
Universidade Federal de Pernambuco
Centro de Informática

Bacharelado em Ciência da Computação

# Avaliação Comparativa de Controladores para Adaptação do RabbitMQ

Matheus Alves Almeida

Trabalho de Graduação

Recife
09/2023

Universidade Federal de Pernambuco
Centro de Informática

Matheus Alves Almeida

# Avaliação Comparativa de Controladores para Adaptação do RabbitMQ

*Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.*

Orientador: *Prof. Nelson Souto Rosa*

Recife
09/2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Almeida, Matheus Alves.
Avaliação Comparativa de Controladores para Adaptação do RabbitMQ / Matheus Alves Almeida. - Recife, 2023.
32 p : il., tab.

Orientador(a): Nelson Souto Rosa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2023.

1. Teoria do Controle. 2. Sistemas Adaptativos. 3. RabbitMQ. I. Rosa, Nelson Souto. (Orientação). II. Título.

000 CDD (22.ed.)

**MATHEUS ALVES ALMEIDA**

# AVALIAÇÃO COMPARATIVA DE CONTROLADORES PARA ADAPTAÇÃO DO RABBITMQ

Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Data de Aprovação: 06/09/2023

Nota: __

BANCA EXAMINADORA:

_____
Prof. Dr. Nelson Souto Rosa (Orientador)
Centro de Informática - UFPE

_____
Prof. Dr. Carlos André Guimarães Ferraz (1º Titular)
Centro de Informática - UFPE

Recife
09/2023

*Dedico este trabalho à minha família, que não teve a oportunidade de ter uma educação de qualidade como eu tive. Eles trabalharam muito para garantir que eu chegasse aqui, mesmo que isso significasse que eles não pudessem ter a mesma oportunidade.*

# Agradecimentos

Agradeço primeiramente a Deus, por me dar a oportunidade de estudar e realizar este trabalho.

Aos meus pais, que sempre me incentivaram a estudar e me deram todo apoio para isso. Além de sempre fazerem questão de (tentar) entender o que eu faço da vida e serem mais presentes na minha jornada. Amo muito vocês!

Ao meu irmão Érik. Apesar das brigas e discussões que todos os irmãos têm, eu sempre pude contar com você.

Às minhas avós e meus avôs, que me ensinaram os valores mais importantes da vida.

Minha tia, Raquel, que foi uma segunda mãe para mim, que me deu bastante suporte, incentivo e puxões de orelha quando necessário.

A minha namorada Lívia Millena, que com muito amor e paciência me deu suporte emocional nessa fase complicada da vida.

Ao meu orientador, Nelson, que me guiou durante a reta final da graduação, minha gratidão é imensa.

Aos meus amigos, que compartilharam comigo experiências de vida.

Aos meus professores do IFPE, que me ensinaram e me orientaram durante todo o meu processo de formação e deram o pontapé inicial na minha jornada acadêmica.

Agradeço a todos que de alguma forma contribuíram para a realização deste trabalho.

Sem vocês, nada disso seria possível.

Obrigado!

*Feedback is a central feature of life. The process of feedback governs how we grow, respond to stress and challenge, and regulate factors such as body temperature, blood pressure, and cholesterol level. The mechanisms operate at every level, from the interaction of proteins in cells to the interaction of organisms in complex ecologies.*

—M. B. HOAGLAND E B. DODSON, *THE WAY LIFE WORKS* (1995)

# Resumo

A crescente demanda por aplicações de execução ininterrupta tem levado ao desenvolvimento de sistemas que adaptam o seu funcionamento em tempo de execução, sem uma parada completa do mesmo. Neste contexto de sistemas adaptativos, a Teoria do Controle desempenha um papel fundamental ao fornecer métodos e técnicas para o projeto e a implementação de controladores capazes de ajustar o comportamento do software de acordo com as condições do ambiente onde ele executa. O presente trabalho descreve um *benchmarking* realizado entre vários controladores atuando ao lado do consumidor de um serviço de mensageria amplamente utilizado, RabbitMQ. O objetivo do controlador é regular o fluxo de mensagens entre o serviço de mensageria e os consumidores. Na prática, em tempo de execução, o valor de uma variável chamada *prefetch count* é ajustado para manter a taxa de recebimento de mensagens próxima a um valor pré-determinado. Desta forma, a utilização dos controladores evita a sobrecarga dos consumidores (em situações de alta demanda) ou leva a um funcionamento otimizado dos mesmos em situações de baixa demanda.

**Palavras-chave:** Teoria do Controle, Sistemas Adaptativos, RabbitMQ

# Abstract

The growing demand for uninterrupted applications has led to the development of systems that adapt their operation at runtime, without a complete stop. In this context of adaptive systems, Control Theory plays a fundamental role by providing methods and techniques for the design and implementation of controllers capable of adjusting the behavior of the software according to the conditions of the environment in which it executes. This work describes a benchmarking performed between several controllers acting alongside the consumer of a widely used messaging service, RabbitMQ. The goal of the controller is to regulate the flow of messages between the messaging service and the consumers. In practice, at runtime, the value of a variable called prefetch count is adjusted to keep the message reception rate close to a predetermined value. In this way, the use of controllers avoids overloading consumers (in high demand situations) or leads to an optimized operation of the same in low demand situations.

**Keywords:** Control Theory, Adaptive Systems, RabbitMQ

# Sumário

# Lista de Figuras

# Lista de Tabelas

CAPÍTULO 1

# Introdução

## 1.1 Contexto e Motivação

Na última década houve um aumento significativo no número de usuários online no mundo. Dados do *World Internet Users Statistic* [5] de 2019 mostram um crescimento de 1392% desde o ano 2000. Acompanhando esse número, também há demanda por aplicações online que executam de forma ininterrupta. Garantir a disponibilidade de um sistema inclui lidar com diversos desafios, como tempo de inatividade, escalabilidade, balanceamento de carga, redundância de servidores, manutenção, entre outros. Observa-se que problemas relacionados ao hardware não são mais tão comuns. Em 1991, Jim Gray [3] dizia que problemas de hardware são relativamente menores em relação a problemas com software. Hoje em dia, devido à evolução dos processadores, essa afirmação de Jim Gray continua legítima. Há diversos problemas que envolvem esses desafios citados, dentre eles, o problema de adaptação.

Em relação à adaptação, espera-se que os sistemas continuem executando normalmente após mudanças e variações recursos disponíveis ou no ambiente. Neste sentido, o ambiente inclui desde interações com usuários até infraestrutura disponível para que os sistemas operem. Danny Weyns [17] define que sistemas adaptativos são aqueles que conseguem usar informações disponíveis para resolver problemas sobre si baseados em seus objetivos, seja se reconfigurando ou se autoajustando para se adaptar à necessidade atual.

Nesse cenário dinâmico, onde o ambiente e os recursos estão em constante mudança, a Teoria de Controle [6] fornece um enquadramento conceitual que apoia a funcionalidade adaptativa dos sistemas ao permitir ajustes em resposta às mudanças ambientais e de recursos. A Teoria de Controle utiliza uma comparação contínua de um valor de referência (meta) e a saída do sistema, i.e., calcula um erro. Então, o controlador atua para minimizar este erro. Se o erro for negativo (sistema acima da meta) o controlador atua para diminuir a saída. Da mesma forma, se o erro for positivo, o controlador atua para aumentar a saída.

## 1.2 Problema

Serviços de mensageria são uma forma comum de comunicação entre componentes de um sistema distribuído [2]. Eles permitem que mensagens sejam enviadas de um componente para outro de forma assíncrona, o que pode tornar os sistemas mais tolerantes a falhas. No entanto, os serviços de mensageria também podem ser uma fonte de gargalos de desempenho. Se um consumidor de mensagens estiver consumindo mensagens muito lentamente, ele pode ser inundado por mensagens pelo serviço de mensageria. Atualmente, existem vários serviços

de mensageria disponíveis, cada um com suas propriedades específicas, mas que herdam essas características gerais de um serviço de mensageria. Alguns exemplos de serviços comumente usados são: Apache Kafka[1], Amazon SQS[2], RabbitMQ[3] e Google Cloud Pub/Sub[4].

Em um sistema que utiliza o RabbitMQ, por exemplo, a Teoria de Controle pode ser usada para otimizar o PC (*prefetch count*) do cliente RabbitMQ [14]. A variável PC controla o número de mensagens que o serviço de mensageria (*broker*) envia para o consumidor sem ter recebido um *acknowledgment* como resposta [1]. Um valor pequeno para essa variável (e.g., PC=1) pode prejudicar o desempenho, pois o *broker* precisa esperar por um *ack* individual para cada mensagem enviada. Enquanto um valor muito grande também pode derrubar um consumidor caso ele não tenha a capacidade de processar as mensagens recebidas do *broker* na taxa em que elas são enviadas. Portanto, otimizar o valor dessa variável otimiza o desempenho do sistema.

Esse projeto apresenta um estudo comparativo de diferentes controladores e métodos de *tuning* disponíveis na Teoria do Controle atuando para controlar o valor da variável PC durante a execução de *subscribers* RabbitMQ. A contribuição desse projeto está na análise comparativa dos controladores e métodos de *tuning* utilizados para este tipo de problema.

## 1.3 Trabalhos relacionados

Nesta seção, apresentamos uma revisão sobre o uso de controladores em sistemas distribuídos.

### 1.3.1 Uso de controladores em sistemas distribuídos

A utilização de controladores em sistemas distribuídos tem sido uma área ativa de pesquisa. Stankovic et al. [16] exploram o desenvolvimento de controladores baseados na Teoria do Controle para permitir que um projetista de sistemas distribuídos possa definir o comportamento desejado de adaptação de um sistema. O estudo mostrou um resultado excelente, considerando os critérios estabelecidos pelos autores. Os critérios usados para avaliar a atuação dos controladores foram estabilidade, *overshoot* (sobrepassagem) e tempo de acomodação.

### 1.3.2 Uso de controladores para adaptação de um serviço de mensageria

Embora o uso de controladores em sistemas computacionais esteja se tornando cada vez mais comum, o uso para controlar serviços de mensageria é bem recente. Nessa linha, há dois trabalhos [14, 13] que introduzem o uso de controladores atuando em serviços de mensageria. Estes estudos apresentam uma análise do uso de alguns controladores. O presente projeto vem, então, como uma extensão desses estudos já iniciados, com a utilização de novos controladores e métodos de *tuning* nas análises realizadas.

[1] https://kafka.apache.org/
[2] https://aws.amazon.com/pt/sqs/
[3] https://www.rabbitmq.com/
[4] https://cloud.google.com/pubsub

## 1.4 Objetivos

O objetivo deste trabalho é realizar um estudo comparativo entre diversos controladores da Teoria do Controle [6] visando controlar a taxa de recebimento de mensagens de um consumidor de uma fila do RabbitMQ em tempo de execução. Esses controladores terão a finalidade de manter essa taxa em um valor previamente definido, por meio da manipulação da variável PC (*prefetch count*).

## 1.5 Estrutura do Documento

O restante dos capítulos estão estruturados a seguinte forma: Capítulo 2 apresenta uma revisão da literatura utilizada para o desenvolvimento do projeto, abordando temas como sistemas adaptativos, teoria do controle e uma breve explicação sobre o funcionamento de serviços de mensageria como o RabbitMQ. Em seguida, no Capítulo 3, será apresentada a metodologia do projeto, o que compreende uma descrição do ambiente onde os experimentos foram executados e detalhes dos controladores utilizados. No Capítulo 4, serão apresentados os resultados dos experimentos realizados, bem como uma análise comparativa dos mesmos. Para finalizar, no Capítulo 5 serão apresentadas as conclusões do trabalho, ressaltando as principais contribuições e direções futuras para o projeto.

