UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANTHONY LUAM CASSOL

ANÁLISE DE ESCALABILIDADE DO PROTOCOLO MQTT

CHAPECÓ
2022

ANTHONY LUAM CASSOL

ANÁLISE DE ESCALABILIDADE DO PROTOCOLO MQTT

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Marco Aurélio Spohn

CHAPECÓ
2022

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Cassol, Anthony Luam
Análise de Escalabilidade do Protocolo MQTT / Anthony Luam Cassol. -- 2022.
47 f.:il.

Orientador: Doutor Marco Aurélio Spohn

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal da Fronteira Sul, Curso de Bacharelado em Ciência da Computação, Chapecó,SC, 2022.

1. Internet das Coisas. 2. Escalabilidade. 3. MQTT. 4. Agrupamento de Brokers. I. Spohn, Marco Aurélio, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

ANTHONY LUAM CASSOL

ANÁLISE DE ESCALABILIDADE DO PROTOCOLO MQTT

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Marco Aurélio Spohn

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 10/8/2022.

BANCA AVALIADORA

Prof. Dr. Marco Aurélio Spohn – UFFS

Prof. Me. Adriano Sanick Padilha – UFFS

Prof. Dr. Braulio Adriano de Mello – UFFS

RESUMO

Cada vez mais objetos do dia a dia estão se conectando à internet. O crescimento da Internet das Coisas (IoT) faz com que surjam novos desafios para a computação, e com isso, a utilização de novos protocolos de comunicação se faz necessária. O MQTT é um protocolo de comunicação baseado no paradigma publicador/assinante e alguns fatores fazem com que ele seja ideal para utilização na IoT. Neste protocolo, o *broker* faz a intermediação entre publicadores e assinantes e tem um papel crucial para o funcionamento do sistema. As abordagens de escalabilidade do protocolo MQTT envolvem o agrupamento de *brokers*. O presente trabalho propõe a construção e análise de um sistema de escalabilidade do protocolo MQTT através de agrupamento de *brokers*. Para a criação do sistema foram utilizadas soluções de código aberto. Com o objetivo de extrair melhores resultados para a análise do desempenho do sistema, diferentes ambientes e cenários foram criados. No final é realizada uma discussão com base nos resultados obtidos.

Palavras-chave: Internet das Coisas. Escalabilidade. *MQTT*. Agrupamento de brokers.

ABSTRACT

More and more everyday objects are connecting to the internet. The growth of the Internet of Things (IoT) creates new challenges to the computing area, and along with that the need to use new communication protocols. MQTT is a communication protocol based on the publisher/subscriber paradigm and some factors make it ideal for use in the IoT. In this protocol, the broker acts as an intermediary between publishers and subscribers and plays a crucial role to the system's operation. Scalability approaches of the MQTT protocol involve clustering of brokers. The present work proposes the construction and analysis of a scalability system of the MQTT protocol through clustering of brokers. In order for the system to be created, open-source code solutions were used. With the intention of getting better results for the analysis of the system's performance, different environments and scenarios were created. Lastly, a discussion is made based on obtained results.

Keywords: Internet of Things. Scalability. MQTT. Clustering of brokers.

LISTA DE ILUSTRAÇÕES

Figura 1 – Processo de publicador/assinante utilizado pelo MQTT. Reproduzido de Ala et al. (1) 17
Figura 2 – Fluxo de mensagem MQTT, reproduzido de Cope (4) 18
Figura 3 – Estrutura da mensagem MQTT, reproduzido de Ala et al. (1) 19
Figura 4 – Escalabilidade horizontal MQTT, reproduzido de Detti; Funari; Blefari-Melazzi (5) 21
Figura 5 – Gráfico escalabilidade VerneMQ e eMQTT. Reproduzido de Detti; Funari; Blefari-Melazzi (5) 23
Figura 6 – Tabela escalabilidade Mosquitto. Reproduzido de Jutadhamakorn1; Pillavas; Visoottiviseth (11) 24
Figura 7 – Ambiente 1 27
Figura 8 – Ambiente 2 (sem balanceador de carga) 27
Figura 9 – Ambiente 3 (Docker) 27
Figura 10 – Resultados Cenário 1 29
Figura 11 – Resultados Cenário 4 30
Figura 12 – Resultados Cenário 5 31
Figura 13 – Resultados Cenário 10 32
Figura 14 – Resultados Cenário 14 33
Figura 15 – Resultados Cenário 15 33
Figura 16 – Resultados Cenário 2 39
Figura 17 – Resultados Cenário 3 39
Figura 18 – Resultados Cenário 6 40
Figura 19 – Resultados Cenário 7 40
Figura 20 – Resultados Cenário 8 41
Figura 21 – Resultados Cenário 9 41
Figura 22 – Resultados Cenário 11 42
Figura 23 – Resultados Cenário 12 42
Figura 24 – Resultados Cenário 13 43
Figura 25 – Retorno MQTTLoader 45
Figura 26 – Monitoramento de CPU no Glances 47

LISTA DE TABELAS

Tabela 1 – Parâmetros de configuração *MQTTLoader* 28
Tabela 2 – Cenários 28

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 16
1.1.1 Objetivos gerais 16
1.1.2 Objetivos específicos 16
2 REVISÃO BIBLIOGRÁFICA 17
2.1 MQTT 17
2.1.1 Fluxo de mensagem 18
2.1.2 Estrutura da mensagem 19
2.1.3 Escalabilidade 20
2.1.3.1 Escalabilidade vertical 20
2.1.3.2 Escalabilidade horizontal 20
2.1.4 Brokers 21
2.1.5 Balanceador de carga 21
2.1.6 Docker 22
3 TRABALHOS RELACIONADOS 23
4 METODOLOGIA 25
4.1 ENTENDENDO O PROTOCOLO MQTT 25
4.2 ESTUDO DA ESCALABILIDADE DO PROTOCOLO MQTT 25
4.3 EXPERIMENTO 25
4.3.1 Ferramentas de análise 25
4.3.2 Ambiente de testes 26
4.3.3 Cenários 27
5 ANÁLISE DOS RESULTADOS 29
5.1 AMBIENTE 1 29
5.2 AMBIENTE 2 31
5.3 AMBIENTE 3 32
6 CONCLUSÃO 35
6.1 TRABALHOS FUTUROS 35
REFERÊNCIAS 37

1 INTRODUÇÃO

Internet das Coisas (*Internet of Things*, IoT), é o termo utilizado para descrever a ideia de conectar os objetos do dia a dia (quaisquer que sejam) à internet. O termo Internet das Coisas começou a ser utilizado no início dos anos 2000, porém começou a tomar força a partir do ano de 2008, através do amadurecimento das RSSFs (Redes de Sensores Sem Fio) e do crescimento das expectativas sobre a IoT. Na atualidade o tema IoT vem sendo muito debatido tanto pela comunidade acadêmica quanto por indústrias, devido ao seu potencial de uso nas mais diversas áreas das atividades humanas. É muito comum vermos diversas aplicações inovadoras que são associadas à ideia de IoT, como *smartwatches*, casas inteligentes (*smart homes*), carros inteligentes, sensores que atuam na agroindústria e indústria, entre outros. Ao conectar objetos à internet, surge uma gama imensa de diferentes tipos de aplicações que podem desempenhar um papel importante na melhoria da qualidade de vida das pessoas. Por outro lado, emergem novos desafios e paradigmas quanto à estrutura da Rede de Computadores tradicional, visto que tais objetos possuem arquiteturas diferentes de dispositivos que geralmente se conectam à rede, como computadores e smartphones. Na maioria das vezes, os dispositivos de IoT possuem restrições energéticas, de comunicação e de processamento e, com isso, novas abordagens de padronização, requisitos e protocolos se fazem necessárias.

No que se refere a protocolos de comunicação da camada de aplicação, as aplicações da internet convencional geralmente seguem o paradigma cliente/servidor, onde cliente e servidor se comunicam através de requisição e resposta, respectivamente. O HTPP (*Hypertext Transfer Protocol*) é um protocolo bastante conhecido que utiliza o modelo cliente/servidor, porém alguns fatores fazem com que ele não seja ideal para aplicações IoT.

Em contrapartida, o MQTT (*Message Queue Telemetry Transport*) é um protocolo de comunicação que utiliza outro paradigma para realizar a comunicação entre os clientes: o publicador/assinante. Mesmo utilizando por padrão o TCP (*Transmission Control Protocol*) na camada de transporte, o MQTT mostra-se eficiente para aplicações de IoT. Nesse protocolo, os publicadores enviam mensagens a assinantes por meio de tópicos e toda comunicação é feita através de uma entidade central chamada *broker*. Por esse motivo, qualquer falha ou diminuição no desempenho do *broker* pode afetar significativamente o sistema inteiro.

Para contornar este problema, é utilizada uma abordagem de agrupamento (*clustering*) de *brokers*. Desse modo, múltiplos *brokers* distribuídos em diferentes máquinas atuam como um sistema MQTT único. Esse sistema tem como ponto de entrada um balanceador de carga, que distribui a carga de trabalho aos *brokers* de uma forma transparente. Portanto, a utilização de agrupamento aumenta a confiança e disponibilidade da aplicação, uma vez que a ocorrência de qualquer falha em um único *broker* não trará grandes problemas ao sistema. Diferentes trabalhos na literatura abordam o agrupamento de *brokers* MQTT.

Detti; Funari; Blefari-Melazzi (5) realizou um experimento utilizando um balanceador de carga com estratégia de escolha aleatória. Os resultados demonstraram um comportamento de

escalabilidade sublinear, onde o aumento do número de *brokers* não resulta numa melhora linear da performance. Como problema, os autores apontaram o roteamento interno entre *brokers*. Em Jutadhamakorn1; Pillavas; Visoottiviseth (11), é proposto um sistema de agrupamento de *brokers* MQTT de baixo custo, onde o balanceador de carga seleciona um *broker* MQTT disponível usando o ID do cliente como entrada de uma função de *hashing*. Os resultados mostram melhorias consideráveis.

Os trabalhos mencionados são realizados em cenários diferentes e bem específicos. Como mencionado, existe uma contraposição entre os seus resultados. Enquanto um aponta melhorias significativas no desempenho, o outro demonstra uma leve melhoria e apresenta problemas no roteamento interno das mensagens. Existe uma gama imensa de cenários e ambientes que podem ser explorados. Assim sendo, a verificação dos mais diversos cenários e possibilidades é extremamente importante para a sequência de estudos e decisões para a área. Com esse objetivo, este trabalho apresenta uma proposta e análise de um sistema de agrupamento de *brokers* MQTT como alternativa aos trabalhos mencionados.

1.1 OBJETIVOS

1.1.1 Objetivos gerais

Construir e analisar o desempenho de um sistema de escalabilidade do protocolo MQTT, através da utilização de agrupamento de *brokers*.

1.1.2 Objetivos específicos

- Apresentar o protocolo MQTT e suas principais características;
- Apresentar os conceitos de escalabilidade;
- Criar ambientes para realização dos testes;
- Realizar os testes em cenários distintos;
- Avaliar o desempenho do sistema.

