Universidade Federal de Pernambuco

Centro de Informática da UFPE

Graduação em Ciência da Computação

Adaptação do RabbitMQ utilizando Teoria de Controle

Trabalho de Graduação

Gabriel Teixeira Soares de Almeida

Orientador: Nelson Souto Rosa

Recife

2023

Gabriel Teixeira Soares de Almeida

Adaptação do RabbitMQ utilizando Teoria de Controle

Monografia apresentada na Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a):
Nelson Souto Rosa

Recife

2023

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

Almeida, Gabriel Teixeira Soares de.
Adaptação do RabbitMQ utilizando Teoria de Controle / Gabriel Teixeira Soares de Almeida. - Recife, 2022.
33 : il., tab.

Orientador(a): Nelson Souto Rosa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2022.
Inclui referências, apêndices, anexos.

1. RabbitMQ. 2. Teoria de Controle. 3. Adaptação. I. Rosa, Nelson Souto. (Orientação). II. Título.

000 CDD (22.ed.)

Gabriel Teixeira Soares de Almeida

Adaptação do RabbitMQ utilizando Teoria de Controle

Monografia apresentada na Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Aprovado em:

Recife, _____ de _____ de 2023

Banca Examinadora:

Nelson Souto Rosa
Centro de Informática da UFPE
(Orientador)

Carlos André Guimarães Ferraz
Centro de Informática da UFPE

Recife

2023

AGRADECIMENTOS

Alguns anos se passaram, e é tanta gente pra ser grato, não é mesmo? Eu agradeço ao apoio que minha família me deu ao longo desses anos, especialmente a Mariomar, Osvaldo e Maria Salete por terem me levantado quando achei que não tinha mais forças.

Aos meus amigos, dos quais conheci durante o curso, Adriano, Aldiberg, Amanda, Bruno, Christian, Dahise, Daniel, Kristian, Leão,Pedro, Vitor, Ze Gabriel dentre muitos outros, que ficaram até tarde nos GRADs, martelando a cabeça com projetos e estudos, que vemos hoje que compensou.

Aos amigos de colégio, Gustavo, João Vitor, Leonardo, Marinho e Nícolas, que venham mais e mais aventuras pela frente nessa grande estrada.

Ao professor Nelson Rosa, que se disponibilizou a me orientar neste trabalho. Obrigado por toda paciência em ensinar, tirar dúvidas e disponibilidade durante todo esse processo.

A professora Valéria Times, fazendo grande diferença me oferecendo a oportunidade de ter sido seu monitor por grande parte dessa jornada

A todos os professores e funcionários do Centro de Informática, que juntos fazem com que este seja um ambiente de grande ensinamento.

E por fim, agradeço a Teixeira, que tentou manter a cabeça erguida diante de suas dificuldades, e não desistiu, espero que hoje esteja colhendo frutos de seu trabalho.

RESUMO

Atualmente, o uso de serviços de mensageria de fácil uso, tal como o RabbitMQ, tem se popularizado no desenvovilmento de aplicações. Porém ele comumente é utilizado com as configurações iniciais, resultando em um ambiente genérico e mal otimizado. Estudos apontam uma das variáveis do sistema, o *Prefetch Count*, para o controle do fluxo de mensagens, porém, sem a demonstração da eficiência e comportamento do sistema ao ocorrer sua alteração. Para tal, neste trabalho será demonstrado o comportamento de diferentes controladores integrados ao sistema RabbitMQ, utilizando o *tunning* de *Root Locus*. Portanto, é possível notar a eficiência das técnicas de controle apresentadas, possibilitando também replicá-las, a fim de otimizar o ambiente configurado.

Palavras-chave: RabbitMQ, Teoria de Controle, Adaptação.

ABSTRACT

Currently, the use of user-friendly messaging services, such as RabbitMQ, has become popular in application development. However, it is commonly used with the initial settings, resulting in a generic and poorly optimized environment. Studies point to one of the system variables, the Prefetch Count, to control the message flow, however, without demonstrating the efficiency and behavior of the system when its alteration occurs. To this end, this work will demonstrate the behavior of different controllers integrated to the RabbitMQ system, using the tuning of Root Locus. Therefore, it is possible to notice the efficiency of the presented control techniques, also making it possible to replicate them, in order to optimize the configured environment. Keywords: RabbitMQ, Control Theory, Adaptation.

LISTA DE FIGURAS

Figura 2.1 Arquitetura sistema de mensageria 15
Figura 2.2 Arquitetura RabbitMQ 15
Figura 2.3 *Open-Loop* 16
Figura 2.4 *Closed-Loop* 16
Figura 2.5 Modelo de caixa-preta 18
Figura 3.1 Arquitetura do sistema 20
Figura 3.2 Data *Open-Loop* 21
Figura 4.1 Desempenho Controlador PID 27
Figura 4.2 Desempenho Controlador PI 27
Figura 4.3 Desempenho Controlador PD 28
Figura 4.4 Desempenho Controlador P 28
Figura 4.5 Desempenho Controlador PID om meta variável 29
Figura 4.6 Desempenho Controlador PI com meta variável 30

LISTA DE TABELAS

Tabela 3.1 Configuração das máquinas 21
Tabela 3.2 Parâmetros resultantes do *Root Locus* 22
Tabela 4.1 Resumo de desempenho dos controladores 29

LISTA DE CÓDIGOS

Código 3.1 Inicialização dos *producers* 22
Código 3.2 Inicialização de *consumer* 24

LISTA DE SIGLAS

AMPQ	*Advanced Message Queuing Protocol*
P	*Proportional*
PI	*Proportional Integral*
PID	*Proportional Integral Derivative*

