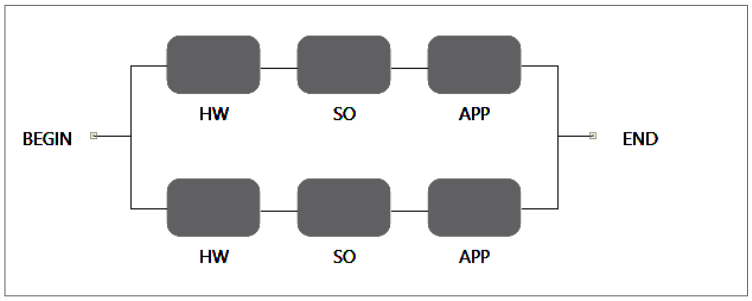
O modelo baseado em CTMC permite modelagem precisa de taxas de falha e reparo, capturando transições entre estados operacionais e não operacionais ao longo do tempo. Toda a pilha LAMP é tratada como um único componente no modelo CTMC mostrado na Figura 18. Essa abordagem simplifica o gerenciamento de componentes do sistema, especialmente caso se queira no futuro usar VM ou contêineres. Quando o sistema entra no estado DDD, uma nova VM ou contêiner pode ser instanciado, retornando o sistema ao estado UUU sem a necessidade de reparos de componentes individuais melhorando, *a priori*, a disponibilidade do sistema.

4.2 ARQUITETURA REDUNDANTE

Uma vez estabelecido o comportamento do sistema em sua arquitetura básica, a etapa subsequente da análise consiste em explorar estratégias de aprimoramento da disponibilidade e do desempenho do Moodle. A introdução de redundância se apresenta como a técnica útil para mitigar pontos de falha, permitindo ampliar a tolerância do sistema e, por consequência, aumentar sua resiliência frente a falhas inesperadas de componentes críticos.

Considerando a arquitetura proposta para o Moodle na Figura 17 e considerando que todos os componentes são vitais para garantir a continuidade do serviço, eventuais falhas em qualquer um destes elementos comprometem de maneira imediata a operacionalidade da plataforma. Visando eliminar tais pontos únicos de falha, optou-se por adotar uma estratégia de redundância completa, replicando todos os componentes do sistema em caminhos paralelos de operação como mostrado na Figura 19

Figura 19 – RBD Moodle Redundante



Fonte: Elaborada pelo autor (2025)

A representação formal dessa abordagem foi construída por meio de um RBD, no qual cada cadeia funcional completa (HW, SO e APP) é duplicada. Esses caminhos redundantes, conectados em paralelo no modelo, permitem que o sistema continue operacional mesmo que ocorra a falha de uma das cadeias inteiras. Assim, o RBD explicita que basta ao menos uma cadeia funcional estar íntegra para assegurar o fornecimento do serviço, refletindo o princípio de tolerância a falhas aplicado ao ambiente Moodle.

Essa estratégia de redundância completa potencializa a disponibilidade do sistema ao contemplar falhas em múltiplos pontos da arquitetura. Diferentemente de modelos que privilegiam apenas a redundância na camada de aplicação, a abordagem aqui empregada amplia a proteção para o nível de infraestrutura (hardware) e o nível de software de base (sistema operacional), garantindo uma cobertura mais abrangente e robusta contra falhas que possam interromper a

prestação do serviço.

4.3 ARQUITETURA VIRTUALIZADA

Após a proposição do modelo de redundância, esta seção abordará a modelagem do ambiente Moodle quando implantado em uma arquitetura virtualizada. A virtualização representa um paradigma fundamental na computação moderna, permitindo a abstração dos recursos de hardware de um servidor físico para criar e gerenciar múltiplas VMs isoladas, cada uma com seu próprio sistema operacional e aplicações.

A introdução da camada de virtualização gerenciada pelo hypervisor cria novas interdependências que impactam diretamente a disponibilidade do serviço final. Em tal ambiente, a disponibilidade do Moodle não depende apenas do software da aplicação em si, mas de toda a pilha de componentes subjacentes: a VM, o hypervisor e o hardware do servidor físico. A falha em qualquer um desses níveis resulta na indisponibilidade do serviço, caracterizando uma dependência em série entre as camadas. Para analisar corretamente o sistema, é imprescindível modelar essa dependência hierárquica.

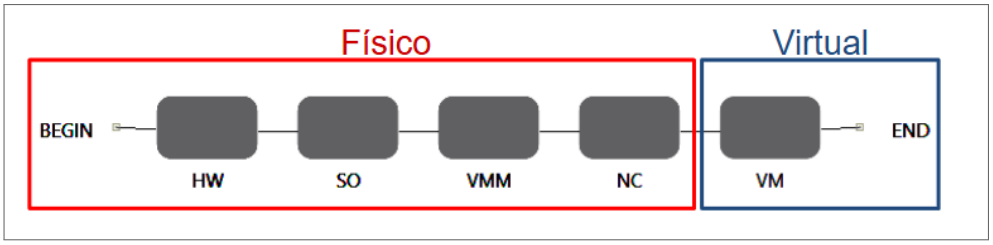
Para capturar tanto a dependência estrutural quanto o comportamento dinâmico do sistema, esta seção adotará uma estratégia de modelagem em dois níveis. Inicialmente, um RBD será utilizado para representar a pilha de virtualização de um único servidor, possibilitando o cálculo de sua disponibilidade e das taxas de falha e reparo equivalentes. Em seguida, uma SPN será empregada para modelar o comportamento do sistema completo, que pode ser composto por múltiplos servidores virtualizados em configurações redundantes, utilizando os parâmetros obtidos na análise do RBD, bem como por um pool de VMs.

4.3.1 Modelo Virtualizado em RBD

A primeira etapa na modelagem da arquitetura virtualizada consiste em analisar a disponibilidade de um único servidor que hospeda o ambiente Moodle. Para isso, partimos de um RBD, conforme ilustrado na Figura 20. Este modelo em série representa a dependência entre as camadas de hardware e software que compõem um servidor virtual. A lógica, mais uma vez, é que a disponibilidade do Moodle depende do funcionamento simultâneo de todas os componentes do sistema.

O modelo é constituído por cinco blocos em série, cada um representando uma camada

Figura 20 – RBD Moodle Virtualizado



Fonte: Elaborada pelo autor (2025).

da pilha de virtualização e aplicação, quais sejam:

- HW (Hardware do host): Representa os componentes físicos do servidor hospedeiro. Este bloco engloba a falha de qualquer elemento de hardware, como CPU, memória RAM, discos de armazenamento, fontes de alimentação ou placas de rede.
- SO (Sistema Operacional do host): Representa o sistema operacional principal que é executado diretamente sobre o hardware. É a camada que serve de base para a execução do software de virtualização.
- VMM (Virtual Machine Monitor): Representa o software de virtualização, ou hypervisor. Esta é a camada responsável por criar, gerenciar e alocar os recursos de hardware para as máquinas virtuais.
- NC (Node Controller): Representa o serviço de software de gerenciamento do nó. Este componente é responsável por controlar o ciclo de vida das VMs no hospedeiro, respondendo a comandos de um controlador de nuvem de nível superior (por exemplo, para iniciar ou parar VMs). Sua falha impede a correta administração das VMs no host.
- VM (Virtual Machine): Componente que representa uma instância completa do Moodle em uma máquina virtual, modelada por meio da CTMC apresentada na Figura 18.

Dado que os cinco blocos estão em uma configuração em série, a falha de qualquer um deles leva à indisponibilidade do Moodle. A disponibilidade A do sistema hospedado neste servidor virtualizado é dado pelo produto da disponibilidade individual de cada um de seus componentes, isto é, $A = A_{HW} \times A_{SO} \times A_{VMM} \times A_{NC} \times A_{VM}$

Este modelo detalhado permite, por exemplo, uma análise de sensibilidade precisa, possibilitando a identificação do componente que mais contribui para a indisponibilidade do sistema.

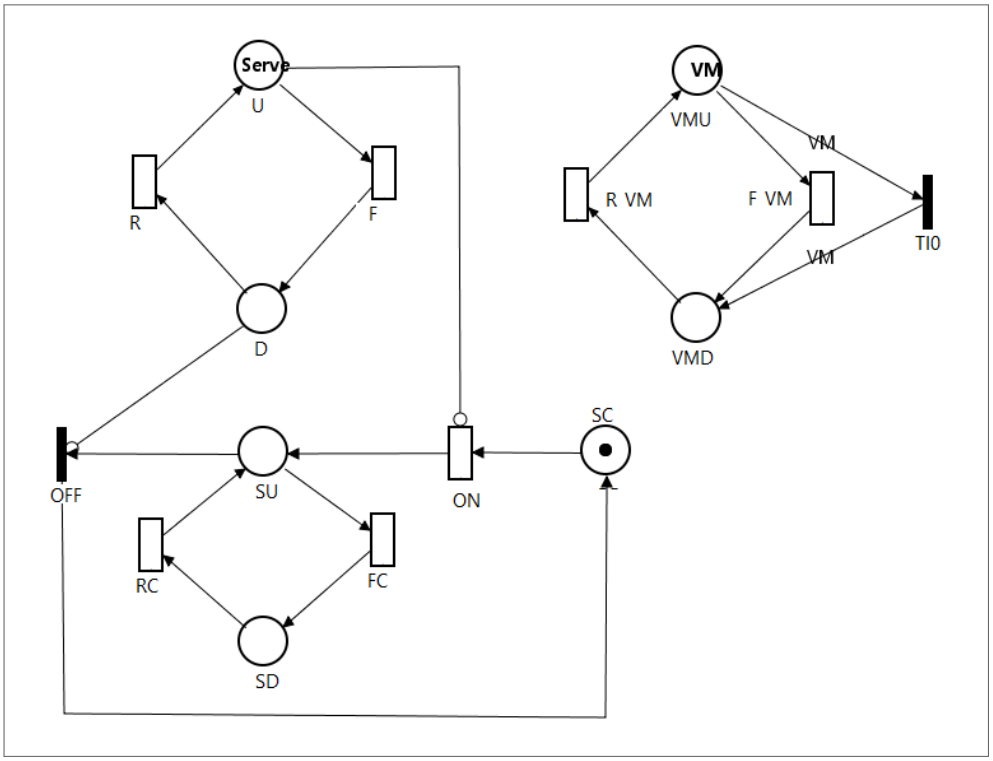
As métricas deste modelo ($MTTF$ e $MTTR$) servirão como parâmetros para o modelo SPN de nível superior.

4.3.2 Modelo Virtualizado em SPN com redundância

O próximo modelo estendido é uma SPN baseada em virtualização desenvolvido para capturar o comportamento dinâmico do Moodle adicionando um servidor redundante no qual um componente secundário (ou de reserva) permanece em um estado inativo ou desligado enquanto o servidor primário está operacional. O servidor reserva só é ativado após a detecção da falha do primário.

A Figura 21 ilustra o modelo proposto para representar uma arquitetura com redundância cold standby (ativo/passivo), mecanismo no qual um componente primário (U) está em operação enquanto um componente secundário (SC) permanece em um estado inativo, sendo ativado somente após a falha do primário. A complexidade da lógica de detecção de falha, chaveamento (*failover*) e retorno ao estado original torna a SPN um formalismo ideal para capturar com precisão o comportamento dinâmico e as diferentes fases de disponibilidade e indisponibilidade do sistema.

Figura 21 – SPN Moodle Virtualizado com redundância no servidor



Fonte: Elaborada pelo autor (2025)

O modelo é composto por dois submodelos principais: um para o componente primário (composto pelos lugares U (servidor principal ativo) e D (servidor principal inativo) e outro para o secundário composto por SU (servidor secundário ativo) e SD (servidor secundário inativo).

O componente primário possui um comportamento simples de falha (transição F) e reparo (transição R), com $MTTF$ associado à transição F e $MTTR$ à R. De maneira semelhante há tempos de falha e reparo associados às transições FS e RS. O arco inibidor da transição ON possibilita a habilitação do lugar SC quando o lugar U estiver vazio.

Nesse modelo o cálculo de disponibilidade é dado pela probabilidade do sistema estar no estado U ou no estado SU.

O subsistema que representa as VMs, por sua vez, é composto pelos lugares VMU (VM ativa) e VMD (VM inativa). Semelhante ao modelo do servidor, existe duas transições temporizadas que são responsáveis por retirar e inserir tokens do lugar VMU representando a falha – F_{VM} – e o reparo das VMs – R_{VM}.

Adicionalmente, existe uma transição imediata nomeada como $TI0$ que possui uma expressão de guarda para sua ativação. Para um servidor no lugar $Server$, a transição será ativada quando a expressão de guarda $((\#D > 0)\text{AND}(\#SD > 0))\text{OR}((\#D > 0)\text{AND}(\#SU = 0)\text{AND}(\#SD = 0))$ ou $((\#U = 0)\text{AND}(\#SU = 0))$ for verdadeira.

Os arcos que chegam e saem da transição $TI0$ possuem peso padrão modificado para VM de maneira que quando a transação é disparada todos os tokens do lugar VMU são retirados sendo colocados no lugar VMD.

O objetivo deste modelo é analisar a disponibilidade do sistema, considerando não apenas as falhas individuais das máquinas virtuais, mas, crucialmente, a sua dependência da infraestrutura física subjacente.

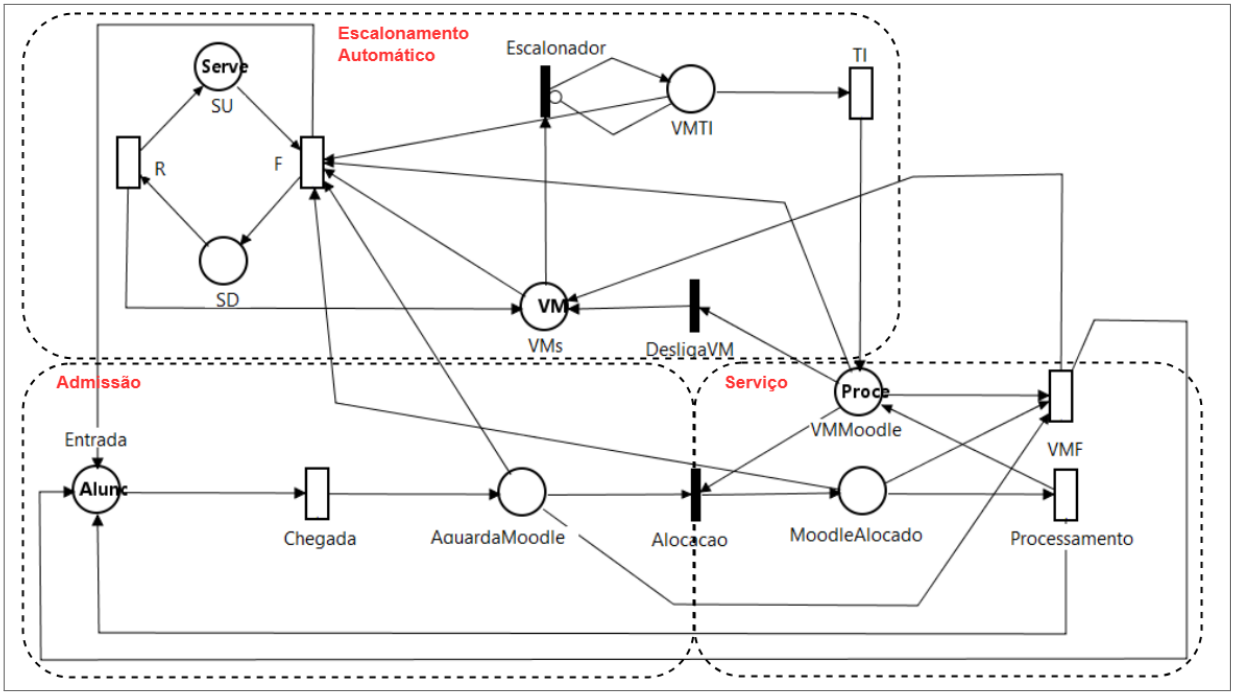
Este modelo abrangente permite uma análise completa do sistema virtualizado redundante. A disponibilidade do serviço é definida pela condição de que pelo menos um dos hosts esteja operacional e haja pelo menos uma VM disponível para processar as requisições nesse host. A disponibilidade total é, portanto, a soma das probabilidades de estado estacionário de todas as marcações que satisfazem essa condição.

4.4 MODELO DE DESEMPENHO

Após a construção e análise dos modelos focados nas arquiteturas básicas, redundantes e virtualizadas, esta seção final do capítulo apresenta um modelo de desempenho. Enquanto os modelos anteriores foram essenciais para quantificar a probabilidade de o sistema estar operacional, métricas como utilização, probabilidade de descarte e vazão podem ser calculadas por meio de um modelo que mapeia o ciclo de vida das requisições ao sistema.

Para este fim, foi desenvolvido o modelo em SPN mostrado na Figura 22 que integra os elementos da arquitetura virtualizada com foco na análise de desempenho. O modelo detalhado a seguir irá representar, portanto, o percurso completo de uma requisição do usuário, desde sua chegada ao sistema, passando pela lógica do balanceador de carga, pela potencial espera em uma fila, pelo processamento em uma das VMs disponíveis até a conclusão do serviço. A análise deste modelo permitirá a avaliação quantitativa de indicadores chave de desempenho, oferecendo uma visão do comportamento do sistema sob diferentes cargas de trabalho e validando a eficácia da arquitetura elástica estudada.

Figura 22 – SPN de Desempenho Moodle



Fonte: Elaborada pelo autor (2025)

O modelo proposto oferece uma representação detalhada da dinâmica do Moodle em nuvem, focando na interação entre a chegada de usuários, a alocação das VMs e as políticas

de escalonamento automático que ajustam a quantidade de Máquinas Virtuais conforme a demanda.

O modelo é dividido em três sub-redes: **Admissão**, que modela a chegada de trabalhos; **Serviço**, responsável pela alocação e processamento dos trabalhos no Moodle; e **Escaloamento automático**, que modela a adição e a remoção de recursos computacionais.

A sub-rede de admissão é composta por dois lugares – **Entrada** e **AguardaMoodle** – que representam a quantidade de trabalhos ou requisições que podem ser realizadas e a espera desses trabalhos na fila do sistema. A transição **Chegada** contém a variável com o tempo de chegada entre requisições. Uma requisição chegando em **AguardaMoodle** e havendo VM e recursos de processamento disponíveis a transição imediata **Alocação** é disparada e o trabalho passa para a próxima sub-rede.

A sub-rede de **Serviço** é composta por dois lugares que gerenciam o ciclo de vida do processamento: **MoodleAlocado** e **VMMoodle**. O lugar **MoodleAlocado** representa os trabalhos que estão sendo ativamente processados; **VMMoodle**, por sua vez, modela a capacidade de processamento disponível em uma instância de VM. A transição temporizada **Processamento**, cuja taxa segue uma distribuição exponencial, modela o tempo de processamento dos trabalhos. Ao ser concluído, o recurso da VM é liberado (uma marca retorna a **VMMoodle**) e o aluno (token que estava em **MoodleAlocado**)) retorna ao lugar entrada de maneira que realizar uma requisição (acesso) ao sistema. O disparo da transição **VMF** causa a falha de um VM instanciada e, através do arco à VM que falhou.

