Publicar e assinar

Bianca da Rocha Bartolomei, Carolina Vasques Moreira, Luís Otávio Malta Conceição

[1]Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
Caixa Postal 50 – 37500 903 – Itajubá – MG – Brazil

biancabartolomei, carolinavasques, luiso.malta@unifei.edu.br

1. Introdução

Este trabalho é um projeto para a disciplina de sistemas distribuídos cujo objetivo é o desenvolvimento de um minicurso sobre publicar/assinar, composto por uma parte teórica e outra parte prática. Todo seu desenvolvimento foi baseado no livro Sistemas Distribuídos: Conceitos e Projeto [Coulouris et al. 2013]. A conceituação e explicação do publicar/assinar pode ser encontrada no documento *minucurso.pdf*. Para maior aprofundamento, recomenda-se a leitura do livro base.

Esse documento está divido da seguinte forma:

- Seção 2: descrição da arquitetura da implementação realizada;
- Seção 3: tutorial para execução da implementação.

2. Arquitetura

A implementação realizada nesse trabalho foi a baseada em tópico. Os nós dessa arquitetura são ao mesmo tempo *publicadores* e *assinantes* e se conectam a um único *broker* centralizado, que é o responsável pelo gerenciamento das mensagens trocadas.

Uma das questões relevantes da implementação é como garantir que uma mensagem iria chegar a um nó assinante mesmo que sua conexão fosse interrompida. Para resolvê-la, escolheu-se o uso do banco de dados relacional postgreSQL.

O banco foi modelado da seguinte forma:

- ps.messages(id int *p_key*, corpo varchar, id_pub int *f_key*, id_sub int *f_key*, id_topic *f_key*, created_at timestamp)
- ps.nodes(id int *p_key*, ip varchar, porta varchar)
- ps.topics(id int *p_key*, name varchar)
- ps.topics_nodes(id_topic int *p_key f_key*, id_sub int *p_key f_key*)

Além disso, o banco possui um trigger, chamado de *remove_old_messages*, que remove após uma nova inserção na tabela *ps.messages* todas as mensagens com 30 dias ou mais de criação.

A implementação foi projetada para atender cinco casos de uso:

- **Conexão do nó com o *broker*:** o broker identifica se o nó é novo, se for então o nó é registrado na tabela *ps.nodes* e é retornado um identificador para ele. Se o nó já for registrado, então é chamada uma função que verifica se existem mensagens para ele, e se houver, elas são enviadas.
- **Listagem de tópicos existentes:** o nó solicita ao *broker* uma lista de nós e ele a envia.

- **Publicação de uma mensagem:** um nó envia ao *broker* uma mensagem composta pelo corpo da mensagem e o tópico a qual ela pertence. O *broker* verifica se o tópico já existe no banco de dados. Se não existir, então esse novo tópico é inserido na tabela *ps.topics*. A mensagem é enviada para todos os nós assinantes do tópico dela que estejam conectados no momento. Caso um nó assinante não esteja conectado, então a sua mensagem é armazenada na tabela *ps.messages* e enviada quando ele se conectar novamente ao *broker*.
- **Assinatura em um tópico:** um nó solicita a subscrição ao *broker* que envia a lista de tópicos disponíveis. Ao enviar o tópico da subscrição, o *broker* armazena a relação do assinante na tabela *ps.topic_node*.
- **Cancelamento de assinatura:** ao solicitar o caneclamento de uma assinatura, o *broker* retorna ao nó uma lista de tópicos em que ele está inscrito e pede que o nó responda com o identificador do tópico. Em seguida, o *broker* remove a assinatura da tabela *ps.topic_node*.

3. Tutorial para execução do código

3.1. Requisitos

- Python 3
- PostgreSQL
- psycopg2

```
    pip3 install psycopg2
```

3.2. Configurando a base de dados

1 - Crie o banco de dados

```
createdb <nome_do_banco>
```

2 - Restaure o esquema do banco com o *dump publisher_subscriber*

```
pg_restore <nome_do_banco> publisher_subscriber
```

3 - Altere as informações de login do banco no código do broker.py

```
con = psycopg2.connect(host='127.0.0.1', database='publisher_subscriber', user='postgres', password='suasenha')
```

3.3. Iniciar o *broker*

```
python3 servidor.py
```

Por padrão o *broker* será usado em rede local. Para outras situações é necessário mudar o HOST defino no início do código. O mesmo se aplica para a porta:

```
HOST = 'localhost'
PORT = 1232
```

3.4. Iniciar os nós

```
python3 node.py
```

Adeque o HOST e a porta para combinar com o do *broker*:

```
HOST = '127.0.0.1'
PORT = 1232
```

Referências

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2013). *Sistemas Distribuídos - 5ed: Conceitos e Projeto*. Bookman Editora.