TESTE DE PERFORMANCE DO BANCO DE DADOS REDIS

Ellen Bastos da Silva Fiscina

Brauliro Gonçalves Leal

RESUMO: Foi feita uma avaliação de performance do gerenciador de banco de dados Redis 5.8 ao executar 60 blocos de 500 mil operações (*insert*, *select* e *update*). A duração média das operações unitárias (DOU) de *insert*, *select* e *update* foram iguais a 0,0004919, 0,0004778 e 0,0004919 s, respectivamente. As equações ajustadas para os valores de duração acumulada das operações (DAO) em função do número de operações (NOP) foram DAO = 0,00049093163512 × NOP + 9,12782245045582, com R^2 = 0,99999971402487 para as operações de *insert*; DAO = 0,00047913525719 × NOP - 10,95460768622710, com R^2 = 0,99999979479763, para as operações de *select*; e DAO = 0,00049096616731 × NOP + 8,31555634164579, com R^2 = 0,99999965861478, para as operações de *update*. Todas as operações tiveram durações similares, principalmente o *insert* e o *update*, que tiveram a mesma média de tempo. O *select* foi a operação que utilizou menos tempo computacional. Os resultados indicaram que o banco de dados Redis mantém consistência de performance em suas três operações básicas, mesmo que operando uma grande quantidade de registros.

PALAVRAS-CHAVE: Banco de Dados, Redis, Performance.

PERFORMANCE TEST OF REDIS DATABASE

ABSTRACT: The Redis 5.8 database was used to insert 60 blocks of 500 thousand data to verify the database performance in situations of insertion, selection and update. The average duration of the single operations insert, select and update were 0.0004919, 0.0004778 and 0.0004919 s respectively. The adjusted equations to the values of operations accumulated duration (DAO) in function of the number of operations (NOP) were DAO = 0,00049093163512 × NOP + 9,12782245045582, with R^2 = 0,99999971402487 to insertion; DAO = 0,00047913525719 × NOP - 10,95460768622710, with R^2 = 0,99999979479763, to selection; and DAO = 0,00049096616731 × NOP + 8,31555634164579, with R^2 = 0,99999965861478, to the update operations. All operations had similar duration, specially insert and update, which had the same average time. Select was the operation that used the least computational time. The results indicated that the Redis database maintains

performance consistency in its three basic operations, even when operating a large amount of records.
KEYWORDS: Database, Redis, Performance.

INTRODUÇÃO:

Bancos de dados são estruturas que armazenam dados. São úteis para manipulação de registros, permitindo acessar, modificar, incluir e remover dados. Os bancos de dados, em geral, são gerenciados por Sistemas de Gerenciamento de Banco de Dados (SGBD) cujo principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, manipulação e organização dos dados.

Redis é um banco de dados não relacional, ou NoSQL (*Not Only SQL* - Não Apenas SQL), que armazena um mapeamento de chaves a cinco tipos diferentes de valores. Esses valores podem ser estruturas de dados do tipo *string*, *list*, *set*, *hash* ou *sorted set* (*zset)*. O Redis tem suporte a armazenamento persistente em disco e oferece duas maneiras de escrever esses dados automaticamente. Suas funcionalidades e características fazem com que ele possa ser usado como um banco de dados primário ou como um banco auxiliar, em conjunto com outros sistemas de armazenamento.

O Redis possui comandos compartilhados entre todos os tipos de dados, e comandos vinculados a um só tipo. Isso permite que o código seja curto, fácil de entender e de manter. Além disso, o Redis é considerado um banco de dados rápido, pois não é preciso ler a base de dados para atualizá-la.

