MongoClient timeout options to avoid unpredictable hangs in your application flow and improve performance.

(https://twitter.com/share?text=Configuring+timeout+options+for+%23mongo+-server+selection%2C+connection+%26+socket&url=https://scalegrid.io/blog/understanding-mongodb-client-timeout-options/&via=dharshanrg)

On an abstract level, whenever you use MongoClient to connect, send, or receive a request, it internally uses predefined timeout options to decide how long the application will wait for an initial connection establishment or the response from the server for your request.

Typical applications are interacting with different database servers based on the business logic. For example, your payment history might be on one database cluster and your analytics records on another cluster. The default time-outs can significantly influence the behavior of your application when there are network errors. If your analytics server is down, then each operation will wait for a default of 30s before failing (which may or may not be what you want).

Server Selection Timeout

Server selection timeout is the number of milliseconds the mongo driver will wait to select a server for an operation before giving up and raising an error.

This option was introduced in the newer version of next-generation Mongo drivers (version 3.2.x+ in Java). For each type of operation and user preference, the MongoClient selects the server using a selection algorithm to execute the operation.

For a write operation on a standalone server, there is only one server available which gets selected. In a replica set or sharded clusters, there can be more than one server which fills the user preference criteria for an operation.

Possible scenarios where server selection timeout can happen include – if a network is down or a primary node failure in a replica set.

Mongo driver uses 30s as a default value of the server selection timeout. Depending on the use case, we can increase or decrease this threshold.

Connection Timeout

Connection timeout is the number of milliseconds the driver will wait before a new connection attempt is aborted.

After selection of the server, the client tries to establish a connection with the server. Depending on network infrastructure and load on the server, the client may have to wait for a connection establishment. Possible scenarios where connection timeout can happen – the server gets shut down, network issues, wrong IP/DNS, port number etc

The default value of a connection timeout depends on the version and language of the driver. Mongo Java & Ruby latest driver versions have a 10s default timeout for connection establishments while the NodeJs driver has no timeout.

If the timeout is too high, you risk stalling your application. If the timeout is too low, you can give up too quickly. It is better to test with different values to find the right timeout for your application.

SocketTimeout

Socket timeout is the number of milliseconds a send or receive on a socket can take before timeout.

After establishing a connection with the server, the client sends a request to the server and receives the response back using an already established connection. Internally, the connection uses a socket to send the client request and receives the response

Mongo Java & Nodejs Driver have a default socket timeout of 0s which means basically no timeout. While Ruby offers a 5s socket timeout. You don't want to put a limit on this timeout as different operations will take the variable time to operate.

Further Exploration

MongoClient Timeout options vary on the different versions and languages of Mongo drivers. We encourage you to go through the documentation of the MongoClient class of your driver to understand your default timeout options. We have provided some sample code below to illustrate the timeout configuration in Java and Ruby.

MongoDB Java Driver

```
List<MongoCredential> creds = new ArrayList<MongoCredential>();  creds.add(MongoCredential.crea

MongoClientOptions.Builder optionsBuilder = MongoClientOptions.builder();

optionsBuilder.connectTimeout(CONNECTION_TIME_OUT_MS);

optionsBuilder.socketTimeout(SOCKET_TIME_OUT_MS);

optionsBuilder.serverSelectionTimeout(SERVER_SELECTION_TIMEOUT_MS);

MongoClientOptions options = optionsBuilder.build();

Mongo m = new MongoClient(new ServerAddress(server , port), creds, options);
```

MongoDB NodeJs Driver

```
var uri =  'mongodb://[username:password@]host[:port1]/[database]';
var options = { server:
               { socketOptions:
                    {
                        socketTimeoutMS: SOCKET_TIME_OUT_MS,
                        connectTimeoutMS: CONNECTION_TIMEOUT_MS
                    }
                }
              };
MongoClient.connect(uri, options, function(err, db) {
  if(!err) {
    console.log("We are connected");
  }
});
```