Navid Malek
navidmalekedu@gmail.com
navidmalek.blog.ir

Testing Database response time

We will test 4 databases:

RDB:

-mysql

-postgres

No-SQL:

-mongo

-cassandra

Setting up the environment

We will install these databases in docker and test them (query them) from host machine (or another docker/VM)

docker command for setting up containers from source image:

for example: PostgreSQL

```
sudo docker run -itd -p 127.0.0.1:5432:5432 -v
/home/navidx/postgres/:/data/db --name postgres-nopayar -e
POSTGRES_PASSWORD=postgres -d postgres
```

MySQL:

```
sudo docker run -itd -p 127.0.0.1:3306:3306 --name mysql-nopayar
-e MYSQL_ROOT_PASSWORD=mysql -d mysql
```

and so on...

only set up images and map the ports to the host machine. (and if you have to set passwords, it is recommended to set them via docker E.V.)

For testing, all of these data bases have terminal emulator, for example for cassandra:

```
sudo docker exec -it cassandra-nopayar bash

cql
```

for mysql:

```
sudo docker exec -it mysql-nopayar bash

mysql
```

Warming up

after that we should spend some amount of time to learn about these databases, my advice is tutorials point**:**

for example cassandra

https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm

then do some query on these databases with their own terminal interface (psql,cql,...)

to warm up a little and better understand the architecture.

Key things to understand:

- how to index a column
  - both in no-sql and sql
- how to insert and select with criteria (condition) on indexed column

after that we should start to think about the testing, next move is to connect to these databases through some programming languages, for start i advice python (because of simplicity)

Key things to understand here:

- make a connection to database from host machine to docker
- generate random string of fixed size for data generation in database
- how to insert and select with criteria (condition) on indexed column

- calculate time of query for see the difference between response time

for example this is for postgres:

```
import psycopg2
import string
import random
import datetime
import time

#random data generation

def strGenerator(size=6, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.SystemRandom().choice(chars) for _ in range(size))

def numGenerator(size=6, chars= string.digits):
    return ''.join(random.SystemRandom().choice(chars) for _ in range(size))
#make connection
conn = psycopg2.connect("dbname='postgres' user='postgres' host='localhost'
password='postgres' ")

#make queries

cur = conn.cursor()
cur.execute("""CREATE TABLE testSpeed (
    code        CHAR (51)PRIMARY KEY,
    title       char(51),
    did         CHAR (51),
    code2        CHAR (51),
    title2     char(51),
    did2         CHAR (51),
    kind3        CHAR (51),
    title3       char(51),
    did3         CHAR (51),
    kind4        CHAR (51)
);""")

conn.commit()
cur.execute("""CREATE INDEX myindex
ON testSpeed (did);""")
conn.commit()

for i in range(500):
    query = "INSERT INTO testSpeed VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
    if i % 14 == 0:
        data = (
        strGenerator(51), strGenerator(51), "mike", strGenerator(51),
strGenerator(51), strGenerator(51),
        strGenerator(51), strGenerator(51), strGenerator(51), strGenerator(51))
    else:
        data =
(strGenerator(51),strGenerator(51),strGenerator(51),strGenerator(51),strGenerat
or(51),strGenerator(51),strGenerator(51),strGenerator(51),strGenerator(51),strG
enerator(51))
    cur.execute(query, data)

conn.commit()
```

```
cur.execute("select * from testSpeed WHERE did = 'mike'")

#mesure time

startTime = datetime.datetime.now().microsecond
rows = cur.fetchall()
endTime = datetime.datetime.now().microsecond
print(endTime - startTime)


cntr = 0
for i in rows:
    cntr += 1

print(cntr)
```

For example for mongodb:

```
from pymongo import MongoClient
import pymongo
import string
import random
import datetime
f = string.ascii_uppercase
print(f)

def strGenerator(size=6, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.SystemRandom().choice(chars) for _ in range(size))

def numGenerator(size=6, chars= string.digits):
    return ''.join(random.SystemRandom().choice(chars) for _ in range(size))


client = MongoClient('127.0.0.1', 27017)

# Get the sampleDB database
db = client.test
tests = db.tests
#result = db.tests.create_index([('author', pymongo.ASCENDING)], unique=False)
#print(sorted(list(db.tests.index_information())))
record = []
for i in range(500):
                record += [{"author": strGenerator(size=51),
                "text": strGenerator(size=51),
                "tags":strGenerator(size=51),
                "author1": strGenerator(size=51),
                "text1": strGenerator(size=51),
                "tags1":strGenerator(size=51),
                "author2": strGenerator(size=51),
                "text2": strGenerator(size=51),
                "tags2":strGenerator(size=51),
                "author3": strGenerator(size=51)
                }]
                if i % 14 == 0 :
                        record += [{"author": "mike",
                                    "text": strGenerator(size=51),
                                    "tags": strGenerator(size=51),
                                    "author1": strGenerator(size=51),
                                    "text1": strGenerator(size=51),
                                    "tags1": strGenerator(size=51),
                                    "author2": strGenerator(size=51),
                                    "text2": strGenerator(size=51),
                                    "tags2": strGenerator(size=51),
                                    "author3": strGenerator(size=51)
                                    }]

record_id = tests.insert_many(record)
# print(record_id.inserted_ids)
startTime = datetime.datetime.now().microsecond

recordsREt = tests.find({"author":"mike"})
endTime = datetime.datetime.now().microsecond
cntr = 0
for i in recordsREt:
        # print(i)
        cntr += 1
print(cntr)
print(endTime - startTime)
```

After that we should do some real stuff, what i mean is to implement a testing program for all of these databases in golang:

postgres:

https://github.com/lib/pq/blob/master/doc.go

https://medium.com/namely-labs/postgres-in-go-cf794adc4c52

https://gowalker.org/github.com/lib/pq

Testing with golang

First we should config the databases via their terminal emulators:

1. create test databases
2. create testtbl
3. create index on desired table columns (here txt2,txt5,txt8)

for example in postgre (psql):

```
CREATE TABLE testtbl(
  id   varchar(100)    PRIMARY KEY     NOT NULL,
  txt1          varchar(100),
  txt2           varchar(100),
  txt3       varchar(100),
  txt4       varchar(100),
   txt5    varchar(100),
   txt6    varchar(100),
   txt7    varchar(100),
   txt8    varchar(100),
   txt9    varchar(100),
   txt10   varchar(100)
);


CREATE INDEX salary_index ON testtbl (txt2,txt5,txt8);
```

MySQL:

```
create table testtbl(
   id   varchar(100)    NOT NULL,
   txt1           varchar(100),
   txt2            varchar(100),
   txt3        varchar(100),
   txt4        varchar(100),
    txt5    varchar(100),
    txt6    varchar(100),
    txt7    varchar(100),
    txt8    varchar(100),
    txt9    varchar(100),
    txt10   varchar(100),
   PRIMARY KEY (id)
);

CREATE INDEX salary_index
ON testtbl (txt2,txt5,txt8);
```

In mongo:

- ✔ we will cover this part in the golang code

In cassandra:

```
CREATE TABLE testtbl (
    id text,
    txt2 text,
    txt5 text,
    txt1 text,
    txt10 text,
    txt3 text,
    txt4 text,
    txt6 text,
    txt7 text,
    txt8 text,
    txt9 text,
    PRIMARY KEY (id)
)

CREATE INDEX myindex89 ON testtbl (txt2);
CREATE INDEX myindex2 ON testtbl (txt5);
```

