AVALIAÇÃO DE UMA ARQUITETURA HÍBRIDA BORDA-NUVEM PARA MONITORAMENTO DE MOTORES ELÉTRICOS EM TEMPO REAL

Guilherme Samuel[*]
Diego Luiz Kreutz[**]

RESUMO

Propomos uma arquitetura híbrida Borda–Nuvem para detecção em tempo real de falhas em motores elétricos industriais. A análise local fornece alertas com latência mediana de 16 ms, enquanto agregações periódicas são enviadas à nuvem. Os resultados mostram que a borda supera amplamente a nuvem em estabilidade e latência, com uso eficiente de CPU e memória mesmo sob múltiplos sensores.

Palavras-chaves: Arquitetura de Software; Computação em Borda.

ABSTRACT

We propose a hybrid Edge–Cloud architecture for real-time fault detection in industrial electric motors. Local analysis generates alerts with a median latency of 16 ms, while periodic aggregations are sent to the cloud. The results show that edge processing significantly outperforms the cloud in stability and latency, maintaining efficient CPU and memory usage even with multiple sensors.

Keywords: Software Architecture; Edge Computing.

1. INTRODUÇÃO

Motores elétricos são elementos centrais da infraestrutura industrial e sua interrupção, mesmo que por poucos minutos, pode resultar em perdas financeiras substanciais e impactos operacionais significativos. Estima-se que paradas não-planejadas custem entre 25.000 e 40.000 euros por hora[1]. Além disso, estudos indicam que entre 20% e 25% dos motores utilizados em ambientes industriais são considerados críticos para a operação e apresentam taxas de falha anual que podem atingir 7% (ALBRECHT et al., 1986). Tais falhas decorrem de diferentes fatores, como sobrecarga elétrica, desgaste ou desalinhamento em rolamentos e aumento crítico de temperatura.

A adoção de estratégias de manutenção proativa tem se mostrado eficaz para reduzir custos e mitigar impactos operacionais. Essas estratégias dependem da coleta contínua de medições físicas dos motores e da análise inteligente desses dados para identificar indícios de degradação antes que uma falha se torne crítica. Evidências mostram que abordagens proativas permitem reduzir custos de manutenção em até 30%, diminuir o tempo de inatividade em até 45% e reduzir a incidência de falhas

[1] <https://shorturl.at/K1Lcv>

[*] Aluno do Curso de Engenharia de Software da Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil
E-mail: guilhermesiqueira.aluno@unipampa.edu.br
[**] Orientador, Professor do Curso de Engenharia de Software da Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil, E-mail: kreutz@unipampa.edu.br

em até 75%[2]. No entanto, implementar tais capacidades em escala industrial impõe desafios arquiteturais importantes, sobretudo em cenários que envolvem centenas de sensores operando simultaneamente e dispositivos com recursos computacionais restritos.

Diante desse contexto, este trabalho propõe uma arquitetura de processamento em dispositivos de borda-nuvem capaz de detectar em tempo real indícios de falhas em motores elétricos, preservando baixa latência e reduzindo a dependência de comunicação com a nuvem. A arquitetura também permite enviar agregações periódicas de dados para um ambiente em nuvem, possibilitando armazenamento histórico e análises posteriores.

2. TRABALHOS RELACIONADOS

A Indústria 4.0 intensificou a integração entre sistemas físicos e digitais, ampliando o volume de dados gerados por sensores e impulsionando estratégias de manutenção orientadas por dados. Nesse contexto, a Manutenção Proativa (PdM) utiliza análises estatísticas e técnicas de *machine learning* para antecipar falhas (ANDRIULO et al., 2024), o que exige arquiteturas capazes de lidar com fluxos contínuos em larga escala. A literatura convergiu para três abordagens principais: soluções baseadas apenas em nuvem, que oferecem alta capacidade computacional mas dependem de redes sujeitas a latência e variabilidade (SAINI; YADAV; RAHMAN, 2024; LEE; CHA, 2016; LI et al., 2020); arquiteturas *edge-only*, que realizam processamento local com baixa latência e maior proteção de dados, embora tendam a criar silos informacionais; e avanços recentes como *TinyML*, que ampliam as capacidades da borda mas ainda apresentam limitações de complexidade (SANCHEZ-IBORRA et al., 2023; PANDEY et al., 2023).

Arquiteturas híbridas surgem como alternativa para conciliar as vantagens desses modelos, combinando a reatividade da borda com a escalabilidade da nuvem (DUC; NGUYEN; ÖSTBERG, 2025; SEPE et al., 2021; PILLAI et al., 2016). Nesse arranjo, decisões imediatas podem ser tomadas localmente, enquanto a nuvem armazena dados históricos e executa modelos mais robustos. Este trabalho se insere nesse cenário ao propor e avaliar uma arquitetura híbrida borda-nuvem para detecção em tempo real de alertas em motores elétricos, examinando seu desempenho sob diferentes cargas e considerando consumo de recursos, latência fim a fim, número de sensores e influência da janela de agregação.