2 REVISÃO BIBLIOGRÁFICA

Internet of Things (IoT) é o nome que se dá ao paradigma que consiste em objetos do dia a dia se conectarem à internet. Na maioria das vezes, esses objetos possuem restrições energéticas, computacionais e de comunicação. Por esse motivo, diferentes paradigmas e protocolos devem ser adotados para que a conexão desses dispositivos funcione corretamente e de uma forma adequada (17).

No que se refere a protocolos de mensagens da camada de aplicação, o protocolo padrão na internet convencional é o HTTP (Hypertext Transfer Protocol). Ele foi desenvolvido para redes de computadores do tipo PC e segue a estratégia de requisição/resposta no paradigma cliente/servidor (16). Embora o HTTP seja amplamente utilizado, ele não possui as características ideais para as aplicações IoT, dado que seus pacotes podem ocupar milhares de bytes (18) . Para contornar esses problemas, foram desenvolvidos outros protocolos de mensagens especificamente visando serem adequados a dispositivos com restrições energéticas e computacionais.

2.1 MQTT

O Message Queue Telemetry Transport (MQTT) é um protocolo de mensagens gerenciado pela Organization for the Advancement of Structured Information Standards (OASIS). Diferentemente dos protocolos de mensagens tradicionais da web que normalmente são baseados no paradigma cliente/servidor, o protocolo MQTT é baseado no paradigma publicador/assinante, que permite a transmissão de mensagens para agrupamentos específicos de clientes de forma intermitente (17).

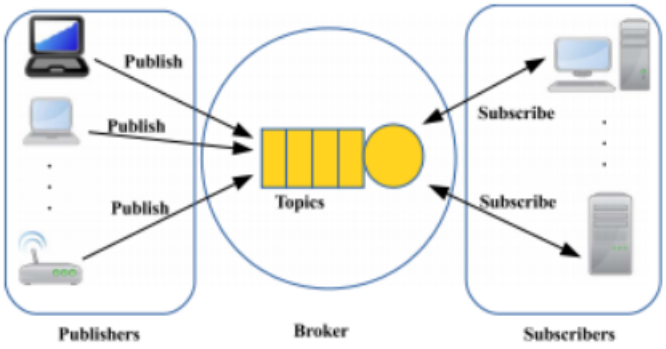

Figura 1 – Processo de publicador/assinante utilizado pelo MQTT. Reproduzido de Ala et al. (1)

A figura 1, mostra o processo de publicador/assinante que é utilizado pelo MQTT. Nesse processo existem três componentes principais: assinante, publicador e *broker*. Os clientes interessados fazem a assinatura num determinado tópico (assinantes). O cliente que provê

conteúdo envia uma informação a esse tópico (publicador). O *broker* faz a intermediação e se encarrega de entregar a informação enviada pelo publicador para os assinantes do tópico.

No MQTT, a operação de conexão usa um mecanismo de roteamento (um para um, um para muitos, muitos para muitos) (1). Além disso, os endereços dos clientes não são conhecidos e todas as mensagens são publicadas diretamente no *broker* através de tópicos, consequentemente, não existe conexão direta entre os clientes (publicadores e assinantes) e a comunicação é feita de uma forma assíncrona. Esses fatores tornam o protocolo MQTT ideal para ser utilizado em aplicações de IoT.

2.1.1 Fluxo de mensagem

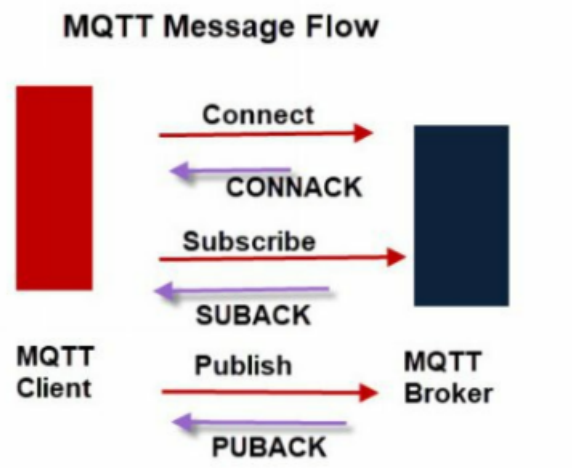

Figura 2 – Fluxo de mensagem MQTT, reproduzido de Cope (4)

O protocolo geralmente utiliza um fluxo em que cada mensagem enviada pelo cliente recebe uma confirmação do servidor, como pode-se observar na figura 2. Por exemplo, o comando de conexão (*Connect*) é confirmado pelo *broker* usando a resposta de confirmação de conexão (*CONNACK*), uma publicação (*Publish*) é reconhecida através de um reconhecimento de publicação (*PUBACK*) e uma assinatura (*Subscribe*) por um reconhecimento de assinatura (*SUBACK*). O protocolo disponibiliza 14 tipos de mensagens para uso, incluindo *Connect*, *Connack*, *Publish*, *Subscribe* e *Unsubscribe* (14).

1. *Connect*: é a primeira mensagem que deve ser enviada do cliente para o servidor. É o pedido de conexão.

2. *Connack*: é uma mensagem enviada do servidor para o cliente confirmando a conexão.

3. *Publish*: é enviada de um cliente publicador para o *broker*, e após, do *broker* para um cliente assinante.

4. *Subscribe*: é enviada do cliente para o *broker* para registrar uma assinatura num tópico de interesse. Com isso, as mensagens de *publish* enviadas para o tópico de interesse serão encaminhadas para esse cliente.

5. *Unsubscribe*: é enviada pelo cliente ao *broker* para cancelar a inscrição dos tópicos.

2.1.2 Estrutura da mensagem

O MQTT é um protocolo baseado em binário em que os elementos de controle são bytes binários e não sequências de texto como no HTTP (4). A figura 3 mostra a estrutura de uma mensagem enviada através do protocolo (pacote de controle MQTT).

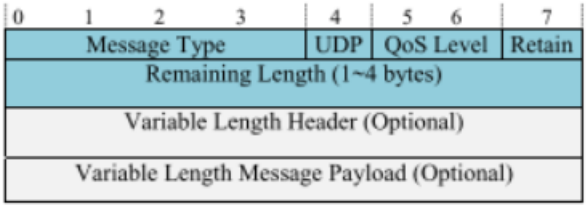

Figura 3 – Estrutura da mensagem MQTT, reproduzido de Ala et al. (1)

O formato do pacote está dividido em três partes, que por padrão, segue sempre a seguinte ordem: cabeçalho fixo, cabeçalho de comprimento variável e o conteúdo que está sendo enviado (*payload*). Cada pacote de controle MQTT deve conter um cabeçalho fixo, que possui informações relevantes da mensagem que está sendo enviada. Entre essas informações, destacam-se o tipo da mensagem (descrito na seção anterior) e os controladores DUP, *Retain* e nível de qualidade de serviço (QoS). O controlador DUP indica a duplicidade de uma mensagem do tipo *PUBLISH*. O controlador *Retain* informa ao *broker* para reter a última mensagem do tipo *PUBLISH* e enviá-la para novos assinantes como uma primeira mensagem. No que se refere a camada de transporte, o MQTT é executado por padrão em TCP/IP. De qualquer forma, podem ser utilizados outros protocolos de rede que forneçam conexões bidirecionais, ordenadas e sem perdas. Existem três níveis de qualidade de serviço para entrega de mensagens (QoS).

- QoS 0 *"At most once"*, onde as mensagens são entregues de acordo com o melhor esforço. Pode ocorrer perda de mensagem.

- QoS 1 *"At least once"*, onde há garantia de entrega. Podem ocorrer duplicações.

- QoS 2 *"Exactly once"*, onde há garantia de entrega e é entregue somente uma vez.

2.1.3 Escalabilidade

O MQTT é um protocolo destinado a aplicações de IoT, e esse fato pode fazer com que milhares/milhões de dispositivos se conectem a um mesmo *broker* simultaneamente. No MQTT, o *broker* tem o papel de receber as informações de publicadores e enviá-las para os assinantes. Dessa forma, ele participa de toda e qualquer mensagem enviada, tornando-o um ponto único de requisições e falhas. Com base nisso, algumas medidas devem ser tomadas para que o *broker* mantenha uma disponibilidade e desempenho adequado. Para isso, existem duas abordagens que geralmente são utilizadas: a escalabilidade vertical e a escalabilidade horizontal.

2.1.3.1 Escalabilidade vertical

A escalabilidade vertical consiste em adicionar mais recursos à máquina física em que o servidor MQTT está localizado. Alguns recursos como maior poder de processamento (aumento de CPUs/cores) e aumento de capacidade de armazenamento (Memória RAM) podem ser aprimorados. Com esse ganho computacional, algumas soluções têm a capacidade de rodar várias instâncias de *brokers* na mesma máquina física, através de *multi-thread*. Pelo fato do poder computacional ser algo finito e financeiramente escalável, grandes demandas de recursos fazem com que essa solução se torne cara e inviável. Outro fator desfavorável é que todo o sistema irá falhar se algum problema ocorrer com a máquina física. Por esse motivo outra solução se mostra necessária, a escalabilidade horizontal.

2.1.3.2 Escalabilidade horizontal

A escalabilidade horizontal consiste na criação de um agrupamento de *brokers* (*clustering*), ou seja, vários *brokers* em diferentes máquinas conectados através de um balanceador de carga.

Na figura 4, observa-se o funcionamento de um agrupamento de *brokers*. Os *brokers* que formam o agrupamento podem estar fisicamente em servidores separados ou então hospedados em máquinas virtuais, de preferência conectadas em uma rede de nuvem dedicada. Os clientes podem se conectar a qualquer *broker* e receber mensagens publicadas em qualquer outro *broker* do agrupamento. Geralmente, o nó do balanceador de carga é usado como um único ponto de entrada TCP do agrupamento. Para estabelecer uma sessão MQTT, um cliente (publicador ou assinante) inicialmente configura uma conexão TCP com o balanceador de carga que redireciona a conexão para um dos *brokers* internos, escolhido aleatoriamente. Basicamente, o cliente "enxerga" o agrupamento de *brokers* como se fosse único, e faz as requisições através do balanceador de carga.

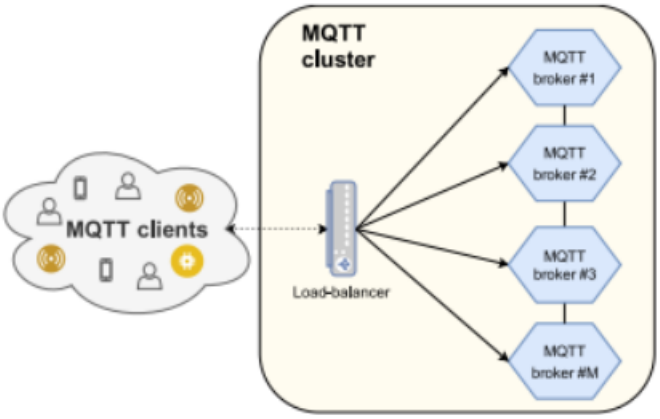

Figura 4 – Escalabilidade horizontal MQTT, reproduzido de Detti; Funari; Blefari-Melazzi (5)

2.1.4 Brokers

Existem várias implementações de *brokers* disponíveis para uso. Um *broker* MQTT de código aberto e popular é o Eclipse Mosquitto (7). No entanto, o *broker* Mosquitto faz o gerenciamento dos tópicos somente com *single-thread* e não tem suporte para agrupamento.