Insert query codes in golang

Postgres:

```
package main

import (
      "database/sql"
      "fmt"
      _ "github.com/lib/pq"
      "time"
      "os"
      "strconv"
      "sync"
      "math/rand"
)

const (
      DB_USER     = "postgres"
      DB_PASSWORD = "postgres"
      DB_NAME     = "postgres"
)
func init() {
      rand.Seed(time.Now().UnixNano())
}


var letterRunes = 
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
      b := make([]rune, n)
      for i := range b {
            b[i] = letterRunes[rand.Intn(len(letterRunes))]
      }
      return string(b)
}
```

```
func timer(TimeProg int, wg *sync.WaitGroup)  {
	startTime := time.Now()
	for{
		elapsed := time.Now().Sub(startTime)
		elapsedMinute := int(elapsed.Minutes())
		if elapsedMinute >= TimeProg {
			wg.Done()
		}
	}

}


func main() {
	f, _ := os.Create("/tmp/dataSecondsInsertPostgre")
	defer f.Close()
	arguments := os.Args[1:]
	Time,_ := strconv.Atoi(arguments[0])
	size,_ := strconv.Atoi(arguments[1])
	wg := &sync.WaitGroup{}
	wg.Add(1)
	go timer(Time,wg)

	dbinfo := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=disable",
		DB_USER, DB_PASSWORD, DB_NAME)
	db, err := sql.Open("postgres", dbinfo)
	checkErr(err)
	defer db.Close()

	go inserter(f,err, db, size)
	wg.Wait()

}

func inserter(log *os.File,err error, db *sql.DB, size int) {
	cntr := 0
	sqlStatement := `INSERT INTO testtbl(id, txt1, txt2, txt3, txt4, txt5, 
txt6, txt7, txt8, txt9, txt10) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, 
$11)`
	for {
		cntr = cntr + 1
		startTime := time.Now()
		id := RandStringRunes(size)
		txt1 := RandStringRunes(size)
		txt2 := RandStringRunes(size)
		txt3 := RandStringRunes(size)
		txt4 := RandStringRunes(size)
		txt5 := RandStringRunes(size)
		txt6 := RandStringRunes(size)
		txt7 := RandStringRunes(size)
		txt8 := RandStringRunes(size)
		txt9 := RandStringRunes(size)
		txt10 := RandStringRunes(size)

		if cntr % 17 == 0{
			txt2 = "17"
			txt5 = "17"
```

```
            }else if cntr % 14 == 0 {
                  txt2 = "17"
            }
            _, err = 
db.Exec(sqlStatement,id,txt1,txt2,txt3,txt4,txt5,txt6,txt7,txt8,txt9,txt10)
            //err = db.QueryRow("INSERT INTO 
tutorials_tbl(tutorial_id,tutorial_title,tutorial_author) VALUES($1,$2,$3);", 
RandStringRunes(size), RandStringRunes(size), RandStringRunes(size)).Scan()
            checkErr(err)
            endTime := time.Now()
            timeElapsed := endTime.Sub(startTime)
            log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))
            str := " Connection Number" + strconv.Itoa(cntr) + "\n"
            log.WriteString(str)
      }
}

func checkErr(err error) {
      if err != nil {
            panic(err)
      }
}
```

MySQL:

```
package main

import (
//"fmt"
"database/sql"
_"github.com/go-sql-driver/mysql"
"strconv"
      "math/rand"
      "time"
      "os"
      "sync"
)
type Tag struct {
      ID   int    `json:"id"`
      Title string `json:"name"`
      Author string `json:"author"`
}

func init() {
      rand.Seed(time.Now().UnixNano())
}

var letterRunes = 
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
      b := make([]rune, n)
      for i := range b {
            b[i] = letterRunes[rand.Intn(len(letterRunes))]
```

```
	}
	return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup)  {
	startTime := time.Now()
	for{
		elapsed := time.Now().Sub(startTime)
		elapsedMinute := int(elapsed.Minutes())
		if elapsedMinute >= TimeProg {
			wg.Done()
		}
	}

}

func main() {

f, _ := os.Create("/tmp/dataSecondsInsertSql")
defer f.Close()
arguments := os.Args[1:]
Time,_ := strconv.Atoi(arguments[0])
size,_ := strconv.Atoi(arguments[1])
wg := &sync.WaitGroup{}
wg.Add(1)
go timer(Time,wg)

db, err := sql.Open("mysql", "root:mysql@tcp(127.0.0.1:3306)/test")

// if there is an error opening the connection, handle it
if err != nil {
panic(err.Error())
}

// defer the close till after the main function has finished
// executing
defer db.Close()
go mysql_inserter(f,err, db, size)
wg.Wait()

}

func mysql_inserter(log *os.File,err error, db *sql.DB, size int) {
	insert, err := db.Prepare("INSERT INTO testtbl VALUES
( ?, ?, ?,?, ?, ?,?, ?, ?,?,?)")
	defer insert.Close()
	if err != nil {
		panic(err.Error()) // proper error handling instead of panic in
your app
	}
	// if there is an error inserting, handle it
	if err != nil {
		panic(err.Error())
	}
	cntr := 0
	for {
		cntr = cntr + 1
		startTime := time.Now()
```

```
        id := RandStringRunes(size)
        txt1 := RandStringRunes(size)
        txt2 := RandStringRunes(size)
        txt3 := RandStringRunes(size)
        txt4 := RandStringRunes(size)
        txt5 := RandStringRunes(size)
        txt6 := RandStringRunes(size)
        txt7 := RandStringRunes(size)
        txt8 := RandStringRunes(size)
        txt9 := RandStringRunes(size)
        txt10 := RandStringRunes(size)

        if cntr % 17 == 0{
            txt2 = "17"
            txt5 = "17"
        }else if cntr % 14 == 0 {
            txt2 = "17"
        }

        _, err =
insert.Exec(id,txt1,txt2,txt3,txt4,txt5,txt6,txt7,txt8,txt9,txt10) // Insert
tuples (i, i^2)
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(),10))
        str := " Connection Number"+ strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
        if err != nil {
            panic(err.Error())
        }
    }
}
```

Mongo DB:

```
package main

import (
    "gopkg.in/mgo.v2"
    "time"
    "math/rand"
    "sync"
    "os"
    "strconv"
)
```

```
type DATAS struct  {
        ID string
        TXT1 string
        TXT2 string
        TXT3 string
        TXT4 string
        TXT5 string
        TXT6 string
        TXT7 string
        TXT8 string
        TXT9 string
        TXT10 string
}
func init() {
        rand.Seed(time.Now().UnixNano())
}



var letterRunes = []rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")


func RandStringRunes(n int) string {
        b := make([]rune, n)
        for i := range b {
                b[i] = letterRunes[rand.Intn(len(letterRunes))]
        }
        return string(b)
}
```

```
func timer(TimeProg int, wg *sync.WaitGroup)  {
        startTime := time.Now()
        for{
                elapsed := time.Now().Sub(startTime)
                elapsedMinute := int(elapsed.Minutes())
                if elapsedMinute >= TimeProg {
                        wg.Done()
                }
        }

}



func main() {
        f, _ := os.Create("/tmp/dataSecondsInsertMongo")
        defer f.Close()
        arguments := os.Args[1:]
        Time,_ := strconv.Atoi(arguments[0])
        size,_ := strconv.Atoi(arguments[1])
        wg := &sync.WaitGroup{}
        wg.Add(1)
        go timer(Time,wg)

        db,e := mgo.Dial("localhost")
        err(e)
        defer db.Close()
        //db.SetMode(mgo.Monotonic,true)
```

```
    c := db.DB("test").C("testtbl")

    i := mgo.Index{
        Key: []string{"txt2","txt5","txt8"},
        Unique: false,
        DropDups: false,
        Background: false,
        Sparse: false,
    }

    e = c.EnsureIndex(i)
    err(e)
}

func insert(c *mgo.Collection,size int,log *os.File){
    cntr := 0
    for  {
        startTime:=time.Now()
        cntr = cntr + 1
        id := RandStringRunes(size)
        txt1D := RandStringRunes(size)
        txt2D := RandStringRunes(size)
        txt3D := RandStringRunes(size)
        txt4D := RandStringRunes(size)
        txt5D := RandStringRunes(size)
        txt6D := RandStringRunes(size)
        txt7D := RandStringRunes(size)
        txt8D := RandStringRunes(size)
        txt9D := RandStringRunes(size)
```

```
		txt10D := RandStringRunes(size)

		if cntr % 17 == 0{

			txt2D = "17"

			txt5D = "17"

		}else if cntr % 14 == 0 {

			txt2D = "17"

		}

		e := c.Insert(&DATAS{ID:id, TXT1:txt1D, TXT2:txt2D,TXT3:txt3D,TXT4:txt4D,TXT5:txt5D,TXT6:txt6D,TXT7:txt7D,TXT8:txt8D,TXT9:txt9D,TXT10:txt10D})

		err(e)

		endTime := time.Now()

		timeElapsed := endTime.Sub(startTime)

		log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))

		str := " Connection Number" + strconv.Itoa(cntr) + "\n"

		log.WriteString(str)

	}

}


func err(e error)  {

	if e != nil{

		panic(e)

	}

}
```

Cassandra:

- ✔ The table definition above is not sufficient for indexed query, cassandra is some kind of different from other 3 databases, it uses partitioning and clustering by default (the primary key is made up of partition keys and clustering keys), so with the deffinition above we should use « allow filtering » in the end of select query which is really really slow...