Por fim, a sub-rede de **Escalonamento automático** contém a lógica de instanciação e desligamento de VMs, bem como o funcionamento do host. Os lugares **SU** e **SD** representam o servidor online e offline, respectivamente. O servidor pode falhar através da transição **F** e se recuperar através da **R**. O lugar **VMs** contém o pool de máquinas virtuais disponíveis. O lugar **VMTI** representa uma VM aguardando para ser instanciada, o que ocorre através do disparo da transição **TI**.

Os lugares principais são parametrizados com variáveis que definem a configuração inicial da SPN como mostrado na Tabela 6. No lugar **SU** há a variável *Server*, indicando o número de servidores disponíveis. No lugar **VMs** há a variável *VM*, que indica a quantidade inicial de máquinas virtuais no pool. O lugar **Entrada** é configurado com a variável *Alunos*, indicando a quantidade de estudantes que utilizam o sistema. Finalmente, o lugar **VMMoodle** possui a variável *ProcessamentoVM*, que representa a capacidade de processamento da instância da VM utilizada (conforme Tabela 3).

Tabela 6 – Descrição dos lugares do modelo SPN

Lugar	Descrição	Marcação inicial
SU	Disponibilidade do servidor	Server
SD	Indisponibilidade do servidor	-
VMs	Pool de VMs	VM
VMTI	VM a ser instanciada	-
VMMoodle	VM Moodle disponível	ProcessamentoVM
Entrada	Entrada das requisições	Alunos
AguardaMoodle	Fila aguardando alocação de recurso para trabalho	-
MoodleAlocado	Um recurso (processamentoVM) foi alocado a um trabalho	-

Fonte: Elaborada pelo autor (2025)

A Tabela 7 apresenta a descrição das variáveis do modelo. A variação desses parâmetros será utilizada na análise de estudos de caso para avaliar o desempenho do Moodle sob diferentes condições.

Tabela 7 – Descrição das variáveis do modelo

Definição	Descrição
Server	Quantidade de servidores disponíveis
VM	Pool de VMs
ProcessamentoVM	Capacidade de processamento de cada VM (Alocado no lugar "VMMoodle")
Alunos	Quantidade de usuários do sistema (alocado no lugar "Entrada")
MTTF	Tempo médio de falha do servidor
MTTR	Tempo médio de reparo do servidor
MTBA	Tempo médio de chegada entre requisições
MTBI	Tempo médio para instanciação de uma VM
MTTF_VM	Tempo médio de falha da VM
MTBP	Tempo médio para processamento de um trabalho pelo Moodle

Fonte: Elaborada pelo autor (2025)

A dinâmica do modelo SPN é governada por suas transições temporizadas ou imediatas. Cada uma dessas das transições temporizadas é associada a uma variável que define seu comportamento temporal o que torna o modelo flexível e parametrizável. A Tabela 8 apresenta

o mapeamento usado nas transições às suas variáveis correspondentes.

Tabela 8 – Descrição das transições do modelo SPN

Transição	Tipo	Descrição	Guarda	Definição associada	Prioridade
R	Temporizada	Reparo do servidor	-	MTTR	-
F	Temporizada	Falha do servidor	-	MTTF	-
Chegada	Temporizada	Chegada de requisições	`(VMMoodle>0) OR (VMAlocada>0)`	MTBA	-
TI	Temporizada	Instanciamento de uma nova VM	`(U>0) AND (VMMoodle <= (ProcessamentoVM*20)/100)`	MTBI	2
VMF	Temporizada	Falha da VM instanciada	`(VMMoodle>0) OR (VMAlocada>0)`	MTTF_VM	-
Processamento	Temporizada	Processamento de requisições pelo Moodle	-	MTBP	-
Escalonar	Imediata	Escalonamento de uma nova VM quando a alocada estiver com menos de 20% de recursos disponíveis	`(((U>0) AND (VMMoodle <= (ProcessamentoVM*20)/100)) OR ((U>0) AND (AguardaMoodle >= 0) AND (VMMoodle <= (ProcessamentoVM*20)/100)) OR ((U>0) AND (AguardaMoodle >= 0) AND (VMMoodle <= (ProcessamentoVM*20)/100) AND (VMAlocada>0)))`	-	2
Alocacao	Imediata	Aloca uma requisição a um processamento Moodle	-	-	1
DesligaVM	Imediata	Desliga uma VM quando o lugar VMMoodle possuir valor superior a 40% da definição ProcessamentoVM	`VMMoodle > (ProcessamentoVM + (ProcessamentoVM*40)/100)`	-	1

Fonte: Elaborada pelo autor (2025)

Enquanto o fluxo básico de requisições é definido por arcos de peso unitário, a lógica central do modelo é implementada por um conjunto de arcos especiais, nos quais esta descrição se concentrará. Tais arcos são cruciais para o comportamento dinâmico do sistema, sendo responsáveis por: (I) acionar o escalonamento e o desligamento de VMs; (II) instanciar uma nova VM, adicionando capacidade de processamento ao sistema através de um arco com peso variável; e (III) zerar as filas e os recursos alocados em caso de uma falha completa do servidor físico. A Tabela 9 apresenta a especificação destes arcos mais importantes. A sintaxe usada para expressar esta condição é a usada pela ferramenta Mercury (SILVA et al., 2015)

Para concluir a descrição do modelo, é fundamental detalhar a funcionalidade das transições que realizam a lógica de escalonamento automático: **Escalonador** e **DesligarVM**. Este mecanismo garante a elasticidade do ambiente, ajustando dinamicamente o número de Máquinas Virtuais à demanda, com o objetivo de garantir a disponibilidade sem incorrer em custos com recursos ociosos. A lógica é implementada da seguinte forma:

- *Scale-Up*: A transição **Escalonador** é acionada em cenários de alta demanda. Seu disparo ocorre quando a quantidade de recursos de processamento disponíveis, representada pelo número de marcas no lugar **VMMoodle**, cai abaixo de um limiar inferior (neste mo-

Tabela 9 – Arcos com pesos condicionais do modelo

Origem	Destino	Peso condicional	Ação
VMTI	F	`IF(Server-#D=1):(#VMTI) ELSE(0)`	Retirar todos os tokens alocados em VMTI
VMMoodle	F	`IF(Server-#D=1):(#VMMoodle) ELSE(0)`	Retirar todos os tokens alocados em VMMoodle
VMs	F	`IF(Server-#D=1):(#VMs) ELSE(0)`	Retirar todos os tokens alocados em VMs
TI	VMMoodle	`ProcessamentoVM`	Adicionar uma VM com Ns tokens definidos em Capacidade de Processamento
F	Entrada	`IF(Server-#D=1):(#MoodleAlocado + #AguardaMoodle) ELSE(0)`	Retirar todos os trabalhos que estavam sendo processados
VMMoodle	VMF	`IF(#MoodleAlocado >= ProcessamentoVM):(0) IF(#VMMoodle >= ProcessamentoVM):(ProcessamentoVM) ELSE(#VMMoodle)`	Falhar uma VM retirando seus tokens de VMMoodle e MoodleAlocado
MoodleAlocado	VMF	`IF(#MoodleAlocado > ProcessamentoVM):(ProcessamentoVM) IF(#VMMoodle < ProcessamentoVM):(#MoodleAlocado) IF(#MoodleAlocado < ProcessamentoVM):(0) ELSE(#MoodleAlocado)`	Falhar uma VM retirando seus tokens de VMMoodle e MoodleAlocado

Fonte: Elaborada pelo autor (2025)

delo, 20% da capacidade total de uma VM). Este comportamento representa a alocação e instanciação de uma nova VM para aumentar a capacidade de serviço do sistema.

- *Scale-Down*: Inversamente, a transição **DesligarVM** é responsável por otimizar o uso de recursos em períodos de baixa demanda. Ela é disparada quando os recursos ociosos no sistema excedem um limiar superior. No modelo, isso ocorre quando o número de marcas em **VMMoodle** indica que há o equivalente a uma VM inteira mais 40% de uma segunda VM ociosa. O seu disparo remove uma VM inativa, reduzindo custos de infraestrutura.

O modelo proposto constitui uma base formal e flexível para a análise de desempenho do sistema Moodle em nuvem. As seções seguintes mostrarão a validação da arquitetura básica bem como a aplicação dos modelos aqui criados, instanciando seus parâmetros com valores numéricos para simular diferentes cenários operacionais e avaliar métricas de interesse.

5 VALIDAÇÃO DO MODELO ARQUITETURA BÁSICA

Após a concepção dos modelos analíticos no capítulo anterior, uma etapa fundamental do processo de análise de sistemas é a validação. A validação busca verificar se o modelo proposto representa, com um grau aceitável de precisão, o comportamento do sistema real. Sem esta etapa, as conclusões tiradas a partir do modelo poderiam não ser críveis. Este capítulo detalha, portanto, o processo experimental e estatístico realizado para validar o modelo da arquitetura básica, que serve como alicerce para todas as análises subsequentes.

Para tal, foi adotada uma metodologia que combina a execução de um experimento prático com a análise estatística dos resultados já detalhado no Capítulo 3. O processo consiste em submeter uma implementação real da arquitetura básica a um ciclo de falhas e reparos controlados, coletar dados de disponibilidade e, por fim, comparar esses dados experimentais com os resultados obtidos através dos modelos. O objetivo é determinar se o modelo tem evidências para ser aceito como uma representação do sistema real.

5.1 AMBIENTE EXPERIMENTAL E INJEÇÃO DE FALHAS

O primeiro passo para a validação foi a construção de um ambiente de teste que replica a arquitetura básica proposta: uma instalação do Moodle sobre uma pilha LAMP (Linux, Apache, MySQL, PHP). Este ambiente foi configurado em um servidor físico com especificações que garantem o funcionamento adequado da aplicação, servindo como o nosso "sistema alvo".

Para simular o comportamento de falha e reparo do sistema ao longo do tempo sem a necessidade de esperar por falhas naturais, foi empregada a técnica de injeção de falhas. Um conjunto de scripts foi desenvolvido para atuar como um injetor de falhas, que, em intervalos de tempo aleatórios uniformes, interrompia um dos serviços essenciais dos componentes da pilha LAMP, no Sistema Operacional ou no no Hardware. De forma análoga, o reparo era simulado pela restauração do serviço interrompido.

Paralelamente, em um segundo servidor, um script atuou monitorando os serviços do Moodle, do Sistema Operacional e do Hardware do servidor principal, verificando a cada 5 segundos o seu o estado do serviço Moodle e registrando a informação em um arquivo de log. Este log de disponibilidade constitui o dado bruto coletado do experimento, contendo o histórico temporal do comportamento do sistema real.

Os trabalhos de (BEZERRA, 2015), (COSTA, 2015) e (DANTAS et al., 2012) mostram os tempos necessários para configurar $MTTFs$ e $MTTRs$ nos scripts dos componentes estudados. Os valores da Tabela 10 foram inseridos no RBD da Figura 16 para se ter os valores padrões desejáveis no experimento.

Tabela 10 – Componentes do sistema

Componente	***MTTF (h)***	***MTTR (h)***
Hardware	8760	1.67
Sistema Operacioanl	2880	1
MySQL	1440	1
Apache	788.4	0.5
PHP	788.4	0.5

Fonte: Elaborada pelo autor (2025)

No contexto experimental, busca-se validar empiricamente que a métrica de disponibilidade aferida no experimento esteja estatisticamente consistente com o comportamento esperado do sistema, previamente modelado. Para isso, estabelece-se como critério que o valor observado de disponibilidade no experimento deve situar-se dentro do intervalo de confiança de 95% estimado a partir do modelo de referência da Figura 16

O Intervalo de Confiança é uma medida estatística crucial para a validação de modelos de simulação e análise de resultados experimentais (CLEMENTE, 2022). Ele fornece uma faixa de valores na qual se espera, com um determinado nível de confiança (geralmente 95%), que o verdadeiro valor de um parâmetro da população se encontre (DEVORE, 2008). A utilização de intervalos de confiança permite verificar se o resultado obtido por um modelo analítico ou de simulação é estatisticamente consistente com os dados medidos em um sistema real (MORAIS et al., 2013).

A partir dos valores mostrados na Tabela 10, derivam-se as métricas de disponibilidade do sistema, calculadas a partir da ferramente Mercury, permitindo estabelecer uma estimativa pontual de disponibilidade. A métrica de disponibilidade obtida nesse experimento será comparada ao intervalo de confiança do modelo base, visando verificar sua consistência estatística e, portanto, a validade do experimento.

A partir da análise do modelo pelo Mercury chega-se aos valores indicados na Tabela 11:

Contudo é inviável validar um experimento quando se tem componentes com tempos de falha ou reparos muito altos. Desafio semelhante foi enfrentado por (BEZERRA, 2015), (COSTA, 2015) e (DANTAS et al., 2012). Para isso é necessário usar um "fator de aceleração" para

Tabela 11 – Valores de referência

Métrica	Valor
MTTF (horas)	270.81150
MTTR (horas)	0.67786
Disponibilidade (%)	99.750
Número de 9's	2.60260
Uptime (horas no ano)	8743.92581
Downtime (horas no ano)	21.88695

Fonte: Elaborada pelo autor (2025)

componentes com tempos de falha elevados de maneira que "falhem" proporcionalmente mais rápidos. Nosso estudo aplicou um fator de aceleração de 1.000 unidades de tempo a todos os $MTTFs$ de maneira que seus tempos iniciais, no script de falha, foram divididos pelo "fator de aceleração" resultando nos valores mostrados na Tabela 12. Para calcular as métricas corretamente e validar a arquitetura básica os tempos foram restaurados à sua grandeza de origem multiplicando-os pelo "fator de aceleração" anteriormente definido.

Tabela 12 – Parâmetros de entrada com fator de acelaração

Componente	*MTTF (h)*	*MTTR (h)*
Hardware	8.76	1.67
Sistema Operacioanl	2.88	1
MySQL	1.44	1
Apache	0.7884	0.5
PHP	0.7884	0.5

Fonte: Elaborada pelo autor (2025)

O sistema ficou sob monitoramento por aproximadamente 103 horas resultando em um arquivo de log pouco mais de 74.222 linhas mostrando todos os períodos de disponibilidade ou indisponibilidade do servidor Moodle através de marcações UP ou $Down$ como mostrado na Seção 3.2.

5.2 ANÁLISE DOS DADOS EXPERIMENTAIS

Com o log de disponibilidade gerado pelo experimento, a próxima fase deve ser a análise do arquivo para extrair as métricas empíricas do sistema. A partir das marcas de tempo, foram calculados todos os Períodos em Operação (*UP*) e Períodos em Falha (*Down*) do Moodle.

Com base nestes dados, foi analisado a disponibilidade do sistema a partir dos valores de MTTF e MTTF, usando a equação 2.8 de forma a verificar se o experimento condiz, com certo grau de confiança, com a realidade.

Os dados compilados a partir do arquivo de log são mostrados na Tabela 13:

Tabela 13 – Valores do experimento

Métrica	Valor
MTTF (horas)	332.6726
MTTR (horas)	1.1525
Disponibilidade (%)	99.6583
Número de 9's	2.4664
Uptime (horas no ano)	8730.07456
Downtime (horas no ano)	29.92543

Fonte: Elaborada pelo autor (2025)

Para a validação formal do modelo analítico, adotou-se o método estatístico proposto por (KEESEE, 1965), que consiste no cálculo do intervalo de confiança para a disponibilidade a partir de dados experimentais. Este método utiliza o número de ciclos de falha/reparo – que no experimento correspondeu a 94. Esse valor corresponde ao grau de liberdade da distribuição F de Snedecor, que se demonstrou a mais adequada para os dados coletados. A partir desta distribuição, com um nível de confiança de 95%, foram obtidos os valores críticos inferior (L) e superior (U), conforme detalhado na Tabela 14.

Tabela 14 – Valores base para grau de liberdade

Distribuição F	Valor
Grau de liberdade	94
Valor crítico inferior – L	0.6658
Valor crítico superior – U	1.502

Fonte: Elaborada pelo autor (2025)

Estes valores críticos são então aplicados na Equação 5.1 para determinar os limites do intervalo de confiança da disponibilidade:

$$\left(\frac{1}{1+U}, \frac{1}{1+L}\right) = \left(\frac{1}{1+1.502}, \frac{1}{1+0.6658}\right) \tag{5.1}$$

A execução do cálculo resulta em um intervalo de confiança para a disponibilidade experimental de $(0,994877894; 0,997723002)$, cujos limites estão apresentados na Tabela 15.

Tabela 15 – Intervalo de Confiança do Experimento

Intervalo de Confiança - 95%	Experimento	Modelo
(0.99487 – 0.99772)	0.99658	0.99750

Fonte: Elaborada pelo autor (2025)

5.3 COMPARAÇÃO E CONCLUSÃO DA VALIDAÇÃO

Como o valor encontrado pelo modelo está contido no intervalo de confiança de 95% obtido experimentalmente, não há evidências estatísticas para rejeitar o modelo. Esta consistência assegura que o modelo proposto reflete adequadamente o comportamento do sistema real e, portanto, pode ser utilizado com confiança como uma base validada para a criação de novos modelos do sistema e para a obtenção das demais métricas.

6 ESTUDOS DE CASO

Com a arquitetura básica devidamente validada no capítulo anterior, e com os modelos gerais para as demais arquiteturas já concebidos, este capítulo dedica-se à aplicação prática destes formalismos. Através de uma série de estudos de caso, os modelos serão utilizados como uma ferramenta preditiva para avaliar quantitativamente os benefícios de diferentes cenários, algo que seria custoso e complexo de se realizar com experimentos em ambientes reais. O objetivo é fornecer uma análise comparativa que possa guiar o planejamento de infraestruturas para o ambiente Moodle, respondendo às questões de pesquisa levantadas por este trabalho.

A análise se debruçará em quatro estudos de caso sequenciais e complementares, conforme a estrutura delineada no trabalho. Primeiramente, será conduzida uma análise de sensibilidade sobre o modelo da arquitetura básica para identificar os componentes mais críticos que impactam a disponibilidade do sistema.

Em seguida, como segundo estudo, será avaliado o ganho de disponibilidade obtido com a introdução de redundância em pontos do sistema bem como em outras métricas de interesse para o modelo. O terceiro estudo de caso será conduzido sobre a arquitetura virtualizada, analisando o impacto da pilha de virtualização na disponibilidade de ponta a ponta do serviço. Por fim, o último estudo de caso mudará o foco da disponibilidade para o desempenho, utilizando o modelo concebido para analisar métricas como utilização e vazão, entre outras, em um cenário de nuvem pública com escalonamento automático.

