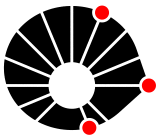

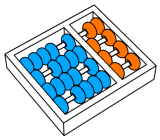

Estudo de caso Emergent Web Server e Digital Twins

J. V. Cardoso *L. F. Bittencourt* *R. R. Filho* *C. Melo*

Relatório Técnico - IC-PFG-25-08
Projeto Final de Graduação
2025 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Estudo de caso: Emergent Web Server e Digital Twins

João Victor Cardoso* Luiz Fernando Bittencourt† Roberto Rodrigues Filho‡
Carlos Melo§

Resumo

Este trabalho aborda o desafio da oscilação de desempenho no Emergent Web Server (EWS), um servidor web auto-adaptativo que utiliza aprendizado por reforço para otimizar dinamicamente suas configurações. Para mitigar este problema, propomos a integração do EWS com Gêmeos Digitais, criando uma camada virtual que simula o comportamento do sistema real através de um modelo preditivo baseado em XGBoost e uma arquitetura não intrusiva de coleta de dados. O gêmeo digital desenvolvido reduz significativamente as flutuações de desempenho durante o aprendizado, como demonstrado em testes locais e em nuvem (AWS).

1 Introdução

Com a crescente complexidade dos sistemas de software e a sua implantação em ambientes cada vez mais dinâmicos e imprevisíveis, cresceram também as pesquisas nas áreas de sistemas autônomos, auto-adaptativos e auto-organizáveis [1]. Essas abordagens visam transferir parte da responsabilidade de gerenciamento para o próprio sistema, reduzindo a dependência de intervenção humana direta e aumentando sua resiliência frente a mudanças [1].

Os sistemas auto-adaptativos surgem como uma resposta eficaz a esse cenário [2]. Eles são projetados para monitorar, analisar e modificar o seu próprio comportamento em tempo de execução com o objetivo de manter níveis desejáveis de desempenho, segurança e usabilidade, mesmo diante de variações inesperadas no ambiente como: flutuações abruptas de carga (ex.: picos de requisições em servidores web) ou degradação inesperada de recursos (ex.: perda de nós em redes de sensores) [2, 4].

Nesse contexto, o Emergent Web Server (EWS) introduz um novo paradigma dentro dos sistemas adaptativos, denominado Emergent Software Systems) [5]. Esses sistemas se caracterizam por uma adaptação contínua e uma composição dinâmica, na qual o próprio software aprende, por meio de técnicas de Reinforcement Learning, quais combinações de componentes proporcionam o melhor desempenho frente às condições reais de execução [5]. Essa abordagem permite ao EWS ajustar-se de forma autônoma e eficiente a diferentes tipos de requisições, otimizando o uso de recursos e reduzindo o tempo de resposta [1, 5].

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.
†Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.
‡Instituto de Informática, Universidade Federal de Goiás, 74690-900 Goiânia, GO.
§Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

Complementarmente, o conceito de Gêmeos Digitais (DT, na sigla em inglês para Digital Twins), discutido em [8], tem ganhado destaque por sua capacidade de fornecer representações digitais de sistemas físicos e virtuais, promovendo uma análise mais precisa, preditiva e adaptativa do comportamento de sistemas complexos.

Diante desse cenário, este projeto propõe a integração dos princípios do Emergent Web Server (EWS) com o conceito de Gêmeos Digitais, visando o desenvolvimento de um servidor auto-adaptativo orientado por gêmeos digitais. A proposta consiste em construir uma arquitetura capaz de coletar dados em tempo real sobre as requisições, utilizando o gêmeo digital como uma camada de abstração para monitoramento, previsão de desempenho e apoio às decisões de reconfiguração. Dessa forma, busca-se potencializar a adaptabilidade e eficiência do sistema, promovendo ajustes automáticos e contínuos conforme as condições operacionais e demandas variáveis.

2 Referencial Teórico

Esta seção apresenta os fundamentos teóricos essenciais para o desenvolvimento deste trabalho, organizados em três eixos principais: (i) **Computação Autonômica e Sistemas Auto-Adaptativos**, que estabelecem os princípios de autogestão; (ii) **Sistemas de Software Emergentes** com foco no framework PAL (*Perception-Assembly-Learning*); e (iii) **Gêmeos Digitais** baseados na arquitetura MODA (*Models and Data*). Estes conceitos fornecem a base para a proposta de integração entre o Emergent Web Server (EWS) e seu gêmeo digital.

2.1 Computação Autonômica e Sistemas Auto-Adaptativos

A **computação autonômica**, proposta pela IBM em 2001 [7], surge como resposta à crescente complexidade dos sistemas de TI, oferecendo autogestão como solução para desafios de configuração, otimização e manutenção. Seu núcleo se baseia em quatro pilares essenciais: auto-configuração, que adapta automaticamente o sistema conforme políticas predefinidas; auto-otimização, voltada à melhoria contínua de desempenho e eficiência; auto-cura, capaz de detectar e corrigir falhas sem intervenção humana; e auto-proteção, que previne e mitiga ameaças externas ou falhas em cascata.

Os **sistemas auto-adaptativos** operacionalizam esses princípios por meio do ciclo MAPE-K [3], um modelo iterativo composto por:

- **Monitorar**: coleta de dados do sistema;
- **Analisar**: interpretação contextual dos dados;
- **Planejar**: geração de estratégias de adaptação;
- **Executar**: aplicação das ações definidas;
- **Conhecimento**: armazenamento dos dados coletados nas fases anteriores para fornecer suporte às decisões.

Como ilustrado na Figura 1, esse ciclo integra os elementos autonômicos, permitindo que ajustes dinâmicos sejam realizados em resposta a mudanças ambientais ou demandas operacionais.

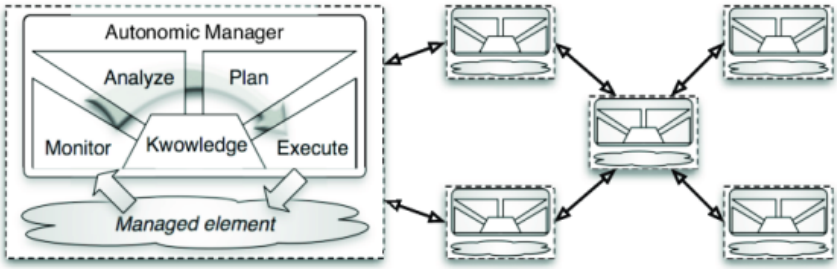

Figura 1: Ciclo MAPE-K em elementos autonômicos. Retirado de [7].