3. CONTEXTO

Para validar a arquitetura proposta, foi escolhido o domínio do monitoramento de motores elétricos industriais, com foco em manutenção preditiva. A escolha foi feita pelo fato deste setor apresentar desafios que testam diretamente os pilares desta arquitetura: (1) a alta demanda por detecção de alertas em tempo real, validando a latência local; e (2) os desafios de alto volume e complexidade computacional (*e.g.* análise de frequência de rolamentos de motores) na coleta e processamento dos dados, validando a eficiência e o impacto no uso de recursos na borda.

[2]<https://shorturl.at/EM2As>

3.1. MANUTENÇÃO

De forma geral, o conceito de manutenção se refere ao conjunto de todas as ações possíveis de serem realizadas com o objetivo de restaurar um componente ou o sistema como um todo ao seu estado original de pleno funcionamento, pelo qual pode performar suas funções (MAZZUCHI; NOORTWIJK; KALLEN, 2007). Esse conceito também pode ser dividido em dois grupos: manutenção proativa e manutenção corretiva (ERBIYIK, 2022). A manutenção proativa tem por objetivo aplicar as ações de correção em um componente antes que uma falha crítica ocorra. Essas ações podem ser aplicadas de forma planejada, dentro de uma janela de tempo adequada (manutenção planejada), ou através de monitoramento e análise contínuas em diferentes aspectos do componente (manutenção preditiva). Essa abordagem permite resulta em menor tempo de indisponibilidade e custo em termos de mão de obra; no entanto, possui maior complexidade de implementação e custo computacional (BASRI et al., 2017). A manutenção corretiva (ou reativa), por outro lado, é aplicada somente após a constatação de uma falha no componente, mediante observação ou detecção (ERBIYIK, 2022). Essa abordagem é mais simples e com menor custo inicial, mas com maior custo na ocorrência de falhas, considerando que esta pode afetar outros componentes e até o sistema como um todo, tornando a manutenção mais complexa. Além disso, a chance de indisponibilidade do sistema é maior, o que aumenta o custo da falha.

3.2. ANÁLISE DE MEDIÇÕES

A arquitetura proposta, baseada em manutenção preditiva, permite a implementação de métricas analíticas em diferentes propriedades do motor. Para realisticamente simular um sensor, as propriedades utilizadas são:

- **Vibração:** A análise da vibração de componentes de motores elétricos pode ser feita através da decomposição espectral, utilizando métodos como o *Fast Fourier Transform* (FFT), que permite identificar padrões associados a falhas específicas, como defeitos em rolamentos, desalinhamentos de eixo, desbalanceamento, etc (PATIL; GAIKWAD, 2013). Para detectar, especificamente, desgaste nos rolamentos, o sinal de vibração foi analisado de duas formas: (1) cálculo do RMS (*root mean square*), que indica aumento da energia global da vibração; e (2) uma FFT é aplicada para verificar se há um pico de amplitude na frequência característica do rolamento (BPFO), típica desse tipo de falha. Se ambos os valores ultrapassam limiares parametrizáveis, ajustados à velocidade nominal do motor, um alerta de desgaste é gerado (KULKARNI; BEWOOR, 2016).
- **Corrente elétrica:** A análise do sinal produzida pela corrente elétrica que flui pelo motor permite identificar anomalias eletromecânicas sem necessidade de um sensor externo adicional (LI; MECHEFSKE, 2006). Da mesma forma da análise da vibração, é possível utilizar a FFT para decompôr o sinal espectral: para detectar, especificamente, sobrecarga nos componentes elétricos, o algoritmo utilizado: (1) calcula o RMS da corrente, que tende a aumentar quando o motor está exige mais torque do que deveria; (2) usa a FFT para medir a presença de harmônicos, especialmente a 3ª harmônica, que cresce quando o motor opera com distorção devido à sobrecarga (TOLIYAT et al., 2012). Se o RMS excede um

limite baseado na corrente nominal e a proporção entre a 3ª harmônica e o componente fundamental ultrapassa um limite, um alerta de sobrecarga é gerado.

- **Temperatura:** A temperatura é um indicador direto de sobrecarga térmica e possíveis problemas de isolamento. É interessante também medir a temperatura em conjunto com outras variáveis, como corrente e vibração, pois o aumento da temperatura pode ocorrer por motivos diferentes (*e.g.* aquecimento por sobrecorrente ou problema mecânico) (JAROS et al., 2023). A temperatura, diferentemente da análise de vibração e corrente, pode ser analisada observando apenas se um valor escalar ultrapassa um determinado limiar crítico; esse valor crítico varia com a classe de isolamento do motor, que indica a temperatura máxima que o isolamento do motor pode suportar sem se degradar (Leroy-Somer, 2014). O NEMA (*National Electrical Manufacturers Association*) define que as classes de isolamento A, B, F, H correspondem a temperaturas máximas de 105 °C, 130 °C, 155 °C e 180 °C, respectivamente (Drives and Automation Ltd, 2020).