Em conjunto, estes estudos fornecem uma análise comparativa e abrangente, oferecendo *insights* para o planejamento e a otimização de infraestruturas para ambientes de gestão de aprendizagem.

6.1 AVALIAÇÃO DO IMPACTO DOS COMPONENTES DA ARQUITETURA BÁSICA SOBRE A DISPONIBILIDADE

O primeiro estudo de caso consiste em uma análise de sensibilidade do modelo validado da arquitetura básica, para tanto usamos o modelo em CTMC descrito na Figura 18. O objetivo desta análise é identificar quantitativamente quais parâmetros são mais críticos para o cálculo da disponibilidade. A identificação desses pontos sensíveis é fundamental, pois permite direcionar os esforços de otimização de forma mais eficaz, visando o maior ganho de disponibilidade

com o menor investimento de recursos.

Utilizando as taxas apresentadas na Tabela 10 e calculando os tempos médios de falha e reparo do componente *app* – representado na CTMC da Figura 18 como a composição funcional do Apache, PHP e MySQL, conforme ilustrado no RBD da Figura 17 – obtêm-se os valores de taxas consolidados na Tabela 16, servindo como valores de referência (*baseline*) empregados nas análises subsequentes.

Tabela 16 – CTMC – Parâmetros de Entrada

Taxa	**Descrição**	**Valor** h^{-1}
λ_{hdw}	Taxa de Falha da VM	$1/4380$
λ_{so}	Taxa de Falha do Software	$1/2880$
λ_{app}	Taxa de Falha da Aplicação	$1/309.48$
α_{hdw}	Taxa de Instanciação da VM	$1/0.0417$
μ_{so}	Taxa de Reboot do Software	$1/0.05$
μ_{app}	Taxa de Reparo da Aplicação	$1/0.607$

Fonte: Elaborada pelo autor (2025)

A metodologia adotada foi a análise de sensibilidade diferencial estudada na Seção 2.8, que avalia a taxa de variação da métrica de saída (disponibilidade) em resposta a uma mudança em um único parâmetro de entrada, enquanto os demais permanecem fixos em seus valores de baseline. A análise de sensibilidade diferencial é caracterizada pelo índice de sensibilidade $S_\lambda(Y)$ que indica o impacto dos parâmetros λ e μ na disponibilidade do sistema.

Importante recordar que as taxas λ e μ são, respectivamente, o inverso dos tempos de $MTTF$ e $MTTR$ dos estados analisados. A Tabela 17 mostra os resultados desta análise, classificando a sensibilidade de cada parâmetro. Os resultados são ordenados de acordo com os valores absolutos dos índices de sensibilidade. O valor absoluto reflete a intensidade com que um parâmetro pode influenciar a disponibilidade. Valores negativos indicam que há uma relação inversa entre os parâmetros e a disponibilidade do sistema.

Os resultados consolidados da análise revelam que a disponibilidade do sistema exibe a maior sensibilidade em relação aos parâmetros do componente de aplicação (APP) . Isso indica que melhorias na taxa de reparo (μ_{app}) ou na redução da taxa de falhas da aplicação (λ_{app}) geram um impacto mais significativo na disponibilidade do serviço quando comparadas a melhorias de mesma magnitude em outros elementos da arquitetura.

O Sistema Operacional (SO) é identificado como o segundo componente mais crítico com base também nos valores de μ e λ.

Tabela 17 – Resultado da Análise de Sensibilidade

Taxa	Valor do Índice de Sensibilidade
λ_{app}	$-1{,}9595 \times 10^{-3}$
μ_{app}	$1{,}9589 \times 10^{-3}$
μ_{so}	$1{,}7360 \times 10^{-5}$
λ_{so}	$-1{,}6947 \times 10^{-5}$
α_{hdw}	$9{,}5204 \times 10^{-6}$
λ_{hdw}	$-9{,}2484 \times 10^{-6}$

Fonte: Elaborada pelo autor (2025)

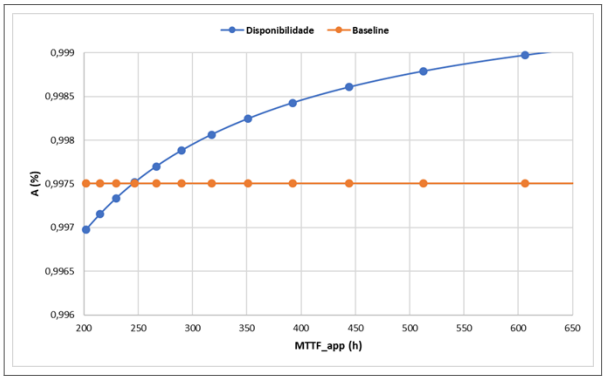
Para compreender de forma mais aprofundada o impacto efetivo dessas taxas, foram conduzidos experimentos nos quais se variaram os parâmetros de falha e de reparo relativos à aplicação e ao sistema operacional, considerados os dois componentes mais críticos do sistema. Tais parâmetros foram ajustados para valores superiores e inferiores em relação às suas taxas originais, de modo a avaliar diferentes cenários de comportamento. A partir desses experimentos, foi possível gerar gráficos que evidenciam em quais condições o sistema apresenta melhores resultados em termos de disponibilidade, permitindo uma análise comparativa mais robusta e fundamentada.

Os gráficos das Figuras 23 e 24 mostram a variação da disponibilidade (A) – delimitada pela linha azul – em relação à taxa de interesse. Para fins de comparação, traçamos uma reta (linha laranja) com o valor da disponibilidade validada pela $baseline$ mostrado em seção anterior.

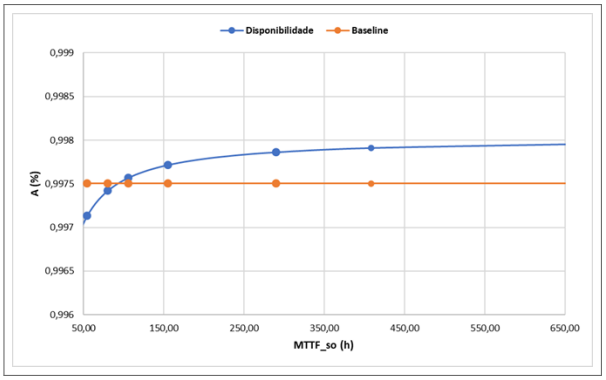
Inicialmente, analisamos a variação da disponibilidade em relação ao tempo de falha da aplicação $MTTF_{app}$, pois a análise de sensibilidade mostrou que este parâmetro foi o mais sensível. Variou-se o tempo entre $200h$ e $650h$. O gráfico dos valores para esses tempos pode ser visto na Figura 23a. Observamos que, à medida que o tempo de falha aumenta, a disponibilidade aumenta, chegando a 99,8% próximo de $650h$.

Uma vez que estamos analisando a taxa de falhas, podemos estudar a falha do sistema operacional (λ_{so}) – o quarto parâmetro mais sensível do modelo. A Figura 23b mostra que com o aumento do tempo de falhas do Sistema Operacional, o Moodle tende a aumentar sua disponibilidade, semelhante ao parâmetro anterior, porém de forma bem menos sensível à variação. Observa-se que o LMS leva muito mais tempo para se distanciar do limiar da da baseline, bem como apresenta significativa diferença na sua curva quando comparado aos 99,8% da análise anterior.

Figura 23 – Variação $A \times MTTF$ para componentes Aplicação e Sistema Operacional



(a) Variação $A \times MTTF_{app}$

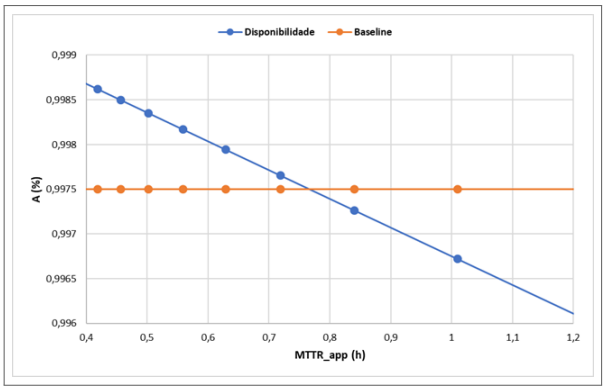


(b) Variação $A \times MTTF_{so}$

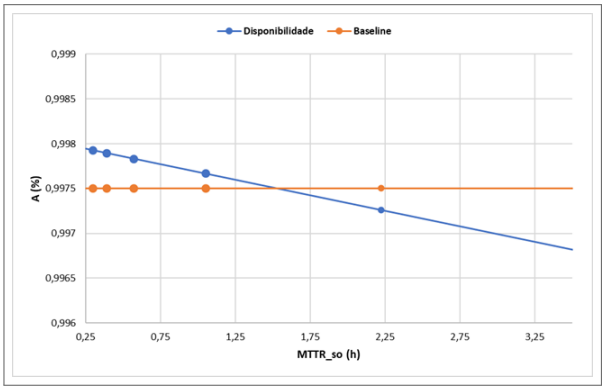
Fonte: Elaborada pelo autor (2025)

Continuando o estudo de caso, percebemos que, de acordo com a Tabela 17, o segundo componente mais sensível é o tempo de reparo do aplicativo (representado pela taxa μ_{app}). De maneira semelhante, plotamos o gráfico de sua variação na Figura 24a. Percebemos que tempos de reparo mais prolongados resultam em menor disponibilidade do sistema,isto é, à medida que a taxa de reparo diminui (ou seja, o tempo médio de reparo aumenta), a disponibilidade do sistema também se reduz de forma correspondente.

Figura 24 – Variação $A \times MTTR$ para componentes Aplicação e Sistema Operacional



(a) Variação $A \times MTTR_{app}$



(b) Variação $A \times MTTR_{so}$

Fonte: Elaborada pelo autor (2025)

Comportamento análogo é observado em relação à taxa de reparo do sistema operacional (μ_{so}), conforme ilustrado na Figura 24b. Um aumento no tempo de reparo implica redução na disponibilidade do sistema; entretanto, como esse parâmetro apresenta sensibilidade menor do que a taxa de reparo do aplicativo (μ_{app}), constata-se uma diminuição bem menos acentuada da disponibilidade mesmo diante de variações mais amplas de tempo em comparação ao efeito verificado para μ_{app}.

É importante notar a diferença visual no comportamento das curvas apresentadas na Figura 23 e na Figura 24. Enquanto a variação da disponibilidade em função do $MTTF$ (Figura 23) exibe um comportamento assintótico, a variação em função do $MTTR$ (Figura 24) apresenta um comportamento praticamente linear. Essa distinção é explicada pela relação matemática fundamental da disponibilidade (A) em estado estacionário, definida pela Equação 6.1.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{6.1}$$

No caso da análise da Figura 23, o $MTTR$ é mantido constante enquanto o $MTTF$ varia. A natureza da função na Equação 6.1 faz com que, para valores baixos de $MTTF$, cada incremento gere um ganho significativo de disponibilidade. Contudo, à medida que o $MTTF$ se torna muito grande, o sistema se aproxima de seu limite teórico de 100% de disponibilidade ($A \to 1$). Cada hora adicional de confiabilidade ($MTTF$) resulta em um ganho marginal cada vez menor, fazendo com que a curva tenha características de uma função assintótica.

Por outro lado, na análise da Figura 24, o $MTTF$ é mantido constante enquanto o $MTTR$ varia. Embora a função ainda seja inerentemente não linear, no contexto de sistemas de alta disponibilidade como o analisado, onde o valor de $MTTF$ possui magnitude maior que o $MTTR$ ($MTTF \gg MTTR$), a relação pode ser aproximada por uma função linear. A indisponibilidade (U) do sistema é dada pela Equação 6.2:

$$U = 1 - A = \frac{MTTR}{MTTF + MTTR} \tag{6.2}$$

Dado que $MTTF$ é um valor muito grande em comparação com a faixa de variação de $MTTR$ nos gráficos, o denominador $(MTTF + MTTR)$ muda muito pouco. Portanto, podemos aproximar $MTTF + MTTR \approx MTTF$. Com isso, a indisponibilidade se torna aproximadamente proporcional ao $MTTR$, Equação 6.3:

$$U \approx \frac{1}{MTTF} \cdot MTTR \tag{6.3}$$

Como a indisponibilidade (U) tem uma relação aproximadamente linear com o $MTTR$, a disponibilidade ($A = 1 - U$) também terá. Isso resulta na reta com inclinação negativa observada no gráfico da Figura 24, onde cada hora adicional de indisponibilidade para reparo ($MTTR$) causa uma redução percentual correspondente e praticamente constante na disponibilidade total do sistema.

Percebe-se, portanto, que a análise de sensibilidade é fundamental no estudo da disponibilidade do sistema, pois permite a identificação dos parâmetros mais críticos que impactam diretamente sua disponibilidade. Além da otimização de recursos, a análise de sensibilidade fornece dados quantitativos que dão suporte a decisões estratégicas, como agendamento de manutenção preventiva, investimento em novos equipamentos ou atualizações de software.

6.2 AVALIAÇÃO DA ARQUITETURA REDUNDANTE

Nesta seção, será realizada a avaliação do impacto da introdução de mecanismos de redundância na arquitetura do Moodle, tomando como base o modelo RBD apresentado anteriormente na Figura 19. Esse modelo serve como referência para representar a dependência estrutural entre os principais componentes do sistema. O objetivo da análise é quantificar, de maneira comparativa, as melhorias obtidas a partir da duplicação de componentes críticos, considerando métricas clássicas de confiabilidade, como o tempo médio até a falha ($MTTF$), o tempo médio de reparo ($MTTR$), a *disponibilidade* e o *número de noves* associada a esta disponibilidade. A análise também contemplará a estimativa de tempos de operação ($uptime$) e de indisponibilidade ($downtime$), permitindo uma visão ampla sobre os benefícios e limitações do uso de redundância no contexto do Moodle.

O modelo proposto RBD foi então resolvido para obter as probabilidades de estado estacionário e as métricas de interesse foram calculadas. Para avaliar a eficácia desta estratégia, a Tabela 18 compara os resultados obtidos para o modelo redundante com os da arquitetura básica.

Tabela 18 – Comparação entre Arquitetura Básica e Arquitetura Redundante Física

Métrica	**Arquitetura Básica**	**Arquitetura Redundante Física**	**Melhoria (%)**
A (%)	99,7503	99,9994	+0,25%
MTTF (h)	270,81	406,22	+49,99%
MTTR (h)	0,6778	0,0025	–99,63%
Números de 9's	2,60	5,20	+100%
Uptime (h/ano)	8743,93	8765,76	+0,25%
Downtime (h/ano)	21,89	0,055	–99,75%

Fonte: Elaborada pelo autor (2025)

A avaliação do modelo sem redundância revelou um tempo médio até a falha ($MTTF$)

de aproximadamente 270,81 horas. Por sua vez, o tempo médio de reparo ($MTTR$) nesse cenário foi de 0,6779 horas. A partir desses valores, a disponibilidade estimada do sistema foi de 99,7503%, equivalente a cerca de 2,6 "noves", demonstrando um desempenho consistente, porém limitado em relação a falhas pontuais. Traduzindo esses indicadores para um horizonte anual, obteve-se um tempo médio de operação (uptime) de aproximadamente $8743,93$ horas e um tempo médio de indisponibilidade (downtime) de $21,89$ horas por ano.

Com a introdução da redundância no modelo RBD, os resultados mostraram avanços significativos. O tempo médio até a falha foi elevado para cerca de $406,21$ horas, evidenciando a maior resiliência do sistema em suportar falhas de componentes individuais sem comprometer sua operação completa. Além disso, o tempo médio de reparo sofreu uma redução expressiva, atingindo $0,0025$ horas.

No que se refere à disponibilidade, a configuração redundante apresentou um valor de 99,999%, o que corresponde a cerca de $5,2$ "noves". Trata-se de um incremento considerável em relação ao cenário sem redundância, representando um salto qualitativo no nível de qualidade de serviço percebido. Em termos absolutos, o sistema redundante passa a oferecer, em média, $8765,76$ horas de operação contínua ao longo do ano, com apenas 0,055 horas de indisponibilidade anual, contra quase 22 horas no cenário básico.

Essa evolução comprova a eficácia da estratégia de duplicação de caminhos completos no modelo RBD, que permite contornar falhas pontuais e garantir a continuidade do serviço com interrupções mínimas. O aumento do MTTF, aliado a um MTTR praticamente nulo, transforma a relação $\frac{MTTF}{MTTF+MTTR}$, elevando de forma substancial a disponibilidade do sistema.

Em síntese, a comparação entre os dois cenários demonstra que a aplicação de redundância em paralelo não apenas aumenta a confiabilidade estrutural do sistema, como também potencializa a qualidade percebida pelos usuários, ao reduzir drasticamente os períodos médios de indisponibilidade.

6.3 AVALIAÇÃO DA ARQUITETURA VIRTUALIZADA

Após a análise de estratégias de redundância em uma arquitetura física tradicional, este estudo de caso volta sua atenção para um paradigma arquitetônico alternativo e predominante na computação moderna: a virtualização. A migração de uma implementação em hardware dedicado para um ambiente virtualizado introduz benefícios como a consolidação de recursos, flexibilidade e agilidade no gerenciamento. Contudo, também adiciona novas camadas à pilha

de dependência do sistema (por exemplo, o hardware do hospedeiro, o sistema operacional do hospedeiro e o hypervisor), cujo impacto na disponibilidade do serviço precisa ser quantificado.

O objetivo desta seção é, portanto, avaliar o comportamento do ambiente Moodle quando implantado em uma infraestrutura virtualizada e, subsequentemente, analisar a eficácia da redundância neste novo contexto. Para conduzir esta análise, serão avaliados dois cenários distintos, ambos utilizando os modelos SPN para arquiteturas virtualizadas desenvolvidos no Capítulo 4.

Inicialmente, será analisada uma arquitetura virtualizada básica, composta por um único servidor hospedeiro (a partir do modelo da Figura 20). Em seguida, será avaliada uma arquitetura virtualizada com redundância cold-standby no nível do servidor (a partir do modelo da Figura 21), empregando dois hosts físicos para aumentar a resiliência do sistema. A comparação entre estes cenários e a arquitetura física original fornecerá uma visão abrangente sobre as vantagens e desvantagens de cada abordagem em termos de disponibilidade.

