Avaliação de uma Arquitetura Híbrida Borda-Nuvem para Monitoramento de Motores Elétricos em Tempo Real

Guilherme Siqueira, Diego Kreutz[1]

[1]LEA e AI Horizon Labs – Universidade Federal do Pampa (UNIPAMPA)

{GuilhermeSiqueira.aluno,diegokreutz}@unipampa.edu.br

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.*

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.*

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 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

[1]https://shorturl.at/K1Lcv
[2]https://shorturl.at/EM2As

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 et al. 2024, Lee and 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 et al. 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. 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 and 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.

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 energia, processamento e conectividade típicas de dispositivos de borda em ambientes remotos [Shi et al. 2016],

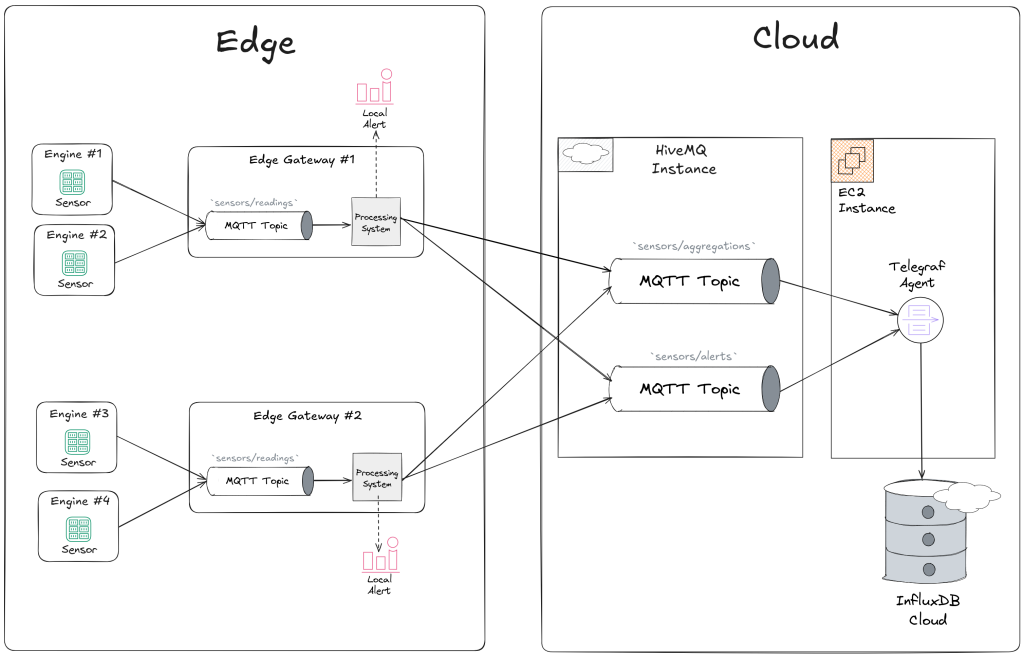

Figura 1. Visão geral da arquitetura

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.

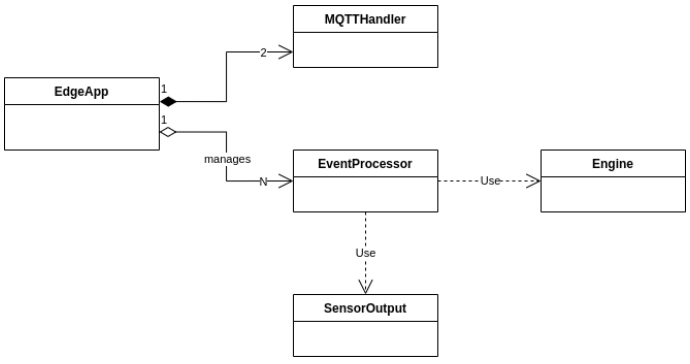

Figura 2. Diagrama de classes do módulo de processamento

4. 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.

4.1. 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.