SUMÁRIO

1 **INTRODUÇÃO** 12

1.1 Motivação 12

1.2 Problema 12

1.3 Solução Existente 12

1.4 Objetivos 13

1.5 Estrutura dos Documentos 13

2 **CONCEITOS BÁSICOS** 14

2.1 Sistemas Distribuídos 14

2.2 RabbitMQ 14

2.3 Teoria de Controle 16

2.3.1 Controladores 16

2.3.2 *Tunning* 18

3 **ARQUITETURA E IMPLEMENTAÇÃO** 20

3.1 Arquitetura 20

3.2 Implementação 22

4 **RESULTADOS** 26

4.1 Análise desempenho de meta fixa 26

4.2 Análise desempenho de meta variável 28

5 **CONCLUSÕES E TRABALHOS FUTUROS** 31

5.1 Contribuições 31

5.2 Trabalhos Futuros 32

REFERÊNCIAS BIBLIOGRÁFICAS 33

1 INTRODUÇÃO

Neste capítulo serão apresentadas as motivações para o desenvolvimento deste trabalho sobre sistemas distribuídos e seu problema associado. Este trabalho possui outros artigos relacionados e serão abordados logo a seguir. Serão apontados também os objetivos alcançados com os resultados apresentados, bem como a estrutura fundamentada do documento como um todo.

1.1 Motivação

Atualmente, a Internet pode ser considerada como um grande sistema distribuído, onde são utilizados diversos mecanismos e tecnologias, dentre estas, há o uso de serviços de mensagerias [1]. Os serviços de mensagerias foram implementados para que possam administrar aplicações com grande fluxo de utilização, como é o caso do RabbitMQ [2]. Esta ferramenta *open source* tem o intuito de gerenciar e monitorar requisições facilmente, lidando com o tráfego de mensagens de forma rápida e confiável. O RabbitMQ possui a capacidade de gerenciar múltiplas conexões, dando suporte a diferentes linguagens de programação, mas esta facilidade poderá implicar em um sistema não otimizado.

1.2 Problema

A utilização do RabbitMQ nos sistemas com sua configuração inicial pode acarretar em um ambiente não otimizado, devido aos seus parâmetros utilizados. Alguns destes parâmetros podem ser configurados no momento da implementação e, posteriormente, alterados. Adicionalmente, se bem configurados, podem evitar a sobrecarga do sistema. Além disso, o ambiente resultará num melhor desempenho, em ocasiões críticas, realizando com êxito seu dever onde há sobrecarga de requisições paralelas.

1.3 Solução Existente

O presente trabalho é uma extensão do que foi proposto em [3]. Nele serão realizados experimentos em um ambiente operacional diferente, com configurações do RabbitMQ distintas, adicionando-se uma análise variável, a fim de checar o comportamento em cenários instáveis.

1.4 Objetivos

O objetivo deste trabalho é demonstrar e comparar o desempenho de diferentes técnicas da Teoria de Controle [4] aplicados à adaptação do serviço de mensageria RabbitMQ. Para auxiliar no melhor cenário desejado, serão utilizados controladores para configurar o ambiente e induzir o serviço às metas desejadas. Serão empregados diferentes modelos de controlador, dos quais atendem situações distintas, sendo demonstrado e comparado seus desempenhos para os cenários apresentados.

1.5 Estrutura dos Documentos

Este trabalho está organizado em 5 capítulos. No Capítulo 2 serão apresentado os conceitos básicos necessários para melhor compreensão do objetivo proposto. Em seguida, no Capítulo 3 serão apresentados a arquitetura e a implementação da solulçao proposta. No Capítulo 4 serão apresentados a avaliação da solução proposta. E por fim, no Capítulo 5 são apresentadas as conclusões e possíveis trabalhos a serem realizados.

2 CONCEITOS BÁSICOS

Neste capítulo serão introduzidos os conceitos básicos para o entendimento do trabalho desenvolvido. Inicialmente serão apresentadas as características do RabbitMQ. Em seguida, são descritos noções básicas de teoria de controle.

2.1 Sistemas Distribuídos

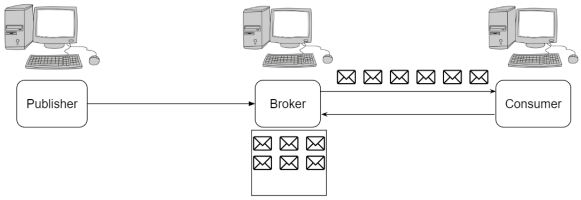
Um sistema distribuído é um conjunto de hardwares e softwares que se comunicam através da rede de computadores, havendo sua coordenação pela troca de mensagens [5]. Neste âmbito há diferentes características importantes, como a concorrência de componentes, independência de falha, dentre outros. A concorrência de componentes é a possibilidade dada ao sistema para coordenar os recursos compartilhados e tornar possível o processamento simultâneo entre seus utilizadores, podendo escalar com as configurações dos computadores conectados. O conceito da independência de falha é a possibilidade de um componente da rede ter seu sistema impedido por qualquer que seja o erro e, este, não impedirá a execução dos demais processos. Os sistemas distribuídos detém a capacidade de tornar seu ambiente transparente [6], ou seja, ocultando para os usuários que estão imersos em uma coleção de componentes, sendo esta percebida como um sistema único. Algumas implementações de transparência são:

- **Concorrência:** Permite múltiplos usuários ou aplicações acessarem os recursos compartilhados, sem a interferência entre si.
- **Acesso:** Possibilidade de recursos remotos serem acessados como o acesso de recursos locais.
- **Falha:** A capacidade do sistema em ocultar as falhas ocorridas, sejam estas do usuário ou da aplicação, permitindo aos demais concluírem suas tarefas.

2.2 RabbitMQ

RabbitMQ é uma ferramenta *Open-source* de mensageria [2], onde sua função base é facilitar a troca de mensagem entre diversas máquinas, utilizando uma estrutura de fila para coordenar a comunicação, como apresenta na Figura 2.1.

