UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RAFAEL TREVISAN

Aquisição de Dados para Digital Twins Utilizando um Cluster MQTT Escalado Dinamicamente

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Juliano Wickboldt

Porto Alegre
2025

“Qualquer tecnologia suficientemente avançada é indistinguível da magia”

— ARTHUR C. CLARKE

AGRADECIMENTOS

A realização deste trabalho de conclusão de curso representa o resultado de uma trajetória repleta de aprendizados, desafios e conquistas, que não seria possível sem o apoio de muitas pessoas. Primeiramente, agradeço à minha família, meus pais Rubesval e Rosângela, sempre exemplos de superação perseverança e à minha irmã Ana Paula, um exemplo e inspiração. Agradeço também ao Matias, que me apresentou ao meu orientador e ajudou a me iniciar no mundo da pesquisa.

Ao meu orientador Juliano, pela paciência, dedicação e pelas lições que foram fundamentais para o desenvolvimento deste trabalho, e também aos meus outros professores, que durante a graduação compartilharam seus conhecimentos e inspiraram a minha busca por aprendizado e crescimento.

Aos meus amigos Haroldo, Castro, Murilo e Emi, pessoas que levo comigo desde os dias do colégio e que sei que posso contar, e meus colegas de curso Tom, Ricco, Ernesto e Zanutto, pela troca de ideias, amizade e apoio que tivemos ao durante esses longos anos na graduação.

Agradeço aos meus colegas do Projeto PETWIN, em especial ao professor Marcos, sempre solícito para reiniciar os computadores que eu graciosamente derrubei com meus experimentos.

Por fim, agradeço a todos que, de alguma forma, contribuíram para a realização deste trabalho. A cada um de vocês, o meu sincero muito obrigado.

RESUMO

Digital Twins são uma tecnologia que é capaz de monitorar algum ativo em tempo real e realizar análises sobre seu estado atual e previsões sobre seu estado futuro. Para garantir que as análises e previsões são certeiras, é preciso garantir a saúde e segurança da aquisição de dados da parte real para a virtual. Um protocolo apontado para implementar a comunicação de dados nesse contexto é o MQTT. Este protocolo utiliza um servidor centralizado para o envio de mensagens, chamado *broker* MQTT, porém como qualquer elemento centralizador em uma arquitetura, quando sob estress ele pode se tornar um gargalo. Este trabalho avaliou o uso dessa tecnologia nesse contexto e propõe um novo elemento para aumentar dinamicamente o número de nodos em um *cluster* de *brokers* MQTT. A proposta foi avaliada por meio de experimentos que analisaram métricas de desempenho, como latência, uso de CPU, consumo de memória e de rede, em diferentes configurações de escalabilidade. Os resultados demonstram que a abordagem proposta apresenta uma melhora na qualidade da comunicação. Sem escalas apenas 51,8% das mensagens críticas atenderam seus requisitos de tempo quando utilizando uma escala baseada em mensagens *inflight* se obteve um total de 90,9%. Nestes experimentos também foi determinado que a escala de novos nodos durante momentos de tráfego intenso prejudicam a qualidade da aquisição de dados atendendo os requisitos de apenas 48,8% das mensagens.

Palavras-chave: Digital Twins. MQTT. Escalabilidade. Aquisição de Dados.

Data Acquisition for Digital Twins Using a Dynamically Scaled MQTT Cluster

ABSTRACT

Digital Twin is a technology capable of monitoring an asset in real-time and performing analyses of its current state as well as predictions about its future state. To ensure that these analyses and predictions are accurate, it is essential to maintain the reliability and security of data acquisition from the physical part to the virtual counterpart. A protocol often suggested for implementing data communication in this context is MQTT. This protocol relies on a centralized server for message exchange, known as the MQTT broker. However, like any centralizing element in an architecture, under stress, it can become a bottleneck. This work evaluated the use of this technology in such a context and proposes a new element to dynamically increase the number of nodes in an MQTT broker cluster. The proposal was evaluated through experiments analyzing performance metrics, such as latency, CPU usage, and memory consumption, under different scalability configurations. The results demonstrate that the proposed approach improves communication quality. Without scaling, only 51.8% of critical messages met their time requirements, while using scaling based on inflight messages achieved a total of 90.9%. These experiments also determined that scaling new nodes during periods of heavy traffic negatively impacts data acquisition quality, meeting the requirements of only 48.8% of the messages.

Keywords: Digital Twins, MQTT, Scalability, Data Acquisition .

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
Bps	*Bytes* por segundo
CPU	*Central Processing Unit*
IoT	*Internet of Things*
IIoT	*Industrial Internet of Things*
DT	*Digital Twin*
IP	*Internet Protocol*
kbps	Quilo *bits* por segundo
Mbps	Mega *bits* por segundo
MQTT	*Message Queuing Telemetry Transport*
ms	Milissegundos
ns	Nanossegundos
QoS	Qualidade de serviço
s	Segundos
UDP	*User Datagram Protocol*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Digital twins, ou gêmeos digitais, são representações virtuais de objetos, processos ou sistemas físicos que permitem monitorar, simular e otimizar seu desempenho em tempo real. Essa tecnologia conecta o mundo físico ao digital por meio de sensores, dados em tempo real e modelos analíticos, criando uma réplica digital dinâmica que reflete o comportamento do objeto ou sistema no mundo real (BATTY, 2018).

O conceito de *digital twin* foi idealizado inicialmente por Michael Grieves em 2002, durante uma apresentação sobre gestão do ciclo de vida do produto (PLM - Product Lifecycle Management) na Universidade de Michigan. Segundo Grieves, "um gêmeo digital é composto por três partes principais: o produto físico no espaço real, a contraparte virtual no espaço digital e as conexões entre os dois, que transmitem dados em ambas as direções" (GRIEVES, 2014). Apesar disso, a aplicação prática do conceito só começou a ganhar força anos depois, com os avanços em computação em núvem, Internet das Coisas (IoT) e inteligência artificial.

Os *digital twin* têm diversas aplicações, destacando-se em áreas como manufatura, saúde e cidades inteligentes. Eles são usados para monitoramento em tempo real, simulação, previsão e análise, e otimização (TAO et al., 2018). Monitoramento em tempo real facilita a identificação de anomalias, prevenção de falhas e simplifica a manutenção de equipamentos, permitindo que organizações tomem decisões baseadas nestes dados (FULLER et al., 2020). Com simulações, é possível testar alterações ou otimizações em sistemas sem interromper as operações reais, tornando *digital twins* valiosos para prever comportamentos e testar cenários hipotéticos (GLAESSGEN; STARGEL, 2012). Com o constante monitoramento é possível realizar previsões e análises para antecipar o desempenho futuro de ativos e prever possíveis cenários com base em dados históricos e padrões atuais (TAO et al., 2018). Através de todas estas funcionalidades é possível otimizar os processos, reduzir custos e realizar manutenções preventivas para garantir o desempenho dos ativos monitorados.

A eficácia das aplicações de *digital twins* está ligada à eficiência na aquisição de dados do mundo físico. Os sensores e dispositivos IoT (Internet of Things) desempenham um papel crucial nesse processo, capturando informações em tempo real sobre variáveis como temperatura, pressão, vibração, localização e desempenho de máquinas ou sistemas. Esses dados precisam ser precisos, consistentes e coletados em alta frequência para garantir que o gêmeo digital reflita com fidelidade o comportamento do objeto ou sistema

físico.

A transmissão desses dados para a plataforma digital deve ser realizada com baixa latência, permitindo que o modelo virtual reaja rapidamente a alterações no mundo real. Ineficiências na coleta de dados, como lacunas, ruídos ou atrasos, podem comprometer a precisão das análises, simulações e previsões realizadas pelo gêmeo digital, reduzindo sua utilidade em aplicações críticas.

Como a coleta de dados deste contexto é fundamental, é preciso selecionar um protocolo de comunicação adequado para atender os requisitos de tempo real dos ativos monitorados. O protocolo MQTT (Machine Queueing Telemetry Transport) é um protocolo projetado para comunicação leve e confiável em redes com baixa largura de banda ou alta latência. O MQTT permite que dispositivos IoT capturem e transmitam dados de forma contínua e eficiente para o gêmeo digital. Ele segue um modelo de publicação/assinatura, em que sensores ou dispositivos publicam dados em tópicos específicos para um servidor centralizador denominado *broker*, e os sistemas digitais assinam esses tópicos para receber as informações em tempo real. Essa abordagem reduz a sobrecarga de comunicação e garante a entrega confiável de dados mesmo em condições de rede instáveis.

Para lidar com grandes volumes de dados, ou muitas conexões simultâneas, algumas implementações de broker MQTT, utilizam uma configuração chamada “modo *cluster*”. Esta configuração permite a interligação de múltiplos *broker* para formar um sistema distribuído, oferecendo maior escalabilidade, resiliência e desempenho. Em um *cluster*, os *broker* trabalham de forma colaborativa para compartilhar a carga de trabalho, distribuindo as conexões de clientes e as mensagens publicadas de maneira equilibrada. Isso reduz o risco de sobrecarga em um único *broker* e minimiza pontos únicos de falha. Apesar da simplicidade e eficiência deste protocolo, ele ainda apresenta limitações como, falta de segurança nativa (OASIS, 2014), ausência de gerenciamento de escalabilidade (RODRIGUES, 2018) e latência em cenários de alta demanda (ZHOU; FANG, 2019).

Desta forma, o presente trabalho tem como principal objetivo explorar as limitações que o MQTT apresenta no contexto de aquisição de dados para *digital twins* e estudar como superá-las. Para isso, este trabalho também propõe um novo elemento para a arquitetura, capaz de analisar a saúde de um *cluster* de *brokers* MQTT, e gerenciar a quantidade de nodos dinamicamente para garantir a saúde da comunicação.

O *Manager* é um componente implementado para verificar métricas de saíde do cluster como fila e mensagens *inflight*. Através do monitoramento de métricas que possuem impacto direto na latência de entrega de mensagens, o *Manager*, pode adicionar

mais nodos a este *cluster* para melhorar a qualidade da comunicação. Para mitigar problemas de segurança foi utilizado o *MQTT over QUIC* (Quick UDP Internet Connections), protocolo que possui TLS (Transport Layer Security), embutido em sua *stack*, porém não foram realizadas análises do uso deste protocolo neste trabalho.

O restante do presente trabalho foi dividido nos seguintes capítulos. O capítulo 2 descreve a fundamentos sobre *digital twins* e comunicação usando o protocolo MQTT. No capítulo 3, são descritos os estudos que se assemelham com o presente trabalho, além de demonstrar como este trabalho difere da literatura atual. O capítulo 4 demonstra a arquitetura desenvolvida para execução de experimentos de testes de desempenho, coleta e análise de dados. No capítulo 5, são discutidos os resultados dos experimentos, demonstrando o desempenho e as limitações encontradas. Por fim, no capítulo 6, são apresentadas com mais detalhes as contribuições teóricas e práticas, as limitações encontradas e sugestões de futuros trabalhos a partir destas descobertas.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma visão geral sobre aquisição de dados para *digital twins*, abordando os componentes e protocolos propostos para realizar esta tarefa. Também é descrito o protocolo de comunicação selecionado, o MQTT.

2.1 Aquisição de dados em Digital Twins

O Digital Twin é uma tecnologia que cria uma réplica virtual de objetos físicos, sistemas ou processos, utilizando dados em tempo real combinados com simulações e inteligência artificial. Por meio da coleta de informações de sensores e modelos matemáticos, o Digital Twin permite monitorar, prever, e otimizar o desempenho, garantindo maior eficiência operacional, redução de custos, e tomada de decisões mais precisas em diversas indústrias, como manufatura, saúde, energia, e infraestrutura. Essa solução digital oferece a possibilidade de realizar análises preditivas, testes virtuais e melhorias contínuas antes de realizar intervenções no mundo físico, transformando a maneira como os ativos e processos são gerenciados ao longo do seu ciclo de vida.