A partir dos trabalhos de (BEZERRA, 2015), (COSTA, 2015), (DANTAS et al., 2012) e (MELO, 2016b) e dos experimentos feitos anteriormente com a dependência APP encontramos os tempos necessários para configurar os parâmetros de entrada do modelo RBD proposto na Figura 20. Esses valores são mostrados na Tabela 19

Tabela 19 – Parâmetros de entrada do modelo RBD

Componente	***MTTF (h)***	***MTTR (h)***
Hardware	8760	1.67
Sistema Operacional	2880	1
VMM	2990	1
NC	788.0	1
VM	279.45	0.64

Fonte: Elaborada pelo autor (2025)

Resolvendo o modelo encontramos os valores expressos na Tabela 20 comparando as saídas do modelo virtualizado com a baseline da Figura 17:

Ao introduzir a arquitetura virtualizada, cujo modelo RBD passou a contemplar seis blocos – incluindo Hardware, Software, Virtual Machine Monitor (VMM), Node Controller (NC), Sistema Operacional da VM (OS_VM) e a Aplicação Moodle instanciada na VM (APP_VM) – observou-se impacto direto nos indicadores de confiabilidade e disponibilidade. O $MTTF$ foi reduzido para $177,20$ horas, evidenciando maior probabilidade de ocorrência de falhas, dada a quantidade superior de componentes e, consequentemente, de pontos de falha potenciais.

Tabela 20 – Comparação entre Arquitetura Básica e Arquitetura Virtualizada

Métrica	Arquitetura Básica	Arquitetura Virtualizada	Diferença (%)
A (%)	99,7503	99,5560	–0,19%
MTTF (h)	270,81	177,20	–34,6%
MTTR (h)	0,6778	0,7901	+16,6%
Números de 9's	2,60	2,35	–9,6%
Uptime (h/ano)	8743,93	8726,89	–0,19%
Downtime (h/ano)	21,89	38,91	+77,8%

Fonte: Elaborada pelo autor (2025)

O $MTTR$ também apresentou um leve aumento, chegando a 0,7903 horas, o que pode ser interpretado como reflexo da maior complexidade do ambiente virtualizado e da maior quantidade de elementos interdependentes envolvidos no processo de reparo.

Em consequência desses fatores, a disponibilidade do sistema virtualizado apresentou um decréscimo, sendo calculada em 99,556%, equivalente a aproximadamente $2,35$ "noves". Na prática, isso se traduziu em uma redução do tempo médio de operação anual para 8726,89 horas e um aumento do *downtime* anual para 38,92 horas (próximos do dobro do tempo de indisponibilidade observado no modelo tradicional).

De forma geral, esses resultados demonstram que, apesar de a virtualização trazer benefícios operacionais como flexibilidade, escalabilidade e facilidade de gerenciamento, ela também introduz novos pontos de falha e aumenta a complexidade estrutural do sistema. Contudo, vale ressaltar que a adoção da virtualização pode compensar essas perdas de disponibilidade ao oferecer estratégias de recuperação rápidas ou por meio de recursos que promovam a melhoria desta métrica, como a implementação de redundâncias.

Agora partimos para a comparação entre a arquitetura básica e a arquitetura Virtualizada com Redundância no Host (Figura 21). O objetivo é, a partir de um novo modelo em SPN, verificar se existem benefícios em termos de disponibilidade e no Downtime anual introduzido pela abordagem.

A Tabela 21 sintetiza os parâmetros de entrada empregados na modelagem da disponibilidade do Moodle, considerando a arquitetura escolhida. Uma vez que estamos modelando via SPN, as métricas serão: disponibilidade (A), número de noves e Downtime anual para o seu cálculo usamos a notação constante na Tabela 22 no Mercury

O modelo SPN foi então resolvido para obter a disponibilidade e demais métricas de

Tabela 21 – Entradas do modelo virtualizado com redundância

Parâmetro	Valor
Server	1
VM	1
MTTF Servidor (horas)	484.2940
MTTR Servidor (horas)	1.0376
MTTSC (horas)	0.019166
MTTF_SU (horas)	581.15
MTTF_VM (horas)	279.4506
MTTR_VM (horas)	0.6460

Fonte: Elaborada pelo autor (2025)

Tabela 22 – Definição das métricas de interesse da SPN com redundância

Métrica	Notação
A (%)	`P{(((#U>0)OR(#SU>0))AND(#VMU>0))}`
N9s	`-LOG{(1-P{(((#U>0)OR(#SU>0))AND(#VMU>0))})}`
Downtime (h/ano)	`(1-P{(((#U>0)OR(#SU>0))AND(#VMU>0))})*8760`

Fonte: Elaborada pelo autor (2025)

interesse. Para avaliar a eficácia desta estratégia, a Tabela 23 compara os resultados obtidos pela arquitetura básica com os valores obtidos através arquitetura virtualizada com redundância aplicada.

Tabela 23 – Comparação entre Arquitetura Básica e Arquitetura Virtualizada com redundância

Métrica	Arquitetura Básica	Arquitetura Virtualizada Redundante	Melhoria (%)
A (%)	0.99750	0.99862	+0,112%
Números de 9's	2.60	2.86	+10,00%
Downtime (h/ano)	21.88695	12.0758	-44,82%

Fonte: Elaborada pelo autor (2025)

A Tabela 23 apresenta os resultados comparativos entre a arquitetura básica, sem mecanismos de redundância, e a arquitetura virtualizada acrescida de redundância no host, aplicadas ao ambiente do Moodle. Observa-se que a arquitetura virtualizada com redundância oferece ganhos em termos de disponibilidade em comparação cenário anterior sem redundância.

Em relação à disponibilidade percentual (A%), verifica-se um acréscimo de aproximadamente 0,112% ao se migrar do modelo básico (99,750%) para o modelo virtualizado redundante

(99,862%). Embora à primeira vista a variação possa parecer modesta, ela traduz a redução de interrupções potencialmente críticas no contexto educacional, sobretudo em períodos de alta utilização do Moodle, como épocas de provas ou entrega de trabalhos.

No indicador de números de noves, houve um incremento de 2,60 para 2,86, correspondendo a uma melhoria percentual de 10,00%. Esse aumento reforça a robustez adicional proporcionada pela combinação de virtualização e redundância.

A métrica de Downtime anual apresentou o maior ganho relativo: passou de 21,89 horas/ano para 12,08 horas/ano, evidenciando uma redução de aproximadamente 44,82%. Esse resultado indica que a utilização de redundância no host virtualizado praticamente dobra a capacidade do sistema Moodle de permanecer operacional ao longo do ano. A redução do Downtime pode ser atribuída principalmente à capacidade de *failover* automático dos recursos virtualizados redundantes, que permite a rápida transferência de cargas de trabalho entre servidores físicos em caso de falhas, reduzindo o tempo médio de reparo.

6.4 ANÁLISE DE DESEMPENHO EM NUVEM PÚBLICA

Esta seção apresenta um estudo de caso utilizando o modelo de SPN proposto para avaliar o desempenho do Moodle sob diversas configurações como modelado na Figura 22. A análise emprega a ferramenta Mercury (SILVA et al., 2015) e avalia as principais métricas de desempenho detalhadas na Tabela 24, conforme proposto em Maciel (2023a).

Para garantir que a análise de desempenho do modelo seja representativa de um cenário real, a parametrização das transições temporizadas foi baseada na literatura. Os valores propostos nos trabalhos base (DANTAS et al., 2012), (COSTA, 2015), (BEZERRA, 2015) e (FE et al., 2017) foram utilizados como referência. A Tabela 25 detalha os tempos associados a cada transição temporizada do modelo previamente descrito na Tabela 8.

Embora não tenham sido derivados de medições empíricas, os tempos de processamento para diferentes instâncias da AWS foram estimados com base nas especificações de desempenho disponibilizadas pelo provedor e ajustados proporcionalmente. Considerando um tempo de processamento de 60 segundos para o tipo de instância menor, os tempos de processamento para as demais quatro instâncias foram calculados reduzindo-se sucessivamente 15% em relação ao tempo da instância anterior. Este método de estimativa, embora simplificado, é válido para fins de modelagem e permite ajustes flexíveis por parte de administradores de sistema em cenários reais de implantação. Os valores estimados de *Mean Time Between Processing*

Tabela 24 – Métricas de Desempenho Avaliadas no Estudo de Caso

Métrica	**Descrição**	**Fórmula**
Utilização	Proporção média de VMs efetivamente alocadas em relação à capacidade total.	$E\{\#\text{VMAlocada}\}/((\text{VM}+1)\times \text{ProcessamentoVM})$
Probabilidade de Descarte	Probabilidade de rejeição de requisição quando todos os recursos estão ocupados e há requisições em espera.	$P\{(\#\text{VMAlocada} = \text{ProcessamentoVM}\times(\text{VM}+1))\text{ and }(\#\text{AguardaMoodle}>0)\}$
Taxa de Descarte	Frequência de perda de requisições por unidade de tempo, calculada pelo produto entre a taxa de chegada e a probabilidade de descarte.	$P\{(\#\text{VMAlocada} = \text{ProcessamentoVM}\times(\text{VM}+1))\text{ and }(\#\text{AguardaMoodle}>0)\}\times(1/\text{MTBA})$
Vazão	Taxa efetiva de requisições concluídas com sucesso, obtida como a diferença entre a taxa de chegada e a taxa de descarte.	$(1/\text{MTBA})-(P\{(\#\text{VMAlocada} = \text{ProcessamentoVM}\times(\text{VM}+1))\text{ and }(\#\text{AguardaMoodle}>0)\}\times(1/\text{MTBA}))$

Fonte: Elaborada pelo autor (2025).

Tabela 25 – Transições Temporizadas, Variáveis e Tempos Associados

Transição	**Variável**	**Tempo (h)**
F	MTTF	484.29406
R	MTTR	1.03769
Chegada	MTBA	0.0083
Processamento	MTBP	variável
VMF	MTTF_VM	279.45069
TI	MTBI	0.0167

Fonte: Elaborada pelo autor (2025).

($MTBP$) para as cinco instâncias da AWS estão detalhados na Tabela 26. Esses valores foram integrados ao modelo para avaliar o impacto do desempenho heterogêneo das instâncias sobre o comportamento do sistema.

Essa abordagem contribui para um cenário de modelagem de desempenho mais realista ao incorporar a variabilidade dos tempos de processamento entre diferentes perfis de máquinas virtuais, o que é típico em ambientes LMSs em nuvem.

O estudo de caso foi conduzido variando-se sistematicamente os parâmetros de número de VMs, número de alunos e capacidade de processamento da VM listados da Tabela 7, com o objetivo de investigar o impacto dessas variáveis sobre as métricas de desempenho previamente definidas. Para cada configuração, foram compilados dados representando a relação entre as

Tabela 26 – MTBP Estimado para Instâncias AWS

Instância	MTBP Estimado (h)
t3.small	0.0167
t3.medium	0.0142
t3.large	0.0128
t3.xlarge	0.0115
t3.2xlarge	0.0103

Fonte: Elaborada pelo autor (2025).

métricas analisadas e os diferentes tipos de instâncias AWS especificados na Tabela 3.

Para a execução do experimento, foi adotada uma metodologia de análise em três etapas. A primeira etapa concentrou-se na avaliação da utilização do sistema em função da escalabilidade da infraestrutura. Para isso, o sistema foi submetido a uma carga de trabalho constante de 200 usuários simultâneos, enquanto se variava o número de máquinas virtuais (VMs) disponíveis. O objetivo dessa etapa foi identificar o ponto de inflexão a partir do qual um aumento no número de VMs não resultava mais em ganhos significativos na redução de saturação dos recursos.

A partir dos pontos de inflexão identificados, realizou-se uma segunda análise, voltada à avaliação da utilização do sistema sob demanda variável. Nessa fase, o número de VMs foi mantido fixo para cada tipo de instância (conforme os resultados obtidos na primeira etapa), e a carga de trabalho foi ajustada de forma progressiva, variando de 10 a 200 usuários simultâneos. Esse procedimento possibilitou observar o comportamento da utilização de recursos sob demanda crescente em uma infraestrutura já previamente dimensionada.

Por fim, a terceira etapa concentrou-se na análise do desempenho do sistema. Mantendo o número de VMs fixo (conforme determinado na segunda etapa), o sistema foi submetido a uma carga de usuários dimensionada explicitamente para a capacidade estimada de cada tipo de instância, conforme apresentado na Tabela 27.

Os parâmetros *ProcessamentoVM* e *MTBP* da Tabela 7 desempenharam um papel crucial nos experimento, pois especificam o tipo de instância da AWS avaliada. O *ProcessamentoVM* foi definida com base no número mínimo de usuários simultâneos suportados por cada instância, com valores variando de 5 a 180, conforme mostrado na Tabela 3, enquanto o *MTBP* foi definido de acordo com as especificações apresentadas na Tabela 26.

Tabela 27 – Estimativa de Usuários por Instâncias AWS

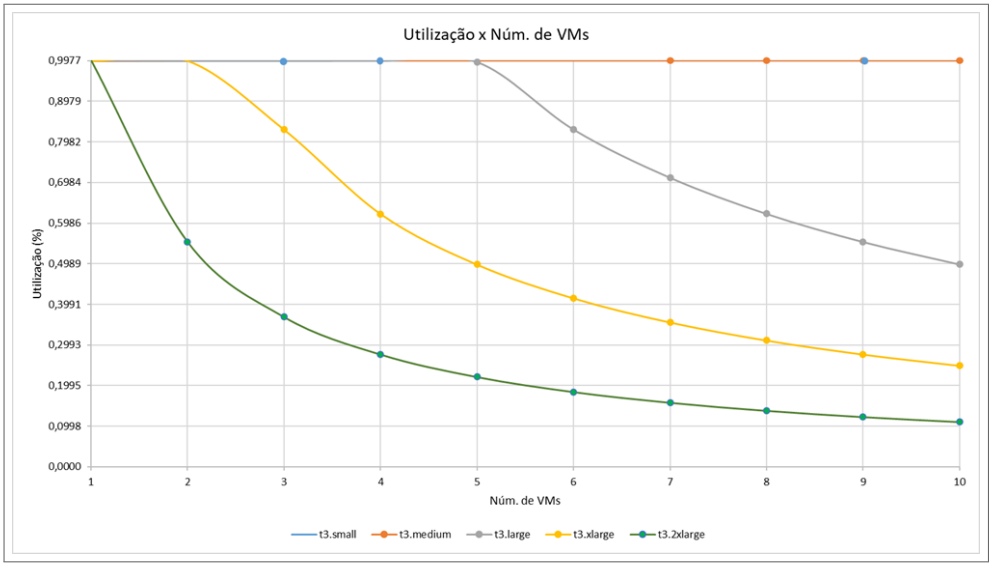
Instância	Número de Usuários por Experimento
t3.small	25 – 31
t3.medium	75 – 81
t3.large	200 – 206
t3.xlarge	400 – 406
t3.2xlarge	900 – 906

Fonte: Elaborada pelo autor (2025).

6.4.1 Análise de Utilização por Número de Máquinas Virtuais

Em relação à avaliação da **utilização** do sistema (Figura 25), observa-se que as duas instâncias de menor porte, t3.small e t3.medium, apresentam um padrão constante de saturação. Ambas mantêm um nível de utilização elevado, próximo de 99,77%, independentemente do aumento no número de máquinas virtuais de 1 até 10. Esse comportamento revela uma limitação clara de capacidade, indicando que essas instâncias operam como gargalos persistentes de desempenho para a carga de trabalho aplicada, mostrando-se insuficientes para processar a demanda, mesmo quando configuradas com o maior número de VMs.

Figura 25 – Utilização x Número de VMs



Fonte: Elaborada pelo autor (2025)

A alteração no padrão de comportamento torna-se evidente a partir da instância t3.large. Essa configuração mantém a utilização máxima até aproximadamente quatro VMs, passando, a partir da quinta máquina, a apresentar uma queda gradual. Ainda assim, a redução na

utilização é modesta, atingindo valores próximos de 50% apenas quando são alocadas dez VMs. Essa característica sugere que a capacidade de processamento da instância t3.large, embora superior às menores, ainda exige uma quantidade relativamente elevada de VMs para aliviar o gargalo imposto pela carga de 200 usuários simultâneos.

Por sua vez, a instância t3.xlarge revela um desempenho superior em termos de capacidade de processamento. Sua curva de utilização começa a apresentar uma queda mais acentuada já a partir da segunda VM, evidenciando uma mitigação mais eficiente do gargalo de recursos conforme o sistema é escalado. Com o acréscimo de mais máquinas virtuais, a utilização continua a declinar, alcançando valores abaixo de 30% quando são utilizadas dez VMs, o que reforça a maior robustez da instância para atender à mesma carga de trabalho.

Por fim, a instância mais robusta, t3.2xlarge, demonstra ser capaz de resolver de forma bastante eficiente o gargalo de recursos. Já com apenas uma VM atinge utilização máxima, mas, ao adicionar uma segunda máquina, verifica-se uma queda substancial na taxa de utilização, que passa a valores levemente superiores a 10% com dez VMs. Esse comportamento denota a presença de capacidade computacional alta, eliminando o gargalo aparente com apenas algumas unidades.

Os resultados apresentados permitem evidenciar uma estratificação clara no desempenho das diferentes instâncias avaliadas. As instâncias de menor porte revelam-se sistematicamente subdimensionadas para a carga de 200 usuários, enquanto as de maior porte, especialmente a t3.2xlarge, asseguram maior escalabilidade e flexibilidade operacional, ressaltando a importância crítica do dimensionamento adequado para garantir a eficiência e o desempenho do Moodle.

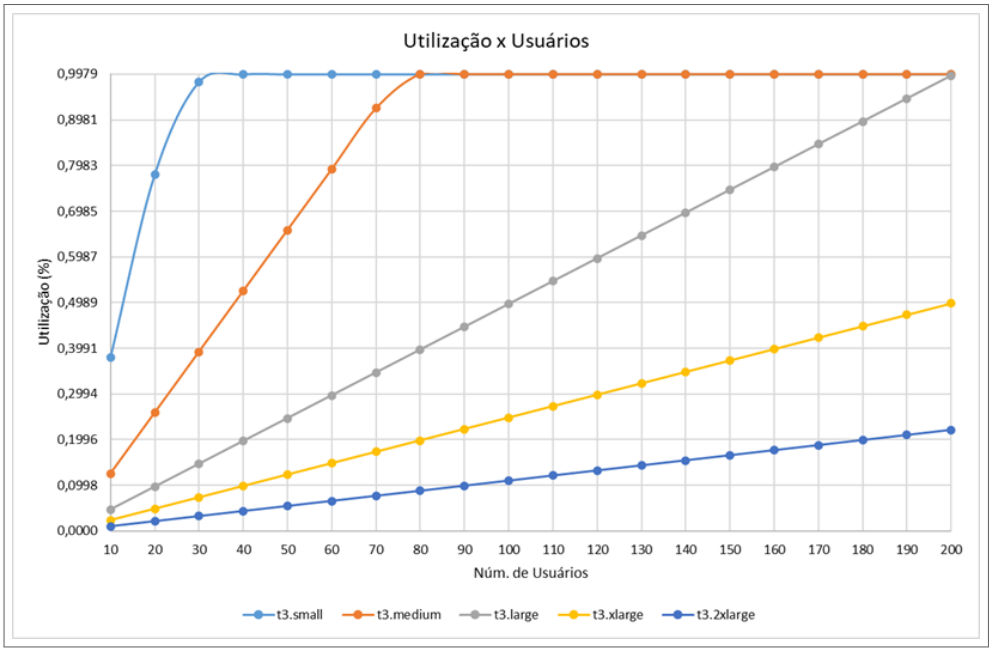
Atendendo ao objetivo de identificar o ponto de inflexão a partir do qual um aumento no número de VMs deixa de gerar ganhos significativos na redução da saturação adotou-se como critério prático o menor número de VMs após o qual o incremento de mais uma VM produz uma redução marginal da utilização considerada não significativa (ordem de grandeza: redução $<$ 5 pontos percentuais no nível de utilização). Com base nas curvas da Figura 25 e nos parâmetros experimentais, os pontos de inflexão observados são, aproximadamente, os seguintes: **t3.small e t3.medium** não alcançaram um ponto de inflexão no intervalo testado (1–10 VMs), mantendo utilização elevada ($\approx 99{,}7\%$) e, portanto, não se mostram solução custo-efetiva para a carga de 200 usuários; **t3.large** apresenta ponto de inflexão próximo de 4–5 VMs, a partir do qual ganhos adicionais passam a ser marginais. A instância **t3.xlarge** tem ponto de inflexão em torno de 2–3 VMs, apresentando diminuições mais acentuadas de

utilização nas primeiras VMs adicionadas. A **t3.2xlarge**, por sua vez, apresenta ponto de inflexão em 1–2 VMs, indicando que uma ou duas unidades já fornecem folga substancial para a carga testada. Em termos de compromisso custo × desempenho, pode-se optar pela t3.xlarge com 2 VMs ou t3.large com 4–5 VMs, visto que ambas as opções aliviam gargalos com número moderado de VMs.