4. METODOLOGIA

A metodologia adotada neste trabalho utiliza de uma abordagem experimental (WOHLIN et al., 2012), estruturada para avaliar sistematicamente o desempenho de uma arquitetura híbrida borda–nuvem aplicada ao contexto do monitoramento em tempo real de motores elétricos para suporte à manutenção proativa. A pesquisa é de natureza *aplicada*, pois busca produzir conhecimento com finalidade prática (GIL, 2008); possui também abordagem *quantitativa*, considerando a análise estatística de métricas de desempenho, tanto de latência de rede quanto em consumo de recursos computacionais (CRESWELL, 2014); e caracteriza-se como *experimental*, pois envolve manipulação controlada de variáveis em um ambiente configurado para observação (KERLINGER; LEE, 2000).

4.1. DELIMITAÇÃO DA PESQUISA

O estudo foi conduzido em três etapas metodológicas principais:

1. **Especificação do problema e construção do modelo conceitual**: fundamentou-se o estudo em literatura sobre manutenção proativa, principais desafios para a implementação de processamento em borda, arquiteturas de processamento distribuído e técnicas de análise de sinais. A partir disso, foi definido um conjunto de hipóteses operacionais relacionadas à latência de processamento e ao consumo de recursos na borda.
2. **Definição das variáveis experimentais**: foram definidas as variáveis independentes a serem utilizadas para direcionar a validação da proposta do trabalho, sendo: o número de sensores simulados e a taxa de geração de eventos. Como variáveis dependentes, estabeleceram-se: (a) uso de CPU, (b) uso de memória, (c) latência local de detecção, (d) latência total até a nuvem. Essa definição permitiu estruturar métricas objetivas e reproduzíveis.
3. **Execução dos experimentos**: os experimentos foram planejados de forma a isolar o comportamento do processamento local e os efeitos da transmissão para a nuvem. Cada execução seguiu um protocolo fixo: estado inicial ocioso, ativação

dos sensores, coleta contínua de métricas e registro dos eventos coletados para posterior análise estatística.

4.2. Procedimentos Experimentais

Os experimentos seguiram princípios de repetibilidade e parametrização, com o objetivo de garantir rigor científico . Cada cenário foi executado cinco vezes de forma independente, por aproximadamente 5 minutos. Durante as execuções, os dados foram coletados por ferramentas sistemáticas e bem testadas como `docker stats` (para consumo de recursos) e registros temporais de eventos, para análise da latência, decomposta entre cada etapa do processamento. Ao utilizar de sensores simulados, foi possível controlar fatores externos e dar enfoque na validação da arquitetura propriamente dita.

A análise da latência foi dividida em dois componentes:

- **latência interna**, correspondente ao tempo de detecção de alertas críticos no dispositivo de borda;
- **latência externa**, correspondente ao tempo de transmissão, roteamento e ingestão no ambiente em nuvem.

Essa decomposição metodológica permitiu identificar gargalos específicos e compreender a contribuição relativa de cada etapa.

4.3. Métodos de Análise dos Dados

A análise dos resultados seguiu uma abordagem quantitativa. Foram aplicadas estatísticas descritivas básicas (mediana, desvio padrão, identificação de valores extremos e percentis) para caracterizar a distribuição das métricas coletadas. Para investigações complementares, como a identificação de variações na comunicação de rede, foi utilizada o `mtr`, uma ferramenta de diagnóstico de rota.

A interpretação dos resultados foi guiada pelas hipóteses formuladas na etapa conceitual, permitindo avaliar:

- a estabilidade do processamento local sob aumento do volume de dados a serem processados;
- a escalabilidade da arquitetura;
- a variabilidade induzida pela rede na comunicação com a nuvem.

4.4. Cronograma de Execução

O desenvolvimento do trabalho seguiu o cronograma apresentado na Tabela 1.

