Redis-benchmark

Redis-benchmark

Redis-benchmark 사용법을 설명합니다.
이 문서는 버전 4.0.9를 기준으로 작성했습니다.

명령 목록

- **기본 Benchmark** 기본 Benchmark를 한다.
- **클러스터 지원** 클러스터를 Benchmark를 한다.
- **단순하게 표시** -q 옵션을 사용한다.
- **CSV 형식으로 표시** --csv
- **명령 지정** -t 옵션으로 benchmark를 실행할 명령을 지정한다.
- **실행 횟수 지정** -n 옵션으로 실행 횟수를 지정한다. 기본은 100,000이다.
- **Client 수 지정** -c 옵션으로 클라이언트 수를 지정한다. 기본은 50이다.
- **Sleep 지정** -sleep 옵션으로 sleep를 지정한다. 단위는 us(microsecond)이다. 기본은 0이다.
- **Thread 수 지정** --threads 옵션으로 쓰레드 수를 지정한다.
- **Value 사이즈 지정** -d 옵션으로 value 크기를 지정할 수 있다. 기본은 3바이트이다.
- **Key, Value 다양하게 지정** -r 옵션으로 key 또는 value를 다양하게 테스트할 수 있다.
- **Key, Value 직접 지정** -r 옵션과 같이 사용한다.
- **Client tracking on 지정** Client side caching을 참조하세요.
- **측정 정밀도 지정** -precision 0|1|2
- **HELP** 도움말을 보여준다.

기본 Benchmark

- 주요 명령을 10만회씩 실행해서 성능을 측정한다. 주요 명령은 PINK, SET, GET, INCR, LPUSH, RPUSH, LPOP, RPOP, SADD, HSET, SPOP, LRANGE, MSET 이다.

- 테스트 서버에서 주요 입력 명령의 초당 처리량(실행 횟수)은 약 20만회이다.

```
$ src/redis-benchmark -p 6001
====== PING_INLINE ======
  100000 requests completed in 0.95 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1
99.92% <= 1 milliseconds
100.00% <= 1 milliseconds
104821.80 requests per second
====== PING_BULK ======
  100000 requests completed in 0.55 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
181488.20 requests per second
====== SET ======
  100000 requests completed in 0.50 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1
100.00% <= 0 milliseconds
199203.20 requests per second
====== GET ======
  100000 requests completed in 0.51 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1
100.00% <= 0 milliseconds
197628.47 requests per second
====== INCR ======
  100000 requests completed in 0.50 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1
100.00% <= 0 milliseconds
198807.16 requests per second
```

- **버전 6.0부터 save, appendonly, threads 가 추가로 표시된다.** 이 항목(옵션)에 따라 성능이 달라질 수 있다.

```
$ src/redis-benchmark -p 6001 -t set -d 100 --threads 8
====== SET ======
  100000 requests completed in 3.52 seconds
  50 parallel clients
  100 bytes payload
  keep alive: 1
  host configuration "save":
  host configuration "appendonly": yes
  multi-thread: yes
  threads: 8
```

클러스터 지원: --cluster

- 버전 6.0부터 클러스터에 benchmark가 가능하다.

```
$ src/redis-benchmark -p 7000 -t set -d 100 --cluster
Cluster has 3 master nodes:

Master 0: 3dc49ccc0dddeb8a621b28744ddee82cf14af040 127.0.0.1:7001
Master 1: 6960872da1aa36e8c4670adbafa3c47f4001efbc 127.0.0.1:7002
Master 2: 745e6d7d18e2501b196d401aa8160bb37b15f754 127.0.0.1:7000

====== SET ======
 100000 requests completed in 3.26 seconds
 50 parallel clients
 100 bytes payload
 keep alive: 1
 cluster mode: yes (3 masters)
 node [0] configuration:
   save: NONE
   appendonly: no
 node [1] configuration:
   save: NONE
   appendonly: no
 node [2] configuration:
   save: NONE
   appendonly: no
 multi-thread: yes
 threads: 3
```

- 버전 6.0 미만에서 클러스터로 구성된 하나의 노드에 benchmark를 실행하면 benchmark는 실행되는 것으로 표시되지만 **cluster redis-server에서는 처리되지 않는다.** 즉, SET, GET 같은 명령이 실행되지 않는다. 그러므로 benchmark 실행 후 해당 노드에 접속해서 데이터를 확인하면 하나도 없고 당연히 aof 파일에도 저장되지 않는다. 다만 ping 명령은 실행된다. Ping 명령으로도 성능을 가늠해볼 수는 있다.
- 버전 6.0 미만에서는 단독 마스터를 실행해서 benchmark를 테스트한다.

