Análise de desempenho de protocolos para computação em névoa

José Vitor Moreno
Universidade Tecnológica Federal do Paraná
Curitiba, Paraná, Brasil
josemoreno@alunos.utfpr.edu.br

Luís Henrique Beltrão Santana
Universidade Tecnológica Federal do Paraná
Curitiba, Paraná, Brasil
luis.2012@alunos.utfpr.edu.br

Daniel Fernando Pigatto
Universidade Tecnológica Federal do Paraná
Curitiba, Paraná, Brasil
pigatto@utfpr.edu.br

Ana Cristina Barreiras Kochem Vendramin
Universidade Tecnológica Federal do Paraná
Curitiba, Paraná, Brasil
criskochem@utfpr.edu.br

ABSTRACT

The rise in popularity of mobile devices has been made possible due to advances in processing power and the miniaturization of modern processors. Mobile devices are often embedded systems with limited resources such as battery, processing power, and memory, which impact the user experience. These devices located at the edge of the network can also generate large amounts of data, which makes it difficult to centralize the data in the cloud. To address the scalability issues, fog computing is used to intermediate storage and communication services between cloud computing and end devices, allowing for decentralized and scalable data. In fog computing, a message server may be used to distribute information through one or more communication channels, reducing computational resources. To transmit messages, it is necessary to choose a communication protocol, and it is important to consider the limitations of mobile devices when analyzing the behavior of fog communication protocols. This paper evaluates the performance of three fog computing protocols: MQTT, AMQP, and STOMP. Results indicate that MQTT achieves the best performance in terms of power and processing consumption, AMQP has lower memory usage, and STOMP has a shorter round trip time for each message.

KEYWORDS

fog computing; performance evaluation; MQTT; AMQP; STOMP.

1 INTRODUÇÃO

Os sistemas de Internet das Coisas (IoT) têm aumentado significativamente nos últimos anos, devido ao crescente uso de dispositivos móveis e à miniaturização de processadores [Silva 2017]. Segundo Patel and Patel [2016], a busca por expandir o paradigma de IoT trouxe uma série de desafios, como a interconectividade, confiabilidade, heterogeneidade, escalabilidade, entre outros.

Para evitar o comprometimento de recursos em dispositivos IoT, são utilizadas as computações em nuvem e névoa. A computação em nuvem é a disponibilização sob demanda de recursos de computação como serviços na Internet [Amanatullah et al. 2013]. A computação em nuvem tem o problema da latência elevada devido à distância entre os dispositivos IoT e os servidores da nuvem, o que é resolvido com a computação em névoa [Coutinho et al. 2016; Magnoni 2014]. A computação em névoa fornece dados, processamento, armazenamento e serviços mais próximos dos dispositivos de IoT, reduzindo a latência. Além disso, a computação em névoa permite que os recursos sejam alocados de maneira dinâmica e eficiente, o que melhora a escalabilidade e a interconectividade em ambientes IoT [CISCO 2015; Coutinho et al. 2016; Sarkar et al. 2015].

Para lidar com a geração descentralizada e cada vez maior de dados, com a heterogeneidade de dispositivos e com a transmissão dos dados de maneira mais organizada e escalável, pode-se aliar a computação em névoa ao paradigma publicação/subscrição. Esse paradigma possibilita o desacoplamento de um cliente produtor de eventos do(s) cliente(s) consumidor(es) por meio de uma estrutura de tópicos inserida em um elemento intermediário conhecido como servidor de mensagens (em inglês *Message Broker*) [Coulouris et al. 2011]. Segundo Kale [2014], um servidor de mensagens é um *Middleware* Orientado a Mensagem que funciona como uma infraestrutura do tipo cliente/servidor responsável por intermediar a comunicação entre aplicações. Os consumidores realizam subscrições no servidor de mensagem indicando o seu interesse em um determinado tópico e tão logo um produtor publique alguma mensagem no respectivo tópico, o servidor de mensagens envia uma notificação para um ou mais consumidores interessados, com o conteúdo da mensagem publicada [Coulouris et al. 2011; Johnsen 2018; Magnoni 2014]. Dessa forma, clientes heterogêneos (produtores e consumidores) interagem indiretamente através de mensagens assíncronas de notificação de eventos transmitidas por um protocolo de comunicação.

Esse artigo analisa o comportamento de três protocolos de comunicação bastante populares em IoT e na computação em névoa: MQTT (*Message Queuing Telemetry Transport*) [OASIS 2014], AMQP (*Advanced Message Queuing Protocol*) [RabbitMQ 2011] e STOMP (*Simple Text Orientated Messaging Protocol*) [STOMP 2013]. O objetivo é auxiliar na escolha do melhor protocolo de comunicação em um ambiente de computação em névoa dada a limitação de recursos dos dispositivos móveis como energia, processamento e memória.