Revisar trabalhos anteriores é importante para compreender os desafios e avanços na aquisição de dados para *digital twins*, uma vez que essa etapa é crucial para garantir o funcionamento eficiente e em tempo real desses sistemas. Estudos prévios oferecem uma visão abrangente sobre os métodos, protocolos e tecnologias empregadas, permitindo identificar as melhores práticas e possíveis limitações. Além disso, a análise de abordagens já desenvolvidas contribui para entender como diferentes indústrias, como manufatura inteligente, engenharia elétrica e casas conectadas, implementaram soluções específicas para coleta e transmissão de dados. A tabela 2.1, apresenta soluções selecionadas, ressaltando a indústria onde foi utilizada, objetivo do sistema implementado e método de transporte de dados.

	Indústria	**Objetivo**	**Transporte de Dados**
(CORONADO et al., 2018)	Manufatura Inteligente	Sistemas de Execução de Manufatura	HTTP e XML
(HU et al., 2018)	Manufatura Inteligente	Conectar múltiplas fábricas	HTTP e XML
(LI et al., 2020)	Engenharia Elétrica	Construir um Digital Twin para sistemas de bateria	MQTT
(BELLAVISTA et al., 2021)	Manufatura Inteligente	Comunicação escalável	MQTT ou CoAP
(CATHEY et al., 2021)	Veículos, Manufatura Inteligente	Segurança de dados	HTTP, MQTT, DDS ou CoAP
(FUKUSHIMA et al., 2021)	Manufatura Inteligente	Melhorar a segurança em torno de braços robóticos automatizados	MQTT
(ZDANKIN et al., 2022)	Casas Inteligentes	Longevidade de software em casas inteligentes	MQTT e HTTP
(RODRIGO et al., 2023)	Engenharia de Redes	Avaliar um ambiente de rede	MQTT

Tabela 2.1 – Propostas selecionadas para implementações de Digital Twin.

Essa revisão é particularmente útil para avaliar a adequação de tecnologias de

aquisição de dados ao longo do tempo. bem como para explorar estratégias de segurança, escalabilidade e otimização do fluxo de informações.

Na literatura anterior, o MTConnect foi amplamente reconhecido como um padrão popular para aquisição de dados de máquinas-ferramenta na área de manufatura inteligente. Conrado et al. utilizaram esse protocolo para implementar um sistema de execução de manufatura capaz de registrar entradas de materiais, consumo de insumos e fluxo de produtos, fornecendo dados a longo prazo sobre eficiência e outras tendências (CORONADO et al., 2018). No mesmo ano, Hu et al. também adotaram esse padrão para construir um gêmeo digital baseado em nuvem, destinado a prever e estimar a demanda de recursos (HU et al., 2018).

Embora tenha sido uma solução popular no passado, o protocolo MTConnect utiliza HTTP (Hypertext Transfer Protocol) para transmitir dados em formato XML (Extensible Markup Language)coletados de sensores (AMT, 2023), o que não é um modelo otimizado para comunicações confiáveis e em larga escala entre gêmeos digitais, que demandam alta conectividade entre dispositivos. Desde então, protocolos mais adequados para IoT e Indústria 4.0 têm sido considerados opções superiores para a implementação de gêmeos digitais (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019).

Propostas mais recentes sugerem ou mencionam o uso do MQTT como um método promissor para aquisição de dados em gêmeos digitais no campo da manufatura inteligente. Esse protocolo foi utilizado por Bellavista et al. e Fukushima et al. para gerenciar a comunicação entre dispositivos físicos e seus correspondentes digitais, oferecendo escalabilidade na comunicação (BELLAVISTA et al., 2021; FUKUSHIMA et al., 2021). Cathey et al. também destacam o MQTT como um candidato para aquisição de dados, implementando múltiplos gêmeos digitais para um mesmo objeto com o objetivo de aumentar a segurança de dados por meio de tolerância a falhas (CATHEY et al., 2021).

Outros autores apresentam diferentes casos de uso de gêmeos digitais utilizando o MQTT como protocolo de aquisição de dados. Liu et al. demonstram um protótipo de gêmeo digital de um sistema de baterias (LI et al., 2020), que possui diversas aplicações na engenharia elétrica. Zdankin et al. apresentam em seu trabalho o gêmeo digital de casas inteligentes, com foco adicional na longevidade dos componentes (ZDANKIN et al., 2022). Rodrigo et al. implementam o gêmeo digital de uma rede de computadores para oferecer melhor serviço aos usuários, além de visualização em tempo real (RODRIGO et al., 2023).

Apesar do uso do MQTT ser citado nesses trabalhos, os autores não exploram

detalhadamente aspectos como configuração, desempenho e implementação do protocolo. Esses detalhes são cruciais para garantir a escalabilidade e a performance das soluções propostas.

2.2 Protocolo MQTT

As seguintes seções abordam o protocolo MQTT. A primeira delas se aprofunda no funcionamento e características base do protocolo, quando a segunda seção se aprofunda sobre configurações relevantes para a implementação do trabalho.

2.2.1 Funcionamento

O MQTT é um protocolo de comunicação leve e baseado em mensagens, projetado para dispositivos que operam em redes com baixa largura de banda, alta latência ou conectividade intermitente. Criado por Andy Stanford-Clark e Arlen Nipper em 1999, o MQTT é amplamente utilizado em aplicações de IoT devido à sua simplicidade, eficiência e baixa sobrecarga de comunicação (STANFORD-CLARK; NIPPER, 1999).

A arquitetura do MQTT segue um modelo assíncrono e baseado em publicação/assinatura. Esse modelo possui três componentes principais: os publicadores, o broker MQTT e os assinantes. Os publicadores servem como a fonte de dados, enviando mensagens com um tópico para o broker, e os assinantes sinalizam para o broker quais são seus tópicos de interesse. O broker MQTT tem a função de receber as mensagens dos publicadores e encaminhá-las para os assinantes que estão inscritos naquele tópico (HUNKELER; TRUONG; STANFORD-CLARK, 2008), conforme a Figura 2.1.

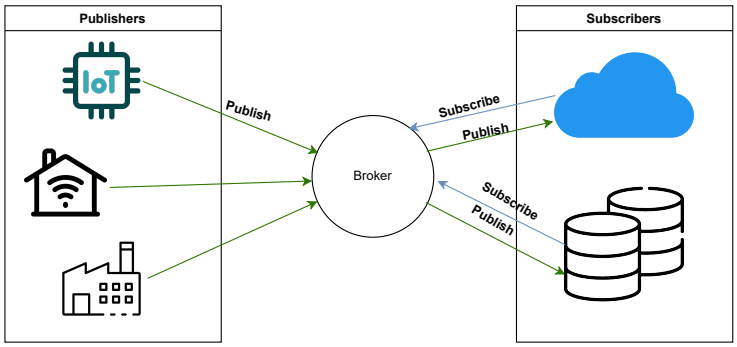

Figura 2.1 – Arquitetura do MQTT

No MQTT, mensagens na fila e mensagens *inflight* são mecanismos relacionados

ao gerenciamento de entrega de mensagens. As mensagens *inflight* são aquelas enviadas, mas ainda não confirmadas pelo destinatário, aplicando-se apenas a QoS 1 e QoS 2. Essas mensagens permanecem no estado *inflight* até que o *broker* ou cliente receba a confirmação de entrega, sendo retransmitidas em caso de falhas (BANKS; GUPTA, 2015). Quando a janela de *inflight* de um cliente está cheia, as mensagens são armazenadas pelo *broker* para garantir que essas mensagens sejam entregues assim que as mensagens *inflight* sejam confirmadas (LIGHT, 2017). Enquanto as mensagens na fila são voltadas para garantir entrega futura em cenários de janela de *inflight* cheia, as *inflight* focam na confiabilidade da entrega durante a conexão, assegurando a integridade e a ordem das mensagens conforme o nível de QoS configurado.

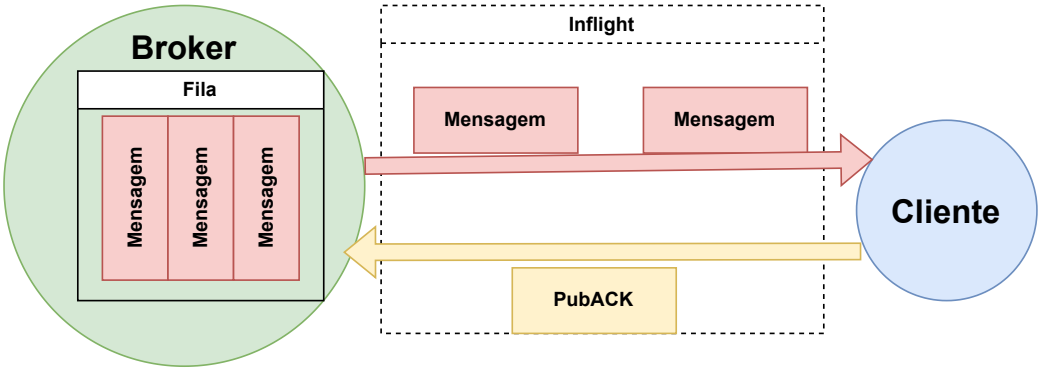

Figura 2.2 – Estrutura de Fila do MQTT

2.2.2 Configurações

O MQTT oferece uma ampla gama de recursos para atender às necessidades de aplicações modernas, incluindo os níveis de QoS, que garantem diferentes níveis de confiabilidade na entrega de mensagens, prioridades de mensagens de acordo com tópicos, além de um modo *cluster* que expande as capacidades de envio de mensagens através do uso de múltiplos *brokers* em paralelo, como demonstrado pela figura 2.3.

Os níveis de QoS no protocolo MQTT apresentam diferenças significativas em termos de confiabilidade, duplicação de mensagens, sobrecarga e usos típicos, conforme resumido na Tabela 2.2. O nível QoS 0 garante entrega no máximo uma vez, sem confirmação de recebimento, sendo ideal para dados descartáveis, como leituras de sensores frequentes, devido à sua baixa sobrecarga. Já o QoS 1 assegura entrega pelo menos uma vez, com possibilidade de duplicação de mensagens, equilibrando confiabilidade e desempenho, o que o torna apropriado para notificações ou sensores com dados mais crí-

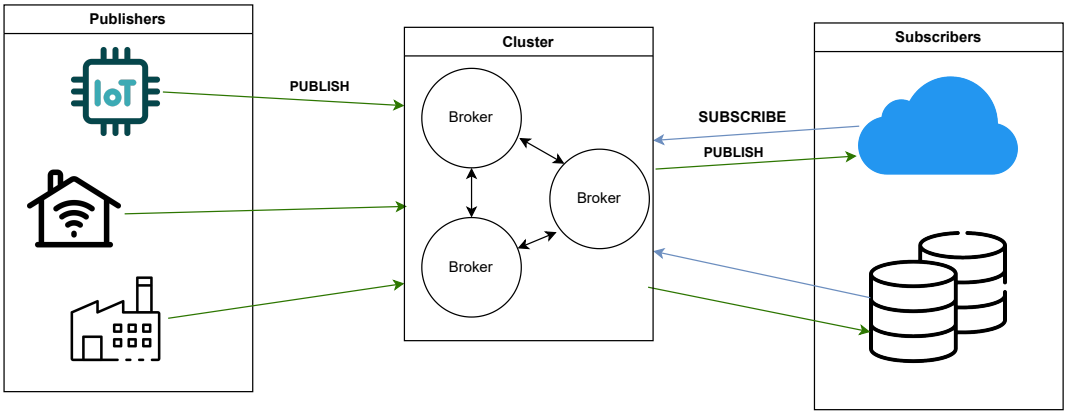

Figura 2.3 – Cluster MQTT com 3 nodos

ticos. Por fim, o QoS 2 proporciona a garantia máxima de entrega exatamente uma vez, eliminando duplicações, mas com um custo de maior complexidade e *overhead*, sendo utilizado em aplicações críticas, como transações financeiras (BANKS; GUPTA, 2015).