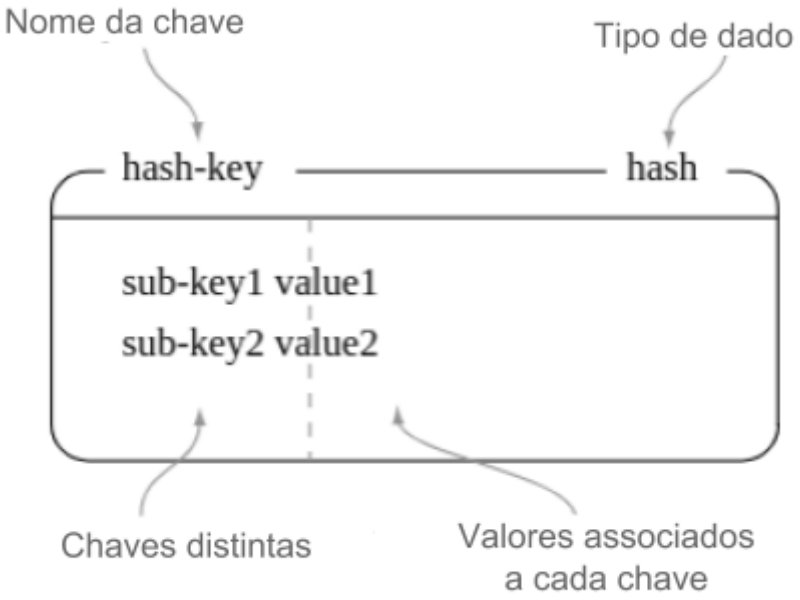
Algumas aplicações conhecidas do Redis incluem Twitter, Github, Hulu, Airbnb e Pinterest, pela sua capacidade de processar grandes quantidades de dados em tempo real.

O presente trabalho visa testar, através de um programa sintético escrito em Python 3.7, com o auxílio da aplicação web Jupyter Notebook, a performance do banco de dados Redis em função de múltiplas inserções, seleções e atualizações de dados por um determinado período de tempo.

MATERIAIS E MÉTODOS:

O Redis oferece cinco tipos de estrutura de dados: *string*, *list*, *set*, *hash* ou *sorted set* (*zset)*. Dentre estes foi escolhido o *hash* para simbolizar as linhas da tabela de um banco de dados relacional. O *hash* criado possui oito campos, conforme descrito no Quadro 01. A Figura 01 mostra a estrutura de um *hash* no Redis. 30 milhões de *hashes* foram inseridos usando esta mesma estrutura.

Figura 01 - Exemplo de um *hash*



Fonte: https://redislabs.com/ebook/part-1-getting-started/chapter-1-getting-to-know-redis/1-2-what-redis-data-structures-look-like/1-2-4-hashes-in-redis/

Quadro 01 - *Hash* inserido no banco de dados.

TableX	
Chave	Tipo
A0	INTEGER NOT NULL
A1	INTEGER
A2	NUMERIC
A3	NVARCHAR
A4	VARCHAR
A5	TIME
A6	DATE

Foi desenvolvido um programa em Python 3.7 para realizar 30 milhões de operações insert, select e update, divididos em 60 blocos de 500 mil. O desempenho foi analisado através da duração acumulada das operações e da duração média da operações unitárias por bloco de insert, select e update em função do número de operações. Utilizou-se a biblioteca redis-py que promove uma interface de conexão entre o Python e o Redis. Também foi utilizada a biblioteca *time* do Python, para marcar a duração de cada bloco.

Para auxiliar na análise de dados utilizou-se ainda as bibliotecas csv, para gravar os dados em uma planilha, pandas e matplotlib para plotar os gráficos.

Quadro 02 - Especificações de *Hardware*

HARDWARE	
Processador	Intel i7 4770
CPU	3,40 GHz
RAM	16 GB

Quadro 03 - Especificações de *Software*

SOFTWARE	
Sistema Operacional	**Tipo**
Windows 10 Pro	64 bits
Módulo	**Versão**
Python	3.6.5
redis-py	3.0.1
csv	1.0
pandas	0.23.0
matplotlib	2.2.2

Os Quadro 02 e 03 mostram as especificações do *hardware* e *software*, respectivamente, utilizados neste projeto.

