



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Adriano Zaniboni de Pellegrin

Controle de Back-Pressure em Filas de Mensagens: Utilizando um Controlador PID
Discreto

Florianópolis
2026

Adriano Zaniboni de Pellegrin

Controle de Back-Pressure em Filas de Mensagens: Utilizando um Controlador PID
Discreto

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.

Orientador: Prof. Publio Macedo Monteiro Lima, Dr.

Pellegrin, Adriano Zaniboni De
Controle de Back-Pressure em Filas de Mensagens :
Utilizando um Controlador PID Discreto / Adriano Zaniboni
De Pellegrin ; orientador, Publio Macedo Monteiro Lima,
2026.

84 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro
Tecnológico, Graduação em Engenharia de Controle e
Automação, Florianópolis, 2026.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Filas de
Mensagens. 3. Rate limit. 4. Controlador discreto. 5.
Banco de dados. I. Lima, Publio Macedo Monteiro. II.
Universidade Federal de Santa Catarina. Graduação em
Engenharia de Controle e Automação. III. Título.

Adriano Zaniboni de Pellegrin

Controle de Back-Pressure em Filas de Mensagens: Utilizando um Controlador PID
Discreto

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 3 de agosto de 2026.

Prof. Publio Macedo Monteiro Lima, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Publio Macedo Monteiro Lima, Dr.
Orientador e Supervisor
UFSC/CTC/EAS

Vinícius Moreno Sanches, Me.
Avaliador
UFSC/CTC/EAS

Prof. Hector Bessa Silveira, Dr.
Presidente da Banca
UFSC/CTC/EAS

AGRADECIMENTOS

Ao professor Publio Macedo Monteiro Lima, pela orientação ao longo deste trabalho.

Aos professores do Departamento de Automação e Sistemas da UFSC, em especial aos professores Daniel Juan Pagano e Marcelo Menezes Morato, pela colaboração para explorar o problema de controle, e ao professor Hector Bessa Silveira, pelo apoio e orientação para estruturar a ideia inicial do projeto.

Aos amigos e colegas da Automação, que são essenciais para a jornada de formação, mas também para a vivência em uma nova fase de nossas vidas.

Aos colegas das empresas em que trabalhei, que, cada um à sua maneira, me ensinaram a ser um profissional e uma pessoa melhores.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 3 de agosto de 2026.

Na condição de representante do Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina, no qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.



Documento assinado digitalmente
Publio Macedo Monteiro Lima
Data: 05/08/2026 17:45:11-0300
CPF: ***.190.417-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Publio Macedo Monteiro Lima, Dr.
Departamento de Automação e Sistemas da UFSC

RESUMO

Sistemas de *software* que processam dados de forma assíncrona utilizam filas de mensagens para desacoplar os componentes que produzem mensagens daqueles que as consomem. Essa independência cria um risco operacional: o consumidor pode processar mensagens mais rapidamente do que o serviço de destino, tipicamente um banco de dados, consegue suportar, saturando seu processamento e degradando todas as aplicações que dele dependem. A prática usual de limitar o consumo por uma taxa fixa opera em malha aberta e não se adapta a variações de carga nem a perturbações externas. O presente trabalho projeta, implementa e valida um sistema de controle de *back-pressure* para consumidores de filas de mensagens, no qual um controlador PID discreto regula a taxa de consumo de modo a manter a utilização de CPU do serviço de destino em torno de um valor de referência. O sistema foi implementado na linguagem Go com arquitetura hexagonal, que torna intercambiáveis a fila (Apache Kafka, AWS SQS), o serviço de destino (MySQL, HTTP) e a fonte de métricas (Prometheus, AWS CloudWatch), e inclui interface de monitoramento e de ajuste de parâmetros em tempo de execução. A planta foi identificada por ensaios de resposta ao degrau em malha aberta; os controladores foram sintonizados a partir do modelo identificado e comparados em malha fechada, avaliando sobressinal, tempos característicos e rejeição de perturbação. Os resultados mostram que a dinâmica do banco de dados é rápida frente à cadência de publicação das métricas, de modo que o atraso dominante da malha vem da medição e da amostragem, e não da planta. Nesse regime, o controlador PI com *anti-windup* condicional, complementado por filtro de *setpoint* e ação *feedforward* de referência, manteve a utilização no valor de referência, partiu sem sobressinal, rejeitou perturbações externas de carga e não perdeu mensagens, com desempenho superior ao da taxa fixa convencional; estruturas dependentes de modelo, como o Preditor de Smith, não trouxeram vantagem sob o modelo imperfeito do processo real. Além do mecanismo de controle pronto para produção, o trabalho entrega um laboratório reproduzível de identificação e monitoramento de plantas de *software*.

Palavras-chave: Controle de back-pressure. Controlador PID discreto. Filas de mensagens.

ABSTRACT

Software systems that process data asynchronously rely on message queues to decouple the components that produce messages from those that consume them. This independence creates an operational risk: the consumer may process messages faster than the destination service, typically a database, can sustain, saturating its processing capacity and degrading every application that depends on it. The usual practice of limiting consumption with a fixed rate operates in open loop and does not adapt to load variations or external disturbances. This work designs, implements, and validates a back-pressure control system for message queue consumers, in which a discrete PID controller regulates the consumption rate so as to keep the CPU utilization of the destination service around a reference value. The system was implemented in Go with a hexagonal architecture that makes the queue (Apache Kafka, AWS SQS), the destination service (MySQL, HTTP), and the metrics source (Prometheus, AWS CloudWatch) interchangeable, and includes an interface for monitoring and adjusting parameters at runtime. The plant was identified through open-loop step-response experiments; the controllers were tuned from the identified model and compared in closed loop, evaluating overshoot, characteristic times, and disturbance rejection. The results show that the database dynamics are fast compared with the publication cadence of the metrics, so the dominant loop delay comes from measurement and sampling rather than from the plant. In this regime, the PI controller with conditional anti-windup, complemented by a setpoint filter and a reference feedforward action, kept utilization at the reference value, started without overshoot, rejected external load disturbances, and lost no messages, outperforming the conventional fixed rate; model-dependent structures such as the Smith Predictor brought no advantage under the imperfect model that the real process offers. Besides the production-ready control mechanism, the work delivers a reproducible laboratory for the identification and monitoring of software plants.

Keywords: Back-pressure control. Discrete PID controller. Message queues.

LISTA DE FIGURAS

Figura 1 – Topologia típica de uma aplicação <i>web</i> : a camada de aplicação escala horizontalmente, mas as requisições convergem para um único banco de dados.	23
Figura 2 – Modelos de entrega <i>push</i> e <i>pull</i> : no primeiro, o intermediário dita o ritmo; no segundo, o consumidor solicita mensagens quando está pronto, o que viabiliza o controle da taxa de consumo.	24
Figura 3 – Mecanismo do <i>token bucket</i> : fichas chegam ao balde a taxa constante r , até a capacidade b ; cada unidade de trabalho consome uma ficha para ser liberada, e trabalho sem ficha espera.	25
Figura 4 – Camadas concêntricas da <i>Clean Architecture</i> : entidades no centro, casos de uso ao redor, adaptadores de interface na camada seguinte e as tecnologias externas na borda; a seta radial marca o sentido único da dependência, sempre para dentro.	27
Figura 5 – Analogia hidráulica do sistema: a fila corresponde ao tanque, o <i>token bucket</i> à válvula de saída e a utilização de CPU do banco é a variável controlada.	29
Figura 6 – Estrutura do Preditor de Smith: o PI opera sobre o erro predito, com o modelo interno sem atraso em paralelo com a planta real.	31
Figura 7 – A cadeia de medição de CPU, do escalonador do kernel aos dois consumidores: a malha de controle e o supervisor leem a mesma fonte.	32
Figura 8 – Papel do modelo de custo do otimizador: os planos candidatos de uma mesma consulta recebem custos estimados em unidades abstratas e o de menor custo é executado; o número serve para comparar planos, não para prever tempo ou consumo.	34
Figura 9 – Fluxo de atualização em massa disputando o mesmo banco de dados da aplicação: as cargas se somam sem qualquer regulação. . .	36
Figura 10 – Os dois fluxos do exemplo do <i>marketplace</i> : a consulta das recomendações é síncrona e sensível à latência; a recomputação disparada pela compra é desacoplada pela fila e tolera atraso. Ambos convergem para o mesmo banco de dados.	37
Figura 11 – Estrutura da solução: a fila absorve o lote, o controlador regula a drenagem e a realimentação da CPU do banco fecha a malha; o tráfego regular atua como perturbação.	39
Figura 12 – Arquitetura <i>Ports and Adapters</i> do sistema: o núcleo de domínio não conhece as tecnologias concretas, acopladas por interfaces.	43

Figura 13 – Ciclo de entrega de uma mensagem pelo orquestrador: a confirmação na fila (<i>ack</i>) só ocorre após a confirmação do destino.	44
Figura 14 – Diagrama de blocos da malha completa: filtro de <i>setpoint</i> , PI com <i>anti-windup</i> , ação <i>feedforward</i> , atuador <i>token bucket</i> , planta e cadeia de medição com filtro opcional na realimentação, com o tráfego regular como perturbação.	46
Figura 15 – Visão geral do painel supervisorio durante a operação: indicadores instantâneos (taxa de processamento, drenagem comandada, CPU do banco, fila), seção do controlador e detalhamento do MySQL (conexões, memória, consultas por tipo, <i>buffer pool</i> , <i>locks</i> e tráfego de rede).	54
Figura 16 – Painéis da fila no supervisorio: total processado, taxa de consumo e tempos de operação, com a retirada da fila em microssegundos e o consumo completo, dominado pelo banco, em milissegundos.	55
Figura 17 – Cabeçalho de estado da interface: controlador ativo e parâmetros vigentes da malha.	55
Figura 18 – Painel de configuração da interface de operação: controlador, medição, parâmetros da malha, fila ativa e perturbação externa, todos ajustáveis em tempo de execução.	56
Figura 19 – Seção do supervisorio dedicada ao controlador: variável de processo, <i>setpoint</i> e drenagem; termos proporcional e integral; histórico de parâmetros; e drenagem contra o limite de saturação.	58
Figura 20 – Curva estática da planta no cenário padrão (tabela de 10^5 linhas): CPU de regime em função da taxa de drenagem.	59
Figura 21 – Curvas de reação e modelos FOPDT ajustados nos três regimes de medição (degrau de drenagem $10 \rightarrow 30$ msg/s, tabela de 10^5 linhas). Os marcadores indicam os pontos de 28,3 % e 63,2 % usados pelo método dos dois pontos.	60
Figura 22 – Respostas ao degrau dos três modelos identificados, normalizadas pelo ganho estático: o mesmo processo físico, visto por cadências de medição diferentes, aparenta dinâmicas separadas por mais de uma ordem de grandeza.	61
Figura 23 – Decomposição do atraso efetivo do laço nos dois regimes: a atuação e a medição, presas à cadência do laço, dominam; a planta responde em cerca de dois segundos em ambos.	62

Figura 24 – Ensaio de demonstração dos tipos de filtro de referência, em cadência acelerada ($T = 5$ s), com o PI puro partindo da planta ociosa e degrau de referência para 50 %: o exponencial conduz a referência assintoticamente; a média móvel fecha a rampa em exatamente oito amostras e a drenagem acompanha em rampa quase linear.	65
Figura 25 – Partida em malha fechada nos dois regimes: resposta prevista pela simulação do PI sintonizado sobre o modelo FOPDT identificado, sobreposta à resposta medida no ensaio real. A referência efetiva é conduzida pelo filtro de <i>setpoint</i>	65
Figura 26 – Malha fechada com o PI no regime de $T = 10$ s: partida sem sobresinal, rejeição e recuperação da perturbação. A referência efetiva, tracejada, é conduzida pelo filtro de <i>setpoint</i> exponencial de duas amostras.	67
Figura 27 – Malha fechada com o PI no regime de $T = 60$ s: mesma estrutura de ensaio, cadência de produção em nuvem. A referência efetiva, tracejada, é conduzida pelo filtro de <i>setpoint</i> exponencial de duas amostras.	68
Figura 28 – Sessão de validação do PI ($T = 10$ s) vista pelo supervisor: referência em rampa pelo filtro de <i>setpoint</i> , CPU regulada, janela de perturbação externa e recuperação.	69
Figura 29 – Comparação controlada no regime de $T = 10$ s: PI, PI com <i>feedforward</i> e PI com Preditor de Smith sob os mesmos ganhos de realimentação, na partida, na perturbação e na recuperação.	70
Figura 30 – PI e PI com <i>feedforward</i> no regime de $T = 60$ s, mesmos ganhos de projeto: o <i>feedforward</i> antecipa a partida; a rejeição de perturbação é equivalente.	71
Figura 31 – Preditor de Smith no regime de $T = 60$ s: com os ganhos de projeto do PI, a malha oscila e o ensaio é interrompido; com ganhos reduzidos, é estável porém mais lento que o PI puro.	72
Figura 32 – Ensaio de diagnóstico em cadência acelerada ($T = 5$ s): Preditor de Smith com ganhos da síntese direta, Preditor de Smith com modelo físico e ganhos do PI, e PI puro com os mesmos ganhos. Linhas verticais: degrau de <i>setpoint</i> e janela de perturbação.	73

LISTA DE QUADROS

Quadro 1 – Consolidação dos requisitos técnicos.	42
Quadro 2 – Rotas da API do sistema.	50
Quadro 3 – Exemplo comentado de configuração de implantação (regime de $T = 60$ s).	52
Quadro 4 – Verificação experimental dos requisitos técnicos.	77

LISTA DE TABELAS

Tabela 1 – Modelos FOPDT identificados por regime de medição (degrau de drenagem $10 \rightarrow 30$ msg/s, tabela de 10^5 linhas).	60
Tabela 2 – Componentes do atraso efetivo do laço nos dois regimes.	61
Tabela 3 – Ganhos do PI por regime, calculados pela regra SIMC com $\tau_c = \theta_{ef}$ sobre os modelos identificados.	63
Tabela 4 – Malha fechada com o PI: métricas nos dois regimes.	66
Tabela 5 – Comparação controlada no regime de $T = 10$ s: métricas das três estruturas com os mesmos ganhos de realimentação.	68
Tabela 6 – Comparação no regime de $T = 60$ s: métricas das três estruturas, cada qual na melhor sintonia viável.	70
Tabela 7 – Diagnóstico em $T = 5$ s: as duas variantes do preditor contra o PI puro com os mesmos ganhos ($K_p = 0,096$; $K_i = 0,035$).	74

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
CPU	<i>Central Processing Unit</i> (Unidade Central de Processamento)
DDD	<i>Domain-Driven Design</i>
E/S	Entrada e Saída
FOPDT	<i>First-Order Plus Dead Time</i>
HPA	<i>Horizontal Pod Autoscaler</i>
PID	Proporcional-Integral-Derivativo
pp	pontos percentuais
REST	<i>Representational State Transfer</i>
SIMC	<i>Simple Internal Model Control</i> (regra de sintonia de Skogestad)
SPOF	<i>Single Point of Failure</i>
SQS	<i>Simple Queue Service</i>
SRE	<i>Site Reliability Engineering</i>
TLS	<i>Transport Layer Security</i>

LISTA DE SÍMBOLOS

K	Ganho estático da planta
τ	Constante de tempo da planta
θ	Atraso de transporte (<i>dead time</i>)
θ_{ef}	Atraso efetivo (atraso de transporte somado ao de amostragem)
T	Período de amostragem
τ_c	Constante de tempo desejada em malha fechada (parâmetro da regra SIMC)
τ_I	Tempo integral do controlador PI
K_p	Ganho proporcional
K_i	Ganho integral
u	Variável manipulada: taxa de drenagem da fila (msg/s)
y	Variável de processo: utilização de CPU do banco de dados (%)
r	Referência (<i>setpoint</i>) de utilização de CPU (%)

SUMÁRIO

1	INTRODUÇÃO	17
1.1	CONTEXTO E MOTIVAÇÃO	17
1.2	DESCRIÇÃO DO PROBLEMA E SUA IMPORTÂNCIA	18
1.3	OBJETIVOS	19
1.4	METODOLOGIA E PRINCIPAIS TÉCNICAS	19
1.5	PRINCIPAIS RESULTADOS E AUTORIA	20
1.6	ORGANIZAÇÃO DO DOCUMENTO	21
2	FUNDAMENTAÇÃO TEÓRICA	22
2.1	SISTEMAS DE SOFTWARE EM PRODUÇÃO E CONFIABILIDADE	22
2.2	O BANCO DE DADOS COMO <i>SINGLE POINT OF FAILURE</i>	23
2.3	FILAS DE MENSAGENS E DESACOPLAMENTO ASSÍNCRONO	24
2.4	LIMITAÇÃO DE TAXA EM MALHA ABERTA	25
2.5	ARQUITETURA DE SOFTWARE: DOMÍNIO ISOLADO E INFRAESTRUTURA PLUGÁVEL	26
2.6	FUNDAMENTOS DE CONTROLE	28
2.6.1	O modelo FOPDT e sua identificação	28
2.6.2	Discretização e o papel da amostragem	29
2.6.3	O controlador PI, o <i>anti-windup</i> e a sintonia SIMC	30
2.6.4	O Preditor de Smith	30
2.6.5	Avaliação da resposta temporal	31
2.7	COMO SE MEDE A CPU: INSTRUMENTAÇÃO COMO SENSOR DA MALHA	32
2.8	PREDIÇÃO DE CUSTO DE CONSULTAS E SUAS LIMITAÇÕES	34
3	DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS	36
3.1	O PROBLEMA FORMALIZADO	36
3.2	MODELAGEM COMO MALHA DE CONTROLE	39
3.3	REQUISITOS FUNCIONAIS	41
3.4	REQUISITOS NÃO FUNCIONAIS	42
4	SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA	43
4.1	VISÃO GERAL DA ARQUITETURA	43
4.2	O ORQUESTRADOR	44
4.3	O CONTROLADOR PLUGÁVEL	45
4.4	O LEITOR DE MÉTRICAS	47
4.5	ADAPTADORES DE FILA E DE DESTINO	47
4.6	METODOLOGIA EXPERIMENTAL	48
5	DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS	49
5.1	IMPLEMENTAÇÃO DO SISTEMA	49

5.2	O AMBIENTE DE IDENTIFICAÇÃO E MONITORAMENTO	52
5.2.1	A cadeia de métricas: cAdvisor, Prometheus e Grafana	52
5.2.2	A interface de operação	53
5.2.3	Implantação e segurança	56
5.3	FERRAMENTAS DE ANÁLISE DOS EXPERIMENTOS	57
5.4	IDENTIFICAÇÃO DA PLANTA	58
5.4.1	Região linear de operação	58
5.4.2	Dois regimes de amostragem	59
5.4.3	O fator de atraso: de onde vem o atraso da malha	61
5.5	SINTONIZAÇÃO DOS CONTROLADORES	62
5.6	VALIDAÇÃO EM MALHA FECHADA	66
5.6.1	O PI nos dois regimes	66
5.6.2	Comparação controlada: o papel de cada estrutura	67
5.6.3	A causa da fragilidade: o atraso espalhado pela medição	71
5.6.4	Comparação com a taxa fixa	74
5.6.5	Exemplo de operação: a rotina experimental vista pela interface	74
5.7	O CONTROLADOR PARA O CASO DO BANCO DE DADOS	75
5.8	ENTREGÁVEIS E IMPACTOS	76
6	CONCLUSÃO	78
	REFERÊNCIAS	81

1 INTRODUÇÃO

Arquiteturas de *software* modernas dependem do processamento assíncrono de mensagens para desacoplar os componentes que produzem dados daqueles que os consomem. Nesse modelo, uma fila de mensagens atua como intermediária entre produtores e consumidores, e cada lado opera em seu próprio ritmo: os produtores publicam mensagens a taxas arbitrárias, enquanto os consumidores as processam conforme sua disponibilidade. O desacoplamento absorve variações de carga e evita que uma sobrecarga momentânea em um componente se propague imediatamente aos demais (KLEPPMANN, 2017).

Essa mesma independência, contudo, introduz um risco operacional. O consumidor não conhece o estado do serviço de destino (*downstream*) que efetivamente executa o trabalho, em geral um banco de dados, e pode drenar mensagens mais rápido do que esse serviço consegue suportar, levando-o à saturação. Um banco de dados saturado responde com lentidão e passa a recusar operações, o que compromete a confiabilidade da aplicação e atinge diretamente o usuário final. A prática usual de mitigar esse risco por meio de uma taxa de consumo fixa é limitada: um limite fixo não se adapta a variações de carga, como mensagens de custo computacional distinto ou perturbações externas sobre o serviço de destino.

O presente trabalho trata a taxa de consumo de mensagens como um problema de controle em malha fechada: em vez de fixá-la *a priori*, um controlador a ajusta continuamente de modo a manter a utilização de recursos do serviço de destino em torno de um valor de referência preestabelecido. Os capítulos seguintes detalham essa proposta, desde a fundamentação teórica até a validação experimental.

1.1 CONTEXTO E MOTIVAÇÃO

O uso de filas de mensagens é prática consolidada em arquiteturas distribuídas e orientadas a serviços, nas quais é comum que uma operação solicitada por um componente seja executada de forma assíncrona por outro (HOHPE; WOOLF, 2003). Picos de demanda são acomodados na fila, e uma indisponibilidade temporária do consumidor não impede que os produtores continuem operando. Filas estão presentes em sistemas de processamento de pagamentos, na ingestão de dados, na integração entre serviços e em rotinas de retaguarda.

O risco de saturação do serviço de destino aparece sobretudo no processamento em massa. Migrações de dados e reprocessamentos enfileiram grandes volumes de operações que, ao serem consumidas, disputam recursos com o tráfego regular da aplicação. Sem regulação do consumo, o banco de dados que atende a essas operações pode ter sua capacidade esgotada, degradando tanto a tarefa em lote quanto as requisições dos usuários. O custo de cada operação tampouco é uniforme:

uma mesma fila pode conter mensagens cujo impacto sobre o serviço de destino varia em ordens de grandeza. Um limite de taxa fixo acaba, portanto, conservador demais em um momento e agressivo demais em outro.

Busca-se, assim, um mecanismo que regule automaticamente a taxa de consumo a partir da carga observada no serviço de destino, mantendo-a em um nível seguro sem sacrificar a vazão. Tratar recursos computacionais como processos dinâmicos a controlar é uma linha de pesquisa estabelecida (HELLERSTEIN *et al.*, 2004), mas pouco presente na prática da infraestrutura de *software*, em que predominam limites fixos configurados manualmente.

O presente trabalho aborda esse problema com as ferramentas da teoria de controle. Ele foi desenvolvido integralmente em âmbito acadêmico, no Departamento de Automação e Sistemas (DAS) do Centro Tecnológico da Universidade Federal de Santa Catarina (UFSC), no contexto da disciplina DAS5511 (Projeto de Fim de Curso) do curso de Engenharia de Controle e Automação; não há, portanto, empresa concedente, e o cenário de aplicação é o de sistemas de *software* em produção em geral, e não o de uma organização específica.

1.2 DESCRIÇÃO DO PROBLEMA E SUA IMPORTÂNCIA

O problema tratado neste trabalho é a saturação de um serviço de destino causada pelo consumo não regulado de uma fila de mensagens. Quando a utilização de processamento (CPU) de um banco de dados se aproxima de 100 %, os tempos de resposta crescem de forma abrupta, consultas passam a falhar por esgotamento de tempo e dados podem ser perdidos. A degradação não fica restrita à tarefa que a causou: todas as aplicações que dependem do mesmo banco são afetadas ao mesmo tempo.

Para as organizações que operam esses sistemas, episódios de saturação têm custo concreto. Geram incidentes que consomem horas de engenharia e, quando recorrentes, minam a confiança dos usuários no serviço. As defesas usuais atacam o sintoma, não a causa: janelas de execução de madrugada e limites de taxa deliberadamente conservadores alongam a duração das tarefas em lote e desperdiçam capacidade nos períodos em que o banco está ocioso.

A limitação dessas defesas é estrutural. Um limite de taxa configurado manualmente opera em malha aberta: a decisão de quanto consumir não considera o estado atual do serviço de destino. Se a carga externa aumenta ou se as mensagens ficam mais custosas, o limite que era seguro passa a ser excessivo, e nada no sistema reage a isso. Fechar essa malha, medindo a carga do destino e ajustando a taxa de consumo em resposta, é a proposta central deste trabalho.

1.3 OBJETIVOS

O objetivo geral deste trabalho é projetar e construir o ferramental completo de controle de *back-pressure* para consumidores de filas de mensagens (o sistema de *software*, com fila, destino e fonte de métricas intercambiáveis, e o ambiente de identificação, sintonia e monitoramento que o acompanha) e validar a abordagem como prova de conceito, demonstrando que um controlador discreto mantém a utilização de CPU do serviço de destino em torno de um valor de referência, rejeitando perturbações de carga, como o tráfego regular da aplicação sobre o mesmo serviço, sem perda de mensagens.

Os objetivos específicos são:

- a) construir o sistema de *software* com arquitetura que isole o núcleo de controle das tecnologias de infraestrutura, tornando fila, serviço de destino e fonte de métricas intercambiáveis;
- b) construir o ferramental de experimentação: modo de malha aberta, injeção de perturbações, painel supervisão com ajuste de parâmetros em tempo de execução e cadeia de extração e análise de dados;
- c) modelar o processo como uma malha de controle em tempo discreto e identificar experimentalmente a planta (ganho, constante de tempo e atraso) sob diferentes condições de medição;
- d) sintonizar e comparar experimentalmente estratégias de controle (PI com *anti-windup*, ação *feedforward* e Preditor de Smith) nos diferentes regimes de medição;
- e) validar a prova de conceito em malha fechada por meio de testes de carga, avaliando seguimento de referência, rejeição de perturbações representativas do uso normal da aplicação (surto de consultas concorrentes que consomem CPU do serviço de destino por fora da fila) e ausência de perda de mensagens.

1.4 METODOLOGIA E PRINCIPAIS TÉCNICAS

A solução desenvolvida é um sistema de *software* que se posiciona entre a fila de mensagens e o serviço de destino. Um orquestrador consome as mensagens da fila e as entrega ao destino a uma taxa regulada; essa taxa é a variável manipulada da malha. A utilização de CPU do serviço de destino, obtida por instrumentação de métricas, é a variável de processo. Um controlador PID (proporcional-integral-derivativo) discreto fecha a malha, elevando a taxa quando há folga de capacidade e reduzindo-a quando a carga se aproxima do valor de referência.