6.4.2 Análise de Utilização sob Carga Variável de Usuários Simultâneos

No experimento que avaliou a **utilização do sistema** sob demanda variável, manteve-se a infraestrutura fixa em 5 máquinas virtuais, enquanto a carga de trabalho foi escalonada de 10 até 200 usuários simultâneos. O comportamento do sistema, ilustrado na Figura 26, revela uma estratificação bastante clara do desempenho das instâncias estudadas, permitindo inferências relevantes sobre sua capacidade de suportar o crescimento da demanda.

Figura 26 – Utilização x Núm. de Usuários



Fonte: Elaborada pelo autor (2025)

As instâncias de menor porte, t3.small e t3.medium, apresentaram esgotamento precoce de seus recursos computacionais. O gráfico evidencia que a t3.small atingiu rapidamente a utilização próxima ao limite máximo (cerca de 99,7%) já a partir de 30 usuários, mantendo-se saturada para quaisquer valores adicionais de carga. A t3.medium, por sua vez, mesmo com um incremento moderado de recursos, atingiu saturação com aproximadamente 80 usuários, não apresentando capacidade adicional para absorver picos. Esse comportamento demons-

tra uma subdimensionamento evidente dessas configurações, caracterizando-as como gargalos persistentes para aplicações de maior porte.

Ao se observar a instância t3.large, percebe-se um padrão intermediário. Inicialmente, sua utilização cresce linearmente em função do aumento de usuários, evidenciando bom aproveitamento da capacidade até um ponto de saturação próximo de 200 usuários. Esse padrão revela que a t3.large é mais adequada ao cenário avaliado, pois consegue distribuir de forma eficiente seus recursos computacionais durante grande parte do crescimento de demanda, apresentando saturação apenas em cargas elevadas, compatíveis com o limite projetado para essa categoria de instância.

Por outro lado, as instâncias mais robustas, t3.xlarge e t3.2xlarge, destacam-se pelo significativo excedente de capacidade computacional. A t3.xlarge inicia com baixa utilização e vai crescendo suavemente, alcançando apenas cerca de 45% de uso com 180 usuários, o que evidencia grande folga de processamento para acomodar aumentos adicionais de carga sem riscos de degradação do serviço. Mais impressionante ainda é a t3.2xlarge, que mantém utilização próximo de 20% mesmo no pico de 200 usuários simulados, demonstrando estar substancialmente sobredimensionada frente ao cenário testado.

Essa análise conjunta sugere que, embora as instâncias de menor porte não atendam de forma satisfatória à demanda em ambientes com alta concorrência, as instâncias de médio e alto porte oferecem margens de segurança amplas, dotadas de capacidade para absorver crescimento futuro de requisições bem como picos de demanda, preservando a estabilidade e a qualidade do serviço..

6.4.3 Análise de Desempenho com Carga Estimada de Usuários por Instância

No estudo de caso final, o desempenho do sistema foi avaliado novamente. Diferentemente dos estudos anteriores, o objetivo aqui foi submeter cada tipo de instância a uma carga de trabalho adaptada explicitamente à sua capacidade estimada, mantendo a configuração da infraestrutura fixa em cinco VMs.

Esse procedimento visou proporcionar uma comparação mais justa entre as instâncias, permitindo mensurar não apenas a capacidade de processamento, mas também a qualidade do serviço e a experiência do usuário final sob uma demanda ideal ou próxima do limite de capacidade.

Para quantificar o desempenho, três métricas de interesse foram avaliadas: **probabilidade**

de descarte, **taxa de descarte** e a **vazão**, conforme mostrado nas Tabelas 28, 29 e 30. A análise combinada desses dados permite avaliar a capacidade de processamento de cada instância.

Tabela 28 – Probabilidade de Descarte para Instâncias AWS com 5 VMs

t3.small		t3.medium		t3.large		t3.xlarge		t3.2xlarge	
Users	**DP**	**Users**	**DP**	**Users**	**DP**	**Users**	**DP**	**Users**	**DP**
25	0	75	0	200	0	400	0	900	0
26	0.6051	76	0.5540	201	0.5190	401	0.4827	901	0.4445
27	0.9077	77	0.8799	202	0.8582	402	0.8332	902	0.8039
28	0.9833	78	0.9758	203	0.9691	403	0.9604	903	0.9493
29	0.9959	79	0.9946	204	0.9932	404	0.9913	904	0.9884
30	0.9975	80	0.9973	205	0.9972	405	0.9968	905	0.9963
31	0.9977	81	0.9977	206	0.9977	406	0.9977	906	0.9976

Fonte: Elaborada pelo autor (2025).

Tabela 29 – Taxa de Descarte para Instâncias AWS com 5 VMs (h^{-1})

t3.small		t3.medium		t3.large		t3.xlarge		t3.2xlarge	
Users	**DR**	**Users**	**DR**	**Users**	**DR**	**Users**	**DR**	**Users**	**DR**
25	0	75	0	200	0	400	0	900	0
26	72.6137	76	66.4831	201	62.2822	401	57.9204	901	53.3421
27	108.9215	77	105.5908	202	102.9880	402	99.9809	902	96.4727
28	117.9992	78	117.0937	203	116.2906	403	115.2532	903	113.9105
29	119.5127	79	119.3497	204	119.1890	404	118.9503	904	118.6106
30	119.7022	80	119.6817	205	119.6627	405	119.6215	905	119.5607
31	119.7213	81	119.7210	206	119.7247	406	119.7191	906	119.7144

Fonte: Elaborada pelo autor (2025).

Tabela 30 – Vazão para Instâncias AWS com 5 VMs

t3.small		t3.medium		t3.large		t3.xlarge		t3.2xlarge	
Users	**TP**	**Users**	**TP**	**Users**	**TP**	**Users**	**TP**	**Users**	**TP**
25	120.0000	75	120.0000	200	120.0000	400	120.0000	900	120.0000
26	47.3863	76	53.5169	201	57.7179	401	62.0797	901	66.6580
27	11.0786	77	14.4092	202	17.0120	402	20.0191	902	23.5273
28	2.0008	78	2.9064	203	3.7094	403	4.7468	903	6.0896
29	0.4874	79	0.6504	204	0.8111	404	1.0498	904	1.3895
30	0.2979	80	0.3183	205	0.3373	405	0.3785	905	0.4393
31	0.2787	81	0.2790	206	0.2754	406	0.2810	906	0.2856

Fonte: Elaborada pelo autor (2025).

Observando os resultados, nota-se que para todas as instâncias, à medida que o número de usuários aumenta, a probabilidade de descarte aumenta, atingindo valores próximos de 1, o

que sinaliza saturação do sistema e filas cheias para o número especificado de usuários. Consequentemente, a taxa de descarte também cresce, aproximando-se da taxa máxima de chegada, o que demonstra que a maioria das requisições adicionais provavelmente será rejeitada.

Esse comportamento afeta diretamente a vazão do sistema, que começa em níveis altos (próximo a 120 requisições por unidade de tempo), mas declina à medida que o sistema se aproxima de sua capacidade máxima, caindo para valores próximos de zero em cenários de maior sobrecarga. Esses resultados demonstram que, embora as instâncias consigam lidar bem com cargas moderadas, elas rapidamente atingem seus limites operacionais e começam a rejeitar requisições.

Assim, o estudo destaca a importância de estratégias de escalabilidade ou balanceamento de carga para garantir um desempenho adequado e uma experiência satisfatória ao usuário diante de picos de demanda.

De forma geral, os resultados evidenciam que, ainda que as instâncias consigam operar de maneira aceitável sob cargas moderadas, o aumento exponencial de usuários impõe uma limitação previsível, tornando inevitável o descarte de requisições e a redução drástica da vazão. Esses achados reforçam a importância de um dimensionamento cuidadoso da infraestrutura e da adoção de políticas de escalabilidade automatizada para atender demandas dinâmicas de forma sustentável e garantir qualidade de serviço.

Ao final desta análise, é importante destacar algumas considerações adicionais acerca das hipóteses e simplificações adotadas. Nesta etapa, assumiu-se que as requisições geradas pelos usuários são homogêneas em termos de custo de processamento, isto é, possuem tempos médios de serviço equivalentes. Essa hipótese, embora coerente com o objetivo de avaliar o comportamento global do sistema sob diferentes configurações de instâncias virtuais, implica a abstração de uma característica relevante dos sistemas de gestão de aprendizagem: a heterogeneidade dos objetos de aprendizagem manipulados. Arquivos de vídeo, apresentações interativas e documentos PDF, por exemplo, demandam diferentes quantidades de processamento, memória e largura de banda, afetando de modo desigual os recursos da pilha LAMP. Considerar explicitamente essas classes de requisição representaria um refinamento importante do modelo, permitindo caracterizar cargas mistas e avaliar o impacto diferencial de cada tipo de conteúdo sobre o desempenho do sistema.

O modelo final proposto também representa o ambiente Moodle de forma agregada, tratando a pilha LAMP como um único bloco funcional. Essa escolha visou reduzir a complexidade estrutural inicial e viabilizar a análise comparativa entre instâncias com base em métricas de

utilização global. No entanto, reconhece-se que os elementos que compõem a pilha impactam de maneira distinta a latência e a capacidade de resposta do sistema. Uma possível extensão deste trabalho consiste, portanto, em estratificar o modelo em subcomponentes correspondentes a cada camada da pilha, de modo a capturar com maior precisão os gargalos específicos e analisar de forma mais granular a contribuição individual de cada serviço para o desempenho global do Moodle.

Além disso, observa-se que a análise de desempenho foi conduzida sob uma perspectiva sistêmica e generalista, com foco no comportamento agregado dos recursos computacionais. A incorporação de métricas específicas do LMS (como o número de acessos simultâneos por módulo, fórum, tarefa, questionário, etc ou os padrões de uso observados em períodos de pico) poderia proporcionar uma análise mais aderente ao comportamento real da aplicação. Tal abordagem permitiria calibrar o modelo a partir de traços de uso empíricos do Moodle, aprimorando a representatividade dos resultados e fortalecendo a conexão entre a modelagem e o contexto de uso educacional.

Por fim, ressalta-se que a validação empírica realizada concentrou-se no modelo básico e na métrica de disponibilidade estacionária, uma vez que esse cenário fornece a referência fundamental sobre a qual os modelos redundantes e virtualizados foram construídos. Essa decisão metodológica buscou assegurar a consistência interna dos resultados e a correção estrutural do modelo antes da aplicação de extensões mais complexas. A validação dos modelos avançados demandaria medições experimentais detalhadas, com instrumentação distribuída, monitoramento de falhas e correlação temporal de eventos entre múltiplas VMs. Embora tal processo extrapole o escopo desta pesquisa, constitui um desdobramento natural do trabalho, possibilitando a verificação empírica das métricas de performabilidade e confiabilidade sob diferentes níveis de virtualização e redundância.

7 CONCLUSÃO E TRABALHOS FUTUROS

Esta dissertação propôs-se a enfrentar o desafio de avaliar e otimizar quantitativamente a disponibilidade e o desempenho do ambiente de gestão de aprendizagem Moodle, uma plataforma de missão crítica para diversas instituições educacionais. Para tal, foi desenvolvido um framework de modelagem híbrido e hierárquico, que se mostrou capaz de capturar a complexidade de diferentes arquiteturas de implantação, desde instalações físicas básicas até ambientes elásticos em nuvem pública. Os objetivos específicos traçados no início deste trabalho foram alcançados, culminando na geração de modelos validados, na quantificação de métricas de interesse e na formulação de recomendações estratégicas alicerçadas na análise de instâncias em nuvem.

O ponto de partida foi o estabelecimento de uma baseline para uma arquitetura básica em hardware físico. A validação experimental, por meio de injeção de falhas, conferiu credibilidade ao modelo dentro de um intervalo de confiança de 95%, que estimou uma disponibilidade de 99,75%, correspondendo a um tempo de inatividade anual de quase 22 horas. A análise de sensibilidade subsequente foi importante, ao identificar a camada de aplicação (APP) como o componente mais crítico, indicando que os esforços de otimização deveriam se concentrar em sua taxa de falha e reparo para obter o maior impacto.

A primeira evolução arquitetural investigada foi a introdução de redundância física completa do sistema. Os resultados foram numericamente significativos, com uma melhoria de mais de 90% na redução do downtime, que caiu para 0,055 horas anuais.

A investigação da arquitetura virtualizada trouxe à tona uma das nuances mais importantes deste trabalho. Ao introduzir novas camadas de software na pilha de dependência, a complexidade do sistema aumentou e, paradoxalmente, a disponibilidade foi degradada, com o downtime anual subindo para 38,92 horas, um aumento de 77,8% em relação ao cenário base. Este resultado demonstrou que sua verdadeira força reside em sua capacidade de atuar como uma plataforma intermediária para técnicas avançadas de recuperação. De fato, a combinação de virtualização com redundância no nível do host superou a arquitetura básica, reduzindo o downtime anual em 44,82% , oferecendo uma solução equilibrada e viável.

Finalmente, a análise de desempenho em nuvem pública, utilizando um modelo SPN, demonstrou o trade-off fundamental entre os tipos de instâncias e o desempenho do sistema. Instâncias subdimensionadas saturaram rapidamente, levando a altas probabilidades de des-

carte e a uma queda abrupta da vazão. Este comportamento evidenciou uma limitação das políticas de escalonamento puramente reativas, apontando para a necessidade de abordagens mais inteligentes e preditivas para o gerenciamento de recursos.

7.1 PRINCIPAIS CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- Framework Analítico Híbrido Validado: A principal contribuição científica foi a proposição e validação empírica de uma metodologia que integrou, de forma hierárquica, o formalismo RBD. Este *framework* demonstrou ser robusto e flexível, capaz de avaliar conjuntamente a disponibilidade e o desempenho de sistemas complexos. Embora aplicado ao Moodle, sua natureza é genérica e pode ser adaptada para a análise de qualquer aplicação web multicamada, servindo como um guia metodológico para pesquisadores e profissionais da área.

- Diretrizes Quantitativas para Planejamento de Capacidade: O trabalho vai além de análises teóricas, ela fornecer um conjunto de dados comparativos que traduzem decisões arquitetônicas (como a adoção de redundância ou virtualização) em métricas de impacto de interesse para gestores e tomadores de decisão (por exemplo, horas de *downtime* por ano e vazão de requisições).

- Metodologia Experimental e Artefatos Reprodutíveis: A validação dos modelos teóricos por meio de um experimento prático de injeção de falhas conferiu um alto grau de credibilidade aos resultados obtidos. A metodologia detalhada e os *scripts* de injeção de falhas, disponibilizados como apêndice da dissertação, constituem um ativo valioso para a comunidade, permitindo a replicação, verificação e extensão desta pesquisa por outros pesquisadores.

Além das contribuições mencionadas, o artigo "Availability Evaluation of a Learning Management Environment" foi aceito no "1st Workshop on Resilience Engineering in Computer Systems" durante o LADC'24: 13th Latin–American Symposium on Dependable and Secure Computing.

Atualmente o artigo "Stochastic Model for the Performance of a Learning Management System" foi submetido ao LADC'25, encontrando-se sob avaliação.

7.2 LIMITAÇÕES E DIFICULDADES

Apesar das contribuições apresentadas, este trabalho enfrentou algumas limitações e desafios que merecem destaque.:

- Modelos de Falha e Reparo: A utilização de distribuições de probabilidade exponenciais para modelar os tempos de falha e de reparo foi uma premissa que viabilizou a análise por meio de Cadeias de Markov. No entanto, essa abordagem pode não retratar todas as realidades, pois falhas no mundo real podem seguir outras distribuições ou ocorrer em "rajadas", e o tempo de reparo pode depender da complexidade da falha, aspectos não capturados pelo modelo.

- Parametrização do Modelo de Desempenho em Nuvem: Os parâmetros de desempenho para as instâncias de nuvem (*AWS*), como a capacidade de usuários e os tempos de processamento, foram estimados com base na documentação oficial do Moodle e em ajustes proporcionais, em vez de medições empíricas diretas sob carga. Essa abordagem foi necessária para viabilizar a análise comparativa entre diferentes tipos de instância. Contudo, implica que os valores absolutos de desempenho (vazão, taxa de descarte) são uma aproximação. O desempenho em um ambiente de produção real pode variar, influenciado por fatores como a complexidade dos cursos e o comportamento específico dos usuários.

- Política de Escalonamento em Nuvem: A análise de desempenho em nuvem focou exclusivamente em políticas de escalonamento reativas, baseadas em limiares de utilização. Embora comuns em implementações padrão, os resultados sugeriram que elas podem apresentar problemas, especialmente para lidar com picos de carga súbitos

7.3 TRABALHOS FUTUROS

Com base nas limitações e descobertas desta pesquisa, delineiam-se direções para investigações subsequentes. Primeiramente, recomenda-se a expansão dos modelos para arquiteturas de microsserviços, investigando o impacto de containerização (*Docker/Kubernetes*) na disponibilidade do Moodle, com ênfase em falhas em cascata e orquestração de *clusters*.

Em segundo lugar, propõe-se a integração de análises econômicas aos formalismos de desempenho, avaliando *trade-offs* entre SLAs, custos operacionais e políticas de *auto-scaling* em ambientes de nuvem híbrida (AWS/Azure/GCP).