2.2 Sistemas de Software Emergentes e Arquitetura PAL

Os **Sistemas de Software Emergentes** representam uma evolução dos paradigmas auto-adaptativos, superando três limitações fundamentais: (i) dependência de regras pré-programadas, (ii) controle centralizado rígido, e (iii) incapacidade de lidar com cenários não previstos, como demonstra [5], esses sistemas utilizam componentes reutilizáveis que realizam auto-composição e auto-otimização dinâmica baseadas em condições ambientais. A Figura 2

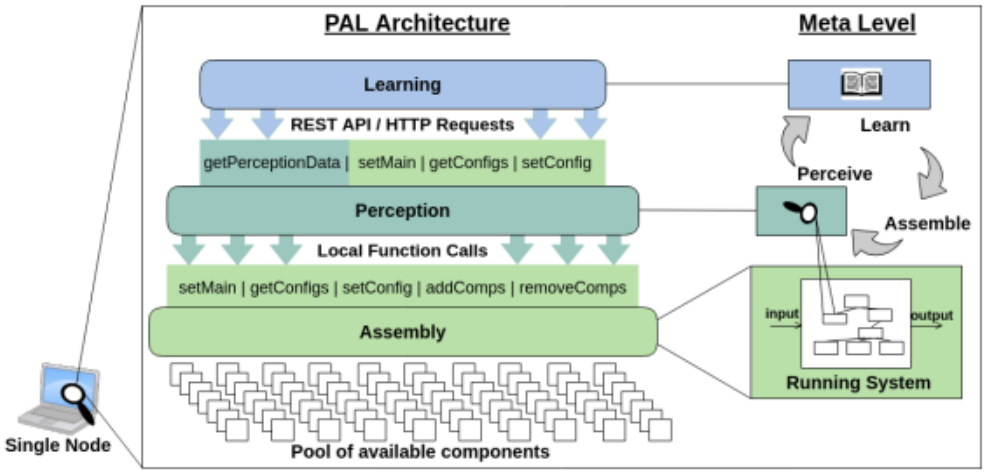

Figura 2: Arquitetura do framework PAL. Retirado de [5].

ilustra a arquitetura do **framework PAL (Perception-Assembly-Learning)**, que concretiza esses princípios emergentes por meio de três módulos principais:

- **Perception**: monitora continuamente o ambiente e o estado do sistema, coletando

métricas como latência, *throughput* e padrões de requisições via *proxies* embutidos.

- **Assembly**: gerencia a reconfiguração dinâmica do sistema em tempo de execução utilizando a linguagem Dana[**?**]. Suas principais operações incluem:
  - `getConfigs()`: lista composições de componentes possíveis;
  - `setConfig(configDesc)`: aplica novas configurações;
  - `addComp(compName)`: adiciona novo componente.
- **Learning**: implementa aprendizado por reforço para explorar o espaço de configurações (42 combinações no EWS [5]), avalia o desempenho e seleciona composições ótimas conforme objetivos pré-definidos.

2.3 Gêmeos Digitais (Digital Twins)

Os **Gêmeos Digitais (DTs)** surgem como réplicas digitais dinâmicas de sistemas físicos, mantendo sincronização bidirecional e contínua com suas contrapartes reais (*Actual System - AS*) por meio de fluxos de dados em tempo real [6]. Conforme formalizado pelo framework **MODA (Models and Data)** (Figura 3), esses sistemas integram três componentes essenciais:

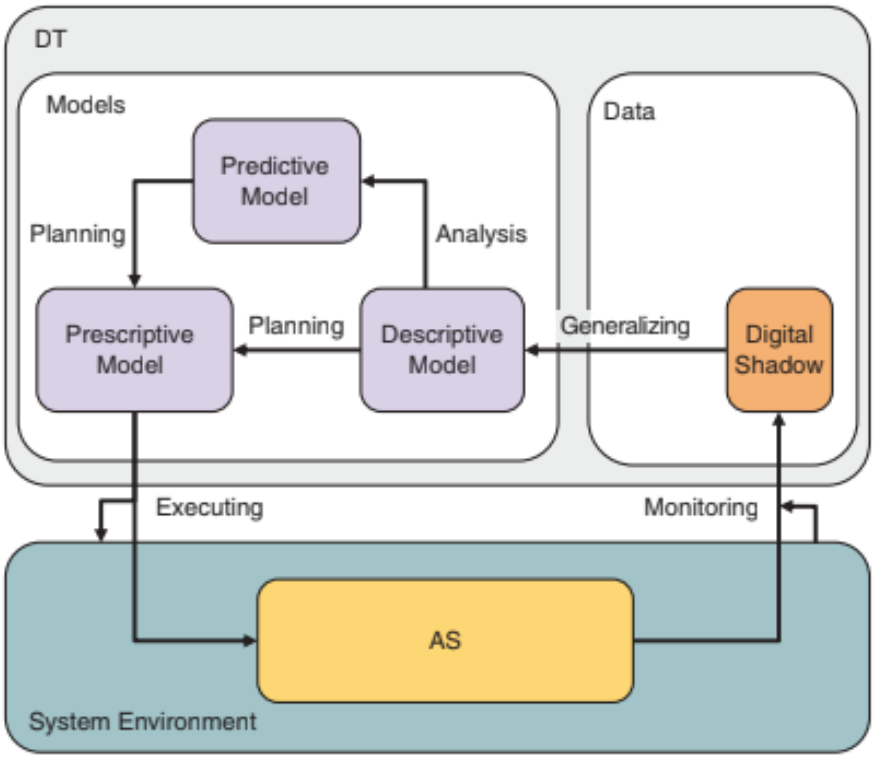

Figura 3: Arquitetura MODA para Gêmeos Digitais. Adaptado de [6].

(1) Digital Shadow: Funciona como núcleo dinâmico de dados, agregando de forma unidirecional o estado operacional do AS. Este componente consolida dados sensoriais brutos (ex: telemetria de IoT, métricas de desempenho) e condições internas, estruturando-os em representações abstraídas para detecção autônoma de anomalias [6].

(2) External Data: Incorpora fontes exógenas ao AS, como dados ambientais (ex: condições climáticas para veículos autônomos), flutuações de mercado e regulatórias. Quando fundidos ao *Digital Shadow*, esses contextos externos habilitam respostas adaptativas a perturbações imprevisíveis [6].

(3) Model Twin: Converte dados brutos em conhecimento acionável através de três camadas analíticas interconectadas:

- *Modelos Descritivos*: Reconstroem o estado corrente do AS baseado em dados históricos e em tempo real;
- *Modelos Preditivos*: Projetam comportamentos futuros via simulações e aprendizado de máquina (ex: detecção de falhas em manufatura);
- *Modelos Prescritivos*: Geram ações otimizadas para intervenção no AS (ex: reconfigurações em sistemas agroindustriais).

Essa tríade fecha o ciclo de inteligência, permitindo desde diagnósticos precisos até intervenções automatizadas alinhadas aos objetivos do sistema físico [6].

3 Objetivos

Este trabalho tem como objetivo central desenvolver um **gêmeo digital** para o Emergent Web Server (EWS) que simule o comportamento adaptativo e otimize dinamicamente o desempenho deste. Busca-se integrar os princípios de sistemas emergentes, implementados pelo **framework PAL**, com a estrutura conceitual de gêmeos digitais do **framework MODA**, visando reduzir significativamente as oscilações de desempenho no sistema real durante processos de adaptação.

Para concretizar essa integração, foram implementados dois componentes essenciais: o **Digital Learning**, que encapsula o serviço de Reinforcement Learning do EWS como um módulo autônomo, e o **EWS Model**, um modelo preditivo capaz de antecipar o desempenho do sistema real em diversas configurações.