HiveMQ (10) é uma implementação de *broker* MQTT construída em Java que suporta *multi-threads* e agrupamento. O HiveMQ está disponível como *Community Edition* de código-fonte aberto, porém essa versão é limitada e não fornece os recursos de agrupamento. Em alternativa, existe uma versão de teste com todos os recursos disponíveis, porém, o número de conexões é limitado em no máximo 25.

O EMQX (8) é um *broker* de implementação de código aberto baseado em Open Telecom Platform (OTP), uma coleção de *middlewares* e bibliotecas escritos em Erlang. Ele tem suporte a diferentes tipos de protocolos, dentre eles o MQTT V5.0 e versões anteriores. O EMQX foi desenvolvido para acesso massivo de clientes e tem suporte para agrupamento.

2.1.5 Balanceador de carga

Para hospedar uma aplicação em mais de um servidor, a utilização de um balanceador de carga eficiente torna-se essencial. Balanceamento de carga refere-se à distribuição eficiente do tráfego de rede de entrada em um grupo de servidores. Nesse sentido, o balanceador de carga atua como um "guarda de tráfego" no *front-end* dos *brokers*, encaminhando todas solicitações dos clientes de uma forma uniforme, a fim de otimizar a utilização de recursos, maximizar o desempenho e garantir que nenhum *broker* fique sobrecarregado (12).

O HAProxy (9) é uma solução gratuita e de código-aberto, que oferece alta disponibilidade, balanceamento de carga e proxy para aplicativos baseados em TCP. É particularmente adequado para aplicações de tráfego muito alto e que distribuem solicitações em vários servidores.

2.1.6 Docker

O Docker é uma plataforma de software que permite a criação, o teste e a implantação de aplicações rapidamente. O Docker cria pacotes de software em unidades padronizadas chamadas de contêineres que têm tudo o que o software precisa para ser executado, inclusive bibliotecas, ferramentas de sistema, código e runtime (2).

3 TRABALHOS RELACIONADOS

Existem diversos trabalhos na literatura que abordam a escalabilidade de *brokers* MQTT através de agrupamento. Em Detti; Funari; Blefari-Melazzi (5) é realizado um experimento de agrupamento MQTT utilizando dois *brokers* que são desenvolvidos em Erlang e que suportam agrupamento: verneMQ e eMQTT. Para o agrupamento, foi escolhido um número máximo de 4 instâncias de *brokers*, que são conectados através Kubernetes, um sistema de código aberto que automatiza a implantação, escalonamento e gerenciamento de aplicativos em contêineres (15). Como ponto de entrada, o balanceador de carga escolhe aleatoriamente qual *broker* fará a operação. Os resultados do experimento revelam um comportamento que os autores chamam de escalabilidade sublinear, demonstrado na figura 5.

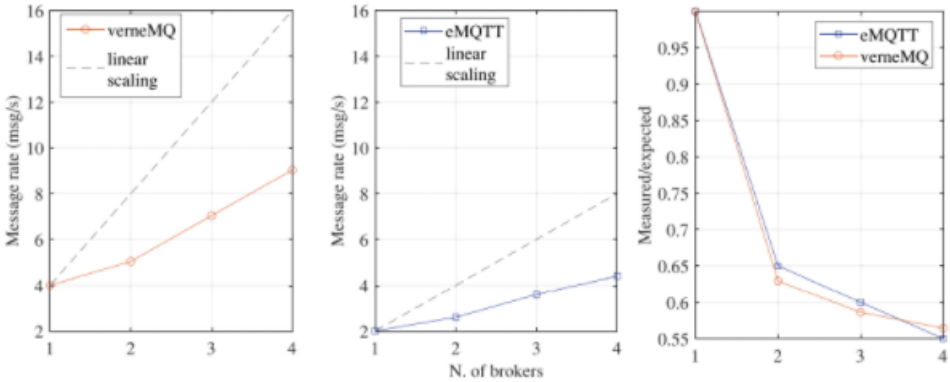

Figura 5 – Gráfico escalabilidade VerneMQ e eMQTT. Reproduzido de Detti; Funari; Blefari-Melazzi (5)

O primeiro gráfico demonstra os resultados com o *broker* verneMQ, que com 1 *broker* consegue enviar até 4000 msg/s com latência de no máximo 2 ms (milisegundos). O comportamento esperado era de que com o incremento de *brokers*, o número de mensagens enviadas fosse crescendo proporcionalmente (linearmente). Entretanto, como visto no gráfico, a partir do incremento de *brokers* o número de mensagens enviadas cresce de uma forma sublinear. O segundo gráfico mostra os resultados obtidos pelo *broker* eMQTT, que demonstra o mesmo comportamento sublinear que o anterior. Segundo os autores, esse comportamento ocorre devido ao roteamento de mensagens entre os *brokers*, chamado de tráfego de entrada. A estratégia de escolha aleatória implica que publicadores e assinantes do mesmo tópico possam estar conectados em diferentes *brokers*, e, portanto, uma troca interna de mensagens é necessária para encaminhar as publicações. Esse tráfego interno é o que dá origem ao comportamento de escalabilidade sublinear (5).

Jutadhamakorn1; Pillavas; Visoottiviseth (11) propõe um sistema de escalabilidade de baixo custo através de agrupamento de *brokers* MQTT. É utilizado o *broker* Mosquitto para o desenvolvimento da aplicação e cada instância de *broker* é executada em 4 dispositivos Raspberry Pi que são conectados através do uso de Docker Swarm, um gerenciador de contêineres que faz

a conexão entre os nodos do agrupamento. O balanceamento de carga é feito através do NGINX (12). A figura 5 mostra os resultados obtidos, que demonstram uma melhora significativa na taxa de transferência das mensagens, visto que o sistema proposto consegue enviar em média 8 vezes mais mensagens por segundo em comparação ao sistema de um único *broker*.

TABLE II. COMPARISON OF THROUGHPUT [MESSAGES/SECOND] AND ELAPSED TIME [MILLISECOND]

Time	Single Broker		Proposed System	
	Throughput	Duration	Throughput	Duration
1	1,234.32	40,508	11,944.58	4,186
2	1,464.82	34,134	11,745.36	4,257
3	1,458.19	34,289	12,065.64	4,144
4	1,657	30,175	11,704.21	4,272
5	1,325.77	37,714	12,033.69	4,155
6	1,386.5	36,062	11,990.04	4,167
7	1,671.74	29,909	11,806.38	4,235
8	1,503.31	33,260	12,248.90	4,082
9	1,315.51	38,008	11,363.64	4,400
10	1,727.83	28,938	12,970.17	3,855
Average	1,474.5	34,299.7	11,987.261	4,175.3

Figura 6 – Tabela escalabilidade Mosquitto. Reproduzido de Jutadhamakorn1; Pillavas; Visoottiviseth (11)

4 METODOLOGIA

O estudo proposto foi realizado a partir da criação de um sistema de escalabilidade do protocolo MQTT. O objetivo principal é medir o desempenho desse sistema, avaliando os resultados em diferentes cenários possíveis. Para isso, foi realizada uma abordagem quantitativa. A abordagem descreve o número de mensagens por segundo recebidas pelo sistema (vazão), a latência do recebimento das mensagens e a porcentagem de utilização de CPU. As etapas para a execução da pesquisa encontram-se descritas a seguir.

4.1 ENTENDENDO O PROTOCOLO MQTT

O entendimento do protocolo MQTT foi feito através do estudo da sua própria documentação **MQTT Version 5.0** (14), que traz por completo todas as funcionalidades disponíveis para uso. Além disso, como forma de melhorar o entendimento, foi realizada a leitura do guia *Begginers* (4). O guia traz alguns exemplos práticos de utilização do protocolo. Por fim, foram explorados alguns artigos que abordam o conteúdo.

4.2 ESTUDO DA ESCALABILIDADE DO PROTOCOLO MQTT

O estudo de escalabilidade do MQTT foi realizado por meio de leituras de publicações e artigos que têm como objetivo a explicação ou criação de sistemas de escalabilidade do protocolo.

4.3 EXPERIMENTO

4.3.1 Ferramentas de análise

Para realização dos experimentos foi utilizada uma ferramenta chamada *MQTTLoader* (3). Essa ferramenta possibilita a criação de clientes MQTT (publicadores e assinantes) e disponibiliza um arquivo de configuração que possibilita a alteração de diversos parâmetros. Destacam-se os parâmetros listados abaixo:

- Quantidade de publicadores do tópico (*num_publishers*);
- Quantidade de assinantes do tópico (*num_subscribers*);
- Número de mensagens enviadas por cada publicador (*num_messages*);
- Tamanho das mensagens publicadas (*payload*);
- Frequência em que as mensagens são publicadas (*interval*);

- Nível de QoS da mensagem publicada (*qos_publisher*);
- Nível de QoS da conexão do assinante (*qos_subscriber*).

A ferramenta permite a simulação de clientes e o envio e recebimento de mensagens. Tem como resultado algumas informações, destacando-se a vazão (msg/s) e a latência média (ms) do recebimento das mensagens.

Outra ferramenta utilizada foi a Glances (13). Glances é uma ferramenta de monitoramento multiplataforma que traz uma grande quantidade de informações. Com ela foi possível realizar a análise da carga em que o sistema está sendo submetido.

4.3.2 Ambiente de testes

Para a criação dos ambientes foram utilizadas máquinas com as mesmas configurações. Cada máquina é composta por processador Intel Core i5-3470 CPU @ 3.20Ghz x 4, 8GiB de memória RAM e 1TB de HD. Como sistema operacional foi utilizado o Ubuntu na versão 20.04.4 LTS.

A solução de *broker* escolhida foi o EMQX, pois é *open source* e tem em sua configuração padrão suporte a agrupamento de *brokers*. Para fazer o balanceamento de carga foi escolhido o HaProxy, também pelo motivo de ser uma solução *open source*. Foi configurado nas opções do balanceador o método de escolha *round-robin*, que distribui de uma maneira uniforme a carga de trabalho.

Na figura 7 observa-se o Ambiente 1, que contém três máquinas executando uma instância do *broker* EMQX em cada. Os clientes (*MQTTLoader*) e o balanceador de carga (HaProxy) são executados em uma outra máquina. As mensagens são enviadas do cliente para o balanceador, que através do método *round-robin* encaminha as mensagens para os *brokers*. De início o planejamento era ter somente o Ambiente 1. Porém, com os resultados obtidos, foram elaborados outros dois ambientes para realização dos testes.

Como pode-se observar na figura 8, no Ambiente 2 foi retirado o balanceador de carga (HaProxy). Contudo, seguem 3 máquinas executando uma instância do *broker* EMQX em cada. Nesse caso, as conexões dos clientes são feitas diretamente nos *brokers*.

O Ambiente 3 é resumido na figura 9. Este ambiente executa instâncias EMQX na mesma máquina através de Docker (6). Da mesma forma que no Ambiente 1, os clientes e o balanceador de carga (HaProxy) são executados em outra máquina.