CAPÍTULO 2

# Fundamentação Teórica

Este capítulo introduz os conceitos básicos necessários ao entendimento do trabalho: sistemas adaptativos, Teoria de Controle e RabbitMQ.

## 2.1 Sistemas adaptativos

*Sistemas adaptativos* [7] são sistemas que mudam o seu comportamento como forma de responder às mudanças no ambiente em que ele está atuando. Seja essa mudança relacionada à disponibilidade de recursos utilizados, às condições externas ao ambiente e até as alterações em seu próprio objetivo como sistema.

Por exemplo, o funcionamento de um ar-condicionado possui características de um sistema adaptativo. Um ar-condicionado é projetado para manter a temperatura ambiente em um nível desejado, mesmo quando as condições externas e internas mudam. O ar-condicionado possui sensores que monitoram a temperatura do ambiente e enviam essas informações para o sistema de controle. Com base nesses dados, o sistema de controle compara a temperatura atual com a temperatura desejada definida pelo usuário. Se a temperatura ambiente estiver acima da temperatura desejada, o sistema de controle ativará mecanismos destinados a resfriar o ambiente. Por outro lado, se a temperatura estiver abaixo do valor desejado, o sistema de controle poderá desligar esses mecanismos ou reduzir o seu funcionamento.

Importante notar que esses ajustes dependem da diferença entre a temperatura ambiente (entrada do sistema) e a temperatura desejada (ponto de referência). Essa resposta dinâmica do ar-condicionado às mudanças nas condições ambientais, como variações de temperatura, ilustra a capacidade dos sistemas adaptativos de responderem de forma apropriada às alterações do ambiente em que estão atuando.

Na Figura 2.1, são apresentados elementos essenciais para o funcionamento de um sistema adaptativo. Esses elementos incluem o ambiente, que representa as partes que interagem com o sistema adaptativo, onde é possível observar e medir os efeitos das saídas do sistema. O ambiente pode ser tanto físico quanto virtual [15].

O sistema gerenciado é composto pelo código da aplicação responsável por executar tarefas e funções em seu domínio. Sua função é avaliar o ambiente e aplicar as ações calculadas para atingir as metas de adaptação. Essas metas de adaptação representam as preocupações específicas do sistema de gerenciamento em relação ao comportamento e desempenho do sistema gerenciado, e podem abranger diversas categorias, como autoconfiguração e auto-otimização [15].

Por fim, o sistema de gerenciamento atua como o orquestrador do sistema gerenciado e de-

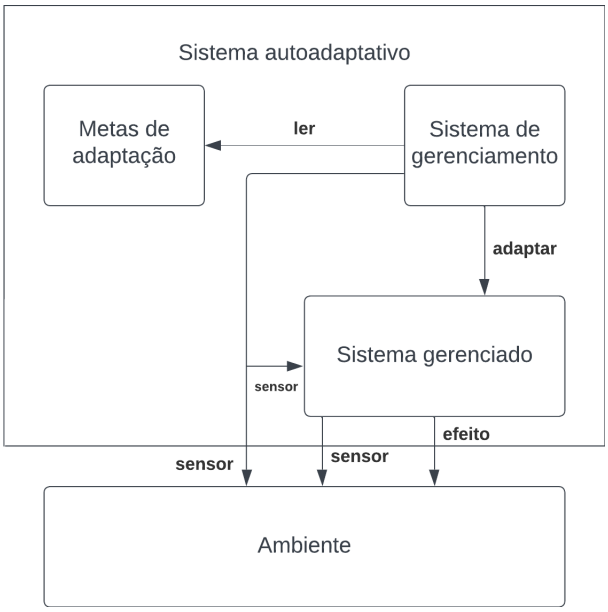


**Figura 2.1** Visão geral de um sistema adaptativo [15].

sempenha um papel fundamental na governança das regras de adaptação, sempre considerando as metas estabelecidas [15]. Sensores presentes no software e no sistema adaptativo permitem a compreensão do impacto das ações de adaptação no ambiente. Esses elementos colaboram para permitir que o sistema se ajuste às condições do ambiente de maneira eficaz.

## 2.2 Teoria do Controle

A *Teoria do Controle* [7, 6] é um conceito da engenharia, que consiste em controlar a saída de sistemas que se comportam de forma dinâmica, i.e., sistemas adaptativos. A Figura 2.2 apresenta os principais componentes de um sistema de controle com *feedback*.

O primeiro elemento importante é o *setpoint*, que compreende no objetivo que se quer chegar com a adaptação. O *setpoint* é usado como entrada no controlador juntamente com um erro, calculado através da subtração do último *output* do *setpoint*, como mostra a Equação 2.1. O controlador, ao receber o erro, atua para minimizá-lo. Após isso, um sinal de controle é enviado para o software que está sendo adaptado, que por sua vez está sujeito a perturbações do ambiente que se encontra e dos recursos que utiliza. Então, todo o processo é repetido.

### 2.2.1 Erro e ações corretivas

Um conceito importante é o conceito de erro, que consiste basicamente no valor resultante da subtração entre a saída do sistema (y) e o *setpoint* (r) [6, 15], como mostrado na Equação 2.1. Então, a partir desse valor, é possível calcular a ação corretiva para se aproximar do *setpoint* desejado. O erro pode ser interpretado da seguinte forma: se o valor for negativo, significa que a saída do sistema está acima do valor desejado, e devem ser feitas mudanças necessárias

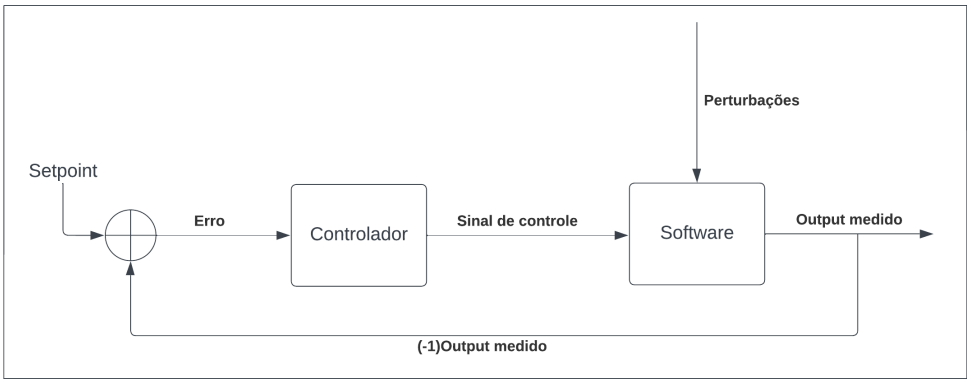


**Figura 2.2** Visão geral de um sistema de controle [15].

para diminuir a saída e vice-versa. Nesse contexto, o erro não só aponta em que direção o erro se encontra (muito acima ou muito abaixo do valor desejado), mas também o valor dessa diferença, o que permite a aplicação de diferentes tipos de ações corretivas.

$$erro = \mathrm{r} - \mathrm{y} \tag{2.1}$$

Sistemas adaptativos possuem algumas desvantagens relacionadas à sua natureza adaptativa. Segundo [7], é importante destacar que sistemas adaptativos apresentam uma propensão natural à oscilação em seu comportamento, o que pode levar a instabilidades indesejadas. Essas oscilações podem ser causadas por diversos fatores, como a presença de atrasos nos processos de *feedback* ou características intrínsecas dos próprios componentes do sistema. Nesse contexto, existem dois conceitos importantes que devem ser levados em consideração no momento de execução de um sistema adaptativo, sendo os conceitos de *lags* e *delays*.

### 2.2.2 *Lags e Delays*

É fundamental destacar que sistemas adaptativos não reagem imediatamente à entrada do sistema [6]. Na prática, quando o sistema recebe a entrada, ocorre o cálculo do erro envolvido e, em seguida, é determinada a ação corretiva necessária. *Lags* referem-se a um tipo de atraso que pode ocorrer no sistema. Eles podem ser causados por diversos fatores, incluindo o tempo de processamento necessário para calcular o erro e determinar a ação apropriada, a velocidade de comunicação entre os componentes do sistema ou a dinâmica intrínseca dos próprios elementos de controle. Por outro lado, *delays* são atrasos temporais previsíveis que ocorrem entre a entrada de um sistema e a saída correspondente. Esses atrasos podem ser intrínsecos ao sistema, como atrasos de transporte em sistemas de comunicação, ou podem ser introduzidos intencionalmente para melhorar a estabilidade do sistema. Esses atrasos podem variar de sistema para sistema, dependendo das especificações e das estratégias de controle implementadas.

### 2.2.3 Ruídos e interferências

Devido à sua dependência das entradas externas ao ambiente, os sistemas adaptativos são altamente suscetíveis a ruídos e perturbações [7]. Os ruídos referem-se a variações não desejadas

e imprevisíveis nas entradas do sistema. Essas variações podem surgir de fontes variadas e sua presença pode afetar diretamente a qualidade da entrada do sistema, comprometendo a precisão do controle e gerando um comportamento indesejado. Por outro lado, as interferências são perturbações deliberadas ou não do sinal de entrada do sistema. Elas podem ser introduzidas por fontes externas ou podem ser resultado de *crosstalk* entre componentes do sistema. As interferências podem levar a um comportamento indesejado, distorcendo a entrada original e prejudicando o desempenho do sistema adaptativo.

### 2.2.4 Controladores

Um controlador é um mecanismo que manipula a entrada de um sistema para atingir uma saída desejada [7, 10]. O controlador recebe um sinal de referência como entrada e gera um sinal de controle como saída. O sinal de controle é então aplicado à entrada do sistema para produzir a saída desejada. Na prática, um controlador computa uma função numérica [6]. Existem vários tipos de controladores, alguns comumente usados são os controladores *On/Off* e *PID* (*Proporcional-Integral-Derivativo*).

Neste trabalho, foram utilizadas variações desses controladores e também mais dois tipos diferentes de controladores, o *AsTAR* [18] e o HPA (*Horizontal Pod Autoscaling*) [8].

#### 2.2.4.1 Controlador *On/Off*

Os controladores *On/Off* [6], também conhecidos como controladores de *bang-bang*, são uma forma simples de controle em que a ação de controle é ativada ou desativada completamente. Esse tipo de controlador é frequentemente empregado em situações em que o sistema precisa ser ligado ou desligado de forma binária, sem considerar ajustes finos ou modulações graduais. Ou seja, se o erro ($e_t$) calculado for negativo, o controlador irá atribuir à variável controlada o maior valor possível ($u_{max}$), e caso o erro seja positivo, o controlador atribuirá à variável controlada o menor valor possível ($u_{min}$), como mostrado na Equação 2.2.

$$u_t = \begin{cases} u_{max}, & \text{se } e_t > 0, \\ u_{min}, & \text{se } e_t < 0, \end{cases} \tag{2.2}$$

Dentro das variações do controlador *On/Off*, duas abordagens comuns são a utilização de *deadzone* e histerese. Essas técnicas adicionam funcionalidades extras ao controlador *On/Off*, permitindo um controle mais refinado e adaptativo em certas situações.