- ✔ in order to query with high speed we should use the partitioning and clustering approach which is going to be an unfair test , so i will include both of these approaches in results.

  The table definition for approach one:

```
CREATE TABLE testtbl (
    id text,
    txt2 text,
    txt5 text,
    txt1 text,
    txt10 text,
    txt3 text,
    txt4 text,
    txt6 text,
    txt7 text,
    txt8 text,
    txt9 text,
    PRIMARY KEY (id,txt2,txt5,txt3)
)

CREATE INDEX myindex89 ON testtbl (txt2);
CREATE INDEX myindex2 ON testtbl (txt5);
```

Golang code for first approach:

```
package main

import (
      "github.com/gocql/gocql"
      "os"
      "strconv"
      "sync"
      "math/rand"
      "time"
)

func init() {
      rand.Seed(time.Now().UnixNano())
}


var letterRunes =
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
      b := make([]rune, n)
      for i := range b {
            b[i] = letterRunes[rand.Intn(len(letterRunes))]
      }
      return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup)  {
    startTime := time.Now()
    for{
        elapsed := time.Now().Sub(startTime)
        elapsedMinute := int(elapsed.Minutes())
        if elapsedMinute >= TimeProg {
            wg.Done()
        }
    }

}


func main() {

    f, _ := os.Create("/tmp/dataSecondsInsertCassandra")
    defer f.Close()
    arguments := os.Args[1:]
    Time,_ := strconv.Atoi(arguments[0])
    size,_ := strconv.Atoi(arguments[1])
    wg := &sync.WaitGroup{}
    wg.Add(1)
    go timer(Time,wg)

    // connect to the cluster
    cluster := gocql.NewCluster("127.0.0.1")
    cluster.Keyspace = "tutorialspoint"
    cluster.Consistency = gocql.One
    session, _ := cluster.CreateSession()
    defer session.Close()

    go inserter(session,f,size)

    wg.Wait()
}

func inserter(session *gocql.Session,log *os.File, size int ) error {
    cntr := 0
    for {
        cntr = cntr + 1
        startTime := time.Now()
        id := RandStringRunes(size)
        txt1 := RandStringRunes(size)
        txt2 := RandStringRunes(size)
        txt3 := RandStringRunes(size)
        txt4 := RandStringRunes(size)
        txt5 := RandStringRunes(size)
        txt6 := RandStringRunes(size)
        txt7 := RandStringRunes(size)
        txt8 := RandStringRunes(size)
        txt9 := RandStringRunes(size)
        txt10 := RandStringRunes(size)
        if cntr % 17 == 0{
            id = "abbas" //comment this line for second approach
            txt2 = "17"
            txt5 = "17"
        }else if cntr % 14 == 0 {
            id ="abbas" //comment this line for second approach
            txt2 = "17"
        }
        err := session.Query("INSERT INTO tsttbl(id, txt1, txt2, txt3, 
txt4,txt5,txt6,txt7,txt8,txt9,txt10) VALUES 
(?,?,?,?,?,?,?,?,?,?,?)",id,txt1,txt2,txt3,txt4,txt5,txt6,txt7,txt8,txt9,txt10)
.Exec()
        checkErr(err)
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))
        str := " Connection Number" + strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
    }
}
func checkErr(err error) {
    if err != nil {
        panic(err)
    }
}
```

Read queries in golang

Postgres:

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
    "time"
    "os"
    "strconv"
    "sync"
    "math/rand"
)

const (
    DB_USER     = "postgres"
    DB_PASSWORD = "postgres"
    DB_NAME     = "postgres"
)
func init() {
    rand.Seed(time.Now().UnixNano())
}
```

```
var letterRunes =
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
	b := make([]rune, n)
	for i := range b {
		b[i] = letterRunes[rand.Intn(len(letterRunes))]
	}
	return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup)  {
	startTime := time.Now()
	for{
		elapsed := time.Now().Sub(startTime)
		elapsedMinute := int(elapsed.Minutes())
		if elapsedMinute >= TimeProg {
			wg.Done()
		}
	}

}


func main() {
	f, _ := os.Create("/tmp/dataSecondsReadPostgre")
	defer f.Close()
	arguments := os.Args[1:]
	Time,_ := strconv.Atoi(arguments[0])
	size,_ := strconv.Atoi(arguments[1])
	wg := &sync.WaitGroup{}
	wg.Add(1)
	go timer(Time,wg)

	dbinfo := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=disable",
		DB_USER, DB_PASSWORD, DB_NAME)
	db, err := sql.Open("postgres", dbinfo)
	checkErr(err)
	defer db.Close()

	go reader(f,err, db, size)
	wg.Wait()
}
func reader(log *os.File,err error, db *sql.DB, size int) {
	cntr := 0
	stmt, err := db.Prepare("select txt8 from testtbl where txt2='17' and txt5='17'")
	checkErr(err)
	for {
        cntr = cntr + 1
        startTime := time.Now()
        _, err = stmt.Exec()
        checkErr(err)
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))
        str := " Connection Number" + strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
    }
}
func checkErr(err error) {
    if err != nil {
        panic(err)
    }
}
```

MySQL:

```
package main

import (
    //"fmt"
    "database/sql"
    _"github.com/go-sql-driver/mysql"
    "strconv"
    "math/rand"
    "time"
    "os"
    "sync"
)
type Tag struct {
    ID   int    `json:"id"`
    Title string `json:"name"`
    Author string `json:"author"`
}

func init() {
    rand.Seed(time.Now().UnixNano())
}

var letterRunes =
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
    b := make([]rune, n)
    for i := range b {
            b[i] = letterRunes[rand.Intn(len(letterRunes))]
    }
    return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup)  {
    startTime := time.Now()
    for{
            elapsed := time.Now().Sub(startTime)
            elapsedMinute := int(elapsed.Minutes())
            if elapsedMinute >= TimeProg {
                    wg.Done()
            }
    }

}

func main() {

    f, _ := os.Create("/tmp/dataSecondsReadSql")
    defer f.Close()
    arguments := os.Args[1:]
    Time,_ := strconv.Atoi(arguments[0])
    size,_ := strconv.Atoi(arguments[1])
    wg := &sync.WaitGroup{}
    wg.Add(1)
    go timer(Time,wg)

    db, err := sql.Open("mysql", "root:mysql@tcp(127.0.0.1:3306)/test")

    // if there is an error opening the connection, handle it
    if err != nil {
            panic(err.Error())
    }

    // defer the close till after the main function has finished
    // executing
    defer db.Close()
    go mysql_reader(f,err, db, size)
    wg.Wait()
}

func mysql_reader(log *os.File,err error, db *sql.DB, size int) {
    read, err := db.Prepare("select txt8 from testtbl where txt2='17' and txt5='17'")
    defer read.Close()
    if err != nil {
            panic(err.Error()) // proper error handling instead of panic in your app
```

