

## mgo - query performance seems consistently slow (500-650ms)

Ask Question

My data layer uses Mongo aggregation a decent amount, and on average, queries are taking 500-650ms to return. I am using mgo.

A sample query function is shown below which represents what most of my queries look like.

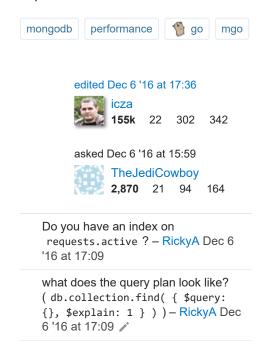
```
func (r userRepo) GetUserByID(id string) (User, error) {
   info, err := db.Info()
   if err != nil {
       log.Fatal(err)
   session, err := mgo.Dial(info.ConnectionString())
   if err != nil {
       log.Fatal(err)
   defer session.Close()
   var user User
   c := session.DB(info.Db()).C("users")
   o1 := bson.M{"$match": bson.M{"_id": id}}
   o2 := bson.M{"$project": bson.M{
        "first":
                          "$first",
       "last":
                          "$last",
        "email":
                          "$email",
                          "$fb_id"
        "fb_id":
                         "$groups",
        "groups":
        "fulfillments": "$fulfillments",
        "denied_requests": "$denied_requests",
        "invites":
                          "$invites",
        "requests": bson.M{
            "$filter": bson.M{
                "input": "$requests",
                "as": "item",
                "cond": bson.M{
                    "$eq": []interface{}{"$$item.active", true},
            },
       },
   pipeline := []bson.M{o1, o2}
   err = c.Pipe(pipeline).One(&user)
   if err != nil {
       return user, err
   return user, nil
```

The user struct I have looks like the following..

```
type User struct {
                                 `json:"id" bson:"_id,omitempty"
    ID
                   string
                                  json:"first" bson:"first"
    First
                   string
                                  json:"last" bson:"last"
    Last
                   string
                                 `json:"email" bson:"email"`
    Email
                   string
                                  json:"facebook_id" bson:"fb_id,omitempty"`
    FacebookID
                   string
                                 `json:"groups" bson:"groups"
                   []UserGroup
    Groups
                                 `json:"requests" bson:"requests"`
    Requests
                   []Request
                   []Fulfillment `json:"fulfillments" bson:"fulfillments"`
    Fulfillments
                   []GroupInvite `json:"invites" bson:"invites"`
    Invites
                                 `json:"denied_requests" bson:"denied_requests"`
    DeniedRequests []string
```

Based on what I have provided, is there anything obvious that would suggest why my queries are averaging 500-650ms?

I know that I am probably swallowing a bit of a performance hit by using aggregation pipeline, but I wouldn't expect it to be this bad.



## 1 Answer

.. is there anything obvious that would suggest why my queriers are averaging 500-650ms?

Yes, there is. You are calling mgo.Dial() before executing each query. mgo.Dial() has to connect to the MongoDB server every time,

which you close right after the query. The connection may very likely take hundreds of milliseconds to estabilish, including authentication, allocating resources (both at server and client side), etc. This is very wasteful.

## This method is generally called just once for a given cluster.

Further sessions to the same cluster are then established using the New or Copy methods on the obtained session. This will make them share the underlying cluster, and manage the pool of connections appropriately.

Create a global session variable, connect on startup *once* (using e.g. a package init() function), and use that session (or a copy / clone of it, obtained by <a href="Session.Copy()">Session.Copy()</a> or <a href="Session.Clone">Session.Clone()</a>). For example:

```
var session *mgo.Session
var info *db.Inf // Use your type here
func init() {
    var err error
    if info, err = db.Info(); err != r
        log.Fatal(err)
    if session, err = mgo.Dial(info.Come)
        log.Fatal(err)
}
func (r userRepo) GetUserByID(id strir
    sess := session.Clone()
    defer sess.Close()
    // Now we use sess to execute the
    var user User
    c := sess.DB(info.Db()).C("users")
   // Rest of the method is unchanged
}
```

answered Dec 6 '16 at 17:01

edited Dec 7 '16 at 1:45



This cut down the average time to ~75ms:) Thanks for pointing this out, I had a feeling I was doing something severely wrong. — TheJediCowboy

