Uma Avaliação do uso de MQTT para a Implementação de Digital Twins

Rafael Trevisan[1], **Francisco Paiva Knebel**[1], **Juliano Araújo Wickboldt**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{rafael.trevisan,francisco.knebel,jwickboldt}@inf.ufrgs.br

Resumo. *O conceito de Digital Twin tem se tornado cada dia mais importante como um método para monitorar elementos e prevenir possíveis defeitos. Uma das tecnologias que pode ser usada para implementar a comunicação de dados desse software é o MQTT. Este trabalho avaliou o uso dessa tecnologia nesse contexto e constatou que existe uma limitação do broker na capacidade de envio de mensagens, que pode ser contornada através da utilização de múltiplos brokers em paralelo. Também foi examinado o comportamento dos brokers em situações onde eles estão centralizados em uma máquina com mais recursos computacionais e quando eles estão distribuídos em máquinas exclusivas, porém com menos recursos. Nessas situações constatamos que os brokers apresentam um comportamento mais estável quando seus serviços estão centralizados.*

1. Introdução

Com o progresso da informática em áreas como a Internet das Coisas (IoT) e inteligência artificial, a indústria tem adotado soluções cada vez mais sofisticadas para a manutenção e monitoramento de seus produtos e estruturas. Uma destas novas soluções é chamada Digital Twin (DT). A definição de DT surgiu em 2002 e tem como objetivo replicar algum elemento que existe em um espaço físico, em um espaço virtual correspondente [Grieves 2016]. Para fazer isso, o DT é atualizado em tempo real com dados vindos de sensores, que captam as características físicas do elemento e as transmitem. O DT então analisa os dados recebidos e é capaz de dar recomendações sobre o uso e manutenção desse elemento. Dessa forma o software pode revelar problemas em potencial para os seus usuários, para que eles sejam resolvidos de forma proativa [Boschert and Rosen 2016].

Para criar uma estrutura como essa, uma das questões que precisa ser resolvida é a necessidade da comunicação entre o DT e os sensores que fornecem dados. Para este fim, o protocolo Message Queue Telemetry Transport (MQTT) é indicado, por ser leve, econômico e compatível com o uso de aparelhos de baixa capacidade computacional [Banks et al. 2019]. O MQTT funciona através de um sistema de *publish*/*subscribe* regulado por um mediador (*broker*). Neste sistema, primeiramente, os clientes que precisam receber dados (*subscribers*) registram os tópicos de seu interesse no *broker*. Em seguida, os clientes que precisam publicar dados (*publishers*) enviam mensagens ao *broker*, que fica então responsável por entregar as informações aos *subscribers*.

Embora o MQTT seja um protocolo considerado confiável para entrega de mensagens, a dependência de um elemento na arquitetura (o *broker*) para mediar toda a

comunicação entre os clientes pode representar um gargalo no sistema. A agilidade para o envio de mensagens depende diretamente do desempenho desse elemento mediador. A aquisição de dados em DTs deve ser projetada como um sistema de tempo real para que a parte virtual possa refletir o seu equivalente real com precisão. O comportamento correto do sistema depende não apenas dos resultados lógicos, mas também do respeito às restrições de tempo real definidas para aquele sistema. Nesse sentido, um estudo sobre o comportamento do *broker* é importante para avaliar a viabilidade do uso desse tipo de comunicação em DTs de larga escala.

Neste trabalho é analisado o comportamento do *broker* em situações onde ele é posto sob estresse considerando a arquitetura prevista pelos DTs e o volume de dados gerado por eles. Estes são cenários onde o *broker* está sendo sobrecarregado e experimentos buscando métodos de lidar com esse fluxo de mensagens distribuindo-as entre mais *brokers* e mais máquinas. Através desses experimentos pode-se observar o comportamento do *broker* em diferentes situações e avaliar quais seriam os impactos no funcionamento e nos requisitos de tempo real de um DT. Através dos resultados obtidos é possível sugerir quais são as formas mais adequadas para a implementação da comunicação do DT e definir suas limitações.

O restante desse trabalho está estruturado da seguinte forma. A seção 2 discute o *background* e os trabalhos relacionados. Na seção 3 são apresentadas a metodologia e os experimentos que foram realizados. Os resultados são discutidos na seção 4 e a seção 5 relata as conclusões obtidas através desses resultados.