```
    }
    // if there is an error inserting, handle it
    if err != nil {
        panic(err.Error())
    }
    cntr := 0
    for {
        cntr = cntr + 1
        startTime := time.Now()
        _, err = read.Exec()
        if err != nil {
            panic(err.Error())
        }
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(),10))
        str := " Connection Number"+ strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
        if err != nil {
            panic(err.Error())
        }
    }
}
```

MongoDB:

```

package main

import (
    "gopkg.in/mgo.v2"
    "time"
    "math/rand"
    "sync"
    "os"
    "strconv"
    "gopkg.in/mgo.v2/bson"
)

//type DATAS struct  {
//      TXT8 string
//}
type DATAS struct {
    ID    string
    TXT1  string
    TXT2  string
    TXT3  string
    TXT4  string
```

```
	TXT5  string
	TXT6  string
	TXT7  string
	TXT8  string
	TXT9  string
	TXT10 string
}

func init() {
	rand.Seed(time.Now().UnixNano())
}

var letterRunes =
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
	b := make([]rune, n)
	for i := range b {
		b[i] = letterRunes[rand.Intn(len(letterRunes))]
	}
	return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup) {
	startTime := time.Now()
	for {
		elapsed := time.Now().Sub(startTime)
		elapsedMinute := int(elapsed.Minutes())
		if elapsedMinute >= TimeProg {
			wg.Done()
		}
	}

}

func main() {
	f, _ := os.Create("/tmp/dataSecondsReadMongo")
	defer f.Close()
	arguments := os.Args[1:]
	Time, _ := strconv.Atoi(arguments[0])
	size,_ := strconv.Atoi(arguments[1])
	wg := &sync.WaitGroup{}
	wg.Add(1)


	db, e := mgo.Dial("localhost")
	err(e)
	defer db.Close()
	//db.SetMode(mgo.Monotonic,true)
```

```
        c := db.DB("test").C("testtbl")

        i := mgo.Index{
                Key:        []string{"txt2", "txt5", "txt8"},
                Unique:     false,
                DropDups:   false,
                Background: false,
                Sparse:     false,
        }

        e = c.EnsureIndex(i)
        err(e)

        go timer(Time, wg)
        go read(c,size,f)

        wg.Wait()

}

func read(c *mgo.Collection,size int,log *os.File){
        cntr := 0
        for  {
                cntr = cntr + 1
                var res []DATAS
                arr := []bson.M{
                        {"txt2": "17"},
                        {"txt5": "17"},
                }
                startTime:=time.Now()
                c.Find(bson.M{"$and": arr}).Select(bson.M{"txt8": 1, "_id": 0}).All(&res)
                endTime := time.Now()
                timeElapsed := endTime.Sub(startTime)
                log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))
                str := " Connection Number" + strconv.Itoa(cntr) + "\n"
                log.WriteString(str)
        }
}

func err(e error) {
        if e != nil {
                panic(e)
        }
}
```

Cassandra:

- ✔ Code for first approach
- ✔ read comment in query for second approach « //add ALLOW FILTERING for second approach and remove id='abbas' »

```
package main

import (
	"github.com/gocql/gocql"
	"os"
	"strconv"
	"sync"
	"math/rand"
	"time"
)

func init() {
	rand.Seed(time.Now().UnixNano())
}


var letterRunes =
[]rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")

func RandStringRunes(n int) string {
	b := make([]rune, n)
	for i := range b {
		b[i] = letterRunes[rand.Intn(len(letterRunes))]
	}
	return string(b)
}

func timer(TimeProg int, wg *sync.WaitGroup)  {
	startTime := time.Now()
	for{
		elapsed := time.Now().Sub(startTime)
		elapsedMinute := int(elapsed.Minutes())
		if elapsedMinute >= TimeProg {
			wg.Done()
		}
	}

}


func main() {

	f, _ := os.Create("/tmp/dataSecondsReadCassandra")
	defer f.Close()
```

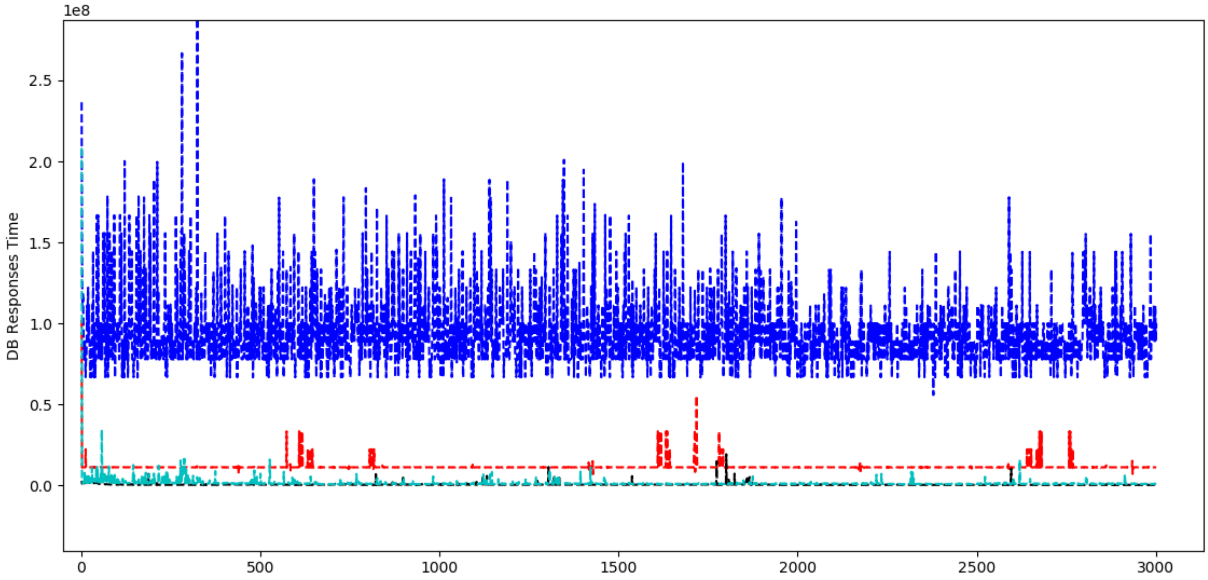
```
        arguments := os.Args[1:]
        Time,_ := strconv.Atoi(arguments[0])
        size,_ := strconv.Atoi(arguments[1])
        wg := &sync.WaitGroup{}
        wg.Add(1)
        go timer(Time,wg)

        // connect to the cluster
        cluster := gocql.NewCluster("127.0.0.1")
        cluster.Keyspace = "tutorialspoint"
        cluster.Consistency = gocql.One
        session, _ := cluster.CreateSession()
        defer session.Close()

        go read(session,f,size)