RESULTADO E DISCUSSÃO:

O quadro 04 apresenta a duração acumulada das operações (DAO) e a duração média das operações unitárias (DOU) por bloco de *insert*, *select* e *update* em função do número de operações.

Os valores da DAO variaram de 243 a 14737,62 s; 237 a 14363,21 s; e 247 a 14736 s para as operações de *insert*, *select* e *update*, respectivamente. Por outro lado, os valores médios da DOU para as operações *insert*, *select* e *update* foram iguais a 0,0004919, 0,0004778 e 0,0004919 s, respectivamente. As figuras 2.a, 2.b e 2.c mostram os gráficos da variação dos valores do DAO em função do número de operações *insert*, *select* e *update*, respectivamente. As figuras 2.d, 2.e, 2.f mostram os gráficos da variação dos valores do

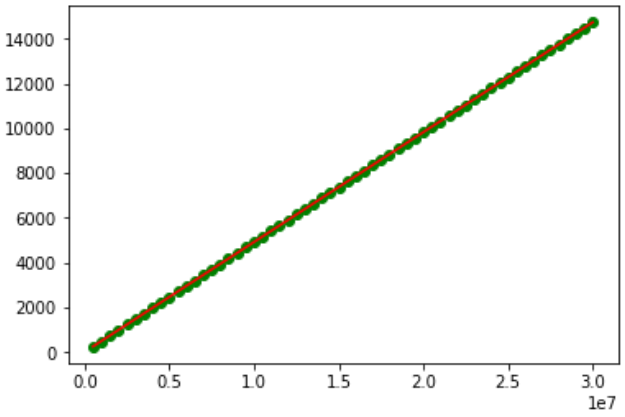
DOU em função do número de operações *insert*, *select* e *update*, respectivamente e uma reta marcando a duração média de cada bloco.

As figuras 2.a, 2.b e 2.c também apresentam equações ajustadas para os valores do DAO em função do número de operações. Para as operações de *insert*, DAO = 0,00049093163512 × NOP + 9,12782245045582, com R^2 = 0,99999971402487; para o *select*, DAO = 0,00047913525719 × NOP - 10,95460768622710, com R^2 = 0,99999979479763; e DAO = 0,00049096616731 × NOP + 8,31555634164579, com R^2 = 0,99999965861478, para as operações de *update.*

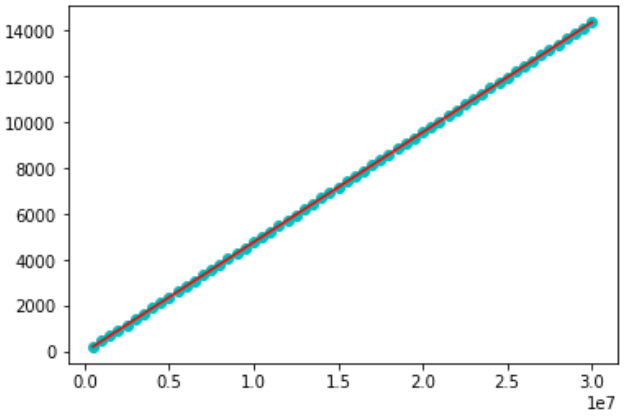
Quadro 04 - Duração acumulada das operações (DAO) e a duração média das operações unitárias (DOU) por bloco de insert, select e update e função do número de operações (NOP) (Amostragem).

NOP (x500000)	INSERT		SELECT		UPDATE	
	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)
1	0,0004865	243,26421	0,0004756	237,81649	0,0004950	247,49983
2	0,0004909	490,86802	0,0004758	475,75889	0,0004952	495,15449
3	0,0004975	746,31350	0,0004742	711,33666	0,0004949	742,30961
4	0,0004964	992,85524	0,0004738	947,60500	0,0004932	986,32715
5	0,0004958	1239,38030	0,0004746	1186,60201	0,0004932	1232,92852
…	…	…	…	…	…	…
56	0,0004913	13756,16676	0.0004778	13405,62959	0,0004913	13755,41500
57	0,0004912	14000,54958	0.0004778	13645,22887	0,0004912	13999,37429
58	0,0004913	14247,80271	0.0004778	13884,32797	0,0004912	14246,03070
59	0,0004912	14491,26072	0.0004778	14123,86920	0,0004912	14489,64630
60	0,0004913	14737,61566	0.0004778	14363,21392	0,0004912	14735,99863
MÉDIA	0,0004919	-	0.0004778	-	0,0004919	-

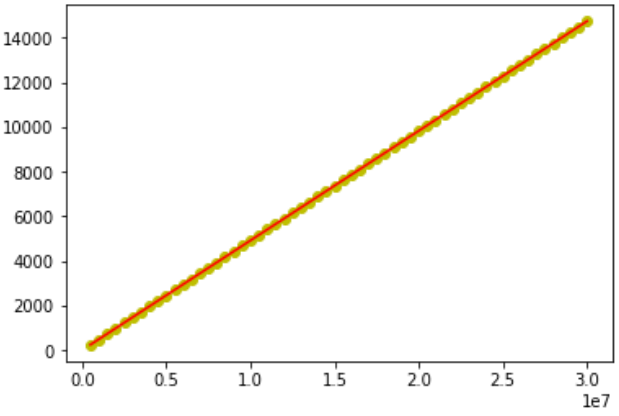
Figura 02 - Resultados.



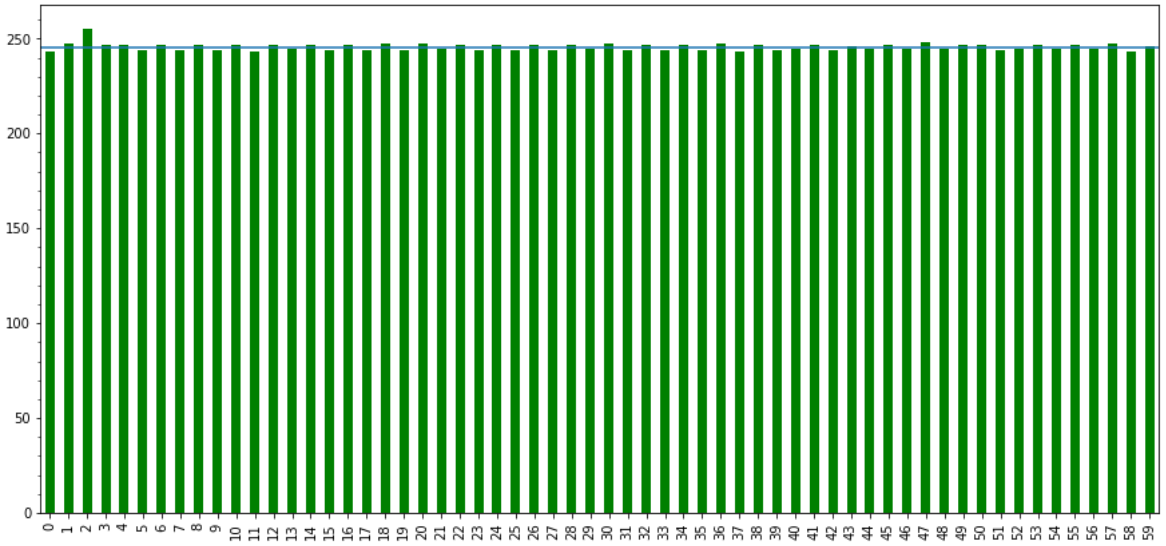
(a)



(b)