A *deadzone* refere-se a uma faixa de valores em torno do ponto de referência em que nenhuma ação de controle é acionada. Isso significa que, se o erro do sistema estiver dentro dessa faixa, o controlador permanecerá inativo, sem alterar o estado do sistema. Essa abordagem é útil quando há um limiar mínimo necessário para justificar uma ação de controle, evitando a ativação desnecessária do sistema em caso de variações pequenas ou ruídos. A Equação 2.3 descreve as regras de controle de um controlador *On/Off* com *deadzone* ($d$).

$$u_t = \begin{cases} 0, & \text{se } |e_t| < d, \\ u_{max}, & \text{se } e_t > 0, \\ u_{min}, & \text{se } e_t < 0, \end{cases} \tag{2.3}$$

Já a histerese é uma técnica que introduz uma diferença entre os pontos de acionamento e desligamento do controlador. Em vez de ter um único valor de referência para ativar e desativar a ação de controle, o controlador *On/Off* com histerese possui dois valores: um ponto de acionamento superior e um ponto de desligamento inferior. Quando o erro do sistema excede o ponto de acionamento, a ação de controle é ativada. No entanto, a ação de controle só será desativada quando o erro do sistema cair abaixo do ponto de desligamento. Essa abordagem cria uma margem de segurança e evita a oscilação frequente do sistema em torno do ponto de referência. A Equação 2.4 descreve as regras de controle de um controlador *On/Off* com histerese ($h$).

$$u_t = \begin{cases} u_{t-1}, & \text{se } |e_t| < h, \\ u_{max}, & \text{se } e_t > 0, \\ u_{min}, & \text{se } e_t < 0, \end{cases} \tag{2.4}$$

A Figura 2.3 apresenta o comportamento de diferentes tipos de controladores *On/Off*.

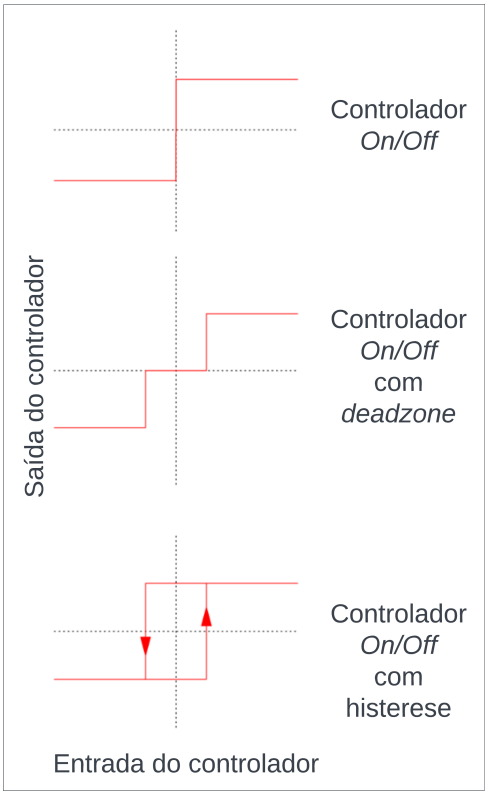


**Figura 2.3** Comportamento dos diferentes tipos de controladores *On/Off*. De cima para baixo estão: *OnOff* Básico, *OnOff* com *deadzone* e *OnOff* com histerese [6].

#### 2.2.4.2 Controlador PID

Os controladores *PID* [6], sigla para *Proporcional-Integral-Derivativo*, são amplamente empregados em sistemas de controle devido à sua eficiência e capacidade de ajuste contínuo. Esses controladores combinam três termos principais: Proporcional, Integral e Derivativo. Cada um desses termos é acompanhado por coeficientes (ganhos) de ajuste que indicam o grau de influência de cada variável. São eles $k_p$ (Proporcional), $k_d$ (Derivativo) e $k_i$ (Integral), conforme representado na Equação 2.7 [6].

O controlador proporcional (P) é o mais simples dos três. Ele responde diretamente ao valor do erro atual. A ação do controlador é proporcional ao erro, ou seja, quanto maior o erro, maior será a ação de controle, conforme a Equação 2.5. O controlador P tem a capacidade de reduzir o erro, mas pode não ser capaz de eliminar completamente o erro em regime permanente. Isso ocorre porque o controlador P não considera o histórico dos erros passados.

$$u_t = k_p e_t, \tag{2.5}$$

Para lidar com o erro em regime permanente, o controlador PI (Proporcional-Integral) adiciona um termo integral ao controlador proporcional, como descrito na Equação 2.6. Esse termo integral leva em conta a soma acumulada dos erros passados ao longo do tempo. A ação integral permite corrigir o erro em regime permanente, ajustando gradualmente a ação de controle. O controlador PI também possui uma resposta mais rápida em relação ao controlador P. No entanto, ele pode resultar em uma sobrecompensação e introduzir oscilações indesejadas no sistema [6].

$$u_t = k_p e_t + k_i \delta_t \sum_{\theta=0}^{t} e_\theta, \tag{2.6}$$

Já o controlador PID (Proporcional-Integral-Derivativo) adiciona um termo derivativo ao controlador PI, conforme a Equação 2.7. O termo derivativo leva em conta a taxa de variação do erro, permitindo que o controlador antecipe as mudanças futuras e ajuste a ação de controle de forma mais precisa. O controlador PID combina os benefícios dos termos proporcional, integral e derivativo, proporcionando uma resposta mais rápida, redução do erro em regime permanente e melhor estabilidade do sistema. Controladores PID são amplamente utilizados em uma variedade de aplicações de controle devido à sua capacidade de lidar com sistemas complexos e variáveis.

$$u_t = k_p e_t + k_i \delta_t \sum_{\theta=0}^{t} e_\theta + \frac{k_d}{\delta_t}(e_t - e_{t-1}), \tag{2.7}$$

Existem variações do controlador PID, cada uma projetada para atender a requisitos específicos de controle. Além do controlador PID padrão, há também o PID com *deadzone*, PID *error square* e PID *incremental form*. Cada uma delas oferece abordagens únicas para o controle, adaptando-se às demandas contextuais.

O controlador PID com *deadzone* introduz a noção de uma "zona morta"(ou *deadzone*), conforme demonstrado na Equação 2.8, em que $d$ representa a largura da *deadzone*. Dentro desta faixa de valores, o controlador não tem nenhuma ação, permanecendo inativo. No entanto,

quando o erro absoluto ($|e_t|$) ultrapassa a *deadzone*, o controlador age conforme os termos P, I e D para corrigir o erro e controlar o sistema.

$$u_t = \begin{cases} 0, & \text{se } |e_t| < d, \\ k_p e_t + k_i \delta_t \sum_{\theta=0}^{t} e_\theta + \frac{k_d}{\delta_t}(e_t - e_{t-1}) & \text{se } |e_t| > d, \end{cases} \tag{2.8}$$

O controlador PID *error square*, representado na Equação 2.9, baseia-se no erro absoluto ($|e_t|$). Este método multiplica o erro absoluto pelo somatório dos termos P, I e D, resultando em um controle que enfatiza a correção do erro, priorizando erros maiores.

$$u_t = |e_t| * (k_p e_t + k_i \delta_t \sum_{\theta=0}^{t} e_\theta + \frac{k_d}{\delta_t}(e_t - e_{t-1})), \tag{2.9}$$

Por outro lado, o controlador PID em sua forma incremental, conforme demonstrado na Equação 2.10, calcula a variação incremental ($\Delta u_t$) do sinal de controle. Isso é obtido considerando as diferenças entre os valores atuais e anteriores do erro, com cada termo P, I e D ponderado pelos coeficientes de ajuste apropriados.

$$u_t = u_{t-1} + \Delta u_t, \tag{2.10}$$

onde $\Delta u_t = k_p(e_t - e_{t-1}) + k_i e_t \delta_t + k_d \frac{e_t - 2e_{t-1} + e_{t-2}}{\delta_t}$.

#### 2.2.4.3 Controlador *AsTAR*

O *AsTAR* [18] é um controlador usado no agendamento de tarefas adaptativas de dispositivos IoT (*Internet of Things*) que usam energia solar. O *AsTAR* é baseado no algoritmo de busca A* [4], que é um algoritmo de busca heurístico. Segundo [18], o *AsTAR* é capaz de alcançar resultados mais eficientes em comparação com algoritmos tradicionais de controle PID e controladores de lógica *fuzzy*. Além disso, o *AsTAR* possui a vantagem de ser capaz de se adaptar dinamicamente às mudanças nas condições ambientais, tornando-o uma opção atrativa para aplicações em dispositivos IoT alimentados por energia solar, que apresentam flutuações na disponibilidade de energia. O controlador *AsTAR* utiliza conceitos da teoria de controle para realizar as ações de controle do sistema, mas sua abordagem é baseada em estratégias de busca inteligente, o que o diferencia das abordagens tradicionais de controle. A equação 2.11 descreve o funcionamento básico do controlador do AsTAR. O controlador é responsável por decidir quantos recursos (e.g., energia) devem ser alocados para cada tarefa em cada momento.

$$u_t = \begin{cases} 0, & \text{se } y_t < s, \\ u_{t-1} + 1, & \text{se } y_t < (r-h) \text{ e } y_t > y_{t-1}, \\ u_{t-1}/2, & \text{se } y_t < (r-h) \text{ e } y_t <= y_{t-1}, \\ u_{t-1} - 1, & \text{se } y_t > (r+h) \text{ e } y_t < y_{t-1}, \\ u_{t-1} * 2, & \text{se } y_t > (r+h) \text{ e } y_t >= y_{t-1}, \\ u_t, & \text{caso contrário}, \end{cases} \tag{2.11}$$

Na equação 2.11, $s$ é a taxa de desligamento, $r$ é a taxa ideal (ou meta) e $h$ é a banda de histerese. A taxa de desligamento é a taxa na qual o controlador reduz o número de recursos alocados a uma tarefa quando o nível de energia cai abaixo do limite de segurança ($s$). A taxa ideal é a taxa na qual o controlador deve alocar recursos a uma tarefa para manter o nível de energia no limite de reserva ($r$). A banda de histerese é a faixa de níveis de energia ao redor do limite de reserva ($r$) dentro da qual o controlador não altera o número de recursos alocados a uma tarefa. A banda de histerese ajuda a impedir que o controlador oscile entre aumentar e diminuir o número de recursos alocados a uma tarefa. Isso pode acontecer se o nível de energia flutuar ao redor do limite de reserva ($r$).

#### 2.2.4.4 Controlador HPA

O *Horizontal Pod Autoscaling* (HPA) é um mecanismo do Kubernetes [8] que automaticamente escala horizontalmente um *deployment*. Ele faz isso medindo uma métrica, como a contagem de solicitações por segundo, e ajustando o número de réplicas. O HPA é definido usando um objeto HPA, que é um tipo de recurso Kubernetes. O objeto HPA especifica a métrica a ser medida, o objetivo para a métrica e o número de réplicas do *deployment* a ser escalado. De forma prática, ele atua como definido na Equação 2.12.

$$u_t = \lceil u_{t-1} \cdot \frac{r}{y_t} \rceil \tag{2.12}$$

Onde $u_t$ é o número de réplicas desejadas, $u_{t-1}$ é o número de réplicas atual, $r$ é o valor atual da métrica e $y_t$ é o valor desejado da métrica. No contexto desse projeto, o valor de $u_t$ representa o *prefetch count* atual, enquanto $r$ e $u_{t-1}$ representam o valor atual da taxa de chegada das mensagens e o valor desejado.