        wg.Wait()
}

func read(session *gocql.Session,log *os.File, size int ) error {
        cntr := 0
        for {
                cntr = cntr + 1
                startTime := time.Now()
                err := session.Query("SELECT txt8 FROM tsttbl WHERE id='abbas' and txt2='17'
and txt5='17' ").Exec()
//add ALLOW FILTERING for second approach and remove id='abbas'
                checkErr(err)
                endTime := time.Now()
                timeElapsed := endTime.Sub(startTime)
                log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(), 10))
                str := " Connection Number" + strconv.Itoa(cntr) + "\n"
                log.WriteString(str)
        }
}
func checkErr(err error) {
        if err != nil {
                panic(err)
        }
}
```

Navid Malek
navidmalekedu@gmail.com
navidmalek.blog.ir

Results

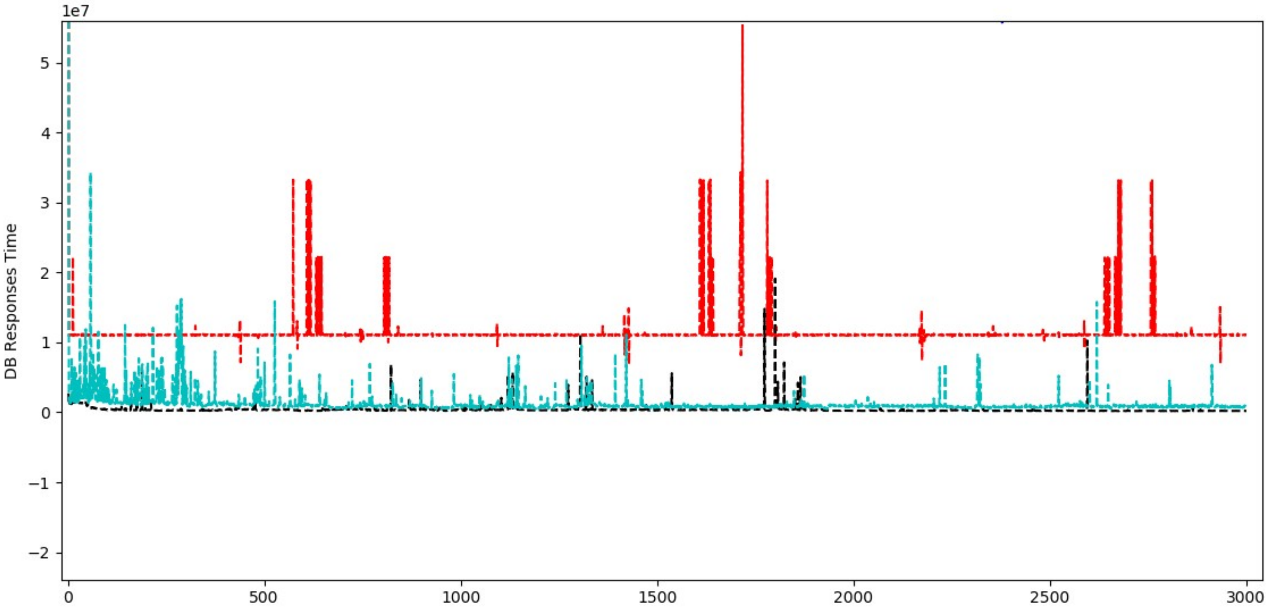
insert query (write time) :

runtime 5 mins, size = 90 chars

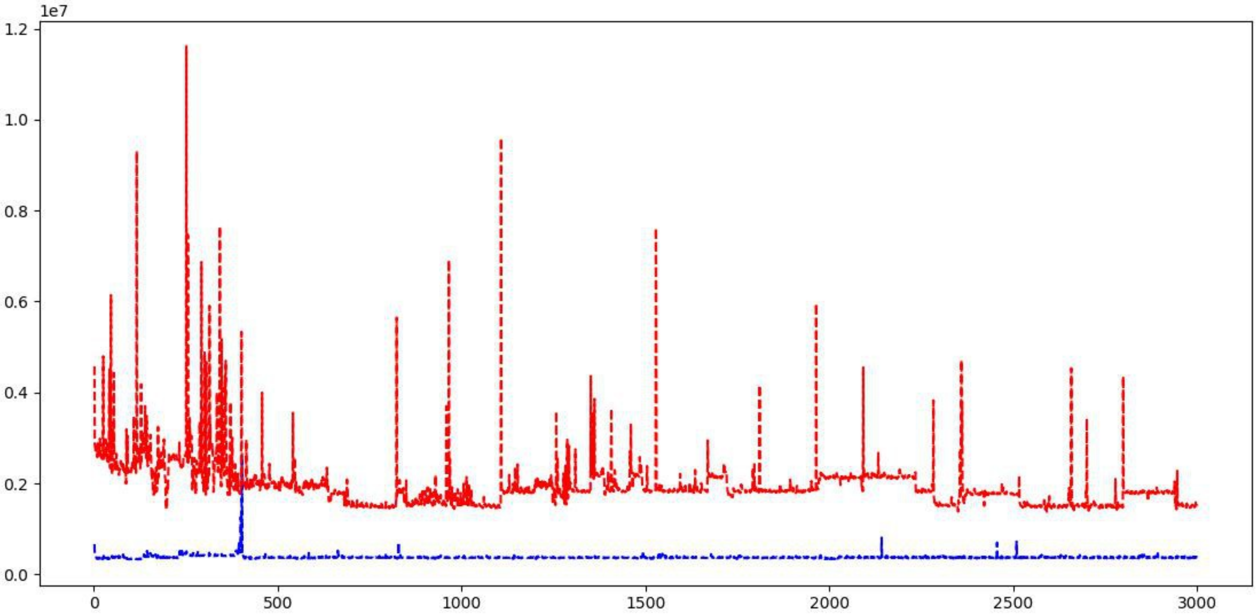


dark blue: MySQL   red: postgres   light blue :cassandra   black: mongodb

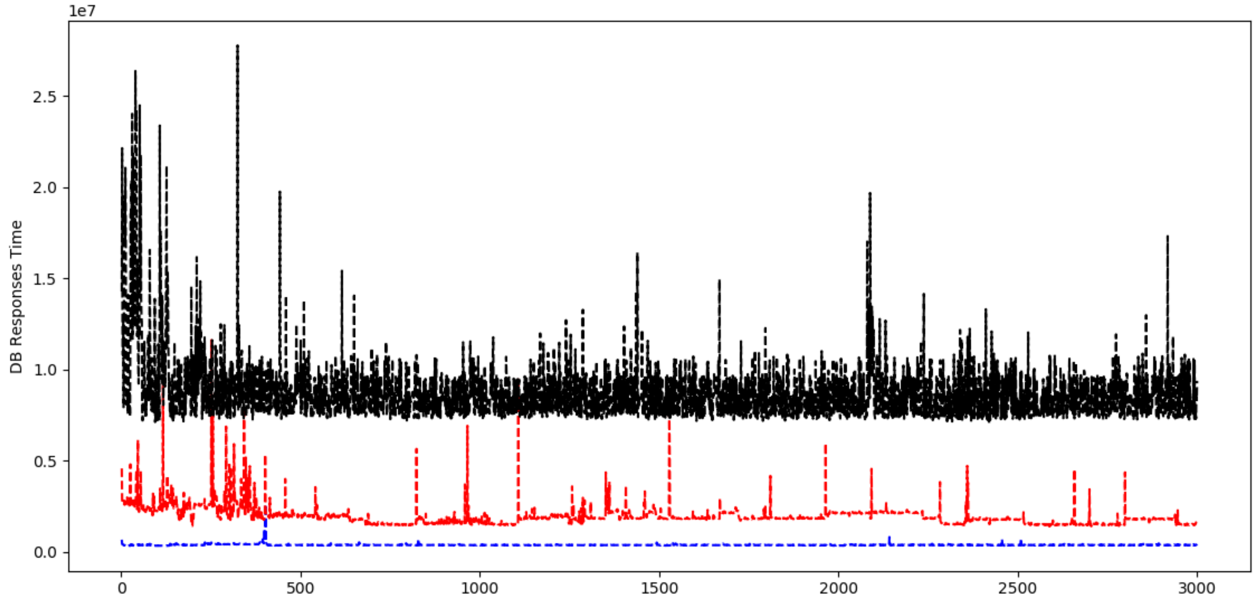
mongo is a little bit better than cassandra (both first and second approach).