Um terceiro eixo envolve o desenvolvimento de mecanismos de tolerância a falhas adaptativa, utilizando aprendizado de máquina para prever falhas e ajustar redundâncias dinamicamente. Paralelamente, estudos sobre segurança cibernética poderiam incorporar ameaças (ex.: DDoS, injeção SQL) aos modelos SPN, quantificando seu impacto na disponibilidade.

Para validar a escalabilidade do *framework*, sugere-se sua aplicação em instituições de grande porte, coletando *datasets* reais de produção.

Por fim, a criação de *plugins* para ferramentas como *Zabbix* permitiria automatizar a coleta de métricas em tempo real, consolidando a ponte entre modelagem teórica e operação real.

REFERÊNCIAS

AKATSU, S.; MASUDA, A.; SHIDA, T.; TSUDA, K. A study of quality indicator model of large-scale open source software projects for adoption decision-making. *Procedia Computer Science*, Elsevier, v. 176, p. 3665–3672, 2020.

AL-AJLAN, A.; ZEDAN, H. Why moodle. In: IEEE. *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*. [S.l.], 2008. p. 58–64.

ALTINPULLUK, H.; KESIM, M. A systematic review of the tendencies in the use of learning management systems. *Turkish Online Journal of Distance Education*, Anadolu University, v. 22, n. 3, p. 40–54, 2021.

APACHE. *Apache, Apache HTTP Server Project*. 2024. <https://httpd.apache.org/ABOUT_APACHE.html>. Accessed: 2024-05-12.

ARAUJO, C. J. M. *Um Modelo de Avaliação de Desempenho de Estratégias de Gerência de Elasticidade de Acordo com o Comportamento da Carga de Trabalho*. Tese (Tese (Doutorado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2015.

ARAUJO, J.; ALVES, V.; OLIVEIRA, D.; DIAS, P.; SILVA, B.; MACIEL, P. An investigative approach to software aging in android applications. In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.: s.n.], 2013. p. 1229–1234.

ASSUNÇÃO, M. de; CARDONHA, C.; NETTO, M.; CUNHA, R. Impact of user patience on auto-scaling resource capacity for cloud services. *Future Generation Computer Systems*, v. 55, p. 41–50, 2016.

AUSTREGéSILO, M.; CALLOU, G. Stochastic models for optimizing availability, cost and sustainability of data center power architectures through genetic algorithm. *Revista de Informática Teórica e Aplicada*, v. 26, p. 27–44, 08 2019.

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.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. [S.l.]: ACM, 2003. p. 164–177.

BEZERRA, M. C. dos S. *Modelos para análise de disponibilidade de arquitetura de um serviço de Vod Streaming na nuvem*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2015.

BISWAS, A.; MAJUMDAR, S.; NANDY, B.; EL-HARAKI, A. An autoscaling framework for controlling enterprise resources on clouds. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. [S.l.]: IEEE, 2015. p. 971–980.

BORGES, I.; ANDRADE, E.; SILVA, F. A.; CALLOU, G. Availability evaluation of a video surveillance system with distributed storage. *Cluster Computing*, 2025. Aceito para publicação.

BüCHNER, A. *Moodle 4 Administration - Fourth Edition: An administrator's guide to configuring, securing, customizing, and extending Moodle*. Packt Publishing, 2022. Accessed 5/12/24. Disponível em: <https://subscription.packtpub.com/book/web-development/9781801816724/2/ch02lvl1sec08/understanding-the-moodle-architecture>.

CALLOU, G.; VIEIRA, M. Availability and performance analysis of cloud services. In: *13th Latin-American Symposium on Dependable and Secure Computing (LADC)*. [S.l.: s.n.], 2024.

CASALE, G.; CREMONESI, P. Performance models for hierarchical grid architectures. In: *2006 IEEE International Conference on Grid Computing (GRID)*. [S.l.]: IEEE, 2006. p. 232–239.

CATELANI, M.; CIANI, L.; VENZI, M. Reliability assessment for complex systems: A new approach based on rbd models. In: *2015 IEEE International Symposium on Systems Engineering (ISSE)*. [S.l.: s.n.], 2015. v. 0, n. 3, p. 286–290.

ČEPIN, M. Reliability block diagram. In: *Assessment of Power System Reliability*. [S.l.]: Springer, 2011. p. 119–123. ISBN 978-0-85729-687-0.

CIARDO, G.; GERMAN, R.; LINDEMANN, C. A characterization of the stochastic process underlying a stochastic petri net. *IEEE Transactions on Software Engineering*, v. 20, n. 7, p. 506–515, 1994.

CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J. G.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A. Live migration of virtual machines. In: *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)*. USENIX Association, 2005. p. 273–286. Disponível em: <https://www.usenix.org/legacy/events/nsdi05/tech/full_papers/clark/clark.pdf>.

CLEMENTE, D. A. M. *Modelagem hierárquica da disponibilidade de serviços hospedados em Centro de Dados*. Tese (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2022.

COMMUNITY, M. *About Moodle*. 2024. <https://docs.moodle.org/403/en/About_Moodle>. Accessed April 02, 2024.

COSTA, I. de O. *Modelos par análise de disponibilidade em uma plataforma de mobile backend as a service*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2015.

DANTAS, J.; LIMA, C.; MACIEL, P.; SOUZA, J.; LIMA, R. Availability evaluation of a transaction-oriented system using a fault injection tool. In: *International Conference on Software Engineering and Knowledge Engineering (SEKE)*. [S.l.: s.n.], 2011. p. 306–311.

DANTAS, J.; MATOS, R.; ARAUJO, J.; MACIEL, P. An availability model for eucalyptus platform: an analysis of warm-standy replication mechanism. *ACM. IEEE Computer Society*, v. 3, n. 3, p. 1664–1669, 2012.

DANTAS, J. L. C. *Um Ambiente para Avaliação de Desempenho de Sistemas Web Clusterizados com Suporte a Replicação de Conteúdo*. Tese (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2008.

DEVORE, J. L. *Probability and statistics for engineering and the sciences*. [S.l.]: Spinger, 2008.

DIAS, F. *A Hierarchical Model for Performance and Availability Evaluation of IaaS Cloud Environments*. Tese (Tese (Doutorado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2017.

FE, I.; MATOS, R.; DANTAS, J.; MELO, C.; MACIEL, P. Stochastic model of performance and cost for auto-scaling planning in public cloud. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2017. p. 2081–2086.

FEITOSA, L.; BARBOSA, V.; SABINO, A.; LIMA, L. N.; FE, I.; SILVA, L. G.; CALLOU, G.; CARVALHO, J. O.; LEAO, E. M.; NGUYEN, T. A.; REGO, P. A. L.; SILVA, F. A. P. A. A comprehensive performance evaluation of container migration strategies. *Computing*, 2025. Aceito para publicação.

FRANK, P. M. *Introduction to System Sensitivity Theory*. [S.l.]: Academic Press Inc., 1978.

Fé I., M. R.; DANTAS, J.; MELO, C.; ARAUJO, J.; MACIEL, P. Performance-cost trade-off in auto-scaling mechanisms for cloud computing. *Sensors*, v. 22, n. 3, p. 1221, 2022.

GANDHI, A.; DUBE, P.; KARVE, A.; KOCHUT, A.; ZHANG, L. Adaptive, model-driven autoscaling for cloud applications. In: *2014 IEEE International Conference on Autonomic Computing (ICAC)*. [S.l.]: IEEE, 2014. p. 157–164.

GIRVIN, D. *What is Apache? In-Depth Overview of Apache Web Server*. 2025. <https://www.sumologic.com/blog/apache-web-server-introduction>. Accessed: 2025-04-02.

GOMES, A.; CALLOU, G. Models for availability evaluation of file servers in private clouds. *Computing*, 2024.

GONÇALVES, C. F.; ANDRADE, E.; MENDONÇA, J.; CALLOU, G. Modelos estocásticos para o planejamento de ambientes avas baseados em contêineres e máquinas virtuais. *Revista de Informática Teórica e Aplicada: RITA*, v. 29, n. 2, 2022.

GONZAGA, C. *Um Modelo para Avaliação de Desempenho e Disponibilidade de Aplicações de Software como Serviço (SaaS)*. Tese (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2014.

GUPTA, A.; CHRISTIE, R.; MANJULA, P. Scalability in internet of things: features, techniques and research challenges. *Int. J. Comput. Intell. Res*, v. 13, n. 7, p. 1617–1627, 2017.

HAAN, K. *Best Learning Management Systems (LMS) Of 2024*. 2024. <https://www.forbes.com/advisor/business/best-learning-management-systems/>. Accessed: 2024-05-12.

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

HERBST, N. R.; KOUGIOUKOTAS, S.; REMANN, R. Elasticity in cloud computing: What it is, and what it is not. In: *10th International Conference on Autonomic Computing (ICAC)*. [S.l.]: USENIX, 2013. (Anais...).

HERBST, N. R.; KOUNEV, S.; REUSSNER, R. Elasticity in cloud computing: What it is, and what it is not. In: *Proceedings of the 10th International Conference on Utility and Cloud Computing (UCC)*. [S.l.]: ACM, 2013. p. 23–32.

HUNG, C.; HU, Y.; LI, K. Auto-scaling model for cloud computing system. *International Journal of Hybrid Information Technology*, v. 5, n. 2, p. 181–186, 2012.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, Inc., 1991.

JOGI, V. D.; SINHA, A. Performance evaluation of mysql, cassandra and hbase for heavy write operation. In: *3rd Int'l Conf. on Recent Advances in Information Technology (RAIT-2016)*. Dhanbad, India: [s.n.], 2016.

KEESEE, W. R. *A Method of Determining a Confidence Interval for Availability*. 1965. Point Mugu, California: Miscellaneous Publication.

KOCH, J.; HAO, W. Apache and http/2 in the cloud: A comparative study of apache architecture in aws. In: *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON 2021)*. Vancouver, BC, Canada: [s.n.], 2021. p. 673–680.

KURIEN, A. T. J.; MATHEW, S. A.; MANA, S. C. Development of php and mysql based digital asset management system for secure organizations. In: *2022 6th International Conference on Trends in Electronics and Informatics, ICOEI 2022 - Proceedings*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2022. p. 1859–1863. ISBN 9781665483285.

LAPRIE, J.-C. Dependable computing and fault tolerance: Concepts terminology. In: *Fault-Tolerant Computing, Twenty-Fifth International Symposium on*. [S.l.]: IEEE, 1995. p. 2–11.

LEONARDO, W.; BEZERRA, T.; CALLOU, G. Stochastic petri net models for availability and performance evaluation of nextcloud service hosted in apache cloudstack. In: *13th Latin-American Symposium on Dependable and Secure Computing (LADC)*. [S.l.: s.n.], 2024.

LEONARDO, W.; CALLOU, G. Avaliação da disponibilidade do serviço nextcloud hospedado em nuvem privada. In: *Workshop de Testes e Tolerância a Falhas (WTF)*. [S.l.: s.n.], 2025. Artigo submetido ao Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2025).

LIAW, S. S.; HUANG, H. M. Perceived satisfaction, perceived usefulness and interactive learning environments as predictors to self-regulation in e-learning environments. *Computers Education*, v. 60, n. 1, 2013.

LIMA, C.; GOMES, A.; ANDRADE, E.; CALLOU, G. Avaliação de desempenho e consumo de energia de um ambiente virtual de aprendizagem em nuvens privadas. *Revista Brasileira de Computação Aplicada*, v. 13, n. 1, p. 74–87, 2021.

LIMA, C. J.; GOMES, A.; ANDRADE, E.; CALLOU, G. Avaliação de desempenho e consumo de energia do ambiente moodle. *Research, Society and Development*, v. 10, n. 5, 2021.

LIMA, C. M. B. *Um Modelo para Avaliação de Desempenho, Custo e Disponibilidade de Ambientes de Nuvens Híbridas*. Tese (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2015.

LIU, Y.; WU, W. Research on the system of reliability block diagram design and reliability prediction. In: *2011 Int. Conf. Syst. Sci. Eng. Des. Manuf. Informatiz. ICSEM 2011*. [S.l.: s.n.], 2011. v. 2, p. 35–38.

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

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

MACIEL, R.; ARAUJO, J.; DANTAS, J.; MELO, C.; GUEDES, E.; MACIEL, P. Impact of a ddos attack on computer systems: An approach based on an attack tree model. In: *2018 Annual IEEE International Systems Conference (SysCon)*. [S.l.: s.n.], 2018. p. 1–8.

MANVI, S. S.; SHYAM, G. K. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of network and computer applications*, Elsevier, v. 41, p. 424–440, 2014.

MARSAN, M. A.; CONTE, G.; BALBO, G. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 2, n. 2, p. 93–122, 1984.

MARWAH, M.; MACIEL, P.; SHAH, A.; SHARMA, R.; CHRISTIAN, T.; ALMEIDA, V. Quantifying the sustainability impact of data center availability. *SIGMETRICS Perform. Eval. Rev.*, New York, NY, USA, v. 37, n. 4, p. 64–68, 2010.

MELL, P.; GRANCE, T. *The nist definition of cloud computing (nist special publication 800-145)*. [S.l.], 2011.

MELO, C. *Um Modelo Hierárquico para Avaliação de Disponibilidade e Desempenho Orientado à Capacidade de Ambientes de Nuvem Privada*. Tese (Tese (Doutorado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2018.

MELO, C. A. S. *Um Modelo de Avaliação de Desempenho e Disponibilidade de Bancos de Dados Relacionais em Ambientes de Nuvens*. Tese (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2016.

MELO, C. A. S. d. *Avaliação da Disponibilidade de Infraestrutura de Sincronização de Dados*. Dissertação (Dissertação de Mestrado) — Universidade Federal de Pernambuco, Centro de Informática, Recife, 2016.

MENASCé, D. A.; ALMEIDA, V. A. F. d.; DOWDY, L. W. *Performance by design: computer capacity planning by example*. [S.l.]: Prentice Hall PTR, 2002.

MENDONçA, J.; LIMA, R.; MATOS, R.; FERREIRA, J.; ANDRADE, E. Availability analysis of a disaster recovery solution through stochastic models and fault injection experiments. In: *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. [S.l.]: IEEE, 2018. p. 135–142.

MOLLOY, M. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, C-31, n. 9, p. 913–917, 1982.

MOODLE. *About Moodle*. 2020. <https://docs.moodle.org/38/en/About_Moodle>. Disponível em: <https://docs.moodle.org/38/en/About_Moodle>.

Moodle Community. *Installing Moodle*. 2024. <https://docs.moodle.org/404/en/Installing_Moodle>. Accessed: May 12, 2024.

MORAIS, I. d. et al. Um modelo para avaliação de desempenho de web service em nuvem no consumo de imagens em dispositivos móveis. In: *Anais do XXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação, 2013.

MySQL. *What is MySQL?* 2024. <https://www.oracle.com/mysql/what-is-mysql/>. Accessed: May 12, 2024.

NARCISO, J. *Principais vantagens do linux e porque muitas empresas adotam esse sistema!* 2023. <https://www.dio.me/articles/principais-vantagens-do-linux-e-porque-muitas-empresas-adotam-esse-sistema>. Accessed: 2024-05-12.

NETCRAFT. *Netcraft, February 2024 Web Server Survey*. 2024. <https://www.netcraft.com/blog/february-2024-web-server-survey>. Accessed: 2024-05-12.

OROZCO, A. M. S. *BALANCEAMENTO ENTRE SEGURANÇA E DESEMPENHO NA COMUNICAÇÃO ENTRE OS PLANOS DE CONTROLE E DADOS EM REDES DEFINIDAS POR SOFTWARE*. Tese (Tese (Doutorado)) — Pontifícia Universidade Católica do Rio Grande do Sul, 2018.

PADALA, P.; HOU, K.; SHIN, K. Automated control of multiple virtualized resources. In: *Proceedings of the 4th ACM European conference on Computer systems*. [S.l.]: ACM, 2009. p. 43–56.

PHP. *What is PHP?* 2024. <https://www.php.net/manual/en/intro-whatis.php>. Accessed: May 12, 2024.

POPEK, G.; GOLDBERG, R. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, v. 17, n. 7, p. 412–421, 1974.

PRANAM, A. Understand the software stack. In: *Product Management Essentials*. Berkeley, CA: Apress, 2018. p. 23–37.

PYPL. *PYPL index. PopularitY of Programming Language*. 2024. <https://pypl.github.io/PYPL.html>. Accessed: May 12, 2024.

SAILELLAH, H. R. P. *Linux Operating System: History, Functions, Advantages, and Disadvantages*. 2023. <https://it.telkomuniversity.ac.id/en/linux-operating-system/>. Accessed: 2024-05-12.

SCHAFFER, S. Babbage's intelligence: Calculating engines and the factory system. *Critical Inquiry*, The University of Chicago Press, 1994.

SILVA, A. da; CALLOU, G.; VALENTIM, T.; DANTAS, J. Análise de desempenho de servidores de arquivos em nuvem privada. In: *Seminário Integrado de Software e Hardware (SEMISH)*. [S.l.: s.n.], 2023. Parte do XLIII Congresso da Sociedade Brasileira de Computação (CSBC).

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: *INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORK CONFERENCE (DSN 2015)*. [S.l.: s.n.], 2015.

SILVA, F. A.; ANTONIO, J. M.; RAFAEL, J.; C., J.; CALLOU, G. Availability assessment and sensitivity analysis of an mbaas platform. *International Journal of Computational Science and Engineering (IJCSE)*, 2024.

SOUZA, E. T. G. de. *Avaliação do impacto de uma política de manutenção na performabilidade de sistemas de transferência eletrônica de fundos*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2009.

SYED, M. *Apache2, PHP8.1, MariaDB (LAMP Stack) on Ubuntu 22.04*. 2022. Accessed 5/12/24. Disponível em: <https://marsown.com/wordpress/install-lamp-stack-ubuntu-20-04/>.

TEAM, E. L. *História do LMS*. 2024. <https://www.easy-lms.com/pt/centro-de-conhecimento/centro-lms/historia-do-lms/item10401>. Accessed: 2024-12-05.

TEIXEIRA, J. C. *Uma Abordagem para Avaliação de Desempenho e Disponibilidade de Sistemas de Bancos de Dados NoSQL em Ambientes de Nuvem Híbrida*. Tese (Tese (Doutorado em Ciência da Computação)) — Universidade Federal de Pernambuco, Recife, 2017.

TESAR, M. Towards a post-covid-19 'new normality?': Towards a post-covid-19 'new normality?': Physical and social distancing, the move to online and higher education. *Policy Futures in Education*, v. 18, n. 5, p. 558–563, 2020.

TRINDADE, F. R. *Predição de Desempenho no Moodle usando Princípios da Andragogia*. Tese (Dissertação (Mestrado em Computação)) — Universidade Federal de Goiás, Goiânia, 2020.

WALDSPURGER, C. Memory resource management in vmware esx server. In: *Proceedings of the 5th symposium on Operating Systems Design and Implementation*. [S.l.]: USENIX Association, 2002. p. 181–194.