2. Trabalhos Relacionados

O conceito de Digital Twin surgiu do processo de *Product Lifecycle Management* (PLM), proposto como uma representação não estática de um produto durante todo o seu ciclo de vida desde a sua fabricação até o fim de seu uso [Grieves 2016]. Desde a criação deste conceito, já era prevista a separação dos espaços reais dos espaços virtuais e o fluxo de dados indo do espaço real para o virtual. Com o tempo o PLM evoluiu para o conceito mais moderno de DT, mantendo as suas principais ideias, mas se adequando as novas demandas do mercado.

Estudos recentes já discutem a possibilidade de implementar um DT com tecnologias de código aberto atuais [Damjanovic-Behrendt and Behrendt 2019]. Segundo Damjanovic-Behrendt e Behrendt, um dos elementos fundamentais para a implementação de um DT são os sistemas de aquisição de dados, do inglês *Data Acquisition Systems* (DAS). Nesse contexto, o MQTT é visto como uma boa alternativa para regular o fluxo das mensagens por ser otimizado para conectar clientes de baixa capacidade computacional com um servidor. Estudos já fizeram análises de desempenho do *broker* MQTT, onde foi possível observar disparidades, sendo elas através da comparação entre diferentes implementações de *broker* ou configurações do protocolo [Mishra 2018]. Porém, esse tipo de análise ainda deixa em aberto outros aspectos que podem influenciar na comunicação entre um sensor e seu DT, como opções de distribuição e balanceamento de carga entre diversos *brokers* a fim de obter escalabilidade na comunicação. Este trabalho irá explorar mais a fundo essas questões com o objetivo de encontrar a maneira mais eficiente de distribuir os elementos da arquitetura de um DT.

3. Metodologia

A avaliação proposta neste trabalho consiste em analisar o comportamento do *broker* MQTT para o envio de volumes de mensagens maiores do que ele é capaz de processar. Esta situação será recriada em diferentes contextos a fim de se determinar qual a configuração mais otimizada para a implementação de um Digital Twin. Para realizar estes experimentos, além do *broker* foram utilizadas outras duas estruturas, o cliente MQTT e a instância de DT.

3.1. *Broker* MQTT

Como *broker* de MQTT foi utilizado o Eclipse Mosquitto versão 1.6.10. Embora existam outras implementações de *brokers* MQTT voltados para a escalabilidade [Pipatsakulroj et al. 2017, Jutadhamakorn et al. 2017] ou para uso em dispositivos embarcados [Espinosa-Aranda et al. 2015], o Mosquitto foi escolhido devido à sua popularidade e ser uma implementação já integrada em produtos comerciais, para uso em dispositivos com recursos limitados [Light 2017]. Ele trabalha ordenando as mensagens recebidas em uma fila e as distribuindo para seus destinatários. Esse software recebe as mensagens e as coloca em uma fila com tamanho limitado, portanto se executarmos esse experimento com as configurações padrão deste *broker*, haveria perda considerável de mensagens. Para contornar isso, as configurações do *broker* foram alteradas para que a limitação de tamanho máximo da fila fosse removido, restando apenas a restrição física da quantidade de memória do computador onde o *broker* executa. A agilidade no envio das mensagens depende do quão eficiente o Eclipse Mosquitto é para processar as regras de encaminhamento e esvaziar a fila.

3.2. Clientes MQTT

Para realizar o experimento foi feita a implementação de um cliente MQTT que envia mensagens de forma controlada. Esse programa cria mensagens do tipo *publish* numeradas sequencialmente com um tamanho de *payload* configurável e as envia ao *broker* utilizando um tópico específico e um intervalo constante também configurável. O momento onde cada uma dessas mensagens é enviada é monitorado através de um *log* gerado localmente pelo cliente.

3.3. Instância de DT

A instância de DT é um protótipo da parte virtual de um twin[1] que implementa os mecanismos necessários para aquisição, armazenamento e processamento de dados oriundos da parte física do DT. Neste estudo, a instância de DT é utilizada como destino das mensagens enviadas pelos clientes MQTT após elas serem distribuídas pelo *broker*. A instância de DT registra as mensagens recebidas em um banco de dados local e gravando também o momento quando uma mensagem é recebida. Neste trabalho não consideramos nenhum tipo de processamento das mensagens recebidas nem envio no sentido inverso (da instância de DT para o cliente), apesar de serem operações possíveis e necessárias, ficam, por hora, fora do escopo do trabalho.