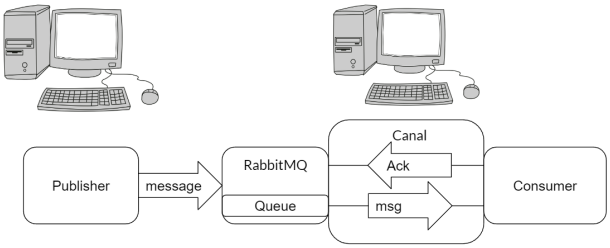
Figura 2.1: Arquitetura sistema de mensageria



Fonte: Elaborada pelo Autor.

O desempenho dos sistemas de mensageria tem se mostrado satisfatório para pequenas e grandes aplicações que utilizam múltiplas unidades de processamento de requisições. Esta popularização se deve a sua estrutura de fila, tratativa de concorrência e acesso através da comunicação por canal, apresentado na [2], pelo qual serão enviadas as mensagens e confirmações *Acknowledgements* [7]. O sistema também permite as suas conexões tomar ciência das características atuais de sua execução, como o tamanho da fila e modificações ocorridas nas variáveis do serviço através da sua comunicação por canal Figura 2.2.

Figura 2.2: Arquitetura RabbitMQ



Fonte: Elaborada pelo Autor.

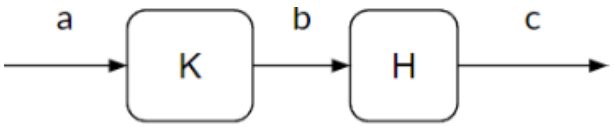
Dentre estas variáveis, vale destacar o *Prefetch Count* (PC), com capacidade de auxiliar no controle de requisições do sistema, onde em sistemas comuns tem seu valor inicialmente configurado como zero. Desta forma, indica que o sistema de mensageria irá enviar o máximo de mensagens que conseguir aos seus consumidores. Contudo, esta abordagem pode ser desnecessária no ponto de vista onde não será possível processar todas as mensagens enviadas, por conta de seu tempo de processamento tomado para cada mensagem. As mensagens não processadas resultarão em *No Acknowledgements* [7]. Os valores de PC iguais ou maiores que um resultarão na restrição de seu envio, porém, valores muito baixos de PC ocasionam a ociosidade de processamento. Visando a otimização do serviço, é recomendada a utilização de controladores e a definição de uma meta sobre

o envio de mensagens que seja condizente com a capacidade do sistema. Desta forma, será possível definir um valor de PC que o sistema consiga realizar o envio de mensagens, evitando-se com isso, tanto o uso excessivo de processamento, quanto à sua ociosidade.

2.3 Teoria de Controle

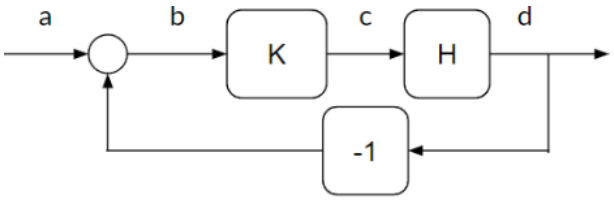
Um controlador tem como função crucial produzir sinais de entrada que servem para uma determinada unidade de processamento a partir de uma entrada base chamada de referência. Este controlador pode ser da natureza *Open-Loop*, (e.g., *feedforward*), como mostrado na Figura 2.3, repassando a referência de forma que possa ser processada pelo sistema. Adicionalmente, o controlador pode ter maior complexidade, como é o caso do *Closed-Loop* (e.g., *feedback*), visível na Figura 2.4. A saída gerada pelo *Closed-Loop* é baseada também no resultado obtido pela unidade de processamento que utilizou seu produto. A diferença entre a referência e a saída gerada pelo controlador é comumente chamada de erro.

Figura 2.3: *Open-Loop*



Fonte: Figura de [8].

Figura 2.4: *Closed-Loop*



Fonte: Figura de [8].

2.3.1 Controladores

Controladores podem ser modelados utilizando diferentes técnicas, podendo desempenhar melhor em ambientes específicos. Algumas das diferentes técnicas de controle

utilizadas é o *Proportional Integral Derivativa* (PID), *Proportional Derivative* (PD), *Proportional Integral* (PI) e o *Proportional* (P). Estas técnicas seguem a Equação 2.1 [8], onde $e(t)$ será o erro definido pelo modelo e as variáveis ki, kd e kp, são os coeficientes de aprendizado. Tais variáveis indicam, respectivamente, o quanto a integral, a derivada e o erro anterior influenciam para determinar a próxima resposta.

Controlador *Proportional*

O controlador P é o mais simples dos controladores contínuos, e de mais rápida resposta e têm como meta minimizar a flutuação na variável do processo. Porém, nem sempre consegue trazer o sistema para a meta designada, pois podem produzir desvio do ponto de ajuste, refletindo sua dificuldade em manter o sistema em um estado estável. Temos a equação do controlador P [9] ao zerar os coeficientes de integral e derivativa (e.g. ki e kd) na Equação 2.1.

Controlador *Proportional Integral*

O Controlador PI é uma combinação entre controlador P e o controlador Integral [9], do qual toma a vantagem de minimizar a flutuação de P, e a remoção de desvios que possam existir do controlador Integral. Sendo assim, o controlador PI [9] busca maior estabilidade e a capacidade de retornar a sua meta original. Podemos adquirir sua fórmula matemática ao zerar kd da Equação 2.1.

Controlador *Proportional Derivative*

Para o controlador PD [9], há novamente a junção do controlador P e o Controlador Derivativo [9], onde este antecipa as condições do processo, analisando as mudanças no coeficiente de erro. O controlador PD, desta forma, realiza o esforço em manter o sistema mais estável e resistente a mudanças. Sua fórmula é adquirida ao zerar ki da Equação 2.1.