WANNAPIROON, P.; KAEWRATTANAPAT, N.; PREMSMITH, J. Development of cloud learning management systems for higher education institutions. In: *2019 Research, Invention, and Innovation Congress (RI2C 2019)*. Bangkok, Thailand: [s.n.], 2019.

APÊNDICE A – SCRIPT PARA INJEÇÃO DE FALHA E REPARO NO HARDWARE

Script para injetar falhas e reparos no hardware

```
#!/usr/bin/python3

import subprocess
import time
from datetime import datetime
import numpy as np

def falha(mttf):
    tempo_falha = np.random.exponential(mttf, 1)[0]
    time.sleep(tempo_falha)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Chama o script de falha.py (certifique-se de fornecer o caminho
       correto se n o estiver na mesma pasta)
    subprocess.run(["python", "hardware_desliga.py"])

    return tempo_falha, timestamp

def reparo(mttr):
    tempo_reparo = np.random.exponential(mttr, 1)[0]
    time.sleep(tempo_reparo)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Chama o script de reparo.py (certifique-se de fornecer o caminho
       correto se n o estiver na mesma pasta)
    subprocess.run(["python", "hardware_religa.py"])

    return tempo_reparo, timestamp

def main():
    mttf = 8.760*3600  # Mean Time To Failure in seconds
    mttr = 1.67*3600  # Mean Time To Repair in seconds
    log_file = "fault_log.txt"

```

```
    try:
        while True:
            # Simula falha
            falha_tempo, falha_timestamp = falha(mttf)
            with open(log_file, "a") as log:
                log.write(f"Hardware Failure - MTTF: {falha_tempo:.2f} -
                    Timestamp: {falha_timestamp}\n")

            # Simula reparo
            reparo_tempo, reparo_timestamp = reparo(mttr)
            with open(log_file, "a") as log:
                log.write(f"Hardware Repair - MTTR: {reparo_tempo:.2f} -
                    Timestamp: {reparo_timestamp}\n")

    except KeyboardInterrupt:
        print("Simula  o interrompida pelo usu rio.")

if __name__ == "__main__":
    main()
```

Código Fonte 1 – Código Python para injetar falhas e reparos no hardware

Script para desligar o hardware (falha)

```
import winrm

def desligar_maquina_fisica(ip_machine, usuario, senha):
    winrm_session = winrm.Session(
        f'http://{ip_machine}:5985/wsman',
        auth=(usuario, senha),
        server_cert_validation='ignore'
    )

    script_path = r'C:\ScriptsHyperV\Shutdown-Remote.ps1'  # Coloque o
        caminho correto do script
    comando_ps = f'powershell -File {script_path} -ComputerName {
        ip_machine}'

    resultado = winrm_session.run_ps(comando_ps)

    if resultado.status_code == 0:
```

```
16          print(f"A m quina '{ip_machine}' foi desligada com sucesso.")
17      else:
18          print(f"Erro ao desligar a m quina '{ip_machine}': {resultado.
                std_err.decode()}")
19
20  # Configura  es de IPMI da m quina f sica
21  ip_machine = '10.255.255.40'
22  usuario_machine = 'administrador'
23  senha_machine = '@vfesc87!'
24
25  # Chamar a fun  o para desligar a m quina f sica
26  desligar_maquina_fisica(ip_machine, usuario_machine, senha_machine)
```

Código Fonte 2 – Código Python para desligar o hardware

Script para religar o hardware (reparo)

```
1   from wakeonlan import send_magic_packet
2   from ping3 import ping, verbose_ping
3   import time
4
5   def verificar_status_da_maquina(ip, tempo_espera=10):
6       try:
7           resposta_ping = ping(ip, timeout=tempo_espera)
8           return resposta_ping is not None
9       except Exception as e:
10          print(f"Erro ao verificar status da m quina: {e}")
11          return False
12
13  def ligar_maquina_remotamente(mac_address, ip_address):
14      if verificar_status_da_maquina(ip_address):
15          print(f"A m quina com o endere o IP '{ip_address}' j  est
                ligada.")
16      else:
17          try:
18              send_magic_packet(mac_address)
19              print(f"Pacote Wake-on-LAN enviado com sucesso para '{
                    mac_address}'.")
20          except Exception as e:
21              print(f"Erro ao enviar pacote Wake-on-LAN: {e}")
22
```

```
23              # Aguarde alguns segundos antes de verificar o status
                    novamente
24              time.sleep(5)
25
26              if verificar_status_da_maquina(ip_address):
27                  print(f"A m quina com o endere o IP '{ip_address}' foi
                        ligada ap s o erro no envio do pacote.")
28              else:
29                  print(f"A m quina com o endere o IP '{ip_address}'
                        ainda est  desligada.")
30
31  # Endere o MAC da placa de rede da m quina
32  mac_address = 'B8:97:5A:8A:34:93'
33  # Endere o IP da m quina
34  ip_address = '10.255.255.40'
35
36  # Chamar a fun  o para enviar o pacote WoL
37  ligar_maquina_remotamente(mac_address, ip_address)
```

Código Fonte 3 – Código Python para religar o hardware

APÊNDICE B – SCRIPT PARA INJEÇÃO DE FALHA E REPARO NO SISTEMA OPERACIONAL

Script para injetar falhas e reparos no Sistema Operacional

```
#!/usr/bin/python3

import subprocess
import time
from datetime import datetime
import numpy as np

def falha(mttf):
    tempo_falha = np.random.exponential(mttf, 1)[0]
    time.sleep(tempo_falha)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Chama o script de falha.py (certifique-se de fornecer o caminho
       correto se n o estiver na mesma pasta)
    subprocess.run(["python", "os_desliga.py"])

    return tempo_falha, timestamp

def reparo(mttr):
    tempo_reparo = np.random.exponential(mttr, 1)[0]
    time.sleep(tempo_reparo)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Chama o script de reparo.py (certifique-se de fornecer o caminho
       correto se n o estiver na mesma pasta)
    subprocess.run(["python", "os_inicia.py"])

    return tempo_reparo, timestamp

def main():
    mttf = 2.8*3600  # Mean Time To Failure in seconds
    mttr = 1*3600  # Mean Time To Repair in seconds
    log_file = "fault_log.txt"

```

```
    try:
        while True:
            # Simula falha
            falha_tempo, falha_timestamp = falha(mttf)
            with open(log_file, "a") as log:
                log.write(f"OS Failure - MTTF: {falha_tempo:.2f} -
                    Timestamp: {falha_timestamp}\n")

            # Simula reparo
            reparo_tempo, reparo_timestamp = reparo(mttr)
            with open(log_file, "a") as log:
                log.write(f"OS Repair - MTTR: {reparo_tempo:.2f} -
                    Timestamp: {reparo_timestamp}\n")

    except KeyboardInterrupt:
        print("Simula  o interrompida pelo usu rio.")

if __name__ == "__main__":
    main()
```

Código Fonte 4 – Código Python para injetar falhas e reparos no Sistema Operacional

Script para desligar o Sistema Operacional (falha)

```
import winrm
from winrm.protocol import Protocol

def pausar_vm(nome_vm, ip_hyper_v, usuario, senha):
    # Configurar a conex o WinRM
    winrm_session = winrm.Session(
        f'http://{ip_hyper_v}:5985/wsman',
        auth=(usuario, senha),
        server_cert_validation='ignore'  # Ajuste isso conforme
            necess rio para ambientes de produ  o
    )

    # Caminho do script no servidor remoto
    script_path = r'C:\ScriptsHyperV\Pause-VM-Remote.ps1'

    # Comando PowerShell para executar o script remoto
    comando_ps = f'powershell -File {script_path} -VMName {nome_vm}'
```

```
17
18      # Executar o comando via WinRM
19      resultado = winrm_session.run_ps(comando_ps)
20
21      # Verificar o resultado
22      if resultado.status_code == 0:
23          print(f"A VM '{nome_vm}' foi pausada com sucesso.")
24      else:
25          print(f"Erro ao pausar a VM '{nome_vm}': {resultado.std_err.
                decode()}")
26
27  # Nome da VM
28  nome_da_vm = 'UbuntuServer-CInMoodle'
29
30  # IP do Hyper-V
31  ip_hyper_v = '10.255.255.40'
32
33  # Credenciais do Hyper-V
34  usuario_hyper_v = 'administrador'
35  senha_hyper_v = '@vfesc87!'
36
37  pausar_vm(nome_da_vm, ip_hyper_v, usuario_hyper_v, senha_hyper_v)
```

Código Fonte 5 – Código Python para desligar o Sistema Operacional

Script para religar o Sistema Operacional (reparo)

```
1   import winrm
2   from winrm.protocol import Protocol
3
4   def iniciar_vm(nome_vm, ip_hyper_v, usuario, senha):
5       winrm_session = winrm.Session(
6           f'http://{ip_hyper_v}:5985/wsman',
7           auth=(usuario, senha),
8           server_cert_validation='ignore'
9       )
10
11      script_path = r'C:\ScriptsHyperV\Start-VM-Remote.ps1'
12      comando_ps = f'powershell -File {script_path} -VMName {nome_vm}'
13
14      resultado = winrm_session.run_ps(comando_ps)
```

```
15
16      if resultado.status_code == 0:
17          print(f"A VM '{nome_vm}' foi iniciada com sucesso.")
18      else:
19          print(f"Erro ao iniciar a VM '{nome_vm}': {resultado.std_err.
                decode()}")
20
21  # Nome da VM
22  nome_da_vm = 'UbuntuServer-CInMoodle'
23
24  # IP do Hyper-V
25  ip_hyper_v = '10.255.255.40'
26
27  # Credenciais do Hyper-V
28  usuario_hyper_v = 'administrador'
29  senha_hyper_v = '@vfesc87!'
30
31  iniciar_vm(nome_da_vm, ip_hyper_v, usuario_hyper_v, senha_hyper_v)
```

Código Fonte 6 – Código Python para religar o Sistema Operacional

APÊNDICE C – SCRIPT PARA INJEÇÃO DE FALHA E REPARO NO APACHE

```
#!/usr/bin/python3

import numpy as np
import time
from datetime import datetime
import paramiko

def falha(mttr, ssh_client):
    tempo_falha = np.random.exponential(mttr, 1)[0]
    time.sleep(tempo_falha)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula falha no servidor remoto (substitua o comando conforme
        necess rio)
    comando_falha = "sudo service apache2 stop"
    stdin, stdout, stderr = ssh_client.exec_command(comando_falha)

    return tempo_falha, timestamp

def reparo(mttf, ssh_client):
    tempo_reparo = np.random.exponential(mttf, 1)[0]
    time.sleep(tempo_reparo)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula reparo no servidor remoto (substitua o comando conforme
        necess rio)
    comando_reparo = "sudo service apache2 start"
    stdin, stdout, stderr = ssh_client.exec_command(comando_reparo)

    return tempo_reparo, timestamp

def conectar_ssh(host, usuario, senha):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(host, username=usuario, password=senha)
    return ssh

```

```
36 def main():
37     host_remoto = "10.255.255.41"
38     usuario_ssh = "gervasio"
39     senha_ssh = "123456"
40 
41     mttf = 0.7884*3600  # Mean Time To Failure
42     mttr = 0.5*3600  # Mean Time To Repair in seconds
43     log_file = "fault_log.txt"
44 
45     ssh_cliente = conectar_ssh(host_remoto, usuario_ssh, senha_ssh)
46 
47     try:
48         while True:
49             # Simula falha
50             falha_tempo, falha_timestamp = falha(mttf, ssh_cliente)
51             with open(log_file, "a") as log:
52                 log.write(f"Apache Failure - MTTF: {falha_tempo:.2f} -
                       Timestamp: {falha_timestamp}\n")
53 
54             # Simula reparo
55             reparo_tempo, reparo_timestamp = reparo(mttr, ssh_cliente)
56             with open(log_file, "a") as log:
57                 log.write(f"Apache Repair - MTTR: {reparo_tempo:.2f} -
                       Timestamp: {reparo_timestamp}\n")
58 
59     except KeyboardInterrupt:
60         print("Simula  o interrompida pelo usu rio.")
61         ssh_cliente.close()
62 
63 if __name__ == "__main__":
64     main()
```

Código Fonte 7 – Código Python para injetar falhas e reparos no Apache

APÊNDICE D – SCRIPT PARA INJEÇÃO DE FALHA E REPARO NO MYSQL

```
#!/usr/bin/python3

import numpy as np
import time
from datetime import datetime
import paramiko

def falha(mttr, ssh_client):
    tempo_falha = np.random.exponential(mttr, 1)[0]
    time.sleep(tempo_falha)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula falha no servidor remoto (substitua o comando conforme
        necess rio)
    comando_falha = "sudo service mysql stop"
    stdin, stdout, stderr = ssh_client.exec_command(comando_falha)

    return tempo_falha, timestamp

def reparo(mttf, ssh_client):
    tempo_reparo = np.random.exponential(mttf, 1)[0]
    time.sleep(tempo_reparo)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula reparo no servidor remoto (substitua o comando conforme
        necess rio)
    comando_reparo = "sudo service mysql start"
    stdin, stdout, stderr = ssh_client.exec_command(comando_reparo)

    return tempo_reparo, timestamp

def conectar_ssh(host, usuario, senha):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(host, username=usuario, password=senha)
    return ssh

```

```
36 def main():
37     host_remoto = "10.255.255.41"
38     usuario_ssh = "gervasio"
39     senha_ssh = "123456"
40 
41     mttf = 1.4*3600  # Mean Time To Failure
42     mttr = 1*3600  # Mean Time To Repair
43     log_file = "fault_log.txt"
44 
45     ssh_cliente = conectar_ssh(host_remoto, usuario_ssh, senha_ssh)
46 
47     try:
48         while True:
49             # Simula falha
50             falha_tempo, falha_timestamp = falha(mttf, ssh_cliente)
51             with open(log_file, "a") as log:
52                 log.write(f"MySQL Failure - MTTF: {falha_tempo:.2f} -
                       Timestamp: {falha_timestamp}\n")
53 
54             # Simula reparo
55             reparo_tempo, reparo_timestamp = reparo(mttr, ssh_cliente)
56             with open(log_file, "a") as log:
57                 log.write(f"MySQL Repair - MTTR: {reparo_tempo:.2f} -
                       Timestamp: {reparo_timestamp}\n")
58 
59     except KeyboardInterrupt:
60         print("Simula  o interrompida pelo usu rio.")
61         ssh_cliente.close()
62 
63 if __name__ == "__main__":
64     main()
```

Código Fonte 8 – Código Python para injetar falhas e reparos no MySQL

APÊNDICE E – SCRIPT PARA INJEÇÃO DE FALHA E REPARO NO PHP

```python
#!/usr/bin/python3

import numpy as np
import time
from datetime import datetime
import paramiko

def falha(mttr, ssh_client):
    tempo_falha = np.random.exponential(mttr, 1)[0]
    time.sleep(tempo_falha)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula falha no servidor remoto (substitua o comando conforme
        necess rio)
    comando_falha = "sudo service php8.1-fpm stop"
    stdin, stdout, stderr = ssh_client.exec_command(comando_falha)

    return tempo_falha, timestamp

def reparo(mttf, ssh_client):
    tempo_reparo = np.random.exponential(mttf, 1)[0]
    time.sleep(tempo_reparo)
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f")[:-3]

    # Simula reparo no servidor remoto (substitua o comando conforme
        necess rio)
    comando_reparo = "sudo service php8.1-fpm start"
    stdin, stdout, stderr = ssh_client.exec_command(comando_reparo)

    return tempo_reparo, timestamp

def conectar_ssh(host, usuario, senha):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(host, username=usuario, password=senha)
    return ssh

```

```
36 def main():
37     host_remoto = "10.255.255.41"
38     usuario_ssh = "gervasio"
39     senha_ssh = "123456"
40 
41     mttf = 0.7884*3600  # Mean Time To Failure
42     mttr = 0.5*3600  # Mean Time To Repair
43     log_file = "fault_log.txt"
44 
45     ssh_cliente = conectar_ssh(host_remoto, usuario_ssh, senha_ssh)
46 
47     try:
48         while True:
49             # Simula falha
50             falha_tempo, falha_timestamp = falha(mttf, ssh_cliente)
51             with open(log_file, "a") as log:
52                 log.write(f"PHP Failure - MTTF: {falha_tempo:.2f} -
                       Timestamp: {falha_timestamp}\n")
53 
54             # Simula reparo
55             reparo_tempo, reparo_timestamp = reparo(mttr, ssh_cliente)
56             with open(log_file, "a") as log:
57                 log.write(f"PHP Repair - MTTR: {reparo_tempo:.2f} -
                       Timestamp: {reparo_timestamp}\n")
58 
59     except KeyboardInterrupt:
60         print("Simula  o interrompida pelo usu rio.")
61         ssh_cliente.close()
62 
63 if __name__ == "__main__":
64     main()
```