O estudo busca responder às seguintes questões fundamentais: (i) como modelar computacionalmente o comportamento adaptativo do EWS em suas múltiplas configurações possíveis, e (ii) em que medida o gêmeo digital pode reduzir a variação de desempenho observada no sistema físico durante o aprendizado.

4 Metodologia

4.1 Diagramas Propostos

Nesta seção, apresentamos três diagramas que detalham a integração entre o Emergent Web Server (EWS) e os Gêmeos Digitais (Digital Twins), fundamentada no framework MODA

(Models and Data) conforme proposto por [6]. O primeiro diagrama proporciona uma visão conceitual geral, ilustrando a interação entre monitoramento, modelagem e adaptação no contexto do EWS. Os dois diagramas subsequentes descrevem implementações práticas para validar as questões deste projeto: uma arquitetura em nuvem utilizando AWS (Amazon Web Services) e uma arquitetura local para execução de testes controlados.

4.1.1 Diagrama Conceitual: Integração MODA-EWS

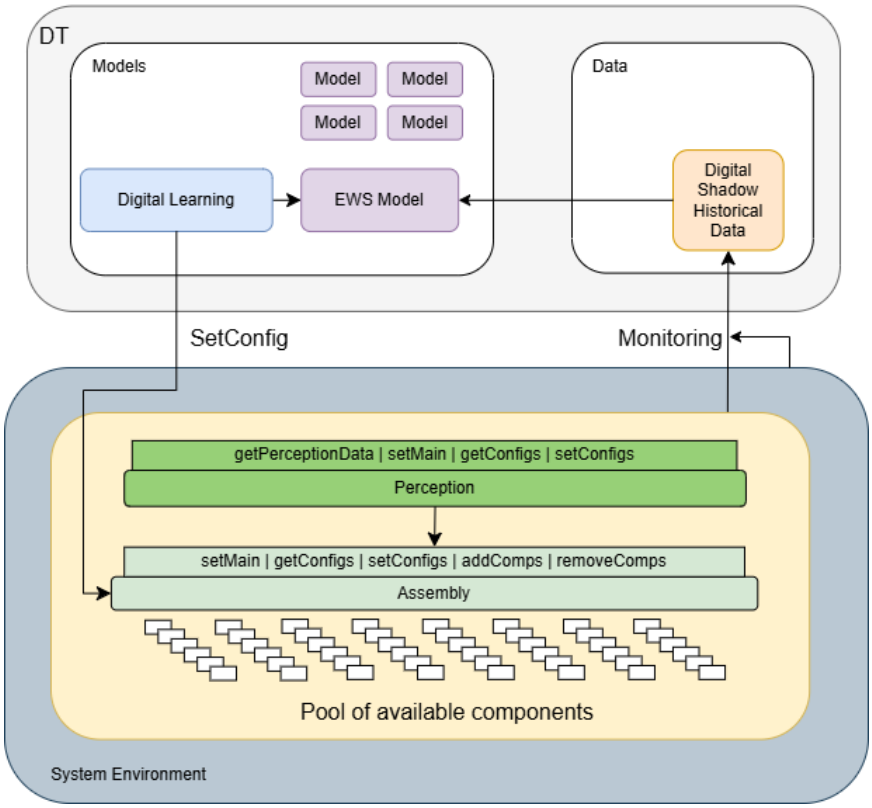

Figura 4: Modelo proposto de integração entre EWS e Gêmeo Digital

.

O diagrama conceitual (Figura 4) propõe a integração do *Emergent Web Server* (EWS) com uma arquitetura de *Digital Twin* (DT), estruturada segundo o framework MODA [6], que diferencia, de forma conceitual, as camadas de **Modelos** e **Dados**.

Na arquitetura proposta, as métricas recolhidas na camada **Perception** do EWS, responsável por capturar métricas internas do servidor, são continuamente enviadas para a camada **Data**, constituindo a *Digital Shadow* do sistema. A *Digital Shadow* pode ser complementada com dados coletados externamente ao EWS, como métricas de infraestrutura, consumo de recursos do ambiente e monitoramento de rede.

A partir da *Digital Shadow*, continuamente populada conforme o uso do sistema, diversos

modelos podem ser desenvolvidos na camada **Modeling**, incluindo, mas não se limitando a:

- **Modelos preditivos de tempo de resposta**: capazes de estimar o comportamento do EWS conforme o tipo de requisição (texto ou imagem) e as 42 combinações de componentes disponíveis.
- **Modelos de detecção antecipada de anomalias**: criados por meio da análise histórica de dados, podem identificar padrões que antecipem falhas ou degradação do ambiente.
- **Modelos de apoio**: sugerem proativamente a melhor configuração do EWS conforme o contexto da operação atual, o que reduz o custo de exploração das configurações no sistema real.
- **Modelos para simulações**: possibilitam prever o comportamento do EWS em diferentes condições de carga e configuração, sem impactar o sistema real.

Na Figura 4, destaca-se o **EWS Model**, um modelo orientado a *Machine Learning* que permite simular o comportamento do servidor frente a diferentes padrões de requisições e configurações. Esse modelo atua como um apoio ao processo adaptativo, podendo antecipar resultados e evitar explorações no ambiente real.

Também é apresentado no diagrama o serviço de **Digital Learning**, algoritmo de aprendizado nativo do EWS. Ele é responsável por realizar automaticamente a adaptação das configurações, compondo os diversos componentes para otimizar o desempenho do EWS conforme a carga de trabalho.

Esse diagrama, portanto, está alinhado com a proposta de [6], que enfatiza que os gêmeos digitais devem atuar não só como uma representação descritiva do sistema, mas também como entidades com capacidades preditivas e prescritivas, utilizando um fluxo de *feedback* que potencializa a auto-adaptação.

4.1.2 Implementações Práticas: Nuvem e Ambiente Local

Para a validação experimental, desenvolvemos duas arquiteturas operacionais equivalentes que implementam a integração entre o EWS e o gêmeo digital proposta nas Figuras 5 e 6. A arquitetura representada na Figura 5 utiliza recursos de nuvem AWS, enquanto a da Figura 6 opera em ambiente local dockerizado para testes controlados, ambas mantêm a mesma funcionalidade central.

Na **implementação em nuvem** (Figura 5), consolidamos todos os componentes em uma única instância EC2 da AWS. Esta hospeda o Emergent Web Server (EWS) por intermédio da imagem Docker `robertovrf/ews:1.0`, expondo as portas 2011-2012 para comunicação. Além disso, um servidor Nginx atua como proxy reverso na porta padrão 80, direcionando requisições de clientes para o EWS enquanto registra todos os acessos em arquivos `access_log`. Complementando essa estrutura, um agente CloudWatch monitora continuamente esses *logs*, coletando e transmitindo os dados para armazenamento centralizado na plataforma Amazon CloudWatch.