단순하게 표시: -q

```
$ src/redis-benchmark -p 6001 -q
PING_INLINE: 166666.66 requests per second
PING_BULK: 199203.20 requests per second
SET: 201612.91 requests per second
GET: 199203.20 requests per second
INCR: 201612.91 requests per second
LPUSH: 202429.16 requests per second
RPUSH: 202429.16 requests per second
LPOP: 200400.80 requests per second
RPOP: 201612.91 requests per second
SADD: 200803.22 requests per second
HSET: 202429.16 requests per second
SPOP: 200803.22 requests per second
LPUSH (needed to benchmark LRANGE): 202839.75 requests per second
LRANGE_100 (first 100 elements): 86430.43 requests per second
LRANGE_300 (first 300 elements): 34328.87 requests per second
LRANGE_500 (first 450 elements): 24752.47 requests per second
LRANGE_600 (first 600 elements): 18986.14 requests per second
MSET (10 keys): 139860.14 requests per second
```

CSV 형식으로 표시: --csv

```
$ src/redis-benchmark -p 6001 --csv
"PING_INLINE","169204.73"
"PING_BULK","196463.66"
"SET","199600.80"
"GET","197628.47"
"INCR","198807.16"
"LPUSH","199600.80"
"RPUSH","198807.16"
"LPOP","198807.16"
"RPOP","198019.80"
"SADD","198807.16"
"HSET","200400.80"
"SPOP","200400.80"
"LPUSH (needed to benchmark LRANGE)","200000.00"
"LRANGE_100 (first 100 elements)","85984.52"
"LRANGE_300 (first 300 elements)","34211.43"
"LRANGE_500 (first 450 elements)","24491.79"
"LRANGE_600 (first 600 elements)","18957.35"
"MSET (10 keys)","140646.97"
```

명령 지정: -t

- -t 옵션으로 benchmark를 실행할 명령을 지정한다.

```
$ src/redis-benchmark -p 6001 -t set
====== SET ======
  100000 requests completed in 0.58 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
171232.88 requests per second
```

실행 횟수 지정: -n

- -n 옵션으로 실행 횟수를 지정한다. Default는 100,000이다.

```
$ src/redis-benchmark -p 6001 -t set -n 500000
====== SET ======
  500000 requests completed in 2.61 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
191570.89 requests per second
```

Client 수 지정: -c

- -c 옵션으로 Clients 수를 지정한다. Default는 50이다.

```
$ src/redis-benchmark -p 6001 -t set -n 500000 -c 10
====== SET ======
  500000 requests completed in 2.56 seconds
  10 parallel clients
  3 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
195694.72 requests per second
```

Sleep 지정: -usleep

- -usleep로 sleep를 지정한다. 단위는 마이크로초(microsecond)이다. 1000000 -> 1초, 1000 -> 1ms 입니다. 위에서 설명한 client 옵션과 같이 사용해서 실행 명령 수를 조정할 수 있습니다. Default는 0이다. 이 옵션은 Enterprise 7.2.0부터 사용할 수 있습니다.

```
$ src/redis-benchmark -p 6001 -t set -n 500000 -usleep 100
```

Thread 수 지정: --threads

- --threads <num> 옵션으로 쓰레드 수를 지정한다. Default는 No이다.

```
$ src/redis-benchmark -p 6001 -t set -n 500000 --threads 8
====== SET ======
  500000 requests completed in 17.04 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1
  host configuration "save":
  host configuration "appendonly": yes
  multi-thread: yes
  threads: 8
```

Value 사이즈 지정: -d

- -d 옵션으로 Value 사이즈를 지정한다. Default는 3이다. Payload에 지정한 사이즈(바이트)가 표시된다.
- SET 명령의 경우 키 이름이 "key:__rand_int__"이다.

```
$ src/redis-benchmark -p 6001 -t set -d 20
====== SET ======
  100000 requests completed in 0.59 seconds
  50 parallel clients
  20 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
168918.92 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> get key:__rand_int__
"xxxxxxxxxxxxxxxxxxxx"

$ src/redis-benchmark -p 6001 -t lpush -d 20
====== LPUSH ======
  100000 requests completed in 0.59 seconds
  50 parallel clients
  20 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
170357.75 requests per second

127.0.0.1:6001> lrange mylist 0 5
 1) "xxxxxxxxxxxxxxxxxxxx"
 2) "xxxxxxxxxxxxxxxxxxxx"
 3) "xxxxxxxxxxxxxxxxxxxx"
 4) "xxxxxxxxxxxxxxxxxxxx"
 5) "xxxxxxxxxxxxxxxxxxxx"
 6) "xxxxxxxxxxxxxxxxxxxx"
```

Key, Value 다양하게 지정: -r

- -r 옵션으로 key 또는 value를 다양하게 테스트할 수 있다.