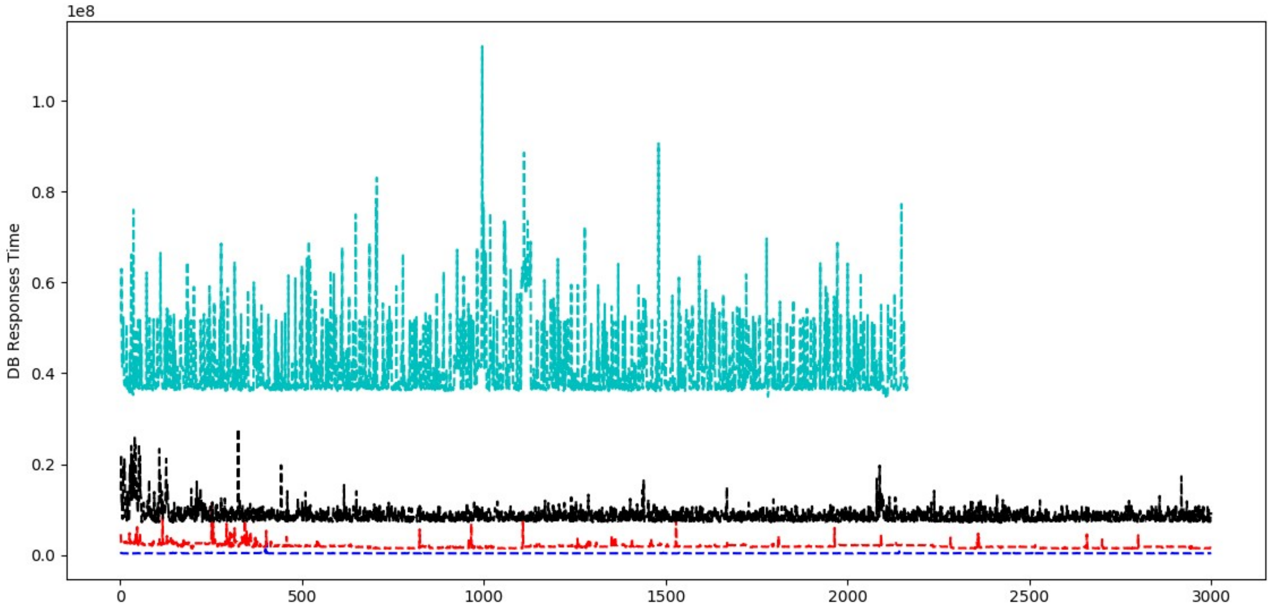
Find query (Read time) :



red: postgre blue: mysql



red: postgre blue: mysql black: mongo

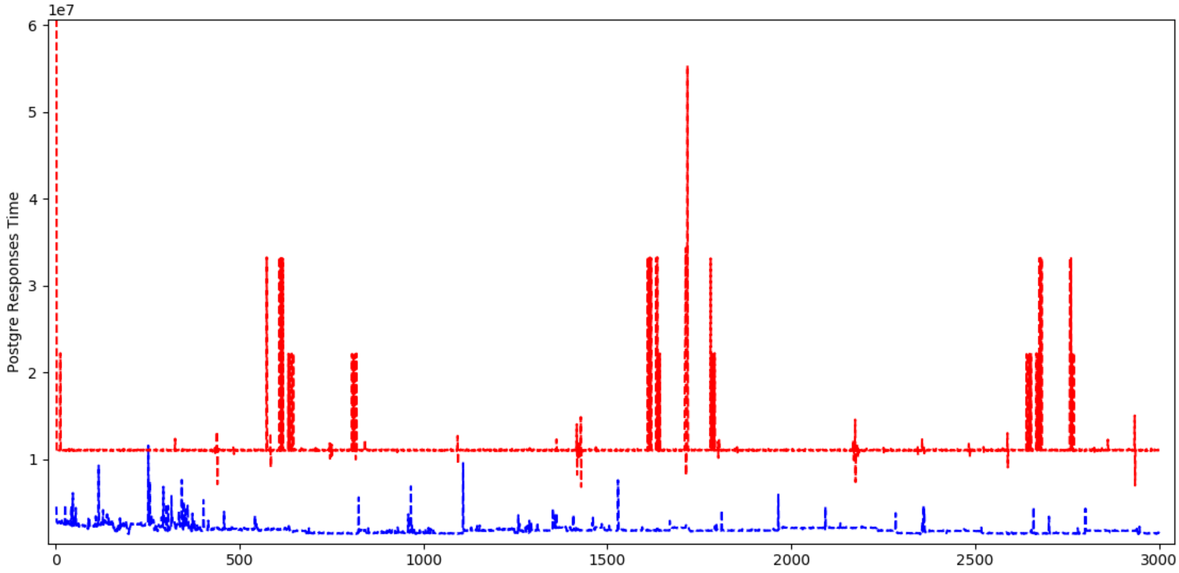


red: postgre blue: mysql black: mongo light blue: cassandra => first approach

- ✔ the second approach for cassandra is a way much slower than first approach (because it lacks indexing) so we only use first approach , but as you see in this amount of data it is still slower than other databases specially MySQL.

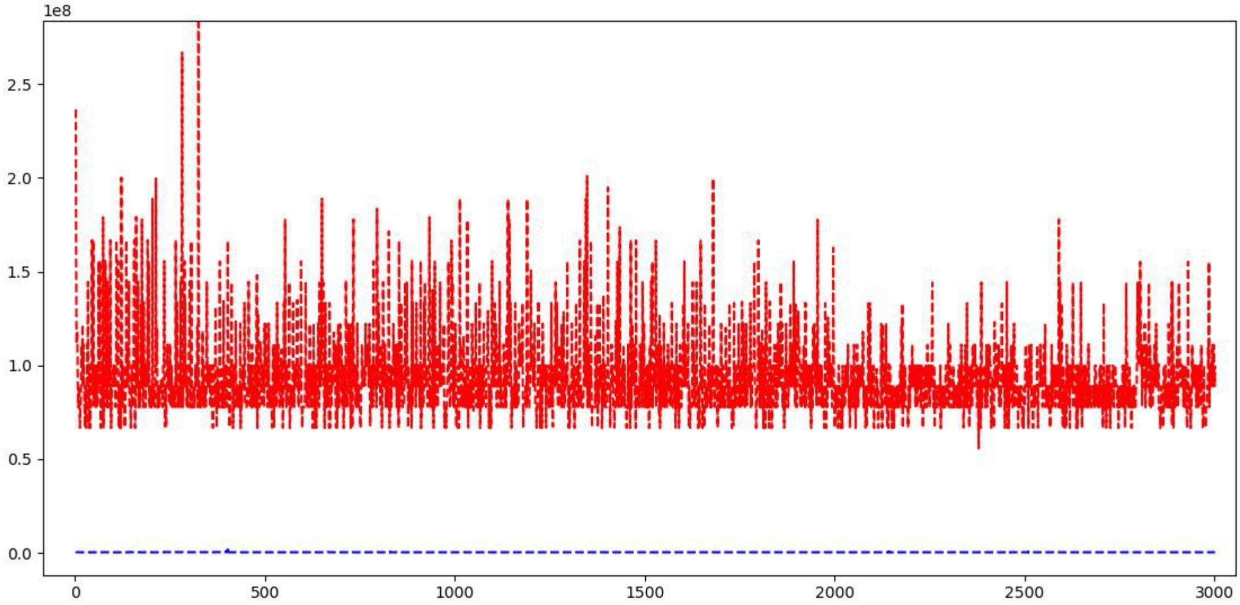
Navid Malek
navidmalekedu@gmail.com
navidmalek.blog.ir

Comparing read and write time in postgres:



red : write   blue : read

comparing read and write time for MySQL:



so for our usage, MySQL is a quite fast database for reading, but not very good at inserting new data.

We will go with MySQL because:

- ✔ We have more read than write (maybe 10x)
- ✔ we can tune MySQL write time
- ✔ partitioning and sharding MySQL can be helpful in large amount of datas

Partitioning MySQL:

MySQL statements in database:

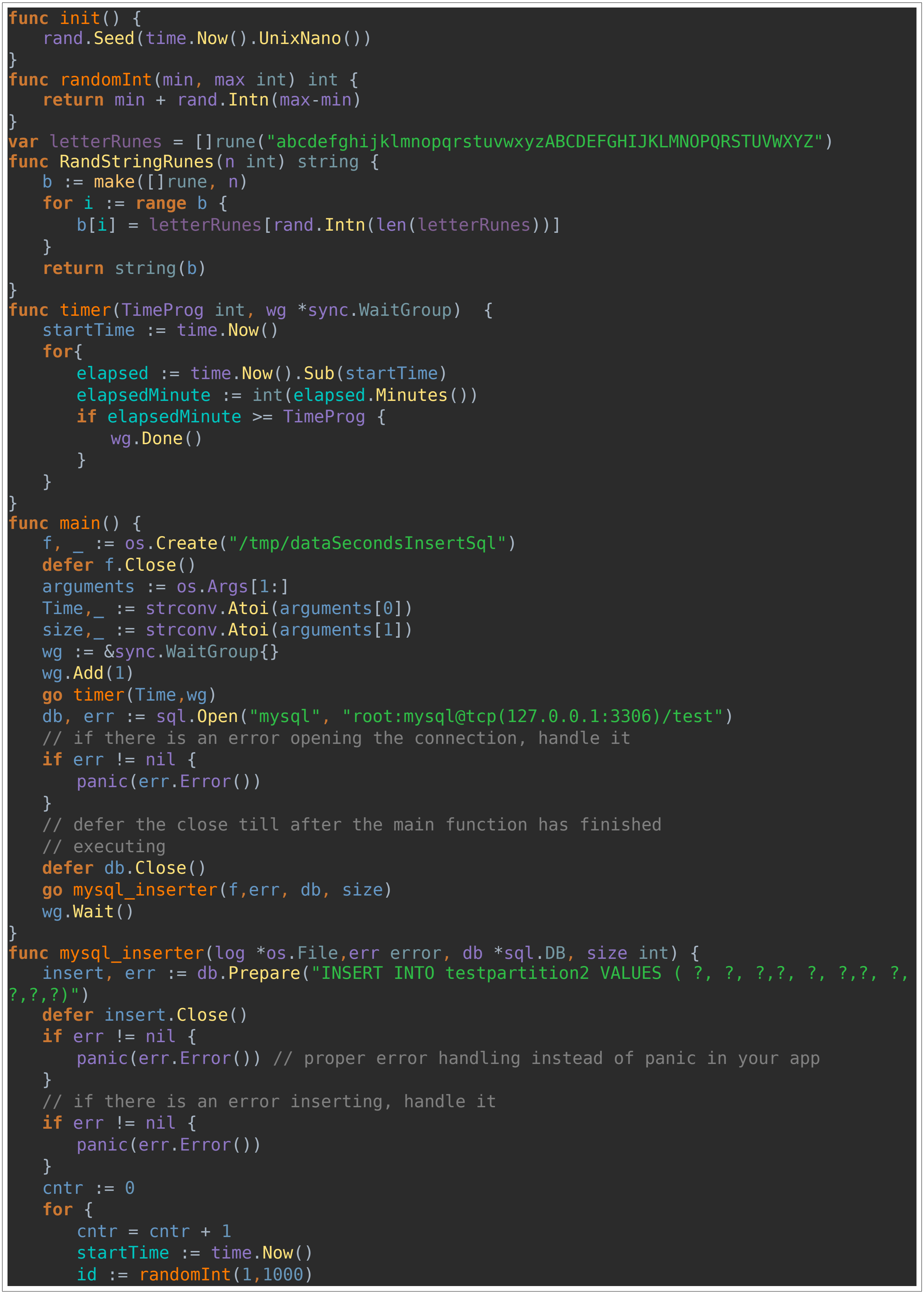
create testpartition2 table