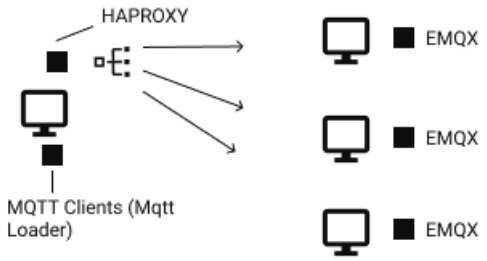

Figura 7 – Ambiente 1

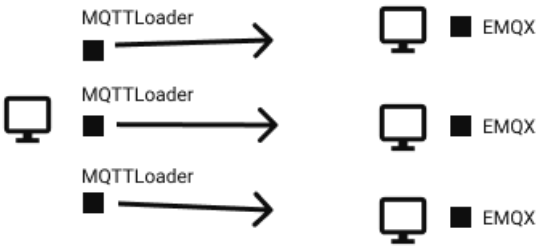

Figura 8 – Ambiente 2 (sem balanceador de carga)

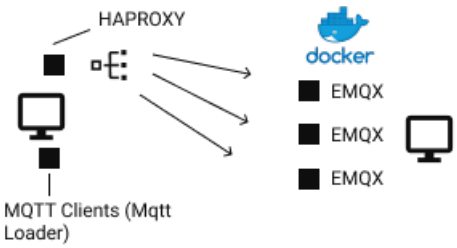

Figura 9 – Ambiente 3 (Docker)

4.3.3 Cenários

Os cenários foram elaborados com o objetivo de avaliar o desempenho do sistema em ocasiões distintas, desde estresse mínimo do sistema até o máximo. Observa-se na tabela 1 os parâmetros de configuração definidos.

Os parâmetros *num_publishers* e *num_subscribers* irão variar entre 16 e 48 e serão acrescidos de 8 em 8. Quanto maior for o número de publicadores e assinantes, maior será o número de mensagens enviadas e recebidas. Os demais parâmetros serão os mesmos em todos os testes. Com isso, todas as mensagens terão um tamanho de 10240 bytes e serão enviadas a cada 60000 μs, fazendo com que cada rodada de teste dure cerca de 60 segundos. A qualidade de serviço (QoS) será sempre 1 (*"At least once"*).

A partir dos três ambientes criados e todas as variações nas configurações dos clientes, a tabela 2 mostra os cenários possíveis para realização dos testes. Nota-se que o incremento do

Parâmetro	Valor
num_publishers	16 até 48
num_subscribers	16 até 48
num_messages	1000
payload (bytes)	10240
interval (μs)	60000
qos_publisher	1
qos_subscriber	1

Tabela 1 – Parâmetros de configuração *MQTTLoader*

número de *brokers* não foi considerado como alteração de cenário.

Cenário	Ambiente	N° Publicadores	N° Assinantes	Mensagens Enviadas	Brokers
1	Ambiente 1	16	16	16.000	1 até 3
2	Ambiente 1	24	24	24.000	1 até 3
3	Ambiente 1	32	32	32.000	1 até 3
4	Ambiente 1	40	40	40.000	1 até 3
5	Ambiente 1	48	48	48.000	1 até 3
6	Ambiente 2	16	16	16.000	1 até 3
7	Ambiente 2	24	24	24.000	1 até 3
8	Ambiente 2	32	32	32.000	1 até 3
9	Ambiente 2	40	40	40.000	1 até 3
10	Ambiente 2	48	48	48.000	1 até 3
11	Ambiente 3	16	16	16.000	1 até 3
12	Ambiente 3	24	24	24.000	1 até 3
13	Ambiente 3	32	32	32.000	1 até 3
14	Ambiente 3	40	40	40.000	1 até 3
15	Ambiente 3	48	48	48.000	1 até 3

Tabela 2 – Cenários

5 ANÁLISE DOS RESULTADOS

As seções a seguir estarão divididas entre os ambientes que foram criados para os testes. Para a discussão das mesmas foram trazidos somente os gráficos de resultados de alguns cenários. Os demais gráficos se encontram na seção de apêndices.

5.1 AMBIENTE 1

Como demonstrado na figura 10 (Cenário 1), o incremento do número de *brokers* não tem impacto nenhum sobre o sistema quando há pouca demanda. Neste caso, todas as mensagens enviadas foram recebidas sem um grande esforço da máquina. A vazão do recebimento das mensagens é a mesma, indiferente da quantidade de *brokers*.

A figura 11 mostra os resultados do Cenário 4, onde o sistema é exposto a uma carga maior de mensagens. Como pode-se observar no gráfico de utilização de CPU, a máquina já estava próxima de seu limite máximo (gargalo) quando utilizado somente um *broker*. O gráfico de latência demonstra que as mensagens estavam levando em média aproximadamente 2 segundos para serem recebidas. Ao contrário do esperado, o incremento do número de *brokers* não resulta em maior vazão do recebimento de mensagens. Na verdade, o acréscimo do segundo *broker* causa uma leve redução na vazão. Em todos os casos (entre 1 e 3 *brokers*) não houve o recebimento total das mensagens ao fim dos testes.

No Cenário 5 (figura 12) foi onde ocorreu a maior demanda de mensagens. Nesta situação, chega-se ao limite da máquina quando utilizado somente um *broker*. O gargalo faz

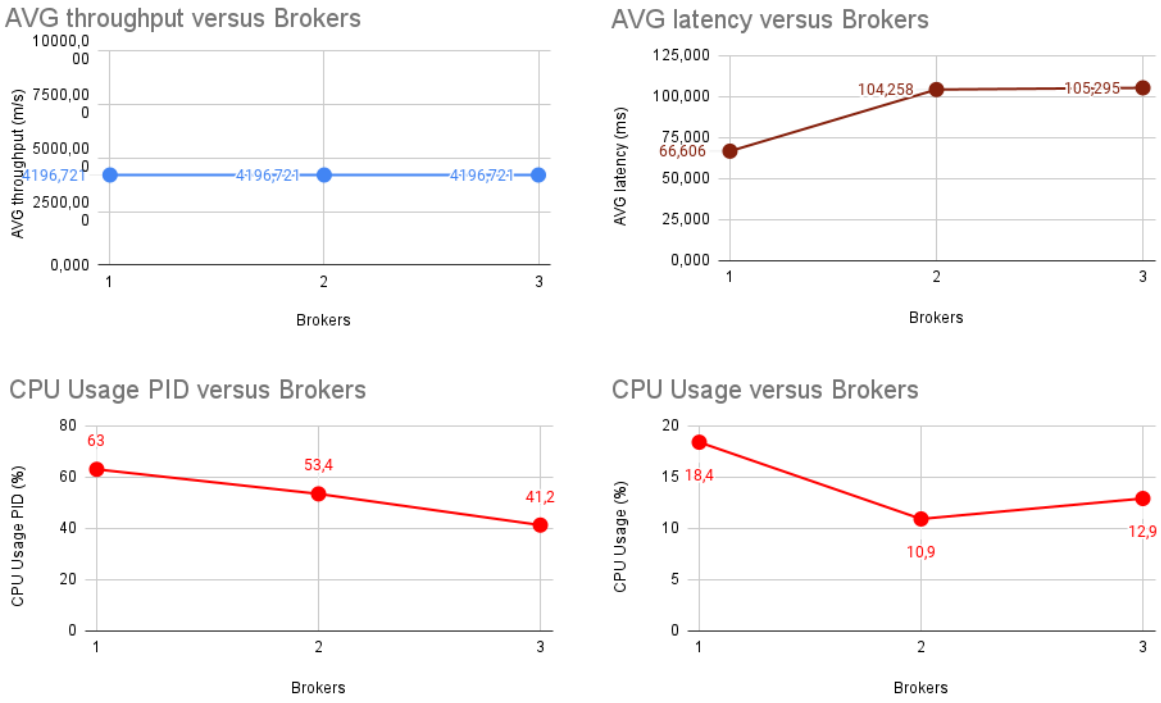

Figura 10 – Resultados Cenário 1

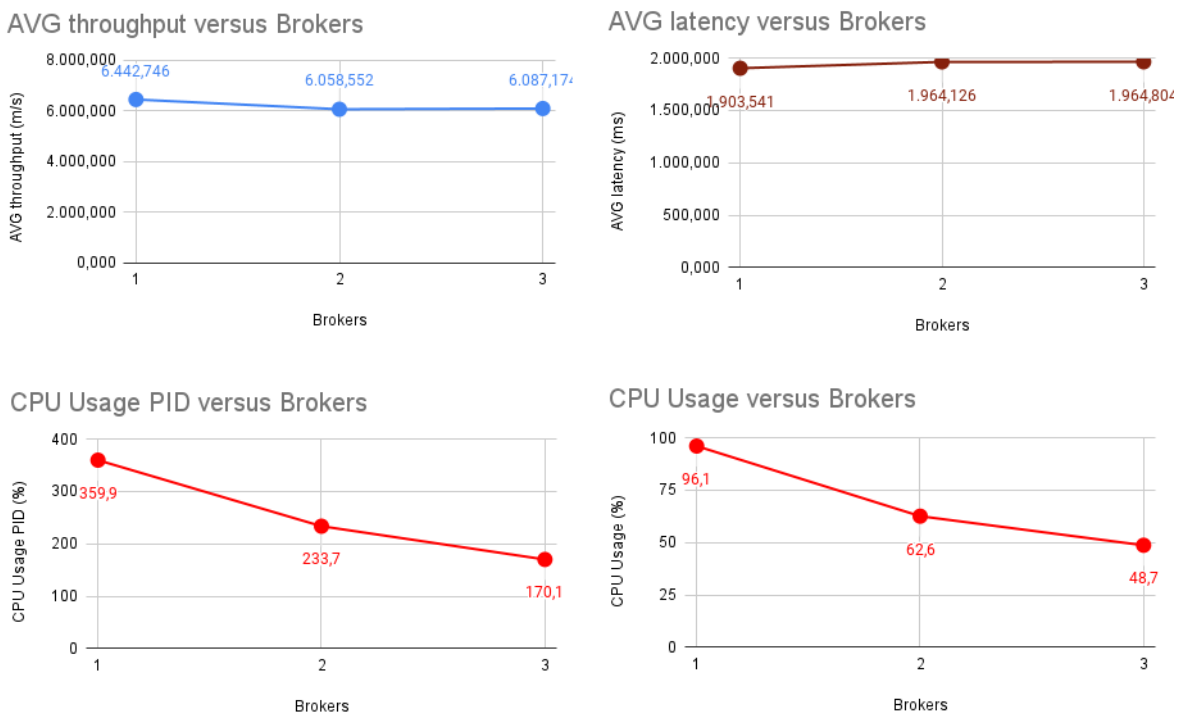

Figura 11 – Resultados Cenário 4

com que o teste finalize com antecedência e ocorra uma diminuição significativa na vazão do recebimento de mensagens. De outra forma, o acréscimo de *brokers* faz com que a carga de trabalho seja dividida e o sistema funcione normalmente. No gráfico de utilização de CPU observam-se as reduções quando mais *brokers* são adicionados ao sistema.