- SET test

```
$ src/redis-benchmark -p 6001 -t set -r 100000
====== SET ======
  100000 requests completed in 0.95 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.87% <= 1 milliseconds
100.00% <= 1 milliseconds
104931.80 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> dbsize
(integer) 63251
127.0.0.1:6001> scan 0
1) "53248"
2)  1) "key:000000080588"
    2) "key:000000027240"
    3) "key:000000009027"
    4) "key:000000028279"
    5) "key:000000057324"
    6) "key:000000044128"
    7) "key:000000035869"
    8) "key:000000027983"
    9) "key:000000003973"
   10) "key:000000002511"
```

- SADD test

```
$ src/redis-benchmark -p 6001 -t sadd -n 100 -r 100000
====== SADD ======
  100 requests completed in 0.00 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.00% <= 1 milliseconds
100.00% <= 1 milliseconds
50000.00 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> scard myset
(integer) 100
127.0.0.1:6001> sscan myset 0
1) "24"
2)  1) "element:000000037570"
    2) "element:000000004318"
    3) "element:000000030866"
    4) "element:000000052834"
    5) "element:000000065944"
    6) "element:000000012470"
    7) "element:000000018875"
    8) "element:000000061924"
    9) "element:000000007416"
   10) "element:000000084806"
```

Key, Value 직접 지정

- **LPUSH**: redis-benchmark -p 6001 -r 1000000 **lpush mylist ele:__rand_int__**

```
$ src/redis-benchmark -p 6001 -r 1000000 lpush mylist ele:__rand_int__
====== lpush mylist ele:__rand_int__ ======
  100000 requests completed in 0.94 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.88% <= 1 milliseconds
100.00% <= 1 milliseconds
105820.11 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> llen mylist
(integer) 100000
127.0.0.1:6001> lrange mylist 0 6
 1) "ele:000000423107"
 2) "ele:000000346547"
 3) "ele:000000216990"
 4) "ele:000000280357"
 5) "ele:000000762985"
 6) "ele:000000956186"
```

- **SADD**: redis-benchmark -p 6001 -r 1000000 **sadd myset mem:__rand_int__**
- redis-benchmark -p 6001 -n 1000000 -r 100000000 **set keyA___rand_int__ valueA___rand_int__**
- redis-benchmark -p 6001 -n 1000000 -r 100000000 **get keyA___rand_int__**
- redis-benchmark -p 6001 -n 1000000 -r 100000000 **del keyA___rand_int__**
- redis-benchmark -p 6001 -n 1000000 -r 100000000 **zadd myzset 0 memA___rand_int__**
- redis-benchmark -p 6001 -n 1000000 **get keyA_000000000345**
- redis-benchmark -p 6001 -n 1000000 **object time keyA_000000000345**
- redis-benchmark -p 6001 -n 1000000 **rpop mylist5**

- -r 숫자를 크게하면 member 중복을 적게 할 수 있다.

```
$ src/redis-benchmark -p 6001 -r 1000000 sadd myset mem:__rand_int__
====== sadd myset mem:__rand_int__ ======
  100000 requests completed in 0.60 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.90% <= 1 milliseconds
100.00% <= 1 milliseconds
166944.92 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> scard myset
(integer) 95177
127.0.0.1:6001> sscan myset 0
1) "12288"
2)  1) "mem:000000170342"
    2) "mem:000000803395"
    3) "mem:000000301154"
    4) "mem:000000949710"
    5) "mem:000000489267"
    6) "mem:000000542854"
    7) "mem:000000767008"
    8) "mem:000000451243"
    9) "mem:000000638887"
   10) "mem:000000724675"
   11) "mem:000000970089"
   12) "mem:000000575203"
```

- **ZADD**: redis-benchmark -p 6001 -r 1000000 **zadd myzset __rand_int__ member:__rand_int__**

```
$ src/redis-benchmark -p 6001 -r 1000000 zadd myzset __rand_int__ member:__rand_int__
====== zadd myzset __rand_int__ member:__rand_int__ ======
  100000 requests completed in 0.64 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.94% <= 1 milliseconds
100.00% <= 1 milliseconds
156739.81 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> zcard myzset
(integer) 95122
127.0.0.1:6001> zscan myzset 0
1) "90112"
2)  1) "member:000000433048"
    2) "105823"
    3) "member:000000650524"
    4) "345104"
    5) "member:000000034870"
    6) "779946"
    7) "member:000000622620"
    8) "130322"
```

- **HSET**: redis-benchmark -p 6001 -r 1000000 **hset myhash field:__rand_int__ value:__rand_int__**

```
$ src/redis-benchmark -p 6001 -r 1000000 hset myhash field:__rand_int__ value:__rand_int__
====== hset myhash field:__rand_int__ value:__rand_int__ ======
  100000 requests completed in 0.94 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.36% <= 1 milliseconds
99.96% <= 2 milliseconds
100.00% <= 2 milliseconds
106044.54 requests per second

$ src/redis-cli -p 6001
127.0.0.1:6001> hscan myhash 0
1) "40960"
2)  1) "field:000000878442"
    2) "value:000000620020"
    3) "field:000000168038"
    4) "value:000000141778"
    5) "field:000000815313"
    6) "value:000000852776"
```

Client tracking on: --enable-tracking

- ```
  $ src/redis-benchmark -p 6001 -t set -d 100 --enable-tracking
  ```

측정(표시) 정밀도: --precision

- --precision 0: millisecond 단위로 표시한다.

  ```
  $ src/redis-benchmark -p 6001 -t set -d 100 --precision 0
  27.88% <= 1 milliseconds
  98.78% <= 2 milliseconds
  99.76% <= 3 milliseconds
  ```

- --precision 1: 0.1 millisecond 단위로 표시한다.

  ```
  $ src/redis-benchmark -p 6001 -t set -d 100 --precision 1
  0.12% <= 0.6 milliseconds
  0.32% <= 0.7 milliseconds
  5.91% <= 0.8 milliseconds
  ```

- --precision 2: 0.01 millisecond 단위로 표시한다.

  ```
  $ src/redis-benchmark -p 6001 -t set -d 100 --precision 2
  0.03% <= 0.68 milliseconds
  0.12% <= 0.69 milliseconds
  0.28% <= 0.70 milliseconds
  ```

HELP
  ```

```
$ src/redis-benchmark --help
Usage: redis-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests>] [-k <boolean>]

 -h <hostname>      Server hostname (default 127.0.0.1)
 -p <port>          Server port (default 6379)
 -s <socket>        Server socket (overrides host and port)
 -a <password>      Password for Redis Auth
 -c <clients>       Number of parallel connections (default 50)
 -n <requests>      Total number of requests (default 100000)
 -d <size>          Data size of SET/GET value in bytes (default 3)
 --dbnum <db>       SELECT the specified db number (default 0)
 --threads <num>    Enable multi-thread mode. ver 6.0
 --cluster          Enable cluster mode. ver 6.0
 --enable-tracking  Send CLIENT TRACKING on before starting benchmark. ver 6.0
 -k <boolean>       1=keep alive 0=reconnect (default 1)
 -r <keyspacelen>   Use random keys for SET/GET/INCR, random values for SADD
  Using this option the benchmark will expand the string __rand_int__
  inside an argument with a 12 digits number in the specified range
  from 0 to keyspacelen-1. The substitution changes every time a command
  is executed. Default tests use this to hit random keys in the
  specified range.
 -P <numreq>        Pipeline <numreq> requests. Default 1 (no pipeline).
 -e                  If server replies with errors, show them on stdout.
                    (no more than 1 error per second is displayed)
 -q                  Quiet. Just show query/sec values
 --precision        Number of decimal places to display in latency output (default 0) ver 6.0
 --csv              Output in CSV format
 -l                 Loop. Run the tests forever
 -t <tests>         Only run the comma separated list of tests. The test
                    names are the same as the ones produced as output.
 -I                 Idle mode. Just open N idle connections and wait.

Examples:

 Run the benchmark with the default configuration against 127.0.0.1:6379:
   $ redis-benchmark

 Use 20 parallel clients, for a total of 100k requests, against 192.168.1.1:
   $ redis-benchmark -h 192.168.1.1 -p 6379 -n 100000 -c 20

 Fill 127.0.0.1:6379 with about 1 million keys only using the SET test:
   $ redis-benchmark -t set -n 1000000 -r 100000000

 Benchmark 127.0.0.1:6379 for a few commands producing CSV output:
   $ redis-benchmark -t ping,set,get -n 100000 --csv

 Benchmark a specific command line:
   $ redis-benchmark -r 10000 -n 10000 eval 'return redis.call("ping")' 0

 Fill a list with 10000 random elements:
   $ redis-benchmark -r 10000 -n 10000 lpush mylist __rand_int__

 On user specified command lines __rand_int__ is replaced with a random integer
 with a range of values selected by the -r option.
```


<< Redis-cli Redis-benchmark Redis Security >>

redisgate@gmail.com 02.503.2235
서울시 강남구 강남대로 342 역삼빌딩 5층 (역삼동) 우 06242