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

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.

mongodb performance go mgo

edited Dec 6 '16 at 17:36

icza
155k 22 302 342

asked Dec 6 '16 at 15:59

TheJediCowboy
2,870 21 94 164

Do you have an index on `requests.active` ? – RickyA Dec 6 '16 at 17:09

what does the query plan look like? (`db.collection.find( { $query: {}, $explain: 1 } )`) – RickyA Dec 6 '16 at 17:09

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 `Session.Copy()` or `Session.Clone()`). 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 != n
        log.Fatal(err)
    }
    if session, err = mgo.Dial(info.Co
        log.Fatal(err)
    }
}

func (r userRepo) GetUserByID(id strin
    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
}
```

edited Dec 7 '16 at 1:45

answered Dec 6 '16 at 17:01

icza
155k 22 302 342

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

Dec 6 '16 at 17:29

This even doesn't work in stressful apps: github.com/go-mgo/mgo/issues/473 – Inanc Gumus
Aug 1 '17 at 15:00