



Redis Mass Insertion

Sometimes Redis instances need to be loaded with a big amount of preexisting or user generated data in a short amount of time, so that millions of keys will be created as fast as possible.

This is called a *mass insertion*, and the goal of this document is to provide information about how to feed Redis with data as fast as possible.

Use the protocol, Luke

Using a normal Redis client to perform mass insertion is not a good idea for a few reasons: the naive approach of sending one command after the other is slow because you have to pay for the round trip time for every command. It is possible to use pipelining, but for mass insertion of many records you need to write new commands while you read replies at the same time to make sure you are inserting as fast as possible.

Only a small percentage of clients support non-blocking I/O, and not all the clients are able to parse the replies in an efficient way in order to maximize throughput. For all of these reasons the preferred way to mass import data into Redis is to generate a text file containing the Redis protocol, in raw format, in order to call the commands needed to insert the required data.

For instance if I need to generate a large data set where there are billions of keys in the form: ``keyN -> ValueN'` I will create a file containing the following commands in the Redis protocol format:

```
SET Key0 Value0
SET Key1 Value1
...
SET KeyN ValueN
```

Once this file is created, the remaining action is to feed it to Redis as fast as possible. In the past the way to do this was to use the `netcat` with the following command:

```
(cat data.txt; sleep 10) | nc localhost 6379 > /dev/null
```

However this is not a very reliable way to perform mass import because netcat does not really know when all the data was transferred and can't check for errors. In 2.6 or later

versions of Redis the `redis-cli` utility supports a new mode called **pipe mode** that was designed in order to perform mass insertion.

Using the pipe mode the command to run looks like the following:

```
cat data.txt | redis-cli --pipe
```

That will produce an output similar to this:

```
All data transferred. Waiting for the last reply...  
Last reply received from server.  
errors: 0, replies: 1000000
```

The `redis-cli` utility will also make sure to only redirect errors received from the Redis instance to the standard output.

Generating Redis Protocol

The Redis protocol is extremely simple to generate and parse, and is [Documented here](#). However in order to generate protocol for the goal of mass insertion you don't need to understand every detail of the protocol, but just that every command is represented in the following way:

```
*<args><cr><lf>  
$<len><cr><lf>  
<arg0><cr><lf>  
<arg1><cr><lf>  
...  
<argN><cr><lf>
```

Where `<cr>` means `"\r"` (or ASCII character 13) and `<lf>` means `"\n"` (or ASCII character 10).

For instance the command **SET key value** is represented by the following protocol:

```
*3<cr><lf>
$3<cr><lf>
SET<cr><lf>
$3<cr><lf>
key<cr><lf>
$5<cr><lf>
value<cr><lf>
```

Or represented as a quoted string:

```
"*3\r\n$3\r\nSET\r\n$3\r\nkey\r\n$5\r\nvalue\r\n"
```

The file you need to generate for mass insertion is just composed of commands represented in the above way, one after the other.

The following Ruby function generates valid protocol:

```
def gen_redis_proto(*cmd)
  proto = ""
  proto << "*" + cmd.length.to_s + "\r\n"
  cmd.each{|arg|
    proto << "$" + arg.to_s.bytesize.to_s + "\r\n"
    proto << arg.to_s + "\r\n"
  }
  proto
end

puts gen_redis_proto("SET","mykey","Hello World!").inspect
```

Using the above function it is possible to easily generate the key value pairs in the above example, with this program:

```
(0...1000).each{|n|
  STDOUT.write(gen_redis_proto("SET","Key#{n}","Value#{n}"))
}
```

We can run the program directly in pipe to redis-cli in order to perform our first mass import session.

```
$ ruby proto.rb | redis-cli --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 1000
```

How the pipe mode works under the hood

The magic needed inside the pipe mode of redis-cli is to be as fast as netcat and still be able to understand when the last reply was sent by the server at the same time.


This is obtained in the following way:

- redis-cli --pipe tries to send data as fast as possible to the server.
- At the same time it reads data when available, trying to parse it.
- Once there is no more data to read from stdin, it sends a special **ECHO** command with a random 20 byte string: we are sure this is the latest command sent, and we are sure we can match the reply checking if we receive the same 20 bytes as a bulk reply.
- Once this special final command is sent, the code receiving replies starts to match replies with these 20 bytes. When the matching reply is reached it can exit with success.

Using this trick we don't need to parse the protocol we send to the server in order to understand how many commands we are sending, but just the replies.

However while parsing the replies we take a counter of all the replies parsed so that at the end we are able to tell the user the amount of commands transferred to the server by the mass insert session.

This website is open source software. See all credits.

Sponsored by  **redislabs**
HOME OF REDIS