[1]Código aberto disponível em `https://github.com/Open-Digital-Twin`.

3.4. Configuração do Ambiente de Experimentação

Como ambiente de experimentação foi utilizado um servidor com processador Intel Xeon E5-2420 (com 6 núcleos físicos e 12 *threads*) e 32 GB de RAM. Nesse servidor foram instanciadas máquinas virtuais, conectadas a uma rede local também virtual estabelecida através de uma Linux Bridge, e com três configurações de recursos computacionais: Máquinas do tipo 1, com 2 GB de memória e 1 VCPU, máquinas do tipo 2, com 4 GB de memória e 2 VCPUs e Máquinas do tipo 3, com 8 GB de memória e 4 VCPUs.

Considerando esse ambiente, foi conduzido um primeiro experimento enviando 1000 mensagens MQTT *publish*, com *payload* fixado em 64 bytes, do cliente MQTT para uma instância de DT em intervalos de tempo cada vez menores, a fim de definir qual capacidade de envio de mensagem de um único *broker*. Para determinar isso foi analisado o número de mensagens da fila do *broker* em diversos instantes de seu funcionamento.

Para contornar a limitação no fluxo de saída de mensagens de um *broker* individual é possível distribuir sua carga de mensagens para outros *brokers* que funcionem em paralelo considerando que o Eclipse Mosquitto é um processo leve. Em um segundo experimento foi avaliada a diferença de desempenho no envio do mesmo número de mensagens quando elas são distribuídas entre mais *brokers*, considerando dois cenários: (*Centralizado*) onde eles estão centralizados em uma máquina virtual apenas, mas com aumento de recursos computacionais ou (*Distribuído*) onde os *brokers* estão distribuídos em máquinas virtuais com menos recursos, porém exclusivas.

Como esses experimentos foram realizados em um servidor controlado, os resultados obtidos não representam os valores que seriam encontrados ao se executar esses testes com dispositivos reais. Porém, com esses experimentos é possível tirar conclusões sólidas sobre os casos estudados por os cenários se assemelharem muito com a realidade.

4. Resultados

4.1. Experimento 1

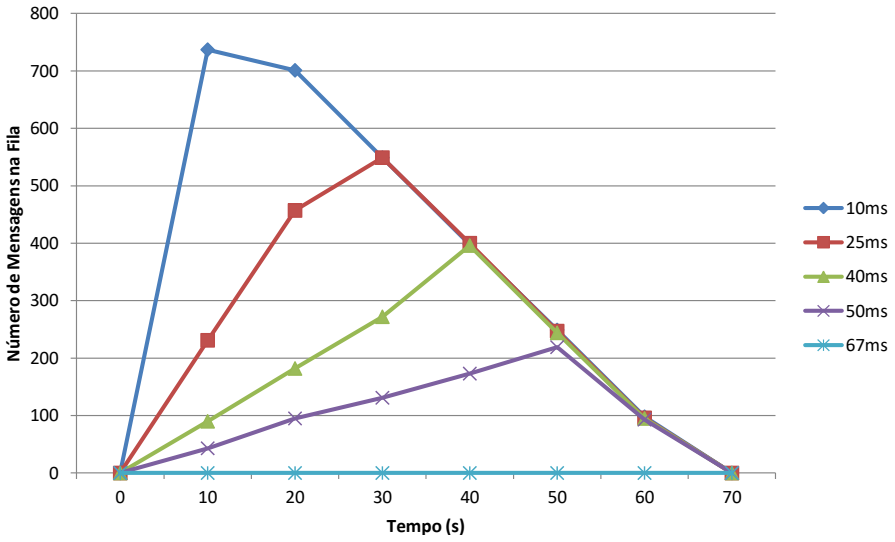

Figura 1. Gráfico que demonstra o tamanho da fila ao longo do tempo.

Baseado na abordagem discutida na seção 3, foi realizado um primeiro experimento para determinar a taxa máxima de envio de mensagens quando se opera com apenas um *broker*. Neste experimento foram enviadas 1000 mensagens pelo MQTT em intervalos gradativamente menores e foi analisado o comportamento do tamanho da fila em diferentes intervalos de tempo. Com essa análise podemos perceber que em cada uma dessas situações é possível determinar a taxa de saída máxima de mensagens do *broker*. Esses testes foram realizados em uma máquina do tipo 3, com todos os serviços centralizados.