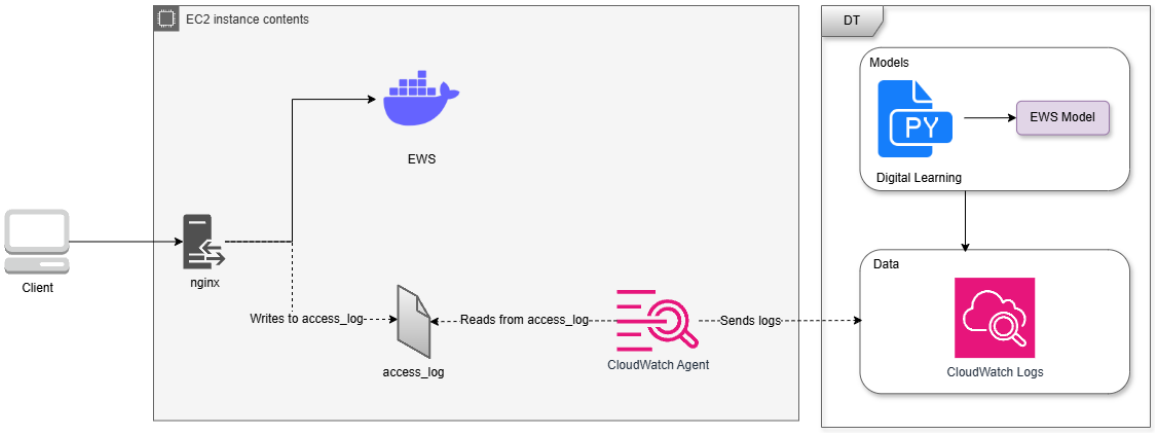

Figura 5: Arquitetura AWS

A **implementação local** (Figura 6) replica essa funcionalidade por meio de uma arquitetura *dockerizada*, mantendo plena equivalência operacional. O *container* principal executa a mesma imagem do EWS (`robertovrf/ews:1.0`) com as mesmas portas expostas, enquanto um *container* separado para o Nginx realiza o mapeamento da porta 8080 local para a porta 80 interna, utilizando um volume persistente para armazenar os logs de acesso. O *container* do CloudWatch Agent, configurado com acesso somente a leitura a esses logs, completa o fluxo ao transmitir os dados para o mesmo repositório na nuvem.

Em ambas as implementações, o fluxo operacional segue um padrão consistente: o Nginx recebe e encaminha requisições de clientes para o EWS; cada operação gera registros detalhados no `access_log`; o CloudWatch Agent captura esses registros em tempo real; e os dados são transmitidos para o CloudWatch, atualizando continuamente a Digital Shadow, que serve como base para os modelos preditivos do gêmeo digital. Essa simetria entre ambientes permite validar resultados em diferentes condições operacionais enquanto mantém a consistência dos dados analíticos.

4.2 EWS Model: Modelo Preditivo de Desempenho

O **EWS Model** constitui o núcleo preditivo do Digital Twin, responsável por modelar computacionalmente o comportamento do Emergent Web Server (EWS) em suas 42 configurações possíveis. Esta seção detalha o seu desenvolvimento completo, desde a coleta de dados até a avaliação de desempenho final.

4.2.1 Coleta e Pré-processamento de Dados

Os dados de treinamento foram gerados a partir de *benchmarks* controlados em dois ambientes distintos: local (Docker) e nuvem (AWS EC2). Para cada uma das 42 configurações de componentes (obtidas via API `/ews-api/meta/get_all_configs`), realizamos testes de desempenho com quatro endpoints representativos: `/ews/image-10b.jpg` (imagem pequena), `/ews/index_big3.html` (HTML grande), `/ews/index.html` (HTML sim-

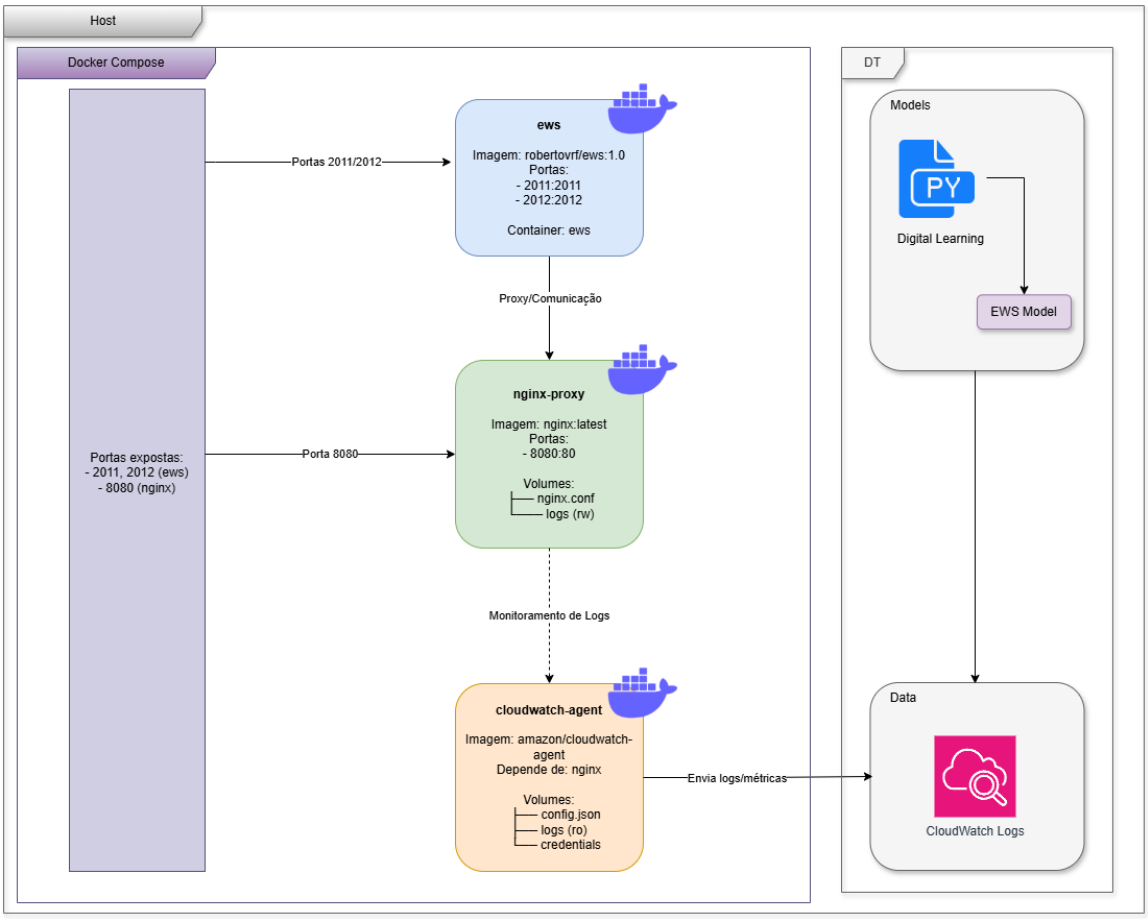

Figura 6: Arquitetura Local

ples) e `/ews/lancaster.jpg` (imagem grande). Cada configuração foi testada durante 5 minutos no ambiente local e 10 minutos na nuvem, coletando sete parâmetros essenciais por requisição: timestamp ISO 8601, configuração ativa, endpoint acessado, tamanho do payload, tamanho da resposta, tempo de resposta ao cliente e código de status HTTP.