Tabela 1 - Cronograma de execução do trabalho

Atividade	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Delimitação do tema de pesquisa	X					
Aprofundamento do domínio do problema		X				
Planejamento da implementação da arquitetura		X				
Implementação da arquitetura			X	X		
Realização de experimentos					X	
Escrita do artigo					X	
Defesa do trabalho						X

5. ARQUITETURA E IMPLEMENTAÇÃO

A arquitetura de dados proposta é baseada no padrão Kappa (KREPS, 2014), que centraliza o fluxo de informações em um único componente de *streaming* e elimina a separação entre processamento em lote e em tempo real, típica da arquitetura Lambda (WARREN; MARZ, 2015). Essa abordagem foi escolhida por sua simplicidade e eficiência, características adequadas a dispositivos de borda com recursos limitados e a cenários que exigem análise contínua com baixa latência. Como ilustrado na Figura 1, o fluxo tem início na camada de sensores, geralmente microcontroladores responsáveis por enviar medições periódicas de parâmetros físicos, que são recebidas por um *message broker* MQTT no próprio dispositivo de borda. Nesse ponto, os dados são processados e agregados, permitindo que um único *edge gateway* trate simultaneamente múltiplos sensores, emitindo alertas imediatos quando limites predefinidos são excedidos e reduzindo o tráfego para a nuvem por meio de agregações periódicas.

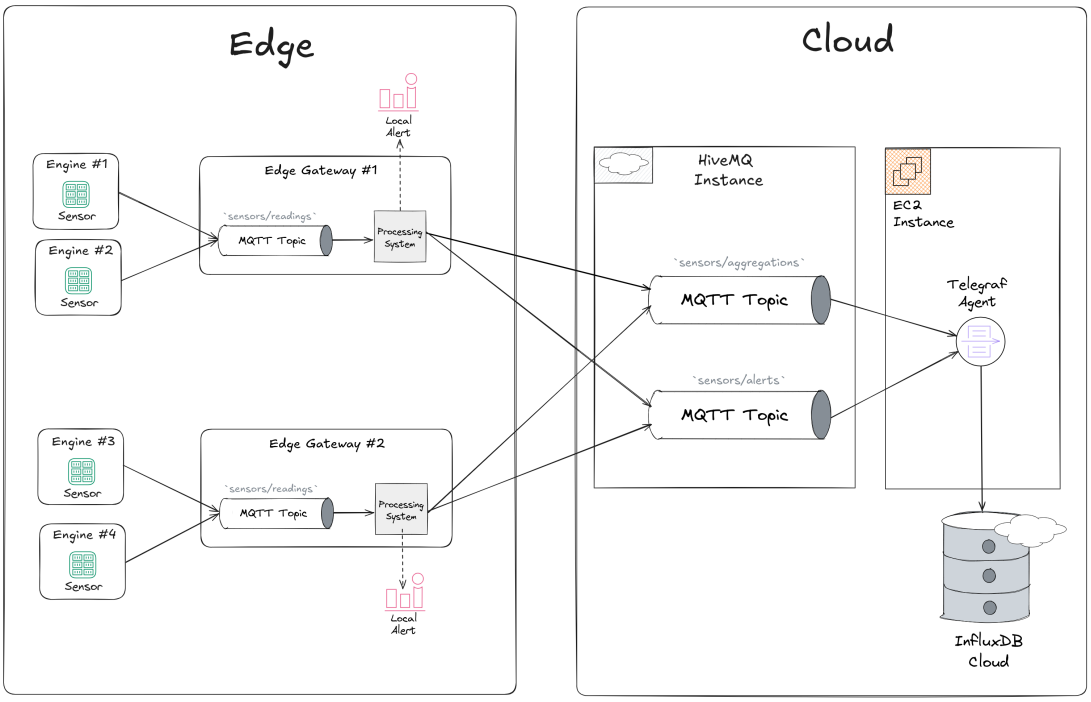

Figura 1 - Visão geral da arquitetura

Após o processamento local, a arquitetura encaminha as medições ao InfluxDB por meio do HiveMQ, enquanto o Telegraf atua como camada intermediária responsável por consumir os tópicos MQTT e padronizar a escrita no banco, garantindo modularidade e escalabilidade entre borda e nuvem. Análises complexas e treinamento de modelos de *machine learning* são mantidos na nuvem devido às restrições de ener-

gia, processamento e conectividade típicas de dispositivos de borda em ambientes remotos (SHI et al., 2016), permitindo monitoramento contínuo com baixo consumo e processamento avançado centralizado.

A Figura 2 mostra o diagrama de classes do módulo de processamento. A classe `EdgeApp` atua como elemento de coordenação entre sensores, processamento e nuvem. Ela mantém dois objetos `MQTTHandler`, um dedicado à recepção de eventos no tópico *sensors/readings* e outro responsável pela publicação de agregações e alertas nos tópicos *sensors/aggregations* e *sensors/alerts*. A classe também controla um conjunto de instâncias de `EventProcessor`, uma para cada sensor identificado, garantindo isolamento do estado de agregação e evitando interferências entre fluxos distintos. Cada `EventProcessor` utiliza a classe `Engine` para interpretar o comportamento esperado do motor e aplicar regras sobre os parâmetros medidos, o que permite personalizar limites e propriedades conforme cada tipo de equipamento. Essa modularidade possibilita que a arquitetura seja reutilizada em outros domínios, como saúde, energia ou mobilidade urbana, com modificações mínimas no código.