Nível de QoS	Garantia de Entrega	Duplicação Possível	Overhead	Uso Típico
QoS 0	No máximo uma vez	Não	Baixo	Dados descartáveis, leituras frequentes
QoS 1	Pelo menos uma vez	Sim	Médio	Notificações, comandos
QoS 2	Exatamente uma vez	Não	Alto	Aplicações críticas, transações

Tabela 2.2 – Comparação dos níveis de QoS no MQTT.

Implementações mais avançadas de broker MQTT possuem a função de priorização de mensagens, que pode ser configurada para atender a cenários onde determinados dados mais relevantes têm precedência para serem processados ou entregues. Embora o MQTT padrão não inclua suporte nativo para priorização de mensagens, o EMQX oferece recursos que permitem implementar esse comportamento. O EMQX permite atribuir um valor numérico, de 0 a 255, a um tópico, e dar preferência para o encaminhamento de mensagens em um tópico com este valor mais alto (EMQ TECHNOLOGIES CO., LTD., 2023).

O uso do MQTT sobre o protocolo QUIC (MQTT over QUIC) é uma abordagem emergente que visa superar limitações do transporte baseado em TCP, amplamente utilizado no MQTT tradicional. O QUIC, originalmente desenvolvido pelo Google, é um protocolo de transporte projetado para melhorar a latência, segurança e confiabilidade das conexões na internet (IYENGAR; THOMSON, 2021). Ao integrar o MQTT com o QUIC, é possível reduzir atrasos em cenários de redes com alta latência ou perda de pacotes, além de eliminar problemas como o "head-of-line blocking" presente no TCP (MOURADIAN; MEDHI, 2018). Essa integração tem potencial para aprimorar significativamente o desempenho do MQTT em aplicações críticas de IoT, como monitoramento em tempo real e dispositivos móveis, onde conexões rápidas e estáveis são essenciais para

a eficiência e a continuidade dos serviços.

O modelo de *cluster* MQTT pode ser estruturado para garantir escalabilidade, alta disponibilidade e eficiência na comunicação entre nodos. Ele é composto por brokers distribuídos, que compartilham assinaturas e persistem mensagens através de armazenamento distribuído. O modo cluster é uma solução introduzida em algumas distribuições de broker mais modernas, como EMQX e HiveMQ (EMQ TECHNOLOGIES CO., LTD., 2023; HIVEMQ GMBH, 2023). Com esta configuração, é possível conectar múltiplos brokers para que eles trabalhem juntos, formando uma rede interconectada. Essa configuração também proporciona maior tolerância a falhas, uma vez que toda a comunicação não está mais totalmente centralizada em apenas um broker.

3 TRABALHOS RELACIONADOS

Neste capítulo, são analisados alguns estudos publicados que apresentam soluções para escalabilidade do MQTT.

Spohn apresenta os desafios atuais de escalabilidade do MQTT e explora abordagens para mitigá-los. Em seu trabalho, são citadas duas estratégias de escalonamento: o escalonamento vertical, onde mais recursos são adicionados à máquina que hospeda o *broker* MQTT para melhorar seu desempenho, e o escalonamento horizontal, onde *broker*adicionais são implantados em diferentes máquinas utilizando um balanceador de carga para distribuir o tráfego de mensagens entre os clientes. Entre essas abordagens, o autor destaca o escalonamento horizontal de um *cluster* MQTT como a solução preferida, pois aumenta a tolerância a falhas e permite a aplicação de mecanismos existentes de controle de tráfego, como o QoS (SPOHN, 2022).

Longo et al. propõem uma nova abordagem para a descentralização do protocolo MQTT. A arquitetura centralizada padrão funciona bem quando o broker é hospedado na nuvem; no entanto, com o aumento previsto no uso de dispositivos IoT, a utilização de um *cluster* de *broker* MQTT na borda (edge) apresenta uma solução para conectar mais dispositivos simultaneamente. A maioria dos *broker* MQTT modernos já possui um modo de *cluster*, no qual múltiplos *broker* cooperam para entregar mensagens aos assinantes; contudo, essas implementações seguem uma abordagem de malha completa (full-mesh). Longo et al. propõem, em seu trabalho, um protocolo para criar uma hierarquia no modo de *cluster*, organizada por meio de uma árvore geradora (spanning tree) (LONGO et al., 2020).

Shahri et al. apresentam uma abordagem baseada em SDN (Redes Definidas por Software) para o gerenciamento de recursos em um *cluster* MQTT (SHAHRI; PEDREIRAS; ALMEIDA, 2024). Em sua solução, técnicas de SDN são utilizadas para ajustar o roteamento entre diferentes *broker* MQTT a fim de reduzir a latência em serviços em tempo real. Essa abordagem utiliza OpenFlow para criar canais dedicados à comunicação em tempo real. Seguindo essa estratégia, mensagens sensíveis ao tempo podem ter suas rotas ajustadas dinamicamente para aquelas que oferecem melhor desempenho.

Esses trabalhos obtiveram resultados que superam a abordagem centralizada em alguns cenários; no entanto, o tamanho do *cluster* permanece estático durante os experimentos, o que impede sua adaptação a mudanças inesperadas e repentinas no padrão de dados enviados. O trabalho proposto tem, portanto, como objetivo utilizar uma aborda-

gem de escalonamento dinâmico, onde novos nodos são elencados a um *cluster*, de acordo com a necessidade do sistema.

4 SOLUÇÃO

Neste capítulo, será explicada a arquitetura apresentada na figura 4.1, utilizada para a realização dos experimentos utilizando o manager, destacando a infraestrutura subjacente e componentes.

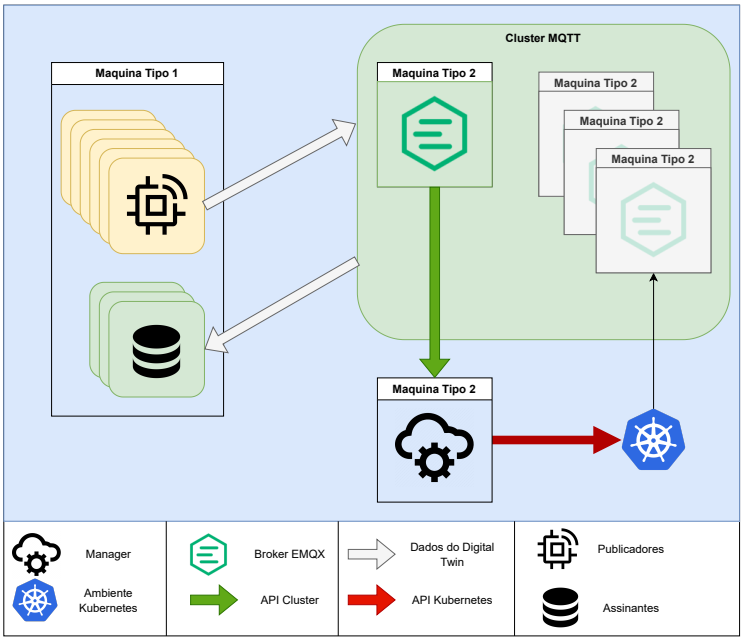

Figura 4.1 – Arquitetura usada para Experimentos.

As seguintes subseções especificam o comportamento dos componentes utilizados para a realização do experimento. Estes são: Clientes MQTT, *broker* MQTT e o *Manager* implementado.

4.1 Ambiente de Experimentação

Este trabalho foi desenvolvido em um *cluster* Kubernetes. O Kubernetes é uma plataforma open-source amplamente utilizada para orquestrar e gerenciar contêineres em ambientes de nuvem, sendo uma solução de referência para a implantação e manutenção de aplicações distribuídas. Ele automatiza tarefas complexas, como o escalonamento automático de recursos, a implantação contínua, o balanceamento de carga e a recuperação de falhas. O Kubernetes organiza os contêineres em unidades lógicas chamadas "pods", que são gerenciadas por um sistema de controle centralizado para garantir que as aplicações estejam sempre disponíveis e em funcionamento. Esta plataforma oferece van-

tagens para o gerenciamento de aplicações em ambientes distribuídos por automatizar a implantação, escalonamento e manutenção de “pods”. Uma de suas principais vantagens é a capacidade de redistribuir cargas de trabalho em caso de falhas nos nós e simplificar o gerenciamento de atualizações e rollbacks com estratégias como“rolling updates”. Ele também facilita a portabilidade entre diferentes ambientes (on-premises e nuvem) e melhora a segurança ao isolar contêineres em “pods”, tornando-o uma escolha robusta para sistemas modernos que demandam flexibilidade, resiliência e escalabilidade. Neste ambiente também foi implantado o Prometheus, uma aplicação para monitoramento de containeres e pods.

O Prometheus é projetado especificamente para ambientes modernos, como os baseados em contêineres e microserviços. Ele coleta métricas numéricas em intervalos regulares, armazenando-as em um banco de dados de séries temporais altamente eficiente. O Prometheus opera através de “exporters”, que são componentes que coletam métricas de sistemas ou aplicativos, como servidores, bancos de dados e serviços em contêineres, e as disponibilizam para coleta. Com ele, é possível monitorar o desempenho de CPU, memória, latência, taxas de erro, entre outras métricas críticas. Além disso, sua integração com o Grafana permite criar dashboards ricos e interativos para visualização dos dados em tempo real.

Neste ambiente, foram utilizados *StatefulSets*, para descrever clientes MQTT e o *Broker* que será selecionado. Este recurso do Kubernetes é projetado para gerenciar aplicações que mantêm estado, como brokers MQTT, bancos de dados e aplicações sensíveis à ordem de inicialização ou ao armazenamento persistente. Diferentemente de *Deployments*, que gerenciam réplicas de maneira indiferente ao estado de cada pod, *StatefulSets* garantem que cada pod tenha uma identidade única, um nome DNS (Domain Name Server) estável e acesso persistente ao armazenamento, mesmo após reinicializações. No contexto do cluster MQTT, *StatefulSets* foram usados para gerenciar os brokers MQTT, garantindo alta disponibilidade e escalabilidade. Cada broker recebe um nome fixo baseado em sua réplica, como *emqx-0*, *emqx-1*, e assim por diante, além de volumes persistentes configurados automaticamente pelo Kubernetes.