### 2.2.5 *Tuning* de controladores

Os controladores PID podem ser melhorados mediante técnicas de *tuning* [6]. Essas técnicas visam atribuir ganhos, que são coeficientes numéricos, às variáveis do controlador: proporcional, integral e derivativo. Cada variável possui um ganho associado e esse ganho pode ser calculado usando-se métodos como Ziegler Nichols [6], Cohen-Coon [6], AMIGO [6] e *Root Locus* [6]. De maneira geral, mudanças nesses ganhos resultam em diferentes impactos na atuação do controlador. Considerando o valor dos ganhos $K_p$, $K_i$ e $K_d$, as seguintes relações podem ser observadas [6]:

- Aumentar *Kp*:
  - Aumenta a velocidade e o ruído, e
  - Diminui a estabilidade.
- Aumentar *Ki*:
  - Diminui a velocidade e a estabilidade,
  - Reduz o ruído,

  - Elimina erros em estado estacionário mais rapidamente, e
  - Aumenta a tendência a oscilar.

- Aumentar *Kd*:
  - Aumenta a velocidade, estabilidade e o ruído.

Cada uma dessas técnicas de *tuning* visa encontrar os valores desses coeficientes de forma que haja um equilíbrio na atuação do controlador e que melhore o seu desempenho. Essas técnicas serão descritas brevemente a seguir.

#### 2.2.5.1 *Root Locus*

A técnica do *Root Locus* [9, 10] é uma abordagem utilizada para ajustar os parâmetros dos controladores PID em sistemas de controle. A técnica consiste em traçar as trajetórias dos polos do sistema em malha fechada no plano complexo, à medida que o ganho do controlador PID varia.

O *Root Locus* é determinado pelo polinômio característico do sistema [9, 10], que é uma equação que descreve o comportamento do sistema. O polinômio característico possui a forma descrita através da Equação 2.13.

$$p(s) = s^n + a_1 s^{n-1} + ... + a_n \tag{2.13}$$

Onde $s$ é a variável complexa e $a_1$, ..., $a_n$ são os parâmetros do sistema. À medida que o ganho do controlador PID varia, os polos do sistema em malha fechada movem-se ao longo do *root locus*. Os polos do sistema determinam a estabilidade e a resposta dinâmica do sistema.

#### 2.2.5.2 Ziegler Nichols

O Ziegler-Nichols [6] um método clássico de *tuning* de controladores PID que envolve identificar os parâmetros do controlador com base na resposta do sistema à uma mudança no *set-point* (referência) ou em uma perturbação. Existem diferentes abordagens dentro do método de Ziegler-Nichols, como o método de sintonia por degrau e por ciclo limite. Em geral, esse método é fácil de implementar e não requer um modelo matemático detalhado do sistema. No entanto, ele pode resultar em controladores com respostas instáveis ou subótimas em algumas situações, sendo necessário ajustes manuais adicionais.

#### 2.2.5.3 Cohen-Coon

O Cohen-Coon [6] é outro método de *tuning* de controladores PID. Ele também utiliza a resposta do sistema à uma perturbação ou mudança no *setpoint* para ajustar os parâmetros do controlador. O método de Cohen-Coon é especialmente adequado para sistemas que apresentam respostas mais lentas ou menos oscilatórias. Ele pode produzir controladores mais suaves e estáveis em comparação com o Ziegler-Nichols.

#### 2.2.5.4 AMIGO

AMIGO (*Approximate M-constrained Integral Gain Optimisation*) é outro método para *tuning* de controladores PID que usa um conjunto de equações para calcular os parâmetros do controlador. Na tabela 2.1 apresenta as equações definidas empriricamente e utililizadas pelo método AMIGO.

Considere que o valor da variável $\tau$ é 1, bem como $T$ é 0,1. $K$ é um valor obtido mediante experimentos.

**Tabela 2.1** Equações para TI, TD e KP

| | Descrição |
|---|---|
| *ti* | Tempo de inicialização $= \left(\frac{0.4\cdot\tau+0.8\cdot \mathrm{T}}{\tau+0.1\cdot \mathrm{T}}\right)\cdot\tau$ |
| *td* | Tempo de desativação $= \frac{0.5\cdot \mathrm{T}}{0.3\cdot\tau+\mathrm{T}}\cdot\tau$ |
| *kp* | Ganho proporcional $= \frac{1}{\mathrm{K}}\cdot\left(0.2+0.45\cdot\frac{\mathrm{T}}{\tau}\right)$ |

Note que estas equações calculam o ganho proporcional, porém não o integral e derivativo. Para estes, são utilizadas as equações descritas na Equação e Equação .

$$ki = \frac{kp}{ti} \tag{2.14}$$

$$kd = kp \cdot td \tag{2.15}$$

## 2.3 Serviços de mensageria

Serviços de mensageria permitem que diferentes sistemas ou aplicações se comuniquem de forma assíncrona, o que é fundamental para garantir a escalabilidade, flexibilidade e confiabilidade de um sistema distribuído. A Figura 2.3 apresenta os componentes essenciais para o serviço de mensageria.

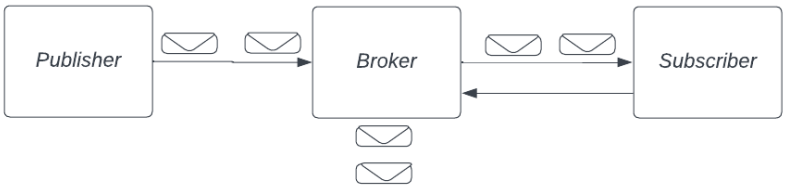


**Figura 2.4** Arquitetura de um serviço de mensageria [12].

Um serviço de mensageria geralmente consiste em um intermediário, conhecido como broker, que gerencia a troca de mensagens entre produtores (*publishers*) e consumidores (*subscribers*) de mensagens.

### 2.3.1 RabbitMQ

Neste projeto, o RabbitMQ [12] foi o serviço de mensageria utilizado. RabbitMQ é um serviço de mensageria de código aberto que implementa o protocolo AMQP (*Advanced Message Queuing Protocol*). Ele oferece recursos como filas, *exchanges* e *bindings*, que permitem que as aplicações distribuídas troquem informações de maneira flexível e confiável. De uma maneira simplificada, o funcionamento desse serviço precisa de quatro componentes principais: o *broker*, a fila, o *publisher* e o *subscriber*.

#### 2.3.1.1 *Broker*

Um broker é um componente central em um sistema de mensageria que gerencia o fluxo de mensagens. Ele atua como um intermediário entre os produtores, que enviam mensagens, e os consumidores, que as recebem. Suas principais responsabilidades incluem: o roteamento de mensagens, armazenamento temporário de mensagens, balanceamento de carga, monitoramento e gerenciamento de filas.

#### 2.3.1.2 Filas, *Publishers* e *Subscribers*

No RabbitMQ [12], as filas desempenham um papel central, atuando como repositórios temporários para as mensagens enviadas pelos *publishers* antes que os *subscribers* as consumam. Essa arquitetura (Figura 2.3) viabiliza a operação independente e assíncrona dos *publishers* e *subscribers*. As aplicações assumem os papéis distintos de remetentes (*publishers*) e destinatários (*subscribers*) das mensagens no ambiente RabbitMQ. Os *publishers* são responsáveis por criar e publicar as mensagens no *broker*, enquanto os *subscribers* têm a função de receber e processar as mensagens distribuídas pelo RabbitMQ. Essa dinâmica ocorre mediante a inscrição dos *subscribers* nas filas do *broker*, onde aguardam a chegada das mensagens.

#### 2.3.1.3 *Prefetch count*

Um parâmetro muito usado para otimizar o consumo de mensagens é o PC (*prefetch count*) [11]. O PC é um valor que representa o número de mensagens enviadas pelo *broker* e que permanecem não reconhecidas, ou seja, que ainda não receberam um *acknowledgment* (*ack*) por parte do *subscriber*. O PC pode ser usado de várias maneiras. O PC pode ajudar a evitar que os *subscribers* fiquem sobrecarregados com muitas mensagens. Se o *prefetch count* for definido para um valor alto, os *subscribers* poderão receber muitas mensagens rapidamente e podem não ser capazes de processá-las todas em tempo hábil. Isso pode levar a mensagens perdidas ou a problemas de desempenho. Por outro lado, quando o valor de PC é pequeno, os *subscribers* recebem um número limitado de mensagens de cada vez. Isso pode ser benéfico em situações em que o processamento de mensagens requer uma quantidade significativa de recursos computacionais ou quando o *subscriber* precisa garantir que todas as mensagens sejam

processadas com alta prioridade. No entanto, um valor muito pequeno de PC pode resultar em subutilização da capacidade de processamento do *subscriber*, i.e., o sistema poderá não estar aproveitando todo o potencial de processamento disponível.

CAPÍTULO 3

# Avaliação Comparativa dos Controladores

Neste capítulo, serão apresentados os passos realizados para a avaliação comparativa dos diferentes controladores. O primeiro passo é a seleção dos controladores que serão avaliados. Em seguida, quando há necessidade de *tuning* (e.g., controladores PID), foram selecionados os métodos de *tuning*. Por fim, serão detalhados os procedimentos de experimentação e análise dos resultados obtidos a partir da aplicação dos controladores selecionados e ajustados.

## 3.1 Seleção de controladores

Neste passo, foram definidos os controladores a serem utilizados na avaliação. Os controladores selecionados foram aqueles comumente utilizados em sistemas computacionais (*OnOff* e PID e suas variantes) e dois novos controladores que não se baseiam no erro (HPA e AsTAR).

## 3.2 *Tuning* dos controladores PID

Nessa seção são apresentadas as etapas para realizar o *tuning* dos controladores *PID*: coleta dos dados de entrada para os algoritmos de cálculo de ganhos e cálculo dos ganhos usando os dados coletados.

### 3.2.1 Experimentos para obtenção dos ganhos

Para coletar os dados de entrada nos métodos de *tuning*, foram realizados os seguintes passos:

- Para o *Root Locus*:
  - Foi atribuído o valor 1 ao PC, e computada uma amostra de 30 execuções do controlador PID com o sistema em *open loop*. Essa amostra continha o valor da taxa de chegada de mensagens no *subscriber*.
  - Foi calculada a média dessa amostra e em seguida atribuído mais 1 ao valor de PC.
  - Isso se repetiu enquanto a média não aumentava mais que 10%.

  O resultado disso foi um *dataset* com duas colunas principais: o valor de PC e a média das 30 amostras para aquele valor.

- Para os demais métodos de *tuning*:

- Foi atribuído o valor 1 ao PC e calculado o valor da taxa para uma única execução.
- Em seguida, foi atribuído o valor 2 ao PC e calculado o valor da taxa para uma única execução.

Após 10 iterações dessa forma, foi montado um *dataset* com o valor de PC e a taxa de chegada de mensagens para aquele valor.

### 3.2.2 Cálculo dos ganhos

Após a coleta dos dados de entrada para as funções de ganho, foram calculados os valores dos coeficientes usando cada uma das técnicas de ganho descritas na Seção 2.2.5. Para cada uma das técnicas, foi obtido o valor dos ganhos $K_p$, $K_i$ e $K_d$.