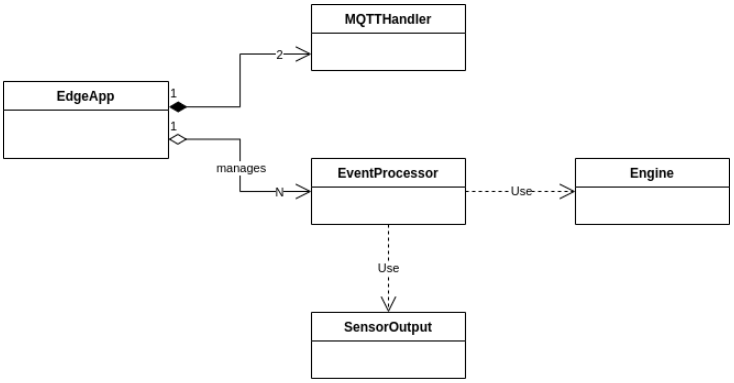

Figura 2 - Diagrama de classes do módulo de processamento

6. AVALIAÇÃO

Para avaliar a arquitetura proposta, foi conduzida uma série de experimentos quantitativos destinados a verificar a viabilidade do processamento em borda sob recursos limitados e identificar os principais gargalos de latência em uma arquitetura híbrida borda-nuvem. As análises buscaram determinar se o dispositivo de borda consegue processar medições de múltiplos sensores sem saturação de CPU ou memória e em que medida o processamento local se mostra mais rápido e previsível que a comunicação com a nuvem. Para isso, foi construído um ambiente de testes híbrido que reproduz características de um cenário industrial realista, permitindo observar tanto o comportamento do processamento local quanto o impacto da transmissão e ingestão de dados nos serviços em nuvem.

6.1. AMBIENTE

O ambiente de testes foi estruturado em dois componentes lógicos, o dispositivo de borda e os serviços em *cloud*. O dispositivo de borda foi simulado em um ambiente Docker executado em uma máquina *host* com processador 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, e teve seus recursos limitados via *docker-compose* para representar um hardware restrito, permitindo apenas 25% de um núcleo de CPU

e 256 MB de memória RAM. O container correspondente integra um *message broker* MQTT (Eclipse Mosquitto) e o módulo de processamento em Python 3.12.3, conforme ilustrado na Figura 1. A camada em *cloud* foi composta por uma instância do HiveMQ como *broker* centralizado, uma instância t2.micro do Amazon EC2 executando o agente Telegraf e uma instância do InfluxDB Cloud para armazenamento das séries temporais, escolhidas pela disponibilidade de uso gratuito e adequação à validação experimental da arquitetura.

6.2. EXPERIMENTOS

Para responder à primeira questão apresentada no início desta seção, foram coletadas estatísticas de consumo de recursos dos containers Docker por meio do comando `docker stats`. O objetivo foi observar como CPU e memória variam conforme aumenta o número de sensores cujos eventos precisam ser processados por um mesmo dispositivo de borda. Para isso, foram realizadas cinco execuções independentes, cada uma com um número crescente de sensores atribuídos ao mesmo *edge gateway*, durante um período de cinco minutos. Nos primeiros 30 segundos, o envio de dados permaneceu desativado, permitindo medir o consumo em estado inativo. A partir desse ponto, os sensores passaram a transmitir eventos, o que provocou um aumento imediato tanto no uso de CPU quanto no de memória. Esse comportamento se explica pela necessidade simultânea de realizar cálculos de detecção de alertas, tarefa intensiva em CPU, e acumular eventos em memória para compor agregações em janelas de 10 segundos.

A Figura 3 mostra que a janela de agregação introduz um comportamento cíclico no uso de memória e CPU do container. A memória cresce enquanto eventos são acumulados no buffer e retorna ao nível inicial após cada limpeza periódica, enquanto a CPU acompanha esse ciclo em função do processamento das agregações. Esse padrão reflete a natureza incremental do mecanismo de buffering, que concentra operações mais intensivas apenas nos pontos de consolidação.

O aumento do número de sensores eleva o volume de mensagens e o total de eventos por janela, o que explica o crescimento moderado do consumo de memória. Mesmo no cenário mais intenso, com cinco motores e cerca de 116 MiB de utilização, os valores permanecem significativamente abaixo dos limites do container, enquanto a execução com um único motor se mantém em torno de 90 MiB. A estabilidade desse patamar inicial indica um custo base dominante do motor de processamento, com acréscimos proporcionais à carga, mas sem introduzir sobrecarga relevante. Em conjunto, esses resultados demonstram que o modelo de buffering é eficiente, escalável e adequado para cenários com múltiplos sensores.