Controlador Proporcional Integral Derivativo

O controlador PID [9] é a combinação dos três tipos de métodos de controle. É o controlador mais comumente utilizado, por extrair as vantagens anteriormente citadas. Obtemos o valor do controlador PID a partir da aplicação da Equação2.1.

$$u_{PID}(t) = u_P(t) + u_I(t) + u_D(t) = k_P e(t) + k_I \int_t^0 e(t)\,dt + k_D \frac{de(t)}{dt} \qquad (2.1)$$

2.3.2 *Tunning*

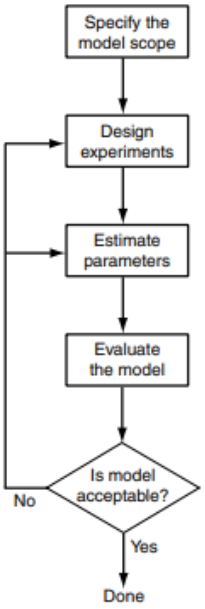
Tunning é o processo de ajustar variáveis de controle a fim de fazer com que o sistema divida seus recursos mais eficientemente em suas atividades. O processo de *tunning* possui diferentes ajustes e abordagens para os variados escopos de problemas. Devido a complexidade de se utilizar desse processo, é possível simplificar este procedimento com uma abordagem de modelo de caixa-preta [10]. Em um modelo de caixa-preta, é de maior importância a correlação do resultado gerado por sua saída e sua entrada. Desta relação é possível quantificar em uma equação linear como apresentado na Equação 2.2.

$$y(k+1) = ay(k) + bu(k) \tag{2.2}$$

Para este modelo, ocorre a dependência contínua dos passos anteriores, onde o k representa o quantitativo de passo influenciando os passos seguintes $y(k+1)$. Sendo assim, é necessário seguir os seguintes passos para melhor refinamento do modelo:

1. Projetar os experimentos;
2. Estimar os parâmetros do modelo (a e b da Equação 2.2); e
3. Avaliar o modelo.

Figura 2.5: Modelo de caixa-preta



Fonte: Figura de [10]

Caso o modelo seja aceitável, de acordo com alguns critérios apresentados em [10], procedemos com o modelo, caso contrário, criamos um novo experimento até que um novo modelo apropriado seja encontrado. Tais passos são apresentados na Figura 2.5.

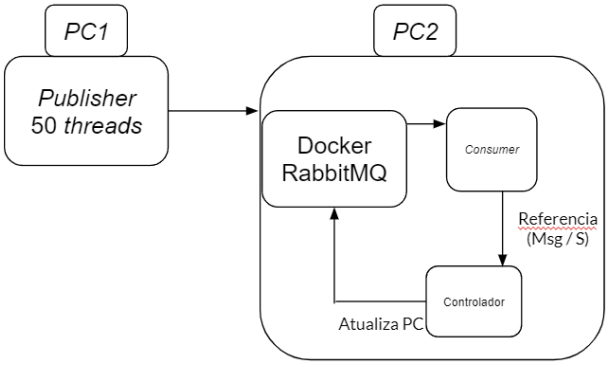
3 ARQUITETURA E IMPLEMENTAÇÃO

Neste capítulo, iremos mostrar como foi preparado o sistema onde foram executados os experimentos realizados, especificações das máquinas, trechos de código utilizados e definição de escopo de controladores para o uso do RabbitMQ.

3.1 Arquitetura

Para que seja possível realizar o *tunning* das variáveis do sistema, é necessário preparar previamente o ambiente RabbitMQ, do qual pode ser feita de diferentes formas. Visto que sua aplicação será disponibilizada na rede, somente necessitamos lidar como o *publisher* e o *consumer* irão se comunicar. Sendo assim, foi preparada a comunicação de acordo com a Figura 3.1, utilizando dois computadores que executam código em *Go* [11]. Além disso, um dos computadores servirá unicamente para o envio de mensagens e o outro hospedará tanto o serviço de mensageria, implantado no *Docker* [12], quanto o *consumer*. Mais detalhes dos computadores utilizados na configuração proposta podem ser vistos na Tabela 3.1.

Figura 3.1: Arquitetura do sistema



Fonte: Elaborada pelo autor.

A fim de que seja possível ter um melhor controle das requisições, foi incrementado a abordagem de caixa-preta. Nesta abordagem, utilizou-se o método de *tunning root locus*, necessitando como entrada o valor da média das mensagens por segundo enviadas ao *consumer*, executadas em *Open-Loop* para cada PC. Onde estas, para cada passo, foram coletadas 300 amostras em um período de 10 segundos. Todos os dados coletados

Tabela 3.1: Configuração das máquinas

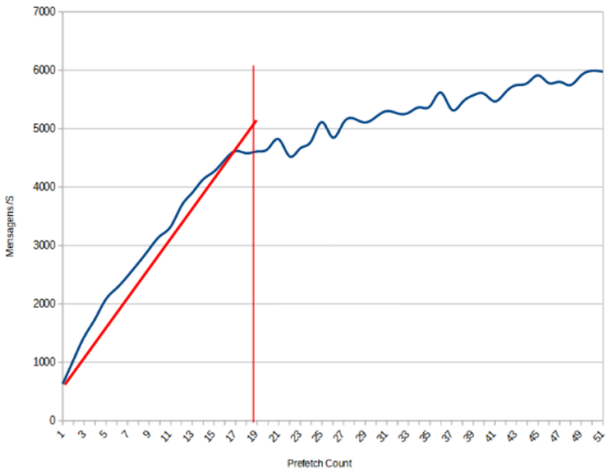
	PC 1	**PC 2**
RAM	32GB	16GB
Processador	AMD Rizen 5 3600X 4Ghz 6-Core	Intel i7-856U 1.8Ghz
Docker	Não hospeda	8 GB 4 Núcleos do hospedeiro
Sistema Operacional	Windows	Windows

Fonte: Elaborada pelo Autor.

podem ser visualizados no Google Sheets[1].