No pré-processamento, aplicamos transformações críticas aos dados brutos: configurações e endpoints foram codificados numericamente via `LabelEncoder` (0-41 para configurações, 0-3 para endpoints), enquanto features numéricas como `payload_size` foram normalizadas em escala min-max [0-1] e `response_size` recebeu transformação logarítmica. O conjunto final foi particionado estratificadamente em 60% treino, 20% validação e 20% teste, garantindo representatividade proporcional de todas as configurações.

4.2.2 Modelagem, Treinamento e Desempenho

Implementamos o modelo preditivo utilizando o algoritmo XGBoost (eXtreme Gradient Boosting), otimizando seus hiperparâmetros por meio do framework Optuna com 100 iterações.

O espaço de busca incluiu `n_estimators` (100-1000), `max_depth` (3-12), `learning_rate` (0.001-0.3 em escala logarítmica), `subsample` (0.5-1.0) e `colsample_bytree` (0.5-1.0), com objetivo de minimizar o MAE (Mean Absolute Error) e parada antecipada após 10 épocas sem melhoria.

No ambiente local, o modelo final alcançou excelente desempenho: MAE de 1.16 ms, RMSE de 2.80 ms, MAPE de 30.32% e R^2 de 0.994. A análise de importância de features (Figura 7) revelou que a configuração ativa e o tamanho da resposta foram os preditores mais relevantes. Na nuvem, observamos padrão semelhante porém com métricas absolutas

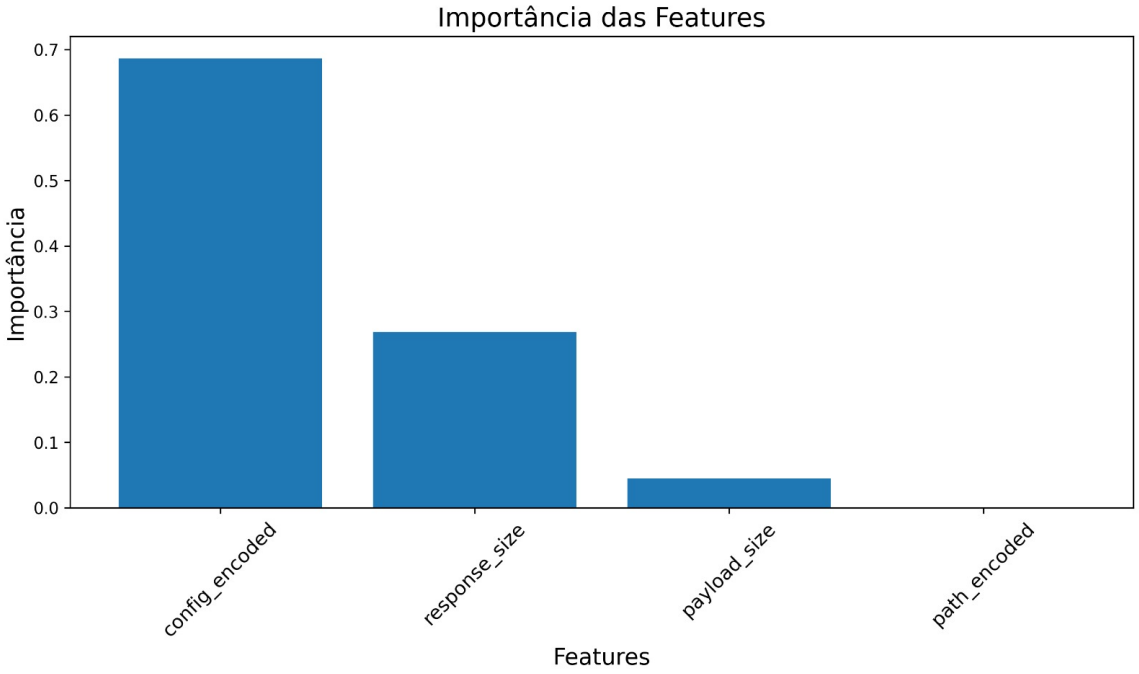

Figura 7: Importância relativa das features no modelo XGBoost Local

mais elevadas devido à variabilidade inerente do ambiente: MAE de 40.99 ms, RMSE de 118.81 ms, MAPE de 4.66% e R^2 de 0.979. A Figura 8 demonstra que, neste ambiente, a configuração ativa teve impacto significativamente maior na precisão preditiva. Estes resultados validam a capacidade do EWS Model em prever tempos de resposta com alta precisão em ambos os ambientes, estabelecendo uma base sólida para recomendações de configuração ótimas no sistema real.

4.3 Serviço de Digital Learning: Mecanismo de Adaptação Autônoma

O serviço **Digital Learning** implementa o núcleo de aprendizagem por reforço do Emergent Web Server, portado para Python e integrado ao Digital Twin. Esse componente autônomo gerencia todo o ciclo de adaptação do EWS, utilizando dados da Digital Shadow (CloudWatch) e previsões do EWS Model para otimizar continuamente o desempenho do sistema sem intervenção direta no servidor produtivo.