```
create table testpartition2(    id    INTEGER    NOT NULL,    txt1
varchar(100),    txt2            INTEGER NOT NULL,    txt3
varchar(100),    txt4        varchar(100),     txt5    INTEGER
NOT NULL,     txt6    varchar(100),     txt7    varchar(100),
txt8    varchar(100),     txt9    varchar(100),     txt10
varchar(100),    PRIMARY KEY (id,txt2,txt5) );
```

Insert data in them using golang:

runtime 15 minutes, size 90 chars

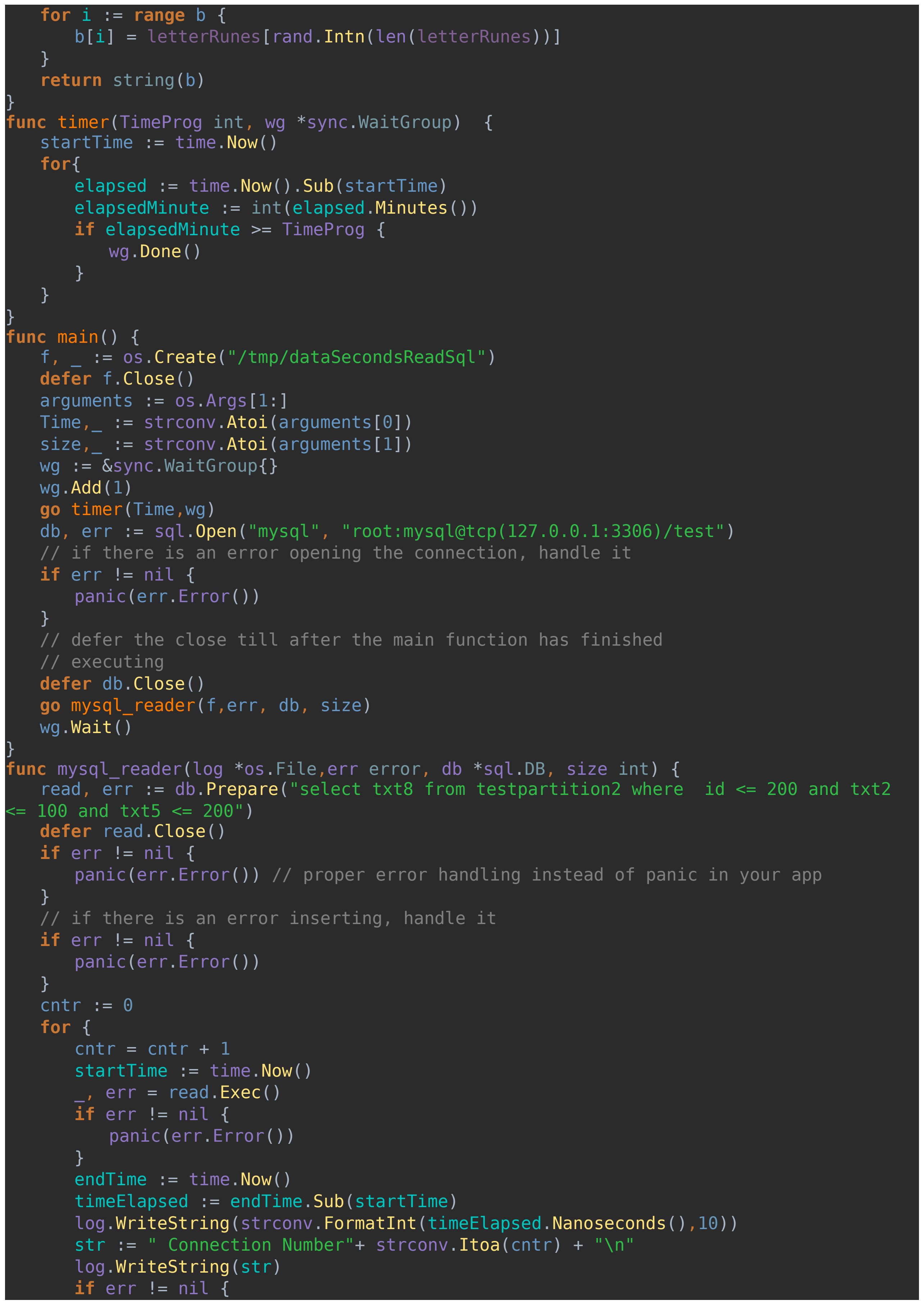
```
package main
import (
   "database/sql"
   _"github.com/go-sql-driver/mysql"
   "strconv"
   "math/rand"
   "time"
   "os"
   "sync"
)
type Tag struct {
   ID   int     `json:"id"`
   Title string `json:"name"`
   Author string `json:"author"`
}
```