Visando maior proveito da otimização com *Root Locus*, devemos analisar a área do gráfico onde seu crescimento é linear. Observando a Figura 3.2, verificamos que a curva começa a decair em valor de PC aproximado a 19. Logo, para os experimentos, tomaremos o intervalo de crescimento estável, com valor de PC entre 1 e 19.

Figura 3.2: Data *Open-Loop*



Fonte: Elaborada pelo autor.

Após aplicado o *tunning*, é possível, enfim, adquirir os parâmetros que serão utilizados pelos controladores da Tabela 3.2. A avaliação de desempenho destes controladores foi realizada em observação das primeiras 100 amostras da execução em *Closed-Loop*, visando manter o número de mensagens consumidas em 1.400. Sendo assim, o cálculo de erro apresentado na Equação 2.1 será de acordo com a Equação 3.1, onde $r(t)$ será a meta

[1] `https://docs.google.com/spreadsheets/d/1MHzgTxKmd9En21dvZDJJTboJAysEfGzYZa9pp0DJcA/edit?usp=sharing`

desejada, e $y(t)$ será o consumo processado.

$$e(t) = r(t) - y(t) \tag{3.1}$$

Tabela 3.2: Parâmetros resultantes do *Root Locus*

	Kp	**Ki**	**Kd**
P	0.009176	0	0
PI	-0.0003	0.0025	0
PD	0.09	0	0.09
PID	0.0010903	0.0024250	0.0007687
A		**B**	
0.3130		143.0941	

Fonte: Elaborada pelo Autor.

Visando uma segunda análise mais dinâmica, foi criado outro ambiente, onde houve a análise de 300 amostras de execução em *Closed-Loop*, com comportamento de sua meta de mensagens dinâmicas, iniciando em 1.400. Após a primeira coleta, o sistema irá variar aleatóriamente entre 800 e 5.000, a fim de que seja testada a reatividade dos dois melhores controladores aplicando mudanças em cenários de instabilidade.

3.2 Implementação

A comunicação do *producer* e do *consumer* são representadas por classes distintas que estabelecem a comunicação com o RabbitMQ. O *producer* tem como função principal, demonstrado no Código 3.1, criar 50 *threads*, demonstrados entre as linhas 3 e 11. Além disso, o serviço de mensageria enviara mensagens de tamanho aleatório até 256Kb, em intervalos de 10 milissegundos, como apresentado entre as linhas 23 e 25. Todo código utilizado pode ser acessado pelo GitHub[2].

Código 3.1: Inicialização dos *producers*

```
1 func main() {
2 ...
3     numOfClients := 50
4     flag.Parse()
5       // make requests to consumer
6       for {
```

[2] https://github.com/WildCLown/RabbitMqController

```
7         for i := 0; i < numOfClients; i++ {
8           // create publisher
9           c := NewClient(*clientIdPtr, *msgSizePtr, *sampleSizePtr, *
                meanRequestTimePtr, *stdDevMeanRequestTimePtr)
10          ws.Add(1)
11          go c.RunWindows(&ws)
12        }
13        . . .
14        ws.Wait()
15        . . .
16    }
17 ...
18 }
19 ...
20 func (c Client) RunWindows(ws *sync.WaitGroup) time.Duration {
21   startTime := time.Now()
22   for {
23     corrId := shared.RandomString(32)
24     interTime := c.Mean + rand.NormFloat64()*c.StdDev
25     time.Sleep(time.Duration(interTime) * time.Millisecond)
26     err = c.Ch.Publish(
27       "",          // exchange
28       "rpc_queue", // routing key
29       false,       // mandatory
30       false,       // immediate
31       amqp.Publishing{
32         ContentType:   "text/plain",
33         CorrelationId: corrId,
34         ReplyTo:       c.Queue.Name,
35         Body:          msg,
36       })
37   }
38   return time.Now().Sub(startTime)
39 }
```

O *consumer* irá contabilizar o número de mensagens recebidas ao longo de 10 segundos. Quando passado este tempo, o *monitor* notificará o sistema, executando assim a verificação de mensagens por segundo do tempo decorrido, visto na linha 3 do Código 3.2. Quando utilizado em *Open-loop*, ao atingir o número limite de amostras, será aumentado

o valor de PC e contabilizando as mensagens do zero, como demonstra a condicional da linha 10.

Código 3.2: Inicialização de *consumer*

```
1  func (s *Server) handleRequestsZieglerNicholsTraining() {
2  ...
3      monitorInterval := time.Now().Sub(t1).Seconds()
4      s.ArrivalRate = float64(count) / monitorInterval
5      q1, err1 := s.Ch.QueueInspect("rpc_queue")
6      ( . . . )
7      if q1.Messages < int(s.ArrivalRate) {
8              time.Sleep(10 * time.Minute)
9      }
10     if countSample < samples {
11             if s.ArrivalRate != 0 {
12                         countSample++
13                 }
14           } else {
15             countSample = 0
16             if !s.IsAdaptive {
17                 s.PC = s.PC + 1
18                 err := s.Ch.Qos(
19                 s.PC, // update prefetch count
20                 0,    // prefetch size
21                         true, // default is false
22                 )
23         }
24     }
25 ...
26 }
```