4.2 Clientes MQTT

Para gerar o trafego de mensagens, foram implementados clientes MQTT, utilizando a funcionalidade de *MQTT over QUIC*, tanto publicadores quanto assinantes. Es-

ses clientes foram implementados em python (código disponível em <https://github.com/raTrevisan/quic-mqtt>) e são configuráveis através de variáveis de ambiente para enviar mensagens em diferentes intervalos e com diferentes tamanhos máximos e mínimos de mensagem, conforme o trecho 4.1. Também foram implementados assinantes para receber os dados gerados, permitindo avaliar o desempenho do sistema em cenários variados.

```
spec:
   containers:
     - name: quic-mqtt-pub
       image: rafatrevisan/mqtt-quic-pub
       env:
         - name: POD_NAME
           valueFrom:
             fieldRef:
               apiVersion: v1
               fieldPath: metadata.name
         - name: MQTT_CLUSTER_IP
           value: emqx-headless.dtwins
         - name: MQTT_CLUSTER_PORT
           value: '14567'
         - name: MQTT_TOPIC
           value: type-1
         - name: MQTT_QOS
           value: '1'
         - name: MQTT_MESSAGE_NUM
           value: '100000'
         - name: MQTT_MIN_MESSAGE_SIZE
           value: '50'
         - name: MQTT_MAX_MESSAGE_SIZE
           value: '200'
         - name: MQTT_MESSAGE_FREQ_MS
           value: '10'
```

Listing 4.1 – Exemplo de Configuração de Cliente

O MQTT sobre QUIC apresenta vantagens significativas em relação ao MQTT sobre TCP, especialmente em termos de desempenho, segurança e resiliência em redes instáveis. O QUIC permite o estabelecimento de conexões mais rápidas, integrando a

configuração de transporte e criptografia em um único passo, o que reduz a latência inicial -(AMT, 2023)). Ele também elimina o problema de bloqueio em cadeia (*head-of-line blocking*) graças à multiplexação de fluxos independentes, permitindo uma transmissão eficiente de mensagens em diferentes tópicos. Além disso, o QUIC oferece melhor tolerância a perdas de pacotes e suporta migração de conexão, garantindo estabilidade mesmo em redes móveis ou com alterações frequentes de IP . Com controle de congestionamento aprimorado e segurança integrada ao nível do transporte, o MQTT sobre QUIC é ideal para aplicações IoT que demandam alta performance, confiabilidade e segurança (FERNáNDEZ et al., 2020), requisitos importantes no contexto de *digital twins*.

O QUIC promove segurança ao incorporar criptografia obrigatória e autenticação diretamente no protocolo, eliminando a necessidade de camadas externas como TLS (Transport Layer Security) no TCP. Baseado em TLS 1.3, o QUIC protege todas as conexões com criptografia ponta a ponta, garantindo confidencialidade, integridade e autenticação. Seus cabeçalhos são criptografados para evitar interceptações ou manipulações, enquanto tokens de origem e controles de fluxo mitigam ataques DDoS (Distributed Disruption of Service) e garantem que recursos sejam alocados apenas para clientes legítimos. Além disso, ao combinar *handshakes* de transporte e segurança em um único passo, o QUIC reduz latência inicial sem comprometer a robustez contra ataques, tornando-o eficiente e seguro (IYENGAR; THOMSON, 2021). Este protocolo foi utilizado com o objetivo de atender requisitos de segurança porém não foram realizadas análises sobre o impacto deste elemento da arquitetura.

Os assinantes se conectam diretamente ao *brokers* MQTT e ao receber mensagens são capazes de registrar métricas como latência, taxa de entrega e possíveis perdas de mensagens. Essas configurações permitem testar escalabilidade do sistema.

4.3 *Broker* MQTT

O *brokers* MQTT é um componente central na arquitetura do protocolo MQTT, desempenhando o papel de mediador entre os clientes publicadores e assinantes. Sua principal função é receber mensagens de publicadores, organizá-las conforme o tópico associado e entregá-las aos assinantes que demonstraram interesse nesses tópicos. O *brokers* garante um modelo de comunicação desacoplada, onde os clientes não precisam conhecer diretamente a existência ou a localização uns dos outros.

Além do encaminhamento de mensagens, o *brokers* também gerencia aspectos

como a aplicação dos níveis de Qualidade de Serviço, autenticação e autorização de clientes, retenção de mensagens e monitoramento do estado de conexão de cada cliente. Em configurações mais avançadas, ele pode ser configurado para operar em modo *cluster*, permitindo maior escalabilidade, tolerância a falhas e suporte a um maior número de dispositivos simultaneamente.

O desempenho e a eficiência do *brokers* são cruciais para a estabilidade e a confiabilidade de aplicações IoT, especialmente em cenários que exigem baixa latência, alta disponibilidade e gerenciamento eficiente de grandes volumes de dados.

Para realizar os experimentos foi selecionado o EMQX, devido sua performance comparado com outras implementações (KOZIOLEK; GRÜNER; RÜCKERT, 2020). O EMQX é um *brokers* MQTT de alto desempenho reconhecido por sua capacidade de lidar com grandes volumes de mensagens e conexões simultâneas em aplicações IoT. Desenvolvido pela EMQ Technologies, o EMQX oferece suporte à especificação MQTT 5.0 e é projetado para ser escalável, robusto e flexível. Esta implementação oferece suporte a extensões como persistência de mensagens, priorização de tráfego por tópico e compatibilidade com protocolos de transporte alternativos, como MQTT sobre QUIC, características que foram exploradas para a realização dos experimentos.

Algumas das configurações padrões deste componente foram alteradas. Primeiramente foi removido o tamanho máximo da fila, as configurações que encerravam os processos do *broker* por consumo elevado de recursos e foram habilitadas e configuradas prioridades baseadas em tópico, conforme demonstrado no trecho do YAML 4.2.

```
data:
  EMQX_API_KEY:'{bootstrap_file="/opt/emqx/etc/bootstrap.conf"}'
  EMQX_CLUSTER__DISCOVERY_STRATEGY: dns
  EMQX_CLUSTER__DNS__NAME: emqx-headless.dtwins.svc.cluster.
     local
  EMQX_FORCE_SHUTDOWN: '{"enable":false}'
  EMQX_LISTENERS__QUIC__DEFAULT__CERTFILE: etc/certs/cert.pem
  EMQX_LISTENERS__QUIC__DEFAULT__ENABLED: 'true'
  EMQX_LISTENERS__QUIC__DEFAULT__KEYFILE: etc/certs/key.pem
  EMQX_MQTT__MAX_INFLIGHT: '32'
  EMQX_MQTT__MAX_MQUEUE_LEN: infinity
  EMQX_MQTT__MQUEUE_DEFAULT_PRIORITY: lowest
  EMQX_MQTT__MQUEUE_PRIORITIES: '{"type-3/+":5,"type-2/+":2,"
     type-1/+":1, "type-4/+":4}'
```

```
13   EMQX_NAME: emqx
```

Listing 4.2 – Configurações do broker EMQX

4.4 Manager

Em trabalhos anteriores, foi estabelecido que a adição de nós a um *cluster* MQTT durante a operação pode ajudar a evitar a interrupção das comunicações e manter alta disponibilidade para mensagens críticas (TREVISAN et al., 2022). Este artigo expande o conceito de aumentar dinamicamente a capacidade de um *cluster* MQTT por meio de escalonamento horizontal. Este trabalho propõe um componente com as características principais de:

- Adicionar nodos ao *cluster* quando ele se encontra sobrecarregado;
- Não interferir na arquitetura padrão do MQTT;
- Não exigir reimplementação dos publicadores ou assinantes;

Este componente foi implementado em python, com seu código fonte disponível em <https://github.com/Open-Digital-Twin/python-manager>. O manager utiliza informações sobre mensagens *inflight* e mensagens na fila para tomar decisões de escala no *cluster* MQTT. As mensagens *inflight*, que representam dados em trânsito aguardando confirmação, são monitoradas para identificar se os *brokers* estão sobrecarregados em termos de entrega e retransmissão. Já as mensagens na fila, que incluem aquelas acumuladas no buffer dos *brokers* aguardando processamento ou envio, são usadas para avaliar a capacidade de cada nó de lidar com a carga atual. Quando o número de mensagens *inflight* ou na fila ultrapassa os limites predefinidos, o manager adiciona novos nós ao *cluster* para distribuir a carga de forma equilibrada e evitar atrasos ou perdas de dados. Para acessar esses dados, o manager utiliza a API do *brokers* MQTT, e para efetuar comandos de escala é utilizada a API do Kubernetes, conforme demonstra a Figura 4.2.

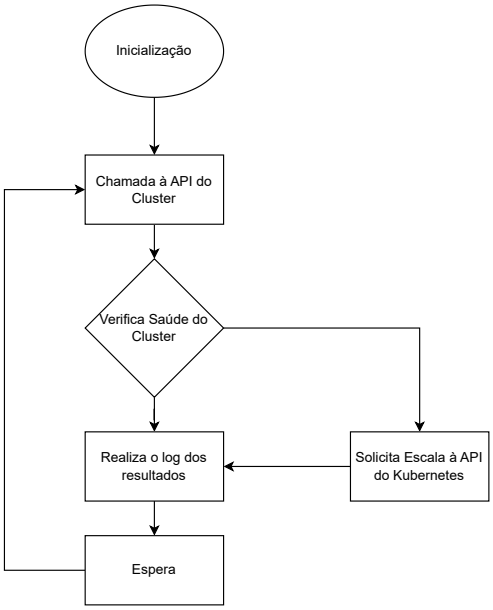

Figura 4.2 – Diagrama de Fluxo do Manager

O objetivo principal deste componente é proporcionar escalabilidade dinâmica ao *cluster* MQTT, garantindo que os recursos sejam utilizados de maneira eficiente e que a qualidade do serviço seja mantida mesmo em cenários de variação de carga. Para alcançar isso, o componente monitora continuamente métricas operacionais, com enfase no tamanho de fila e número de mensagens *inflight* nos *brokers* do *cluster*. Com base nesses dados, decisões automatizadas são tomadas para ajustar a topologia do *cluster* em tempo real.

5 RESULTADOS

Este capítulo introduz a infraestrutura para realização dos testes, quais foram os experimentos realizados e as métricas utilizadas. Também apresenta os resultados dos experimentos e as conclusões geradas a partir da análise dos dados coletados.

5.1 Experimentos

Nos experimentos realizados, foi avaliado o impacto no desempenho da aquisição de dados ao introduzir o Manager à arquitetura do MQTT, para escalar autonomamente os nós de um *cluster* de *brokers* MQTT em resposta às demandas de tráfego. O gerenciador monitora métricas em tempo real, como número de conexões ativas, taxa de mensagens por segundo e utilização de CPU e memória em cada *broker* do *cluster*. Quando os limites configurados são atingidos, o gerenciador adiciona dinamicamente novos nós ao *cluster* para distribuir a carga de forma equilibrada, garantindo a continuidade do serviço e mantendo baixos níveis de latência. Os experimentos foram conduzidos simulando cenários de tráfego variável, utilizando clientes MQTT configurados para enviar mensagens com diferentes tamanhos e QoS. Os resultados demonstraram que o gerenciador foi eficaz em evitar sobrecargas, mantendo a QoS mesmo em situações de alta demanda.

Os publicadores foram configurados para simular os diferentes padrões de carga de componentes de DTs, conforme a tabela 5.1, gerada com especificações industriaisQoS (HAMIDOVIC et al., 2023) (LI et al., 2024) .

Caso de Uso	**Latência Máxima**	**Frequência de Perdas**	**Tamanho Típico de Pacote**	**Prioridade**
Missão Criticas	50 ms	$< 10^{-4}$	64-128 bytes	Mais Alta
Realidade Aumentada	20 ms	$< 10^{-4}$	1000-1500 bytes	Alta
Sensores IIoT	200 ms	$< 10^{-3}$	50-200 bytes	Média
Automação de Processos	500 ms	$< 10^{-2}$	300-500 bytes	Baixa
Dados de Câmeras	100 ms	$< 10^{-2}$	1500 bytes	Baixa

Tabela 5.1 – Parâmetros de QoS

Com essas configurações, foram instanciados 30 clientes publicadores para cada categoria, que se conectaram ao *cluster* de forma gradual. Paralelamente, 3 clientes assinantes também estabeleceram suas conexões de maneira progressiva, aumentando gradualmente o tráfego no *cluster*. Estes experimentos totalizaram 90.090.000 mensagens, sendo elas distribuídas em:

- 3.000.000 Mensagens IIoT

- 60.000 Mensagens de Realidade Aumentada
- 30.000 Mensagens de Missão Crítica
- 3.000.000 Mensagens Automação de Processos
- 3.000.000 Mensagens de Dados de Camera

O *cluster* MQTT é iniciado com apenas um nodo e o manager é configurado para adicionar mais nodos conforme o *cluster* atinge limites de fila ou mensagens *inflight*, conforme a Figura 4.1. Os limites selecionados para fila foram 1000 e 5000, quando os limites selecionados para mensagens *inflight* foram de 16 e 32. Estes limites foram escolhidos com base nos valores padrões máximos para estes parâmetros definidos pelo *broker* EMQX, 32 para *inflight* e 5000 para a fila.

