Middleware Orientado a Mensagem (MOM)

Antonio Adryson* Vitor Azevedo Silva†

12 de Agosto de 2022

1 Introdução

Com o rápido desenvolvimento da tecnologia da informação, os sistemas tradicionais de aplicação única não podem atender às necessidades reais dos requisitos de negócios, em vez disso, vários sistemas de aplicativos são obrigados a cooperar para resolver o problema de negócios. Cada vez mais aplicações industriais e científicas integram sistemas de aplicativos existentes para formar aplicativos distribuídos, para atender às necessidades de negócios. No entanto, a maioria dos aplicativos é distribuída entre sistemas de hardware heterogêneo, usando diferentes sistemas operacionais em execução no ambiente e contando com diferentes arquiteturas de rede [1].

Ambientes heterogêneos complexos tornam a integração da aplicação muito difícil. Para reduzir o custo de integração de aplicativos distribuídos, geralmente é necessário fornecer serviços de conexão heterogênea blindada e fracamente acoplados através de componentes distribuídos. A tecnologia de *middleware* é uma tipo de tecnologia que ajuda os usuários a resolver a heterogeneidade e problemas de distribuição. MOM (*Message Oriented Middleware*) é uma tecnologia de *middleware* que consiste em um mecanismo de entrega sábio ou um padrão de fila de mensagens, que é frequentemente usado para construir aplicativos distribuídos complexos sistemas. Simplifica a transferência de dados entre aplicativos e protege os sistemas operacionais heterogêneos subjacentes e plataformas [1].

MOM Fornece comunicação consistente e padrões de desenvolvimento de aplicativos para garantir confiabilidade, plataforma de transferência de informações e troca de dados em ambiente de rede. Com o amadurecimento gradual de grandes sistemas distribuídos, os produtos MOM são infinitos e têm suas próprias vantagens. MOM tradicional, foca no desacoplamento do sistema heterogêneo, mensagens assíncronas e fornece soluções de integração de aplicativos distribuídos de modo que tornam seus mecanismos confiáveis na garantia de transações e segurança para torná-lo amplamente utilizado em finanças, *e-commerce* e outros

*vitorsilva@aluno.uespi.br
†vitorsilva@aluno.uespi.br

campos. Com o advento da era da *big data*, o LinkedIn projetou e desenvolveu o Kafka para lidar com os requisitos mais elevados para rendimento de fluxo de dados explosivos. Alibaba projetou e desenvolveu RocketMQ capaz de rodar aplicações de alta confiabilidade e rendimento [1].

Enfrentando com multi-cenários de computação de tarefas, o MOM fornece mensagens serviços de alto rendimento. Ao mesmo tempo, na IoT (*Internet of Things*) ambiente com alto desempenho e restrições de rede. há um grande número de dados e dispositivos heterogêneos, e é necessário fornecer informações mais confiáveis e em tempo real [1].

2 Arquitetura e Características

O MOM é um sistema de *middleware* que permite a vários participantes enviar e receber mensagens em um sistema distribuído. Dentro um sistema distribuído, o MOM permite dois ou mais aplicativos trocar dados na forma de mensagens. As mensagens entre eles são um pacote que inclui dados de negócios e informações de controle. Os participantes do sistema são divididos em produtores de mensagens e consumidores de mensagens de acordo com as mensagens que são usadas. Conforme mostrado na Figura 1, o produtor envia uma mensagem para o MOM usando a interface do usuário fornecido pelo MOM. Após a mensagem chegar ao *middleware*, ela será armazenada na fila especificada de acordo com às informações de controle na mensagem.

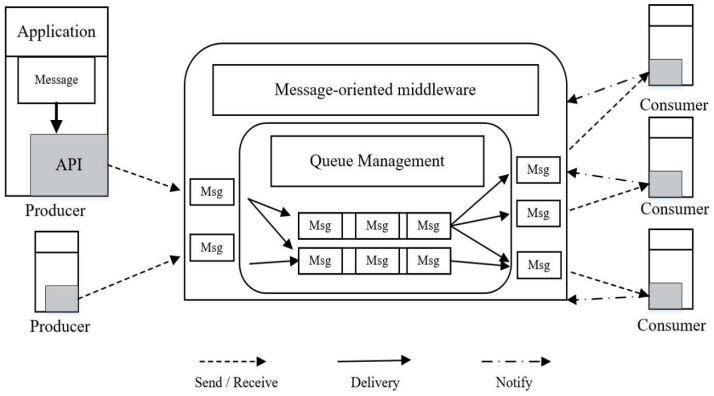

Fig. 1. Basic structure of message middleware.

Figure 1: Exemplo de arquitetura do MOM

O consumidor também usa a interface do usuário para enviar um nome de fila de mensagens ou condição de assinatura ao MOM para indicar as informações da mensagem que o consumidor precisa. Produtores e consumidores são referidos coletivamente como clientes, e cada cliente é consumidor ou pro-

dutor. O MOM transmite a mensagem para o cliente usando um *peer-to-peer* (ponto-a-ponto) ou *pub-sub* (publicação-assinatura) como modo de transporte. A estrutura informada possibilita as seguintes características [**?**]:

1. Mensagens unificadas
2. Provisionamento e monitoramento
3. Escalabilidade dinâmica
4. Gerenciamento e ferramentas de controle
5. Flexibilidade da qualidade do serviço
6. Comunicação segura
7. Facilidade de integração com outras ferramentas

3 Vantagens e Desvantagens

3.1 As Vantagens

1. Fracamente acoplado
2. Escalabilidade
3. Velocidade
4. Reusabilidade
5. Confiabilidade
6. Disponibilidade

3.2 As Desvantagens

1. Requer componentes extras na arquitetura
2. Abstração de programação pobre
3. Abstração de filas uma por uma
4. Não implementável para algumas plataformas

4 Exemplo de Código usando Python

No Listing 1 é mostrado a instalação do Kafka-Python, a versão de utilização do Kafka com o Python, pelo *pip*.

```
$ pip install kafka-python
```

Vai ser criado um *producer* que emite números entre 1 e 1000 que serão mandados para o Kafka. Depois disso, um *consumer* vai ler esses números e os armazenar num banco de dados. Num arquivo *producer.py*, importamos as funções *dumps* da biblioteca *json*, *sleep* da *time* e *KafkaProducer* da biblioteca do Kafka-Python.

```
from time import sleep
from json import dumps
from kafka import KafkaProducer
```

Então será inicializado o novo *producer* em Kafka, passando os parâmetros *bootstrap_servers*, que define a porta que o *producer* irá contatar para inicializar, e *value_serializer*, que vai dizer como os dados serão processados antes de ir para o *broker* (nó do Kafka), onde, no caso do Listing 3, está sendo convertido para um arquivo *json* pela função *dumps* com codificação *utf-8*.

```
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'))
```

Tem-se então uma geração de números entre 1 e 1000 por meio de um *for* com estes números gerados armazenados em um dicionário e enviados para o *broker* a partir da função *send* do *producer*. Nesta função, é fornecido uma etiqueta e o dicionário que foi criado anteriormente. Além disso, é dado 5 segundos de tempo para a operação ser concluída.

```
for e in range(1000):
    data = {'number': e}
    producer.send('numtest', value=data)
    sleep(5)
```

Agora em um arquivo *consumer.py*, para a criação do *consumer* é preciso a importação das funções *loads* do *json*, *KafkaConsumer* do Kafka-Python e *MongoClient* da biblioteca pymongo, que vai ser o banco de dados usado.

```
from kafka import KafkaConsumer
from pymongo import MongoClient
from json import loads
```

Para criar o *consumer* em Kafka, usamos como parâmetros a etiqueta que escolhemos (*numtest* aqui), *bootstrap_servers* para a porta (usando a mesma do *producer*), *auto_offset_reset*, que é onde irá ser definido o local de onde o *consumer* vai começar a ler os dados (será usado *earliest* aqui pois se quer começar do início), *enable_auto_commit* como verdadeiro para que seja enviado o que foi lido depois de cada intervalo, *auto_commit_interval_ms*, que é onde se define o intervalo entre os envios (como foi usado 5 segundos no *producer*, aqui é usado 1 segundo para ser tempo suficiente), *group_id* onde se estabelece o grupo que o *consumer* faz parte, e *value_deserializer*, que é o contrário que o *value_serializer* fez no *producer*, agora decodificando o dado.

```
consumer = KafkaConsumer(
    'numtest',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group',
    value_deserializer=lambda x: loads(x.decode('utf-8')))
```

A extração dos dados do *consumer* é feito com uma repetição em que ele irá ser repetido enquanto o *broker* estiver respondendo. Cada mensagem será substituída pelo seu valor e, então, inseridas na coleção do *MongoClient*. No fim, imprime-se a confirmação de adição.

```
for message in consumer:
    message = message.value
    collection.insert_one(message)
    print('{} added to {}'.format(message, collection))
```

References

[1] Jiang Yongguo, Liu Qiang, Qin Changshuai, Su Jian, and Liu Qianqian. Message-oriented middleware: A review. pages 88–97. Institute of Electrical and Electronics Engineers Inc., 8 2019.