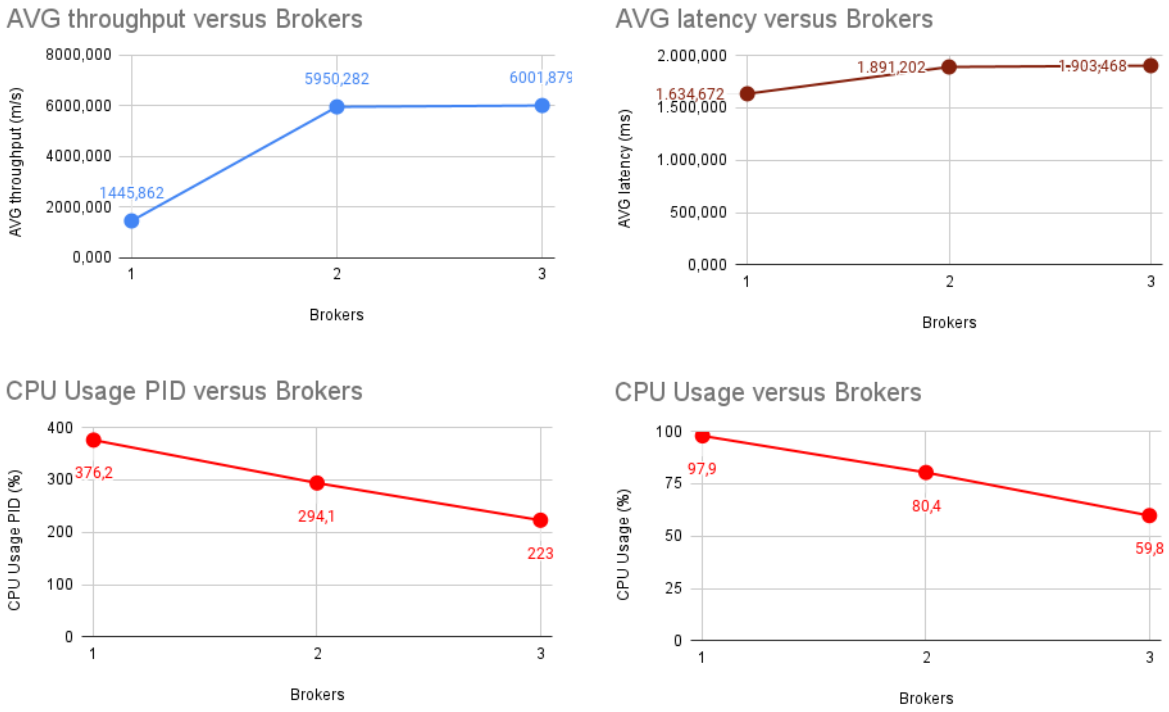

Figura 12 – Resultados Cenário 5

5.2 AMBIENTE 2

Os resultados obtidos sem o balanceador de carga (com conexão direta aos *brokers*) se mostraram semelhantes aos resultados do Ambiente 1. Da mesma forma, o incremento do número de *brokers* não resulta numa maior vazão do recebimento das mensagens. Também ocorre uma pequena redução na vazão quando adicionado o segundo *broker*. No Cenário 10 (figura 13), teste com maior demanda do Ambiente 2, observa-se que também ocorre o gargalo quando utilizado somente um *broker*. Com isso, há uma redução significativa na vazão. Como no Ambiente 1, nota-se uma estabilização quando a carga é dividida entre mais *brokers*.

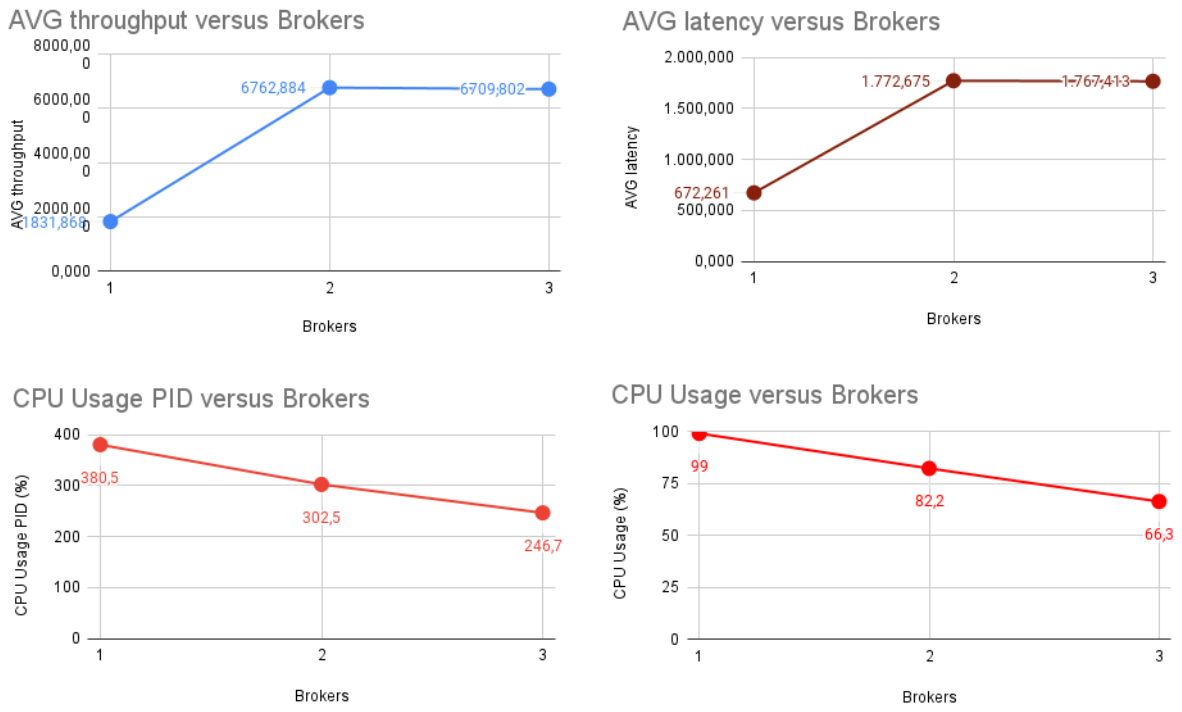

Figura 13 – Resultados Cenário 10

5.3 AMBIENTE 3

Os resultados obtidos no Ambiente 3 foram similares aos anteriores. Observa-se que quando o sistema é exposto a pouca e média demanda de mensagens, a adição de *brokers* também não resulta no aumento da vazão do recebimento das mensagens. A figura 14 mostra que o limite do sistema é atingido antecipadamente, no penúltimo teste do ambiente. No último teste os gargalos ocorrem independente da quantidade de *brokers*. Os gargalos fazem com que ocorra uma grande redução na vazão. A adição de *brokers* faz com que essa redução não seja tão grande, como é mostrado na figura 15.

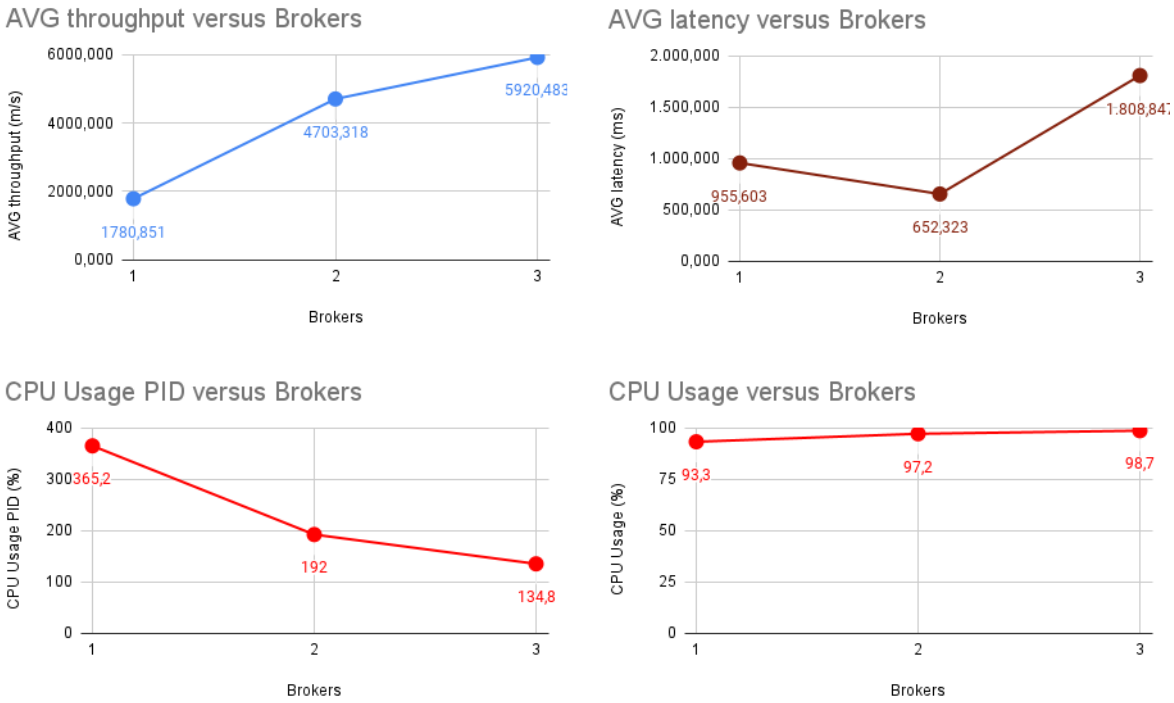

Figura 14 – Resultados Cenário 14

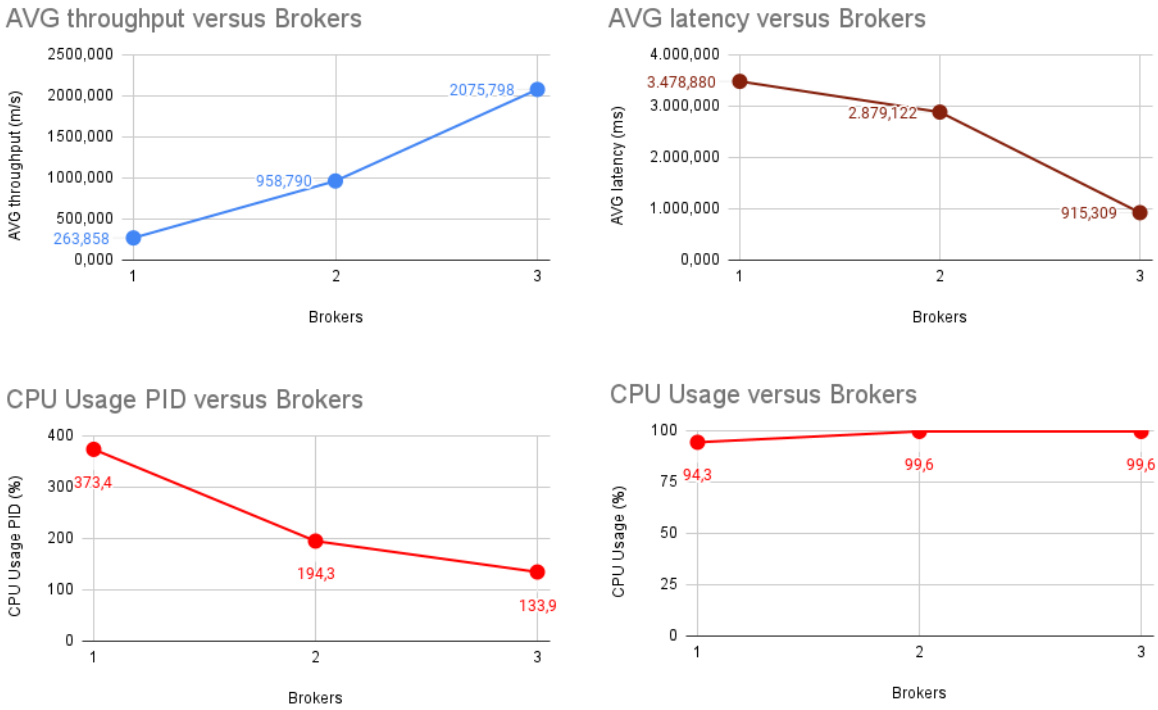

Figura 15 – Resultados Cenário 15

6 CONCLUSÃO

O presente trabalho possibilitou avaliar de forma parcial as vantagens e desvantagens de se utilizar a escalabilidade no protocolo MQTT. Nos cenários específicos em que os testes foram realizados, os resultados mostram que a escolha de se utilizar ou não o agrupamento de *brokers* depende diretamente da demanda em que o sistema estará exposto. Só foi possível notar uma melhora na taxa de transferência do recebimento de mensagens em casos de saturação do sistema. Portanto, a utilização do agrupamento de *brokers* só se faz necessária quando há uma demanda muito grande de mensagens, resultando em maior disponibilidade e confiabilidade do sistema.