5.2 Infraestrutura

Nesta infraestrutura, foram utilizados dois tipos distintos de dispositivos: máquinas do tipo 1, equipadas com processadores Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz com 4 núcleos e 4 GB de memória RAM, e máquinas do tipo 2, que possuem processadores Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz com 4 núcleos e 32 GB de memória RAM.

5.3 Dados Coletados

Com os dados coletados pelos assinantes MQTT, foram realizadas análises específicas para cada classe de aplicação. Essas análises incluíram a avaliação do desempenho em termos de latência e perda de pacotes, considerando as características e requisitos individuais de cada classe, como missão crítica, realidade aumentada, sensores IIoT e monitoramento remoto. Foi examinada a eficiência do sistema em diferentes cenários de escala realizando uma repetição com cada configuração.

5.3.1 Performance

A análise de performance foi conduzida para avaliar o desempenho do sistema em termos de latência de entrega das mensagens para diferentes classes de aplicação.

Foram coletados os tempos de publicação, retransmissão e entrega para cada mensagem, permitindo a medição precisa dos atrasos introduzidos pela rede e pelo *cluster* de *brokers*. Essa análise é crucial para aplicações de missão crítica e de tempo real, onde a latência impacta diretamente a qualidade do serviço.

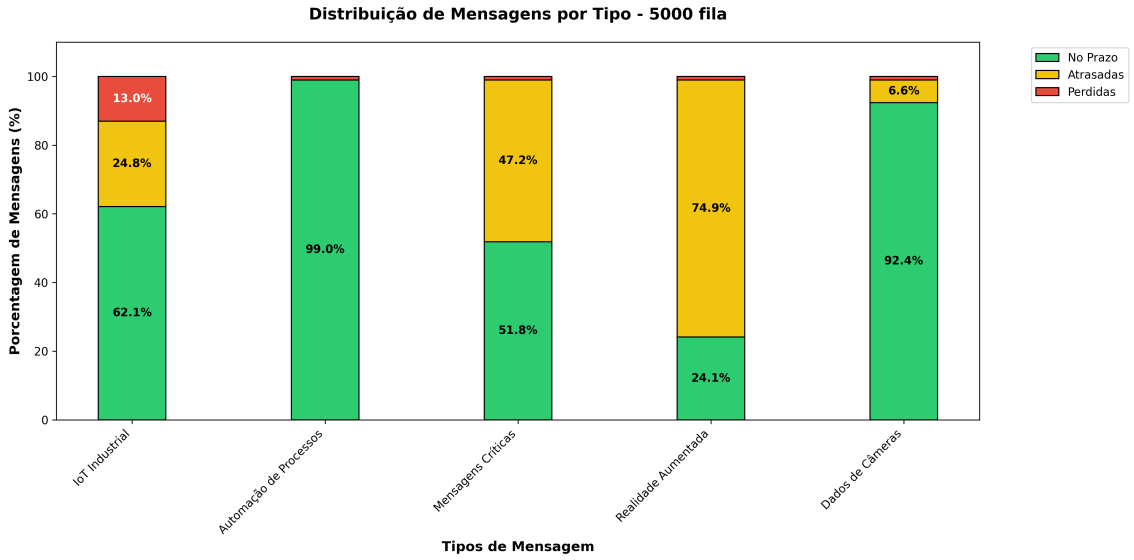

Figura 5.1 – Perda de mensagens escalando o *cluster* com a Fila em 5000.

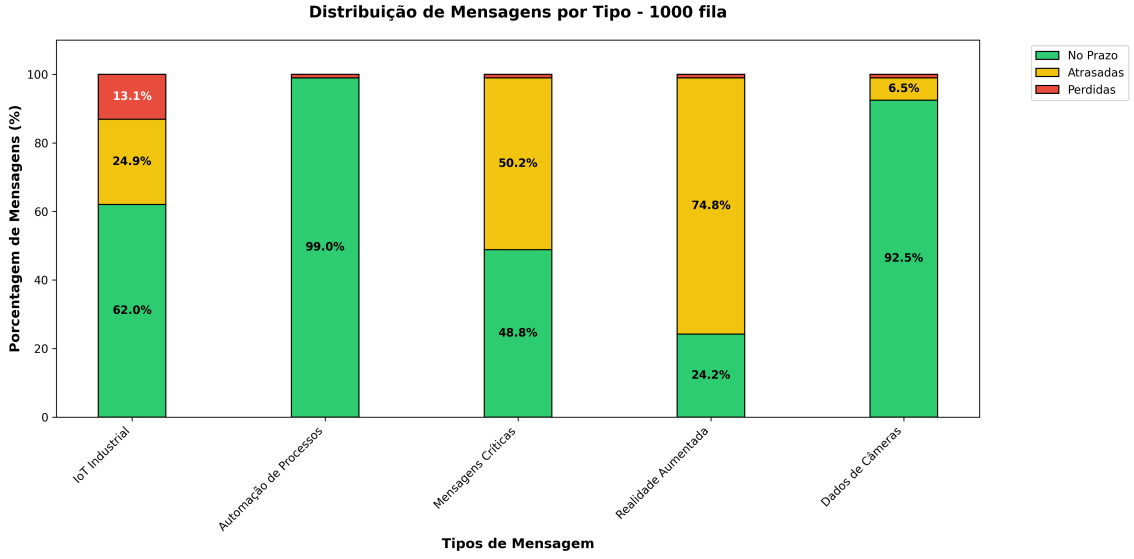

Figura 5.2 – Perda de mensagens escalando o *cluster* com a Fila em 1000.

Ao realizar experimentos com a configuração da fila em 1000 mensagens, foi gerado a Figura 5.2 e e utilizando a escala com 5000 mensagens, foi gerada a Figura 5.1. Com ambas estas configurações foi possível atender aos requisitos tanto para o envio de dados de câmeras, no entanto, as outras aplicações apresentaram níveis inaceitáveis de atraso ou perda de mensagens. Esses cenários demonstram que o ajuste da escala usando

a fila desempenha um papel garantindo a continuidade do serviço, porém não é capaz de atender as QoS de outros serviços.

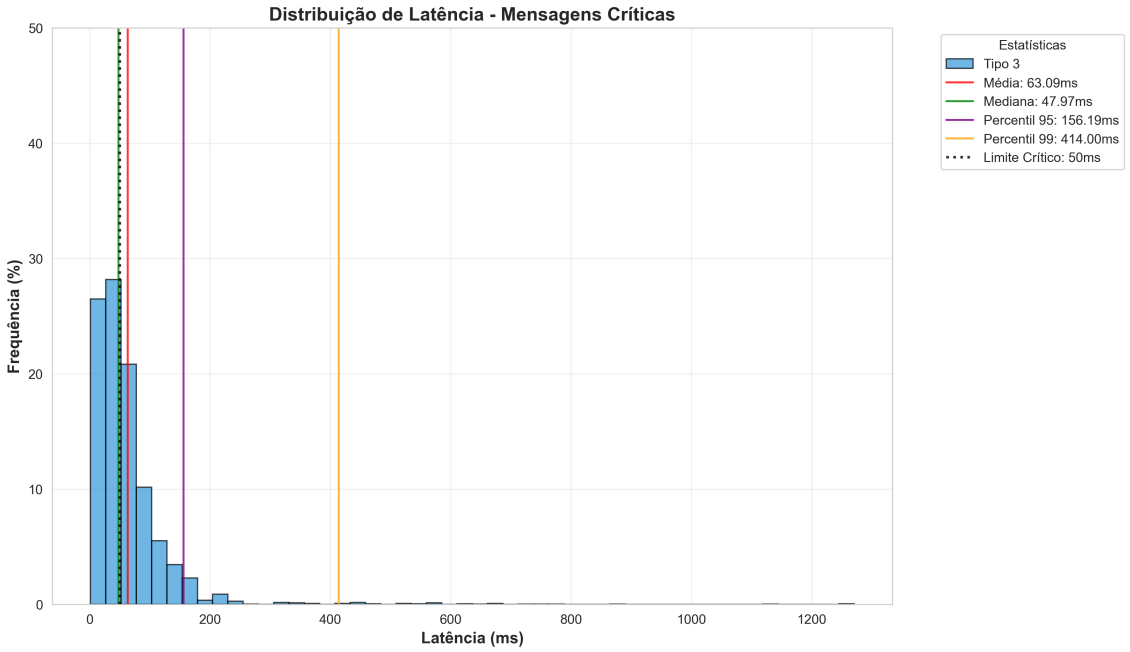

Figura 5.3 – Histograma de Mensagens Críticas com escala em 5000.

Analisando o histograma das latências da entrega de mensagens críticas, apresentado na figura 5.3, é possível observar uma distribuição com diversas latências acima de 50 ms, não atendendo aos requisitos de tempo real para aplicações de missão crítica. A média da latência destas mensagem foi de 64.09 ms, com 95% delas estando abaixo de 156.19 ms e 99% abaixo de 414.00 ms.

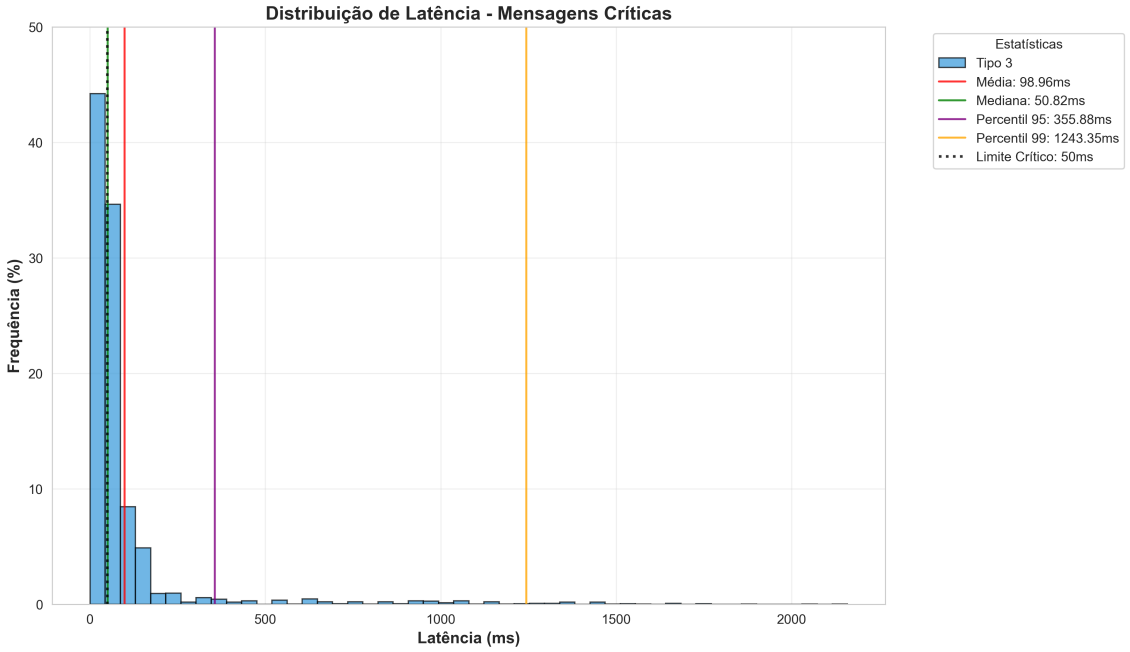

Figura 5.4 – Histograma de Mensagens Críticas com escala em 1000.

