

Simulação de um problema utilizando uma tabela hash distribuída (DHT) em Python

Helena Vargas Tannuri, Vinícios Bidin Santos

Universidade do Estado de Santa Catarina

[helenavargastannuri, vinibidin]@gmail.com

Professor: Guilherme Piegas Goslovski

26/11/2024

Sumário

- 1 Introdução
- 2 Solução
- 3 Proposta
- 4 Implementação
- 5 Resultados
- 6 Dificuldades
- 7 Melhorias
- 8 Referências

Introdução - Descrição do problema

- Um sistema de supermercado com os dados dos clientes armazenados num repositório de dados na nuvem;
- Caso a conexão com o repositório de dados caia, o mercado não deve parar de operar, contornando o problema com a solução proposta;
- A disponibilidade do sistema deve ser 99,99%.

Introdução - Ponto de partida

- Durante a compra, os dados do cliente são recuperados a partir do CPF;
- Garantir a disponibilidade do sistema, mesmo numa situação de falha por omissão por parte do repositório de dados;
- Não podemos contar com a internet: baixa disponibilidade;
- Caso o repositório seja inacessível, como recuperar estes dados?

Estimativas de valores baseadas numa rede de supermercados conhecida:

- 64 lojas;
- 10 milhões de clientes;
- Repositório de dados de tamanho 5TB.

Caso o repositório seja inacessível, como recuperar estes dados?

Redundância e replicação.

- Armazenar os dados localmente em cada loja;
 - Separar o banco em 64 *chunks* e dividi-los ao longo das lojas (cada *chunk* representa uma loja);
 - Definir uma função hash que determina, para um dado CPF, a loja na qual os dados serão armazenados (**módulo é uma boa solução?**).
- Estabelecer um link direto dedicado entre as lojas, formando um anel;
- Caso a nuvem esteja inacessível durante uma compra, utiliza-se a função hash para calcular a loja na qual os dados do cliente estão armazenados e a loja é consultada.

Solução - Ilustração

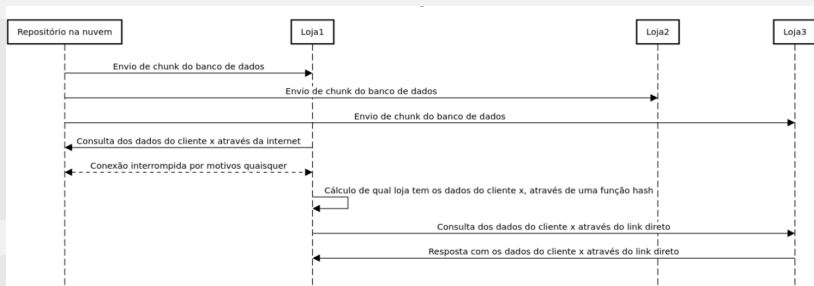


Figura: Diagrama de sequência representando o problema.

Proposta

Simular o funcionamento do sistema caso o repositório da nuvem esteja inacessível, utilizando uma tabela hash distribuída (DHT) em Python, utilizando a biblioteca `kademlia`, que implementa o algoritmo de mesmo nome (MULLER, 2021). Esta DHT em particular possui uma complexidade $\mathcal{O}(\log(n))$ para a maioria das operações realizadas e, por isso, é uma escolha consistente quando comparada com outros algoritmos de DHT (MAYMOUNKOV; MAZIÈRES, 2002).

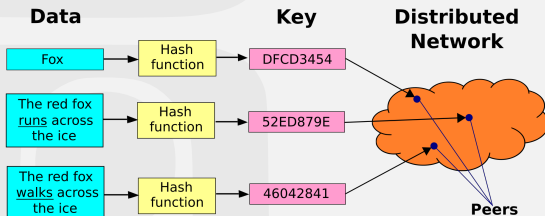


Figura: Ilustração de uma DHT.

Dividimos o sistema em módulos com funcionalidades bem definidas:

- `node.py`
- `set.py`
- `get.py`
- `cpf_generator.py`
- `define_chunks.py`
- `populate_dht.py`

- `create_bootstrap_node(port: int);`
 - Cria e inicializa um nó *bootstrap* para ser o primeiro ponto de entrada na rede;
 - Coloca o nó em estado de escuta na porta especificada.
- `connect_to_bootstrap_node(ip: str, remote_port: int, local_port: int).`
 - Conecta o nó local a um nó existente na rede usando o endereço IP e porta remota;
 - Prepara o nó local para enviar e receber dados na rede DHT;
 - Mantém o nó funcionando enquanto a aplicação está rodando.

- `run(ip: str, remote_port: int, local_port: int, key: str, value: str).`
 - Configura o nó local para escutar em uma porta específica;
 - Conecta o nó local a um nó bootstrap existente na rede;
 - Insere uma chave e um valor na DHT.
 - Encerra a conexão após a conclusão.

- `run(ip: str, remote_port: int, local_port: int, key: str).`
 - Configura o nó local para escutar em uma porta específica;
 - Conecta o nó local a um nó bootstrap existente na rede;
 - Recupera o valor associado a uma chave fornecida;
 - Encerra o servidor após a conclusão.

- `gerar_cpfs(quantidade)`.
 - Gera uma dada quantidade de CPFs e os salva num arquivo de texto.

- `split_in_chunks(chunks: int)`.
 - Lê o arquivo com os CPFs e, a partir da função de módulo, divide ele em um número fornecido de *chunks*.

- `populate()`.
 - Lê o arquivo de cada *chunk* e utiliza a função do módulo *set.py* para inserir a chave (CPF) e valor (loja correspondente) na DHT.

Resultados - Criando o primeiro nó da rede

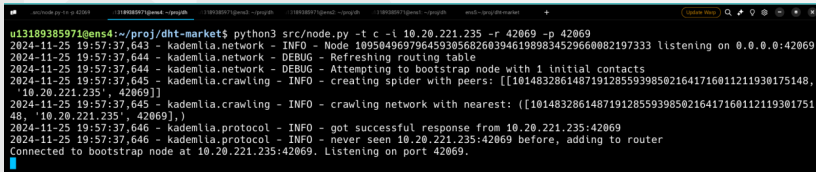


```
src/node.py 4 in p 42069
u13189385971@ens5: ~/proj/dht-market git:(main) (0.046s)
clear

u13189385971@ens5 ~/proj/dht-market git:(main):$
python3 src/node.py -t n -p 42069
2024-11-25 19:57:21,720 - kademia.network - INFO - Node 1014832861487191285593985021641716011211930175148 listening on 0.0.0.0:42069
2024-11-25 19:57:21,721 - kademia.network - DEBUG - Refreshing routing table
```

Figura: Primeiro nó da rede

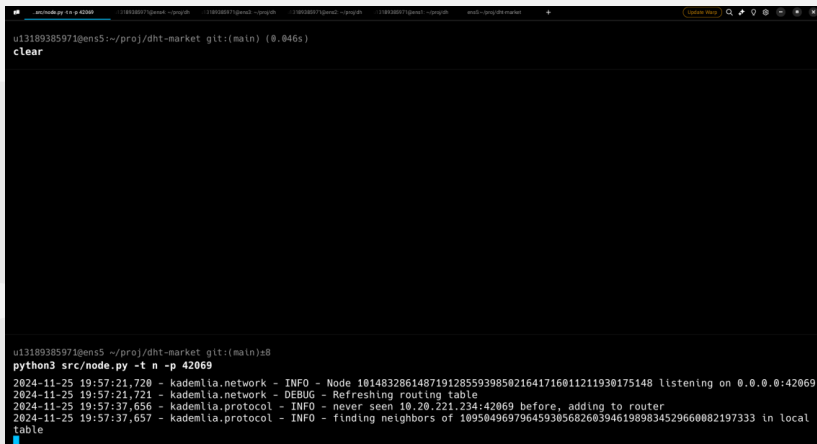
Resultados - Criando o Segundo nó e conectando na rede



```
u13189385971@ens4:~/proj/dht-market$ python3 src/node.py -t c -i 10.20.221.235 -r 42069 -p 42069
2024-11-25 19:57:37,643 - kademia.network - INFO - Node 1095049697964593056826039461989834529660082197333 listening on 0.0.0.0:42069
2024-11-25 19:57:37,644 - kademia.network - DEBUG - Refreshing routing table
2024-11-25 19:57:37,644 - kademia.network - DEBUG - Attempting to bootstrap node with 1 initial contacts
2024-11-25 19:57:37,645 - kademia.crawling - INFO - creating spider with peers: [[1014832861487191285593985021641716011211930175148,
'10.20.221.235', 42069]]
2024-11-25 19:57:37,645 - kademia.crawling - INFO - crawling network with nearest: ([10148328614871912855939850216417160112119301751
48, '10.20.221.235', 42069],)
2024-11-25 19:57:37,646 - kademia.protocol - INFO - got successful response from 10.20.221.235:42069
2024-11-25 19:57:37,646 - kademia.protocol - INFO - never seen 10.20.221.235:42069 before, adding to router
Connected to bootstrap node at 10.20.221.235:42069. Listening on port 42069.
```

Figura: Segundo nó da rede

Resultados - Resposta da conexão



```
src/node.py 6 n p 42069
u13189385971@ens5: ~/proj/dht-market git:(main) (0.046s)
clear

u13189385971@ens5 ~/proj/dht-market git:(main)±8
python3 src/node.py -t n -p 42069
2024-11-25 19:57:21,720 - kademia.network - INFO - Node 1014832861487191285593985021641716011211930175148 listening on 0.0.0.0:42069
2024-11-25 19:57:21,721 - kademia.network - DEBUG - Refreshing routing table
2024-11-25 19:57:37,656 - kademia.protocol - INFO - never seen 10.20.221.234:42069 before, adding to router
2024-11-25 19:57:37,657 - kademia.protocol - INFO - finding neighbors of 1095049697964593056826039461989834529660082197333 in local table
```

Figura: Resposta da conexão

Resultados - Última resposta

```
u13189385971@ens5:~/proj/dht-market git:(main) (0.046s)
clear

u13189385971@ens5 ~/proj/dht-market git:(main)±8
python3 src/node.py -t n -p 42069
2024-11-25 19:57:21,720 - kademlia.network - INFO - Node 1014832861487191285593985021641716011211930175148 listening on 0.0.0.0:42069
2024-11-25 19:57:21,721 - kademlia.network - DEBUG - Refreshing routing table
2024-11-25 19:57:37,656 - kademlia.protocol - INFO - never seen 10.20.221.234:42069 before, adding to router
2024-11-25 19:57:37,657 - kademlia.protocol - INFO - finding neighbors of 1095049697964593056826039461989834529660082197333 in local
table
2024-11-25 19:57:46,214 - kademlia.protocol - INFO - never seen 10.20.221.233:42069 before, adding to router
2024-11-25 19:57:46,216 - kademlia.protocol - INFO - finding neighbors of 630229889140288091358019147562514972994246523768 in local t
able
2024-11-25 19:58:00,200 - kademlia.protocol - INFO - never seen 10.20.221.232:42069 before, adding to router
2024-11-25 19:58:00,202 - kademlia.protocol - INFO - finding neighbors of 448513671561056775058715580719874602726378794619 in local t
able
2024-11-25 19:58:10,334 - kademlia.protocol - INFO - never seen 10.20.221.231:42069 before, adding to router
2024-11-25 19:58:10,336 - kademlia.protocol - INFO - finding neighbors of 643294416616256972188501489313376235909091138734 in local t
able
```

Figura: Última resposta de conexão

Resultados - Populando a DHT

```
u13189385971@ens5 ~/proj/dht-market glt:(main)±8 (1m 2.75s)  
./migrate.sh --ip 10.20.221.235 --remote-port 42069 --port 42070  
  
Gerando CPFs válidos: 100% | ████████████████████████████████████████ | 10000/10000 [00:00<00:00, 37402.72it/s]  
10000 CPFs válidos foram gerados e salvos em 'db/cpfs.txt'.  
Separating CPFs into chunks: 100% | ████████████████████████████████████████ | 5/5 [00:00<00:00, 442.32it/s]  
10000 CPFs foram separados em 5 chunks e salvos em db/chunk{1..5}.txt'.  
Connected to bootstrap node at 10.20.221.235:42069  
Processed chunk 1  
Processed chunk 2  
Processed chunk 3  
Processed chunk 4  
Processed chunk 5  
DHT populated with 5 chunks.  
  
u13189385971@ens5 ~/proj/dht-market glt:(main)±8
```

Figura: Populando a DHT

Resultados - Resposta da população

```
u13189385971@ens5:~/proj/dht-market git:(main) (0.046s)
clear

u13189385971@ens5 ~/proj/dht-market git:(main)±8
python3 src/node.py -t n -p 42069
2024-11-25 19:57:21,720 - kademia.network - INFO - Node 1014832861487191285593985021641716011211930175148 listening on 0.0.0.0:42069
2024-11-25 19:57:21,721 - kademia.network - DEBUG - Refreshing routing table
2024-11-25 19:57:37,656 - kademia.protocol - INFO - never seen 10.20.221.234:42069 before, adding to router
2024-11-25 19:57:37,657 - kademia.protocol - INFO - finding neighbors of 1095049697964593056826039461989834529660082197333 in local
table
2024-11-25 19:57:46,214 - kademia.protocol - INFO - never seen 10.20.221.233:42069 before, adding to router
2024-11-25 19:57:46,216 - kademia.protocol - INFO - finding neighbors of 630229889140288091358019147562514972994246523768 in local t
able
2024-11-25 19:58:00,200 - kademia.protocol - INFO - never seen 10.20.221.232:42069 before, adding to router
2024-11-25 19:58:00,202 - kademia.protocol - INFO - finding neighbors of 448513671561056775058715580719874602726378794619 in local t
able
2024-11-25 19:58:10,334 - kademia.protocol - INFO - never seen 10.20.221.231:42069 before, adding to router
2024-11-25 19:58:10,336 - kademia.protocol - INFO - finding neighbors of 643294416616256972188501489313376235909091138734 in local t
able
```

Figura: Resposta da população

Resultados - Set

```
u13189385971@ens5 ~  
ssh ens1  
0d5ec493412ac044190d2='4'  
2024-11-25 20:52:56,364 - kademia.protocol - INFO - got successful response from 10.20.221.235:42069  
2024-11-25 20:52:56,365 - kademia.crawling - INFO - crawling network with nearest: ([10849324474945217617254152777322710183654559372  
8, '10.20.221.234', 42069], [106690975456132861634211251344457143798900184233, '10.20.221.235', 42070], [5299867905699168516756554077  
93345187447844006719, '10.20.221.233', 42069], [546955495182841102032158262165707390688369615308, '10.20.221.232', 42069], [113119668  
7911882932427893867948918268526473282585, '10.20.221.235', 42070], [1326160882426443228751170823581794818487039892469, '10.20.221.235  
, 42069])  
2024-11-25 20:52:56,368 - kademia.protocol - INFO - got successful response from 10.20.221.234:42069  
2024-11-25 20:52:56,368 - kademia.protocol - INFO - never seen 10.20.221.234:42069 before, adding to router  
2024-11-25 20:52:56,369 - kademia.protocol - INFO - got successful response from 10.20.221.235:42070  
2024-11-25 20:52:56,369 - kademia.protocol - INFO - never seen 10.20.221.235:42070 before, adding to router  
2024-11-25 20:52:56,369 - kademia.protocol - INFO - got successful response from 10.20.221.233:42069  
2024-11-25 20:52:56,369 - kademia.protocol - INFO - never seen 10.20.221.233:42069 before, adding to router  
2024-11-25 20:52:56,370 - kademia.crawling - INFO - crawling network with nearest: ([10849324474945217617254152777322710183654559372  
8, '10.20.221.234', 42069], [106690975456132861634211251344457143798900184233, '10.20.221.235', 42070], [5299867905699168516756554077  
93345187447844006719, '10.20.221.233', 42069], [546955495182841102032158262165707390688369615308, '10.20.221.232', 42069], [113119668  
7911882932427893867948918268526473282585, '10.20.221.235', 42070], [1326160882426443228751170823581794818487039892469, '10.20.221.235  
, 42069])  
2024-11-25 20:52:56,373 - kademia.protocol - INFO - got successful response from 10.20.221.235:42070  
2024-11-25 20:52:56,373 - kademia.protocol - INFO - never seen 10.20.221.235:42070 before, adding to router  
2024-11-25 20:52:56,373 - kademia.protocol - INFO - got successful response from 10.20.221.232:42069  
2024-11-25 20:52:56,374 - kademia.protocol - INFO - never seen 10.20.221.232:42069 before, adding to router  
2024-11-25 20:52:56,374 - kademia.protocol - DEBUG - got a store request from ('10.20.221.232', 42069), storing '62b36034b6eb3dd9b26  
0eeec96215184baf02eed='4'  
Connected to bootstrap node at 10.20.221.235:42069. Listening on port 42069.  
2024-11-25 20:54:58,178 - kademia.protocol - INFO - never seen 10.20.221.235:42070 before, adding to router  
2024-11-25 20:54:58,180 - kademia.protocol - INFO - finding neighbors of 1268304880664731754728019198789477067658619674858 in local  
table  
2024-11-25 20:54:58,188 - kademia.protocol - INFO - finding neighbors of 1268304880664731754728019198789477067658619674858 in local  
table  
2024-11-25 20:54:58,194 - kademia.protocol - DEBUG - got a store request from ('10.20.221.235', 42070), storing '1f8ac10f23c5b5bc116  
7bda84b8335ec057a77d2='123456'
```

Figura: Set chave e valor

Resultados - Get

```
u13189385971@ens5 ~/proj/dht-market glt:(main)±8 (9m 48.03s)
python3 src/node.py -t c -i 10.20.221.235 -r 42069 -p 42070

2024-11-25 20:46:02,410 - kademia.protocol - INFO - got successful response from 10.20.221.234:42069
2024-11-25 20:47:52,567 - kademia.protocol - INFO - finding neighbors of 529986790569916851675655407793345187447844006719 in local
able
2024-11-25 20:47:52,567 - kademia.protocol - INFO - never seen 10.20.221.233:42069 before, adding to router
2024-11-25 20:47:52,567 - kademia.protocol - INFO - finding neighbors of 529986790569916851675655407793345187447844006719 in local
able
2024-11-25 20:47:52,569 - kademia.protocol - INFO - got successful response from 10.20.221.233:42069
2024-11-25 20:52:17,458 - kademia.protocol - INFO - finding neighbors of 546955495182841102032158262165707390688369615308 in local
able
2024-11-25 20:52:17,458 - kademia.protocol - INFO - never seen 10.20.221.232:42069 before, adding to router
2024-11-25 20:52:17,461 - kademia.protocol - INFO - finding neighbors of 546955495182841102032158262165707390688369615308 in local
able
2024-11-25 20:52:56,376 - kademia.protocol - INFO - finding neighbors of 12272131176174051862278955071036862734061635836 in local
able
2024-11-25 20:52:56,377 - kademia.protocol - INFO - never seen 10.20.221.231:42069 before, adding to router
2024-11-25 20:52:56,381 - kademia.protocol - INFO - finding neighbors of 12272131176174051862278955071036862734061635836 in local
able
^CShutting down the node.

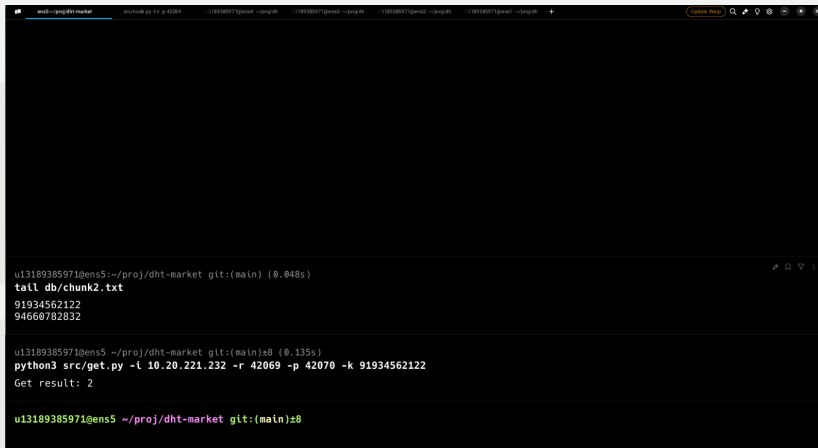
u13189385971@ens5 ~/proj/dht-market glt:(main)±8 (0.126s)
python3 src/set.py -i 10.20.221.231 -r 42069 -p 42070 -k abcdef -v 123456

u13189385971@ens5 ~/proj/dht-market glt:(main)±8 (0.128s)
python3 src/get.py -i 10.20.221.232 -r 42069 -p 42070 -k abcdef
Get result: 123456

u13189385971@ens5 ~/proj/dht-market glt:(main)±8
```