2 PROTOCOLOS DE COMUNICAÇÃO

O protocolo MQTT [MQTT 2022] foi criado em 1999 e é padronizado pela OASIS [OASIS 2014]. Ele utiliza o modelo de publicação/subscrição e o TCP (*Transmission Control Protocol*) na camada de transporte, proporcionando uma solução leve e eficiente para o envio de mensagens, otimizando o uso de largura de banda na Internet. Devido a estas características, o MQTT tem sido amplamente utilizado em aplicações IoT [Naik 2017]. Segundo Cope [2018] e OASIS [2014], um servidor de mensagens MQTT funciona como um filtro de mensagens baseado em tópicos. Quando um cliente envia mensagens para um servidor de mensagens, ele indica para qual tópico essas mensagens devem ser direcionadas. Os clientes,

com interesse em consumir mensagens, se subscrevem nos tópicos que desejarem. Portanto, um servidor de mensagens MQTT recebe mensagens dos clientes, as separa por seu respectivo tópico e, por fim, as distribui aos clientes que fizeram subscrição em cada tópico.

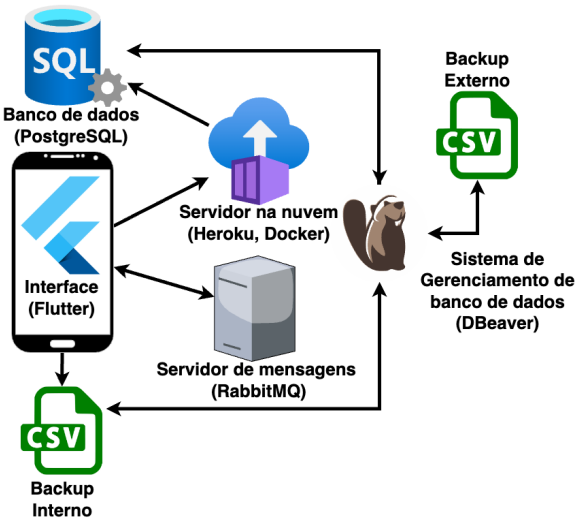
O protocolo de comunicação AMQP [OASIS 2022] foi desenvolvido em 2003 pela empresa JPMorgan para ser utilizado em mercados financeiros. O AMQP foi desenvolvido para ser confiável e seguro, com alto nível de interoperabilidade e provisionamento [Naik 2017]. Por esse motivo, o AMQP é muito utilizado em sistemas IoT que exigem modelos de comunicação robustos e de fácil utilização. RabbitMQ [2011] descreve que em um servidor de mensagens AMQP existe um elemento chamado *exchange*, o qual é utilizado como um tópico em um servidor de mensagens, porém com mais funcionalidades. Um produtor envia as mensagens para um servidor de mensagens direcionando-as para uma determinada *exchange*. Diferentemente dos tópicos, as mensagens não são redirecionadas diretamente para os consumidores, mas para filas contidas no próprio servidor de mensagens AMQP.

O STOMP é um protocolo leve de comunicação assíncrona que também utiliza o modelo publicação/subscrição e cuja comunicação entre os clientes é intermediada por um servidor de mensagens [Mimouni and Bouhdadi 2015; STOMP 2013]. O STOMP foi criado para prover facilidade de uso e compreensão, buscando melhorar a conexão de servidores de mensagens realizada a partir de linguagens de *script*, como Ruby e Python [STOMP 2013]. Mesmo com o seu desenvolvimento ao longo dos anos, o STOMP continua sendo fiel aos seus princípios iniciais de simplicidade e interoperabilidade. Isto resultou em um protocolo de comunicação com um menor conjunto de operações disponíveis para o envio de mensagens.

3 METODOLOGIA

Para analisar o desempenho dos protocolos de comunicação MQTT, AMQP e STOMP utiliza-se a arquitetura ilustrada na Figura 1, onde evidenciam-se os componentes empregados e as suas relações.

Figura 1: Arquitetura do Sistema



O aplicativo desenvolvido na ferramenta Flutter permite a comunicação entre o dispositivo móvel com sistema operacional Android 12 e o servidor de mensagens. O dispositivo móvel é o ponto central da aplicação, atuando como produtor e consumidor de eventos. O RabbitMQ é escolhido como servidor de mensagens devido ao suporte nativo aos protocolos MQTT, STOMP e AMQP e por ser leve, fácil de usar e suportado por várias linguagens de programação. O servidor de mensagens gerencia e distribui os dados relacionados a eventos. Os dados referentes à comunicação entre o dispositivo e o servidor de mensagens são armazenados temporariamente no próprio dispositivo em formato CSV e podem ser transmitidos para um banco de dados externo (PostgreSQL) que é executado como um microsserviço Docker em um servidor de computação em nuvem (Heroku). A ferramenta DBeaver é empregada para sincronizar e gerenciar os dados nos arquivos CSV e no banco de dados.