Ao configurar o Manager para escalar novos nodos com a fila em 1000 mensagens, obtemos os resultados apresentados na figura 5.4. A média da latência dessas mensagens foi de 98,06 ms, com 95% delas abaixo de 355.88 ms e 99% abaixo de 1243.35 ms. Apresentando pior eficiência para a latência desta aplicação.

A escala utilizando as mensagens *inflight* é outro parâmetro que foi utilizado para aumentar o número de nodos no *broker*. Quando as mensagens *inflight* aumentam além de um limite predefinido, indica que o sistema pode estar sobrecarregado e que o *broker* precisa de mais recursos para continuar entregando mensagens com alta eficiência. Experimentos escalando com este parâmetro em 16 mensagens e 32 mensagens foram realizados para gerar as Figuras 5.6 e 5.5 respectivamente.

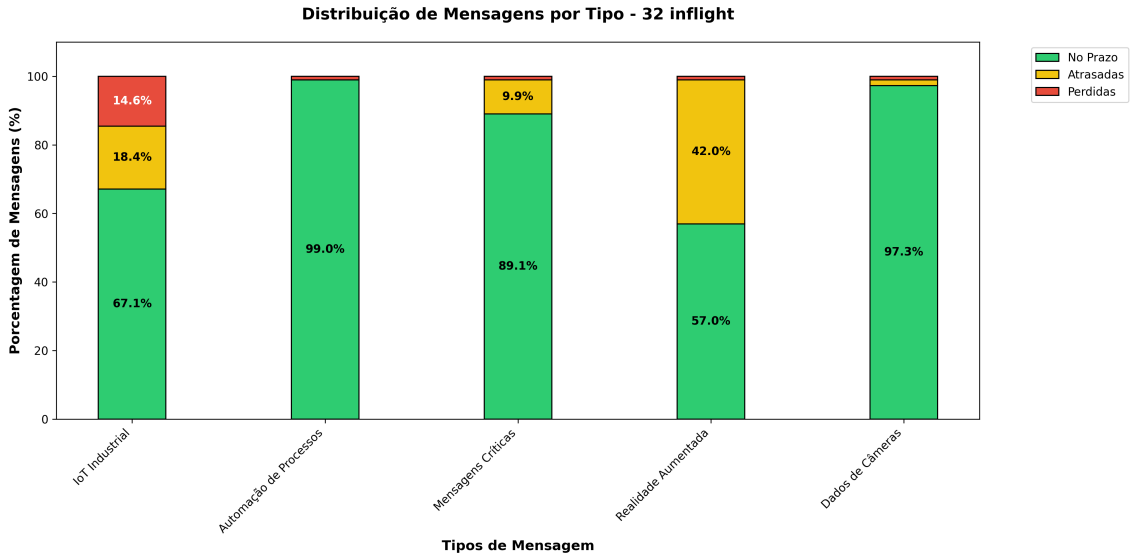

Figura 5.5 – Perda de mensagens escalando o *cluster* com o *inflight* em 32.

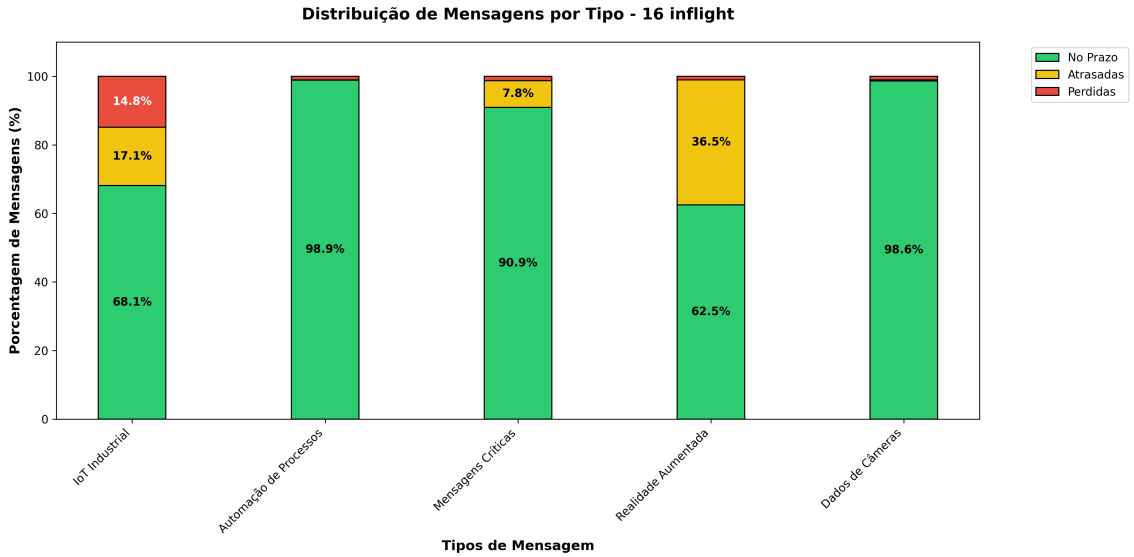

Figura 5.6 – Perda de mensagens escalando o *cluster* com o *inflight* em 16.

Com estas configurações, há uma melhora no desempenho quando comparado aos experimentos escalando com o uso da fila como parâmetro. Os experimentos escalando novos nodos ao atingir 1000 e 5000 mensagens na fila,as mensagens críticas que foram recebidas a tempo foram 48.8% e 51.8% do total, os experimentos com 16 e 32 mensagens *inflight*e atingiram respectivamente 90.9% e 89.1% de mensagens atendem seus requisitos de tempo.

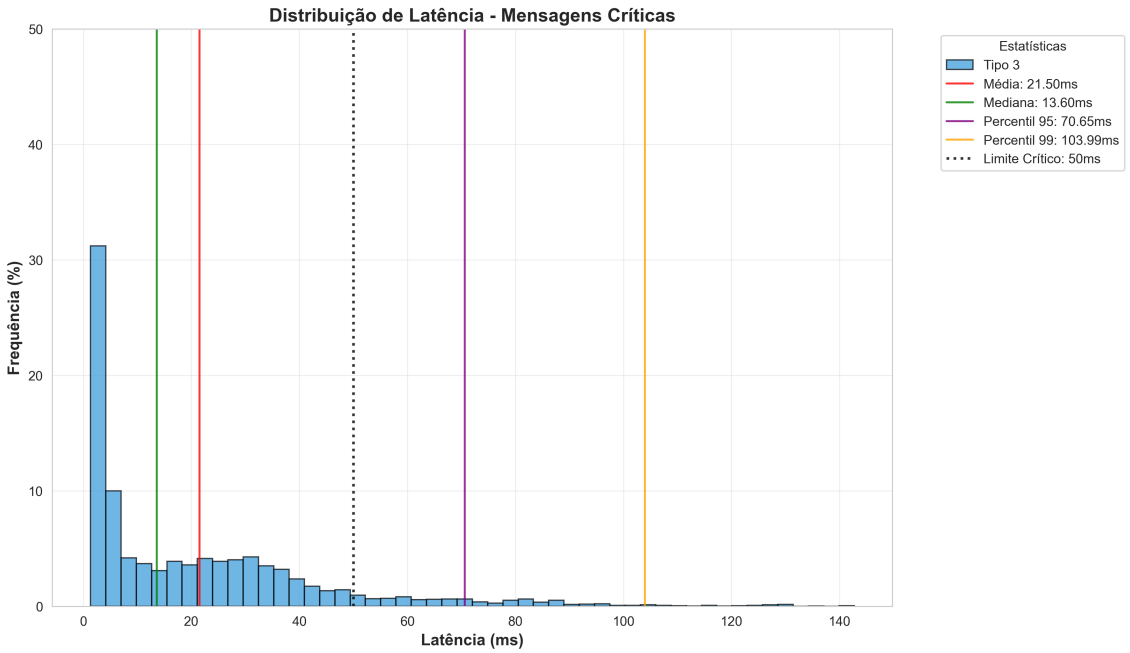

Figura 5.7 – Histograma de Latência de Mensagens Críticas com escala em *inflight* 32.

Ao configurar o Manager para escalar novos nodos quando o número de mensagens *inflight* alcança 32, observamos uma melhora significativa nos resultados, conforme apresentado na figura 5.7. A média da latência foi reduzida para 21,40 ms, com 95% das mensagens entregues em até 70.65 ms e 99% em até 103.99 ms. Esses resultados demonstram maior eficiência na entrega de mensagens críticas, atendendo melhor aos requisitos de tempo real e demonstrando vantagem na performance ao utilizar mensagens *inflight* como métrica para o escalonamento dinâmico.

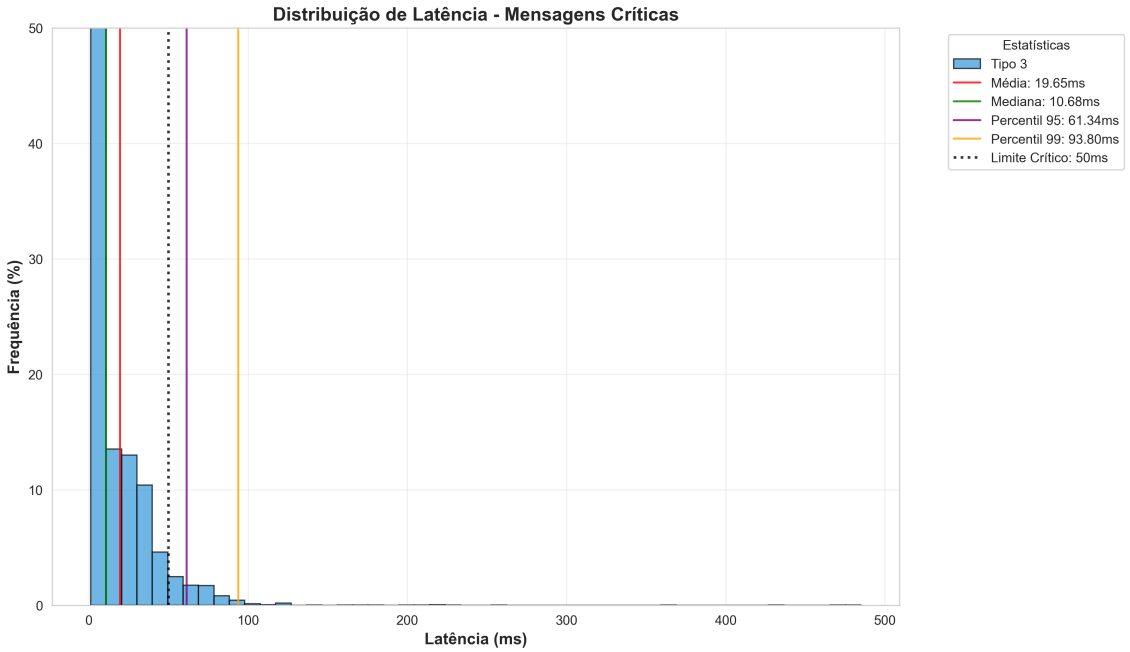

Figura 5.8 – Histograma de Latência de Mensagens Críticas com escala em *inflight* 16.