Inicialmente, os *publishers* irão enviar um volume de mensagens muito maior do que se é passado pelo RabbitMQ para o *consumer*. Porém, com o aumento do PC na execução em *Open-Loop*, esta ordem se inverte. Portanto, o consumo de mensagens se torna significativamente maior do que a taxa de recepção da mensageria. Sendo assim, a fim de sempre haver mensagens suficientes para serem processadas, o programa irá parar de processar novas mensagens por 10 minutos, como apresentado na linha 8 do código 3.2. Desta forma, há a oportunidade dos *publishers* preencherem a fila de consumo, evitando com

que as amostras apresentem apenas a capacidade máxima de envio, e sim, a capacidade de consumo do sistema. Não há o interesse de aumentar o número de *threads* de envio no momento em que o consumo se torna maior, tendo em vista que o RabbitMQ e o *consumer* estão dividindo o processamento do mesmo computador. Caso aumentado, haveria maior necessidade de desempenho por parte do serviço de mensageria, o que influenciaria também na capacidade de leitura de mensagens do *consumer*.

4 RESULTADOS

Neste capítulo, será apresentado um comparativo entre os diferentes tipos de controladores anteriormente apresentados, visando uma meta fixa em sua execuçao. Para os dois controladores de melhor desempenho, será comparado seu desempenho em relação aos demais controladores, e realizando uma segunda análise, alterando as suas metas ao longo do tempo para um comportamento dinâmico.

4.1 Análise desempenho de meta fixa

Devido à oscilação das amostras, dificilmente será possível enxergar pelos gráficos de desempenho sua eficiência em alcançar a meta desejada. Para tal, foi traçada uma linha de tendência e coletada a constante desta equação, podendo assim saber, aproximadamente, a média ao longo do gráfico. Com esta constante, é possível assim comparar quão preciso o controlador estar na tentativa de ajuste do fluxo de mensagens, obtendo o valor de precisão seguindo a Equação 4.1

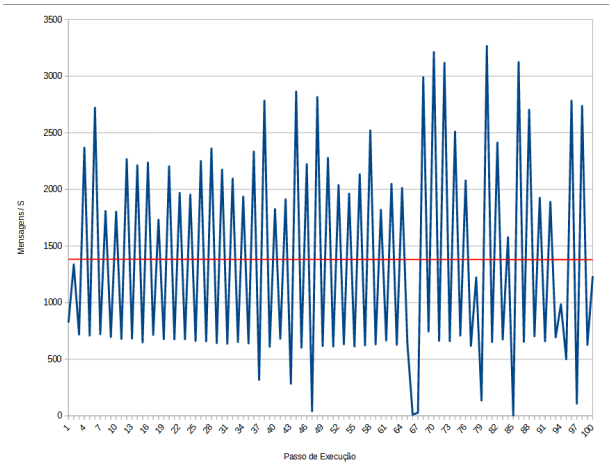
$$P = 100\frac{Min(Cl, M)}{Max(Cl, M)} \tag{4.1}$$

Nesta equação, temos Cl o valor da constante fornecida pela linha de tendência e, a constante M, como o valor da meta que buscamos alcançar. Outro parâmetro de comparação cuja importância é essencial para nossa análise é o desvio padrão. Pois este aponta quão disperso os dados se apresentam na amostra, sendo visível maiores picos, quanto maior for o seu valor.

Por apresentar maior precisão, é destacável na Figura 4.1 o controlador PID, além de possuir o segundo menor desvio padrão, sendo superado somente pelo controlador PI, visto na Figura 4.2.

Em contraste, vemos pela Figura 4.3 o desempenho do controlador PD, com o maior desvio padrão e menor precisão entre todos os controladores. Este mal desempenho apresentado se dá pela sua resistência a mudanças, somado a um sistema instável. Quando esta resistência é flexibilizada, o sistema já está tendendo a um outro cenário diferente ao do previsto. Vale ressaltar que o desempenho proporcionado pelo controlador P, observado na Figura 4.4 deixou a desejar pelo seu desvio padrão. Entretanto, por este ser o mais

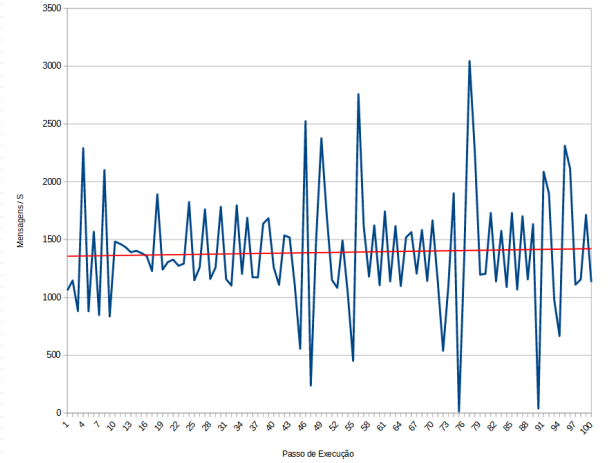
Figura 4.1: Desempenho Controlador PID



Desvio Padrão	914.57
Constante Linear	1383.60
Precisão	98.82%

Fonte: Elaborada pelo autor.

Figura 4.2: Desempenho Controlador PI



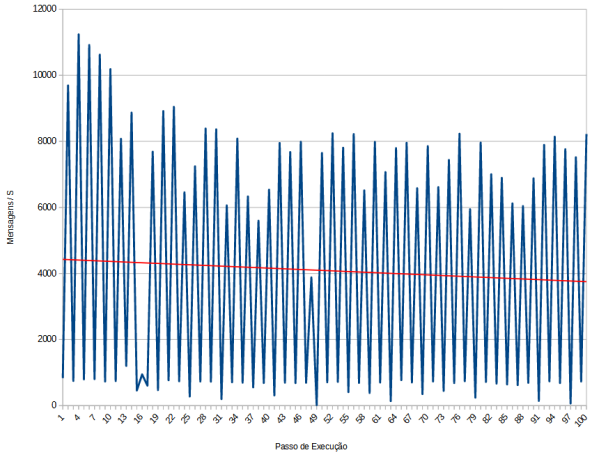
Desvio Padrão	502.64
Constante Linear	1355.49
Precisão	96.81%

Fonte: Elaborada pelo autor.

simples dos controladores, era esperado um menor desempenho em relação ao controlador PD, do qual teve precisão 47,44% menor.