Avaliando o comportamento descrito na Figura 1, podemos verificar que o *broker* é capaz de enviar as 1000 mensagens em torno de 67 segundos para intervalos de envio menores que 67ms. Com esses valores conseguimos determinar que o Eclipse Mosquitto consegue enviar no máximo aproximadamente 15 mensagens por segundo nessas condições. Para intervalos de envio maiores que 67 ms, não ocorre a criação de fila e as mensagens são encaminhadas antes das próximas serem recebidas.

4.2. Experimento 2

Para tentar aumentar a vasão de mensagens foi conduzido outro experimento. Nele as mensagens foram enviadas com uma frequência fixa de 10ms, porém essa carga foi distribuídas para mais *brokers* que funcionam em paralelo. Com esse experimento pretendíamos avaliar como eles se comportariam ao resolver as filas criadas. Como base foi utilizado o cenário onde existe apenas um *broker* em uma máquina do tipo 1. A partir deste cenário, foram criados outros dois, o Centralizado e o Distribuído, para verificar se seria mais eficiente aglomerar todos os *brokers* em uma máquina com mais recursos ou distribuí-los em máquinas menores, sempre mantendo uma equivalência de recursos entre os dois cenários. Para o cenário Centralizado foi usada uma máquina do tipo 2 para realizar os testes com 2 *brokers* e uma máquina tipo 3 para realizar os testes com 4. O cenário Distribuído utilizou uma máquina do tipo 1 para cada.

4.2.1. Cenário Centralizado

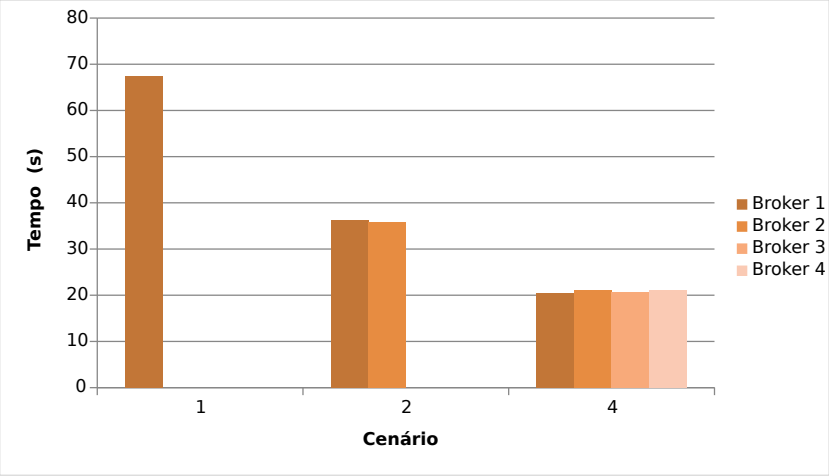

Figura 2. Gráfico que mostra o tempo de operação dos brokers no cenário centralizado.

Com a análise do comportamento dos diferentes *brokers* apresentado na figura 2, podemos perceber que embora distribuir as mensagens seja um bom método para diminuir a quantidade de tempo total de envio de mensagens, a performance individual dos *brokers* é prejudicada. Quando foi utilizado apenas um ele apresentou uma taxa de envio de 14.85 mensagens por segundo. Avaliando a implementação de 2 *brokers* a frequência de envio foi em média 13.84 (27.68 no total). Avaliando 4, ela caiu para 12.02 (48.08 no total).

4.2.2. Cenário Distribuído

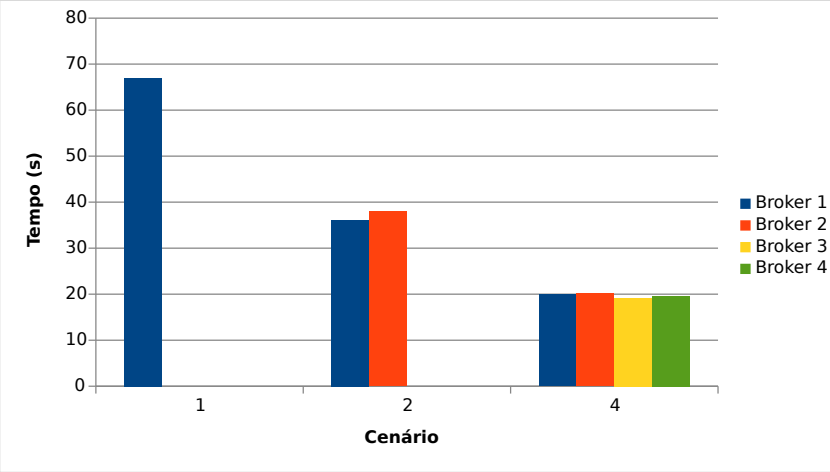

Figura 3. Gráfico que mostra o tempo de operacão dos brokers no cenário distribuído.