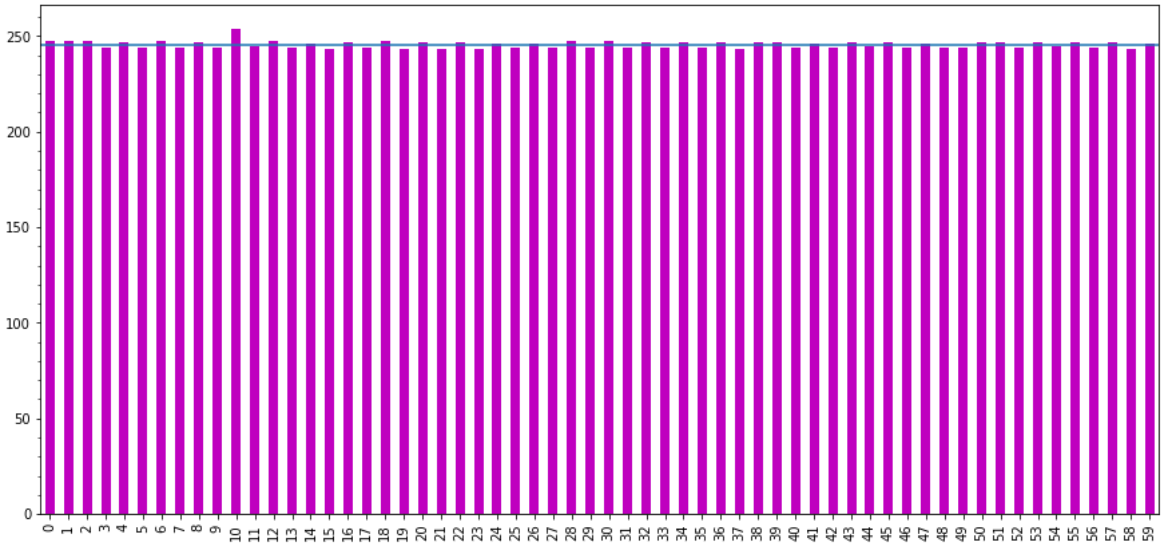


(c)



(d)

(e)



(f)

É possível ver, pelos gráficos, como o tempo se manteve consistente durante a execução dos blocos, com leves alterações. Dentre as três operações, o *select* apresentou o menor tempo, enquanto *insert* e *update* se mostraram similares. Isto era esperado, pois o Redis apenas sobrescreve o valor se for solicitada a escrita de um dado cuja chave já existe, ao contrário de outros SGBD que precisam buscar e ler a base de dados antes de atualizá-la. Sendo assim, as duas operações são na verdade uma só.

O *select* ser mais rápido que as outras operações pode ser explicado pela forma como o dado é armazenado. O Redis funciona de forma similar a um cache, com a diferença que os dados são persistentes, então os dados estão sempre disponíveis e fáceis de recuperar.

Conforme pode-se observar nas Figuras 1.a, 1.b e 1.c, os valores de DAO variaram de modo linear para as três operações.

CONCLUSÃO:

Avaliou-se a performance do Redis ao executar 30 milhões de operações (*insert*, *select* e *update*), cujas durações médias foram: *insert* = 0,0004919 s; *select* = 0,0004778 s e *update* = 0,0004919 s. Operações de *update* e *insert* requerem o mesmo tempo.

As equações ajustadas para os valores de duração acumulada das operações (DAO) em função do número de operações (NOP) foram DAO = 0,00049093163512 × NOP + 9,12782245045582, com R^2 = 0,99999971402487 para as operações de *insert*; DAO = 0,00047913525719 × NOP - 10,95460768622710, com R^2 = 0,99999979479763, para as operações de *select*; e DAO = 0,00049096616731 × NOP + 8,31555634164579, com R^2 = 0,99999965861478, para as operações de *update*.

Pode-se observar que o banco de dados Redis é de fato um banco de dados que mantém sua eficiência com grandes quantidades de dados, como foi possível notar pela figura 2.d, 2.e e 2.f, que mantém o comportamento constante independente da quantidade de registros presentes na tabela.