4.2. 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

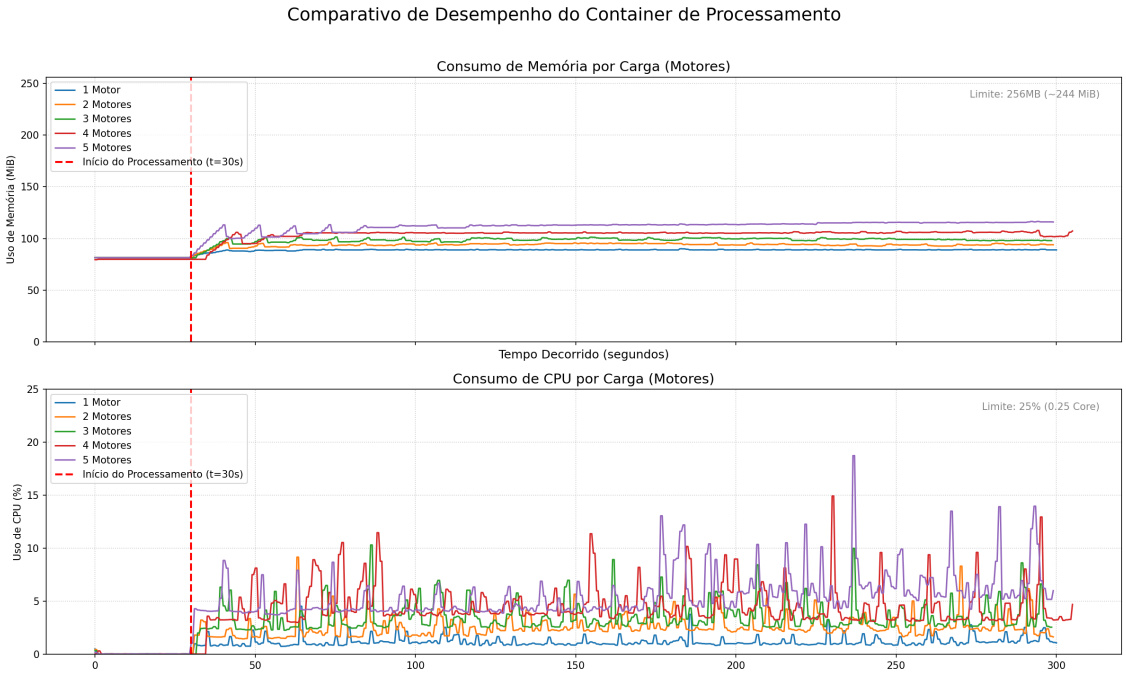

Figura 3. Consumo de recursos computacionais

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 esquerda 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 4.2 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.

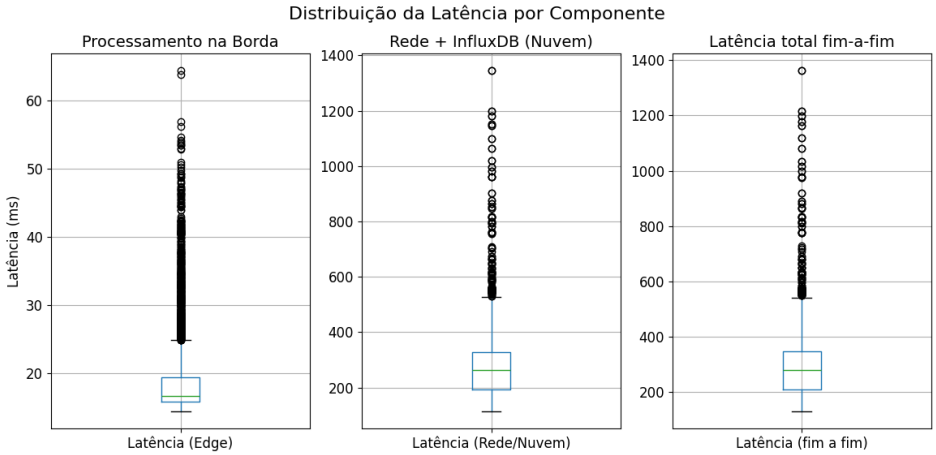

Figura 4. Distribuição da latência por componente

Tabela 1. 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

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 2. Topologia da rede fim-a-fim

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 and Nichols 2012].

5. 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., Appiarius, J., McCoy, R., Owen, E., and Sharma, D. (1986). Assessment of the reliability of motors in utility applications-updated. *IEEE Transactions on Energy conversion*, pages 39–46.

Andriulo, F. C., Fiore, M., Mongiello, M., Traversa, E., and Zizzo, V. (2024). Edge computing and cloud computing for internet of things: A review. In *Informatics*.

Duc, T. L., Nguyen, C., and Östberg, P.-O. (2025). Workload prediction for proactive resource allocation in large-scale cloud-edge applications. *Electronics*, 14(16):3333.

Gettys, J. and Nichols, K. (2012). Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1):57–65.

Kreps, J. (2014). Questioning the lambda architecture. *Online article, July*, 205:18–34.

Lee, H. and Cha, J. H. (2016). New stochastic models for preventive maintenance and maintenance optimization. *European Journal of Operational Research*, 255(1):80–90.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60.

Pandey, R., Uziel, S., Hutschenreuther, T., and Krug, S. (2023). Towards deploying dnn models on edge for predictive maintenance applications. *Electronics*, 12(3):639.

Pillai, P., Kaushik, A., Bhavikatti, S., Roy, A., and Kumar, V. (2016). A hybrid approach for fusing physics and data for failure prediction. *International Journal of Prognostics and Health Management*, 7(4).

Saini, N., Yadav, A. L., and Rahman, A. (2024). Cloud based predictive maintenance system. In *ICRITO*, pages 1–5. IEEE.

Sanchez-Iborra, R., Zoubir, A., Hamdouchi, A., Idri, A., and Skarmeta, A. (2023). Intelligent and efficient iot through the cooperation of tinyml and edge computing. *Informatica*, 34(1):147–168.

Sepe, M., Graziano, A., Badora, M., Di Stazio, A., Bellani, L., Compare, M., Zio, E., et al. (2021). A physics-informed machine learning framework for predictive maintenance applied to turbomachinery assets. *J. of the Global Power and Propulsion Society*, 2021.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646.

Warren, J. and Marz, N. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Simon and Schuster.