Como podemos observar na figura 3, quando dispostos em um ambiente distribuído as mensagens são processadas em uma velocidade similar ao cenário Centralizado. A configuração de dois *brokers* apresentou uma taxa de 13.50 mensagens por segundo (27.00 no total), enquanto a configuração de quatro *brokers* apresentou 12.68 (50.72 no total).

Embora os cenários Centralizado e Distribuído apresentem resultados parecidos, o cenário Distribuído apresenta uma variância consideravelmente maior no tempo tomado por cada um de seus *brokers*. No cenário Centralizado as variâncias apresentadas são de 7.51 e 0.61 para 2 e 4 *brokers* respectivamente. No Cenário Distribuído esses valores aumentaram para 14.36 e 5.24. Esses valores foram encontrados com 5 iterações do experimento.

5. Conclusão

Neste trabalho, estudamos o MQTT com a intenção de usá-lo para o desenvolvimento de um DT. Foram realizados experimentos envolvendo o envio de mensagens em diferentes taxas para determinar as limitações de desempenho de um cenário com um único *broker*. Além disso, foram avaliadas duas configurações viáveis para a arquitetura desse sistema, centralizada ou distribuída, visando melhorar o seu desempenho. Através dos resultados descobrimos que existe um limite da frequência de envio de mensagens através do *broker*, que no caso da configuração do ambiente de experimentação foi de aproximadamente 15

mensagens por segundo. Contornamos essa limitação empregando diversos *brokers* em paralelo em dois cenários (centralizado e distribuído). Ambos os cenários apresentaram resultados similares, porém constatamos que adotar uma arquitetura centralizada tende a gerar uma variância menor nos resultados criando um ambiente mais estável. Embora aglomerar os *brokers* gere um ambiente mais estável, essa organização cria um único ponto de falha no sistema.

Os experimentos realizados não levaram em conta algumas configurações do MQTT que podem influenciar na agilidade do envio de mensagens, como a qualidade do serviço e a persistência das mensagens, mesmo esse fatores tendo baixo impacto no desempenho do *broker*. Eles também não levaram em conta fatores externos que podem impactar na transmissão das mensagens como latência na rede, *jitter* ou perda de pacotes. Como trabalhos futuros, pretendemos realizar mais experimentos levando em conta esses fatores para determinar de forma mais precisa o comportamento do *broker* e suas consequências na implementação de um DT em situações mais realísticas.

Referências

Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). MQTT Version 5.0. `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`. Online; acessado em 09 Ago, 2020.

Boschert, S. and Rosen, R. (2016). Digital twin—the simulation aspect. In *Mechatronic futures*, pages 59–74. Springer.

Damjanovic-Behrendt, V. and Behrendt, W. (2019). An open source approach to the design and implementation of Digital Twins for Smart Manufacturing. *International Journal of Computer Integrated Manufacturing*, 32(4-5):366–384.

Espinosa-Aranda, J. L., Vallez, N., Sanchez-Bueno, C., Aguado-Araujo, D., Bueno, G., and Deniz, O. (2015). Pulga, a tiny open-source MQTT broker for flexible and secure IoT deployments. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 690–694.

Grieves, M. (2016). Origins of the Digital Twin Concept. Technical report, Florida Institute of Technology / NASA.

Jutadhamakorn, P., Pillavas, T., Visoottiviseth, V., Takano, R., Haga, J., and Kobayashi, D. (2017). A scalable and low-cost MQTT broker clustering system. In *2017 2nd International Conference on Information Technology (INCIT)*, pages 1–5.

Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2(13):265.

Mishra, B. (2018). Performance evaluation of MQTT broker servers. In *International Conference on Computational Science and Its Applications*, pages 599–609. Springer.

Pipatsakulroj, W., Visoottiviseth, V., and Takano, R. (2017). mumq: A lightweight and scalable mqtt broker. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE.