

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 $DAO = 0,00049093163512 \times NOP + 9,12782245045582$, com $R^2 = 0,99999971402487$ para as operações de *insert*; $DAO = 0,00047913525719 \times NOP - 10,95460768622710$, com $R^2 = 0,99999979479763$, para as operações de *select*; e $DAO = 0,00049096616731 \times NOP + 8,31555634164579$, 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 $DAO = 0,00049093163512 \times NOP + 9,12782245045582$, with $R^2 = 0,99999971402487$ to insertion; $DAO = 0,00047913525719 \times NOP - 10,95460768622710$, with $R^2 = 0,99999979479763$, to selection; and $DAO = 0,00049096616731 \times NOP + 8,31555634164579$, 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.

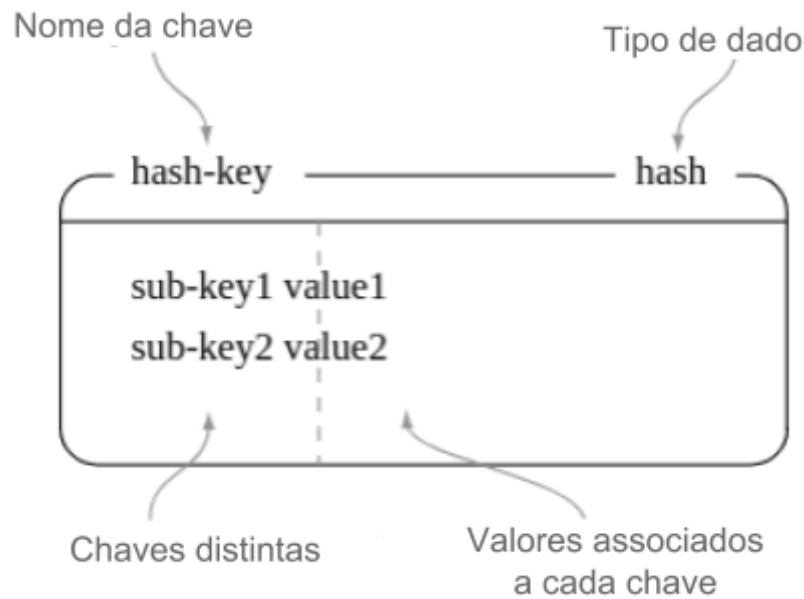
Algumas aplicações conhecidas do Redis incluem Twitter, Github, Hulu, Airbnb e Pinterest, pela sua capacidade de processar grandes quantidades de dados em tempo real.

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 das 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

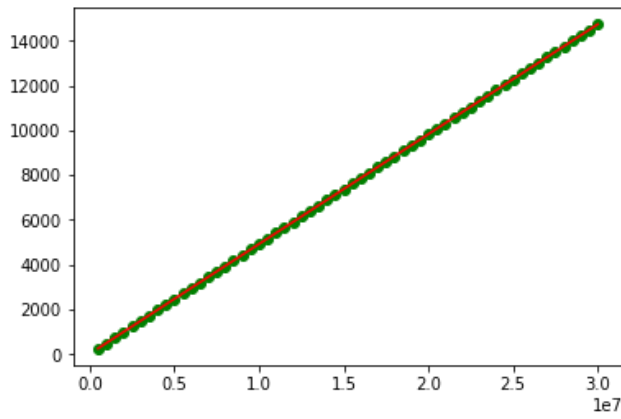
DOU em função do número de operações *insert*, *select* e *update*, respectivamente e uma reta marcando a duração média de cada bloco.

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*, $DAO = 0,00049093163512 \times NOP + 9,12782245045582$, com $R^2 = 0,99999971402487$; para o *select*, $DAO = 0,00047913525719 \times NOP - 10,95460768622710$, com $R^2 = 0,99999979479763$; e $DAO = 0,00049096616731 \times NOP + 8,31555634164579$, com $R^2 = 0,99999965861478$, para as operações de *update*.

