Redis Clients Handling

This document provides information about how Redis handles clients from the point of view of the network layer: connections, timeouts, buffers, and other similar topics are covered here.

The information contained in this document is **only applicable to Redis version 2.6 or greater**.

How client connections are accepted

Redis accepts clients connections on the configured listening TCP port and on the Unix socket if enabled. When a new client connection is accepted the following operations are performed:

- The client socket is put in non-blocking state since Redis uses multiplexing and non-blocking I/O.
- The `TCP_NODELAY` option is set in order to ensure that we don't have delays in our connection.
- A *readable* file event is created so that Redis is able to collect the client queries as soon as new data is available to be read on the socket.

After the client is initialized, Redis checks if we are already at the limit of the number of clients that it is possible to handle simultaneously (this is configured using the `maxclients` configuration directive, see the next section of this document for further information).

In case it can't accept the current client because the maximum number of clients was already accepted, Redis tries to send an error to the client in order to make it aware of this condition, and closes the connection immediately. The error message will be able to reach the client even if the connection is closed immediately by Redis because the new socket output buffer is usually big enough to contain the error, so the kernel will handle the transmission of the error.

In what order clients are served

The order is determined by a combination of the client socket file descriptor number and order in which the kernel reports events, so the order is to be considered as unspecified.

However Redis does the following two things when serving clients:

- It only performs a single `read()` system call every time there is something new to read from the client socket, in order to ensure that if we have multiple clients connected, and a few are very demanding clients sending queries at an high rate, other clients are not penalized and will not experience a bad latency figure.

- However once new data is read from a client, all the queries contained in the current buffers are processed sequentially. This improves locality and does not need iterating a second time to see if there are clients that need some processing time.

Maximum number of clients

In Redis 2.4 there was a hard-coded limit for the maximum number of clients that could be handled simultaneously.

In Redis 2.6 this limit is dynamic: by default it is set to 10000 clients, unless otherwise stated by the `maxclients` directive in Redis.conf.

However, Redis checks with the kernel what is the maximum number of file descriptors that we are able to open (the *soft limit* is checked). If the limit is smaller than the maximum number of clients we want to handle, plus 32 (that is the number of file descriptors Redis reserves for internal uses), then the number of maximum clients is modified by Redis to match the amount of clients we are *really able to handle* under the current operating system limit.

When the configured number of maximum clients can not be honored, the condition is logged at startup as in the following example:

```
$ ./redis-server --maxclients 100000
[41422] 23 Jan 11:28:33.179 # Unable to set the max number of files l
```

When Redis is configured in order to handle a specific number of clients it is a good idea to make sure that the operating system limit to the maximum number of file descriptors per process is also set accordingly.

Under Linux these limits can be set both in the current session and as a system-wide setting with the following commands:

- ulimit -Sn 100000 # This will only work if hard limit is big enough.
- sysctl -w fs.file-max=100000

Output buffers limits

Redis needs to handle a variable-length output buffer for every client, since a command can produce a big amount of data that needs to be transferred to the client.

However it is possible that a client sends more commands producing more output to serve at a faster rate at which Redis can send the existing output to the client. This is especially true with Pub/Sub clients in case a client is not able to process new messages fast enough.

Both the conditions will cause the client output buffer to grow and consume more and more memory. For this reason by default Redis sets limits to the output buffer size for

different kind of clients. When the limit is reached the client connection is closed and the event logged in the Redis log file.

There are two kind of limits Redis uses:

- The **hard limit** is a fixed limit that when reached will make Redis closing the client connection as soon as possible.
- The **soft limit** instead is a limit that depends on the time, for instance a soft limit of 32 megabytes per 10 seconds means that if the client has an output buffer bigger than 32 megabytes for, continuously, 10 seconds, the connection gets closed.

Different kind of clients have different default limits:

- **Normal clients** have a default limit of 0, that means, no limit at all, because most normal clients use blocking implementations sending a single command and waiting for the reply to be completely read before sending the next command, so it is always not desirable to close the connection in case of a normal client.
- **Pub/Sub clients** have a default hard limit of 32 megabytes and a soft limit of 8 megabytes per 60 seconds.
- **Slaves** have a default hard limit of 256 megabytes and a soft limit of 64 megabyte per 60 second.

It is possible to change the limit at runtime using the CONFIG SET command or in a permanent way using the Redis configuration file `redis.conf`. See the example `redis.conf` in the Redis distribution for more information about how to set the limit.

Query buffer hard limit

Every client is also subject to a query buffer limit. This is a non-configurable hard limit that will close the connection when the client query buffer (that is the buffer we use to accumulate commands from the client) reaches 1 GB, and is actually only an extreme limit to avoid a server crash in case of client or server software bugs.

Client timeouts

By default recent versions of Redis don't close the connection with the client if the client is idle for many seconds: the connection will remain open forever.

However if you don't like this behavior, you can configure a timeout, so that if the client is idle for more than the specified number of seconds, the client connection will be closed.

You can configure this limit via `redis.conf` or simply using `CONFIG SET timeout <value>`.

Note that the timeout only applies to normal clients and it **does not apply to Pub/Sub clients**, since a Pub/Sub connection is a *push style* connection so a client that is idle is the norm.

Even if by default connections are not subject to timeout, there are two conditions when it makes sense to set a timeout:

- Mission critical applications where a bug in the client software may saturate the Redis server with idle connections, causing service disruption.
- As a debugging mechanism in order to be able to connect with the server if a bug in the client software saturates the server with idle connections, making it impossible to interact with the server.

Timeouts are not to be considered very precise: Redis avoids to set timer events or to run O(N) algorithms in order to check idle clients, so the check is performed incrementally from time to time. This means that it is possible that while the timeout is set to 10 seconds, the client connection will be closed, for instance, after 12 seconds if many clients are connected at the same time.

CLIENT command

The Redis client command allows to inspect the state of every connected client, to kill a specific client, to set names to connections. It is a very powerful debugging tool if you use Redis at scale.

CLIENT LIST is used in order to obtain a list of connected clients and their state:

```
redis 127.0.0.1:6379> client list
addr=127.0.0.1:52555 fd=5 name= age=855 idle=0 flags=N db=0 sub=0 psu
addr=127.0.0.1:52787 fd=6 name= age=6 idle=5 flags=N db=0 sub=0 psub=
```

In the above example session two clients are connected to the Redis server. The meaning of a few of the most interesting fields is the following:

- **addr**: The client address, that is, the client IP and the remote port number it used to connect with the Redis server.
- **fd**: The client socket file descriptor number.
- **name**: The client name as set by CLIENT SETNAME.
- **age**: The number of seconds the connection existed for.
- **idle**: The number of seconds the connection is idle.
- **flags**: The kind of client (N means normal client, check the full list of flags).
- **omem**: The amount of memory used by the client for the output buffer.
- **cmd**: The last executed command.

See the CLIENT LIST documentation for the full list of fields and their meaning.

Once you have the list of clients, you can easily close the connection with a client using the CLIENT KILL command specifying the client address as argument.

The commands CLIENT SETNAME and CLIENT GETNAME can be used to set and get the connection name. Starting with Redis 4.0, the client name is shown in the SLOWLOG output, so that it gets simpler to identify clients that are creating latency issues.

TCP keepalive

Recent versions of Redis (3.2 or greater) have TCP keepalive (`SO_KEEPALIVE` socket option) enabled by default and set to about 300 seconds. This option is useful in order to detect dead peers (clients that cannot be reached even if they look connected). Moreover, if there is network equipment between clients and servers that need to see some traffic in order to take the connection open, the option will prevent unexpected connection closed events.