Para responder à segunda questão e identificar possíveis gargalos, foi conduzida uma análise estatística da latência fim a fim com base em 6853 alertas. Os registros de tempo permitiram decompor a latência total em duas parcelas: o tempo de processamento local utilizado para a detecção e o tempo relacionado à transmissão dos dados até a nuvem e à ingestão pelos serviços remotos. Essa separação possibilita isolar o impacto da rede e comparar diretamente o desempenho da borda com o comportamento dos serviços em nuvem, conforme ilustrado na Figura 4.

A Figura 4 mostra que o processamento local apresenta latência significativamente menor do que a observada na comunicação com a nuvem. O painel da es-

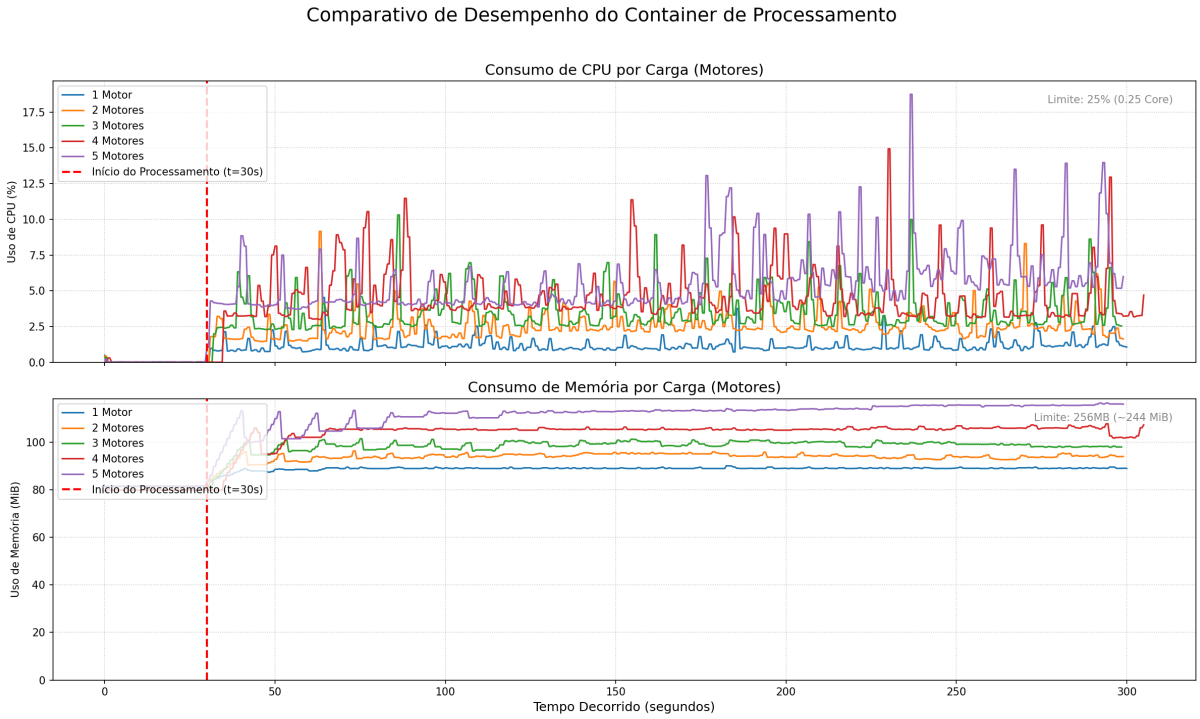

Figura 3 - Consumo de recursos computacionais

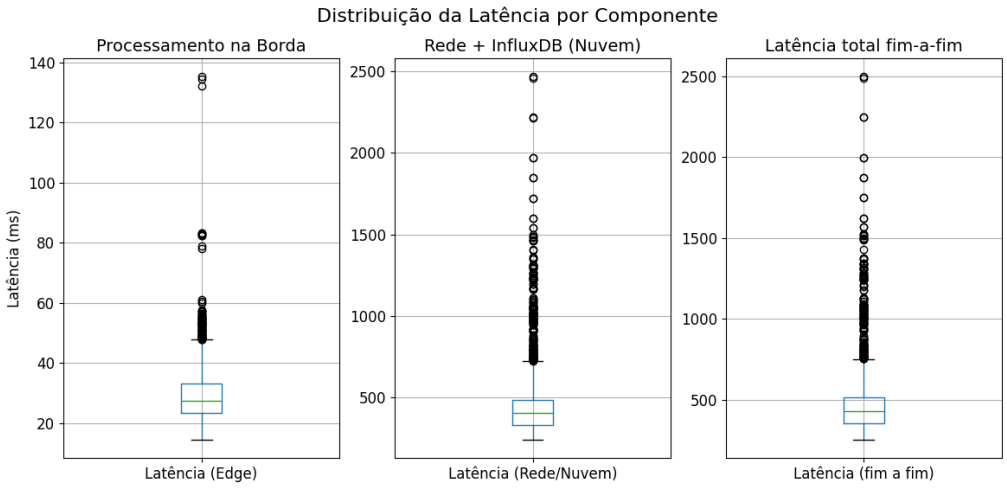

Figura 4 - Distribuição da latência por componente

querda revela comportamento estável, com mediana de 16 ms e desvio padrão de 5 ms, indicando que a detecção ocorre de forma rápida e com baixa variabilidade. Embora alguns valores atípicos atinjam aproximadamente 64 ms, a análise da Tabela 6.2 mostra que cerca de 70% do tempo total é gasto na desserialização dos eventos recebidos pelo tópico MQTT, enquanto a etapa de detecção representa cerca de 18% do tempo de execução. Essa etapa envolve operações computacionalmente intensivas, como o cálculo de transformadas rápidas de Fourier em sequências com dezenas de milhares de elementos por segundo, o que explica seu custo relativo dentro do fluxo de processamento.

Tabela 2 - Resumo das etapas do processamento no edge

Etapa	Min	Mediana	p99	Max	Proporção (%)
Desserialização	2.75	4.48	12.55	78.32	71.72
Serialização	0.01	0.03	0.05	1.77	0.73
Detecção	0.65	0.95	2.70	77.23	18.26
Agregações	6.25	7.05	14.18	15.69	9.29