## 3.3 Realização dos experimentos

Uma vez que foram calculados os ganhos dos controladores PID, foram realizados experimentos para avaliar o desempenho de cada um deles no controle da taxa de recebimento de mensagens do consumidor da fila do RabbitMQ em um valor pré-determinado. Durante os experimentos, foram coletados dados, como a taxa de recebimento de mensagens nos últimos 5 segundos, o valor de referência (*setpoint*) e o valor de saída do controlador. Esses dados foram analisados e comparados para determinar qual controlador obteve o melhor desempenho.

O controle da taxa de recebimento de mensagens foi realizado por meio da manipulação da variável PC, efetuada pelo controlador. Cada controlador colaborou com o *subscriber* na tentativa de manter a taxa de recebimento de mensagens em níveis específicos, a saber, 17.000, 10.000, 20.000, 5.000 e 8.000 mensagens por segundo. Para cada uma dessas metas, foram conduzidas 60 coletas de dados, registrando informações como a taxa de chegada atual das mensagens, a taxa desejada e o valor do *prefetch count*.

## 3.4 Ambiente de execução dos experimentos

Os experimentos foram conduzidos em um ambiente controlado, utilizando dois computadores distintos: um para atuar como *publisher* e outro como *subscriber*. O servidor do RabbitMQ foi instalado e configurado no computador que desempenhou o papel de *subscriber*. Essa configuração permitiu a análise do desempenho do controlador no ambiente real, sem a interferência de gargalos de conexão com outros dispositivos.

Ao separar as funções de *publisher* e *subscriber* em computadores diferentes, foi possível simular um ambiente mais próximo das condições reais de uso. Essa configuração reduziu possíveis interferências e atrasos causados por compartilhamento de recursos ou limitações de conexão, garantindo maior precisão nos resultados obtidos durante os experimentos.

Foi utilizado o RabbitMQ como sistema de mensageria, e os controladores foram implementados e testados utilizando a linguagem Java, na sua versão 17. Todo o código usado para

a implementação dos controladores, experimentos e geração dos gráficos utilizados neste documento está disponível em um repositório do github[1].

## 3.5 Métricas para comparação do controladores

A avaliação dos controladores foi realizada utilizando a métrica estatística RMSE (*Root Mean Squared Error*), conforme apresentado na Equação 3.1, que oferece uma compreensão objetiva do quão eficaz foi o controlador em relação aos valores de referência. Seu uso permitiu uma análise precisa e quantitativa da discrepância entre as saídas do controlador e os valores desejados.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.1}$$

[1]Repositório que contém a implementação dos controladores e o setup do RabbitMQ: https://github.com/MatheusAlvesAlmeida/FeedbackControllers

CAPÍTULO 4

# Resultados

Neste capítulo, serão apresentados os resultados obtidos no experimento de comparação entre diferentes controladores atuando com o *subscriber* do serviço de mensageria RabbitMQ. Os controladores analisados foram: *OnOff* básico, *OnOff* com *deadzone*, *OnOff* com histerese, PID básico, PID com *deadzone*, PID com histeres, HPA e AsTAR.

Como mencionado na Seção 3.5, a métrica usada para a avaliação foi o RMSE. Para os controladores PID, foram utilizadas 4 (quatro) tipos diferentes de técnicas de *tuning*: *AMIGO, Root Locus, Cohen* e *Ziegler-Nichols*.

Cada controlador foi analisado levando em conta duas abordagens diferentes que serão descritas nas próximas seções.

## 4.1 Abordagem com *setpoint* fixo

A primeira abordagem compreendeu em avaliar a atuação de cada controlador considerando uma meta fixa, i.e., o *setpoint* foi mantido constante ao longo do experimento. A Figura 4.1 apresenta graficamente a atuação do controlador HPA. O HPA obteve um bom resultado, com um desvio padrão (*std*) de 1590,376 e um RMSE de 1627,978. Esse desempenho dá-se devido às características do controlador, que age conforme a razão entre o desejado e o valor atual. Dessa forma, o algoritmo promove adaptações mais precisas no valor do *prefetch count*.

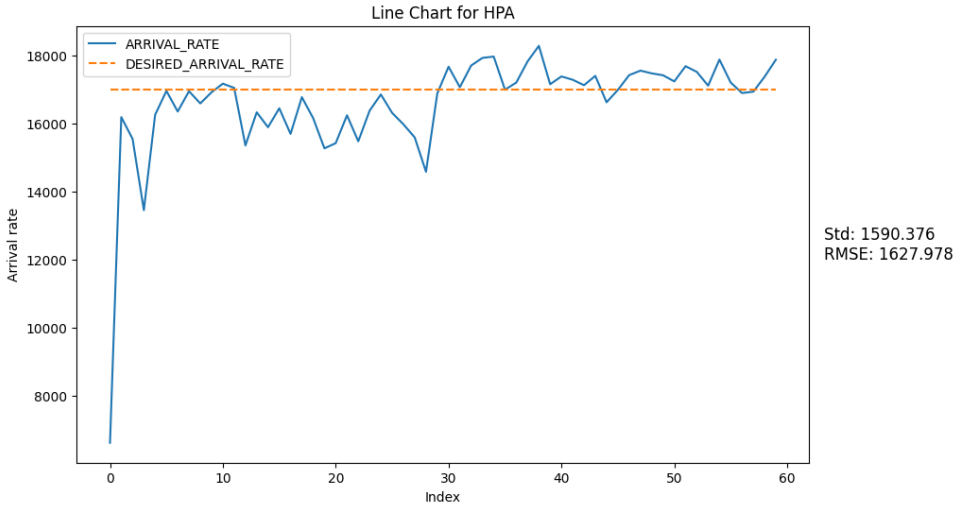


**Figura 4.1** Atuação do controlador HPA com uma meta fixa.

O controlador *incremental* PID com o método de *tuning* AMIGO obteve um desempenho semelhante ao HPA, conforme mostrado na Figura 4.2. Esse resultado pode ser explicado pela sua propriedade de se ajustar rapidamente às perturbações no sistema, o que se mostrou muito eficaz nesse tipo de problema, onde uma pequena alteração no *prefetch count* já provocava uma mudança considerável na taxa de chegada das mensagens.

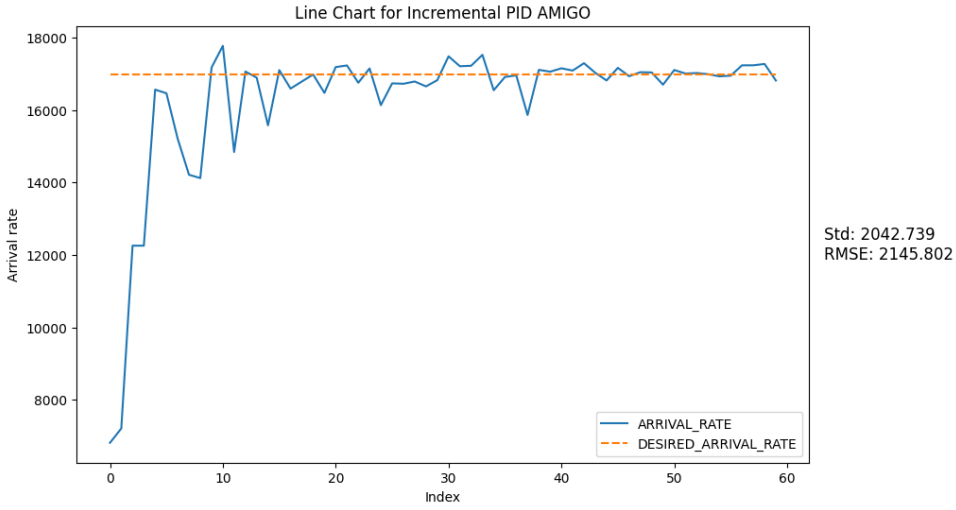


**Figura 4.2** Atuação do controlador PID incremental com método de *tuning* AMIGO e com uma meta fixa.

Em contrapartida, o controlador *On/Off* básico que executa um modelo mais simples de controle, oscilando entre o maior e menor valor possível, apresentou um desempenho inferior aos controladores HPA e PID *incremental*, como mostrado na Figura 4.3, com valores altos para o desvio padrão e RMSE. Estes valores, quando comparados com o HPA, apresentam um aumento de 5177,574 no valor do desvio padrão e 6566,091 no RMSE. Todavia, comparado às variações do controlador *On/Off* (*deadzone* e histerese), o *On/Off* básico obteve o melhor resultado.

O AsTAR que é um controlador do tipo *plug and play*, i.e., dispensa a utilização de métodos de *tuning* para ajustá-lo, apresentou um bom resultado, conforme a Figura 4.4. Quando comparado com outros controladores *plug and play*, o AsTAR possui um desempenho que é inferior apenas ao HPA. Tendo um desempenho melhor que todos os controladores *On/Off*, com um ganho de aproximadamente 44% no RMSE em relação ao controlador *On/Off* básico.

Apesar disso, devido ao AsTAR eventualmente dobrar ou dividir pela metade o valor do PC, algumas perturbações são causadas. Isso explica seu desempenho inferior em relação ao desempenho de controladores como o *deadzone* PID e PID básico, usando os métodos de *tuning* AMIGO e Cohen, respectivamente. Como mostrado na Figura 4.5 e Figura 4.6.

Contudo, o AsTAR não teve um desempenho inferior a todos os controladores PID. A Figura 4.7 que apresenta o resultado do desempenho do controlador PID *error square* com método de *tuning* AMIGO, mostra um resultado inferior ao AsTAR. Onde o AsTAR tem um ganho de aproximadamente 25%.

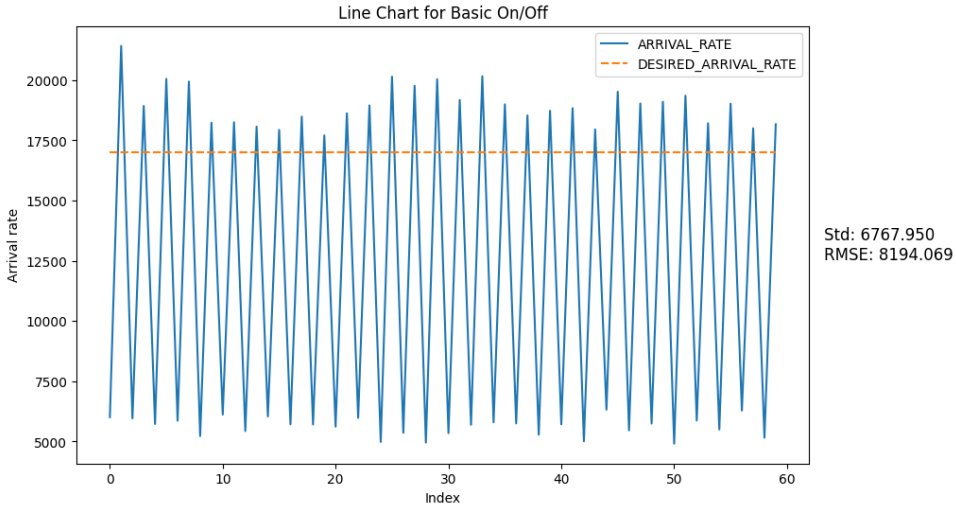


**Figura 4.3** Atuação do controlador *On/Off* com uma meta fixa.

## 4.2 Abordagem com *setpoint* variável

Na segunda abordagem, foi avaliada a atuação do controlador considerando uma meta variável. Neste caso, o valor de referência para o sistema foi variado ao longo do experimento, simulando situações onde a meta a ser alcançada pode mudar com o tempo.