Apesar de possuir maior precisão por parte do controlador PID, vale notar que a sua diferença de precisão para o controlador PI é somente 2,01% superior. Para que seja possível comparar quão variável PID está em relação a PI, observamos o coeficiente de variação. Tal coeficiente segue a Equação 4.2, onde DP é o desvio padrão, e Cl é a constante linear da linha de tendência. Ao analisar seu coeficiente de variação, PID esteve em 66,10%, este 29,02% maior comparado ao do PI. Com isto, é possível afirmar sobre o PID uma capacidade de manter sua média sobre a meta eficiente, porém, variando com

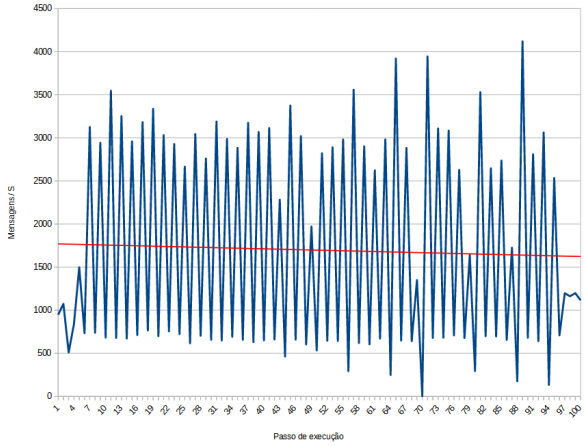
Figura 4.3: Desempenho Controlador PD



Desvio Padrão	3701.82
Constante Linear	4433.64
Precisão	31.57%

Fonte: Elaborada pelo autor.

Figura 4.4: Desempenho Controlador P



Desvio Padrão	1202.20
Constante Linear	1771.85
Precisão	79.01%

Fonte: Elaborada pelo autor.

valores elevados, apontando picos distantes em relação ao seu objetivo.

$$CV = 100\frac{DP}{Cl} \tag{4.2}$$

É possível comparar os diferentes valores de precisão e coeficiente de variação de todos os controladores sobre os valores obtidos de PI e PID, uma vez que estes possuiram os melhores resultados, sendo observado na Tabela 4.1.

4.2 Análise desempenho de meta variável

Ao depender do quão restrito se deseja o funcionamento de um sistema, será mais viável utilizar um controlador mais estável ao invés do mais preciso. Em um ambiente

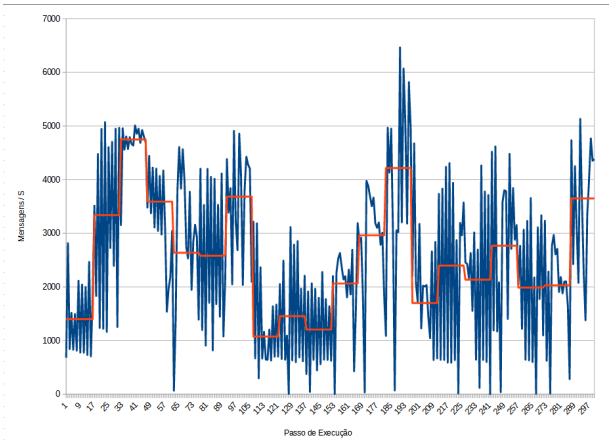
Tabela 4.1: Resumo de desempenho dos controladores

Controlador	Precisão	Coeficiente Variação (CV)	Precisão X Precisão PI	CV X CV PI	Precisão X Precisão PID	CV X CV PID
PI	96.81%	37.08%	-	-	-2.01%	+29.02%
PID	98.82%	66.10%	+2.01%	-29.02%	-	-
P	79.01%	67.84%	-17.8%	-30.76%	-19.81%	-1.74%
PD	31.57%	83.49%	-65.24%	-46.41%	-67.25%	-17.39%

corporativo, pode haver a necessidade de diminuir o poder de processamento em horários de menor troca de mensagens e aumentá-lo nos horários de pico. Visando este cenário, foi analisado o desempenho para os controladores PI e PID utilizando meta variável.

É notável pela Figura 4.5 a dificuldade do controlador em acompanhar mudanças quando ocorre uma grande troca de meta. Podemos observar tal mudança nos passos 60 e 165, onde se vê uma modificação gradativa do controlador, até que comece a ocilar de fato na linha da meta. Tal comportamente, novamente, se deve a resistência a mudanças da derivada.

Figura 4.5: Desempenho Controlador PID om meta variável

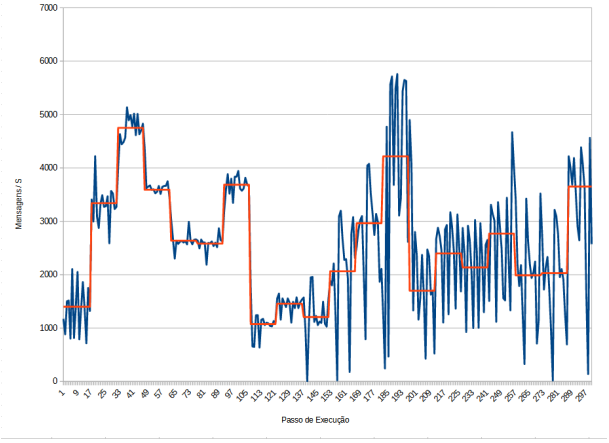


Fonte: Elaborada pelo autor.