São realizados 30 experimentos, cada um enviando 10 blocos de 10 mensagens, totalizando 3.000 mensagens trocadas. Um intervalo de 5 segundos é definido para estabilização do dispositivo móvel antes do envio de mensagens. As mensagens são enviadas no formato JSON (*JavaScript Object Notation*). O intervalo entre cada bloco de mensagens é aleatório, variando de 1 ms a 1 segundo para evitar previsibilidade na transmissão de dados pelo dispositivo móvel. Quando não variados, o intervalo de envio de mensagens é de 250 ms e o tamanho das mensagens é de 1000 caracteres.

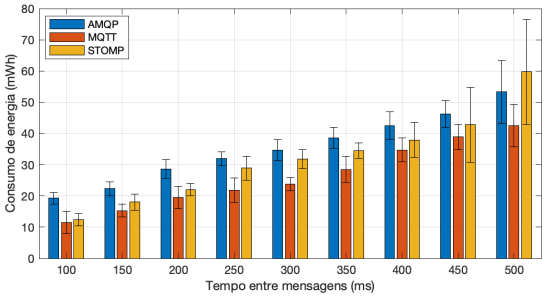
Os protocolos AMQP e STOMP são tratados de forma similar no servidor de mensagens: há uma fila exclusiva para cada protocolo de comunicação, onde o dispositivo móvel publica as mensagens e de onde as consome. No caso do protocolo MQTT, o servidor de mensagens possui um tópico para a publicação de mensagens. O dispositivo se subscreve nesse tópico e envia mensagens para ele. As mensagens do MQTT são enviadas utilizando o nível 1 de QoS (*Quality of Service*), pois é a maior qualidade de serviço que o servidor de mensagens RabbitMQ permite.

4 RESULTADOS

Essa seção apresenta os resultados ao analisar o desempenho do dispositivo móvel durante a troca de mensagens com o servidor de mensagens utilizando os protocolos MQTT, AMPQ e STOMP.

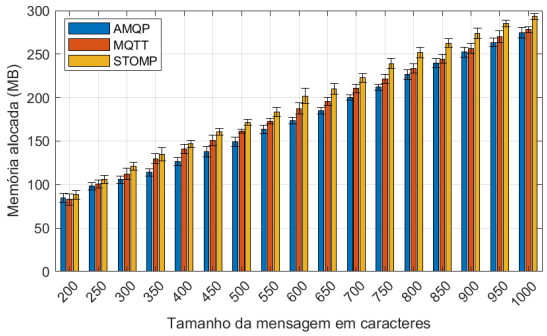
A Figura 2 mostra a média do consumo de energia no dispositivo móvel variando o intervalo de envio de mensagens. O MQTT apresenta o menor consumo de energia em todos os intervalos. Com um tempo entre mensagens menor que 500 ms, o STOMP tem o segundo melhor consumo de energia, enquanto o AMQP tem o pior consumo. Com um tempo de 500 ms, onde menos mensagens são transmitidas, o comportamento observado é o oposto, sendo que o AMQP consome menos energia do que o STOMP.

Figura 2: Consumo de energia



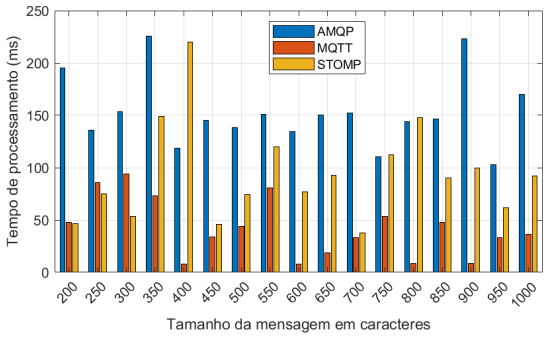
A Figura 3 apresenta a média do consumo de memória em função do tamanho das mensagens. Para mensagens de 200 caracteres, o MQTT obtém um menor consumo médio de memória, enquanto para mensagens acima de 250 caracteres, o AMQP consome menos memória. Apesar das diferenças nos consumos médios com mensagens de 200, 250 e 300 caracteres, o desvio-padrão mostra que os resultados dos três protocolos não são estatisticamente diferentes. Entre mensagens de 350 a 750 caracteres, os resultados são diferentes, com o AMQP apresentando o menor consumo de memória. Acima de 750 caracteres, o AMQP e o MQTT apresentaram resultados estatisticamente iguais. Portanto, conclui-se que o AMQP provê os melhores resultados gerais para o consumo de memória.