Ao configurar o Manager para escalar novos nodos quando o número de mensagens *inflight* atinge 16, os resultados apresentados na figura 5.8 mostram um desempenho ainda mais eficiente. A média da latência foi reduzida para 19,65 ms, com 95% das mensagens entregues em até 61,34 ms e 99% em até 93,80 ms. Esses resultados indicam uma latência significativamente menor, atendendo de forma eficiente aos requisitos de tempo real para aplicações críticas. Comparado a configurações com limites de escalonamento mais altos, como *inflight* em 32 ou filas em 1000 e 5000, a configuração com 16 mensagens *inflight* proporciona o melhor desempenho observado.

5.3.2 Uso de Recursos

A análise de uso de recursos focou na mensuração do consumo de CPU, memória e número de nodos do *cluster* MQTT. Esse estudo permite identificar o impacto da utilização do manager sobre os recursos disponíveis,

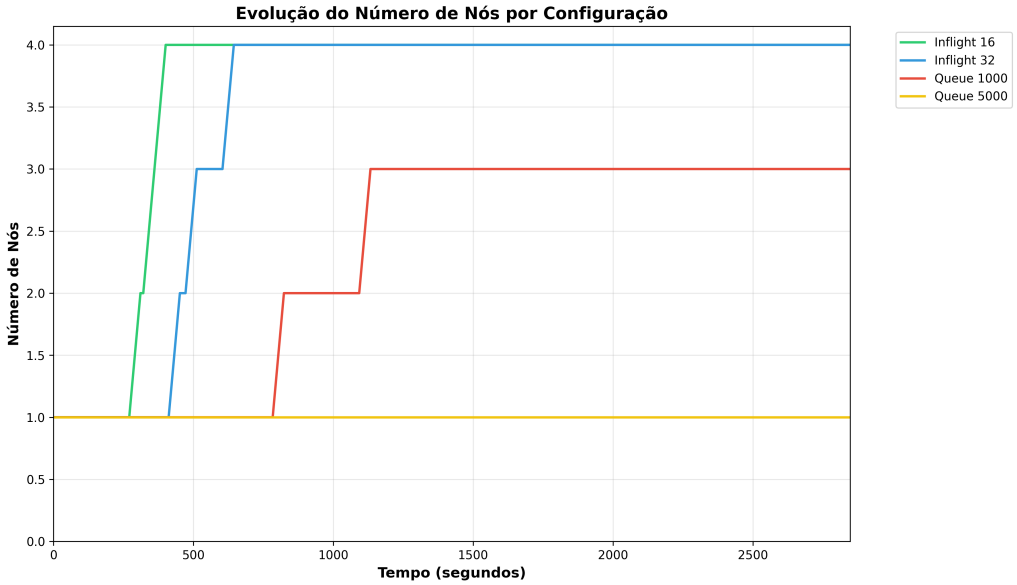

Figura 5.9 – Numero de nodos ao longo dos experimentos.

O gráfico apresentado na Figura 5.9 ilustra a evolução do número de nós no *cluster* MQTT ao longo do tempo. Observa-se um aumento gradual do número de nós durante os primeiros minutos, correlacionado com a conexão progressiva de clientes publicadores e assinantes. Picos no número de nós ocorrem em momentos de maior carga, quando limites de mensagens *inflight* ou filas configurados no Manager foram atingidos, acionando o escalonamento automático. Para a configuração do manager que adiciona novos nodos com a fila em 5000 mensagens, novos nodos não foram adicionados.

Foi realizada uma análise detalhada sobre o uso de CPU durante os experimento para verificar o impacto da adição de novos nós ao *cluster* MQTT no consumo de recursos. Com os resultados foram gerados as Figuras 5.10.

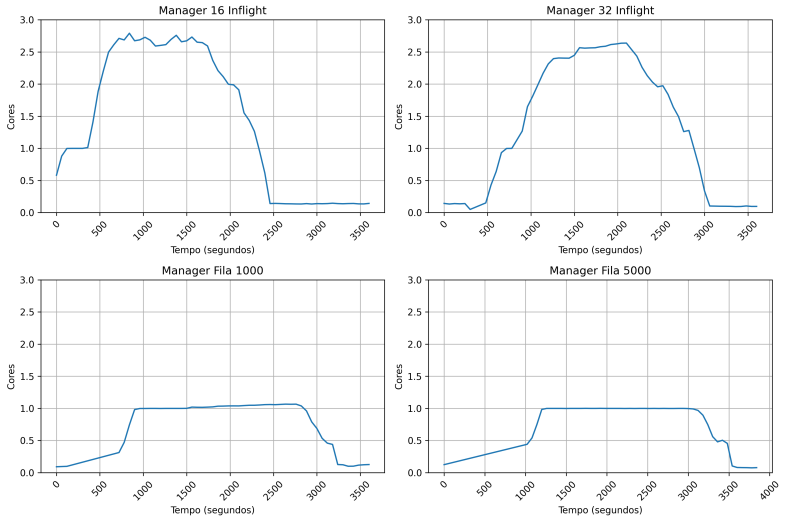

Figura 5.10 – Uso de CPU por experimento.

Com uma análise do uso de CPU ao longo do experimento, é possível perceber que com o Manager configurado para escalar com 16 mensagens *inflight*, o uso máximo de CPU alcançou 2.79, enquanto o uso médio foi de 1.42. Na configuração com 32 mensagens *inflight*, o pico foi reduzido para 2.64 e a média para 1.37. Por outro lado, as configurações baseadas na fila mostraram menor impacto no uso de CPU: com uma fila de 1000 mensagens, o pico registrado foi de 1.07 e a média de 0.78, enquanto com uma fila de 5000 mensagens, o pico foi de 1.00 e a média de 0.79. Esses resultados indicam que as configurações baseadas em filas têm menor impacto no consumo de CPU, enquanto o uso de mensagens *inflight* demanda maior capacidade computacional.

Também foram realizadas análises do consumo de memória por nodos do *broker* durante a execução destes experimentos, gerando as Figuras 5.11.

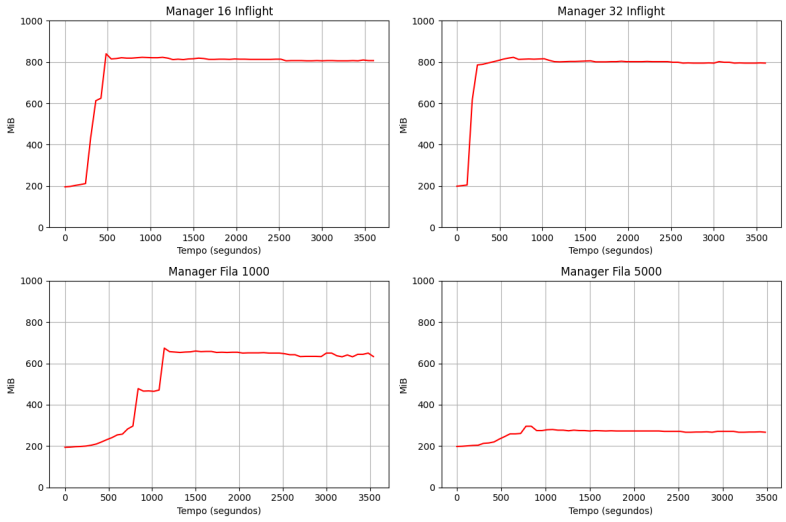

Figura 5.11 – Uso de memória por experimento.

Na configuração onde o Manager escala com *inflight* 16, foi registrado o maior uso de pico, atingindo 840.00, com uma média de 750.79 MiB. Escalando com 32 mensagens *inflight* observa-se um pico levemente inferior, de 823.00 MiB, mas com uma média maior, de 769.51 MiB. No caso onde a escala acontece com 1000 mensagens na fila, o pico foi reduzido para 674.00, e a média caiu para 534.95 MiB. Por último, a configuração usando 5000 mensagens na fila 5000 apresentou os menores valores, com um pico de 296.00 MiB e média de 262.27 MiB. O Uso de memória presente nas Figuras 5.11 coincide com o número de nodos escalados, mostrado na Figura 5.9.

Finalmente, foi realizado um estudo sobre o uso de recursos de rede pelos *brokers* em cada um dos cenários de experimentação. Através desta análise foi possível gerar Figuras que apresentam dados recebidos 5.12 e transmitidos 5.13 por cada um dos nodos do *cluster* durante a execução dos experimentos.

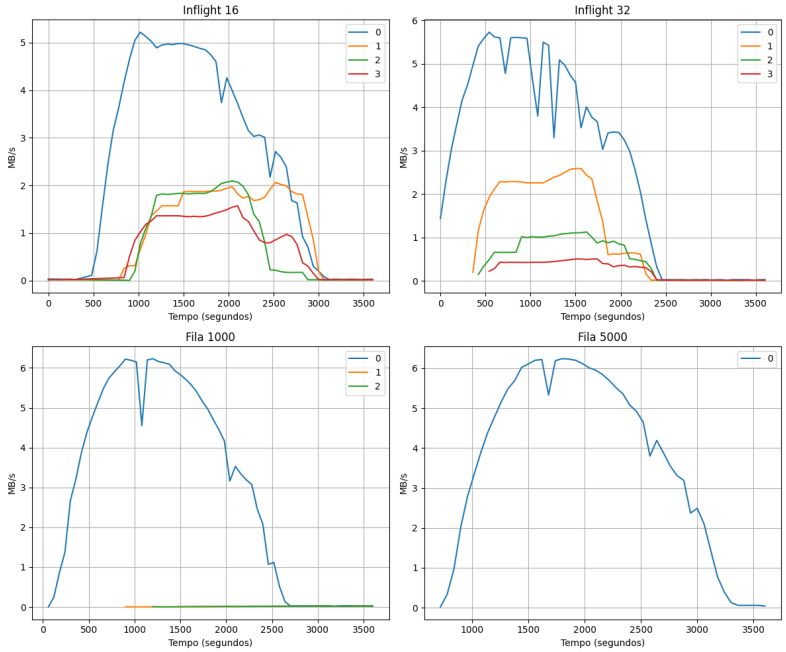

Figura 5.12 – Dados recebidos ao Longo dos experimento

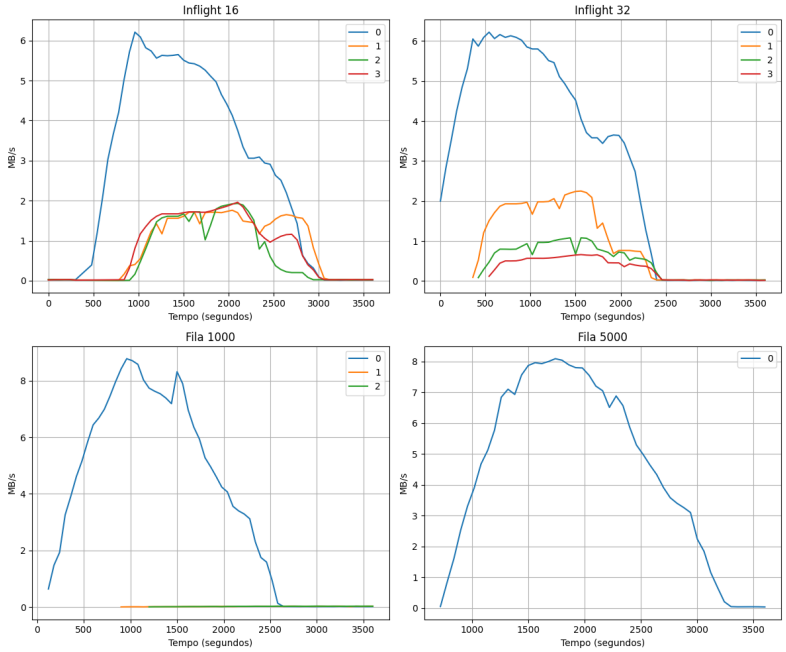

Figura 5.13 – Dados transmitidos ao longo dos experimentos.