Os resultados apresentados na Figura 4.8 e Figura 4.9 mostraram que o controlador HPA teve o melhor desempenho, com um RMSE de 1767,524. O controlador PID com *deadzone* com *tuning AMIGO* teve o segundo melhor desempenho, com um RMSE de 2138,828.

Diferente do resultado avaliado considerando uma meta fixa, o controlador PID com *deadzone* usando o método de *tuning* AMIGO, obteve um resultado melhor que o controlador *incremental* PID, como mostrado na Figura 4.10. Essa melhora foi de aproximadamente 5,3% no valor do RMSE.

Os outros controladores tiveram um desempenho pior, com RMSEs superiores a 2500.

Dentre os controladores *On/Off*, ao atuar com a meta variável, o controlador *On/Off* básico mostrou-se melhor que os demais controladores, a Figura 4.11 mostra graficamente a atuação do *On/Off* básico e de suas variações. Ele obteve um RMSE de 6261,230. Enquanto o *deadzone On/Off*, o segundo melhor, obteve um valor de RMSE de 9089,209, sendo um dos piores controladores avaliados ao lado do *On/Off* com histerese, conforme mostrado na Figura 4.11.

O *On/Off* básico mostrou um desempenho semelhante ao controlador *error square* PID usando os métodos *Root Locus* e Ziegler-Nichols, com um pequeno ganho no valor do RMSE de 0,09% e 1,28%, respectivamente, a Figura 4.14 apresenta os resultados referentes a atuação do controlador *error square* PID. Os controladores *On/Off*, embora úteis para sistemas mais simples e menos exigentes, demonstraram limitações ao lidar com o problema.

O controlador AsTAR foi o segundo melhor avaliado em relação aos controladores do tipo *plug and play*, repetindo as observações da atuação com meta fixa. Com um ganho no RMSE de aproximadamente 45% em relação ao *On/Off* básico. O HPA foi o melhor avaliado em relação a todos os controladores, ao compará-lo com o AsTAR, o HPA obteve uma diminuição no valor

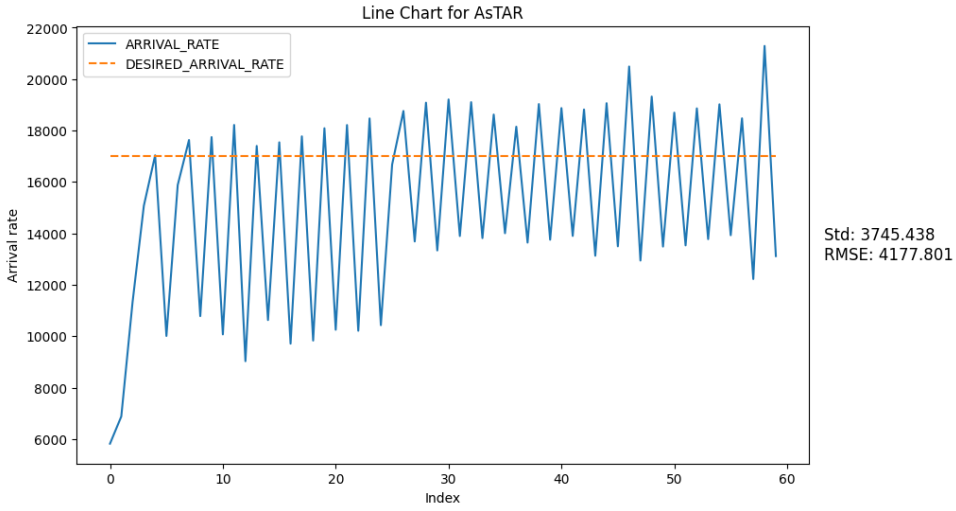


**Figura 4.4** Atuação do controlador AsTAR com uma meta fixa.

do RMSE de aproximadamente 48%. Em relação ao *On/Off* básico, obteve uma diminuição de aproximadamente 71%.

Os controladores PID exibiram uma variação mais ampla de desempenho de acordo com as diferentes técnicas de *tuning* aplicadas. Observou-se que, dependendo da técnica de ajuste específica, os controladores PID alcançaram resultados distintos em termos de precisão e estabilidade do sistema. O método *AMIGO*, por exemplo, demonstrou boa eficácia ao melhorar o desempenho dos controladores PID incremental, básico e com *deadzone*, enquanto outras técnicas como Ziegler-Nichols e *Root Locus* resultaram em resultados variáveis. A Figura 4.13 apresenta o resultado do controlador PID básico, usando os quatro métodos de *tuning* citados.

A Figura 4.14 mostra a atuação dos controladores *error square* PID com diferentes métodos de *tuning*. Ele obteve um resultado inferior aos outros controladores PID, o melhor resultado foi obtido com o método de *tuning* Cohen, com um RMSE de 4233,348, ficando inferior à média de RMSE de todos os controladores analisados.

Os resultados dos controladores atuando com uma meta variável estão resumidos na tabela 4.1.

Os gráficos com os resultados dos demais controladores testados estão nos Apêndices.