Código Fonte 9 – Código Python para injetar falhas e reparos no PHP

APÊNDICE F – SCRIPT PARA MONITORAR O SISTEMA E SALVAR O ESTADO "UP"OU "DOWN"EM ARQUIVO DE LOG

```
import paramiko
import os
import subprocess
import datetime
import time
import mysql.connector
from ping3 import ping
import threading  # Certifique-se de importar o m dulo threading

# Configura  es do servidor MySQL
mysql_server_ip = "10.255.255.41"
mysql_port = 3306

# Configura  es do servidor Apache
apache_server_ip = "10.255.255.41"
apache_log_file = "apache_status.log"

# Configura  es do servidor PHP
php_server_ip = "10.255.255.41"
php_log_file = "php_status.log"

# Configura  es do servidor MySQL
mysql_log_file = "mysql_status.log"

# Configura  es para acesso SSH
ssh_username = "gervasio"
ssh_password = "123456"

# C digos ANSI para controle de cores
GREEN = '\033[92m'
RED = '\033[91m'
ENDC = '\033[0m'

# Configura  es do servidor hardware
hardware_server_ip = "10.255.255.40"
```

```
hardware_log_file = "hardware_status.log"

# Configura  es do servidor SO
os_server_ip = "10.255.255.41"
os_log_file = "os_status.log"

def check_ping(server_ip, service_name):
    try:
        response = ping(server_ip, timeout=1, unit="ms")
        return response is not None
    except Exception as e:
        print(f"Erro ao verificar ping para {service_name}: {e}")
        return False

def check_mysql_service(log_file, ssh_client=None):
    try:
        # Se um cliente SSH for fornecido, executa o comando remotamente
        if ssh_client:
            command = "service mysql status"  # Use o comando apropriado
                para verificar o status do MySQL remotamente
            stdin, stdout, stderr = ssh_client.exec_command(command)
            result = stdout.channel.recv_exit_status()
            return result == 0
        else:
            # Tenta conectar ao MySQL usando o m dulo mysql-connector-
                python
            conn = mysql.connector.connect(
                host=mysql_server_ip,
                port=mysql_port,
                user='moodleuser',
                password='Oz83.*5Fm6Mj0Hd9',
                database='moodle'
            )
            conn.close()
            return True
    except Exception as e:
        # Se houver uma exce  o , o servi o est  indispon vel
        log_failure(log_file, "MySQL")
        return False
```

```
73
74 def check_apache_service(log_file, ssh_client=None):
75     try:
76         # Se um cliente SSH for fornecido, executa o comando remotamente
77         if ssh_client:
78             command = "pgrep -f apache2"
79             stdin, stdout, stderr = ssh_client.exec_command(command)
80             result = stdout.channel.recv_exit_status()
81             return result == 0
82         else:
83             # Verifica se o processo do Apache est  em execu  o
84             subprocess.check_output(["pgrep", "-f", "apache2"])
85             return True
86     except subprocess.CalledProcessError as e:
87         # Se houver uma exce  o , o servi o est  indispon vel
88         log_failure(log_file, "Apache")
89         return False
90
91 def check_php_service(log_file, ssh_client=None):
92     try:
93         # Se um cliente SSH for fornecido, executa o comando remotamente
94         if ssh_client:
95             command = "pgrep -f php"
96             stdin, stdout, stderr = ssh_client.exec_command(command)
97             result = stdout.channel.recv_exit_status()
98             return result == 0
99         else:
100            # Substitua este bloco de c digo conforme necess rio para
                verificar o status do PHP
101            # Aqui estamos apenas imitando uma verifica  o bem-sucedida
102            # usando a fun  o subprocess.check_output para n o gerar
                exce  es.
103            subprocess.check_output(["echo", "PHP is running"])
104            return True
105    except subprocess.CalledProcessError as e:
106        # Se houver uma exce  o , o servi o est  indispon vel
107        log_failure(log_file, "PHP")
108        return False
109
```

```
110 def check_hardware(log_file):
111     try:
112         hardware_reachable = check_ping(hardware_server_ip, "Hardware")
113         if not hardware_reachable:
114             log_failure(log_file, "Hardware")
115         else:
116             log_recovery(log_file, "Hardware")
117     except Exception as e:
118         print(f"Erro ao monitorar hardware: {e}")
119 
120 def check_os(log_file):
121     try:
122         os_reachable = check_ping(os_server_ip, "OS")
123         if not os_reachable:
124             log_failure(log_file, "OS")
125         else:
126             log_recovery(log_file, "OS")
127     except Exception as e:
128         print(f"Erro ao monitorar sistema operacional: {e}")
129 
130 def log_failure(log_file, service_name):
131     current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f
        ")[:-3]
132     new_entry = f"{service_name} - Not Reachable at {current_time}"
133 
134     # Verifica se a nova entrada j  est  no arquivo de log
135     with open(log_file, "r") as file:
136         if new_entry in file.read():
137             print(f"{service_name} - Not Reachable already logged.")
138             return
139 
140     # Se n o estiver, adiciona a nova entrada
141     with open(log_file, "a") as file:
142         file.write(f"{RED}{new_entry}{ENDC}\n")
143         print(f"{service_name} - Not Reachable. Logged to {log_file} at {
            current_time}")
144 
145 def log_failure_with_service(log_file, service_name, additional_message="
    "):
```

```
146     current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f
            ")[:-3]
147     new_entry = f"{service_name} - Service unavailable at {current_time}"
148 
149     # Cria o arquivo de log se n o existir
150     if not os.path.exists(log_file):
151         with open(log_file, "a"):
152             pass
153 
154     # Verifica se a nova entrada j  est  no arquivo de log
155     with open(log_file, "r") as file:
156         if new_entry in file.read():
157             print(f"{service_name} - Service unavailable already logged."
                    )
158             return
159 
160     # Se n o estiver, adiciona a nova entrada
161     with open(log_file, "a") as file:
162         file.write(f"{RED}{new_entry}{ENDC}\n")
163         print(f"{service_name} - Service unavailable . Logged to {
                log_file} at {current_time}")
164 
165 def log_recovery(log_file, service_name):
166     current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:%f
            ")[:-3]
167     with open(log_file, "a") as file:
168         file.write(f"{GREEN}{service_name} - Service recovered at {
                current_time}{ENDC}\n")
169         print(f"{service_name} - Service recovered. Logged to {log_file}
                at {current_time}")
170 
171 def establish_ssh_connection(server_ip, username, password):
172     ssh_client = paramiko.SSHClient()
173     ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
174     ssh_client.connect(server_ip, username=username, password=password)
175     return ssh_client
176 
177 def monitor_mysql_service(ssh_client=None):
178     try:
```

```
179         mysql_service_available = check_mysql_service(mysql_log_file,
                ssh_client)
180         if not mysql_service_available:
181             log_failure(mysql_log_file, "MySQL")
182         else:
183             log_recovery(mysql_log_file, "MySQL")
184     except Exception as e:
185         print(f"Erro ao monitorar o servi o MySQL: {e}")
186 
187 def monitor_apache_service(ssh_client=None):
188     try:
189         apache_service_available = check_apache_service(apache_log_file,
                ssh_client)
190         if not apache_service_available:
191             log_failure(apache_log_file, "Apache")
192         else:
193             log_recovery(apache_log_file, "Apache")
194     except Exception as e:
195         print(f"Erro ao monitorar o servi o Apache: {e}")
196 
197 def monitor_php_service(ssh_client=None):
198     try:
199         php_service_available = check_php_service(php_log_file,
                ssh_client)
200         if not php_service_available:
201             log_failure(php_log_file, "PHP")
202         else:
203             log_recovery(php_log_file, "PHP")
204     except Exception as e:
205         print(f"Erro ao monitorar o servi o PHP: {e}")
206 
207 def monitor_hardware():
208     while True:
209         try:
210             check_hardware(hardware_log_file)
211         except Exception as e:
212             print(f"Erro ao monitorar hardware: {e}")
213 
214         # Pausa por 30 segundos antes de realizar a pr xima
```

```
            verifica  o
215         time.sleep(30)
216
217 def monitor_os():
218     while True:
219         try:
220             check_os(os_log_file)
221         except Exception as e:
222             print(f"Erro ao monitorar sistema operacional: {e}")
223
224         # Pausa por 30 segundos antes de realizar a pr xima
            verifica  o
225 #         time.sleep(30)
226 def system_status():
227     apache_reachable = check_ping(apache_server_ip, "Apache")
228     php_reachable = check_ping(php_server_ip, "PHP")
229     mysql_reachable = check_ping(mysql_server_ip, "MySQL")
230     hardware_reachable = check_ping(hardware_server_ip, "Hardware")
231     os_reachable = check_ping(os_server_ip, "OS")
232
233     if apache_reachable and php_reachable and mysql_reachable and
        hardware_reachable and os_reachable:
234         apache_ssh_client = establish_ssh_connection(apache_server_ip,
            ssh_username, ssh_password)
235         php_ssh_client = establish_ssh_connection(php_server_ip,
            ssh_username, ssh_password)
236         mysql_ssh_client = establish_ssh_connection(mysql_server_ip,
            ssh_username, ssh_password)
237
238         apache_service_available = check_apache_service(apache_log_file,
            apache_ssh_client)
239         php_service_available = check_php_service(php_log_file,
            php_ssh_client)
240         mysql_service_available = check_mysql_service(mysql_log_file,
            mysql_ssh_client)
241
242         apache_ssh_client.close()
243         php_ssh_client.close()
244         mysql_ssh_client.close()
```

```
245
246         if apache_service_available and php_service_available and
                mysql_service_available:
247             return "System - UP"
248         else:
249             return "System - Down"
250     else:
251         return "System - Down"
252
253 def main():
254     system_log_file = "system_log.txt"
255
256     while True:
257         try:
258             status = system_status()
259             current_time = datetime.datetime.now().strftime("%Y-%m-%d %H
                    :%M:%S:%f")[:-3]
260             with open(system_log_file, "a") as file:
261                 file.write(f"{current_time} - {status}\n")
262                 print(f"System Status: {status}. Logged to {
                        system_log_file} at {current_time}")
263         except Exception as e:
264             print(f"Erro: {e}")
265
266         # Pausa por 05 segundos antes de realizar a pr xima
                verifica  o
267         time.sleep(5)
268
269 if __name__ == "__main__":
270     main()
```

Código Fonte 10 – Código Python para monitorar e salvar status do sistema - roda a cada 5seg

APÊNDICE G – SCRIPT PARA ANALISAR DADOS DE LOG E MOSTRAR MÉTRICAS DE INTERESSE

```
'''
O script Python processa um arquivo de log do sistema para calcular
    m tricas de disponibilidade, como Tempo M dio at  a Falha (MTTF),
Tempo M dio para Reparo (MTTR), e outras estat sticas relacionadas.
Ele l  o arquivo de log, identifica per odos de funcionamento (UP) e
    per odos de falha (Down),
calcula as m tricas mencionadas e gera um arquivo de sa da com essas
    informa  es. Al m disso,
o script realiza an lises estat sticas adicionais, como o c lculo de
    intervalos de confian a e outras m tricas de disponibilidade.
As m tricas calculadas s o detalhadas e armazenadas em um arquivo de
    sa da para refer ncia posterior.
'''

import sys
import math
from datetime import datetime

fator_aceleracao = int(1000)

def processar_logs(caminho_entrada):
    with open(caminho_entrada, 'r') as arquivo:
        linhas = arquivo.readlines()

    grupos_up = []
    grupos_down = []
    grupo_atual = {'status': None, 'inicio': None, 'fim': None}

    total_mttf = 0  # Adicionado para calcular o total de MTTF
    total_mttr = 0  # Adicionado para calcular o total de MTTR

    for i in range(len(linhas) - 1):
        linha_atual = linhas[i]
        linha_seguinte = linhas[i + 1]

```

```
31          partes_atual = linha_atual.split(' - ')
32          partes_seguinte = linha_seguinte.split(' - ')
33
34          if len(partes_atual) < 3 or len(partes_seguinte) < 3:
35              # Linhas mal formatadas, ignorar
36              continue
37
38          timestamp_atual_str, status_atual = partes_atual[0], partes_atual
                [-1].strip()
39          timestamp_seguinte_str, status_seguinte = partes_seguinte[0],
                partes_seguinte[-1].strip()
40
41          timestamp_atual = datetime.strptime(timestamp_atual_str, '%Y-%m-%
                d %H:%M:%S:%f')
42          timestamp_seguinte = datetime.strptime(timestamp_seguinte_str, '%
                Y-%m-%d %H:%M:%S:%f')
43
44          if grupo_atual['status'] is None:
45              grupo_atual['status'] = status_atual
46              grupo_atual['inicio'] = timestamp_atual
47
48          if status_atual != status_seguinte:
49              grupo_atual['fim'] = timestamp_seguinte
50
51              if grupo_atual['status'] == 'UP':
52                  grupos_up.append({'inicio': grupo_atual['inicio'], 'fim':
                        grupo_atual['fim']})
53                  total_mttf += (grupo_atual['fim'] - grupo_atual['inicio'
                        ]).total_seconds()  # Adicionado para calcular MTTF
54              elif grupo_atual['status'] == 'Down':
55                  grupos_down.append({'inicio': grupo_atual['inicio'], 'fim
                        ': grupo_atual['fim']})
56                  total_mttr += (grupo_atual['fim'] - grupo_atual['inicio'
                        ]).total_seconds()  # Adicionado para calcular MTTR
57
58              grupo_atual = {'status': status_seguinte, 'inicio':
                    timestamp_seguinte, 'fim': None}
59
60      # Tratar o  ltimo  grupo
```

```
61      grupo_atual['fim'] = timestamp_seguinte
62      if grupo_atual['status'] == 'UP':
63          grupos_up.append({'inicio': grupo_atual['inicio'], 'fim':
                grupo_atual['fim']})
64          total_mttf += (grupo_atual['fim'] - grupo_atual['inicio']).
                total_seconds()  # Adicionado para calcular MTTF
65      elif grupo_atual['status'] == 'Down':
66          grupos_down.append({'inicio': grupo_atual['inicio'], 'fim':
                grupo_atual['fim']})
67          total_mttr += (grupo_atual['fim'] - grupo_atual['inicio']).
                total_seconds()  # Adicionado para calcular MTTR
68
69      return grupos_up, grupos_down, total_mttf, total_mttr
70
71  def imprimir_resultados(grupos_up, grupos_down, total_mttf, total_mttr):
72      with open(caminho_saida, "w") as arquivo_saida:
73          for i, grupo in enumerate(grupos_up, start=1):
74              if i <= len(grupos_down):
75                  linha_up = f"Sistema UP - In cio: {grupo['inicio']}, Fim
                        : {grupo['fim']}, TTF: { (grupo['fim'] - grupo['inicio
                        ']).total_seconds():.3f}s"
76                  linha_down = f"Sistema Down - In cio: {grupos_down[i
                        -1]['inicio']}, Fim: {grupos_down[i-1]['fim']}, TTR: {
                         (grupos_down[i-1]['fim'] - grupos_down[i-1]['inicio
                        ']).total_seconds():.3f}s"
77                  print(linha_up)
78                  print(linha_down)
79                  arquivo_saida.write(linha_up + "\n")
80                  arquivo_saida.write(linha_down + "\n")
81              else:
82                  linha_up = f"Sistema UP - In cio: {grupo['inicio']}, Fim
                        : {grupo['fim']}, TTF: { (grupo['fim'] - grupo['inicio
                        ']).total_seconds():.3f}s"
83                  print(linha_up)
84                  arquivo_saida.write(linha_up + "\n")
85
86          # Adicionar a quantidade de UP e Down
87          qtde_up = f"\nTotal repairs (System UP): {len(grupos_up)}"
88          qtde_down = f"Total failures (System Down): {len(grupos_down)}"
```

```
89          print(qtde_up)
90          print(qtde_down)
91          arquivo_saida.write(qtde_up + "\n")
92          arquivo_saida.write(qtde_down + "\n")
93
94          # Adicionar o total de MTTF e MTTR
95          total_mttf_str = f"\nTTF(s) [*fator aceleracao]: {total_mttf:.3f}
                "
96          total_mttr_str = f"TTR(s): {total_mttr:.3f}"
97          mttf=(total_mttf/3600)*fator_aceleracao
98          mttr=total_mttr/3600
99
100         ttf_str=f"TTF(h): {mttf:.6f}"
101         ttr_str=f"TTR(h): {mttr:.6f}"
102
103         MTTF=f"\nMTTF(h): {mttf / len(grupos_up):.10f}"
104         MTTR=f"MTTR(h): {mttr / len(grupos_down):.10f}"
105         A=f"\nA: {mttf / (mttf + mttr):.10f}"
106         U=f"U: {1-(mttf / (mttf + mttr)):.10f}"
107         Num9=f"Number of 9's: {-math.log(1-(mttf / (mttf + mttr))) / math
                .log(10):.10f}"
108         Uptime=f"Uptime: {8760-(1-(mttf / (mttf + mttr)))*8760:.10f}"
109         Downtime=f"Downtime: {(1-(mttf / (mttf + mttr)))*8760:.10f}"
110         ##############################################
111         alfa=0.05 # n vel de confian a 95%
112         ##############################################
113         nivelConfianca=f"N vel de confian a: {alfa}"
114         RO=f"RO: {(mttr/mttf):.10f}"
115         GrauLiberdadeNumeradorF=f"Distribui ao F = GL Numerador: {len(
                grupos_up)}"
116         GrauLiberdadeDenominadorF=f"Distribui ao F = GL Denominador: {
                len(grupos_down)}"
117
118
119         valorCritico1=0.7992 #GL ANALISADO
120         valorCritico2=1.251 #GL ANALISADO
121         v1=f"Valor Cr tico Inferior: {valorCritico1}"
122         v2=f"Valor Cr tico Superior: {valorCritico2}"
123         ##############################################
```

```
124         Rl=f"R_l: {(mttr/mttf)/valorCritico2:.10f}"
125         Ru=f"R_u: {(mttr/mttf)/valorCritico1:.10f}"
126         intervaloConfiancaInferior=f"Intervalo inferior: {1/(1+(mttr/mttf
                )/valorCritico1):.10f}"
127         intervaloConfiancaSuperior=f"Intervalo Superior: {1/(1+(mttr/mttf
                )/valorCritico2):.10f}"
128
129         print(total_mttf_str)
130         print(total_mttr_str)
131         print(ttf_str)
132         print(ttr_str)
133         print(MTTF)
134         print(MTTR)
135         print("\nPar metros distribui  o")
136         print(GrauLiberdadeNumeradorF)
137         print(GrauLiberdadeDenominadorF)
138         print(nivelConfianca)
139         print(v1)
140         print(v2)
141         print(RO)
142         print(Rl)
143         print(Ru)
144         print("\nIntervalo de confian a 95%")
145         print(intervaloConfiancaInferior)
146         print(intervaloConfiancaSuperior)
147         print(A)
148         print(U)
149         print(Num9)
150         print(Uptime)
151         print(Downtime)
152         print(f"\nDados salvos em {caminho_saida}\n")
153
154         arquivo_saida.write(total_mttf_str + "\n")
155         arquivo_saida.write(total_mttr_str + "\n")
156         arquivo_saida.write(ttf_str + "\n")
157         arquivo_saida.write(ttr_str + "\n")
158         arquivo_saida.write(MTTF + "\n")
159         arquivo_saida.write(MTTR + "\n")
160         arquivo_saida.write("\n"+"Par metros distribui  o"+ "\n")
```

```
161         arquivo_saida.write(GrauLiberdadeNumeradorF + "\n")
162         arquivo_saida.write(GrauLiberdadeDenominadorF + "\n")
163         arquivo_saida.write(nivelConfianca + "\n")
164         arquivo_saida.write(v1 + "\n")
165         arquivo_saida.write(v2 + "\n")
166         arquivo_saida.write(RO + "\n")
167         arquivo_saida.write(Rl + "\n")
168         arquivo_saida.write(Ru + "\n")
169         arquivo_saida.write("\n"+"Intervalo de confian a 95%"+ "\n")
170         arquivo_saida.write(intervaloConfiancaInferior + "\n")
171         arquivo_saida.write(intervaloConfiancaSuperior + "\n")
172         arquivo_saida.write(A + "\n")
173         arquivo_saida.write(U + "\n")
174         arquivo_saida.write(Num9 + "\n")
175         arquivo_saida.write(Uptime + "\n")
176         arquivo_saida.write(Downtime + "\n")
177
178 if __name__ == "__main__":
179     caminho_entrada = r"D:\\Users\\Gervasio Teixeira\\Downloads\\
            system_log.txt"
180
181     caminho_saida = r"D:\\Users\\Gervasio Teixeira\\Downloads\\
            System_MTTF_MTTR.txt"
182     grupos_up, grupos_down, total_mttf, total_mttr = processar_logs(
            caminho_entrada)
183
184     imprimir_resultados(grupos_up, grupos_down, total_mttf, total_mttr)
```

Código Fonte 11 – Código Python para analisar arquivo de LOG e mostrar métricas de interesse