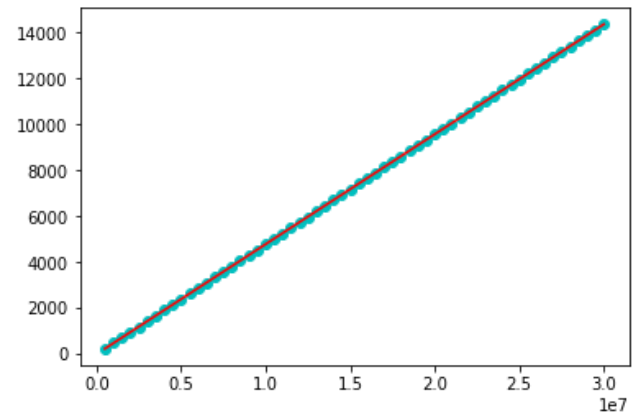
Quadro 04 - Duração acumulada das operações (DAO) e a duração média das operações unitárias (DOU) por bloco de insert, select e update e função do número de operações (NOP) (Amostragem).

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	-

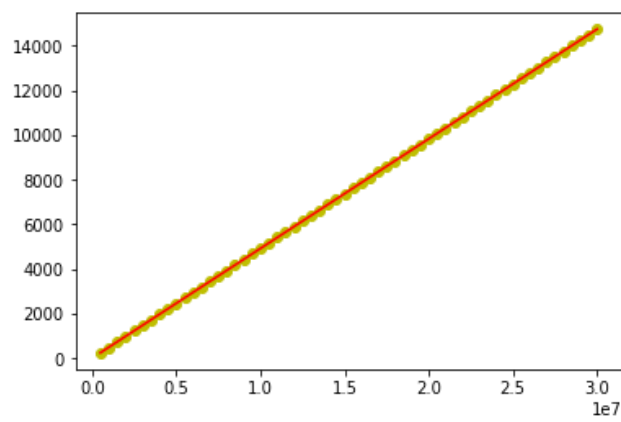
Figura 02 - Resultados.



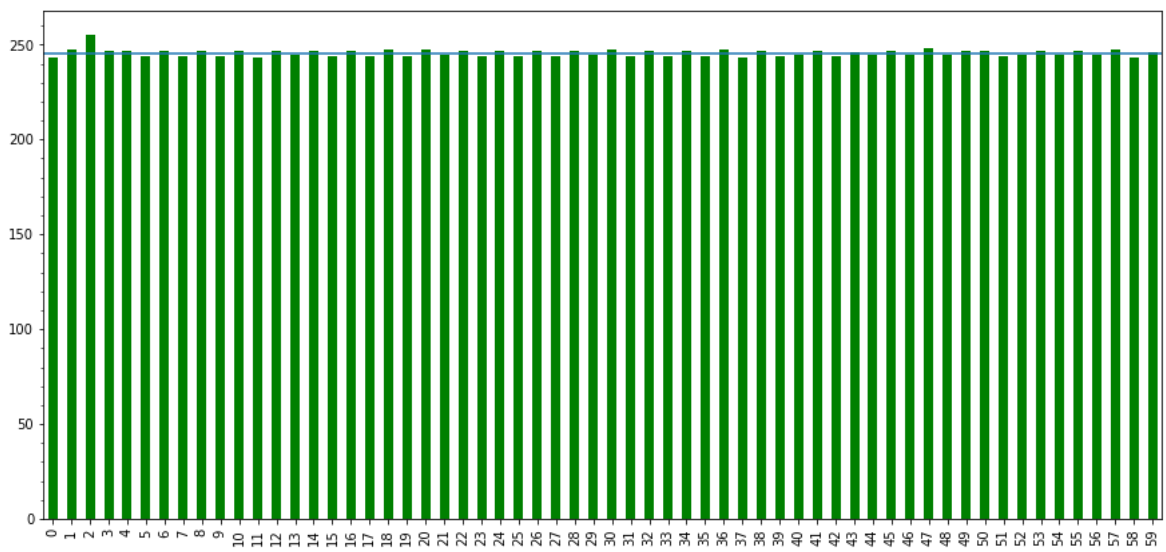
(a)