Em todas as variações de ambientes e cenários os resultados se mostraram sólidos. As ferramentas e soluções utilizadas mostraram-se ser efetivas. Por fim, todos os objetivos propostos foram alcançados.

6.1 TRABALHOS FUTUROS

A ferramenta MQTTLoader (3) possibilita a configuração de diversos parâmetros. Neste trabalho foram criados cenários bem específicos e alguns desses parâmetros não foram tão explorados. Pode-se fazer alterações no número de mensagens enviadas por cada publicador, tamanho da mensagem, intervalo de publicação e qualidade de serviço (QoS).

Outro fator que também pode ser alterado é a utilização de outro método de escolha feita pelo balanceador de carga. O HAProxy disponibiliza por padrão métodos diferentes do *roundrobin* para realizar o balanceamento, como o *leastconn* e *static-rr*. Também pode-se escolher outra solução de balanceador de carga.

REFERÊNCIAS

1 ALA, Al-Fuqaha et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. **IEEE COMMUNICATION SURVEYS TUTORIALS**, v. 9, n. 4, p. 2347–2376, set. 2015.

2 AWS Docker. [S.l.: s.n.]. Disponível em: <https://aws.amazon.com/pt/docker/>. Acesso em: 30 mai. 2022.

3 BANNO. **MQTTLoader**. [S.l.: s.n.]. Disponível em: <https://github.com/dist-sys/mqttloader>. Acesso em: 22 mar. 2022.

4 COPE, Stephen. **MQTT For Complete Begginers**. [S.l.: s.n.], 2020.

5 DETTI, Andrea; FUNARI, Ludovico; BLEFARI-MELAZZI, Nicola. Sub-Linear Scalability of MQTT Clusters in Topic-Based Publish-Subscribe Applications. **IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT**, v. 17, n. 3, p. 1954–1968, set. 2020.

6 DOCKER. [S.l.: s.n.]. Disponível em: <https://www.docker.com/>. Acesso em: 28 mai. 2022.

7 ECLIPSE. **Eclipse Mosquitto, An open source MQTT broker**. [S.l.: s.n.]. Disponível em: <https://mosquitto.org/>. Acesso em: 17 jul. 2021.

8 EMQ TECHNOLOGIES CO. **An Open-Source, Cloud-Native, Distributed MQTT Broker for IoT**. [S.l.: s.n.]. Disponível em: <https://www.emqx.io/>. Acesso em: 17 jul. 2021.

9 HAPROXY. **The Reliable, High Performance TCP/HTTP Load Balancer**. [S.l.: s.n.]. Disponível em: <https://www.haproxy.org/>. Acesso em: 17 jul. 2021.

10 HIVEMQ. **Reliable Data Movement for Connected Devices**. [S.l.: s.n.]. Disponível em: <https://www.hivemq.com/>. Acesso em: 17 jul. 2021.

11 JUTADHAMAKORN1, Pongnapat; PILLAVAS, Tinnapat; VISOOTTIVISETH, Vasaka. A Scalable and Low-Cost MQTT Broker Clustering System. In.

12 NGINX Instance Manager. [S.l.: s.n.]. Disponível em: <https://www.nginx.com/>. Acesso em: 16 out. 2021.

13 NICOLARGO. **Glances**. [S.l.: s.n.]. Disponível em: <https://nicolargo.github.io/glances/>. Acesso em: 28 mai. 2022.

14 OASIS. **MQTT Version 5.0**. [S.l.: s.n.]. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>. Acesso em: 17 jul. 2021.

15 PRODUCTION-GRADE Container Orchestration. [S.l.: s.n.]. Disponível em: <https://kubernetes.io/>. Acesso em: 24 ago. 2021.

16 SANTOS, Bruno et al. Internet das Coisas: da Teoria à Prática. In: p. 20–22.

17 SILVIO, Quincozes; EMILIO, Tubino; KAZIENKO, Juliano. MQTT Protocol: Fundamentals, Tools and Future Directions. **IEEE LATIN AMERICA TRANSACTIONS**, v. 9, n. 17, p. 1439–1448, set. 2019.

18 UFRJ. **Constrained Application Protocol (CoAP)**. [S.l.: s.n.]. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/coap/>. Acesso em: 16 jul. 2021.

APÊNDICE A - RESULTADOS DOS CENÁRIOS

As figuras abaixo mostram os gráficos dos resultados de cenários que não foram abordados no texto.