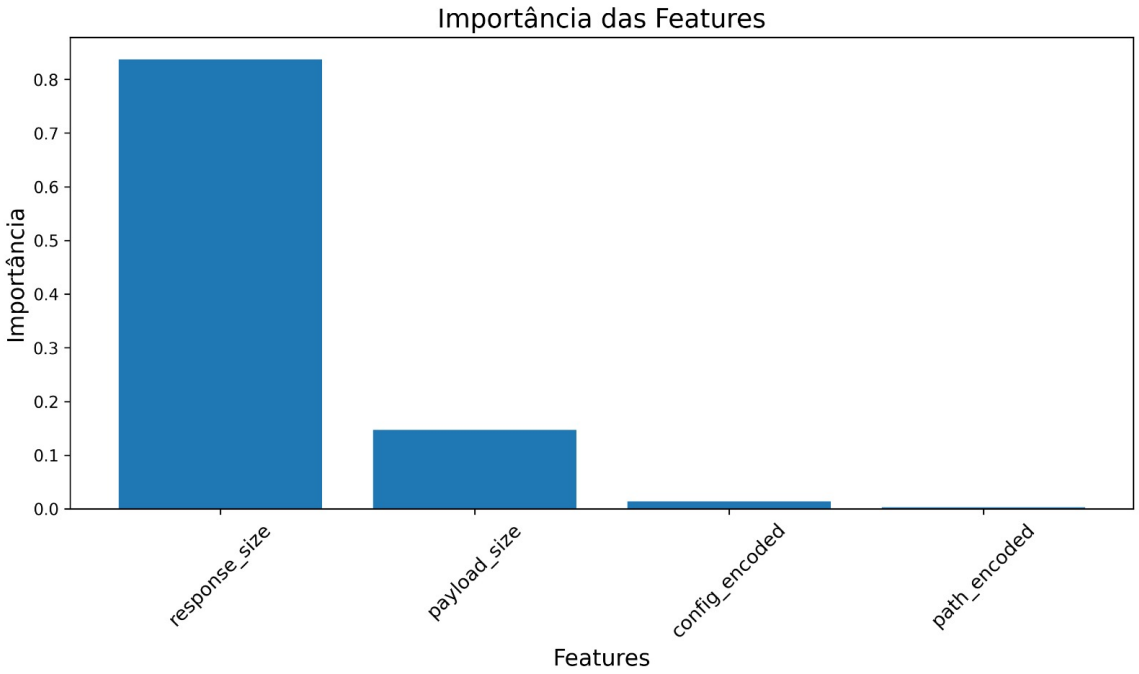

Figura 8: Importância relativa das features no modelo XGBoost Nuvem

O mecanismo opera a partir de um processo cíclico de aprendizado por reforço, nativo do EWS, ilustrado na Figura 9, que alterna entre fases de exploração e explotação. Na fase de exploração, o sistema testa iterativamente cada uma das 42 configurações disponíveis seguindo o seguinte ciclo: seleciona uma configuração C_i, aplica-a virtualmente, aguarda uma janela de tempo fixa w_t (gerando um tempo total de exploração $T_{\text{exploração}} = w_t \times N$), e recupera métricas de desempenho por meio da Digital Shadow. Ao final desse processo, a configuração com melhor desempenho (tipicamente menor tempo de resposta) é selecionada para a fase de explotação, sendo aplicada continuamente enquanto o sistema monitora seu desempenho em tempo real. Se detectada degradação ou mudança significativa no padrão de eventos, o ciclo retorna à fase de exploração. A principal inovação implementada no Digital Twin reside na abordagem não intrusiva de coleta de dados, que substitui o método original intrusivo. Enquanto a versão física exigia alteração direta da configuração do servidor via `setConfig(`C_i`)`, o novo método opera por meio de configuração virtual sem impacto no sistema real pela coleta passiva de logs do CloudWatch durante w_t, simulação preditiva do tempo de resposta via EWS Model e geração de métricas virtuais que alimentam transparentemente o núcleo de aprendizado. Essa transição elimina riscos operacionais, enquanto mantém a lógica original intacta, como detalhado no fluxo da Figura 10. A implementação em Python segue uma arquitetura modular composta por seis classes principais. O módulo `EWSMonitoring` (app/monitoring/ews_monitoring.py) coordena a percepção virtual, coletando logs do CloudWatch e utilizando o `EWSPredict` para gerar dados de percepção padronizados. Este último (app/monitoring/ews_predict.py) encapsula o modelo preditivo,

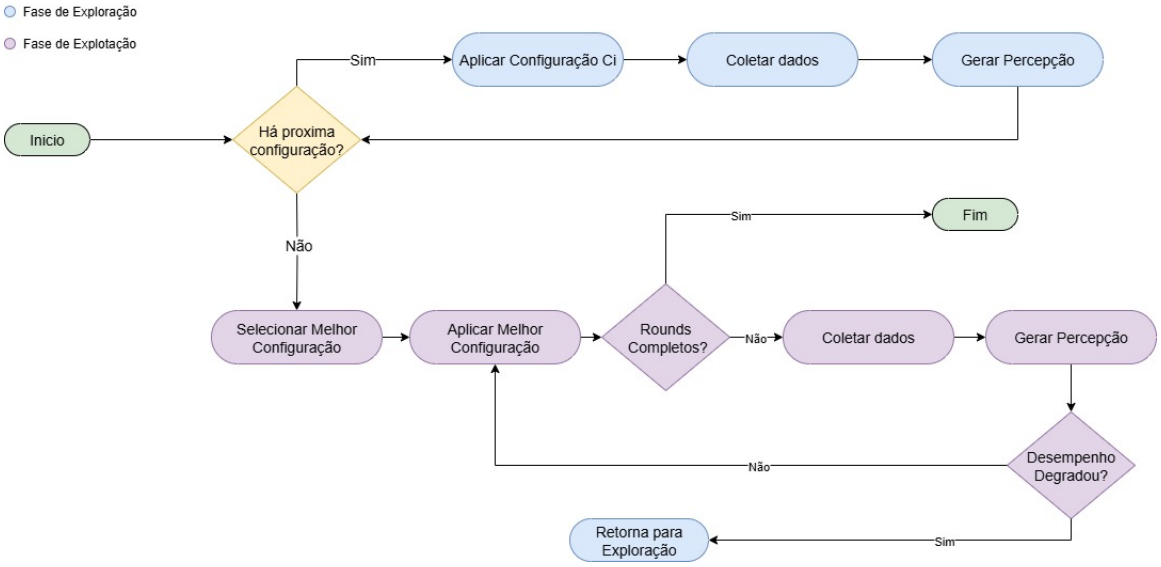

Figura 9: Fluxo do processo de aprendizado por reforço nativo do EWS

.

convertendo configurações e endpoints em representações numéricas para alimentar o EWS Model.

O núcleo lógico reside no `LearningCore` (app/learning/learning_core.py), que gerencia as fases de exploração (armazenando dados de ciclos) e explotação (selecionando configurações ótimas), mantendo uma base de conhecimento histórico. O módulo `Exploration` (app/learning/exploration.py) administra o espaço de configurações, reiniciando processos e verificando conclusões.

O controlador principal `EWSLearning` (app/learning/ews_learning.py) implementa uma máquina de estados com três modos operacionais: `START_EXPLORING` (inicialização), `EXPLORATION` (teste de configurações), e `EXPLOITATION` (monitoramento contínuo), orquestrando transições por meio do método `_execute_learning_cycle()`. Finalmente, o ponto de entrada `app.py` configura parâmetros, integra todos os componentes, executa ciclos de aprendizado e envia configurações ótimas ao servidor físico, completando o ciclo adaptativo.

5 Resultados Experimentais

Para avaliar a eficácia do Digital Twin (DT) na otimização do Emergent Web Server (EWS), conduzimos uma bateria de testes comparativos utilizando dois clientes especializados: um para recursos textuais (`text_client.py`) e outro para recursos de imagem (`image_client.py`). Ambos os clientes simularam carga contínua no servidor, registrando tempos de resposta com precisão de milissegundos. Os testes foram realizados em dois ambientes distintos, local e nuvem (AWS), comparando sistematicamente o desempenho do EWS com e sem a utilização do DT.