O sistema foi implementado na linguagem Go, segundo a arquitetura *Ports and Adapters* (hexagonal), que isola a lógica de controle das tecnologias de infraestrutura. Essa separação permite trocar a fila, o serviço de destino ou a fonte de métricas sem alterar o núcleo do sistema. Foram desenvolvidos adaptadores para Apache Kafka e AWS SQS como filas, MySQL como serviço de destino e Prometheus e AWS CloudWatch como fontes de métricas. Um painel supervisorio em Grafana acompanha as variáveis da malha em tempo real, e os parâmetros do controlador podem ser lidos e ajustados durante a operação por meio de uma API REST.

Do ponto de vista experimental, adotou-se o fluxo clássico de projeto de sistemas de controle. Primeiro, a planta foi identificada por ensaios em malha aberta, aplicando degraus na taxa de consumo e registrando a resposta de CPU do banco de dados. Aos dados ajustou-se um modelo de primeira ordem com atraso de transporte. Em seguida, os controladores foram sintonizados a partir do modelo identificado e, por fim, validados em malha fechada com ensaios de seguimento de referência e de rejeição de perturbação, avaliados por métricas de desempenho consagradas na literatura de controle.

1.5 PRINCIPAIS RESULTADOS E AUTORIA

Os ensaios de identificação revelaram uma característica decisiva do processo: a dinâmica do banco de dados é rápida em comparação com o intervalo com que as métricas de carga ficam disponíveis. O atraso dominante da malha não está na planta, mas na própria medição. Esse resultado orientou a escolha e a sintonia dos controladores, favorecendo estruturas simples e ganhos moderados.

Em malha fechada, o sistema manteve a utilização de CPU do banco de dados em torno do valor de referência, rejeitou perturbações externas de carga e não perdeu mensagens durante os ajustes de taxa. A moldagem da referência por um filtro de *setpoint*, combinada ao *anti-windup* condicional, praticamente eliminou o sobressinal de partida. Na comparação com a estratégia convencional de taxa fixa, o controle em malha fechada aproveitou melhor a capacidade disponível do banco e reagiu a perturbações que a taxa fixa simplesmente ignora. O resultado é uma ferramenta utilizável em cenários reais de produção, com implantação automatizada em nuvem e comunicação criptografada.

Todo o desenvolvimento apresentado neste documento foi realizado pelo autor: o *software*, a infraestrutura de experimentação, as ferramentas de análise e os ensaios de identificação, sintonia e validação. O trabalho apoiou-se em ferramentas de código aberto consolidadas, como Apache Kafka, Prometheus e Grafana, identificadas ao longo do texto, e foi conduzido sob orientação acadêmica no Departamento de Automação e Sistemas da UFSC.

1.6 ORGANIZAÇÃO DO DOCUMENTO

O presente documento está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica sobre os principais conceitos e técnicas necessárias para o entendimento do problema abordado e da solução proposta. O problema tratado neste trabalho é descrito em detalhes no Capítulo 3, juntamente com os requisitos técnicos a serem atendidos. O Capítulo 4 aborda a solução proposta e a metodologia envolvida. O desenvolvimento realizado e a análise dos resultados obtidos são mostrados no Capítulo 5. Por fim, no Capítulo 6, são apresentadas as conclusões deste trabalho e algumas sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos necessários para o entendimento do problema tratado e da solução proposta. As Seções 2.1 a 2.4 cobrem o lado de engenharia de *software*: confiabilidade de sistemas em produção, o papel do banco de dados como ponto único de falha, o desacoplamento por filas de mensagens e a limitação de taxa em malha aberta. A Seção 2.5 descreve os princípios de arquitetura que orientaram a construção do sistema. A Seção 2.6 apresenta os fundamentos de controle empregados no trabalho. As duas seções finais tratam das grandezas em jogo na malha: a Seção 2.7 examina como a utilização de CPU é de fato medida em sistemas computacionais, e a Seção 2.8 discute por que o custo de processamento de uma consulta é difícil de prever, argumento que sustenta a opção deste trabalho pela realimentação.

2.1 SISTEMAS DE SOFTWARE EM PRODUÇÃO E CONFIABILIDADE

Um sistema de *software* em produção é avaliado menos pelas funcionalidades que oferece e mais pela regularidade com que as entrega. A literatura de engenharia de confiabilidade (*Site Reliability Engineering*, SRE) formaliza essa expectativa em indicadores e objetivos de nível de serviço, que quantificam disponibilidade e latência do ponto de vista do usuário (BEYER *et al.*, 2016). Violações desses objetivos têm um processo operacional próprio: abre-se um incidente, engenheiros de plantão são acionados e a causa é investigada em retrospectiva. Cada episódio consome horas de engenharia e, quando recorrente, corrói a confiança no serviço.

Grande parte desses incidentes não decorre de defeitos de lógica, mas de excesso de carga: um sistema dimensionado para certa demanda recebe mais trabalho do que consegue processar e degrada de forma abrupta (NYGARD, 2018). Sob saturação, o tempo de resposta cresce, as requisições se acumulam e os clientes reenviam pedidos que expiram, realimentando a sobrecarga. Nygard (2018) cataloga esse comportamento entre os padrões clássicos de falha de sistemas distribuídos e recomenda que toda integração entre componentes imponha limites explícitos ao trabalho aceito.

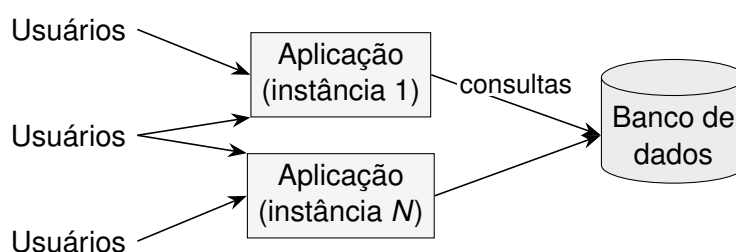
Proteger um serviço contra a própria demanda é, portanto, um requisito de projeto: os padrões de prontidão para produção descritos por Fowler (2017) colocam estabilidade e confiabilidade como condições para um serviço operar, ao lado de escalabilidade e tolerância a falhas. As técnicas usuais incluem limitação de taxa, descarte seletivo de requisições e amortecimento por filas. Este trabalho se concentra no caso em que o componente a proteger é um banco de dados e o tráfego excedente vem de um consumidor de filas, cenário detalhado nas seções seguintes.

2.2 O BANCO DE DADOS COMO *SINGLE POINT OF FAILURE*

Para situar o problema, convém recapitular como uma aplicação *web* típica atende a seus usuários. Cada ação na interface (abrir uma página, buscar um produto, confirmar uma compra) torna-se uma requisição ao servidor de aplicação, que a atende executando uma ou mais consultas ao banco de dados e montando a resposta com os resultados. O tempo que o usuário percebe é dominado, na maior parte dos casos, pelo tempo dessas consultas. As leituras respondem perguntas; as escritas registram fatos novos e, com frequência, disparam trabalho adicional, como recomputações e invalidação de *caches*.

Sobre essa mecânica assenta-se a arquitetura típica: a camada de aplicação, que não guarda estado, separa-se da camada de dados, que o concentra. As instâncias da aplicação podem ser replicadas livremente atrás de um balanceador de carga; o banco de dados, porém, é em geral único, porque replicar estado com consistência é caro e complexo (KLEPPMANN, 2017). A Figura 1 ilustra essa topologia: por mais que a camada de aplicação escale horizontalmente, todos os caminhos de requisição convergem para o mesmo banco.

Figura 1 – Topologia típica de uma aplicação *web*: a camada de aplicação escala horizontalmente, mas as requisições convergem para um único banco de dados.



Fonte: Elaborada pelo autor (2026).

Essa convergência faz do banco o ponto único de falha (*Single Point of Failure*, SPOF) mais comum dessas arquiteturas: quando ele satura, nenhuma réplica de aplicação compensa. A saturação de CPU é particularmente traiçoeira porque a degradação não é linear. Com folga de processamento, o tempo de resposta de uma consulta é praticamente constante; perto do limite, pequenos acréscimos de carga multiplicam a latência, consultas expiram e as conexões se esgotam. O efeito se propaga para todas as aplicações que compartilham o banco, mesmo as que nada têm a ver com a origem da carga.

Os gatilhos mais frequentes desse cenário são justamente as tarefas legítimas de manutenção e evolução: migrações de esquema, correções de dados em massa e reprocessamentos. São operações necessárias, mas cuja carga se soma ao trá-

fego regular da aplicação sobre o mesmo recurso. O desafio não é eliminá-las, e sim executá-las na velocidade que o banco tolera a cada instante.

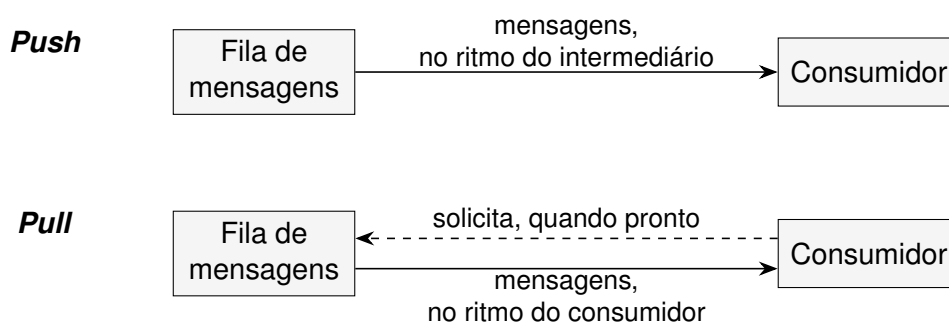
2.3 FILAS DE MENSAGENS E DESACOPLAMENTO ASSÍNCRONO

Uma fila de mensagens é um intermediário que armazena mensagens produzidas por um componente até que outro as consuma. O padrão produtor–consumidor desacopla os dois lados no tempo: o produtor não espera o processamento, e o consumidor não precisa estar disponível no momento da produção (HOHPE; WOOLF, 2003). Sistemas de mensageria modernos, como o Apache Kafka, acrescentam durabilidade (as mensagens sobrevivem a reinícios) e rastreamento de posição de leitura (*offset*), o que permite retomar o consumo do ponto exato em que parou (APACHE SOFTWARE FOUNDATION, 2026).

Para este trabalho, a distinção mais importante entre tecnologias de fila é o modelo de entrega. No modelo *push*, o intermediário empurra as mensagens para o consumidor assim que chegam, e o consumidor tem pouco ou nenhum controle sobre o ritmo de recebimento. No modelo *pull*, é o consumidor quem solicita mensagens quando está pronto para processá-las. O modelo *pull*, adotado pelo Kafka e pelo AWS SQS, é o que viabiliza o controle fino da taxa de consumo: basta espaçar as solicitações para reduzir a vazão, sem descartar nem acumular mensagens no consumidor (KLEPPMANN, 2017).

A Figura 2 contrasta os dois modelos: no *push*, o fluxo de mensagens parte do intermediário e o consumidor apenas o recebe; no *pull*, cada entrega é precedida de uma solicitação do consumidor, e é exatamente nesse ponto que uma taxa pode ser imposta.

Figura 2 – Modelos de entrega *push* e *pull*: no primeiro, o intermediário dita o ritmo; no segundo, o consumidor solicita mensagens quando está pronto, o que viabiliza o controle da taxa de consumo.



Fonte: Elaborada pelo autor (2026).

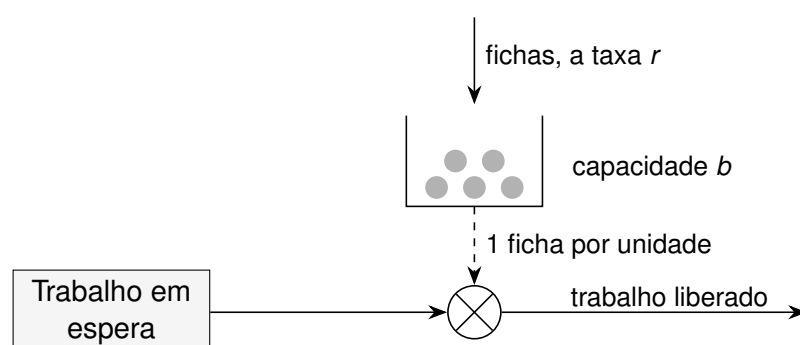
A capacidade de um consumidor de sinalizar ou impor um ritmo sustentável ao fluxo que recebe é conhecida como *back-pressure* (contrapressão). Em sistemas

com filas, a contrapressão tem um custo visível: as mensagens não consumidas se acumulam, e o atraso da fila (*lag*) cresce. Esse acúmulo é aceitável por construção, pois a fila existe exatamente para absorvê-lo; o que não é aceitável é transferir o excesso para o serviço que executa o trabalho. A questão central passa a ser, então, com que taxa drenar a fila.

2.4 LIMITAÇÃO DE TAXA EM MALHA ABERTA

O mecanismo clássico para impor uma taxa máxima a um fluxo é o algoritmo *token bucket*, ilustrado na Figura 3. Um balde acumula fichas a uma taxa constante r , até um limite máximo b ; cada unidade de trabalho consome uma ficha, e trabalho sem ficha disponível espera. O parâmetro r define a vazão média permitida, e a capacidade do balde define a tolerância a rajadas (TANENBAUM; WETHERALL, 2011). É o mecanismo por trás dos limites de “requisições por minuto” oferecidos por praticamente todas as plataformas e APIs comerciais.

Figura 3 – Mecanismo do *token bucket*: fichas chegam ao balde a taxa constante r , até a capacidade b ; cada unidade de trabalho consome uma ficha para ser liberada, e trabalho sem ficha espera.



Fonte: Adaptada de Tanenbaum e Wetherall (2011).

Os dois parâmetros têm papéis operacionais distintos, e convém distingui-los. A taxa r é o compromisso de longo prazo: em qualquer janela suficientemente longa, o fluxo médio não excede r , qualquer que seja o padrão de chegada do trabalho. A capacidade do balde governa o curto prazo: um balde fundo permite que um lote acumulado seja despachado de uma vez (as fichas guardadas cobrem a rajada), enquanto um balde raso força o fluxo a ser quase uniforme. A variante *leaky bucket*, que drena a fila a intervalos estritamente regulares, equivale ao caso extremo de balde unitário (TANENBAUM; WETHERALL, 2011). Para o problema deste trabalho, a distinção importa porque a variável que interessa proteger é a carga média do banco de dados: a taxa média é o que o controle precisa manipular, e a capacidade do balde deve ser

pequena, apenas o suficiente para absorver a granularidade do escalonamento, sem permitir rajadas que a medição lenta não veria a tempo.

O *token bucket* resolve o problema de *impor* uma taxa, mas não o de *escolhê-la*. A escolha é feita por um operador humano, com base em testes ou experiência, e fica fixa na configuração. Em termos de controle, trata-se de uma atuação em malha aberta: a decisão de quanto consumir não usa nenhuma informação sobre o estado atual do serviço que recebe o trabalho. E o valor escolhido envelhece: o limite calibrado em um teste de carga vale para o estado do sistema naquele dia, com aquele volume de dados, aquela composição de tráfego e aquele *hardware*. Qualquer um desses fatores muda sem aviso, e a prática usual é recalibrar apenas depois do incidente que revela a defasagem.

As consequências práticas são conhecidas de quem opera esses sistemas. Se as mensagens ficam mais custosas, o limite que era seguro passa a saturar o serviço; se uma carga externa ocupa o banco, o limite fixo não recua; e nos períodos de folga, o limite conservador desperdiça capacidade que poderia encurtar a tarefa. Qualquer valor fixo é um compromisso ruim entre segurança e vazão. A alternativa natural é fechar a malha: medir a carga do serviço de destino e ajustar a taxa continuamente, que é o objeto das seções e capítulos seguintes.

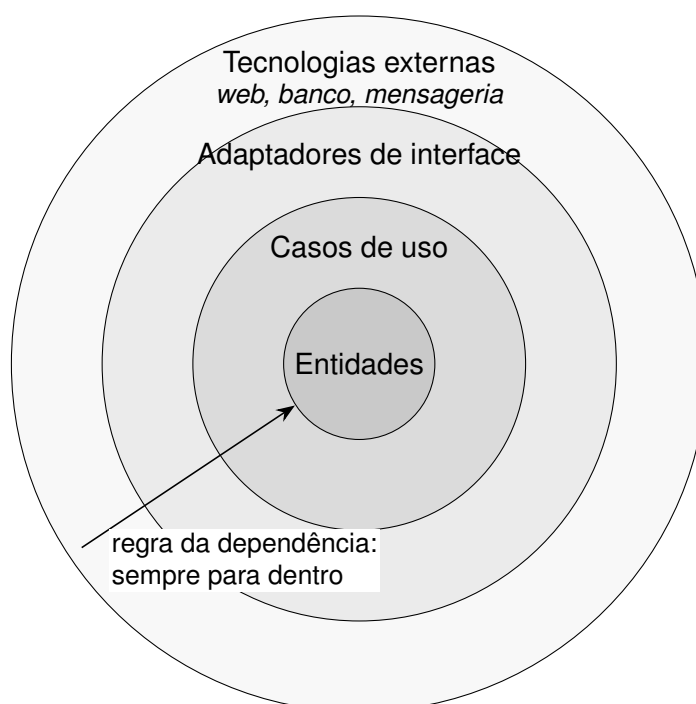
2.5 ARQUITETURA DE SOFTWARE: DOMÍNIO ISOLADO E INFRAESTRUTURA PLUGÁVEL

A construção de um sistema que precisa integrar-se com tecnologias intercambiáveis (diferentes filas, bancos e fontes de métricas) pede uma arquitetura que separe a lógica central dos detalhes de integração. Três correntes de projeto sustentam essa separação. O *Domain-Driven Design* (DDD) propõe organizar o código em torno do modelo do domínio, mantendo as regras de negócio explícitas e isoladas (EVANS, 2003). A *Clean Architecture* generaliza o princípio na regra da dependência: o código de domínio não depende de código de infraestrutura; as dependências apontam sempre para dentro (MARTIN, 2017). A arquitetura *Ports and Adapters* (hexagonal) materializa a regra: o núcleo define interfaces (*ports*) para tudo o que precisa do mundo externo, e cada tecnologia é um adaptador (*adapter*) que implementa uma dessas interfaces (COCKBURN, 2005).

A regra da dependência ordena o interior do núcleo em camadas concêntricas. No centro ficam as entidades, os tipos que representam os conceitos do domínio e que nada sabem sobre o mundo externo. Ao redor delas, os casos de uso orquestram as entidades para cumprir as ações que o sistema oferece, falando com o exterior apenas pelas interfaces das portas. Na borda, os adaptadores traduzem entre o vocabulário do domínio e o de cada tecnologia concreta (MARTIN, 2017). A Figura 4 ilustra essa organização: as setas de dependência atravessam as camadas sempre de fora para

dentro, de modo que o mundo externo (interfaces *web*, bancos de dados, dispositivos) conhece o núcleo, mas o núcleo não conhece o mundo externo. A consequência estrutural é que uma mudança em qualquer tecnologia externa fica confinada ao seu adaptador, e uma mudança de regra de negócio fica confinada ao núcleo, sem que uma contamine a outra.

Figura 4 – Camadas concêntricas da *Clean Architecture*: entidades no centro, casos de uso ao redor, adaptadores de interface na camada seguinte e as tecnologias externas na borda; a seta radial marca o sentido único da dependência, sempre para dentro.



Fonte: Elaborada pelo autor (2026), com base em Martin (2012).

Cockburn (2005) distingue ainda dois papéis de porta. As portas condutoras (*driving*) são aquelas pelas quais o mundo externo aciona o núcleo, como uma API ou uma interface de usuário; as portas conduzidas (*driven*) são aquelas pelas quais o núcleo aciona o mundo externo, como bancos de dados e serviços. A distinção é útil porque os dois lados evoluem por pressões diferentes: as portas condutoras acompanham quem usa o sistema, e as conduzidas acompanham a infraestrutura disponível. Um sistema que pretende operar em ambientes heterogêneos concentra a variabilidade nas portas conduzidas, que é o caso das filas, dos bancos e das fontes de métricas deste trabalho.

O efeito prático é que trocar a tecnologia de fila, o banco de dados ou o provedor de métricas não altera o núcleo do sistema: escreve-se um novo adaptador e liga-o à interface existente. A ligação entre interfaces e implementações concretas é feita por

injeção de dependência, idealmente concentrada em um único ponto de composição do programa (*composition root*). Além da flexibilidade, o isolamento torna o núcleo testável sem infraestrutura: nos testes, os adaptadores reais são substituídos por implementações simuladas. Essa arquitetura tem um custo conhecido, a indireção: cada porta é uma camada a mais entre a intenção e a execução, e para sistemas que nunca trocarão de tecnologia o investimento pode não se pagar. Ela se justifica quando a intercambialidade é requisito, e não especulação; neste trabalho, ela é um requisito explícito, formalizado no Capítulo 3.

Para este trabalho, essa arquitetura é parte do argumento: um mecanismo de controle de consumo só é útil na prática se puder ser acoplado às tecnologias que cada organização já usa. O Capítulo 4 mostra como esses princípios se concretizam no sistema desenvolvido.

2.6 FUNDAMENTOS DE CONTROLE

Levar uma variável de interesse de um sistema dinâmico a um valor desejado, manipulando uma entrada e observando a saída, é o problema central da teoria de controle, ainda que não o único. A aplicação dessas ferramentas a sistemas computacionais, com recursos como CPU, memória e filas tratados como processos dinâmicos, é uma linha consolidada na literatura (HELLERSTEIN *et al.*, 2004), e é sob essa perspectiva que esta seção apresenta os conceitos e as equações usadas no restante do trabalho.

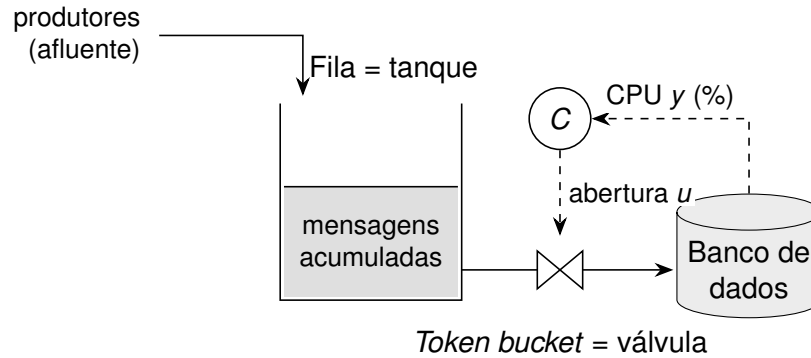
Antes do formalismo, convém apresentar uma analogia que orienta toda a modelagem deste trabalho. Considere o clássico problema de controle de nível ilustrado na Figura 5: um tanque recebe um afluente com vazão variável e tem, na saída, uma válvula comandada por um controlador. No sistema aqui estudado, a fila de mensagens faz o papel do tanque, acumulando o que chega; o *token bucket* faz o papel da válvula, pois sua taxa r regula a vazão de saída; e a variável controlada é a utilização de CPU do banco de dados que recebe o fluxo. A diferença é que não se controla o nível do tanque, e sim o esforço do processo que recebe a vazão. Vista por essa lente, a malha de consumo de uma fila é um problema de controle de processos convencional.

2.6.1 O modelo FOPDT e sua identificação

Processos estáveis e monotônicos, como o deste trabalho, são bem descritos por um modelo de primeira ordem com atraso de transporte (*First Order Plus Dead Time*, FOPDT), cuja função de transferência é

$$G(s) = \frac{K e^{-\theta s}}{\tau s + 1}, \quad (1)$$

Figura 5 – Analogia hidráulica do sistema: a fila corresponde ao tanque, o *token bucket* à válvula de saída e a utilização de CPU do banco é a variável controlada.



Fonte: Elaborada pelo autor (2026).

em que K é o ganho estático (quanto a saída varia, em regime, por unidade de entrada), τ é a constante de tempo (relacionada à rapidez da resposta) e θ é o atraso de transporte (tempo até a saída começar a reagir) (ÅSTRÖM; HÄGGLUND, 2006; AGUIRRE, 2015).

Os três parâmetros são identificáveis por um ensaio de resposta ao degrau em malha aberta. Neste trabalho adota-se o método dos dois pontos: sobre a curva de reação, registram-se os instantes t_{28} e t_{63} em que a saída atinge 28,3 % e 63,2 % da variação total, e os parâmetros são estimados por

$$\tau = 1,5 (t_{63} - t_{28}), \quad \theta = t_{63} - \tau, \quad K = \frac{\Delta y}{\Delta u}, \quad (2)$$

em que Δy e Δu são as variações de regime da saída e da entrada (AGUIRRE, 2015). A estimativa de (2) serve de ponto de partida para um refinamento numérico por mínimos quadrados sobre a curva completa, procedimento aplicado no Capítulo 5.

2.6.2 Discretização e o papel da amostragem

Quando a medição só está disponível a cada período de amostragem T , o controlador é projetado em tempo discreto (FRANKLIN; POWELL; WORKMAN, 1998). Com segurador de ordem zero na entrada, o modelo (1) discretizado é

$$G(z) = z^{-d} \frac{K(1-a)}{z-a}, \quad a = e^{-T/\tau}, \quad d = \text{round}\left(\frac{\theta}{T}\right), \quad (3)$$

em que a é o polo discreto, d é o atraso expresso em número inteiro de amostras e $\text{round}(\cdot)$ denota o arredondamento ao inteiro mais próximo (FRANKLIN; POWELL; WORKMAN, 1998). A relação $d = \text{round}(\theta/T)$ evidencia o limite estrutural que domina este trabalho: o efeito de qualquer ação de controle, e de qualquer perturbação, só se torna observável após d amostras, e nenhum ajuste de ganhos elimina essa espera.

Se T é muito maior que τ , a planta acomoda dentro de uma única amostra ($a \rightarrow 0$) e o processo visto pelo controlador degenera em um ganho estático com uma amostra de atraso. Nesse regime, o desempenho alcançável é limitado pela amostragem, e não pela planta; a agressividade dos ganhos apenas converte atraso em oscilação (ÅSTRÖM; HÄGGLUND, 2006).