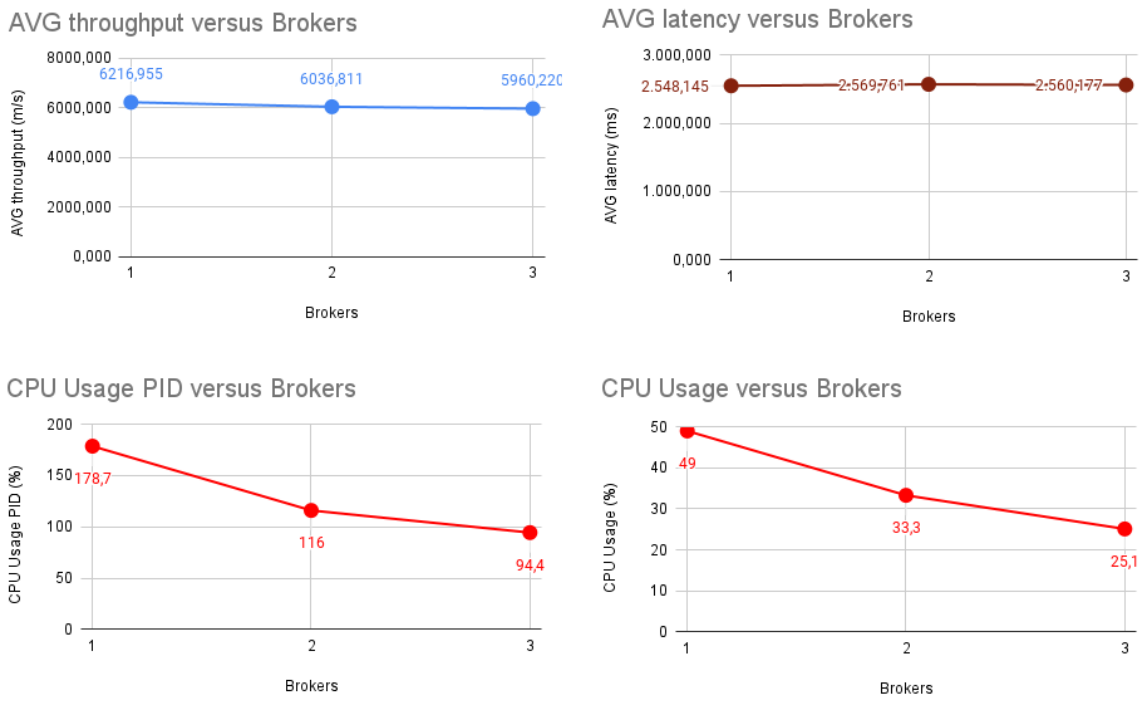

Figura 16 – Resultados Cenário 2

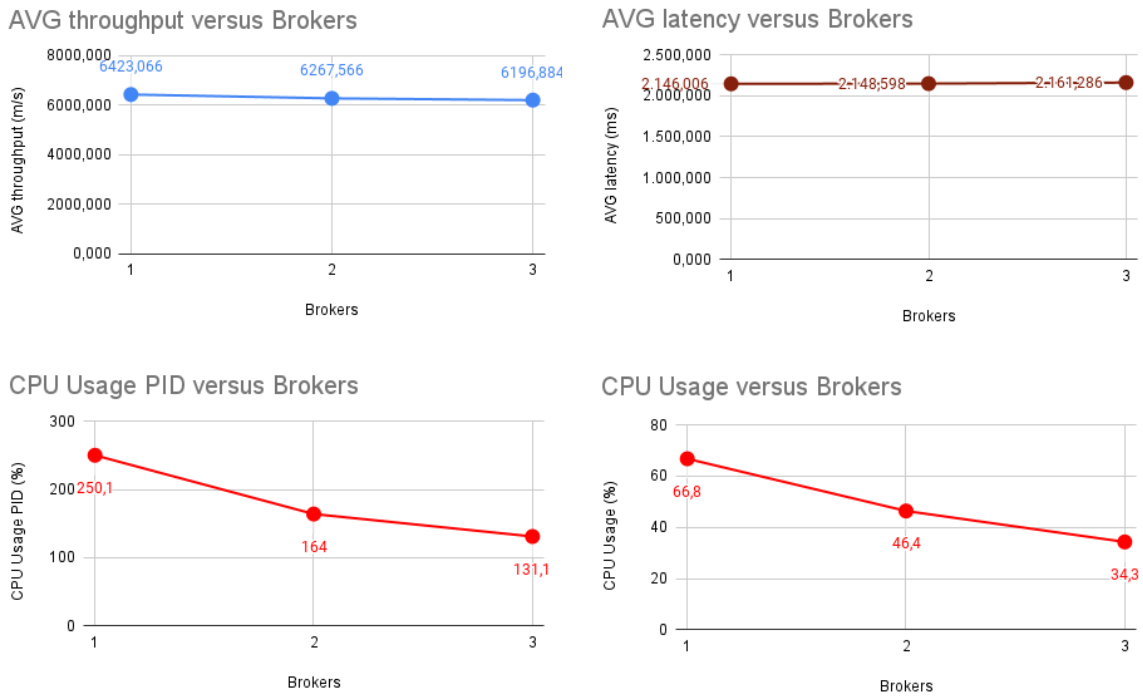

Figura 17 – Resultados Cenário 3

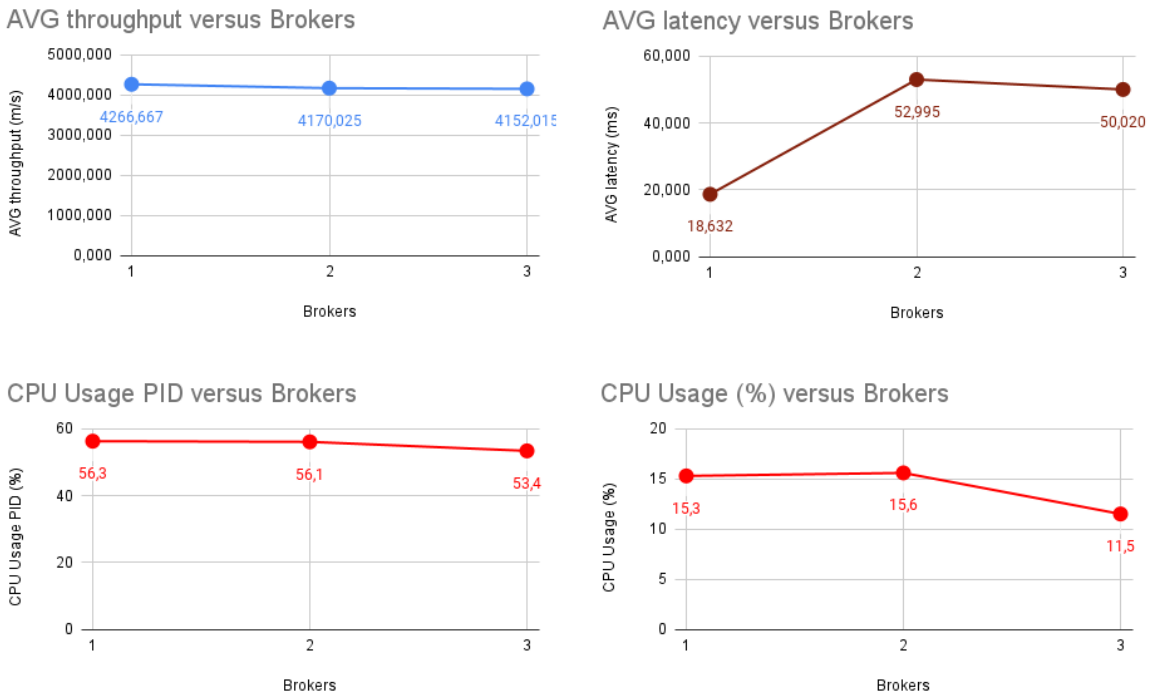

Figura 18 – Resultados Cenário 6

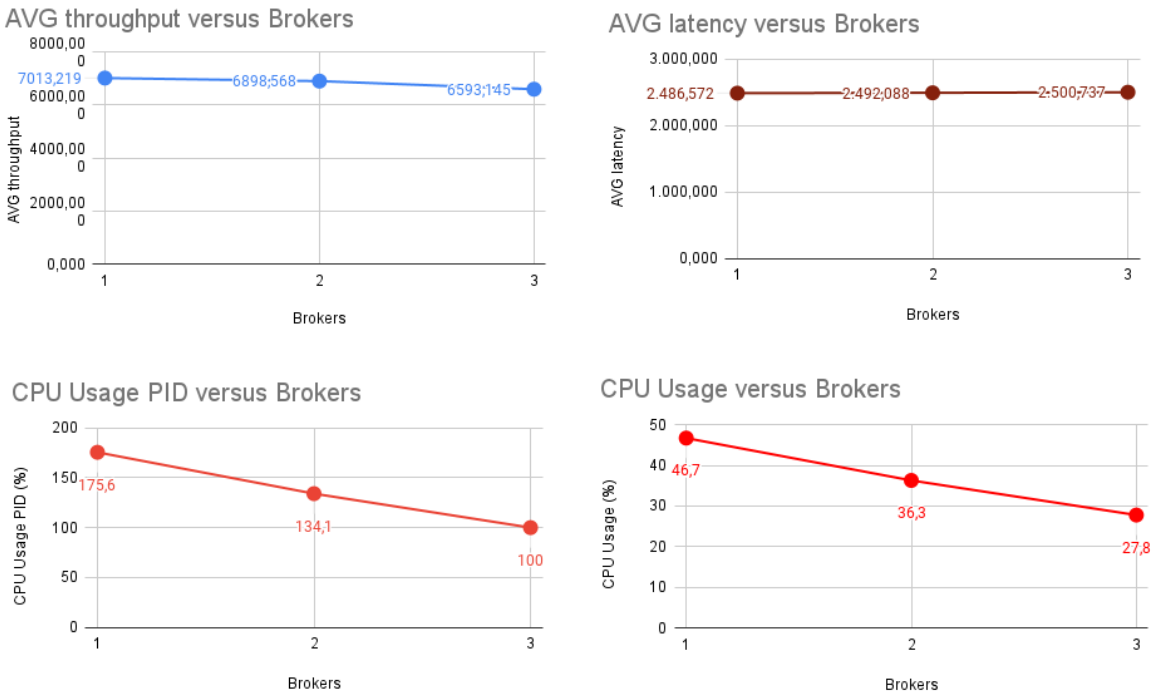

Figura 19 – Resultados Cenário 7

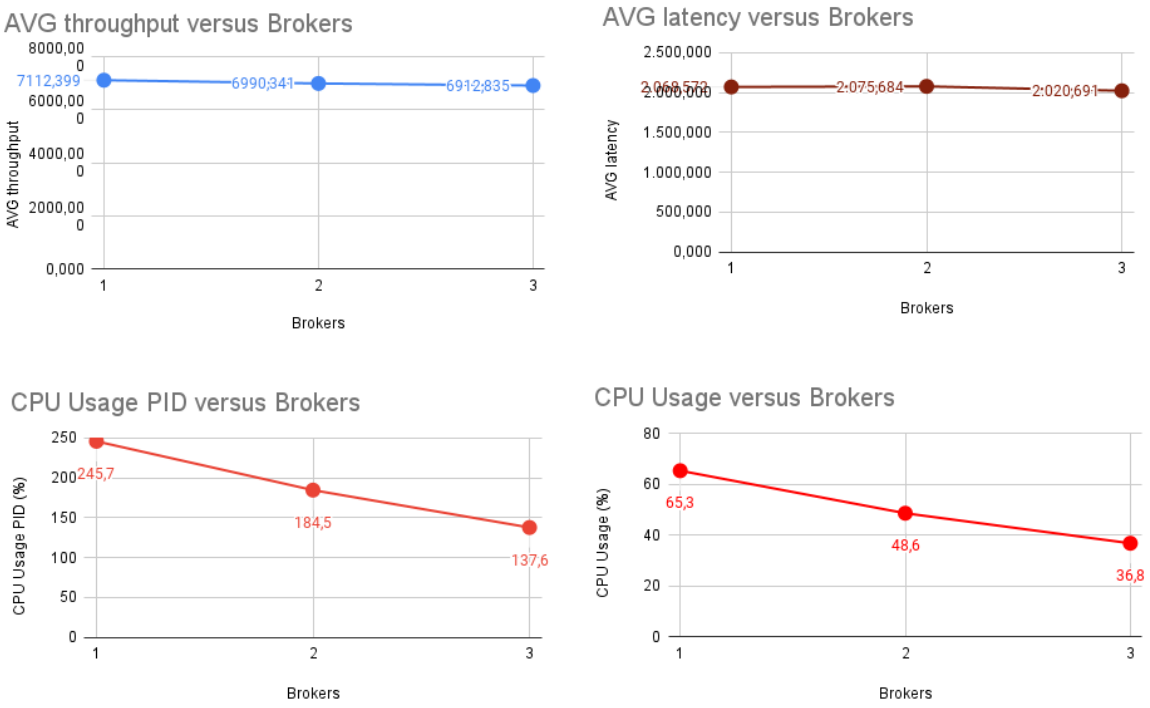

Figura 20 – Resultados Cenário 8

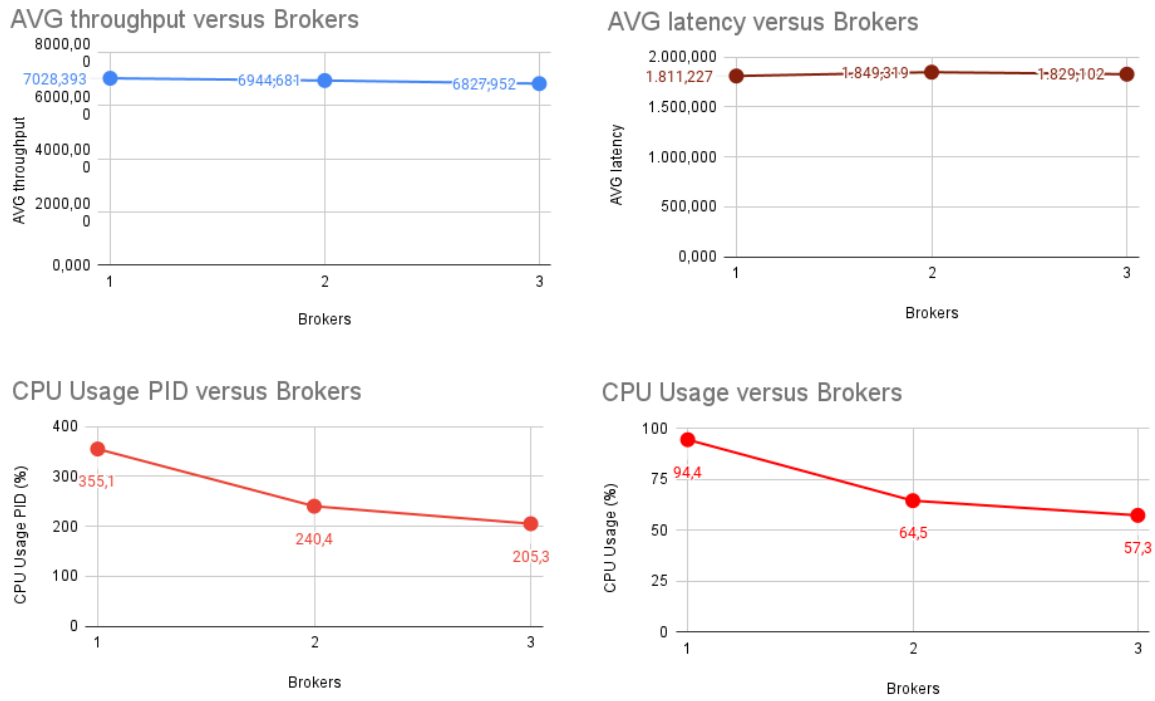

Figura 21 – Resultados Cenário 9

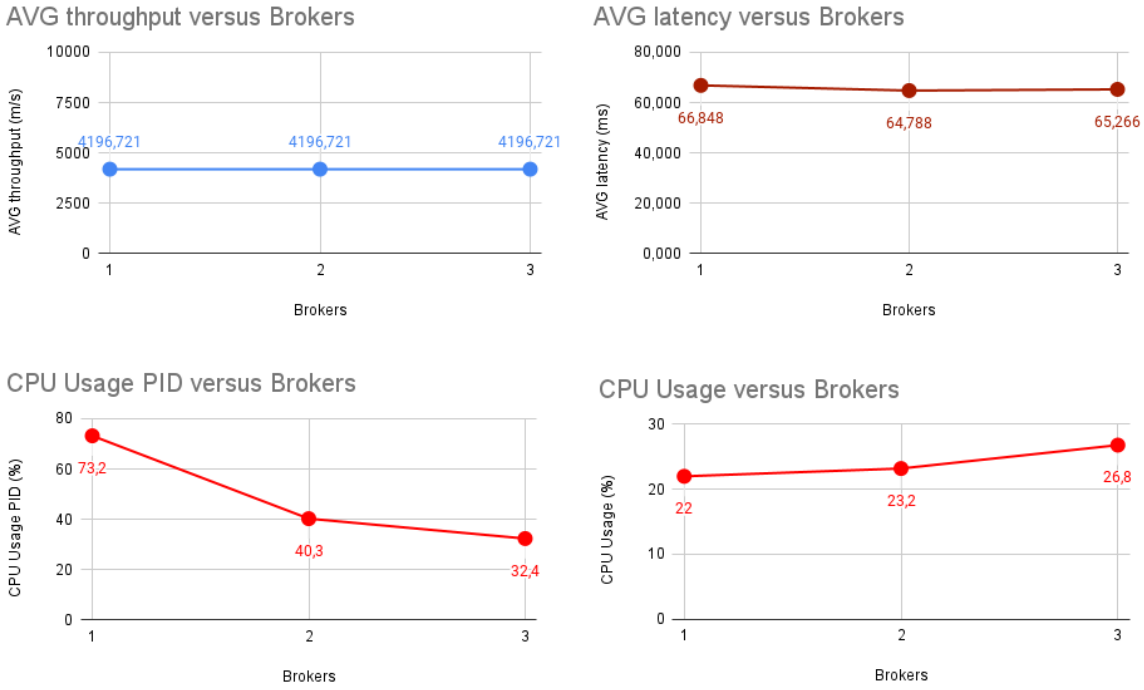

Figura 22 – Resultados Cenário 11

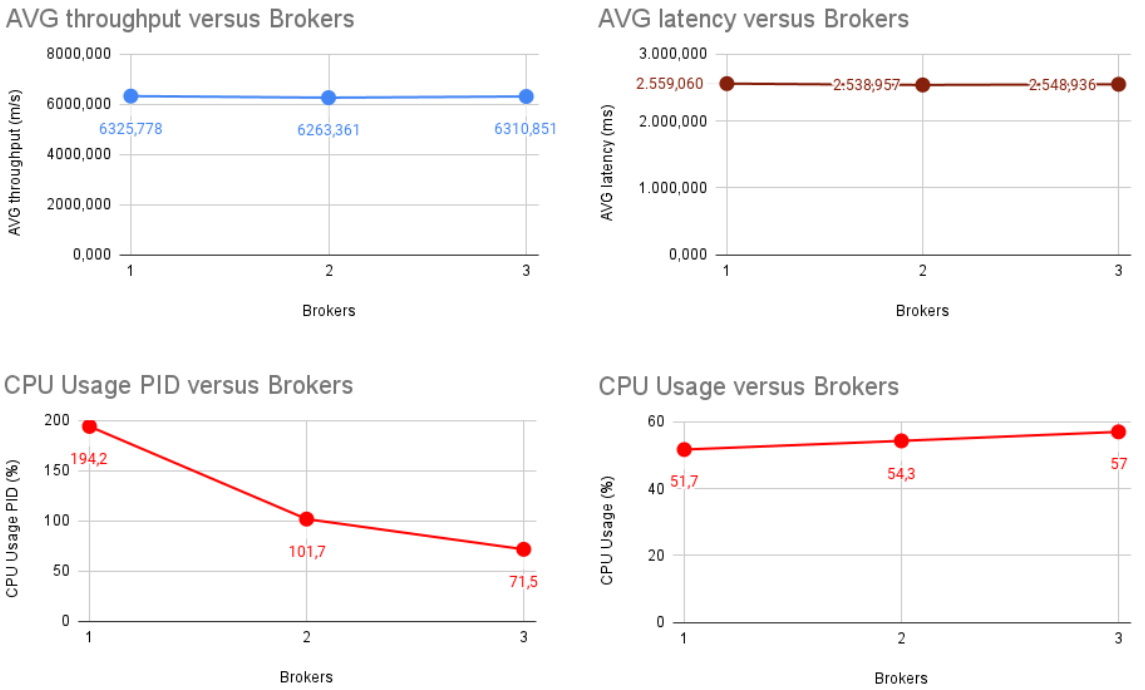

Figura 23 – Resultados Cenário 12

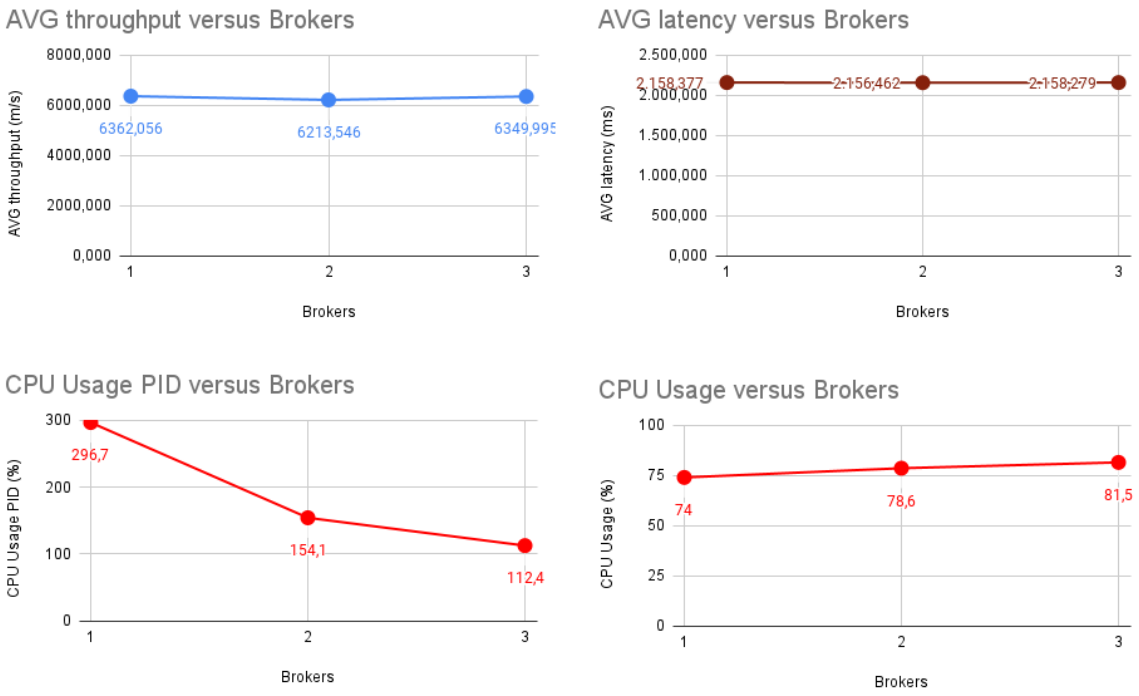

Figura 24 – Resultados Cenário 13

APÊNDICE B - RETORNO MQTTLOADER

Todas as informações do retorno de um teste realizado através do MQTTLoader. A figura em questão mostra um dos testes realizados no Cenário 8.

```
1   -----Publisher-----
2   Maximum throughput [msg/s]: 544
3   Average throughput [msg/s]: 533,333
4   Number of published messages: 32000
5   Per second throughput [msg/s]: 544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512,
6   544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512,