Em contraste, o painel da direita evidencia a rede como principal fonte de variabilidade, com latências que podem atingir 2500 ms. A análise da topologia de comunicação, obtida por meio do comando `mtr -rwz -c <EC2-IP>`, mostrou um trajeto extenso entre o dispositivo de borda e a instância EC2 que executa o Telegraf, com oscilações significativas de RTT já nos primeiros saltos da rede do provedor. Foram observados picos superiores a 1000 ms ainda dentro da infraestrutura local do ISP, o que indica que a maior parte da instabilidade ocorre antes do tráfego alcançar o *backbone* internacional ou a rede da AWS. Apesar da latência elevada, não houve perda de pacotes, o que caracteriza um caso típico de *bufferbloat* (GETTYS; NICHOLS, 2012).

Hop	Host	Loss%	Snt	Last	Avg	Best	Wrst
1	_gateway	0.0%	100	1.6	3.8	1.2	188.4
2	xxx.user3p.v-tal.net.br	0.0%	100	7.6	4.3	3.0	9.3
3	100.120.xxx.xxx	0.0%	100	4.8	14.9	2.5	1006.0
4	100.120.xxx.xxx	0.0%	100	12.0	12.3	9.7	23.2
5	100.120.xxx.xxx	0.0%	100	21.3	20.6	18.4	27.1
6	100.120.xxx.xxx	1.0%	100	28.4	28.5	25.5	50.8
7	100.120.xxx.xxx	0.0%	100	24.0	24.4	22.1	55.0
8	201.10.xxx.xxx	1.0%	100	27.3	26.7	24.9	29.4
9	200.16.69.xxx	1.0%	100	33.8	35.2	31.2	69.5
10	200.16.69.xxx	0.0%	100	70.7	70.7	68.6	93.6
11	200.16.69.xxx	0.0%	100	140.6	140.7	137.6	178.2
12	*** ofuscado ***	100.0%	100	0.0	0.0	0.0	0.0
13	*** ofuscado ***	100.0%	100	0.0	0.0	0.0	0.0
14	*** ofuscado ***	100.0%	100	0.0	0.0	0.0	0.0
15	*** ofuscado ***	100.0%	100	0.0	0.0	0.0	0.0
16	*** ofuscado ***	100.0%	100	0.0	0.0	0.0	0.0
17	ec2.compute-1.amazonaws.com	0.0%	100	286.9	149.2	145.5	286.9

Tabela 3 - Topologia da rede fim-a-fim

7. CONSIDERAÇÕES FINAIS

A arquitetura proposta mostrou capacidade de processar medições de múltiplos sensores em tempo real, detectar anomalias com baixa latência e enviar agregações compactas para a nuvem, mantendo desempenho eficiente mesmo sob carga crescente. Como continuidade, pretende-se validá-la em um dispositivo de borda físico,

como um Raspberry Pi, e explorar modelos de *machine learning* e estratégias híbridas que integrem inferência local e treinamento em nuvem.

REFERÊNCIAS

ALBRECHT, P. et al. Assessment of the reliability of motors in utility applications-updated. **IEEE Transactions on Energy conversion**, IEEE, p. 39–46, 1986.

ANDRIULO, F. C. et al. Edge computing and cloud computing for internet of things: A review. In: **Informatics**. [S.l.: s.n.], 2024.