2.6.3 O controlador PI, o *anti-windup* e a sintonia SIMC

O controlador empregado é o PID em sua forma reduzida PI. O termo derivativo é omitido de propósito: em um processo de medição ruidosa e amostragem lenta como o deste trabalho, a derivada do erro amplifica o ruído da medição sem benefício prático sobre a resposta (ÅSTRÖM; HÄGGLUND, 2006). Na forma posicional em tempo discreto, com período T , a lei de controle é

$$e_k = r_k - y_k, \quad (4)$$

$$I_k = \text{sat}(I_{k-1} + K_i e_k T), \quad (5)$$

$$u_k = \text{sat}(K_p e_k + I_k, u_{\min}, u_{\max}), \quad (6)$$

em que r_k é a referência, y_k é a medição, K_p e K_i são os ganhos proporcional e integral, e a saturação de u_k reflete os limites físicos da taxa de drenagem. A saturação do termo integral em (5) implementa o *anti-windup* na sua forma condicional: além de um teto absoluto configurável, I_k é limitado, a cada passo, à faixa que mantém a soma dos demais termos da saída dentro de $[u_{\min}, u_{\max}]$. Assim, o integrador para de acumular no instante em que o atuador satura, e não apenas quando o próprio termo integral atinge o teto, o que evita o sobressinal causado pelo acúmulo durante grandes erros transitórios (ÅSTRÖM; HÄGGLUND, 2006).

Os ganhos são obtidos do modelo identificado pela regra SIMC (*Simple Internal Model Control*) de Skogestad para plantas FOPDT, que expõe um único parâmetro de projeto, a constante de tempo desejada em malha fechada τ_c :

$$K_c = \frac{1}{K} \cdot \frac{\tau}{\tau_c + \theta}, \quad \tau_I = \min(\tau, 4(\tau_c + \theta)), \quad (7)$$

em que K_c é o ganho do controlador e τ_I é o tempo integral (SKOGESTAD, 2003). Para a forma posicional (4)–(6), o mapeamento é direto: $K_p = K_c$ e $K_i = K_c/\tau_I$. Quanto menor τ_c , mais rápida (e menos robusta) a malha; Skogestad (2003) recomenda $\tau_c = \theta$ como o compromisso robusto de referência, escolha adotada neste trabalho com o atraso efetivo θ_{ef} no papel de θ (Seção 5.5).

2.6.4 O Preditor de Smith

Quando o atraso θ é comparável à constante de tempo τ ou maior, o desempenho do PI é limitado pelo atraso, e a estrutura clássica para contorná-lo é o Preditor de

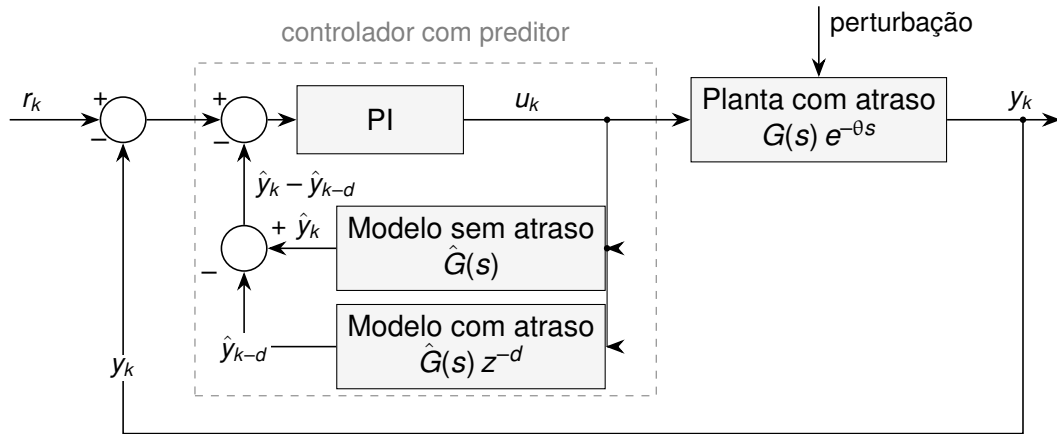
Smith (SMITH, 1957), ilustrado na Figura 6: o controlador incorpora um modelo interno da planta $\hat{G}(s) = \hat{K}e^{-\hat{\theta}s}/(\hat{\tau}s + 1)$ e passa a realimentar uma predição sem atraso da saída,

$$y_k^p = y_k + (\hat{y}_k - \hat{y}_{k-d}), \quad (8)$$

em que $\hat{y}_k$ é a resposta do modelo *sem* o atraso à entrada aplicada e $\hat{y}_{k-d}$ é essa mesma resposta atrasada de d amostras. O modelo sem atraso é propagado recursivamente pela discretização de primeira ordem

$$\hat{y}_k = a\hat{y}_{k-1} + K(1-a)u_{k-1}, \quad a = e^{-T/\tau}. \quad (9)$$

Figura 6 – Estrutura do Preditor de Smith: o PI opera sobre o erro predito, com o modelo interno sem atraso em paralelo com a planta real.



Fonte: Adaptada de Maia (1993).

O controlador PI opera então sobre o erro predito $e_k = r_k - y_k^p$, usando (5) e (6). Quando o modelo coincide com a planta, o termo $y_k - \hat{y}_{k-d}$ se anula e o PI enxerga a planta sem atraso, o que admite ganhos mais agressivos; sob descasamento de modelo, porém, o termo de correção reintroduz dinâmica não compensada, e a estabilidade degrada. Essa sensibilidade a erros de modelo é a limitação conhecida da estrutura (NORMEY-RICO; CAMACHO, 2007), e sua manifestação prática é avaliada experimentalmente no Capítulo 5.

2.6.5 Avaliação da resposta temporal

A comparação entre controladores usa as métricas clássicas da resposta temporal: sobressinal, tempos característicos (subida e acomodação) e, na rejeição de perturbação, pico de desvio e tempo de recuperação (OGATA, 2010). O Capítulo 4 e o Capítulo 5 aplicam essas ferramentas ao projeto dos controladores e aos experimentos.

2.7 COMO SE MEDE A CPU: INSTRUMENTAÇÃO COMO SENSOR DA MALHA

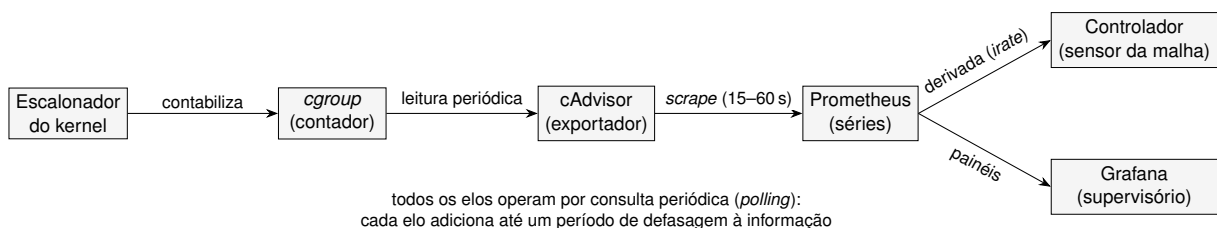
Em uma malha de controle sobre *software*, o papel do sensor é desempenhado pela infraestrutura de métricas. Sistemas como o Prometheus coletam periodicamente contadores expostos pelos serviços e os armazenam como séries temporais (PROMETHEUS, 2026); provedores de nuvem oferecem serviços equivalentes, como o AWS CloudWatch.

A utilização de CPU, em particular, não existe como grandeza diretamente legível: o núcleo do sistema operacional não mantém um número “CPU = 37 %” à disposição. O que o escalonador do Linux contabiliza é o tempo acumulado que cada processo efetivamente ocupou um núcleo, registrado nos *cgroups*, o mesmo mecanismo que isola contêineres. O coletor cAdvisor (GOOGLE, 2026) lê periodicamente esse registro e o expõe como um contador monotônico de segundos de CPU consumidos, que o Prometheus armazena. A Figura 7 resume a cadeia e seus consumidores. Em nenhum elo (escalonador, *cgroup*, coletor, base de séries temporais) existe um percentual: a utilização nasce apenas ao derivar o contador, tomando dois valores $C(t_1)$ e $C(t_2)$ e calculando

$$\bar{y} = \frac{C(t_2) - C(t_1)}{t_2 - t_1}, \quad (10)$$

que expressa a fração de núcleo consumida no intervalo. A analogia é a do hodômetro e do velocímetro: o contador acumula quilômetros; a velocidade é a sua derivada, e é ela que se controla. A variável de processo realimentada pela malha deste trabalho é exatamente a derivada (10); realimentar o contador cru não teria sentido, pois o erro crescería indefinidamente. Segue daí que a forma de derivar importa: uma derivada sobre os dois pontos mais recentes é reativa, porém ruidosa; uma taxa média sobre uma janela longa é suave, porém atrasada. Essa escolha é parte do projeto do sensor, e seus efeitos sobre a malha são avaliados experimentalmente no Capítulo 5.

Figura 7 – A cadeia de medição de CPU, do escalonador do kernel aos dois consumidores: a malha de controle e o supervisor leem a mesma fonte.



Fonte: Elaborada pelo autor (2026).

Essa cadeia opera, em todos os elos, por consulta periódica (*polling*): o coletor lê o *cgroup* em intervalos regulares, o Prometheus raspa cada alvo no seu intervalo de

coleta, tipicamente entre 15 e 60 segundos, o controlador consulta a base de métricas em ciclo próprio, e os provedores de nuvem publicam pontos em períodos fixos, da ordem de um minuto ou mais, com latência adicional até a disponibilização. O modelo de consulta periódica é o que permite a uma infraestrutura de monitoração observar milhares de alvos de forma desacoplada e barata, e por isso é o modelo adotado pelas principais plataformas de monitoração (PROMETHEUS, 2026; KUBERNETES, 2026). A consequência, porém, é estrutural: cada elo adiciona até um período de defasagem, e a leitura mais recente disponível já nasceu velha; no pior caso, a idade da informação é a soma dos períodos de todos os elos.

Para a malha de controle, essa defasagem entra exatamente como o atraso de transporte θ da Seção 2.6, com um agravante: ela não pode ser eliminada por projeto. O período de publicação do provedor de métricas é um piso que nenhuma configuração do controlador remove. E como o processo controlado neste trabalho responde em segundos, enquanto o sensor responde em dezenas de segundos ou minutos, é o sensor, e não a planta, que dita a dinâmica alcançável da malha. Essa inversão, o atraso dominante vindo da medição e não do processo, é o fator principal de todo o projeto dos controladores deste trabalho, e sua confirmação experimental é um dos resultados centrais do Capítulo 5.

Resta a suavização, a segunda propriedade do sensor: derivar taxas sobre janelas longas equivale a aplicar uma média móvel, que reduz ruído mas adiciona atraso da ordem de metade da janela. Filtragem, portanto, não é gratuita; o compromisso entre ruído e atraso é uma decisão de projeto da malha. Neste trabalho, o compromisso é tratado como decisão explícita de projeto do sensor, retomada na descrição do leitor de métricas (Seção 4.4).

A mesma cadeia de medição sustenta os mecanismos de controle mais difundidos da infraestrutura de produção. O escalonador automático horizontal do Kubernetes (*Horizontal Pod Autoscaler*, HPA) obtém a CPU de cada réplica pelo mesmo caminho, um *cAdvisor* embutido no agente do nó lendo o mesmo registro de *cgroup*, e o servidor de métricas deriva o contador entre duas amostras recentes, tal como em (10). Sobre essa utilização média $\bar{y}$, o HPA aplica a lei de ajuste

$$n_{\text{des}} = \left\lceil n_{\text{atual}} \cdot \frac{\bar{y}_{\text{atual}}}{\bar{y}_{\text{alvo}}} \right\rceil, \quad (11)$$

na qual n é o número de réplicas, reavaliada em ciclos periódicos com banda de tolerância e janelas de estabilização contra oscilação (KUBERNETES, 2026). A Equação (11) é, em essência, um controlador proporcional discreto com saturação, operando sobre a derivada de um contador de CPU. O presente trabalho estuda, sobre a mesma grandeza e a mesma cadeia de medição, estruturas de controle mais ricas: ação integral, *anti-windup* e compensação de atraso. Os fenômenos investigados nos experimentos (ruído da derivada de dois pontos, atraso de amostragem e variabilidade entre execu-

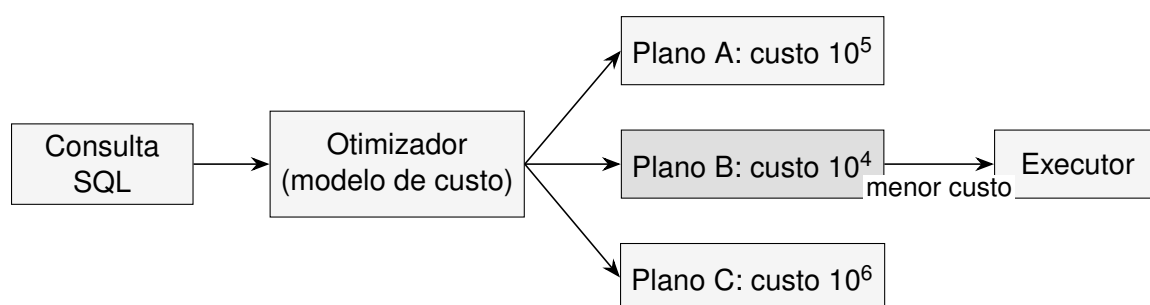
ções) são os mesmos que qualquer escalonador automático de produção enfrenta.

2.8 PREDIÇÃO DE CUSTO DE CONSULTAS E SUAS LIMITAÇÕES

A seção anterior mostrou que medir a carga presente é viável, desde que se respeitem as propriedades do sensor. Prever a carga que uma operação ainda não executada vai causar é um problema de outra natureza, e esta seção examina o porquê.

Bancos de dados mantêm, dentro do otimizador de consultas, um modelo de custo cuja função é escolher planos de execução: como ilustra a Figura 8, cada plano candidato recebe um custo estimado, e o de menor custo é executado. No MySQL, essa estimativa é o *Query Cost*, expresso em unidades abstratas que combinam acessos a disco e trabalho de CPU; o componente de CPU é contabilizado por constantes como o custo de avaliar uma linha, configuráveis nas tabelas de custo do servidor (ORACLE, 2026). É importante entender o que esse número é e o que ele não é: ele foi projetado para *comparar planos alternativos de uma mesma consulta*, tarefa em que erros sistemáticos se cancelam. Seu valor absoluto não é uma predição de tempo de execução nem de consumo de CPU.

Figura 8 – Papel do modelo de custo do otimizador: os planos candidatos de uma mesma consulta recebem custos estimados em unidades abstratas e o de menor custo é executado; o número serve para comparar planos, não para prever tempo ou consumo.



Fonte: Elaborada pelo autor (2026), com base em Oracle (2026).

A literatura que tentou usá-lo como predição chegou a conclusões consistentes. Wu *et al.* (2013) mostram que a abordagem ingênua, escalar o custo do otimizador por uma constante, falha; uma predição razoável exige calibrar empiricamente as unidades de custo no sistema concreto, executando cargas de calibração e medindo o *hardware* real, e mesmo o modelo calibrado degrada na presença de erros de estimativa de cardinalidade. Li *et al.* (2012) e Ganapathi *et al.* (2009) atacam o problema por outro flanco: abandonam as funções analíticas do otimizador e treinam modelos estatísticos sobre execuções observadas do próprio sistema, evidência de que os efeitos reais (ar-

quietura de *hardware*, *caches*, casos especiais do executor) não cabem nas fórmulas manuais.

Há ainda o fator tempo, que agrava qualquer das abordagens: o custo real de uma mesma consulta muda com o volume e a distribuição dos dados, com o estado das *caches*, com o *hardware* e com a carga concorrente. Um preditor calibrado hoje degrada amanhã, e mantê-lo fiel exige recalibração ou retreinamento contínuos. A predição de custo por operação é, em síntese, um problema aberto, caro de resolver e de manter.

Para este trabalho, a lição é dupla. Primeiro, o custo estimado tem um papel legítimo como *caracterização*: ordenar cenários por custo e documentar as condições de um experimento, uso que o Capítulo 5 faz sistematicamente. Segundo, como base para decidir o ritmo de consumo em tempo real, a predição por operação é frágil, e é essa fragilidade que a realimentação dispensa, como o próximo capítulo formaliza.

Fechado o panorama conceitual, o próximo capítulo retoma o problema específico deste trabalho e o formaliza como uma malha de controle, definindo requisitos técnicos para a solução.

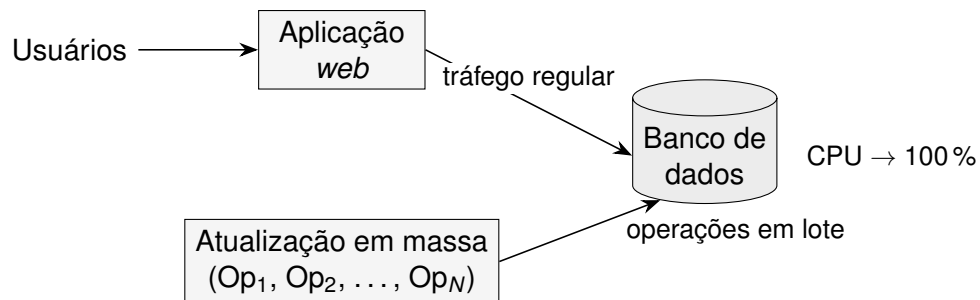
3 DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS

Este capítulo retoma o problema apresentado na Introdução e o aprofunda tecnicamente. A Seção 3.1 caracteriza o cenário de saturação e suas fontes de variabilidade. A Seção 3.2 formaliza o sistema como uma malha de controle em tempo discreto, atribuindo os papéis de planta, variável de processo, atuador e perturbação. As Seções 3.3 e 3.4 enumeram os requisitos técnicos, funcionais e não funcionais, que a solução deve atender.

3.1 O PROBLEMA FORMALIZADO

O cenário de interesse é o de uma aplicação *web* em operação normal que passa a dividir seu banco de dados com um fluxo de atualização em massa. A Figura 9 ilustra a situação: além do tráfego regular dos usuários, um processo em lote (uma migração de dados, um reprocessamento, uma carga de correção) executa uma sequência de operações $Op_1, Op_2, \dots, Op_N$ sobre o mesmo banco. As duas cargas se somam no único recurso compartilhado, e nada no caminho do lote considera o estado do banco: é um fluxo em malha aberta.

Figura 9 – Fluxo de atualização em massa disputando o mesmo banco de dados da aplicação: as cargas se somam sem qualquer regulação.



Fonte: Elaborada pelo autor (2026).

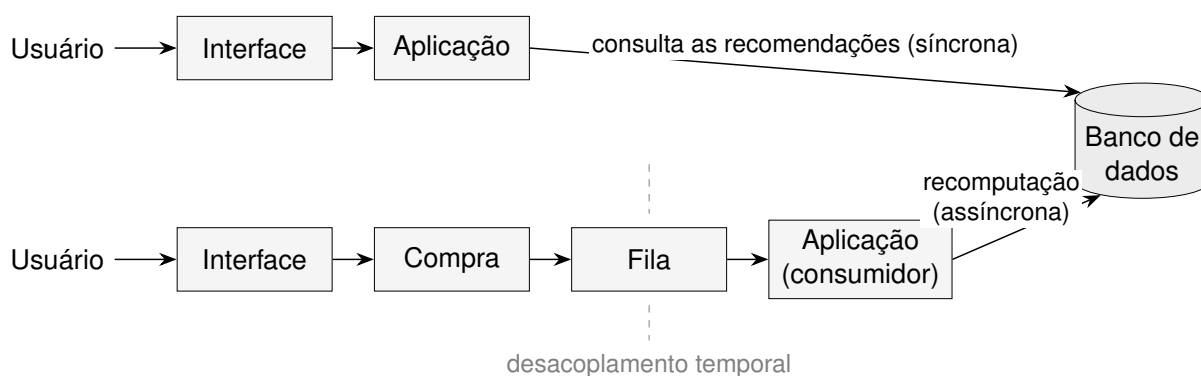
Quando a soma das cargas excede a capacidade de processamento do banco, a utilização de CPU atinge o limite e a degradação descrita na Seção 2.2 se instala: latência crescente, esgotamento de tempo nas consultas e impacto direto no usuário da aplicação. O objetivo operacional é executar o lote na maior velocidade que *não* provoque esse quadro, mantendo a utilização de CPU do banco em um patamar seguro, por exemplo 50 %, que preserve folga para o tráfego regular.

Um exemplo concreto ilustra o cenário. Considere um *marketplace* cujo serviço de recomendações sugere produtos a cada usuário com base no histórico de compras e de visitas. Sempre que um usuário abre o site, uma requisição consulta no banco as recomendações previamente calculadas: é tráfego síncrono, sensível à latência e

de volume difícil de prever. Cada compra, por sua vez, exige recalcular as pontuações dos produtos envolvidos, um processamento custoso executado no próprio banco. Em uma temporada de promoções, as compras se multiplicam e, com elas, as escritas e as recomputações. Sem regulação, a CPU do banco satura e o efeito aparece onde mais custa ao negócio: os usuários passam a encontrar erros e lentidão na vitrine de recomendações, consequência direta da CPU elevada do banco.

O exemplo expõe a assimetria que a solução explora. Os dois tráfegos que disputam o banco têm requisitos opostos. A leitura das recomendações é síncrona e não pode esperar; o processamento de uma nova compra pode ser adiado em segundos ou minutos sem impacto perceptível ao negócio, pois a recomendação apenas demora um pouco mais para refletir a compra recente. É essa tolerância a atraso que torna o fluxo controlável: enfileirando as recomputações e regulando seu ritmo pela CPU medida do banco, as promoções passam a acelerar o crescimento da fila, e não a carga do banco, enquanto a vitrine continua respondendo. Esse é o critério geral de elegibilidade para o caminho controlado da solução: fluxos cuja conclusão tolera atraso, trocado pela estabilidade do serviço compartilhado. A Figura 10 contrasta os dois fluxos: a consulta das recomendações percorre o caminho síncrono até o banco, enquanto a compra publica a recomputação em uma fila, que desacopla o momento da compra do momento do processamento; os dois caminhos, porém, terminam no mesmo banco, e é ali que disputam CPU.

Figura 10 – Os dois fluxos do exemplo do *marketplace*: a consulta das recomendações é síncrona e sensível à latência; a recomputação disparada pela compra é desacoplada pela fila e tolera atraso. Ambos convergem para o mesmo banco de dados.



Fonte: Elaborada pelo autor (2026).

Duas características tornam o problema genuinamente dinâmico. A primeira é a variabilidade do custo por operação entre tarefas: o esforço que uma mensagem impõe ao banco depende do tipo de comando (uma busca por índice, uma varredura completa, uma junção, uma alteração de esquema) e do estado da tabela no momento,

como o volume de linhas e a distribuição dos índices. Dentro de uma mesma tarefa, porém, o custo tende a ser homogêneo: migrações e reprocessamentos tipicamente repetem uma mesma consulta, variando apenas os parâmetros, com custo semelhante entre operações. É esse o cenário assumido neste trabalho: uma fila cujo custo por operação é razoavelmente uniforme, com o ganho da malha definido para essa classe de custo. Filas que intercalam operações de custos muito distintos ficam fora do escopo, e o caminho para tratá-las é discutido como extensão no Capítulo 6. A segunda característica é a perturbação externa: o tráfego regular da aplicação varia ao longo do tempo e consome uma fração imprevisível da capacidade do banco. Qualquer taxa fixa escolhida para o lote está, portanto, errada na maior parte do tempo, para mais ou para menos.

Em termos de controle, o custo por operação é o próprio ganho da planta: a CPU de regime é o produto da taxa de consumo pelo custo médio de cada operação executada. Os experimentos do Capítulo 5 quantificam essa sensibilidade para uma única consulta de varredura completa. Sobre uma tabela de cem mil linhas, o otimizador do MySQL estima para ela um custo (o *Query Cost*, Seção 2.8) de aproximadamente 10 mil, e o ganho identificado da planta é de cerca de 2,3 % de CPU por mensagem por segundo; sobre um milhão de linhas, o custo estimado da mesma consulta sobe para cerca de 103 mil, e poucas mensagens por segundo bastam para saturar o banco. A mesma consulta, dez vezes mais custosa por causa do estado da tabela, transforma a planta em outra. Uma busca por índice, no extremo oposto, custa ordens de grandeza menos que a varredura.

Prever esse custo por operação, como discutido na Seção 2.8, é um problema aberto: exige calibração empírica no sistema concreto, como mostram Wu *et al.* (2013), e mesmo a calibração se degrada com o tempo. A abordagem deste trabalho contorna a predição: em vez de prever o custo de cada mensagem antes de executá-la, mede o efeito agregado da execução sobre o serviço e reage a ele. Uma predição precisa estar certa antes de agir; a realimentação tolera estar errada e se corrige depois, o que a torna estruturalmente imune aos erros que degradam os preditores. O princípio recomendado por esses autores é, ainda assim, plenamente atendido: a etapa de identificação do Capítulo 5 cumpre, na prática, o papel dessa calibração empírica, medindo no sistema real quanto de CPU cada mensagem por segundo de fato produz. A diferença é o uso que se faz da medida: em vez de alimentar um preditor por operação, ela dimensiona os ganhos de um controlador que permanece válido quando o sistema muda. A carga efetivamente medida é o sinal da decisão de ritmo, e o custo estimado fica com o papel que cumpre bem, o de metadado de caracterização e reprodutibilidade dos cenários.