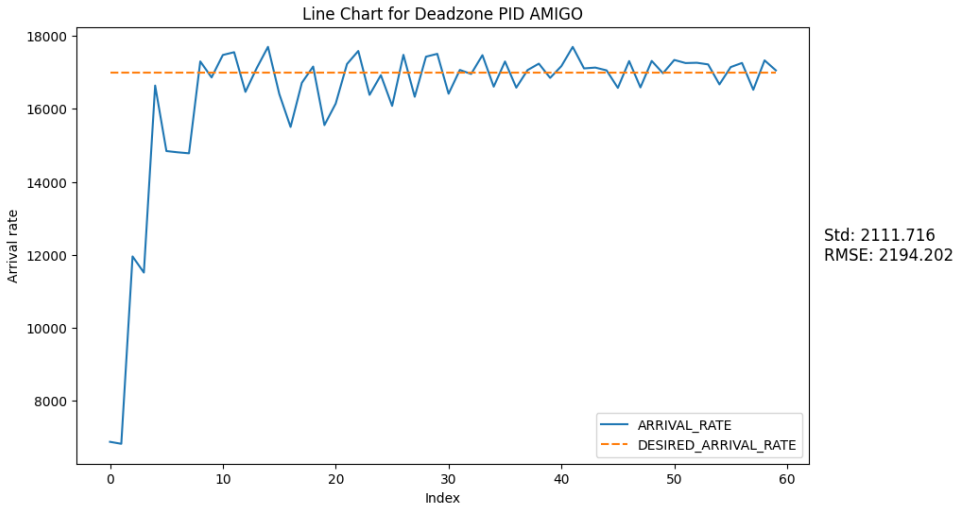


**Figura 4.5** Atuação do controlador *deadzone* PID com método de *tuning* AMIGO e com uma meta fixa.

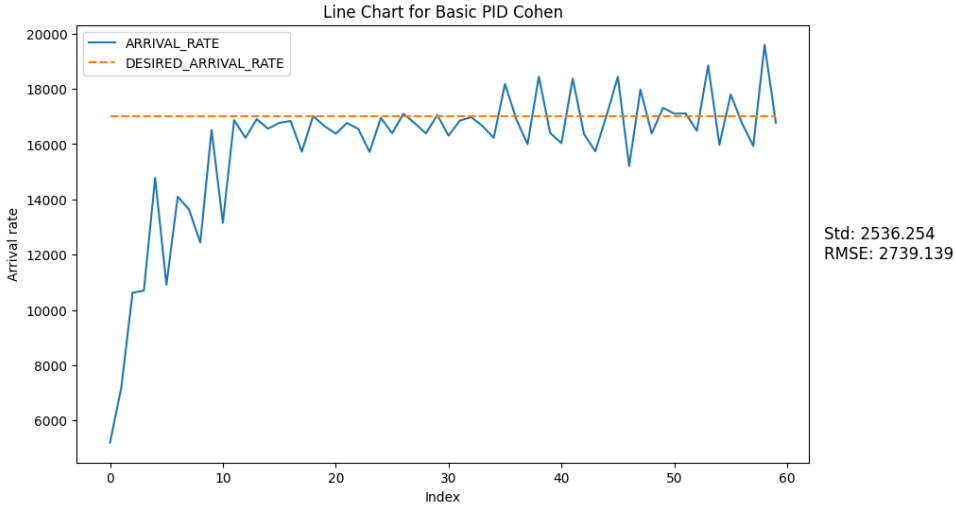


**Figura 4.6** Atuação do controlador PID básico com método de *tuning* Cohen e com uma meta fixa.

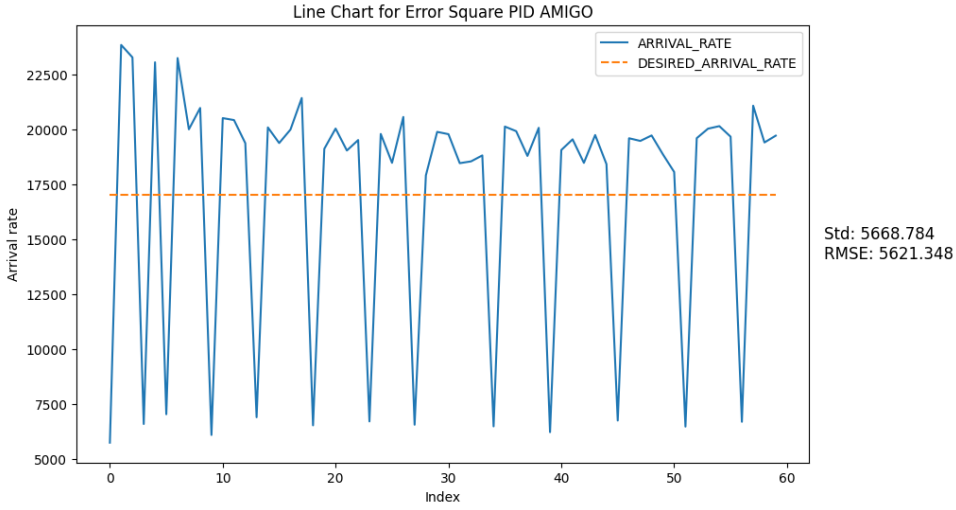


**Figura 4.7** Atuação do controlador PID *error square* com método de *tuning* AMIGO e com uma meta fixa.

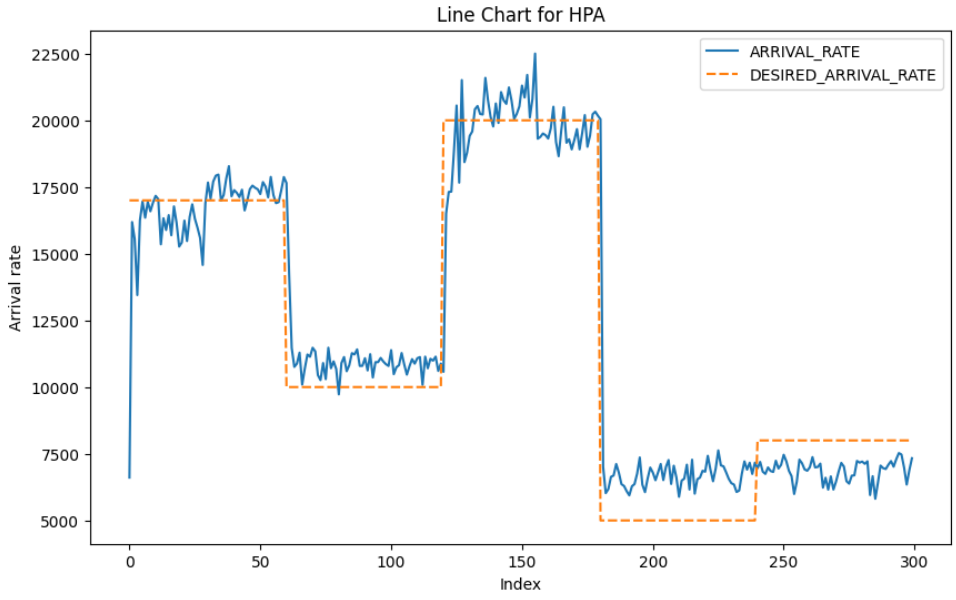


**Figura 4.8** Atuação do controlador HPA com meta variável.

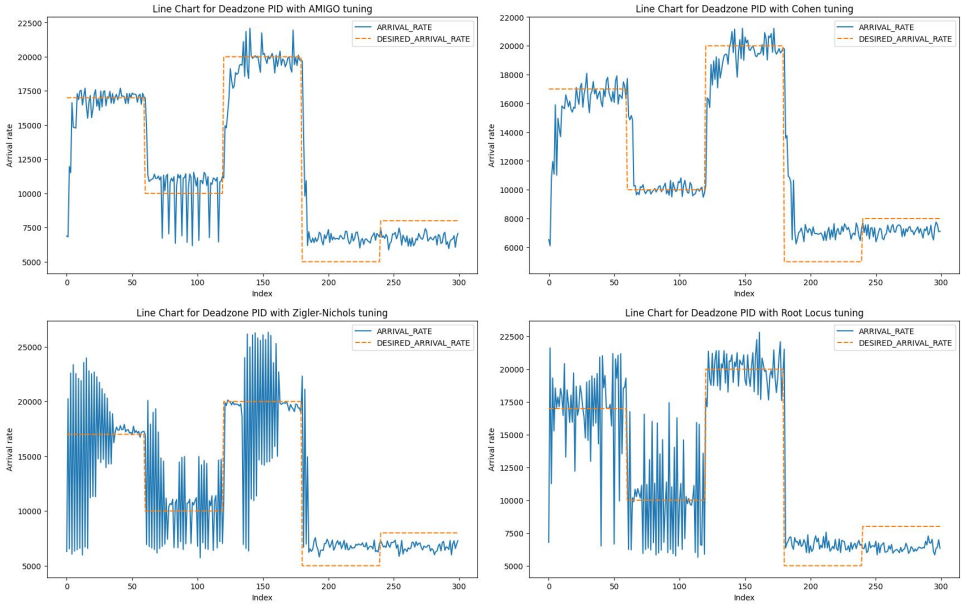


**Figura 4.9** Atuação do controlador *deadzone* PID com meta variável.

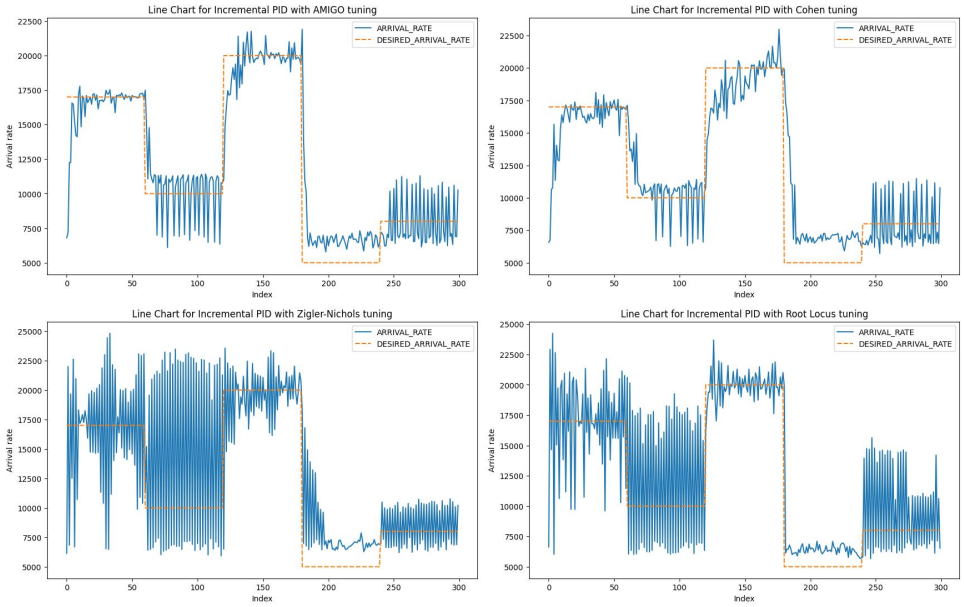


**Figura 4.10** Atuação do controlador *incremental* PID com meta variável.

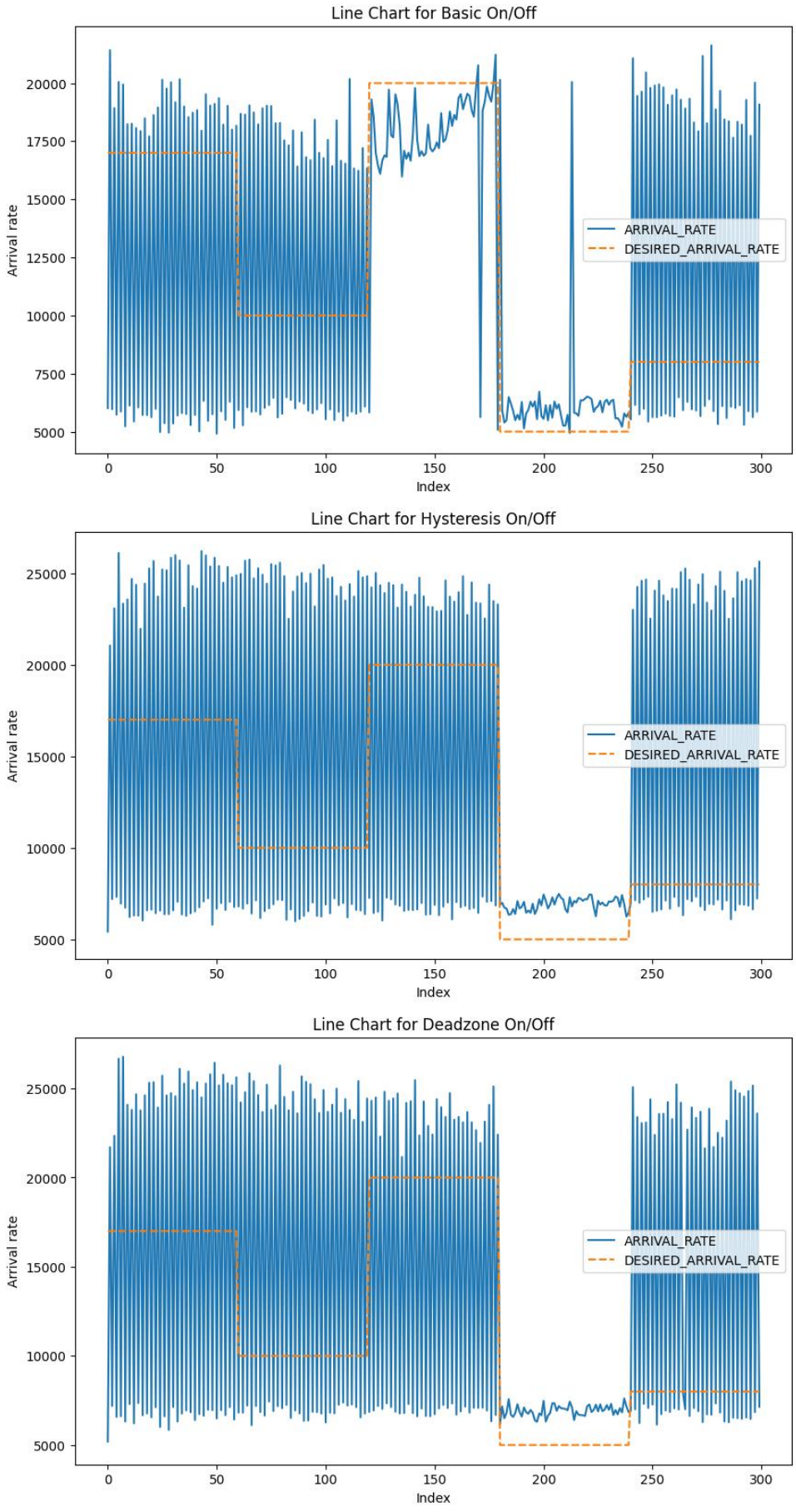


**Figura 4.11** Atuação do controlador *On/Off* básico e suas variações com meta variável.

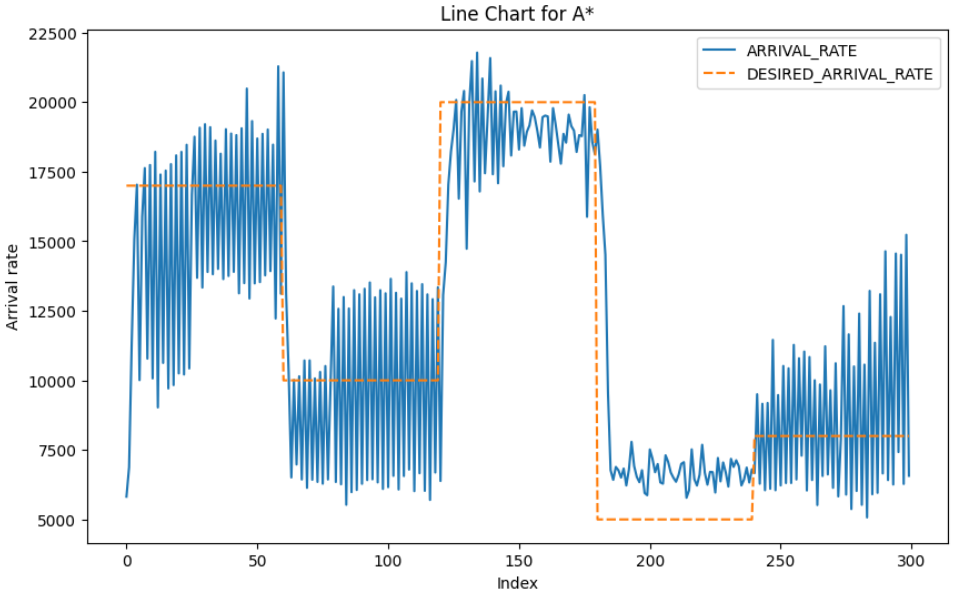


**Figura 4.12** Atuação do controlador AsTAR com meta variável.

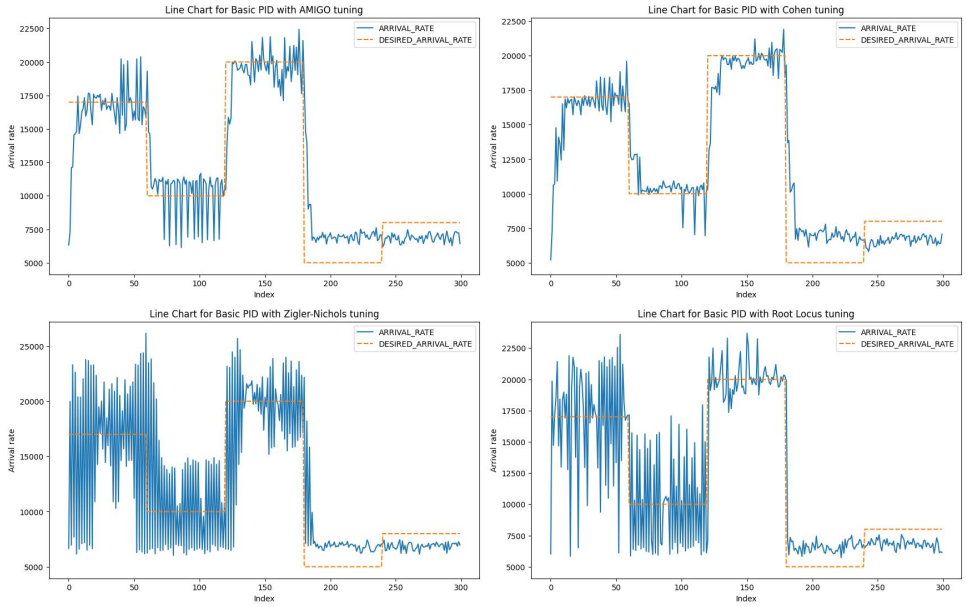


**Figura 4.13** Atuação do controlador PID básico com meta variável.

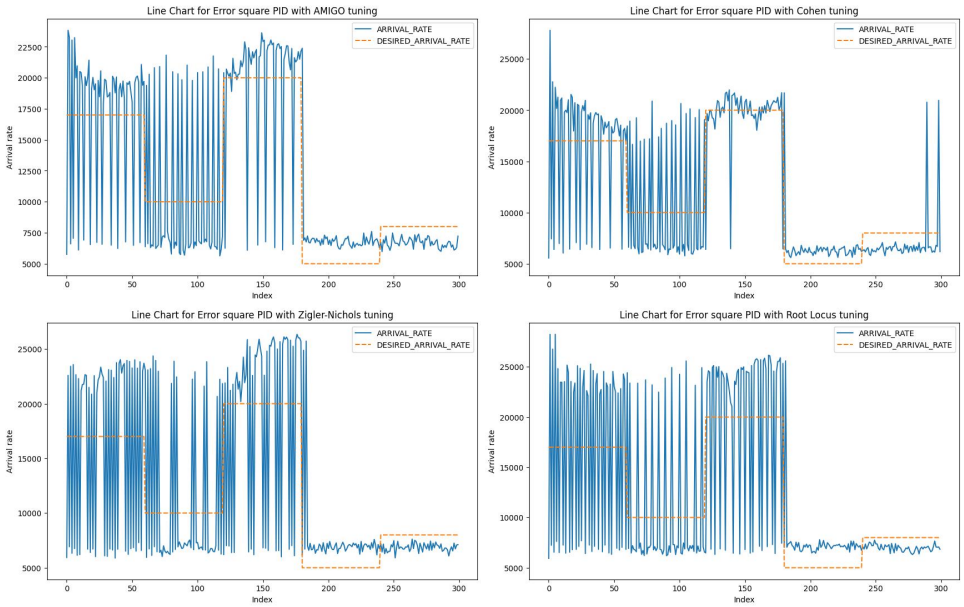


**Figura 4.14** Atuação do controlador *error square* PID com meta variável.

**Tabela 4.1** Resultados RMSE para os controladores

| Controlador | Método de Tuning | RMSE |
|---|---|---|
| HPA | - | 1767.524 |
| *Deadzone PID* | AMIGO | 2138.828 |
| *Incremental PID* | AMIGO | 2258.445 |
| *Deadzone PID* | Cohen | 2278.008 |
| *Basic PID* | Cohen | 2328.080 |
| *Basic PID* | AMIGO | 2336.601 |
| *Deadzone PID* | Root Locus | 2701.504 |
| *Incremental PID* | Cohen | 2724.580 |
| *Basic PID* | Root Locus | 3116.416 |
| AsTAR | - | 3419.310 |
| *Incremental PID* | Root Locus | 3837.503 |
| *Deadzone PID* | Ziegler-Nichols | 4084.072 |
| *Errorsquare PID* | Cohen | 4233.348 |
| *Basic PID* | Ziegler-Nichols | 4508.998 |
| *Errorsquare PID* | AMIGO | 4567.479 |
| *Incremental PID* | Ziegler-Nichols | 5035.184 |
| *Basic On/Off* | - | 6261.230 |
| *Errorsquare PID* | Root Locus | 6267.488 |
| *Errorsquare PID* | Ziegler-Nichols | 6342.929 |
| *Deadzone On/Off* | - | 9089.209 |
| *Hysteresis On/Off* | - | 9274.755 |

CAPÍTULO 5

# Conclusão e Trabalhos Futuros

Em um contexto de adaptação de sistemas distribuídos, a Teoria do Controle fornece um arcabouço conceitual valioso para abordar essa adaptação, permitindo a contínua sintonia dos sistemas com as mudanças no ambiente e nos recursos disponíveis.

Este trabalho se concentrou em resolver um problema específico relacionado a otimização do uso da variável PC de *subscribers* RabbitMQ. Essa otimização desempenha um papel crucial na prevenção de gargalos de desempenho e na garantia da disponibilidade do sistema. A abordagem e os resultados apresentados oferecem percepções valiosas para aprimorar o desempenho desses sistemas e demonstram a aplicabilidade da Teoria do Controle em um cenário cada vez mais relevante e desafiador.

Este trabalho analisou o desempenho de 9 (nove) controladores diferentes atuando junto a um *subscriber* do serviço de mensageria RabbitMQ. Os controladores foram avaliados conforme o erro quadrático médio (RMSE), que é uma medida da distância entre a saída real do sistema e a saída desejada. Além disso, para os controladores PID, foram utilizados 4 (quatro) tipos diferentes de métodos de *tuning*: *AMIGO*, *Root Locus*, Cohen e Ziegler-Nichols.

Os controladores foram avaliados usando duas abordagens: uma com *setpoint* fixo e outra com *setpoint* variável.

Na abordagem com *setpoint* fixo, destacou-se o Controlador HPA e o Controlador PID Incremental com método de *tuning* AMIGO, ambos alcançando resultados sólidos com baixos valores de RMSE. O HPA mostrou uma adaptação mais precisa às perturbações do sistema, enquanto o PID Incremental AMIGO demonstrou rápida resposta a alterações no *prefetch count*. O Controlador *On/Off* Básico, apesar de ser um modelo mais simples, também apresentou desempenho superior aos demais controladores do tipo *On/Off*.

Na abordagem com *setpoint* variável, o Controlador HPA se destacou novamente, mantendo um baixo RMSE e adequando o valor da taxa de chegada considerando as metas em mudança. O Controlador PID com *deadzone* e método AMIGO também obteve um bom desempenho. O controlador *On/Off* básico foi melhor avaliado em relação aos outros controladores do tipo *On/Off* quando a meta foi variável.