BIBLIOGRAFIA:

[1] Redis in Action. Disponível em: https://redislabs.com/community/ebook/

[2] Redis-py. Disponível em: https://github.com/andymccurdy/redis-py

ANEXO 01:

NOP (x500000)	Insert		Select		Update	
	DOU (s)	DAO (s)	DOU (s)	DAO (s)	DOU (s)	DAO (s)
1	0,0004865	243,26421	0,0004756	237,81649	0,0004950	247,49983
2	0,0004909	490,86802	0,0004758	237,94241	0,0004952	495,15449
3	0,0004975	746,31350	0,0004742	235,57777	0,0004949	742,30961
4	0,0004964	992,85524	0,0004738	236,26834	0,0004932	986,32715
5	0,0004958	1239,38030	0,0004746	238,99700	0,0004932	1232,92852
6	0,0004944	1483,20795	0,0004750	238,43614	0,0004922	1476,63080
7	0,0004942	1729,83803	0,0004756	239,68842	0,0004926	1724,07546
8	0,0004934	1973,52550	0,0004761	239,61237	0,0004920	1967,97074
9	0,0004935	2220,57732	0,0004764	239,64479	0,0004922	2214,86518
10	0,0004929	2464,65893	0,0004768	240,05302	0,0004917	2458,72897
11	0,0004930	2711,32403	0,0004770	239,26710	0,0004932	2712,75654
12	0,0004924	2954,40744	0,0004773	240,29510	0,0004929	2957,12022
13	0,0004925	3201,22941	0,0004773	238,92276	0,0004931	3204,83147
14	0,0004923	3445,76791	0,0004774	239,39786	0,0004927	3448,95220
15	0,0004923	3692,61127	0,0004775	239,24633	0,0004927	3695,17305
16	0,0004921	3936,60805	0,0004776	239,68273	0,0004923	3938,51612
17	0,0004922	4183,40542	0,0004776	238,91536	0,0004924	4185,15547
18	0,0004919	4427,31349	0,0004778	240,70818	0,0004922	4429,47336
19	0,0004921	4674,65699	0,0004779	239,14397	0,0004923	4676,77792
20	0,0004918	4918,24049	0,0004780	240,03545	0,0004920	4920,14705
21	0,0004920	5165,51960	0,0004780	239,23965	0,0004921	5166,99351
22	0,0004918	5410,07220	0,0004780	239,36987	0,0004919	5410,57942
23	0,0004919	5656,89902	0,0004781	239,94552	0,0004920	5657,42832
24	0,0004917	5900,48583	0,0004781	239,11369	0,0004917	5900,93928

25	0,0004918	6147,19871	0,0004782	240,00707	0,0004918	6147,08886
26	0,0004916	6391,25468	0,0004782	238,99592	0,0004916	6391,07900
27	0,0004917	6637,76762	0,0004782	239,73779	0,0004916	6637,24961
28	0,0004916	6881,83121	0,0004782	239,09069	0,0004915	6881,29635
29	0,0004916	7128,82358	0,0004783	240,09390	0,0004916	7128,43049
30	0,0004915	7373,12499	0,0004783	239,48138	0,0004915	7372,50046
31	0,0004916	7620,42632	0,0004783	239,36210	0,0004916	7619,82636
32	0,0004915	7864,44435	0,0004784	239,76321	0,0004915	7863,86970
33	0,0004916	8110,89672	0,0004784	240,09405	0,0004915	8110,52795
34	0,0004915	8354,70169	0,0004784	239,18821	0,0004914	8354,22734
35	0,0004915	8601,52902	0,0004785	240,10784	0,0004915	8601,01342
36	0,0004914	8845,11942	0,0004785	239,76668	0,0004914	8844,96595
37	0,0004915	9092,41829	0,0004786	240,35420	0,0004915	9092,04721
38	0,0004914	9335,80370	0,0004786	240,13119	0,0004913	9335,59395
39	0,0004914	9582,44851	0,0004786	239,63337	0,0004914	9582,71653
40	0,0004913	9826,57213	0,0004786	239,44031	0,0004915	9829,62826
41	0,0004913	10071,10537	0,0004786	238,69150	0,0004914	10073,27064
42	0,0004913	10318,00211	0,0004786	239,87906	0,0004914	10319,63602
43	0,0004913	10562,11259	0,0004787	240,14314	0,0004913	10563,42211
44	0,0004913	10808,18613	0,0004787	239,18437	0,0004914	10809,87162
45	0,0004912	11052,51030	0,0004787	240,11410	0,0004913	11054,41008
46	0,0004913	11299,25767	0,0004787	240,24918	0,0004914	11301,38786
47	0,0004912	11543,66085	0,0004787	239,06031	0,0004913	11545,29292
48	0,0004913	11791,54405	0,0004787	239,29398	0,0004913	11791,45405
49	0,0004913	12035,95788	0,0004787	239,02321	0,0004912	12035,28620
50	0,0004913	12283,01547	0,0004787	239,56265	0,0004912	12279,40505
51	0,0004914	12529,55002	0,0004787	239,25597	0,0004912	12526,47238
52	0,0004913	12773,35699	0,0004787	240,11876	0,0004913	12773,19713