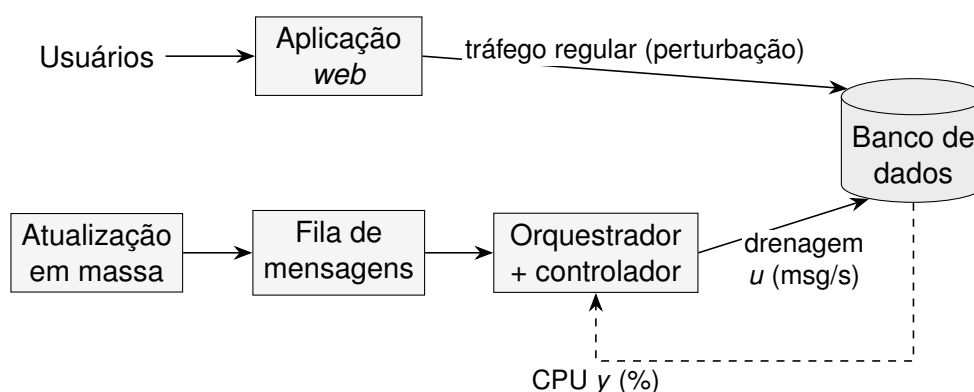
O efeito das camadas de *cache* ilustra bem por que o custo de uma operação não pode ser previsto olhando apenas para ela. Bancos de dados e aplicações mantêm

em memória os resultados e as páginas de dados mais acessados, e é essa memória que sustenta a maior parte das leituras baratas. Uma única requisição de escrita, aparentemente inofensiva, pode alterar um dado quente, invalidar as entradas de *cache* que dependiam dele e forçar as leituras seguintes a recomputar tudo contra o disco, com aumento expressivo e não linear de CPU. O custo real de uma operação depende, assim, do estado interno do serviço no instante em que ela executa, estado que o produtor da mensagem não conhece. Essa é mais uma razão para medir a carga efetiva do serviço em vez de tentar estimá-la mensagem a mensagem.

3.2 MODELAGEM COMO MALHA DE CONTROLE

A solução proposta interpõe uma fila de mensagens entre o produtor do lote e o banco, e regula a drenagem dessa fila em malha fechada. A Figura 11 apresenta a estrutura: as operações do lote entram na fila; um orquestrador as consome e as entrega ao banco a uma taxa u (mensagens por segundo) comandada por um controlador; a utilização de CPU do banco, y (%), é medida e realimentada ao controlador, fechando a malha. O tráfego regular da aplicação continua chegando ao banco por fora do caminho controlado, como uma perturbação de carga.

Figura 11 – Estrutura da solução: a fila absorve o lote, o controlador regula a drenagem e a realimentação da CPU do banco fecha a malha; o tráfego regular atua como perturbação.



Fonte: Elaborada pelo autor (2026).

Na terminologia da analogia hidráulica da Seção 2.6 (Figura 5), os papéis ficam assim atribuídos:

- planta**: o banco de dados, cujo esforço de processamento responde à vazão de operações recebida;
- variável de processo y** : a utilização de CPU do banco (%), obtida da infraestrutura de métricas;

- c) **variável manipulada** u : a taxa de drenagem da fila (msg/s), imposta pelo *token bucket* do orquestrador, que faz o papel da válvula;
- d) **referência** r : o patamar de CPU desejado (*setpoint*);
- e) **perturbações**: o tráfego regular da aplicação e as variações de custo por operação;
- f) **sensor**: o provedor de métricas, com seu período de amostragem T e seu atraso de medição.

A escolha da utilização de CPU como variável de processo merece justificativa, pois a carga de um banco de dados não é função apenas das consultas que ele executa. O número de conexões abertas tem custo próprio: no MySQL, cada conexão é atendida por uma *thread* do servidor, que consome memória e tempo de escalonamento, e um número de execuções simultâneas muito superior ao de núcleos multiplica as trocas de contexto (BOTROS; TINLEY, 2021). A pressão de memória também vira carga: quando a área de *cache* do banco (*buffer pool*) é insuficiente, leituras que seriam atendidas em memória tornam-se acessos a disco, com trabalho adicional de entrada e saída (E/S) e de processamento. Somam-se tarefas de fundo do próprio banco, como limpeza de versões antigas, *checkpoints* e replicação. A CPU foi escolhida como variável de processo por três razões: é o recurso em que boa parte desses efeitos converge, pois conexões, faltas de *cache* e tarefas de fundo acabam refletidas em ciclos de processamento; é a métrica mais universalmente disponível nos provedores de monitoração, o que preserva a portabilidade da solução; e, no cenário estudado, de recomputações intensivas, é o recurso que satura primeiro. As demais dimensões (conexões, memória, disco) permanecem fora da malha neste trabalho; a porta de leitura de métricas permite trocá-las ou combiná-las como variável de processo sem alterar o restante do sistema, extensão registrada no Capítulo 6.

A fila cumpre na malha o papel do tanque: absorve a diferença entre o que o produtor gera e o que o controlador permite drenar. Essa é a razão de a contrapressão ser viável sem perda de dados: reduzir u não descarta mensagens, apenas as retém na fila, cujo acúmulo (*lag*) é observável e finito. O preço da segurança é pago em tempo de conclusão do lote, não em integridade.

Sobre o atuador, convém desfazer uma confusão comum. O *token bucket* não exige atribuir a cada mensagem um custo de CPU: cada ficha é a permissão para entregar uma mensagem, e o controlador manipula apenas a taxa de reposição de fichas. A relação entre mensagens por segundo e CPU fica implícita na realimentação. Se as operações da fila ficarem mais custosas, a CPU sobe, o erro muda de sinal e o controlador reduz a taxa, sem que custo algum tenha sido declarado por mensagem. O ganho K da planta só é usado fora do caminho de execução: no projeto dos ganhos do controlador e nos componentes opcionais que dependem de modelo (a ação *feed-*

forward e o modelo interno do Preditor de Smith). Se o custo médio real se deslocar de forma moderada, a malha continua regulando, pois a ação integral encontra o novo ponto de equilíbrio; o que se altera é a qualidade da resposta transitória, projetada para o ganho identificado. Essa tolerância tem limite: uma fila que intercale operações de custos muito distintos (uma consulta pesada em meio a consultas leves) muda o ganho instantâneo da planta a cada mensagem, e a regulação perde qualidade. Para custos que derivam lentamente, a extensão natural é a adaptação contínua de ganhos; para cargas heterogêneas, a separação das operações por classe de custo em múltiplas filas, cada qual com seu peso. Ambas são discutidas no Capítulo 6.

Por fim, é preciso delimitar o que a malha *não* controla. O nível da fila não é regulado: assume-se que o produtor eventualmente para e que a fila comporta o acúmulo. O tempo de resposta do banco tampouco é a variável controlada, embora se beneficie indiretamente da limitação de CPU. Essas escolhas mantêm o problema unidimensional, uma entrada e uma saída, e são revisitadas como possíveis extensões no Capítulo 6.

3.3 REQUISITOS FUNCIONAIS

Do problema formalizado derivam os requisitos funcionais da solução:

- a) **regulação em malha fechada**: manter a utilização de CPU do serviço de destino em torno de um *setpoint* configurável, ajustando continuamente a taxa de drenagem da fila;
- b) **entrega sem perda**: cada mensagem deve ser confirmada na fila somente após o destino confirmar sua execução (sequência *dequeue* → *push* → *ack*), de modo que nenhum ajuste de taxa, por mais agressivo, descarte mensagens;
- c) **ajuste em tempo de execução**: os parâmetros do controlador (K_p , K_i , *setpoint*, saturações, filtro) devem poder ser lidos e alterados durante a operação, sem reinicialização;
- d) **estratégia de controle intercambiável**: o controlador deve ser acoplado ao núcleo por interface, como as demais dependências, de modo que a estratégia (PI, Preditor de Smith ou outra) seja selecionada na configuração de cada implantação sem alterar o restante do sistema. No ambiente de experimentação, adicionalmente, exige-se a troca da estratégia em tempo de execução e um modo de malha aberta com taxa constante, sem os quais os ensaios de identificação e a comparação entre controladores seriam impraticáveis;
- e) **infraestrutura plugável**: fila, serviço de destino e fonte de métricas devem ser acopláveis por interfaces, permitindo trocar as tecnologias sem alterar o núcleo do sistema.

3.4 REQUISITOS NÃO FUNCIONAIS

Complementam os funcionais os seguintes requisitos de qualidade:

- a) **observabilidade**: todas as variáveis da malha (variável de processo, saída do controlador, acúmulo da fila, parâmetros vigentes) devem ser expostas como métricas e visualizáveis em painel supervisorio em tempo real;
- b) **reprodutibilidade**: o ambiente completo de experimentação deve subir com um único comando, garantindo que os ensaios possam ser repetidos em condições idênticas;
- c) **segurança**: a comunicação com a aplicação e com o banco de dados deve ser criptografada (TLS), condição para uso em ambientes reais;
- d) **compatibilidade com produção**: as tecnologias suportadas devem incluir serviços gerenciados de nuvem, aproximando os experimentos das condições reais de operação.

O Quadro 1 consolida os requisitos e indica onde cada um é atendido nos capítulos seguintes.

Quadro 1 – Consolidação dos requisitos técnicos.

Código	Requisito	Atendimento
RF1	Regulação da CPU em malha fechada	Controlador (Seções 4.3 e 5.5)
RF2	Entrega sem perda de mensagens	Orquestrador (Seção 4.2)
RF3	Ajuste de parâmetros em tempo de execução	API REST (Seção 4.3)
RF4	Estratégia de controle intercambiável (porta); troca em execução e malha aberta no ambiente experimental	Seção 4.3
RF5	Fila, destino e métricas plugáveis	Adaptadores (Seção 4.5)
RNF1	Observabilidade em tempo real	Supervisorio (Seção 5.2)
RNF2	Reprodutibilidade dos experimentos	Infraestrutura (Seção 5.2)
RNF3	Comunicação criptografada	Infraestrutura (Seção 5.2)
RNF4	Compatibilidade com serviços de nuvem	Adaptadores (Seção 4.5)

Fonte: Elaborado pelo autor (2026).

Definidos o problema e os requisitos, o próximo capítulo apresenta a solução proposta e a metodologia adotada para construí-la e validá-la.

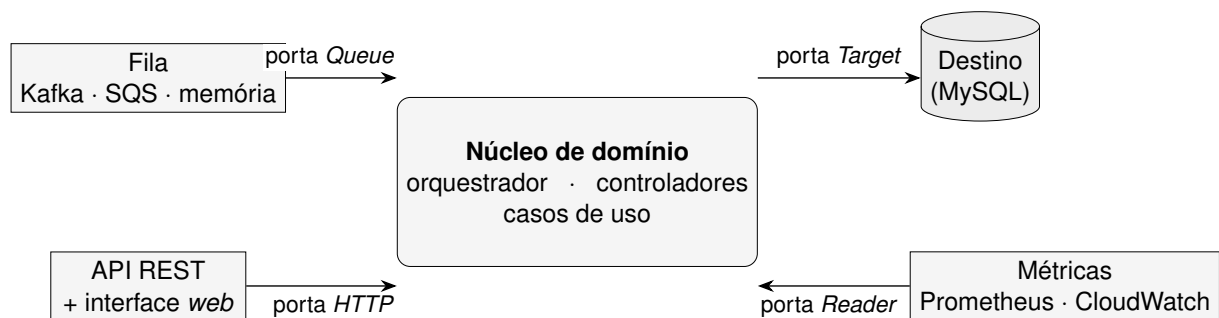
4 SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA

Este capítulo descreve a solução proposta e justifica as decisões de projeto que a levaram a atender os requisitos do Capítulo 3. A Seção 4.1 apresenta a arquitetura geral; as Seções 4.2 a 4.5 detalham os blocos da solução: o orquestrador, o controlador, o leitor de métricas e os adaptadores de fila e de destino. A Seção 4.6 fecha o capítulo com a metodologia experimental adotada para identificar a planta, sintonizar e validar os controladores. A implementação concreta desses blocos e o ambiente construído para os experimentos são apresentados no Capítulo 5.

4.1 VISÃO GERAL DA ARQUITETURA

O sistema foi implementado na linguagem Go segundo a arquitetura *Ports and Adapters* apresentada na Seção 2.5. A Figura 12 mostra a organização: no centro fica o núcleo de domínio, que contém o orquestrador, os controladores e os casos de uso; ao redor, cada tecnologia externa é acoplada por uma interface (porta) e implementada por um adaptador. A fila de mensagens, o serviço de destino e a fonte de métricas são todos trocáveis sem alteração do núcleo, o que atende diretamente ao requisito RF5.

Figura 12 – Arquitetura *Ports and Adapters* do sistema: o núcleo de domínio não conhece as tecnologias concretas, acopladas por interfaces.



Fonte: Elaborada pelo autor (2026).

Quatro portas estruturam o sistema. A porta *Queue* abstrai a fila de mensagens (operações de *dequeue* e *ack*); a porta *Target* abstrai o serviço de destino que executa as operações; a porta *Reader* abstrai a fonte da variável de processo; e a porta *HTTP* expõe o sistema a operadores e a outras aplicações, tanto para consulta de estado quanto para ajuste de parâmetros. A escolha da linguagem Go se deve à sua adequação a serviços concorrentes de longa duração, à disponibilidade de clientes maduros para as tecnologias envolvidas e à familiaridade prévia do autor com a linguagem, fator relevante para a produtividade do desenvolvimento.

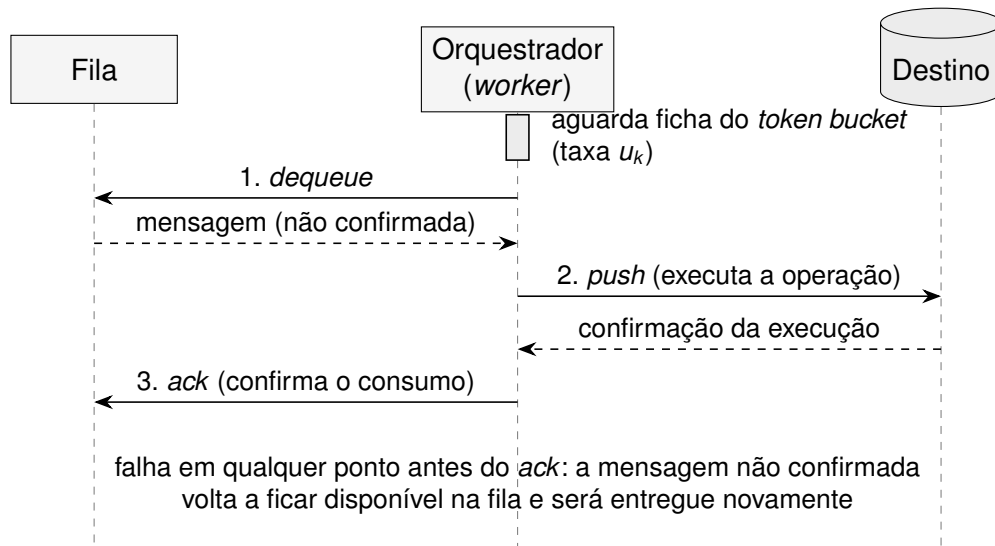
4.2 O ORQUESTRADOR

O orquestrador é o componente que executa o consumo regulado. Ele mantém um limitador do tipo *token bucket* (Seção 2.4) cuja taxa de reposição de fichas é comandada pelo controlador: a saída u_k , em mensagens por segundo, torna-se a vazão do balde. Trabalhadores concorrentes (*workers*) aguardam uma ficha e, ao obtê-la, executam a sequência de entrega de uma mensagem:

- a) *dequeue*: a mensagem é retirada da fila, sem confirmação;
- b) *push*: a operação é enviada ao serviço de destino e sua execução é aguardada;
- c) *ack*: somente após a confirmação do destino, a mensagem é confirmada na fila.

A Figura 13 ilustra o ciclo completo e a posição da garantia de entrega. A ordem é o que garante o requisito RF2, de entrega sem perda: se o processo falhar entre o *dequeue* e o *ack*, a mensagem não confirmada volta a ficar disponível na fila e será entregue novamente.

Figura 13 – Ciclo de entrega de uma mensagem pelo orquestrador: a confirmação na fila (*ack*) só ocorre após a confirmação do destino.



Fonte: Elaborada pelo autor (2026).

A propriedade vale inclusive sob ajustes agressivos de taxa, porque a redução de u apenas espaça a concessão de fichas; nenhuma mensagem em trânsito é descartada. O custo dessa garantia é a possibilidade de entrega duplicada em caso de falha, compromisso clássico de sistemas de mensageria que privilegiam a não perda (KLEPPMANN, 2017).

4.3 O CONTROLADOR PLUGÁVEL

O controlador é definido por uma interface única, com operações para calcular a saída a partir da medição (*Compute*), ler e alterar parâmetros e reiniciar o estado interno. Quatro implementações satisfazem essa interface:

- a) **step**: saída constante e configurável, sem realimentação. É o modo de malha aberta exigido pelos ensaios de identificação (RF4): aplica degraus de drenagem diretamente, sem que nenhuma lei de controle interfira na resposta observada;
- b) **pid**: o PI discreto das Equações (4) a (6), com *anti-windup* condicional. Completa a estrutura um filtro de *setpoint*, prática padrão em controle de processos: em vez de aplicar mudanças de referência como degraus, o filtro conduz a referência ao novo valor. Tipo e tamanho são configuráveis, o tipo entre nenhum, média móvel e exponencial, e o tamanho em número de amostras. No tipo exponencial, a referência efetiva obedece a um filtro de primeira ordem,

$$\bar{r}_k = a_f \bar{r}_{k-1} + (1 - a_f) r_k, \quad a_f = e^{-T/\tau_f}, \quad (12)$$

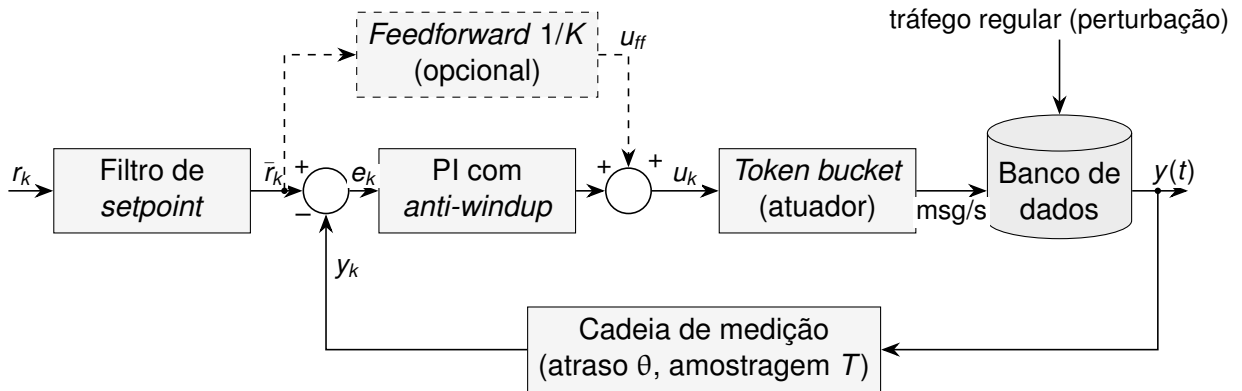
em que $\bar{r}_k$ é a referência efetiva usada no cálculo do erro e τ_f é a constante de tempo do filtro, dada pelo tamanho configurado. A referência sobe em rampa exponencial e o erro inicial deixa de saturar o atuador. A média móvel de N amostras transforma o degrau numa rampa linear que se completa em exatamente N amostras, o que dá acomodação da referência em tempo finito e previsível. O tipo nenhum entrega o degrau cru, útil nos ensaios em que se quer a excitação sem suavização. O papel desse filtro na partida da malha é avaliado experimentalmente na Seção 5.5;

- c) **pidff**: o mesmo PI acrescido de ação *feedforward* $u_{ff} = \bar{r}_k/K$, que posiciona a drenagem no ponto de operação previsto pelo modelo, sem esperar a ação integral, seguindo a referência filtrada;
- d) **smith**: o Preditor de Smith da Seção 2.6.4, com modelo interno FOPDT configurável.

A Figura 14 consolida esses elementos no diagrama de blocos da malha completa, na configuração que combina todos os complementos do PI (a mesma que este trabalho recomenda ao final da campanha experimental do Capítulo 5). O caminho direto vai da referência à planta; a realimentação passa pela cadeia de medição; e o tráfego regular da aplicação incide sobre a planta como perturbação, fora do caminho controlado.

Bloco a bloco, cada elemento cumpre um papel distinto na malha. O **filtro de setpoint**, primeiro bloco do caminho direto, aplica a Equação (12) à referência r_k e entrega a referência filtrada $\bar{r}_k$: mudanças de referência chegam ao restante da malha

Figura 14 – Diagrama de blocos da malha completa: filtro de *setpoint*, PI com *anti-windup*, ação *feedforward*, atuador *token bucket*, planta e cadeia de medição com filtro opcional na realimentação, com o tráfego regular como perturbação.



Fonte: Elaborada pelo autor (2026).

como rampas exponenciais, e não como degraus. O **somador de erro** compara $\bar{r}_k$ com a medição y_k e produz o erro e_k da Equação (4). O **PI com anti-windup** calcula a parcela de realimentação pelas Equações (5) e (6), com o integrador limitado à faixa que mantém a saída fora da saturação. Em paralelo, a **ação feedforward** multiplica a referência filtrada pelo inverso do ganho estático identificado, $u_{ff} = \bar{r}_k/K$, posicionando a drenagem no ponto de operação previsto pelo modelo sem depender do acúmulo do integrador; a soma das duas parcelas, saturada nos limites configurados, é a saída u_k . O **token bucket** do orquestrador converte u_k em vazão efetiva de mensagens, e o **banco de dados** é a planta, sobre a qual o tráfego regular da aplicação atua como perturbação de carga. Fechando a malha, a **cadeia de medição** da Seção 2.7 devolve a utilização de CPU ao controlador com o atraso θ e o período de amostragem T que dominam a dinâmica alcançável, opcionalmente com um **filtro de medição** aplicado antes do somador de erro. Desses blocos, apenas o *feedforward* depende do modelo identificado, e de forma benigna: um ganho K errado desloca o ponto de partida, e a ação integral do PI corrige o resíduo.

As duas formas de intercambialidade têm papéis distintos. Em uma implantação de produção, o controlador é uma dependência como as demais: a estratégia é selecionada na configuração da implantação, pela porta correspondente, e não se espera trocá-la em operação. A comutação em tempo de execução, por chamada à API REST, é um recurso do ambiente de experimentação (RF4): os ensaios comparativos do Capítulo 5 alternam controladores dezenas de vezes numa mesma sessão, e cada reinício descartaria o estado da malha e invalidaria a comparação. Os parâmetros (K_p , K_i , *setpoint*, saturações da saída e do integrador, tamanhos dos filtros de medição e de *setpoint*, modelo interno), por sua vez, podem ser lidos e alterados em execução

tanto no laboratório quanto em produção (RF3), pois o ajuste fino sem reinicialização é útil também na operação real.

4.4 O LEITOR DE MÉTRICAS

O leitor de métricas é o sensor da malha. Em ciclo próprio, ele consulta a fonte configurada e publica a leitura mais recente da variável de processo para o controlador. Três adaptadores implementam a porta *Reader*: a consulta à API do Prometheus (usada nos experimentos), que obtém a utilização de CPU do contêiner do banco a partir dos contadores expostos pelo cAdvisor; o AWS CloudWatch, que consome a métrica equivalente de instâncias gerenciadas na nuvem; e a raspagem direta de um *endpoint* de exportador.

Uma decisão de projeto foi deliberada: a suavização da medida é definida na própria consulta à fonte de métricas, configurada no leitor. É ali que se escolhe, por exemplo, entre a derivada sobre os dois pontos mais recentes e uma taxa média sobre uma janela longa. Como discutido na Seção 2.7, cada janela de suavização acrescenta um atraso da ordem de metade da sua largura. Deixar essa escolha explícita na configuração do sensor, em vez de aceitá-la como padrão invisível do provedor, permite identificar a planta sob a mesma medição usada depois em malha fechada: o atraso introduzido pela suavização fica capturado no modelo identificado, e o compromisso entre ruído e atraso torna-se um parâmetro conhecido do projeto. Além da janela definida na consulta, a cadeia oferece um filtro geral opcional da medição, com os mesmos tipos disponíveis para a referência (média móvel ou exponencial, tamanho em amostras), aplicado à variável de processo antes de qualquer controlador, para os casos em que a fonte não permita ajustar a consulta. Como a identificação lê a mesma medição filtrada, esse filtro é absorvido pelo modelo identificado e passa a integrar a planta aparente, o que mantém coerentes identificação e operação.

4.5 ADAPTADORES DE FILA E DE DESTINO

Quatro adaptadores implementam a porta *Queue*. O adaptador Apache Kafka é a implementação de referência para produção: consumo *pull*, confirmação por *offset* e durabilidade das mensagens. O adaptador AWS SQS cobre o cenário de nuvem gerenciada (RNF4), com semântica de confirmação por recibo. O adaptador em memória atende ao desenvolvimento, sem depender de infraestrutura externa. O adaptador constante, por fim, é uma fila sintética que emite indefinidamente a mesma operação: é ele que garante custo homogêneo por mensagem nos ensaios de identificação.

A porta *Target* tem duas implementações. A de referência é o MySQL: cada mensagem carrega um comando SQL, executado pelo adaptador, que só então confirma a operação. A segunda é um destino HTTP genérico, que entrega o conteúdo

da mensagem como requisição a um serviço *web* arbitrário, abrindo o mecanismo de controle para qualquer aplicação cuja carga se queira regular, e não apenas bancos de dados. O cenário de *cache* discutido no Capítulo 3 é o exemplo típico: em serviços HTTP na frente de um banco, requisições de escrita podem invalidar as camadas de *cache* e disparar picos de CPU desproporcionais ao tamanho da requisição, o tipo de dinâmica que favorece regular pela carga medida.

4.6 METODOLOGIA EXPERIMENTAL

A validação segue o fluxo clássico de projeto de malhas de controle de processos: identificação, sintonia e validação em malha fechada.

Na **identificação**, o controlador *step* é ativado e a drenagem é levada de um valor de repouso a um degrau, dentro da região de operação linear da planta. A resposta de CPU é registrada e, sobre a curva de reação, ajusta-se o modelo FOPDT, primeiro pelo método dos dois pontos e depois por refinamento numérico. O custo por operação é fixado usando o modo de operações idênticas da fila em memória. Além de controle experimental, a escolha é representativa do caso de uso alvo (Seção 3.1). Cada ensaio registra os metadados do cenário (volume de dados, comando executado e custo estimado da consulta), garantindo reprodutibilidade.

Na **sintonia**, os ganhos dos controladores são obtidos diretamente do modelo identificado, pela regra SIMC da Equação (7) (SKOGESTAD, 2003), sem otimização fina: o objetivo é uma sintonia mínima, suficiente para validar a prova de conceito.