BASRI, E. I. et al. Preventive maintenance (pm) planning: a review. **Journal of quality in maintenance engineering**, Emerald Publishing Limited, v. 23, n. 2, p. 114–143, 2017.

CRESWELL, J. W. **Research design: qualitative, quantitative, and mixed methods approaches**. 4. ed. Thousand Oaks: Sage, 2014.

Drives and Automation Ltd. **NEMA Insulation Classes for Motors**. 2020. <https://www.drivesandautomation.co.uk/useful-information/nema-insulation-classes/>. Acesso em: 13 nov. 2025.

DUC, T. L.; NGUYEN, C.; ÖSTBERG, P.-O. Workload prediction for proactive resource allocation in large-scale cloud-edge applications. **Electronics**, MDPI, v. 14, n. 16, p. 3333, 2025.

ERBIYIK, H. Definition of maintenance and maintenance types with due care on preventive maintenance. In: **Maintenance Management-Current Challenges, New Developments, and Future Directions**. [S.l.]: IntechOpen, 2022.

GETTYS, J.; NICHOLS, K. Bufferbloat: dark buffers in the internet. **Communications of the ACM**, ACM New York, NY, USA, v. 55, n. 1, p. 57–65, 2012.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008.

JAROS, R. et al. Advanced signal processing methods for condition monitoring. **Archives of Computational Methods in Engineering**, Springer Nature BV, v. 30, n. 3, p. 1553–1577, 2023.

KERLINGER, F. N.; LEE, H. B. **Foundations of Behavioral Research**. 4. ed. Fort Worth: Harcourt College Publishers, 2000.

KREPS, J. Questioning the lambda architecture. **Online article, July**, v. 205, p. 18–34, 2014.

KULKARNI, S.; BEWOOR, A. Vibration based condition assessment of ball bearing with distributed defects. **Journal of Measurements in Engineering**, JVE International Ltd., v. 4, n. 2, p. 87–94, 2016.

LEE, H.; CHA, J. H. New stochastic models for preventive maintenance and maintenance optimization. **European Journal of Operational Research**, Elsevier, v. 255, n. 1, p. 80–90, 2016.

Leroy-Somer. **Insulation Class / Temperature Rise Class – Technical Note TN11**. [S.l.], 2014. Acesso em: 13 nov. 2025. Disponível em: <https://www.leroy-somer.com/documentation_pdf/5202_en.pdf>.

LI, T. et al. Federated learning: Challenges, methods, and future directions. **IEEE signal processing magazine**, IEEE, v. 37, n. 3, p. 50–60, 2020.

LI, W.; MECHEFSKE, C. Induction motor fault detection using hybrid methods. In: SPRINGER. **Engineering Asset Management: Proceedings of the 1st World Congress on Engineering Asset Management (WCEAM) 11–14 July 2006**. [S.l.], 2006. p. 276–287.

MAZZUCHI, T. A.; NOORTWIJK, J. M. V.; KALLEN, M.-J. Maintenance optimization. **Encyclopedia of Statistics in Quality and Reliability**, p. 1000–1008, 2007.

PANDEY, R. et al. Towards deploying dnn models on edge for predictive maintenance applications. **Electronics**, MDPI, v. 12, n. 3, p. 639, 2023.

PATIL, S.; GAIKWAD, J. Vibration analysis of electrical rotating machines using fft: A method of predictive maintenance. In: IEEE. **2013 fourth international conference on computing, communications and networking technologies (ICCCNT)**. [S.l.], 2013. p. 1–6.

PILLAI, P. et al. A hybrid approach for fusing physics and data for failure prediction. **International Journal of Prognostics and Health Management**, v. 7, n. 4, 2016.

SAINI, N.; YADAV, A. L.; RAHMAN, A. Cloud based predictive maintenance system. In: IEEE. **ICRITO**. [S.l.], 2024. p. 1–5.

SANCHEZ-IBORRA, R. et al. Intelligent and efficient iot through the cooperation of tinyml and edge computing. **Informatica**, SAGE Publications Sage UK: London, England, v. 34, n. 1, p. 147–168, 2023.

SEPE, M. et al. A physics-informed machine learning framework for predictive maintenance applied to turbomachinery assets. **J. of the Global Power and Propulsion Society**, v. 2021, 2021.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE internet of things journal**, IEEE, v. 3, n. 5, p. 637–646, 2016.

TOLIYAT, H. A. et al. **Electric machines: modeling, condition monitoring, and fault diagnosis**. [S.l.]: CRC press, 2012.

WARREN, J.; MARZ, N. **Big Data: Principles and best practices of scalable realtime data systems**. [S.l.]: Simon and Schuster, 2015.

WOHLIN, C. et al. **Experimentation in Software Engineering**. Berlin: Springer, 2012.