Ao verificar o gráfico de desempenho variável de PI, observado na Figura 4.6, inicialmente é notável um bom desempenho, não ocorrendo oscilações excessivas em torno de sua meta. Porém, aproximadamente após o passo 165 de execução, o sistema demonstrou maior variação, mas esta, ainda sim, apresentou ser menor comparada ao da Figura 4.5.

A oscilação presente na Figura 4.6 pode ter sido causada por certa instabilidade da arquitetura utilizada, havendo uma concorrência de recursos de processamento pelo

Figura 4.6: Desempenho Controlador PI com meta variável



Fonte: Elaborada pelo autor.

sistema de mensageria e o consumidor. Contudo, esta instabilidade pode estar presente em todas as amostras apresentadas, mas é mais notável nas análises de meta variável.

5 CONCLUSÕES E TRABALHOS FUTUROS

Os sistemas distribuídos vêm sendo utilizados cada vez mais no mercado, além do RabbitMQ, existem diversas outras ferramentas que estão se popularizando. Tais ferramentas necessitam de otimizações específicas, sendo assim, as melhorias anteriormente apresentadas podem ou não servir em modelos semelhantes.

O uso do RabbitMQ dependerá muito do ambiente do qual está sendo configurado. Ao depender do problema abordado, seu uso pode ser exagerado para quem está desenvolvendo o ambiente do escopo do desafio. Durante este desenvolvimento, é importante ter em mente da complexidade, métricas de requisições durante o dia, picos de acesso, dentre outras informações, podendo assim, utilizar-se metas mais condizentes com o próprio sistema.

O presente trabalho replicou um dos possíveis cenários da arquitetura e configuração do RabbitMQ. Este cenário é o mais próximo de configurações de pequenas empresas, onde existe limitação do uso de servidores e, ou, computadores. Todavia, servidores empresariais possuem de poder de processamento muito mais potente se comparados com máquinas de uso doméstico. Sendo assim, a instabilidade exageradas presente nos dados resultantes se apresentariam ligeiramente menores.

Por meio dos resultados obtidos nas análises, mostrou-se que o controlador PI possui vantagens em relação aos demais controladores em ambientes simplificados, uma vez que este apresentou menor coeficiente de variação entre suas amostras. Além do mais, demonstrou efetividade em se manter na meta requisitada, tomando como parâmetro de observação a sua média ao longo da execução.

5.1 Contribuições

A contribuição efetiva deste trabalho está em decidir qual técnica de controle se deve utilizar, quando aplicado o *tunning Root Locus* e apresentar seu desempenho. Ao possuir um sistema de controle otimizado é possível tornar o sistema mais regulado sobre seu processamento e consumo. Desta forma, implica diretamente em custos gerados, sejam estes de energia, manutenções ou de melhorias, uma vez que tais métricas podem apontar necessidade de maior processamento.

5.2 Trabalhos Futuros

É possível traçar futuras análises a serem realizadas, trazendo como comparativo ao trabalho atual, aplicando o uso de diferentes formas de *tunning*, como o uso do *AMIGO* [13] ou de *Zieglen-Nichols* [14]. Uma vez que o *Root Locus* possui a necessidade de observação prévia do comportamento do sistema, ele toma grande parcela de tempo. Adicionalmente, há a necessidade de parar todas as demais requisições que são recebidas para realizar tal otimização, sendo um processo indesejado por empresas. As análises deste trabalho podem ser realizadas também com diferentes técnicas de controle mais complexas como *Fuzzy PI*, *Fuzzy PID* e *Fuzzy PD* [15].

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Brito, Thiago. *01 — Mensageria*. Disponível em: <https://medium.com/@devbrito91/mensageria-1330c6032049>.

[2] RabbitMQ. Disponível em: <https://www.rabbitmq.com/>.

[3] Rosa, N. S. and Cavalcanti. *D. J. M. Using Controllers to Adapt Messaging Systems: An Initial Experience. In Proceedings of the 10th Workshop on Software Visualization, Evolution and Maintenance*. [S.l.]: CBSOFT, 2022.

[4] Guimarães, Carlos. *Teoria de Controle e suas aplicações*. 2019.

[5] COULOURIS, G. et al. *Sistemas Distribuídos-: Conceitos e Projeto*. [S.l.]: Bookman Editora, 2013.

[6] SOARES, D. *RPC-QUIC: Middleware baseado em RPC utilizando protocolo QUIC*. [S.l.]: UFPE, 2020.

[7] KUROSE; ROSS. *Computer Networking, A Top-Down Approach*. [S.l.]: Pearson, 2000.

[8] JANERT, P. *Feedback Control for Computing Systems*. [S.l.]: O'Reilly, 2013.

[9] WOOLF, P. *Chemical Process Dynamics And Controls*. [S.l.]: LibreTexts, 2022.

[10] HELLERSTEIN, J. L. *Feedback Control of Computing Systems*. [S.l.]: John Wiley & Sons, 2004.

[11] Go Lang. *Introducing Go*. Disponível em: <https://go.dev>.

[12] Docker. *Introducing Docker*. Disponível em: <https://www.docker.com/>.

[13] Perez J. A. and Herrero P. Balaguer. *Extending the AMIGO PID tuning method to MIMO systems*. [S.l.]: Department of Systems Engineering and Design, Universitat Jaume I. Campus del Riu Sec, E-12080 Castell on de la Plana. Spain., 2012.

[14] Ellis George. *Control System Design Guide*. 2012.

[15] Kim Jong-Hwan, Park Jong-Hwan, Lee Seon-Woo and Chong Edwin K. P. *Novel Approach to Fuzzy Logic Controller Design for Systems With Deadzones*. 1992.