Na **validação**, cada controlador é submetido ao mesmo protocolo de ensaio: partida ao *setpoint*, espera de acomodação, injeção de uma perturbação externa de carga (um gerador de consultas independente, acionado pela API), e recuperação. As séries temporais de todas as variáveis são extraídas da base de métricas para arquivos CSV, e as métricas de desempenho (sobressinal, tempos característicos, pico e recuperação da perturbação) são calculadas por ferramentas próprias de análise. O Capítulo 5 apresenta a implementação de todos esses blocos, o ambiente de experimentação e os resultados obtidos.

5 DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS

Este capítulo apresenta o desenvolvimento realizado e analisa os resultados, em duas partes que se complementam. A primeira cobre a construção: a Seção 5.1 detalha a implementação do sistema, a Seção 5.2 apresenta o ambiente de identificação e monitoramento montado ao redor dele, e a Seção 5.3 descreve as ferramentas de análise dos experimentos. A segunda parte é a prova de conceito que valida esse ferramental: a Seção 5.4 apresenta a identificação da planta, a Seção 5.5 cobre a sintonia dos controladores, a Seção 5.6 os compara em malha fechada, e a Seção 5.7 sintetiza a recomendação para o caso do banco de dados. A Seção 5.8 fecha o capítulo com os entregáveis e impactos do trabalho.

5.1 IMPLEMENTAÇÃO DO SISTEMA

A solução do Capítulo 4 foi implementada como um único serviço em Go, cuja organização interna segue a estratificação em camadas do *Domain-Driven Design* (Seção 2.5), com quatro camadas de responsabilidade única e dependências apontando sempre para o domínio.

A camada de **entidades** define os tipos que circulam pelo sistema: a mensagem (identificador e conteúdo da operação), a resposta do destino (sucesso, erro e latência), a medição de métricas e o conjunto de parâmetros do controlador. São estruturas puras de dados, sem dependência de qualquer tecnologia, e formam o vocabulário comum a todas as demais camadas.

A camada de **casos de uso** implementa as ações do sistema, uma por pacote, cada qual orquestrando entidades e portas para cumprir uma única finalidade: consumir uma mensagem da fila e entregá-la ao destino; criar mensagens na fila; processar uma nova medição de métricas e acionar o controlador; consultar o estado da fila; ler e alterar os parâmetros do controlador; trocar o controlador ativo; trocar a fila ativa; e configurar a perturbação externa usada nos ensaios. A granularidade de um caso de uso por ação mantém cada arquivo pequeno e testável isoladamente, e torna o efeito de qualquer chamada externa rastreável a um único ponto do código.

A camada de **manipuladores** (*handlers*) é a tradução da porta HTTP para os casos de uso: cada rota da API valida a requisição, converte os dados para os tipos do domínio, invoca o caso de uso correspondente e formata a resposta. Os manipuladores não contêm lógica de negócio.

O Quadro 2 consolida as rotas expostas pela porta HTTP. A lista espelha a camada de casos de uso: cada rota de escrita aciona exatamente um deles, e a leitura de qualquer recurso tem a rota correspondente. As rotas de leitura e ajuste do controlador atendem ao requisito RF3; as trocas de controlador e de fila em execução são os recursos do ambiente de experimentação discutidos na Seção 4.3 (RF4); as

rotas de injeção de mensagens e de perturbação são a base dos ensaios do restante do capítulo.

Quadro 2 – Rotas da API do sistema.

Método	Rota	Função
GET	/	interface de operação embarcada
GET	/ping	verificação de vida do serviço
GET	/metrics	métricas internas, no formato raspado pelo Prometheus
POST	/messages	enfileira uma mensagem
POST	/messages/flood	injeta um lote de mensagens (carga de ensaio)
GET, PATCH	/pid	lê e altera os parâmetros do controlador ativo
PUT	/controller/type	troca a estratégia de controle (experimentação)
GET, PUT	/queue/type	consulta e troca a fila ativa (experimentação)
GET, PUT	/queue/query	consulta e altera a operação sintética da fila constante
GET, POST	/disturbance	consulta e configura a perturbação externa dos ensaios

Fonte: Elaborado pelo autor (2026).

A camada de **repositórios** reúne os adaptadores que implementam as portas, organizados por porta: controladores, filas, destinos e leitores de métricas. Completam o sistema pacotes de infraestrutura interna reutilizável, como os filtros de sinal, média móvel e exponencial, empregados em dois papéis distintos (filtro de referência dos controladores e filtro geral opcional da cadeia de medição), os clientes HTTP e SQL, o servidor *web*, o registro de métricas da aplicação e utilitários de configuração e de relógio, este último abstraído para permitir testes determinísticos de comportamento temporal.

As decisões de implementação dos adaptadores de fila mostram o valor da porta como fronteira, porque as duas tecnologias de produção realizam a mesma semântica por mecanismos bem diferentes. No Kafka, mensagens não são removidas do tópico: a confirmação é implementada como o avanço do *offset* comprometido do grupo de consumidores, e uma mensagem não confirmada volta a ser lida porque o *offset* não avançou. No SQS, a leitura torna a mensagem invisível por uma janela configurável (*visibility timeout*) e a confirmação é a remoção explícita da mensagem; se o processo falhar antes de confirmar, a mensagem reaparece ao fim da janela, que deve, portanto, ser dimensionada acima do tempo de processamento esperado. O orquestrador não conhece nenhum dos dois mecanismos: enxerga apenas *dequeue* e *ack*, e a garantia de não perda do RF2 vale idêntica sobre ambos. As outras duas filas servem ao laboratório: a fila em memória para testes, e a fila constante, que emite indefinidamente uma mesma operação configurável e é a base dos ensaios com custo homogêneo por construção.

No sensor e no atuador, as escolhas seguiram a mesma lógica de reaproveitar semântica consolidada. O leitor do Prometheus faz uma consulta instantânea à API

com uma expressão configurável, associada à variável de processo; o do CloudWatch busca estatísticas da métrica com período configurável, sobre uma janela de dois períodos, para garantir que sempre exista um ponto publicado a retornar, uma acomodação à latência de publicação discutida na Seção 2.7. O *token bucket* do orquestrador é o limitador da biblioteca de taxa mantida pelo próprio projeto da linguagem Go (THE GO AUTHORS, 2026), com a saída do controlador aplicada como novo limite a cada ciclo, em execução e sem pausa do consumo. O gerador de perturbação dos ensaios usa o mesmo limitador, em instância própria: a carga externa que desafia a malha é imposta pelo mesmo mecanismo que a malha usa para regular, o que torna a taxa da perturbação tão precisa quanto a da atuação.

A composição de todos os componentes é feita por injeção de dependência em um único ponto do programa, usando a biblioteca Uber fx (UBER TECHNOLOGIES, 2026): os adaptadores concretos são escolhidos por configuração e ligados às portas na inicialização. Nos testes automatizados, os adaptadores são substituídos por objetos simulados (*mocks*) gerados a partir das interfaces, o que permite exercitar o orquestrador e os controladores sem qualquer infraestrutura.

O Quadro 3 ilustra uma configuração de implantação comentada, para o regime de 60 s. É nesse único arquivo que o engenheiro responsável pela implantação escolhe os adaptadores de cada porta e registra os parâmetros da malha; as decisões de projeto discutidas nos capítulos anteriores ficam documentadas nos próprios campos: o período do laço na cadência do provedor de métricas, a consulta de medição definida explicitamente no leitor, a saturação e o teto do integrador na faixa da drenagem de regime, o filtro de *setpoint* exponencial com tamanho de duas amostras e o *feedforward* pelo inverso do ganho estático identificado.

Quadro 3 – Exemplo comentado de configuração de implantação (regime de $T = 60$ s).

```

http:
  addr: ":8080"          # API e interface de operacao
queue:
  protocol: kafka        # fila ativa: kafka | sqs | inmemory | constant
kafka:
  brokers: ["kafka:9092"]
  topic: "messages"
target:
  protocol: sql          # destino: sql | http
  driver: mysql
  dsn: "usuario:senha@tcp(mysql:3306)/aplicacao"
metrics_poller:
  interval_ms: 60000     # tique do laço (T = 60 s)
  protocol: prometheusapi # leitor: prometheusapi | cloudwatch | exporter
  endpoint: "http://prometheus:9090/api/v1/query"
  mappings:              # a consulta define a medicao e a suavizacao
    'irate(container_cpu_usage_seconds_total{...}[3m])*100': cpu
controller:
  type: "pidff"          # estrategia: pid | pidff | smith | step
orchestrator:
  workers: 1             # consumidores concorrentes
pid:
  setpoint: 50           # CPU alvo (%)
  kp: 0.137              # ganhos de projeto do regime de 60 s
  ki: 0.003
  min: 0
  max: 40                # saturacao da drenagem (msg/s)
  i_clamp: 40            # anti-windup: teto do integrador
  ff_gain: 2.3           # feedforward u_ff = r/K (0 desliga)
  setpoint_filter_type: exponential # nenhum | media movel | exponencial
  setpoint_filter_size: 2 # tamanho em amostras (2T)

```

Fonte: Elaborado pelo autor (2026).

5.2 O AMBIENTE DE IDENTIFICAÇÃO E MONITORAMENTO

O sistema não opera sozinho: os ensaios de identificação e a operação supervisionada dependem de uma cadeia de observabilidade que foi montada como parte do desenvolvimento. Todo o ambiente é declarado em Docker Compose e sobe com um único comando (RNF2), compreendendo o banco MySQL, o coletor cAdvisor, os exportadores de métricas do banco e da fila, o Prometheus, o Grafana e a aplicação.

5.2.1 A cadeia de métricas: cAdvisor, Prometheus e Grafana

Os contêineres colaboram em uma cadeia com papéis bem definidos. Na base, o **cAdvisor** inspeciona o executor de contêineres e expõe, para cada um, os contadores acumulados de tempo de CPU; é dele que sai a medida de esforço do banco de dados, lida no nível do contêiner e, portanto, imune a particularidades do processo interno. Exportadores dedicados complementam a base com métricas específicas do MySQL e da fila. O **Prometheus** raspa periodicamente todos esses alvos, inclusive a

própria aplicação, que publica suas variáveis internas (medição filtrada, saída do controlador, acúmulo da fila, parâmetros vigentes, tempos de enfileiramento e consumo), e armazena tudo como séries temporais consultáveis.

Sobre essa base, dois consumidores leem os mesmos dados para fins distintos. O sistema de controle consulta a API do Prometheus como sensor da malha (Seção 4.4). O **Grafana**, provisionado automaticamente com fonte de dados e painéis versionados junto ao código, exibe o supervisório em tempo real (RNF1): variável de processo contra *setpoint*, taxa de drenagem, acúmulo da fila e histórico dos parâmetros do controlador. A Figura 15 mostra a visão geral do painel durante a operação, com a malha regulando a CPU do banco no *setpoint* de 50 %.

Essa dualidade tem uma consequência metodológica valiosa: as curvas analisadas nos experimentos e as curvas vistas pelo operador saem da mesma fonte, eliminando divergência entre a medição realimentada pela malha e a exibida pelo supervisório.

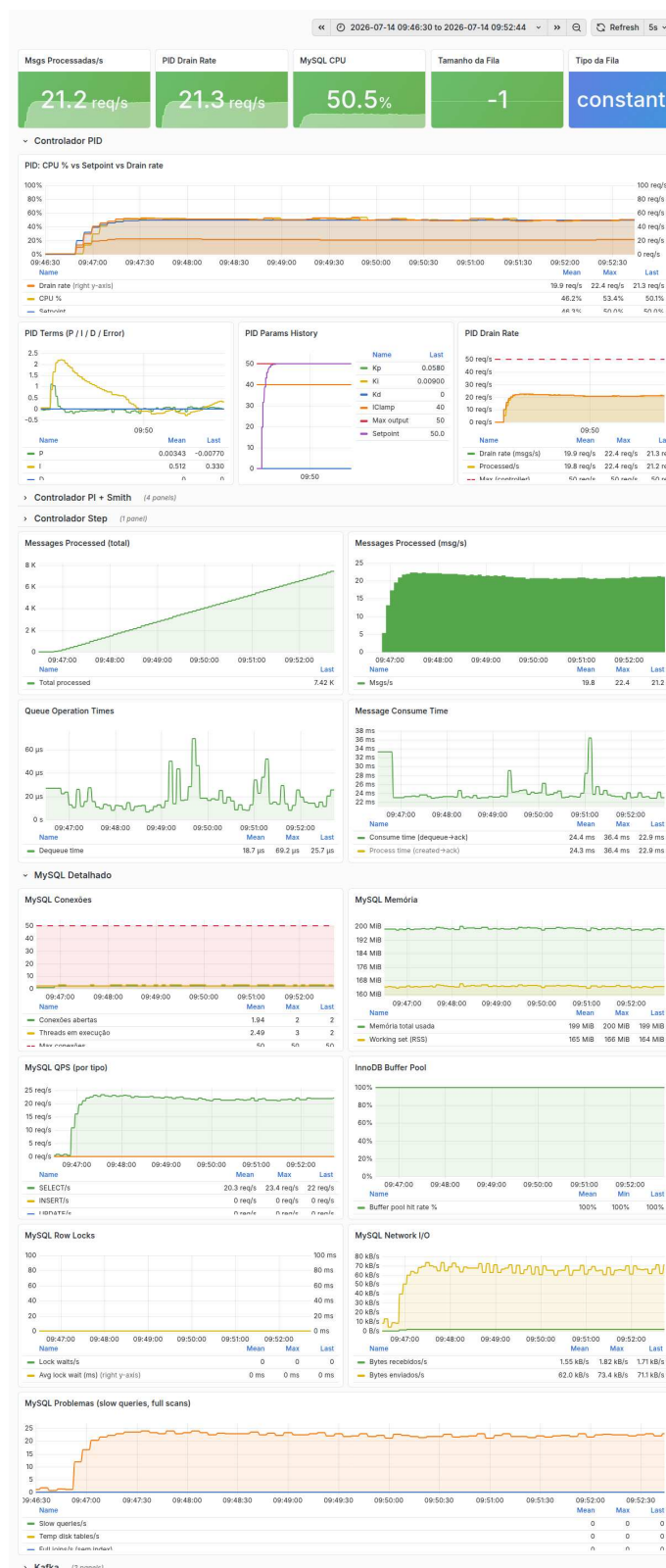
As métricas publicadas pela aplicação instrumentam o orquestrador por dentro. A Figura 16 mostra os painéis da fila durante uma sessão em malha fechada: o total acumulado de mensagens processadas, a taxa instantânea de consumo e, na linha inferior, os tempos de operação. A leitura dos tempos resume bem a divisão de custos do sistema: a retirada da fila (*dequeue*) custa dezenas de microssegundos, enquanto o ciclo completo de consumo de uma mensagem, dominado pela execução da consulta no banco, custa dezenas de milissegundos. O custo próprio do orquestrador é, portanto, três ordens de grandeza menor que o do trabalho que ele regula.

5.2.2 A interface de operação

Uma interface *web* embarcada na aplicação fecha o ciclo de operação, cumprindo o papel de interface homem-máquina do sistema: o corpo da página embute os painéis do supervisório, e um cabeçalho de estado (Figura 17) mantém o controlador ativo e seus parâmetros vigentes sempre visíveis durante a operação.

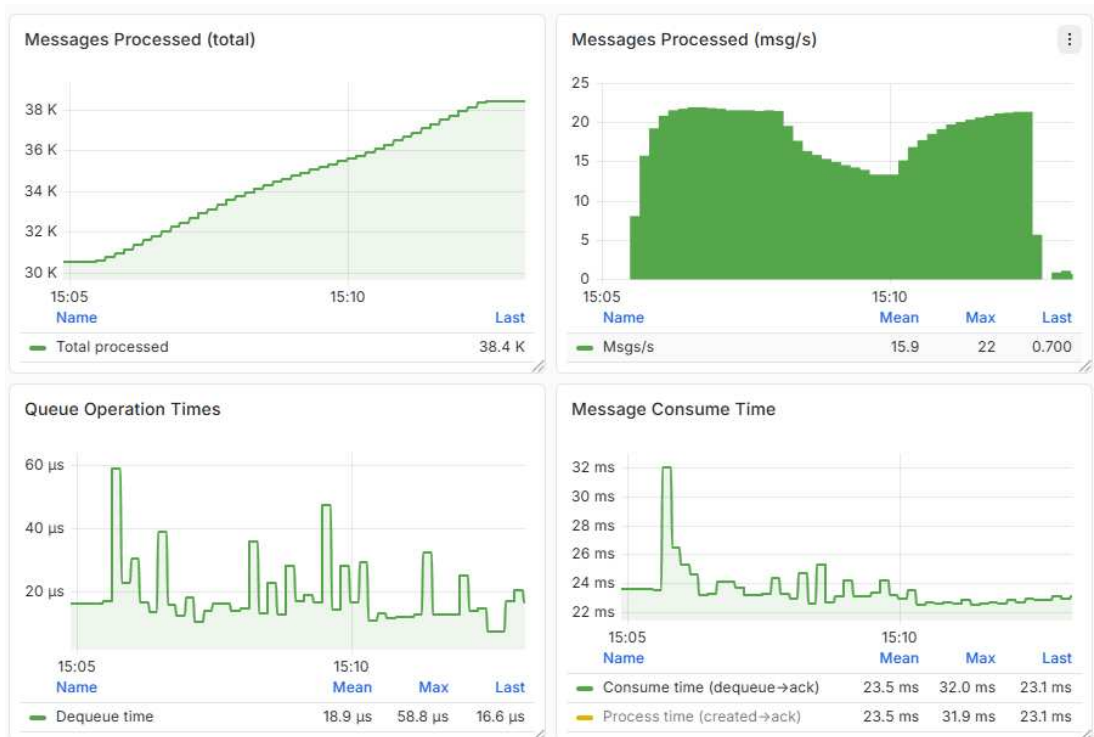
Em um painel de configuração (Figura 18), o operador executa sem reinicialização todas as ações que os ensaios exigem: troca o controlador ativo por um seletor (com os campos de parâmetros ajustados dinamicamente ao tipo escolhido), edita *setpoint*, ganhos, saturações e filtro, ajusta o intervalo e a consulta de medição, troca a fila ativa e sua operação sintética, injeta lotes de mensagens para carga, e configura a perturbação externa (taxa, duração e consulta) usada nos ensaios de rejeição. Um atalho leva ao painel do Grafana, de modo que ajuste e observação convivem lado a lado. Foi essa interface que tornou praticável a rotina experimental do trabalho, em que uma sessão de sintonia encadeia dezenas de alterações de parâmetros.

Figura 15 – Visão geral do painel supervisorio durante a operação: indicadores instantâneos (taxa de processamento, drenagem comandada, CPU do banco, fila), seção do controlador e detalhamento do MySQL (conexões, memória, consultas por tipo, *buffer pool*, *locks* e tráfego de rede).



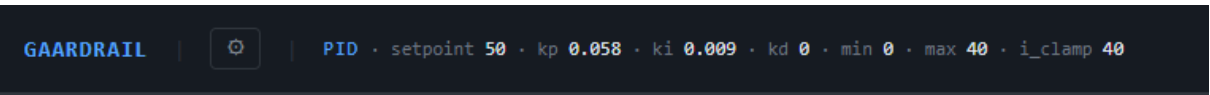
Fonte: Elaborada pelo autor (2026).

Figura 16 – Painéis da fila no supervisório: total processado, taxa de consumo e tempos de operação, com a retirada da fila em microssegundos e o consumo completo, dominado pelo banco, em milissegundos.



Fonte: Elaborada pelo autor (2026).

Figura 17 – Cabeçalho de estado da interface: controlador ativo e parâmetros vigentes da malha.



Fonte: Elaborada pelo autor (2026).

Figura 18 – Painel de configuração da interface de operação: controlador, medição, parâmetros da malha, fila ativa e perturbação externa, todos ajustáveis em tempo de execução.

CONFIGURAÇÃO

CONTROLADOR

pid

controlador ativo pid

AQUISIÇÃO (POLLING)

intervalo 10000 ms (10 s)

QUERY DA MEDIÇÃO (CPU)

irate(container_cpu_usage_seconds_

PARÂMETROS

SETPOINT: 50

KP: 0.058

KI: 0.009

KD: 0

MIN: 0

MAX: 40

I_CLAMP: 40

✓ Salvar

FILA

constant

QUERY CONSTANTE

SELECT id, val, score FROM flood_

✓ aplicar query

PERTURBAÇÃO

QUERY

SELECT id, val, score FROM flood_

RATE (QUERIES/S, 0 = PARAR)

0

DURAÇÃO (S, 0 = ATÉ PARAR)

120

✓ aplicar

■ parar

Fonte: Elaborada pelo autor (2026).

5.2.3 Implantação e segurança

Para aproximar os experimentos das condições reais, o ambiente também foi implantado em um servidor remoto na nuvem, com publicação automática a cada atualização do código. As comunicações com a aplicação e com o banco de dados usam TLS (RNF3). Esses cuidados não alteram os resultados de controle, mas sustentam a alegação de que a ferramenta está pronta para cenários de produção, e não apenas para bancada.

5.3 FERRAMENTAS DE ANÁLISE DOS EXPERIMENTOS

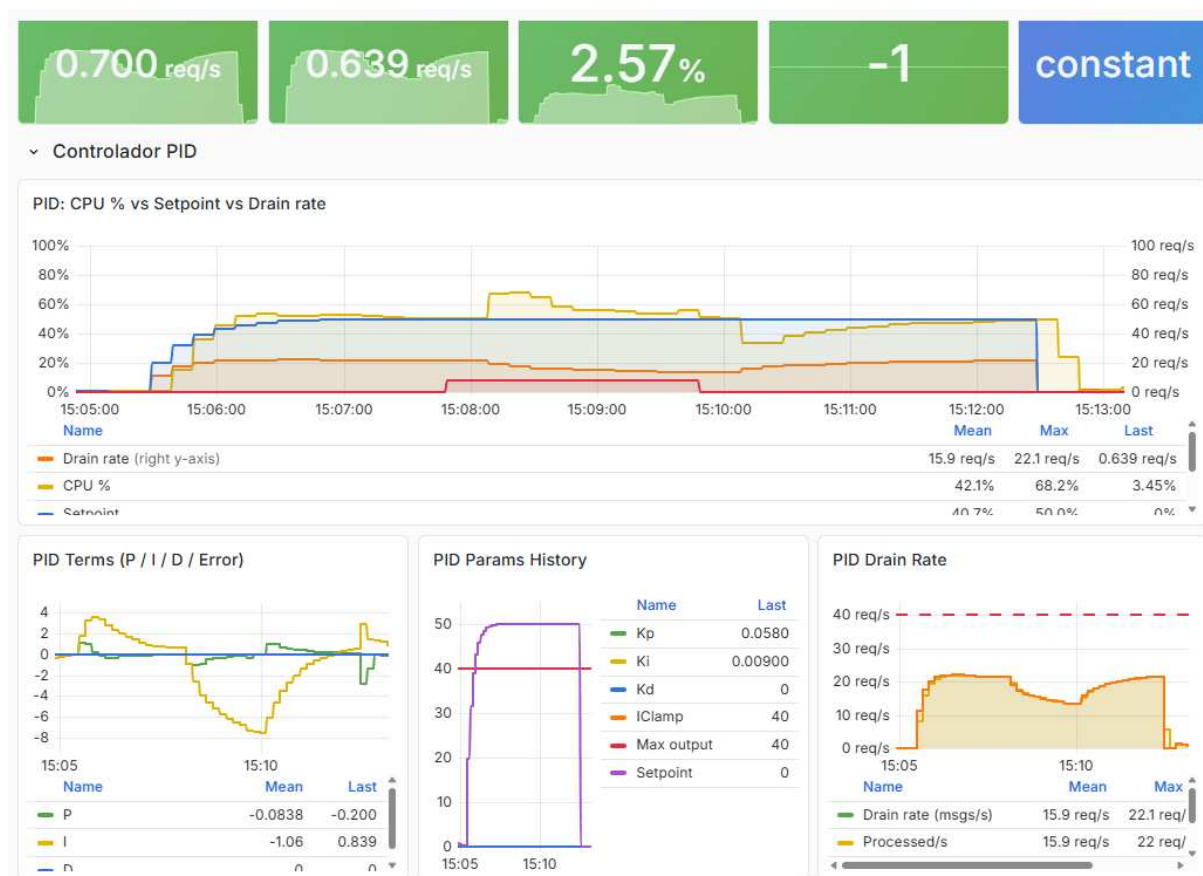
Os resultados deste capítulo dependem de um ferramental de medição e análise que foi, ele próprio, parte do desenvolvimento. O princípio organizador é o desacoplamento entre coleta e análise por meio de um formato intermediário: cada experimento produz um arquivo CSV com todas as séries temporais relevantes, arquivado em uma pasta própria e nunca sobrescrito.

O fluxo é o seguinte. Durante o ensaio, todas as variáveis da malha (medição de CPU, saída do controlador, acúmulo da fila, parâmetros vigentes) são publicadas como métricas e armazenadas pelo Prometheus. Ao final, uma ferramenta de extração consulta a base de métricas na janela do experimento e produz o CSV: os pontos de todas as séries são alinhados em uma grade temporal comum, as colunas recebem nomes curtos e descritivos (por exemplo, `measured_cpu` no lugar da expressão de consulta que origina a série) e uma coluna de tempo relativo facilita a comparação entre ensaios. Esse único artefato alimenta duas cadeias de análise independentes: no MATLAB, o ajuste do modelo FOPDT por mínimos quadrados (com a estimativa dos dois pontos como ponto de partida) e o cálculo dos ganhos de projeto; em Python, o cálculo das métricas de desempenho, dos tempos característicos e das figuras de comparação.

Completem o ferramental um registrador de metadados, que grava para cada ensaio o volume de dados, o comando executado e o custo estimado da consulta (o *Query Cost* do MySQL, usado como aproximação do custo de CPU por operação), e um protocolo padronizado de malha fechada, que executa as fases de partida, acomodação, perturbação e recuperação com tempos padronizados. Com isso, cada experimento citado neste capítulo é reproduzível a partir da sua pasta de artefatos.

Durante os ensaios, o painel supervisorio acompanha a malha em tempo real, e expõe o interior do controlador, não apenas as variáveis externas da malha. A Figura 19 mostra a seção dedicada ao controlador durante um ensaio em malha fechada: além da variável de processo contra o *setpoint* e a drenagem comandada, os painéis inferiores exibem os termos proporcional e integral com o erro, o histórico dos parâmetros aplicados e a drenagem contra o seu limite de saturação. Essa visibilidade encurta o diagnóstico de sintonia: saturação do atuador, acúmulo do integrador e mudanças de parâmetros no meio do ensaio ficam evidentes no mesmo quadro em que se observa a resposta.