Figura: Get chave

Resultados - Get de um cpf na database



```
u13189385971@ens5: ~/proj/dht-market git:(main) (0.048s)
tail db/chunk2.txt
91934562122
94660782832

u13189385971@ens5 ~/proj/dht-market git:(main)±8 (0.135s)
python3 src/get.py -l 10.20.221.232 -r 42069 -p 42070 -k 91934562122
Get result: 2

u13189385971@ens5 ~/proj/dht-market git:(main)±8
```

Figura: Get de um cpf na database

Comparação dos tempos de Get para diferentes tamanhos de database

CPFs	Tempo de Get
10	0,135s
100	0,14s
1000	0,233s
10000	0,744s
100000	1.432s

Table: Tempo de processamento por quantidade de CPFs.

Comparação dos tempos de Get para diferentes tamanhos de database

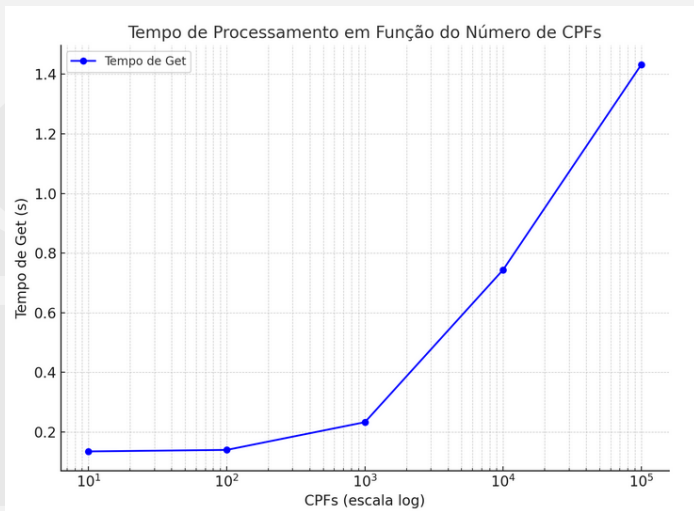


Figura: Gráfico em escala logarítmica

Comparação dos tempos de Get para diferentes tamanhos de database

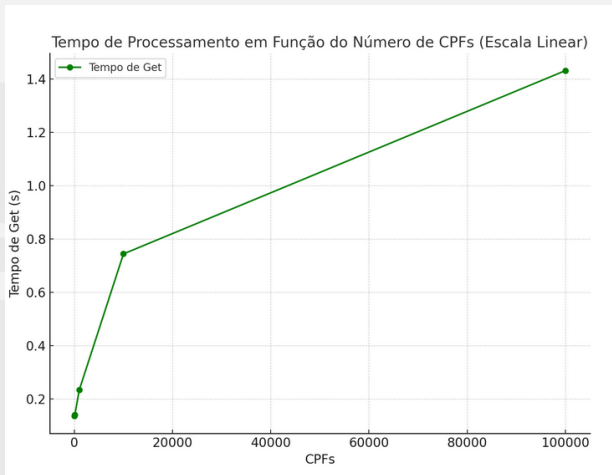



Figura: Gráfico em escala normal

Durante o desenvolvimento do trabalho final, encontramos algumas dificuldades relacionadas com a implementação da biblioteca kademlia e com a natureza dos dados.

- A implementação da biblioteca peca na remoção de um nó quando este se desconecta da rede;
- Os CPFs não possuem uma distribuição uniforme em relação ao módulo, portanto, algumas lojas ficam com mais clientes armazenados que outras.

Seria possível expandir e melhorar o sistema de diversas formas:

- Estabelecer uma função hash que distribua de maneira mais uniforme os CPFs ao longo das lojas;
- Modificar a DHT para que clientes frequentes de uma determinada loja tenham seus dados armazenados nela, aproximando os dados do cliente e otimizando a consulta;
- Implementar uma DHT do zero, independente da apresentada na biblioteca, para melhor atender as necessidades específicas do sistema em relação a atualização da tabela de nós.

 MAYMOUNKOV, P.; MAZIÈRES, D. Kademlia: A peer-to-peer information system based on the xor metric. In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Berlin, Heidelberg: Springer-Verlag, 2002. (IPTPS '01), p. 53–65. ISBN 3540441794.

 MULLER, B. *Kademlia*. 2021. Disponível em: <<https://pypi.org/project/kademlia/>>.