7   544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512,
8   544, 544, 512, 544, 544, 512, 544, 544, 512, 544, 544, 512
9
10  -----Subscriber-----
11  Maximum throughput [msg/s]: 7547
12  Average throughput [msg/s]: 7039,106
13  Number of received messages: 464581
14  Per second throughput [msg/s]: 5600, 6443, 6399, 6618, 6730, 6711, 7022, 6754, 6629, 7292,
15  7361, 7469, 7347, 7341, 7173, 7242, 7007, 7096, 6639, 7384, 7406, 7272, 7445, 7204, 7432,
16  7234, 7094, 6959, 6576, 7307, 7514, 7479, 7526, 7501, 7492, 7520, 7482, 7183, 6814, 7492,
17  7492, 7489, 7498, 7513, 7499, 7489, 7507, 7191, 6784, 7498, 7547, 7516, 7496, 7514, 7463,
18  7499, 7506, 7223, 6751, 7427, 6632, 6093, 6064, 6078, 6708, 915
19  Maximum latency [ms]: 5353,245
20  Average latency [ms]: 2076,770
```

Figura 25 – Retorno MQTTLoader

APÊNDICE C - MONITORAMENTO DE CPU NO GLANCES

Monitoramento do Cenário 5 feito através do Glances. Na figura percebe-se um alto uso de CPU e alguns avisos sobre o fato. Este é um dos casos onde ocorrem os gargalos.

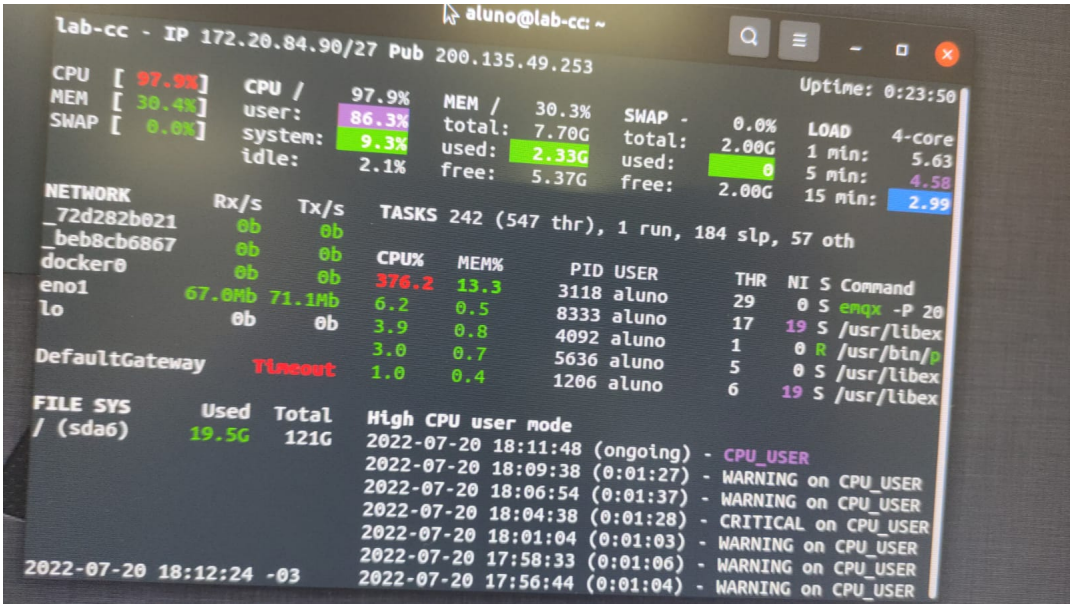

Figura 26 – Monitoramento de CPU no Glances