Figura 19 – Seção do supervisório dedicada ao controlador: variável de processo, *set-point* e drenagem; termos proporcional e integral; histórico de parâmetros; e drenagem contra o limite de saturação.



Fonte: Elaborada pelo autor (2026).

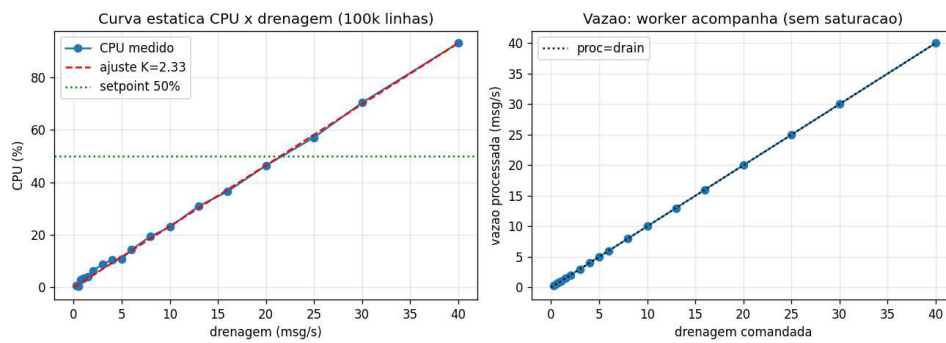
5.4 IDENTIFICAÇÃO DA PLANTA

5.4.1 Região linear de operação

Os primeiros ensaios de identificação, executados sobre uma tabela de um milhão de linhas, produziram resultados inválidos: a consulta de varredura completa sobre esse volume custa tanto que poucas mensagens por segundo levam a CPU acima de 100 % (o contador soma os núcleos do contêiner), saturando o processo fora de qualquer região linear. A solução foi reduzir o cenário padrão para uma tabela de cem mil linhas: o custo estimado pelo otimizador para a mesma consulta (*Query Cost*) cai de aproximadamente 103 mil para cerca de 10 mil, uma redução de dez vezes que recoloca a planta na região linear. O episódio confirma, na prática, a sensibilidade discutida na Seção 3.1: o mesmo comando, sobre estados diferentes da tabela, define plantas de ganhos muito diferentes. É por essa razão que todos os ensaios registram o *Query Cost* do cenário como metadado de reprodutibilidade.

A Figura 20 mostra a curva estática levantada nesse cenário: a drenagem foi elevada em escada de 0,25 a 40 mensagens por segundo, registrando-se a CPU de regime em cada patamar. A relação é aproximadamente linear em toda a faixa de interesse, com ganho estático $K \approx 2,3\%$ de CPU por (msg/s), sem saturação até cerca de 90 % de CPU. Desse levantamento sai o ponto de operação nominal: para um *setpoint* de 50 %, a drenagem de regime é de aproximadamente 21 mensagens por segundo.

Figura 20 – Curva estática da planta no cenário padrão (tabela de 10^5 linhas): CPU de regime em função da taxa de drenagem.



Fonte: Elaborada pelo autor (2026).

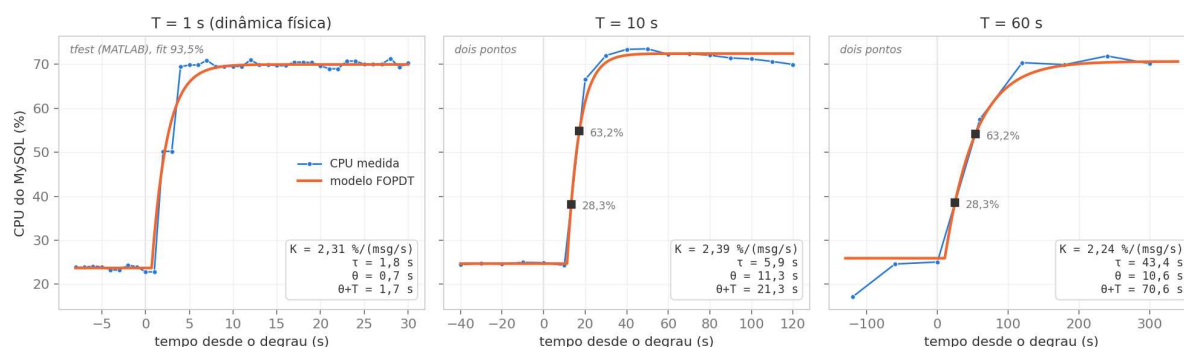
5.4.2 Dois regimes de amostragem

A campanha experimental foi conduzida em dois regimes de amostragem, escolhidos por representarem as duas famílias de provedores de métricas encontradas em produção: $T = 60$ s, a cadência típica dos serviços de monitoramento de nuvem, e $T = 10$ s, a cadência de agentes de monitoração mais rápidos e do escalonador automático do Kubernetes. Em cada regime, a planta foi identificada por degraus de drenagem de 10 para 30 mensagens por segundo, com o modelo FOPDT ajustado pelo método dos dois pontos e o atraso de amostragem incorporado ($\theta_{ef} = \theta + T$).

A Figura 21 mostra as três curvas de reação com os modelos ajustados sobrepostos. O painel da esquerda, com medição rápida ($T = 1$ s), é a referência que revela a dinâmica física: o banco acomoda uma mudança de carga em cerca de dois segundos, e o modelo refinado por mínimos quadrados sobre a curva completa adere aos dados com 93,5 % de ajuste. Nos dois regimes da campanha, os marcadores indicam os cruzamentos de 28,3 % e 63,2 % da variação total, dos quais a Equação (2) extrai τ e θ . Nos três casos a resposta de primeira ordem descreve bem o que se mede; a Tabela 1 consolida os parâmetros.

A leitura da Tabela 1 confirma o comportamento previsto na Seção 2.6. O ganho estático é invariante ($K \approx 2,3$) nos três regimes, pois a planta é a mesma; re-

Figura 21 – Curvas de reação e modelos FOPDT ajustados nos três regimes de medição (degrau de drenagem $10 \rightarrow 30$ msg/s, tabela de 10^5 linhas). Os marcadores indicam os pontos de 28,3% e 63,2% usados pelo método dos dois pontos.



Fonte: Elaborada pelo autor (2026).

Tabela 1 – Modelos FOPDT identificados por regime de medição (degrau de drenagem $10 \rightarrow 30$ msg/s, tabela de 10^5 linhas).

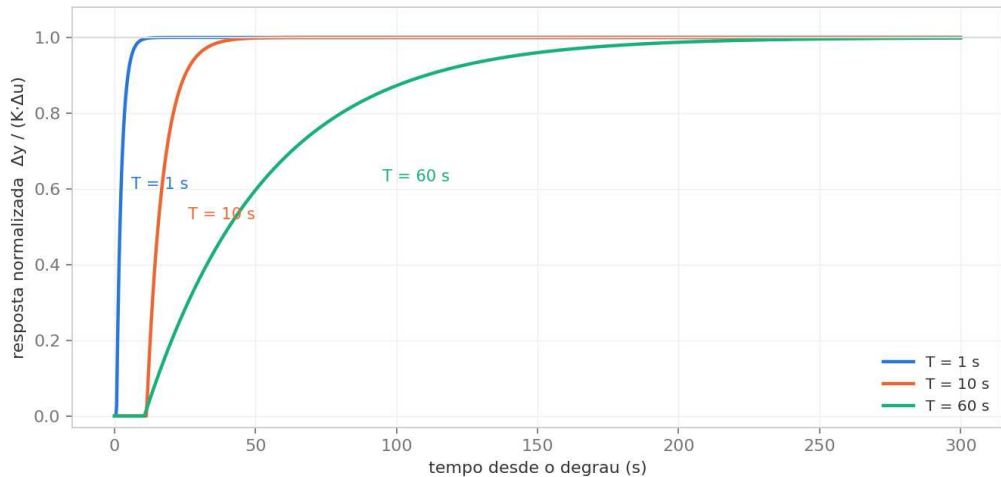
Regime	K (%/(msg/s))	τ (s)	θ (s)	θ_{ef} (s)
$T = 1$ s (referência)	2,31	1,8	0,7	1,7
$T = 10$ s	2,39	5,9	11,3	21,3
$T = 60$ s	2,24–2,30	43,4	10,6	70,6

Fonte: Elaborada pelo autor (2026).

identificações em dias distintos ao longo da campanha retornaram valores entre 2,23 e 2,39, o que dá uma medida da reprodutibilidade. A constante de tempo aparente, ao contrário, cresce mais de vinte vezes entre a medição de referência e o regime de 60 s: a amostragem fina resolve a dinâmica real, e a lenta dilui a planta rápida em uma resposta que parece de outro processo. O atraso extraído da curva pouco distingue os dois regimes da campanha; a diferença entre eles concentra-se em τ e no termo de amostragem que compõe θ_{ef} . Quando $T \gg \tau$, a própria estimativa de τ fica mal condicionada, com uma armadilha prática: se a série do ensaio é extraída da base de métricas em passo mais fino que o da coleta, a interpolação cria uma escada falsa na curva de reação e produz um τ artificialmente pequeno. Todos os ensaios passaram a extrair as séries no passo nativo da coleta por essa razão.

A Figura 22 resume o efeito num único gráfico, com as respostas ao degrau dos três modelos normalizadas pelo ganho. A resposta que se completa em poucos segundos na medição de referência estende-se por mais de três minutos quando vista a 60 s. As três curvas descrevem o mesmo processo físico; o que muda é a cadeia de medição interposta entre a planta e o controlador, e a subseção seguinte decompõe de onde vem essa lentidão aparente.

Figura 22 – Respostas ao degrau dos três modelos identificados, normalizadas pelo ganho estático: o mesmo processo físico, visto por cadências de medição diferentes, aparenta dinâmicas separadas por mais de uma ordem de grandeza.



Fonte: Elaborada pelo autor (2026).

5.4.3 O fator de atraso: de onde vem o atraso da malha

Para separar o que é físico do que é imposto pela medição, o atraso efetivo do laço foi decomposto experimentalmente, aplicando degraus em malha aberta e cronometrando cada elo com uma fonte de medição rápida (leitura direta do coletor de contêineres, que atualiza em cerca de dois segundos). A Figura 23 mostra a decomposição nos dois regimes, e a Tabela 2 detalha os componentes.

Tabela 2 – Componentes do atraso efetivo do laço nos dois regimes.

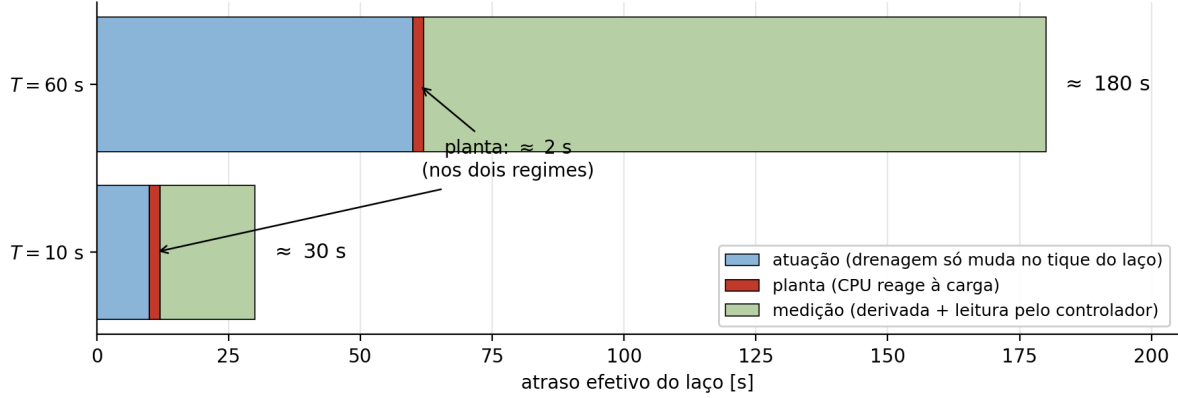
Componente	$T = 60\text{ s}$	$T = 10\text{ s}$	Físico?
Atuação (drenagem só muda no passo do laço)	~60 s	~10 s	não
Planta (CPU reage à mudança de carga)	~2 s	~2 s	sim
Medição (derivada + leitura pelo controlador)	60–120 s	~30 s	não
Atraso efetivo total no laço	~180 s	~30 s	—

Fonte: Elaborada pelo autor (2026).

Convém relacionar esses números ao modelo identificado: o atraso efetivo da Tabela 1 ($\theta_{ef} \approx 70\text{ s}$ no regime de 60 s) captura apenas o caminho percorrido pelo ensaio em malha aberta, do degrau de drenagem à medição; em malha fechada, uma correção completa percorre também a espera da atuação, presa ao passo do laço, e o pior caso da derivada em janela, elevando o total ao valor da Tabela 2.

O resultado é o achado central do trabalho: dos cerca de 180 segundos de atraso enfrentados pelo controle no regime de 60 s, apenas dois são da planta. Todo o

Figura 23 – Decomposição do atraso efetivo do laço nos dois regimes: a atuação e a medição, presas à cadência do laço, dominam; a planta responde em cerca de dois segundos em ambos.



Fonte: Elaborada pelo autor (2026).

restante é autoimposto pela cadência do próprio laço de medição e atuação, e escala de forma aproximadamente linear com T : mudar o provedor de métricas de 60 s para 10 s reduz o atraso em cerca de seis vezes sem tocar no banco de dados. É o mesmo dilema de um escalonador automático de produção: ele não luta contra a lentidão da aplicação, mas contra a latência do seu próprio sistema de medição.

5.5 SINTONIZAÇÃO DOS CONTROLADORES

As especificações de malha fechada adotadas na campanha foram um sobressinal máximo de 5% e um tempo de acomodação alvo de quatro amostras do regime: $t_s = 40$ s a $T = 10$ s e $t_s = 240$ s a $T = 60$ s. A regra SIMC da Equação (7) converte o tempo de acomodação na constante de tempo desejada em malha fechada pela aproximação de primeira ordem $t_s \approx 4\tau_c$, sujeita a um piso de robustez: $\tau_c = \max(t_s/4 - \theta_{ef}, \theta_{ef})$. Nos dois regimes o alvo de quatro amostras pede mais do que o atraso efetivo permite ($t_s/4 < \theta_{ef}$) e o piso prevalece, recaindo na recomendação robusta de Skogestad (2003), $\tau_c = \theta_{ef}$: exigir da malha resposta mais rápida que o próprio atraso obrigaria o controlador a corrigir sem ter observado o efeito da correção anterior. A mesma escolha atende à especificação de sobressinal, pois com $\tau_c = \theta_{ef}$ a resposta a mudanças de referência tem caráter de primeira ordem, com sobressinal de poucos por cento (SKOGESTAD, 2003).

Sobre os modelos da Tabela 1, no regime de 60 s, com $K = 2,24$, $\tau = 43,4$ s e $\theta_{ef} = 70,6$ s, obtém-se $K_p = (1/2,24) \cdot 43,4/(70,6 + 70,6) \approx 0,137$ e, com $\tau_I = \min(43,4; 4 \cdot 141,2) = 43,4$ s, $K_i = K_p/\tau_I \approx 0,003$. O mesmo cálculo no regime de 10 s, com $K = 2,39$, $\tau = 5,9$ s e $\theta_{ef} = 21,3$ s, resulta em $K_p \approx 0,058$ e $K_i \approx 0,0098$. A

Tabela 3 resume os ganhos de operação dos dois regimes.

Tabela 3 – Ganhos do PI por regime, calculados pela regra SIMC com $\tau_c = \theta_{ef}$ sobre os modelos identificados.

Regime	$\tau_c = \theta_{ef}$ (s)	K_p	K_i
$T = 10$ s	21,3	0,058	0,009
$T = 60$ s	70,6	0,137	0,003

Fonte: Elaborada pelo autor (2026).

Dois aspectos desses números ajudam a entender o comportamento observado adiante. Primeiro, nos dois regimes a regra resulta em $\tau_I = \tau$: o zero do controlador cancela o polo dominante do modelo, e a dinâmica de malha fechada passa a ser governada por τ_c e pelo atraso (SKOGESTAD, 2003). Disso sai uma previsão verificável: acomodação de regulação da ordem de $\theta_{ef} + 4\tau_c$, cinco vezes o atraso efetivo, ou seja, cerca de 107 s a 10 s e 353 s a 60 s; os tempos medidos na rejeição de perturbação da Seção 5.6, de 106 e 340 segundos, são coerentes com ela. Segundo, o ganho proporcional do regime lento é o maior dos dois (0,137 contra 0,058), o que surpreende à primeira vista, mas decorre da regra: K_p é proporcional à razão $\tau/(\tau_c + \theta_{ef})$, e a planta aparente de 60 s tem constante de tempo grande em relação ao próprio atraso. A cautela do regime lento aparece no ganho integral, três vezes menor (0,003 contra 0,009): cada acréscimo do integrador só é observado uma amostra depois, e a 60 s essa espera é seis vezes maior.

Duas lições de sintonia, aprendidas ao longo da campanha, condicionaram os ganhos finais.

A primeira é a origem do piso de robustez do cálculo: o tempo de resposta pedido ao projeto deve respeitar o atraso efetivo do laço, e não a dinâmica aparente da planta. Ensaios que especificaram acomodação mais agressiva produziram ganhos elevados e levaram a malha a ciclo limite, com a drenagem alternando entre zero e o máximo. Uma varredura do ganho integral no cenário de rejeição, conduzida em fase anterior da campanha no regime de 60 s, quantificou o fenômeno: com o ganho proporcional fixo, K_i foi multiplicado por 1, 2, 4 e 8, e cada nível recebeu a mesma perturbação externa de carga. O desvio-padrão da CPU regulada cresceu de forma monotônica com o ganho (de 10,5 para 21,6 pontos percentuais (pp) entre os extremos), enquanto o pico de desvio e o tempo de recuperação da perturbação não melhoraram: além de certo ponto, aumentar o ganho não acelera a recuperação, apenas eleva a variância da malha, porque nenhum ganho elimina a espera pela próxima medição.

A segunda: sobressinal causado pela trajetória da referência se resolve moldando a referência, não endurecendo o controlador. Na partida a frio (repouso até o *setpoint* de 50 %), o erro inicial enorme saturava a drenagem no máximo e, como a planta responde em segundos, a CPU disparava a quase 90 %: um comportamento

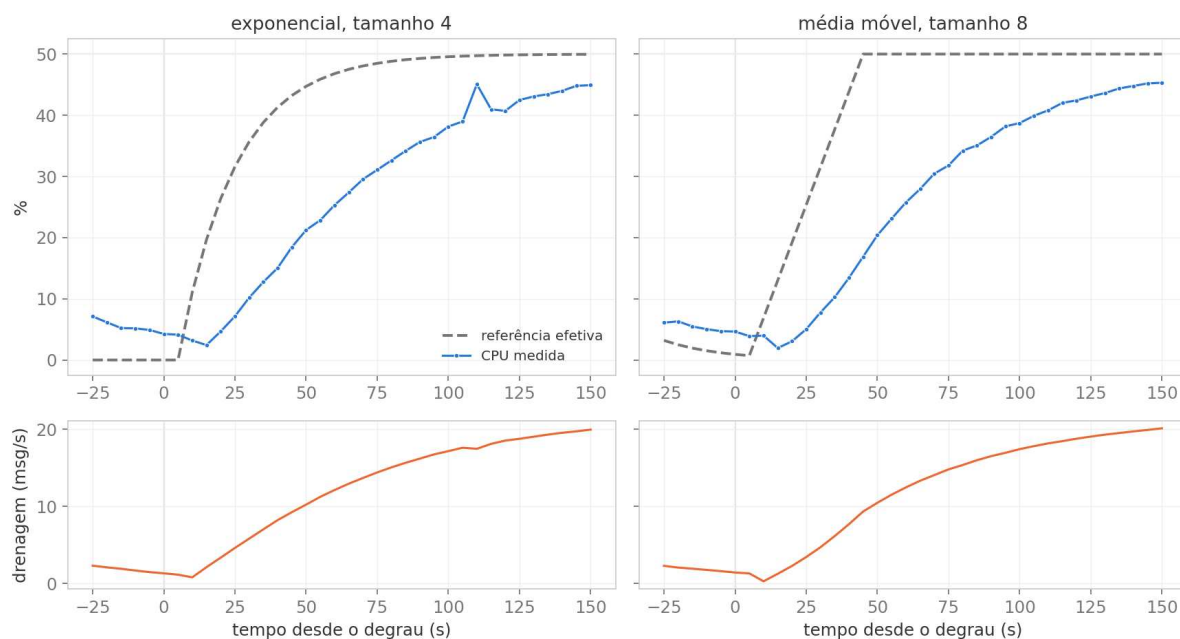
que, em produção, derrubaria o banco que o sistema existe para proteger. O *anti-windup* condicional atenuou o problema, mas o que o eliminou foi o filtro de *setpoint* da Equação (12), do tipo exponencial e com tamanho de duas amostras, combinado à partida com o controlador sempre ativo e o degrau aplicado na referência. O tipo foi escolhido pela forma da aproximação: o exponencial conduz a referência ao novo valor de modo assintótico, sem quinas na trajetória que o controlador precise perseguir. O tamanho de duas amostras é o compromisso entre amortecer a partida e não atrasar demais o alcance do ponto de operação. O sistema oferece ainda a média móvel, cuja rampa fecha em tempo finito, como alternativa de configuração. Com as duas medidas, o sobressinal de partida caiu de dezenas de pontos percentuais para praticamente zero nos dois regimes, e a rejeição de perturbação não foi afetada. O remédio, porém, tem um custo próprio: com a referência conduzida em rampa e ganhos que respeitam o atraso, a malha passa a levar alguns minutos para alcançar o ponto de operação na partida. A Seção 5.6.2 mostra que esse custo é recuperável pela ação *feedforward*.

Os dois tipos de filtro de referência foram exercitados em um ensaio de demonstração do ferramental, à parte da campanha dos dois regimes. No ambiente de laboratório, com cadência acelerada de $T = 5$ s e o PI puro, o protocolo partiu da planta ociosa: a referência foi zerada, aguardou-se a CPU estabilizar abaixo de 5 % e só então o degrau para 50 % foi aplicado, uma vez com cada tipo. A Figura 24 sobrepõe a referência efetiva à CPU medida e à drenagem em cada caso. O período e a carga não são os da campanha, de modo que os números não se comparam aos das seções seguintes; o que a figura mostra é a diferença de trajetória entre os tipos e o efeito dela sobre a planta.

Fechada a sintonia, verificou-se o que essa cadeia de projeto é capaz de prever. A lei de controle implementada no sistema, o PI posicional das Equações (4) a (6) com o filtro de referência da Equação (12), foi simulada numericamente sobre o modelo FOPDT identificado de cada regime, partindo do estado registrado no início do ensaio real e sem nenhum parâmetro ajustado depois da identificação. A Figura 25 sobrepõe as respostas prevista e medida na partida em malha fechada dos dois regimes.

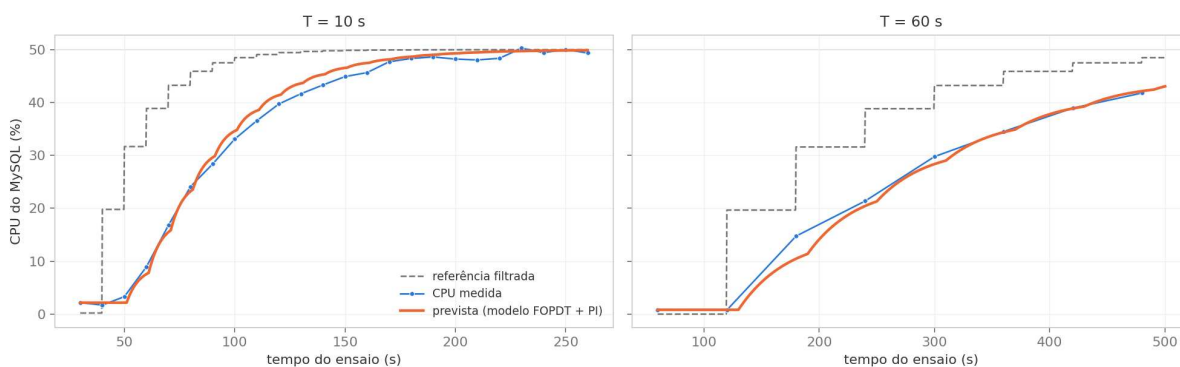
O modelo prevê a malha real com erro médio absoluto de 1,1 ponto percentual a 10 s e de 0,9 a 60 s, computado nos instantes de amostragem da partida. A 10 s, a malha real cruzou 90 % do alvo em 160 s, contra 137 s previstos, diferença da mesma ordem da variabilidade observada entre corridas repetidas. O maior desvio, de 4,3 pontos percentuais no início da subida a 60 s, tem explicação na própria cadeia de medição: a derivada em janela deslizante que produz a métrica só se aproxima de uma dinâmica de primeira ordem, e a resposta real adianta-se um pouco ao modelo antes de as duas convergirem na acomodação. Um modelo de primeira ordem com atraso, identificado com um único degrau em malha aberta, mostrou-se portanto suficiente para antecipar o comportamento da malha fechada, e é essa previsibilidade

Figura 24 – Ensaio de demonstração dos tipos de filtro de referência, em cadência acelerada ($T = 5$ s), com o PI puro partindo da planta ociosa e degrau de referência para 50 %: o exponencial conduz a referência assintoticamente; a média móvel fecha a rampa em exatamente oito amostras e a drenagem acompanha em rampa quase linear.



Fonte: Elaborada pelo autor (2026).

Figura 25 – Partida em malha fechada nos dois regimes: resposta prevista pela simulação do PI sintonizado sobre o modelo FOPDT identificado, sobreposta à resposta medida no ensaio real. A referência efetiva é conduzida pelo filtro de *setpoint*.



Fonte: Elaborada pelo autor (2026).

que autoriza usá-lo como base da sintonia e da comparação entre controladores da Seção 5.6.