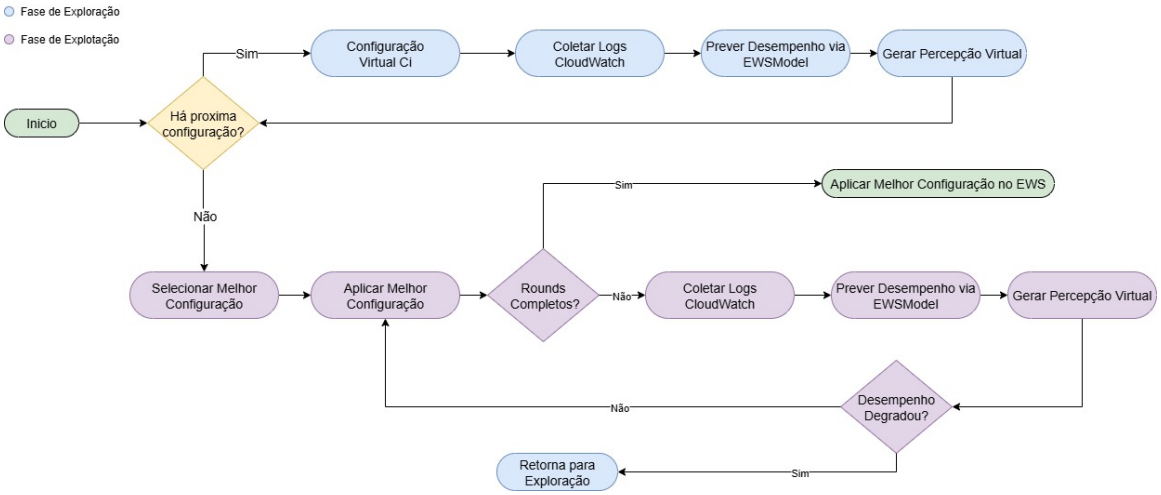

Figura 10: Fluxo do processo de aprendizado não-intrusivo do Digital Learning

5.1 Desempenho no Ambiente Local

Nos testes com recursos textuais, observamos que o DT cumpre o seu propósito sobre a abordagem tradicional, pois, enquanto a versão sem DT apresenta flutuações significativas de desempenho durante a fase de exploração, a com DT manteve uma operação estável, convergindo rapidamente para uma configuração ótima que reduziu o tempo médio de resposta de 16ms para 9ms (Figura 11). Este resultado é particularmente relevante pois demonstra que o DT não apenas evita a degradação de serviço durante o aprendizado, mas também identifica configurações que otimizam substancialmente o desempenho do servidor físico. Para recursos de imagem, a efetividade do DT foi ainda mais pronunciada. O sistema

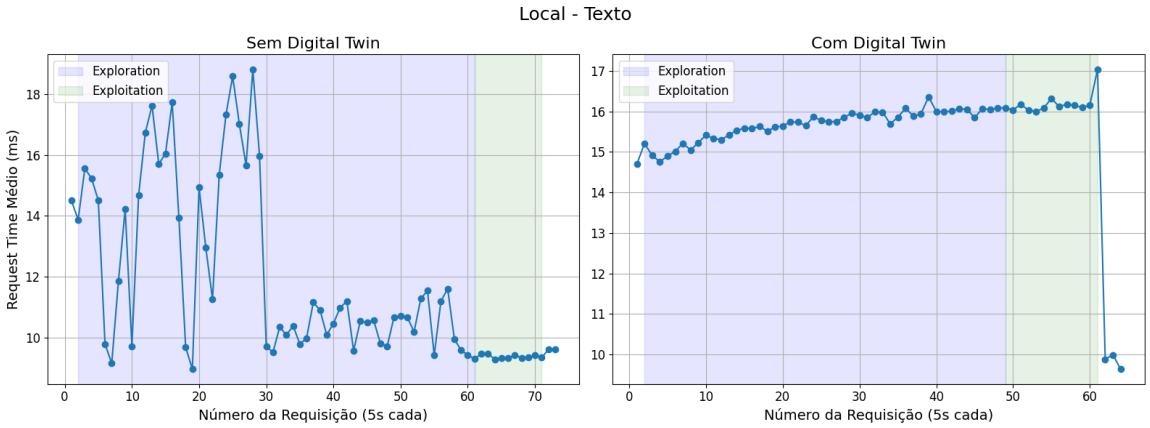

Figura 11: Comparativo de desempenho para recursos textuais (ambiente local)

tradicional operava com tempos médios de 150ms na configuração padrão, enquanto o DT conseguiu identificar e aplicar uma configuração que reduziu esse valor para menos que

20ms. A Figura 12 mostra claramente como o DT elimina os picos de latência durante a fase de exploração, estabilizando rapidamente o sistema no patamar de desempenho ótimo.

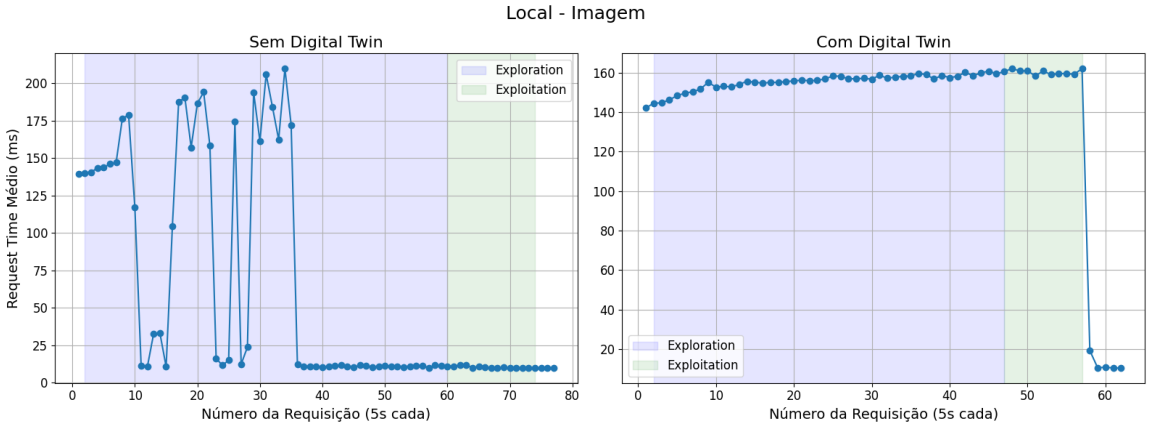

Figura 12: Comparativo de desempenho para recursos de imagem (ambiente local)

5.2 Desempenho em Nuvem AWS

Os resultados em ambiente de nuvem apresentaram características distintas. Para recursos textuais (Figura 13), a alta latência de rede (superior a 200ms) ofuscou os ganhos proporcionados pelo DT. Não se pôde avaliar o devido ganho da solução, pois a diferença dos tempos de resposta da pior e da melhor configuração do EWS são muito menores do que o tempo de latência de rede. Contudo, para recursos de imagem na nuvem (Figura 14), foi

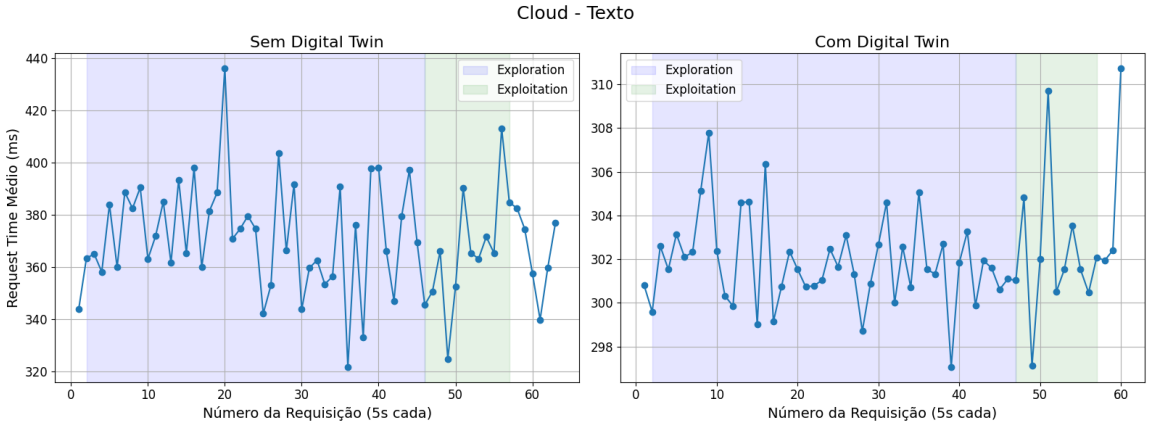

Figura 13: Comparativo de desempenho para recursos textuais (nuvem AWS)

possível visualizar o ganho da solução com o DT. Enquanto a implementação tradicional do EWS falhou em convergir para a configuração ótima, mantendo tempos de resposta entre

700-800ms, o DT identificou a melhor configuração, reduzindo o tempo médio para 300ms. Este caso é especialmente significativo por duas razões: primeiro, comprova que o DT converge para a melhor configuração em ambientes complexos, e, segundo, demonstra que, mesmo em infraestruturas cloud com latência elevada, o DT consegue eliminar a oscilação de performance durante o aprendizado.