Analisando estas Figuras podemos verificar o que o consumo de rede para a transmissão de dados. Para os experimentos usando o *inflight*, para a escala, nodos são adicionados mais cedo. Desta forma, mais clientes se conectam e estes servidores, sendo capazes de gerar um tráfego maior. Utilizando a escala em 16 inflight foi obtido um máximo de 10.57 MB/s e uma média de 5.07 MB/s, e utilizando 32 de inglifht foi obtido um máximo de 9.36 MB/s e uma média de 4.47 MB/s. No Experimento escalando com 1000 mensagens na fila, a escala acontece muito tardiamente, em um momento do experimento

onde todos clientes já estão conectados ao nodo inicial, emqx-0. Neste experimento é transmitido no máximo, 8.79 MB/s, porém em média 3.72 MB/s. No experimento usando a escala em 5000 de fila novos nodos não instanciados fazendo com que apenas o nodo inicial receba ou transmita dados. Este experimento apresenta um pico menor, com 8.09 MB/s porém média de 4.45 MB/s.

5.3.3 Análise de Resultados

Utilizando a configuração de escala com 5000 mensagens na fila, não houve escala de nodos adicionais, com uso de pico de 296.00 MiB e média de 262.27 MiB. Nesta configuração, apenas 51.8% das mensagens atenderam aos requisitos de tempo.

Em termos de latência, a melhor performance foi com *inflight* 16, com média de 19,65 ms e 90.9% das mensagens atendem seus requisitos de tempo. Esta configuração também mostrou o maior uso de memória, com pico de 840.00 MiB e média de 750.79 MiB e maior uso de CPU com pico de 2.79 e média de 1.42. Usando a configuração com 32 mensagens *inflight* há um pico menor, de 823.00 MiB e maior média 769.51 MiB. O uso de CPU também é menor, utilizando 2.64 no pico e 1.37 em média, porém a latência das mensagens críticas também é maior, com 89.1% das mensagens atendendo aos requisitos. Essas configurações por gerar nodos mais cedo, criam um cenário com conexões melhor distribuídas, o que por consequência faz com que o tráfego de mensagens seja maior, apresentando um pico de 10.57 MB/s no *inflight* 16 .

Com a escala ocorrendo com 1000 mensagens na fila, o pico foi reduzido para 674.00 MiB, com 2 nodos sendo adicionados no total. Esta configuração apresentou maior uso de recursos que a configuração que escalava com 5000 mensagens, porém demonstrou uma performance pior, com apenas 48.8% das mensagens atendendo aos requisitos de tempo.

As configurações de escala usadas foram mensagens *inglight* (16 e 32) e mensagens na fila (1000 e 5000). Nestes experimentos, o resultado que melhor atenderam os requisitos de tempo foi a configuração de *inflight* 16, onde 90.9% das mensagens enviadas atenderam os requisitos de tempo. Este também foi o cenário com maior consumo de CPU e memória, utilizando 1.42 *cores* em média, 2.79 *cores* em momentos de pico e uma média de 750.79 MiB e 840.00 MiB em momentos de pico. Nestes experimentos, também se percebe que a escalando novos nodos ao *cluster* tardiamente, prejudica a performance da aquisição de dados.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A tecnologia de *digital twins* permite a criação de representações digitais de objetos físicos, sistemas ou processos, tornando possível monitorar, simular e otimizar o desempenho desses ativos em tempo real. Este trabalho teve como proposta explorar como o protocolo MQTT pode ser utilizado para implementar a aquisição de dados de *digital twins*.

Com este fim, foi implementado o Manager, um componente adicional à arquitetura do MQTT, com o objetivo de escalar novos nodos a um *cluster* dinamicamente para lidar com demandas flutuantes enquanto atende requisitos de tempo e foi realizada uma série de experimentos, utilizando diferentes limites para escalar novos nodos.

Com o manager foram realizados experimentos, testando diferentes configurações para a escala de novos nodos, sendo elas 16 mensagens *inflight*, 32 mensagens *inflight*, 1000 mensagens na fila, e 5000 mensagens na fila. Dentre estas configurações as que utilizaram mensagens *inflight* apresentaram performance melhor que configurações usando a fila. Foi também perceber também que realizar a escala de maneira muito tardia é prejudicial para a performance do sistema, como demonstrado pelo experimento utilizando fila 1000, ter ter pior performance que o fila 5000, que não escala.

É importante garanstir a qualidade da comunicação para que *digital twins* produzam análises corretas e precisas. O uso de um manager para orquestrar a quantidade de nodos em um *cluster* MQTT, apresenta uma melhora na qualidade dos serviços prestados, porém ainda não atinge os padrões definidos pela indústria.

A partir deste trabalho, podemos realizar novos experimentos com os componentes implementados. Como trabalho futuro podem ser levados em consideração diversos fatores para escala por vez, como por exemplo adicionar o uso de CPU ou rede ao processo de decisão de escala do Manager. Outra possibilidade seria determinar também fatores de de-escala, onde o Manager remove nodos do *cluster* para liberar recursos que estão alocados, porém não sendo utilizados. Isso poderia ser realizado quando ao monitorar o uso de recursos do *broker* remove nodos ao detectar que um deles está ocioso a muito tempo, Finalmente, também podem ser utilizadas técnicas de *machine learning* para determinar o número ideal de nodos em algum instante.

REFERÊNCIAS

AMT. **MTConnect docs**. 2023. <https://www.mtconnect.org/standard-download20181>. Acessed May 24th 2024.

BANKS, A.; GUPTA, R. **MQTT Version 3.1.1**. 2015. OASIS Standard. Available from Internet: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.

BATTY, M. **Digital twins**. [S.l.]: SAGE Publications Sage UK: London, England, 2018. 817–820 p.

BELLAVISTA, P. et al. Application-driven network-aware digital twin management in industrial edge environments. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 11, p. 7791–7801, 2021.

CATHEY, G. et al. Edge Centric Secure Data Sharing with Digital Twins in Smart Ecosystems. In: **2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)**. [S.l.]: IEEE Computer Society, 2021. p. 70–79.

CORONADO, P. D. U. et al. Part data integration in the Shop Floor Digital Twin: Mobile and cloud technologies to enable a manufacturing execution system. **Journal of manufacturing systems**, Elsevier, v. 48, p. 25–33, 2018.

DAMJANOVIC-BEHRENDT, V.; BEHRENDT, W. An open source approach to the design and implementation of digital twins for smart manufacturing. **International Journal of Computer Integrated Manufacturing**, Taylor & Francis, v. 32, n. 4-5, p. 366–384, 2019.

EMQ TECHNOLOGIES CO., LTD. **EMQX Documentation**. [S.l.], 2023. Available at https://www.emqx.io/docs/en/latest/.

FERNáNDEZ, F. et al. And quic meets iot: performance assessment of mqtt over quic. In: **2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)**. [S.l.: s.n.], 2020. p. 1–6.

FUKUSHIMA, Y. et al. Digimobot: Digital twin for human-robot collaboration in indoor environments. In: IEEE. **2021 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2021. p. 55–62.

FULLER, A. et al. Digital twin: Enabling technologies, challenges and open research. **IEEE access**, IEEE, v. 8, p. 108952–108971, 2020.

GLAESSGEN, E.; STARGEL, D. The digital twin paradigm for future nasa and us air force vehicles. In: **53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA**. [S.l.: s.n.], 2012. p. 1818.

GRIEVES, M. Digital twin: manufacturing excellence through virtual factory replication. **White paper**, v. 1, n. 2014, p. 1–7, 2014.

HAMIDOVIC, D. et al. **5G Campus Network Factory Floor Measurements with Varying Channel and QoS Flow Priorities**. 2023. Available from Internet: <https://arxiv.org/abs/2306.03671>.

HIVEMQ GMBH. **HiveMQ Documentation**. 2023. Available at https://www.hivemq.com.

HU, L. et al. Modeling of cloud-based digital twins for smart manufacturing with mt connect. **Procedia manufacturing**, Elsevier, v. 26, p. 1193–1203, 2018.

HUNKELER, U.; TRUONG, H.-L.; STANFORD-CLARK, A. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In: IEEE. **Proceedings of the 3rd international conference on communication systems software and middleware and workshops**. [S.l.], 2008. p. 791–798.

IYENGAR, J.; THOMSON, M. Quic: A udp-based multiplexed and secure transport. **RFC 9000**, 2021.

KOZIOLEK, H.; GRÜNER, S.; RÜCKERT, J. A comparison of mqtt brokers for distributed iot edge computing. In: SPRINGER. **Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings 14**. [S.l.], 2020. p. 352–368.

LI, K. et al. Cross-layer resource allocation for urllc industrial automation over multi-connectivity. **IEEE Transactions on Wireless Communications**, v. 23, n. 7, p. 7334–7348, 2024.

LI, W. et al. Digital twin for battery systems: Cloud battery management system with online state-of-charge and state-of-health estimation. **Journal of Energy Storage**, Elsevier, v. 30, p. 101557, 2020. ISSN 2352-152X.

LIGHT, R. Mosquitto: Server and client implementation of the mqtt protocol. **Journal of Open Source Software**, v. 2, n. 13, p. 265, 2017.

LONGO, E. et al. Mqtt-st: a spanning tree protocol for distributed mqtt brokers. In: IEEE. **ICC 2020-2020 IEEE International Conference on Communications (ICC)**. [S.l.], 2020. p. 1–6.

MOURADIAN, C.; MEDHI, D. Optimizing iot application performance using mqtt over quic. **IEEE Access**, v. 6, p. 73849–73863, 2018.

OASIS. Mqtt version 3.1.1. **OASIS Standard**, 2014. Available from Internet: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

RODRIGO, M. S. et al. Digital twins for 5g networks: A modeling and deployment methodology. **IEEE Access**, IEEE, 2023.

RODRIGUES, B. e. a. Scaling mqtt for the internet of things. **IEEE IoT Journal**, 2018.

SHAHRI, E.; PEDREIRAS, P.; ALMEIDA, L. A scalable real-time sdn-based mqtt framework for industrial applications. **IEEE Open Journal of the Industrial Electronics Society**, IEEE, 2024.

SPOHN, M. A. On mqtt scalability in the internet of things: issues, solutions, and future directions. **Journal of Electronics and Electrical Engineering**, p. 4–4, 2022.

STANFORD-CLARK, A.; NIPPER, A. Mq telemetry transport (mqtt). 1999.

TAO, F. et al. Digital twin in industry: State-of-the-art. **IEEE Transactions on industrial informatics**, IEEE, v. 15, n. 4, p. 2405–2415, 2018.

TREVISAN, R. et al. Aquisição de dados escalável e ciente da aplicação para gêmeos digitais. In: **Anais do XXVII Workshop de Gerência e Operação de Redes e Serviços**. Porto Alegre, RS, Brasil: SBC, 2022. p. 57–70. ISSN 2595-2722. Available from Internet: <https://sol.sbc.org.br/index.php/wgrs/article/view/21477>.

ZDANKIN, P. et al. A digital-twin based architecture for software longevity in smart homes. In: IEEE. **2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)**. [S.l.], 2022. p. 669–679.

ZHOU, L.; FANG, W. Evaluating latency in high-demand iot systems using mqtt. **Elsevier Computer Networks**, 2019.