5.6 VALIDAÇÃO EM MALHA FECHADA

A validação seguiu o protocolo da Seção 4.6, em duas rodadas: primeiro a validação do controlador base, o PI, nos dois regimes; em seguida, a comparação controlada entre as três estruturas, com fases alongadas. Em ambas, o ciclo de ensaio é o mesmo: o controlador permanece ativo desde o início, estabiliza com referência nula, recebe o degrau de referência para 50 %, acomoda, sofre a perturbação externa de carga (8 consultas por segundo, equivalentes a cerca de 20 pontos percentuais de CPU) e se recupera. Em nenhum ensaio houve perda de mensagens, confirmando o requisito RF2 sob condições reais.

5.6.1 O PI nos dois regimes

As Figuras 26 e 27 mostram a malha fechada com o PI nos regimes de 10 e 60 segundos, e a Tabela 4 resume as métricas. O controlador é estável e limpo nos dois regimes: o sobressinal de partida é praticamente nulo (1,3 e 0,0 ponto percentual), a drenagem acomoda no ponto de operação previsto pela curva estática e a perturbação é rejeitada com desvio de pico moderado e retorno ao *setpoint*. A referência efetiva tracejada torna visível o trabalho do filtro: a malha persegue a referência conduzida, e não o degrau aplicado pelo operador. O filtro em uso é o exponencial com tamanho de duas amostras, pelas razões discutidas na Seção 5.5.

Tabela 4 – Malha fechada com o PI: métricas nos dois regimes.

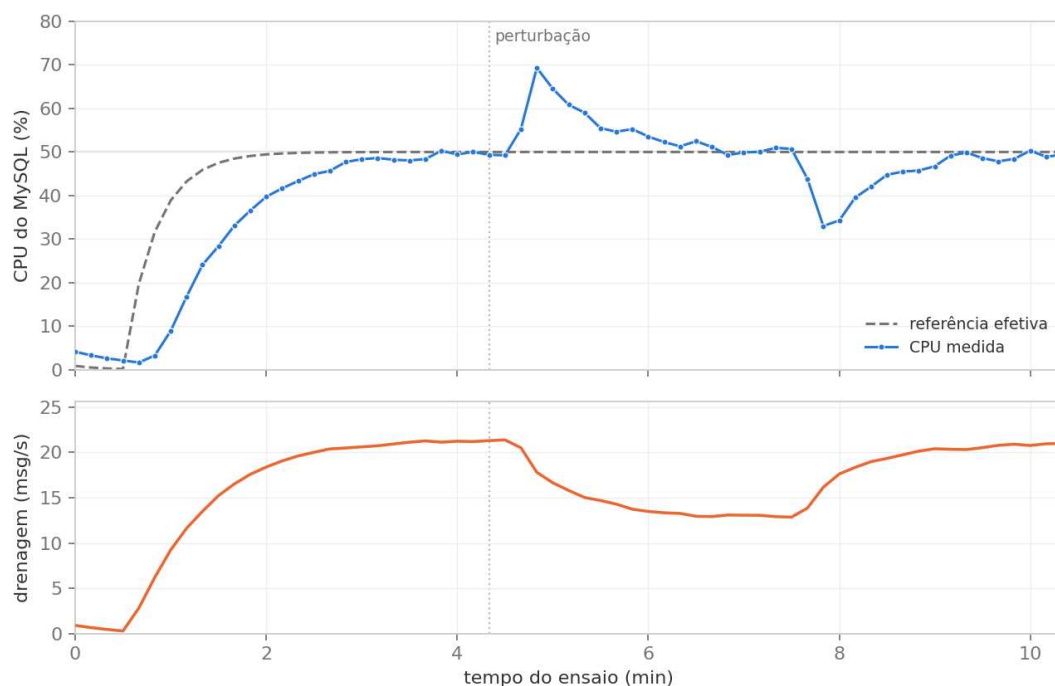
Métrica	$T = 10\text{ s}$	$T = 60\text{ s}$
Sobressinal de partida (pp)	1,3	0,0
Desvio de pico na perturbação (pp)	17,4	15,8
Drenagem de regime (msg/s)	~ 21	~ 21

Fonte: Elaborada pelo autor (2026).

A Figura 28 complementa as curvas de análise com a visão do operador: a mesma classe de ensaio registrada pelo painel do supervísório, no regime de 10 s. Nela ficam visíveis os elementos da metodologia em ação: a referência conduzida em rampa pelo filtro de *setpoint*, e não em degrau; a janela retangular da perturbação externa; a drenagem cedendo durante a perturbação e retomando o ponto de operação na recuperação.

Há uma limitação metodológica na comparação entre regimes: o desvio de pico aparentemente maior a 10 s é, em parte, artefato de medição. A derivada sobre janela longa usada a 60 s suaviza e esconde o pico real da perturbação, enquanto a

Figura 26 – Malha fechada com o PI no regime de $T = 10$ s: partida sem sobressinal, rejeição e recuperação da perturbação. A referência efetiva, tracejada, é conduzida pelo filtro de *setpoint* exponencial de duas amostras.



Fonte: Elaborada pelo autor (2026).

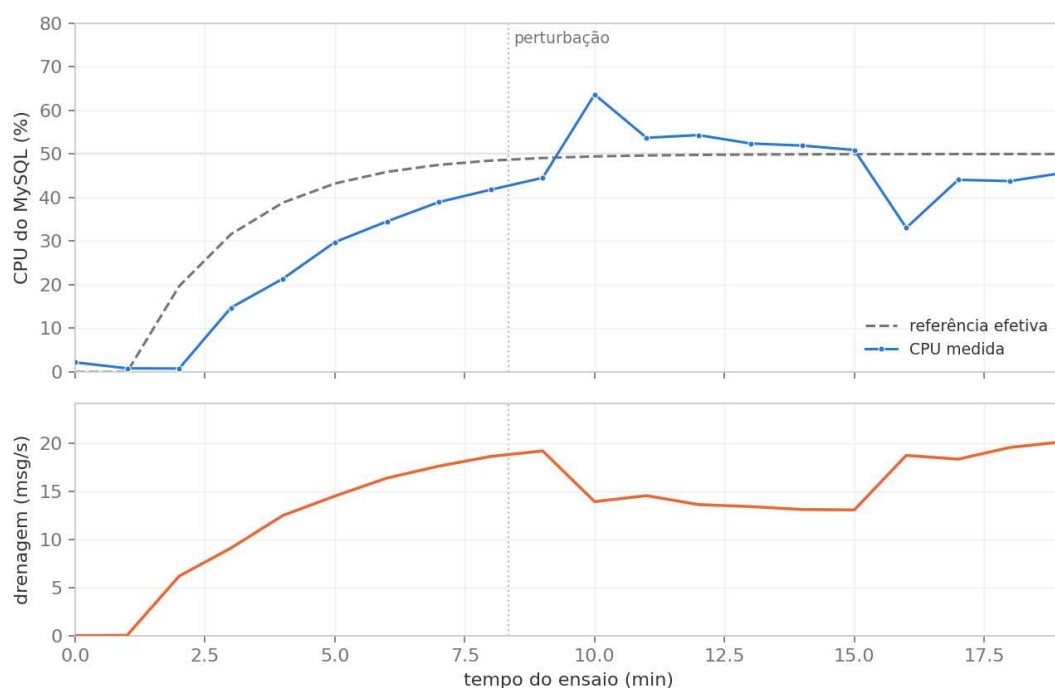
amostragem fina de 10 s o revela. Os números de pico entre regimes não são, portanto, diretamente comparáveis; dentro de cada regime, a comparação permanece válida.

5.6.2 Comparação controlada: o papel de cada estrutura

Validado o controlador base, a campanha comparou as três estruturas implementadas (PI puro, PI com *feedforward* e PI com Preditor de Smith) sob um protocolo desenhado para isolar o efeito de cada acréscimo: os três controladores operam com os *mesmos* ganhos de realimentação, o mesmo filtro de *setpoint* e o mesmo cenário, de modo que qualquer diferença de comportamento é atribuível à estrutura, e não à sintonia. Os tempos de ensaio foram alongados para garantir acomodação completa no *setpoint* antes da perturbação e carga externa longa o suficiente para a malha assentar sob perturbação. Ensaios comparativos anteriores da campanha, com sintonia distinta por controlador e fases curtas, confundiam estrutura com sintonia e transiente com regime, e foram refeitos sob este protocolo.

Antes dos números, cabe delimitar a confiança que eles comportam. As corridas são únicas, sem repetição estatística, e a variabilidade observada entre corridas ao longo da campanha é da ordem de 30 a 60 segundos nos tempos de acomodação;

Figura 27 – Malha fechada com o PI no regime de $T = 60$ s: mesma estrutura de ensaio, cadência de produção em nuvem. A referência efetiva, tracejada, é conduzida pelo filtro de *setpoint* exponencial de duas amostras.



Fonte: Elaborada pelo autor (2026).

diferenças dessa magnitude estão dentro do ruído e não devem ser lidas como vantagem de uma estrutura. As conclusões estruturais desta seção são robustas a essa ressalva; os valores absolutos, não.

A Figura 29 mostra as três respostas no regime de 10 s, e a Tabela 5 resume as métricas.

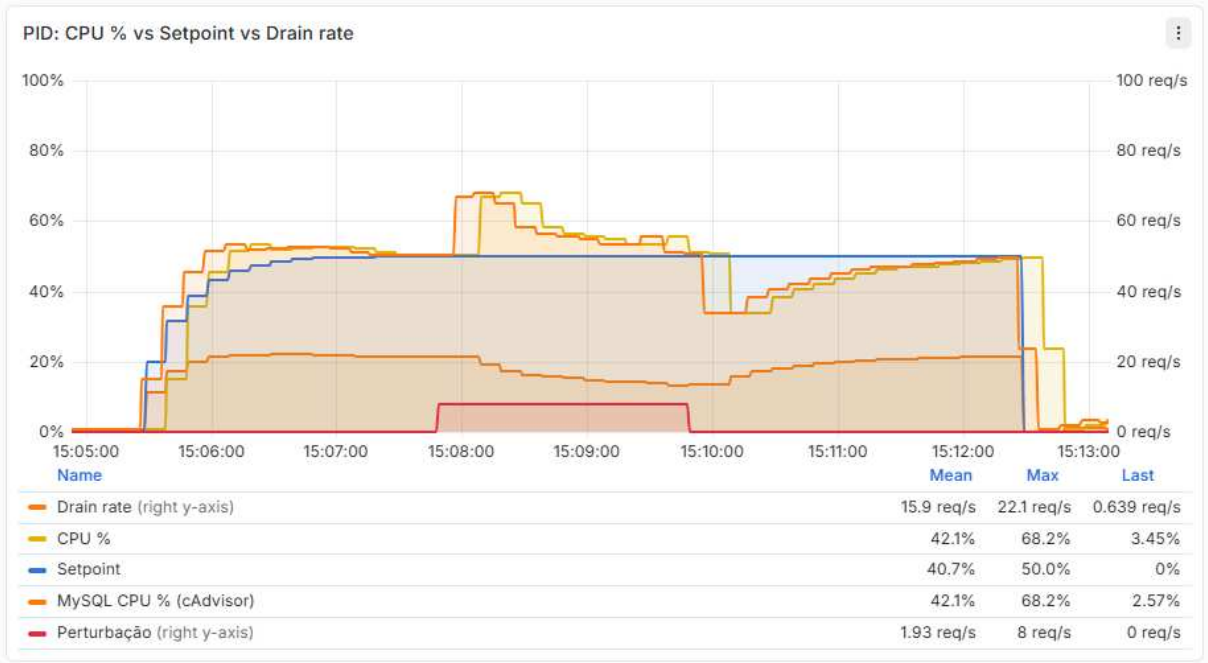
Tabela 5 – Comparação controlada no regime de $T = 10$ s: métricas das três estruturas com os mesmos ganhos de realimentação.

Métrica	PI	PI+FF	PI+Smith
Tempo até o <i>setpoint</i> na partida (s)	~ 350	~ 80	~ 350
Desvio de pico na perturbação (pp)	17,4	17,6	16,6
Acomodação da rejeição (s)	106	106	166

Fonte: Elaborada pelo autor (2026).

A ação *feedforward* empregada é função apenas da referência e permanece parada durante a perturbação, que é rejeitada pela ação integral; por isso, pico e tempo de acomodação equivalem aos do PI puro, dentro da variabilidade entre corridas. Antecipar também a perturbação exigiria um *feedforward* de perturbação, que pressupõe medi-la antes de seu efeito aparecer. A cadeia de medição disponível, porém, não

Figura 28 – Sessão de validação do PI ($T = 10$ s) vista pelo supervisor: referência em rampa pelo filtro de *setpoint*, CPU regulada, janela de perturbação externa e recuperação.



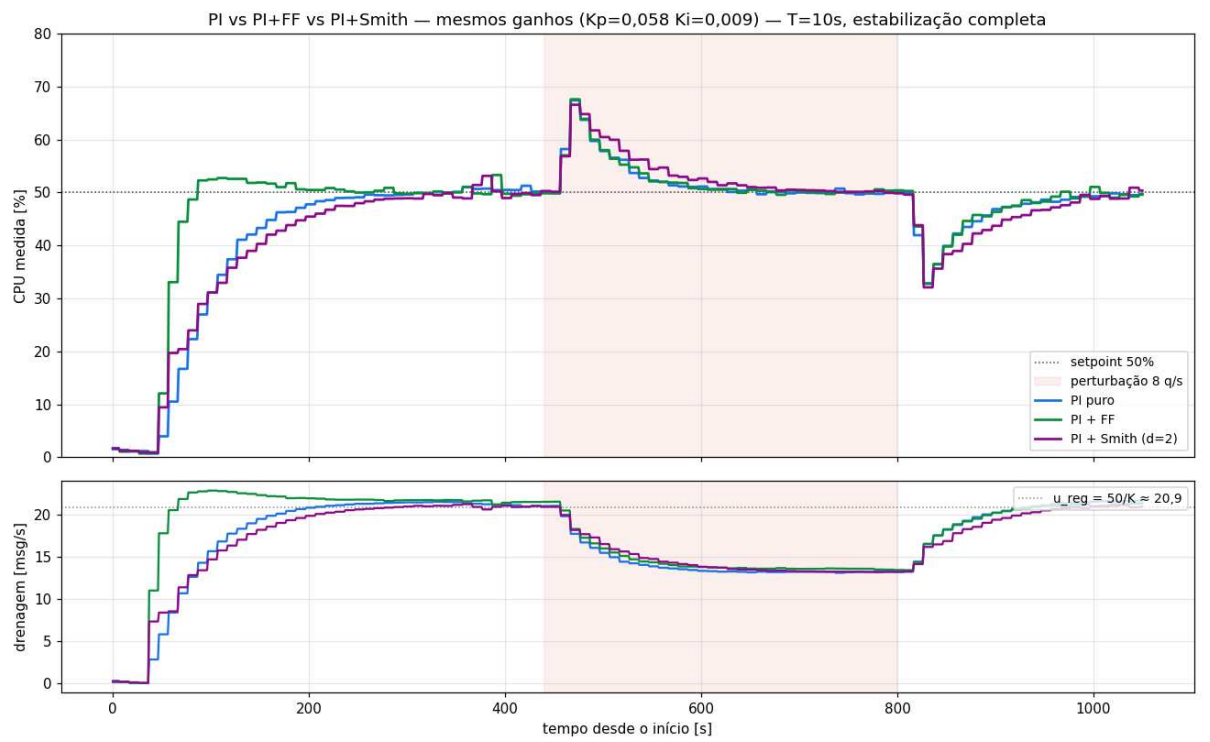
Fonte: Elaborada pelo autor (2026).

mede o tráfego concorrente em si, e sim o seu efeito agregado na CPU, quando a perturbação já está dentro da malha. A contribuição do termo de referência aparece onde a teoria a coloca: na partida, ele posiciona a drenagem no ponto de operação previsto pelo modelo sem esperar o acúmulo do integrador, e a malha alcança o *setpoint* em cerca de 80 segundos, contra cerca de 350 do PI puro. A divisão de papéis é a clássica: o *feedforward* compensa o que se conhece antes de o efeito chegar à saída, neste caso apenas a referência, e a realimentação corrige o que não se mede nem se modela, como a perturbação de carga e os erros do modelo.

O Preditor de Smith, com o modelo interno casado à identificação e o atraso interno dimensionado pelo atraso efetivo do laço, operou estável, mas não trouxe benefício neste regime: a rejeição foi cerca de 60 % mais lenta que a do PI (166 contra 106 segundos), sem redução do pico. O resultado é coerente com a teoria: a 10 s o atraso efetivo é de poucas amostras, e há pouco a compensar; e a rejeição de perturbação é o ponto fraco conhecido do preditor, pois a resposta à perturbação carrega a dinâmica de malha aberta da planta com o atraso, que nenhuma predição remove (NORMEY-RICO; CAMACHO, 2007).

O mesmo protocolo foi repetido no regime de 60 s, com uma adaptação: o Smith só operou de forma estável com ganhos abaixo dos de projeto do PI, de modo que a comparação o inclui na sua melhor sintonia viável, enquanto PI e PI com *feedforward*

Figura 29 – Comparação controlada no regime de $T = 10$ s: PI, PI com *feedforward* e PI com Preditor de Smith sob os mesmos ganhos de realimentação, na partida, na perturbação e na recuperação.



Fonte: Elaborada pelo autor (2026).

usam os ganhos de projeto. A Figura 30 compara PI e PI com *feedforward*, e a Tabela 6 resume as métricas das três estruturas.

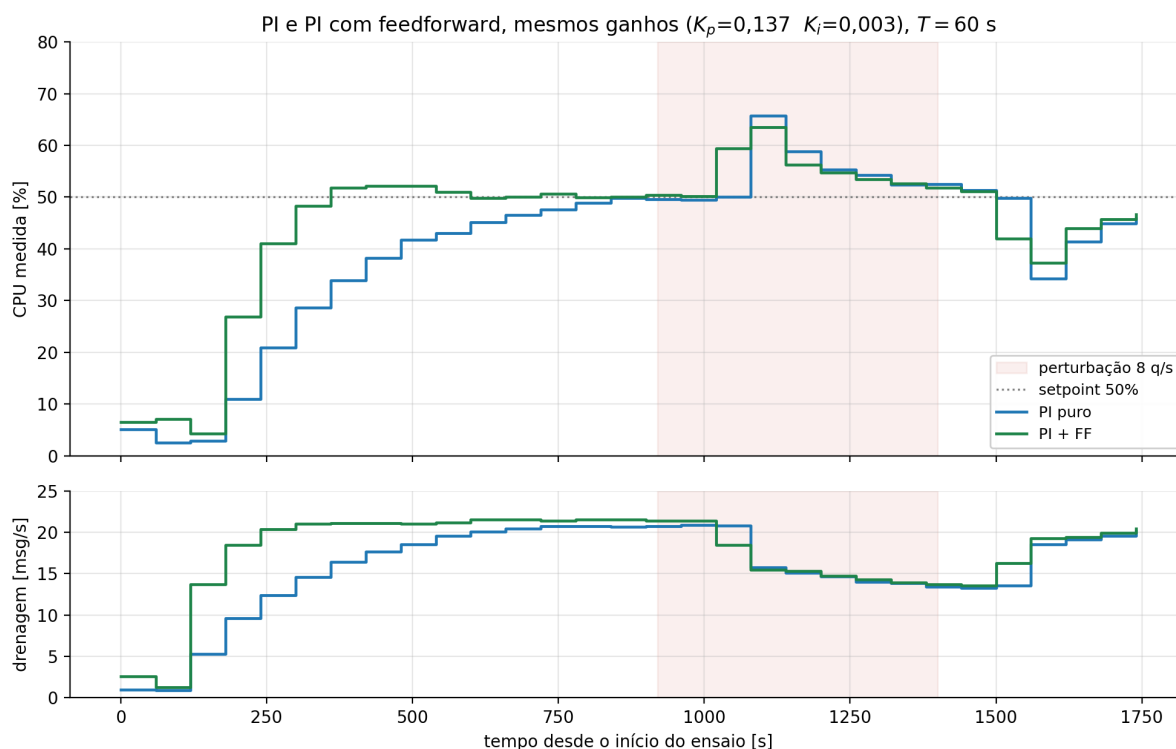
Tabela 6 – Comparação no regime de $T = 60$ s: métricas das três estruturas, cada qual na melhor sintonia viável.

Métrica	PI	PI+FF	PI+Smith
Tempo até o <i>setpoint</i> na partida (s)	~ 800	~ 280	não alcançou
Desvio de pico na perturbação (pp)	15,8	13,5	14,5
Acomodação da rejeição (s)	340	400	340

Fonte: Elaborada pelo autor (2026).

Os resultados do regime lento repetem o padrão do regime rápido, agora no território onde a compensação de atraso deveria valer a pena. O *feedforward* volta a resolver a partida (cerca de 280 contra 800 segundos) sem alterar a rejeição (a diferença de 340 para 400 segundos na acomodação está dentro da variabilidade entre corridas delimitada no início da seção), e o PI segue respondendo pela regulação. A conclusão dos dois regimes é a mesma: o PI, com seus complementos simples, resolve o problema; a estrutura com modelo embarcado não paga o próprio custo.

Figura 30 – PI e PI com *feedforward* no regime de $T = 60$ s, mesmos ganhos de projeto: o *feedforward* antecipa a partida; a rejeição de perturbação é equivalente.



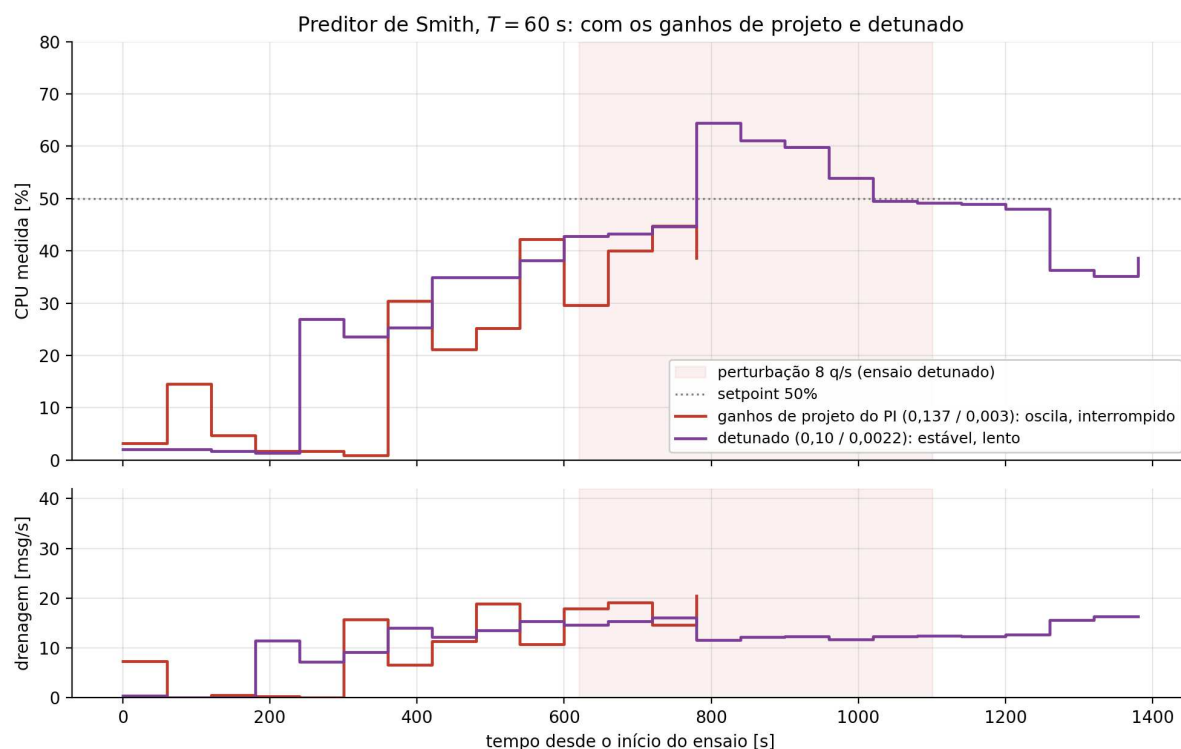
Fonte: Elaborada pelo autor (2026).

A Figura 31 documenta o desempenho ruim do Preditor de Smith nesse regime. Com os ganhos de projeto do PI, a malha oscila de forma sustentada, com a CPU alternando dezenas de pontos percentuais a cada amostra, e o ensaio foi interrompido; na única configuração estável encontrada, com ganhos reduzidos, a malha chega ao *setpoint* mais devagar que o PI puro e rejeita a perturbação sem qualquer vantagem sobre ele.

5.6.3 A causa da fragilidade: o atraso espalhado pela medição

A fragilidade documentada acima levanta uma objeção natural: se a planta foi identificada e o atraso é conhecido, por que a compensação falha? A pergunta foi investigada em duas frentes. Primeiro, a lei de controle do preditor foi replicada numericamente, passo a passo, e simulada sobre dois processos: um mundo ideal, em que a planta é exatamente o modelo FOPDT com atraso puro, e um mundo realista, que reproduz a cadeia de medição do laboratório (planta física rápida, média da derivada sobre o intervalo de coleta, amostragem e a defasagem entre o instante da coleta e o passo do laço). Depois, as previsões da simulação foram registradas e submetidas a ensaios reais em cadência acelerada ($T = 5$ s), que reproduz em minutos o fenômeno

Figura 31 – Preditor de Smith no regime de $T = 60$ s: com os ganhos de projeto do PI, a malha oscila e o ensaio é interrompido; com ganhos reduzidos, é estável porém mais lento que o PI puro.



Fonte: Elaborada pelo autor (2026).

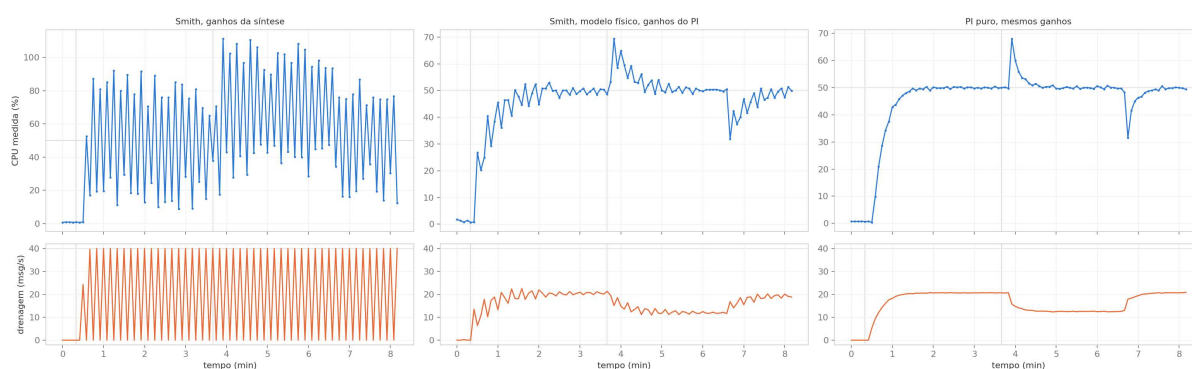
do regime lento, com a planta identificada no próprio ensaio.