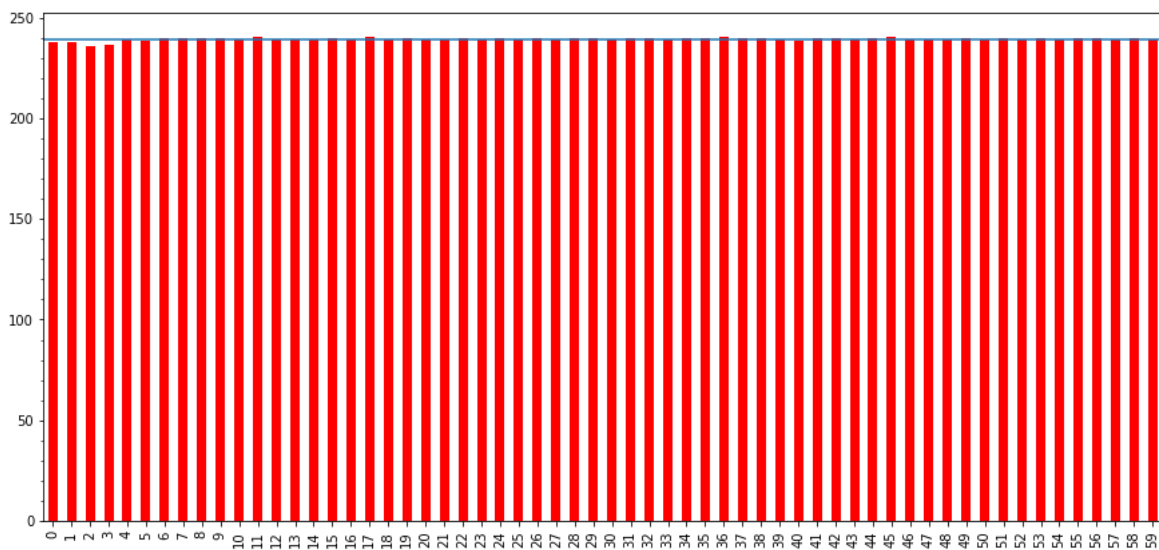
(b)



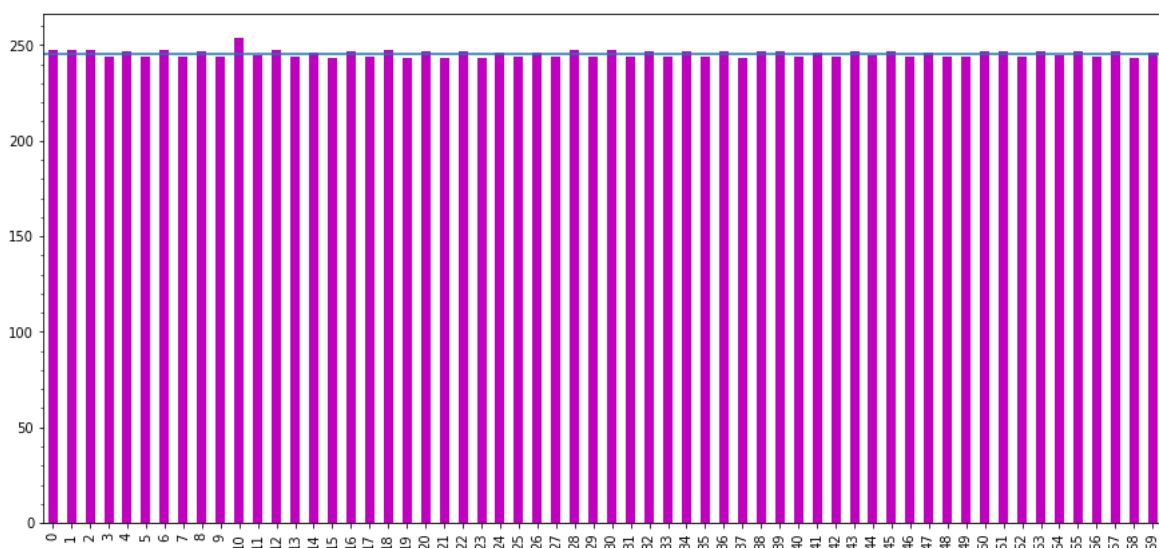
(c)



(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 $DAO = 0,00049093163512 \times NOP + 9,12782245045582$, com $R^2 = 0,99999971402487$ para as operações de *insert*; $DAO = 0,00047913525719 \times NOP - 10,95460768622710$, com $R^2 = 0,99999979479763$, para as operações de *select*; e $DAO = 0,00049096616731 \times NOP + 8,31555634164579$, 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>