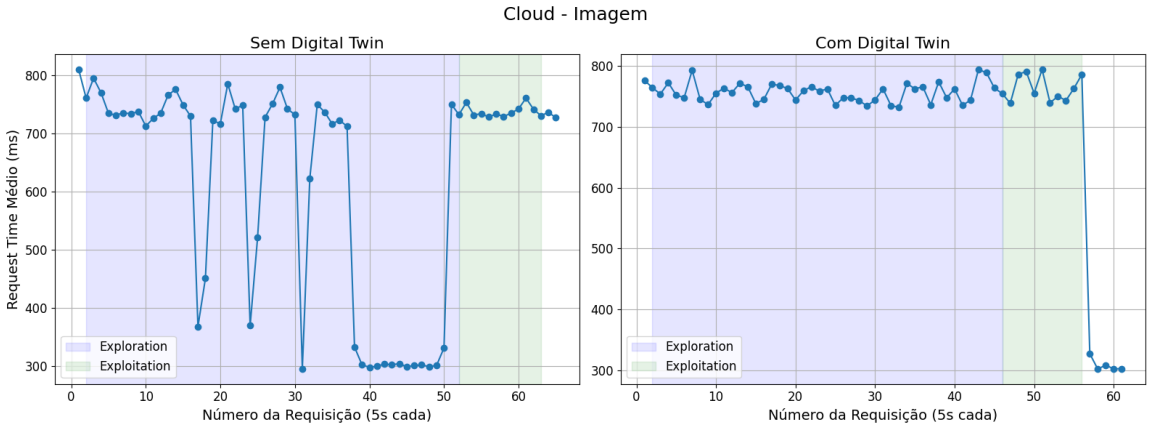

Figura 14: Comparativo de desempenho para recursos de imagem (nuvem AWS)

6 Conclusão e Trabalhos Futuros

Este trabalho demonstrou a viabilidade de um gêmeo digital para o Emergent Web Server, cumprindo os objetivos propostos ao integrar os frameworks PAL (sistemas emergentes) e MODA (gêmeos digitais). Por meio da implementação de um serviço autônomo de Reinforcement Learning, acoplado a um modelo preditivo de desempenho, validamos experimentalmente que o gêmeo digital reduz significativamente a oscilação de desempenho do sistema real durante a fase de aprendizado. Os resultados mostram reduções drásticas nos tempos de resposta: de 150ms para menos de 20ms em recursos de imagem no ambiente local, e de 700-800ms para 300ms na nuvem AWS, comprovando que o sistema não apenas modela fielmente o comportamento adaptativo do EWS, mas efetivamente otimiza sua operação em cenários dinâmicos.

A análise comparativa respondeu afirmativamente às questões centrais da pesquisa: (1) a modelagem computacional do comportamento adaptativo foi alcançada por meio da replicação precisa do núcleo de tomada de decisão do EWS; e (2) a oscilação de desempenho foi reduzida durante a fase de aprendizado, eliminando os picos de latência durante fases de exploração. Particularmente relevante foi o desempenho do gêmeo digital na nuvem, na qual superou o mecanismo tradicional ao convergir consistentemente para configurações ótimas mesmo quando o EWS nativo falhou nesta tarefa.

A solução desenvolvida neste projeto oferece benefícios operacionais tangíveis: mitigação de riscos durante aprendizado e estabilidade de desempenho em cenários dinâmicos. Embora limitações relacionadas à latência de rede persistam em ambientes na nuvem, a arquitetura

proposta abre caminho para aplicações em sistemas críticos em que a adaptabilidade e a estabilidade são requisitos complementares essenciais, validando o gêmeo digital como uma abordagem efetiva para a geração de sistemas adaptativos.

Como trabalhos futuros, propõe-se: aprimorar o EWSModel com a inclusão de novos algoritmos de ML alternativos para otimizar previsibilidade; expandir o *Data Shadow* mediante a incorporação de métricas adicionais; explorar novos modelos de aprendizado; e ampliar a bateria de testes com *clients* diversificados, simulando cenários extremos de carga e padrões de acesso heterogêneos, para validar a robustez do sistema em condições operacionais diversas.

Referências

[1] PORTER, B.; FILHO, R. R. **Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning**. In: 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Augsburg, Germany, 2016. Anais... IEEE, 2016. p. 40-49. DOI: 10.1109/SASO.2016.10.

[2] ELKHODARY, A.; ESFAHANI, N.; MALEK, S. **FUSION: A framework for engineering self-tuning self-adaptive software systems**. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '18), Santa Fe, NM, USA, 2010. Anais... New York: ACM, 2010. p. 7-16. DOI: 10.1145/1882291.1882296.

[3] de Lemos, R. et al. **Software Engineering for Self-Adaptive Systems: A Second Research Roadmap**. Springer, 2013.

[4] SALEHIE, M.; TAHVILDARI, L. **Self-adaptive software: Landscape and research challenges**. ACM Transactions on Autonomous and Adaptive Systems, v. 4, n. 2, p. 1-42, 2009. DOI: 10.1145/1516533.1516538.

[5] FILHO, Roberto Rodrigues. **Emergent Software Systems**. 2018. Tese (Doutorado em Ciência da Computação) – Lancaster University, Lancaster, Reino Unido, 2018.

[6] ERAMO, R. et al. **Conceptualizing Digital Twins**. IEEE Software, v. 39, n. 2, p. 39-46, 2022. DOI: 10.1109/MS.2021.3130755.

[7] KEPHART, J. O.; CHESS, D. M. **The vision of autonomic computing**. Computer, v. 36, n. 1, p. 41-50, 2003. DOI: 10.1109/MC.2003.1160055.

[8] RODRIGUES FILHO, R. et al. **Emergent software systems: Theory and practice**. In: Minicursos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'21), Evento Online, 2021. Anais... Porto Alegre: Sociedade Brasileira de Computação, 2021. p. 1-50.

[9] OSWALDO, G. H. R. et al. **Gestão transparente do estado para sistemas autodistribuídos**. Campinas: Instituto de Computação, UNICAMP, 2021.

[10] PROJECT DANA. **Dana Programming Language**. [Online]. Disponível em: `https://www.projectdana.com`. Acesso em: 9 jul. 2025.