53	0,0004912	13017,67389	0,0004787	239,47018	0,0004912	13017,13721
54	0,0004913	13264,29765	0,0004788	240,11794	0,0004913	13264,09602
55	0,0004912	13509,09798	0,0004788	238,90764	0,0004912	13508,77609
56	0,0004913	13756,16676	0,0004788	239,98938	0,0004913	13755,41500
57	0,0004912	14000,54958	0,0004788	239,59928	0,0004912	13999,37429
58	0,0004913	14247,80271	0,0004788	239,09909	0,0004912	14246,03070
59	0,0004912	14491,26072	0,0004788	239,54124	0,0004912	14489,64630
60	0,0004913	14737,61566	0,0004788	239,34471	0,0004912	14735,99863
MÉDIA	0,0004919	-	0,0004778	-	0,0004919	-

ANEXO 02:

Resumo do código utilizado, o código completo é possível encontrar no endereço <https://github.com/ellfiscina/RedisAnalysis>, onde também estão disponíveis este artigo e a tabela de resultados sem tratamento.

```
import redis
import time

## Conexão com o Redis
r = redis.Redis(host='localhost', port=6379, db=0)

## Variáveis
insert_time = []
sel_time = []
up_time = []

NBLOCK = 60
BLOCK = 500000

## Operações
### Insert
def insert():
    initial_time = time.time()
    for i in range(BLOCK):
        r.hset('table' + str(i), 'A0', i)
        r.hset('table' + str(i), 'A1', '1')
        r.hset('table' + str(i), 'A2', '1.2')
        r.hset('table' + str(i), 'A3', 'aaaaa')
        r.hset('table' + str(i), 'A4', 'aaaaa')
        r.hset('table' + str(i), 'A5', '11:11:11')
        r.hset('table' + str(i), 'A6', '2007-09-23')

    final_time = time.time()
    return final_time - initial_time

### Select
def select():
    initial_time = time.time()
    for i in range(BLOCK):
        r.hget('table' + str(i), 'A0')
        r.hget('table' + str(i), 'A1')
        r.hget('table' + str(i), 'A2')
        r.hget('table' + str(i), 'A3')
        r.hget('table' + str(i), 'A4')
```

```
        r.hget('table' + str(i), 'A5')
        r.hget('table' + str(i), 'A6')

    final_time = time.time()
    return final_time - initial_time

### Update
def update():
    initial_time = time.time()
    for i in range(BLOCK):
        r.hset('table' + str(i), 'A0', i)
        r.hset('table' + str(i), 'A1', '2')
        r.hset('table' + str(i), 'A2', '2.2')
        r.hset('table' + str(i), 'A3', 'aaaaaa')
        r.hset('table' + str(i), 'A4', 'bbbbb')
        r.hset('table' + str(i), 'A5', '22:22:22')
        r.hset('table' + str(i), 'A6', '2017-09-23')

    final_time = time.time()
    return final_time - initial_time

## Salva o tempo de execução de cada bloco
for i in range(NBLOCK):
    insert_time.append(insert())
    sel_time.append(select())
    up_time.append(update())
```