A simulação separa as duas hipóteses: a implementação está correta, e o que falha é a premissa. No mundo ideal, o preditor com os ganhos da síntese direta é estável e limpo; no mundo realista, os mesmos ganhos produzem o ciclo limite observado nos experimentos, com período e amplitude compatíveis. O isolamento de causas completa o quadro: um erro de meia amostra no atraso já dispara o ciclo com os ganhos da síntese; nenhum comprimento de *buffer* (uma, duas ou três amostras) estabiliza o mundo realista; um erro de 20 % no ganho estático, sozinho, é inofensivo. A causa não está em saber ou não o valor do atraso, e sim na sua natureza: a cadeia de medição espalha o atraso ao longo de mais de uma amostra, com um perfil que depende do alinhamento entre a coleta e o passo do laço, alinhamento que não é observável de dentro da malha e muda a cada implantação. A identificação absorve parte desse espalhamento na constante de tempo aparente (a origem do crescimento de τ na Tabela 1), e a síntese direta, calculada sobre essa dinâmica fictícia na região em que sua fórmula é hipersensível a τ , entrega ganhos sem margem nenhuma para o erro de atraso que a estrutura da medição garante. Essa hipersensibilidade tem medida experimental direta:

duas identificações do mesmo processo, no mesmo regime de 60 s e em dias distintos, retornaram τ de 43,4 e de 68,7 s, e o ganho proporcional da síntese quase triplicou entre elas (de 0,20 para 0,53); o ensaio repetido no próprio regime de 60 s, com o modelo identificado no dia, reentrou em ciclo limite nas primeiras amostras após o degrau de referência. Um procedimento de projeto cujo resultado varia quase três vezes entre duas identificações honestas da mesma planta não é reproduzível o bastante para operar. Duas variantes tentadas em simulação reforçam o diagnóstico: dar ao preditor o modelo físico da planta (deixando todo o atraso no *buffer* puro) e substituir o ramo atrasado por uma média que imita o espalhamento; ambas continuam oscilando com ganhos acima dos do PI, porque nenhuma estrutura fixa representa um atraso cujo perfil depende de um alinhamento desconhecido (NORMEY-RICO; CAMACHO, 2007).

Os ensaios reais confirmaram as três previsões. A Figura 32 mostra as corridas e a Tabela 7 consolida as métricas: o preditor com os ganhos da síntese entra em ciclo limite de trilho a trilho (drenagem alternando entre zero e o máximo a cada amostra, CPU entre 9 e 111 %); o preditor com o modelo físico e os ganhos do PI é estável, mas carrega um serrilhado permanente que é o resíduo do próprio preditor convertido em correções espúrias; e o PI puro, com exatamente os mesmos ganhos, vence em todas as métricas de regulação e rejeição. Os picos acima de 100 % na primeira corrida têm leitura direta: a metade alta de cada ciclo crava a drenagem no máximo, que sozinha demanda um núcleo inteiro do banco, e a perturbação externa soma sua carga em cima, transbordando o processo para um segundo núcleo (o contador soma os núcleos do contêiner, como na Seção 5.4). Quem satura o banco não é a carga: é o controlador oscilando.

Figura 32 – Ensaio de diagnóstico em cadência acelerada ($T = 5$ s): Preditor de Smith com ganhos da síntese direta, Preditor de Smith com modelo físico e ganhos do PI, e PI puro com os mesmos ganhos. Linhas verticais: degrau de *setpoint* e janela de perturbação.



Fonte: Elaborada pelo autor (2026).

O resultado eleva a conclusão da campanha de um registro empírico para um

Tabela 7 – Diagnóstico em $T = 5$ s: as duas variantes do preditor contra o PI puro com os mesmos ganhos ($K_p = 0,096$; $K_i = 0,035$).

Métrica	Smith (síntese)	Smith (modelo físico)	PI
Desvio-padrão em regulação (pp)	31,3	1,6	0,2
Pico na perturbação (%)	111,2	69,3	68,0
Acomodação da rejeição (s)	não acomoda	55	40
Drenagem saturada (% das amostras)	100	0	0

Fonte: Elaborada pelo autor (2026).

mecanismo: dado o contrato do sensor deste sistema, um provedor de métricas cuja janela de suavização e cujo alinhamento de coleta são opacos para o controlador, nenhum compensador de atraso de estrutura fixa consegue operar com ganhos acima dos que o PI já usa. A partida das corridas não é comparável entre as colunas da Tabela 7, pois o filtro de *setpoint* atua apenas no PI; a comparação justa, de regulação e rejeição, é inteira dele.

5.6.4 Comparação com a taxa fixa

Contra esses números, a estratégia convencional de taxa fixa tem uma limitação estrutural, que independe de sintonia: diante da mesma perturbação, a drenagem não se move, e o desvio de CPU persiste enquanto durar a carga externa. As fases de referência dos próprios ensaios, em que a drenagem é mantida constante antes de fechar a malha, exibem esse comportamento. Em termos quantitativos: a perturbação usada na campanha adiciona cerca de 20 pontos percentuais de CPU; sob taxa fixa, esse desvio é permanente, enquanto a malha fechada o converte em um transiente com pico de 14 a 18 pontos percentuais e recuperação em cerca de 2 minutos no regime de 10 s e 6 minutos no de 60 s (Tabelas 5 e 6). A taxa fixa, para reagir, exige que um operador perceba o problema e intervenha. Essa é, em síntese, a vantagem que motivou o trabalho.

5.6.5 Exemplo de operação: a rotina experimental vista pela interface

As curvas de análise das seções anteriores são extraídas da base de métricas após cada ensaio; esta subseção complementa essa visão mostrando como um ensaio é vivido pelo operador, do ajuste à observação, inteiramente pela interface de operação da Seção 5.2.

Em um ensaio de rejeição típico, o operador confere no cabeçalho da interface o controlador ativo e os ganhos de projeto da Tabela 3, acompanha nos indicadores instantâneos a CPU do banco junto à referência e a taxa efetiva de processamento coincidindo com a drenagem comandada, e observa nos painéis da fila a taxa de consumo ceder durante a janela de perturbação e retomar em seguida, com o total

processado sempre crescente, sem perda de mensagens.

Uma sessão experimental completa encadeia dezenas desses ensaios sem reinicialização da aplicação: os parâmetros e o controlador ativo são alterados em execução (RF3 e RF4), e cada mudança fica registrada na própria série histórica do supervisor, lado a lado com os comportamentos que produziu, das oscilações sob sintonias agressivas às partidas em rampa conduzidas pelo filtro de *setpoint*. Foi essa capacidade de alternar configurações e observar o efeito imediatamente que tornou praticável a campanha experimental, em que uma única sessão de sintonia percorre dezenas de configurações.

5.7 O CONTROLADOR PARA O CASO DO BANCO DE DADOS

Os resultados sustentam uma recomendação mais simples do que a esperada no início da campanha: o **PI com *anti-windup* condicional, filtro de *setpoint* e ação *feedforward* de referência** é a escolha para os dois regimes de produção estudados. Cada elemento cobre um requisito distinto: o PI rejeita as perturbações no limite do que o atraso de medição permite; o *anti-windup* dá robustez sob saturação do atuador; o filtro de *setpoint* elimina o sobressinal de partida; e o *feedforward* devolve a velocidade de partida que o filtro e os ganhos conservadores retiram, sem custo, pois é um termo em malha aberta que não afeta a estabilidade nem a rejeição. Dos quatro, o único que depende do modelo é o *feedforward*, e de forma benigna: um ganho estático errado desloca o ponto de partida, e a ação integral corrige o resíduo.

Estruturas com modelo embarcado na malha, como o Preditor de Smith, não se justificaram em nenhum dos regimes estudados, e a Seção 5.6.3 mostra que a limitação não é de ajuste: com o alinhamento e a janela da medição opacos para o controlador, nenhum compensador de atraso de estrutura fixa opera com ganhos acima dos do PI. Nesta classe de sistema, tais estruturas devem ser tratadas como refinamento de especialista para sensores de comportamento conhecido, não como padrão.

Uma variante conhecida, o Preditor de Smith filtrado (NORMEY-RICO; CAMACHO, 2007), acrescenta ao preditor um filtro de robustez que atenua a sensibilidade ao erro de modelo. Ela não foi ensaiada por uma razão estrutural: o filtro pressupõe um atraso nominal bem definido em torno do qual robustecer a malha, enquanto a Seção 5.6.3 mostra que o atraso efetivo desta cadeia de medição é espalhado ao longo de mais de uma amostra, com perfil que depende do alinhamento entre a coleta e o passo do laço; não há valor único que o modelo interno possa adotar. Além disso, a robustez do filtro é comprada com resposta mais lenta à perturbação, e o teto de desempenho observado já é o do PI. O ensaio da variante permanece como continuidade natural para sensores de cadência fixa e conhecida (Capítulo 6).

A recomendação de maior impacto, porém, não é sobre o controlador, e sim so-

bre o sensor: como o atraso da malha é dominado pela cadência de medição (Tabela 2), encurtar essa cadência melhora o desempenho mais do que qualquer sofisticação da lei de controle. Um provedor de métricas de 10 s reduz o atraso efetivo em cerca de seis vezes em relação ao de 60 s, sem tocar na planta. Na prática, a troca do sensor por um mais rápido precede, em impacto, qualquer troca de controlador.

Os demais controladores desenvolvidos permanecem no sistema como ferramentas à disposição do engenheiro que configurar a malha: o modo de malha aberta para identificar a planta durante o comissionamento e o Preditor de Smith para cenários futuros em que um modelo fiel esteja disponível. Todos compartilham a mesma interface e são selecionados na configuração de cada implantação; no ambiente de experimentação, também em tempo de execução, o que permite tratar a escolha, durante o comissionamento, como um parâmetro de ensaio e não como decisão congelada.

5.8 ENTREGÁVEIS E IMPACTOS

O desenvolvimento resultou em dois entregáveis de natureza distinta, e a distinção ajuda a entender o valor de cada um.

O primeiro é o **projeto completo como está**: um laboratório de identificação e monitoramento de plantas de *software*. O conjunto formado pela cadeia de métricas (Seção 5.2), pela interface de operação, pelo controlador de malha aberta e pela cadeia de extração e análise (Seção 5.3) permite, diante de um serviço qualquer, levantar a curva estática, identificar o modelo dinâmico, projetar ganhos e validar a malha, tudo com o ambiente reproduzível que sobe em um comando. Foi esse laboratório que produziu os resultados deste capítulo, e ele tem valor próprio: cada nova planta (outro banco, outro serviço, outra fonte de métricas) pode ser caracterizada com o mesmo protocolo, sem reconstruir ferramental.

O segundo é o **produto mínimo de produção** que o laboratório valida: um conector enxuto composto do adaptador de fila, do leitor de métricas de CPU e do controlador PI com *anti-windup* condicional. É essa fração pequena do sistema que precisa existir dentro de um ambiente de produção para entregar o controle de *back-pressure*; supervisório, ensaios, perturbação e análise ficam de fora. A arquitetura plugável é o que permite esse recorte: as portas isolam as três dependências que o conector precisa carregar.

A verificação dos requisitos fecha o ciclo aberto no Capítulo 3: enquanto o Quadro 1 indicava onde cada requisito seria atendido pelo projeto, o Quadro 4 relaciona cada um à evidência experimental que o comprova neste capítulo. Todos os requisitos, funcionais e não funcionais, foram verificados na prática ao longo da campanha.

Do ponto de vista operacional, o mecanismo substitui uma decisão manual e estática, o limite de taxa, por uma regulação automática e contínua, com garantia de não perda de mensagens. A utilização do banco permanece no patamar escolhido

Quadro 4 – Verificação experimental dos requisitos técnicos.

Código	Requisito	Evidência experimental
RF1	Regulação da CPU em malha fechada	Seguimento e rejeição nos dois regimes (Figuras 26 e 27; Tabelas 4, 5 e 6)
RF2	Entrega sem perda de mensagens	Nenhuma perda em nenhum ensaio da campanha (Seção 5.6)
RF3	Ajuste de parâmetros em tempo de execução	Varredura de K_i por alterações em execução (Seção 5.5); sessões de sintonia encadeadas (Seção 5.6.5)
RF4	Estratégia intercambiável; malha aberta e troca em execução no laboratório	Identificação com o controlador de malha aberta (Seção 5.4); comparação alternando PI, PI+FF e Smith na mesma sessão (Seção 5.6.2)
RF5	Fila, destino e métricas plugáveis	Adaptadores Kafka, SQS, memória e constante; MySQL e HTTP; Prometheus, CloudWatch e exportador (Seção 5.1)
RNF1	Observabilidade em tempo real	Supervisório e interface de operação (Figuras 15 e 19)
RNF2	Reprodutibilidade dos experimentos	Ambiente completo em um comando (Seção 5.2); pasta de artefatos e metadados por ensaio (Seção 5.3)
RNF3	Comunicação criptografada	Implantação remota com TLS na aplicação e no banco (Seção 5.2)
RNF4	Compatibilidade com serviços de nuvem	Adaptadores SQS e CloudWatch; implantação automatizada em nuvem (Seções 5.1 e 5.2)

Fonte: Elaborado pelo autor (2026).

mesmo sob perturbações, o que reduz o risco dos episódios de saturação descritos no Capítulo 3 e, com eles, os incidentes e acionamentos de plantão associados. Nos períodos de folga, a mesma malha eleva a drenagem e encurta as tarefas em lote, recuperando a capacidade que um limite conservador desperdiçaria.

Fechado o desenvolvimento, o capítulo final sintetiza o trabalho, discute seus limites e aponta as direções de continuidade.

6 CONCLUSÃO

Este trabalho tratou a saturação de serviços de destino por consumidores de filas de mensagens como um problema de controle em malha fechada. Partiu-se de um cenário recorrente em sistemas de *software* em produção: tarefas de atualização em massa que disputam o banco de dados com o tráfego regular da aplicação e, quando consumidas sem regulação, o levam à saturação, com perda de dados, latência e incidentes. A solução desenvolvida interpõe uma fila entre o produtor e o banco e regula a taxa de drenagem com um controlador discreto, mantendo a utilização de CPU do banco em torno de um valor de referência configurável.

O sistema foi construído em Go, com arquitetura que isola o núcleo de controle das tecnologias de infraestrutura, integrando-se a Apache Kafka, AWS SQS, MySQL, Prometheus e AWS CloudWatch por adaptadores. A planta foi identificada por ensaios de resposta ao degrau em malha aberta; os controladores foram sintonizados a partir do modelo obtido e comparados em malha fechada, com métricas de desempenho calculadas sobre as séries medidas. Em todos os ensaios, a regulação manteve a CPU no patamar desejado, rejeitou perturbações externas de carga e não perdeu mensagens.

Em relação aos objetivos traçados na Introdução, os cinco foram atingidos: o sistema de *software* foi construído com o núcleo isolado e as três dependências intercambiáveis, incluindo serviços de nuvem; o ferramental de experimentação foi construído e sustentou toda a campanha experimental; a planta foi modelada e identificada sob diferentes condições de medição; as estratégias de controle foram sintonizadas e comparadas experimentalmente nos diferentes regimes; e a prova de conceito foi validada em malha fechada com critérios quantitativos. Dois achados técnicos não estavam previstos no plano original. O primeiro: o atraso dominante da malha não vem do banco de dados, que responde em segundos, mas da cadência com que as métricas ficam disponíveis, e escala com ela; encurtar a cadência de medição melhora o desempenho mais do que qualquer sofisticação do controlador. O segundo: as estruturas mais sofisticadas não superaram as simples; o Preditor de Smith, candidato natural ao regime dominado por atraso, não trouxe vantagem porque a cadeia de medição espalha o atraso com um perfil que nenhuma estrutura fixa de compensação representa, enquanto a ação *feedforward* de referência, muito mais simples, resolveu a partida lenta sem custo algum na rejeição. As duas constatações sustentam a recomendação final do trabalho: o PI com *anti-windup* condicional, filtro de *setpoint* e *feedforward* de referência como escolha robusta para os regimes de produção estudados.

O trabalho tem limitações que delimitam a validade dos resultados. Os ensaios usaram carga homogênea, com custo semelhante por operação. A condição é representativa do caso de uso alvo, já que migrações repetem uma mesma consulta variando

parâmetros, mas é também um pressuposto dos resultados: o ganho da malha é definido para uma classe de custo, e uma fila que intercale operações muito pesadas entre operações leves muda o ganho da planta a cada mensagem, degradando a regulação. Esse cenário heterogêneo não é coberto pelos resultados apresentados. A malha regula exclusivamente a utilização de CPU: outros gargalos do banco, como disco ou conexões, não são observados. E o acúmulo da fila, embora observável, não é controlado, o que pressupõe produtores finitos e filas dimensionadas para o pico.

Dessas limitações saem as direções naturais de continuidade:

- a) **conectores para cargas de trabalho emergentes**: um alvo particularmente atual são os servidores de modelos de linguagem e de inteligência artificial, em que cada requisição (*prompt*) tem custo computacional alto e variável; regular a taxa de envio pela carga medida da máquina de inferência é o mesmo problema deste trabalho com outra planta;
- b) **múltiplas filas com prioridades e classes de custo**: estender o orquestrador para consumir de várias filas com prioridades e pesos distintos, tanto para repartir a capacidade regulada entre tarefas de urgências diferentes quanto para separar operações por classe de custo (uma fila por tipo de consulta, cada qual com seu peso), a resposta direta à limitação de cargas heterogêneas apontada no Capítulo 3;
- c) **canal de retorno por mensagem**: acompanhar o resultado individual das operações entregues (sucesso, erro, latência) e devolver a resposta de cada uma (o resultado de uma consulta, a saída de um modelo de inteligência artificial) a um servidor de retorno na aplicação de origem, de forma desacoplada do fluxo principal, o que habilita usos em que o produtor precisa da resposta, e não apenas da execução;
- d) **compensação preditiva da perturbação**: o tráfego regular de muitas aplicações segue padrões diários conhecidos (madrugada ociosa, pico no início da manhã em aplicações de alcance nacional); com o histórico de métricas já coletado, é possível prever a perturbação em algum grau e antecipá-la no controlador, na forma de uma ação *feedforward* de perturbação prevista;
- e) **compensação de atraso com preditor filtrado**: ensaiar o Preditor de Smith filtrado (NORMEY-RICO; CAMACHO, 2007), que acrescenta ao preditor um filtro de robustez contra erros de modelo, sobre um provedor de métricas de cadência fixa e conhecida, o cenário em que o atraso efetivo se aproxima de um valor nominal único e as estruturas com modelo embarcado voltam a ser candidatas;
- f) **supervisão por alarmes externos**: sistemas de monitoração de produção emitem alarmes de alto uso de recursos (gerenciadores de alertas do Pro-

metheus, alarmes do CloudWatch e serviços equivalentes); esses eventos podem servir de entrada supervisória para a malha, acionando um modo conservador que reduz o *setpoint*, os ganhos ou o limite de drenagem enquanto o alarme persistir. A interface de ajuste em tempo de execução já oferece o ponto de integração; o passo futuro é formalizar a camada supervisória que combina a regulação contínua com esses eventos discretos, nos moldes da teoria de sistemas a eventos discretos;

- g) **outras variáveis de processo:** regular métricas de qualidade de serviço, como latência ou taxa de erros, isoladas ou combinadas com a CPU;
- h) **adaptação e sintonia automática de ganhos:** automatizar a identificação e a sintonia na ativação do sistema (executando o ensaio de degrau e o cálculo de ganhos como uma operação única) e, além disso, reestimar o ganho da planta continuamente em operação, a partir da relação drenagem–CPU observada, adaptando os ganhos sob carga de composição variável.

Para a prática de engenharia, o resultado é uma ferramenta que substitui um limite de taxa escolhido manualmente por uma regulação automática com garantias explícitas, pronta para as tecnologias comuns de produção. Para a formação em controle e automação, o trabalho ilustra um movimento de mão dupla: as ferramentas clássicas de identificação e controle de processos resolvem um problema real de infraestrutura de *software*, e o *software*, por sua vez, expõe com clareza os limites que a teoria prevê, como o papel do atraso de medição no desempenho alcançável de qualquer malha. Sob essa lente, a infraestrutura de *software* em produção é um conjunto de processos dinâmicos a controlar, com sensores, atuadores e perturbações bem definidos: um campo de aplicação ainda pouco ocupado por engenheiros de controle, e que este trabalho procura demonstrar ser acessível às ferramentas que a profissão já domina.

REFERÊNCIAS

AGUIRRE, Luis Antonio. **Introdução à Identificação de Sistemas: técnicas lineares e não lineares aplicadas a sistemas reais**. 4. ed. Belo Horizonte: Editora UFMG, 2015.

APACHE SOFTWARE FOUNDATION. **Apache Kafka Documentation**. 2026.
Disponível em: <https://kafka.apache.org/documentation/>. Acesso em: 12 jul. 2026.

ÅSTRÖM, Karl Johan; HÄGGLUND, Tore. **Advanced PID Control**. Research Triangle Park: ISA – The Instrumentation, Systems, and Automation Society, 2006.

BEYER, Betsy; JONES, Chris; PETOFF, Jennifer; MURPHY, Niall Richard (Ed.). **Engenharia de Confiabilidade do Google: como o Google administra seus sistemas de produção**. São Paulo: Novatec, 2016.

BOTROS, Silvia; TINLEY, Jeremy. **High Performance MySQL: Proven Strategies for Operating at Scale**. 4. ed. Sebastopol: O'Reilly Media, 2021.

COCKBURN, Alistair. **Hexagonal Architecture (Ports and Adapters)**. 2005.
Disponível em: <https://alistair.cockburn.us/hexagonal-architecture/>. Acesso em: 12 jul. 2026.

EVANS, Eric. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Boston: Addison-Wesley, 2003.

FOWLER, Susan J. **Microsserviços prontos para produção: construindo sistemas padronizados em uma organização de engenharia de software**. São Paulo: Novatec, 2017.

FRANKLIN, Gene F.; POWELL, J. David; WORKMAN, Michael L. **Digital Control of Dynamic Systems**. 3. ed. Menlo Park: Addison-Wesley, 1998.

GANAPATHI, Archana; KUNO, Harumi; DAYAL, Umeshwar; WIENER, Janet L.; FOX, Armando; JORDAN, Michael; PATTERSON, David. Predicting multiple metrics for queries: Better decisions enabled by machine learning. *In*: PROCEEDINGS of the 25th IEEE International Conference on Data Engineering (ICDE). [S.l.: s.n.], 2009. P. 592–603.

GOOGLE. **cAdvisor: Analyzes resource usage and performance characteristics of running containers**. 2026. Disponível em: <https://github.com/google/cadvisor>. Acesso em: 13 jul. 2026.

HELLERSTEIN, Joseph L.; DIAO, Yixin; PAREKH, Sujay; TILBURY, Dawn M. **Feedback Control of Computing Systems**. Hoboken: Wiley-IEEE Press, 2004.

HOHPE, Gregor; WOOLF, Bobby. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. Boston: Addison-Wesley, 2003.

KLEPPMANN, Martin. **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems**. Sebastopol: O'Reilly Media, 2017.

KUBERNETES. **Horizontal Pod Autoscaling — Kubernetes Documentation**. 2026. Disponível em: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>. Acesso em: 12 jul. 2026.

LI, Jiexing; KÖNIG, Arnd Christian; NARASAYYA, Vivek; CHAUDHURI, Surajit. Robust estimation of resource consumption for SQL queries using statistical techniques. **Proceedings of the VLDB Endowment**, v. 5, n. 11, p. 1555–1566, 2012.

MAIA, M. R. **Controlador PID-AA: desenvolvimento de novas técnicas de pré-ajuste e de compensação de atraso de transporte**. 1993. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Santa Catarina, Florianópolis.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Boston: Prentice Hall, 2017.

MARTIN, Robert C. **The Clean Architecture**. 2012. Disponível em: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Acesso em: 13 jul. 2026.

NORMEY-RICO, Julio Elias; CAMACHO, Eduardo Fernández. **Control of Dead-time Processes**. London: Springer, 2007.

NYGARD, Michael T. **Release It!: Design and Deploy Production-Ready Software**. 2. ed. Raleigh: Pragmatic Bookshelf, 2018.

OGATA, Katsuhiko. **Engenharia de Controle Moderno**. Tradução: Heloísa Severo. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

ORACLE. **The Optimizer Cost Model — MySQL 8.0 Reference Manual**. 2026. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/cost-model.html>. Acesso em: 12 jul. 2026.

PROMETHEUS. **Prometheus Documentation**. 2026. Disponível em: <https://prometheus.io/docs/>. Acesso em: 12 jul. 2026.

SKOGESTAD, Sigurd. Simple analytic rules for model reduction and PID controller tuning. **Journal of Process Control**, v. 13, n. 4, p. 291–309, 2003.

SMITH, Otto J. M. Closer control of loops with dead time. **Chemical Engineering Progress**, v. 53, n. 5, p. 217–219, 1957.

TANENBAUM, Andrew S.; WETHERALL, David J. **Computer Networks**. 5. ed. Boston: Pearson, 2011.

THE GO AUTHORS. **Package rate — golang.org/x/time/rate**. 2026. Disponível em: <https://pkg.go.dev/golang.org/x/time/rate>. Acesso em: 13 jul. 2026.

UBER TECHNOLOGIES. **Fx: A dependency injection system for Go**. 2026. Disponível em: <https://uber-go.github.io/fx/>. Acesso em: 13 jul. 2026.

WU, Wentao; CHI, Yun; ZHU, Shenghuo; TATEMURA, Junichi; HACIGÜMÜŞ, Hakan; NAUGHTON, Jeffrey F. Predicting query execution time: Are optimizer cost models really unusable? *In*: PROCEEDINGS of the 29th IEEE International Conference on Data Engineering (ICDE). [S.l.: s.n.], 2013.