Figura 3: Consumo de memória



A Figura 4 mostra o consumo de processamento variando o tamanho das mensagens. Com mensagens de 200 a 300 caracteres, o STOMP apresenta um menor consumo médio de processamento. Entretanto, a partir de mensagens com 350 caracteres, o MQTT obteve os menores valores de uso de processador. Portanto, conclui-se que o MQTT provê o melhor desempenho, de forma geral, em relação ao consumo de processamento.

Figura 4: Consumo de processamento

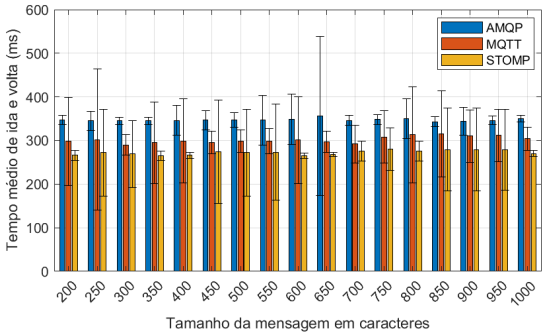


A Figura 5 mostra o tempo de ida e volta de mensagens. Nota-se que a alteração no tamanho das mensagens afeta pouco esse tempo. O protocolo STOMP fornece o melhor desempenho em relação ao tempo de ida e volta de mensagens seguido pelo MQTT e AMQP.

5 CONCLUSÃO

A escolha de um protocolo de comunicação para computação em névoa depende das limitações dos dispositivos onde será implementado. Os resultados obtidos mostram que é necessário avaliar os requisitos de projeto para determinar qual é a melhor escolha. Quando houver limitações quanto ao uso de processamento, o protocolo mais recomendado é o MQTT. Caso o consumo de memória do dispositivo seja limitado, então o melhor protocolo é o AMQP. Caso haja limitação em relação ao consumo de energia, o MQTT e o STOMP são os mais indicados. Caso seja de interesse reduzir o tempo de envio de mensagens, o STOMP é o mais indicado. Como nos experimentos apresentados o MQTT obteve os melhores resultados ou foi o protocolo com o segundo melhor desempenho, ele é o protocolo mais recomendado para dispositivos móveis.

Figura 5: Tempo de ida e volta de mensagens



REFERÊNCIAS

Y. Amanatullah, C. Lim, H. P. Ipung, and A. Juliandri. 2013. Toward Cloud Computing Reference Architecture: Cloud Service Management Perspective. *International Conference on ICT for Smart Society* (2013).

CISCO. 2015. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. *White Paper - Cisco* (2015).

S. Cope. 2018. Beginners Guide To The MQTT Protocol. http://www.steves-internet-guide.com/mqtt/

G. Coulouris, J. Dollimore, and T. Kindberg. 2011. *Distributed Systems.* Addison Wesley Longman. 242 – 253 pages.

Antonio Augusto T. R. Coutinho, Elisângela O. Carneiro, and Fabíola Greve. 2016. Computação em Névoa: Conceitos, Aplicações e Desafios. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* 6 (2016), 266–315.

Frank T. Johnsen. 2018. Using Publish/Subscribe for Short-Lived IoT Data. *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)* (2018).

Vivek Kale. 2014. *Guide to Cloud Computing for Business and Technology Managers.* Chapman and Hall/CRC.

L. Magnoni. 2014. Modern Messaging for Distributed Sytems. *Journal of Physics: Conference Series* (2014).

Sanae E. Mimouni and Mohamed Bouhdadi. 2015. Formal Modeling of the Simple Text Oriented Messaging Protocol using Event-B Method. *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)* (2015).

MQTT. 2022. MQTT: The Standard for IoT Messaging. https://mqtt.org/

Nitin Naik. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Systems Engineering Symposium (ISSE)* (2017).

OASIS. 2014. MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

OASIS. 2022. AMPQ - Advanced Message Queuing Protocol. https://www.amqp.org/

Keyur K. Patel and Sunil M. Patel. 2016. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. *Faculty of Technology and Engineering-MSU* (2016).

RabbitMQ. 2011. AMQP 0-9-1 Model Explained. https://www.rabbitmq.com/tutorials/amqp-concepts.html

S. Sarkar, S. Chatterjee, and S. Misra. 2015. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing* (2015).

Derlone Araújo Jarcelon Silva. 2017. Desenvolvimento e construção de processadores: Uma breve história da micro a nanotecnologia. *Univ. Federal do Maranhão* (2017).

STOMP. 2013. STOMP Protocol Specification, Version 1.2. https://stomp.github.io/stomp-specification-1.2.html