Quando se tratou dos controladores do tipo *plug and play*, o AsTAR se destacou como uma opção promissora, apresentando um desempenho comparável ao HPA em ambas as abordagens. O AsTAR mostrou-se mais eficaz do que outros controladores *plug and play* e superou o desempenho dos controladores *On/Off*.

Os métodos de *tuning* desempenharam um papel crucial no desempenho dos controladores PID. O método AMIGO se destacou como uma escolha eficaz para melhorar o desempenho dos controladores PID Incremental, Básico e com *Deadzone*. No entanto, os resultados variaram significativamente com outros métodos de *tuning*, como Ziegler-Nichols e *Root Locus*, desta-

cando a importância da seleção cuidadosa do método de *tuning* dependendo das características específicas do sistema.

Em resumo, os resultados indicam que a escolha do controlador e da técnica de *tuning* adequados pode ter um impacto significativo no desempenho do sistema de comunicação, especialmente em cenários onde as metas podem variar. O Controlador HPA e o Controlador AsTAR provaram ser opções promissoras para lidar com essas situações, com o Controlador HPA liderando em termos de desempenho geral.

Este estudo proporcionou uma visão abrangente do desempenho dos controladores avaliados, abrindo assim novas perspectivas para investigações futuras e aprimoramentos nesta área de pesquisa. Como direções para trabalhos futuros, pode ser feito a realização de experimentos com uma variedade de outros controladores, a exploração de métodos de *tuning* alternativos e a investigação mais profunda de outras variáveis do RabbitMQ que possam ser dinamicamente controladas. Estes estudos adicionais podem oferecer percepções valiosas no contexto de adaptação de *message brokers*, contribuindo assim para avanços significativos neste campo.

# Referências Bibliográficas


[1] CLOUDAMQP. How to optimize the rabbitmq prefetch count, 2023.

[2] EUGSTER, P., FELBER, P., GUERRAOUI, R., AND KERMARREC, A.-M. The many faces of publish/subscribe. *ACM Computing Surveys 35*, 2 (6 2003), 114–131.

[3] GRAY, J., AND SIEWIOREK, D. P. High-availability computer systems. *Computer 24*, 9 (September 1991), 39–48.

[4] HART, P. E., NILSSON, N. J., AND RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[5] INTERNET WORLD STATS. World internet users statistics and 2019 world population stats. Disponível em: `https://www.internetworldstats.com/stats.htm`.

[6] JANERT, P. K. *Feedback control for computer systems*. O'Reilly, Beijing ; Sebastopol, CA, 2013.

[7] JOHAN, K. *Feedback Systems: An Introduction for Scientists and Engineers*, 2nd ed. Princeton University Press, S.L., 2021.

[8] KUBERNETES. Horizontal pod autoscale. `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`.

[9] MATLAB SIMULINK. Root locus design. Disponível em: `https://www.mathworks.com/help/control/ug/root-locus-design.html`.

[10] NISE, N. S. *Control Systems Engineering*, 8th ed. John Wiley & Sons, Inc., 2019.

[11] RABBITMQ. Consumer prefetch - rabbitmq. `https://www.rabbitmq.com/consumer-prefetch.html`.

[12] RABBITMQ. Rabbitmq. `https://www.rabbitmq.com/`.

[13] ROSA, N., AND CAVALCANTI, D. Using controllers to adapt messaging systems: An initial experience. In *Anais do X Workshop de Visualização, Evolução e Manutenção de Software* (Porto Alegre, RS, Brasil, 2022), SBC, pp. 46–50.

[14] ROSA, N. S., AND CAVALCANTI, D. J. M. A control-theoretical approach to adapt message brokers. In *Advanced Information Networking and Applications* (Cham, 2023), L. Barolli, Ed., Springer International Publishing, pp. 261–273.

[15] SHEVTSOV, S., BEREKMERI, M., WEYNS, D., AND MAGGIO, M. Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering 44*, 8 (2018), 784–810.

[16] STANKOVIC, J., HE, T., ABDELZAHER, T., MARLEY, M., TAO, G., SON, S., AND LU, C. Feedback control scheduling in distributed real-time systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)* (2001), pp. 59–70.

[17] WEYNS, D. *An Introduction to Self-adaptive Systems*. John Wiley & Sons, [s.l.], 2020.

[18] YANG, F., AND [ET AL.]. Astar: Sustainable energy harvesting for the internet of things through adaptive task scheduling. *ACM Transactions on Sensor Networks 18*, 1 (February 2022), 1–34.