```
func init() {
    rand.Seed(time.Now().UnixNano())
}
func randomInt(min, max int) int {
    return min + rand.Intn(max-min)
}
var letterRunes = []rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
func RandStringRunes(n int) string {
    b := make([]rune, n)
    for i := range b {
        b[i] = letterRunes[rand.Intn(len(letterRunes))]
    }
    return string(b)
}
func timer(TimeProg int, wg *sync.WaitGroup)  {
    startTime := time.Now()
    for{
        elapsed := time.Now().Sub(startTime)
        elapsedMinute := int(elapsed.Minutes())
        if elapsedMinute >= TimeProg {
            wg.Done()
        }
    }
}
func main() {
    f, _ := os.Create("/tmp/dataSecondsInsertSql")
    defer f.Close()
    arguments := os.Args[1:]
    Time,_ := strconv.Atoi(arguments[0])
    size,_ := strconv.Atoi(arguments[1])
    wg := &sync.WaitGroup{}
    wg.Add(1)
    go timer(Time,wg)
    db, err := sql.Open("mysql", "root:mysql@tcp(127.0.0.1:3306)/test")
    // if there is an error opening the connection, handle it
    if err != nil {
        panic(err.Error())
    }
    // defer the close till after the main function has finished
    // executing
    defer db.Close()
    go mysql_inserter(f,err, db, size)
    wg.Wait()
}
func mysql_inserter(log *os.File,err error, db *sql.DB, size int) {
    insert, err := db.Prepare("INSERT INTO testpartition2 VALUES ( ?, ?, ?,?, ?, ?,?, ?,
?,?,?)")
    defer insert.Close()
    if err != nil {
        panic(err.Error()) // proper error handling instead of panic in your app
    }
    // if there is an error inserting, handle it
    if err != nil {
        panic(err.Error())
    }
    cntr := 0
    for {
        cntr = cntr + 1
        startTime := time.Now()
        id := randomInt(1,1000)
        txt1 := RandStringRunes(size)
        txt2 := randomInt(1,1000)
        txt3 := RandStringRunes(size)
        txt4 := RandStringRunes(size)
        txt5 := randomInt(1,1000)
        txt6 := RandStringRunes(size)
        txt7 := RandStringRunes(size)
        txt8 := RandStringRunes(size)
        txt9 := RandStringRunes(size)
        txt10 := RandStringRunes(size)
        if cntr % 17 == 0{
            id = randomInt(1,200)
            txt2 = randomInt(1,100)
            txt5 = randomInt(1,50)
        }else if cntr % 14 == 0 {
            id = randomInt(1,200)
            txt2 = randomInt(1,100)
            txt5 = randomInt(1,200)
        }
        _, err = insert.Exec(id,txt1,txt2,txt3,txt4,txt5,txt6,txt7,txt8,txt9,txt10) // Insert tuples (i, i^2)
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(),10))
        str := " Connection Number"+ strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
        if err != nil {
            panic(err.Error())
        }
    }
}
```

The Read query will be:

```
package main
import (
    //"fmt"
    "database/sql"
    _"github.com/go-sql-driver/mysql"
    "strconv"
    "math/rand"
    "time"
    "os"
    "sync"
)
type Tag struct {
    ID    int     `json:"id"`
    Title string `json:"name"`
    Author string `json:"author"`
}
func init() {
    rand.Seed(time.Now().UnixNano())
}
var letterRunes = []rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
func RandStringRunes(n int) string {
    b := make([]rune, n)
```



```
    for i := range b {
        b[i] = letterRunes[rand.Intn(len(letterRunes))]
    }
    return string(b)
}
func timer(TimeProg int, wg *sync.WaitGroup)  {
    startTime := time.Now()
    for{
        elapsed := time.Now().Sub(startTime)
        elapsedMinute := int(elapsed.Minutes())
        if elapsedMinute >= TimeProg {
            wg.Done()
        }
    }
}
func main() {
    f, _ := os.Create("/tmp/dataSecondsReadSql")
    defer f.Close()
    arguments := os.Args[1:]
    Time,_ := strconv.Atoi(arguments[0])
    size,_ := strconv.Atoi(arguments[1])
    wg := &sync.WaitGroup{}
    wg.Add(1)
    go timer(Time,wg)
    db, err := sql.Open("mysql", "root:mysql@tcp(127.0.0.1:3306)/test")
    // if there is an error opening the connection, handle it
    if err != nil {
        panic(err.Error())
    }
    // defer the close till after the main function has finished
    // executing
    defer db.Close()
    go mysql_reader(f,err, db, size)
    wg.Wait()
}
func mysql_reader(log *os.File,err error, db *sql.DB, size int) {
    read, err := db.Prepare("select txt8 from testpartition2 where  id <= 200 and txt2
<= 100 and txt5 <= 200")
    defer read.Close()
    if err != nil {
        panic(err.Error()) // proper error handling instead of panic in your app
    }
    // if there is an error inserting, handle it
    if err != nil {
        panic(err.Error())
    }
    cntr := 0
    for {
        cntr = cntr + 1
        startTime := time.Now()
        _, err = read.Exec()
        if err != nil {
            panic(err.Error())
        }
        endTime := time.Now()
        timeElapsed := endTime.Sub(startTime)
        log.WriteString(strconv.FormatInt(timeElapsed.Nanoseconds(),10))
        str := " Connection Number"+ strconv.Itoa(cntr) + "\n"
        log.WriteString(str)
        if err != nil {
```

```
            panic(err.Error())
        }
    }
}
```

We will run the read query three times for 5 minutes:

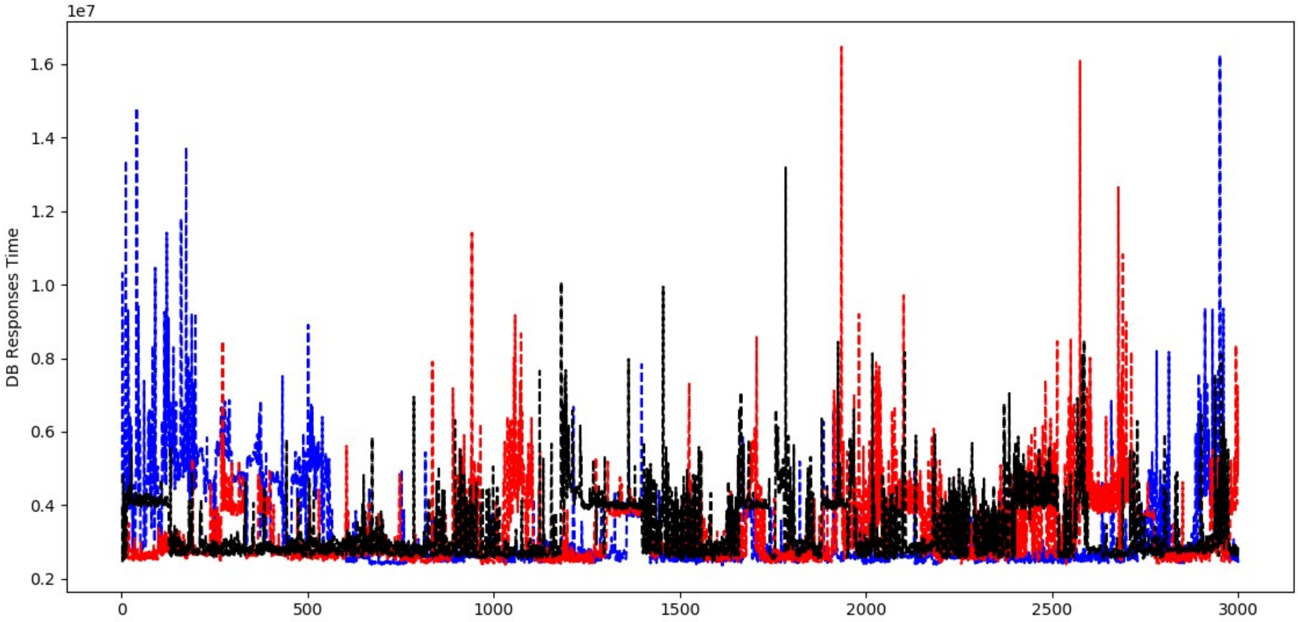
1. without anything
2. with index on txt2,txt5
3. with index and partitioning

for partitioning and indexing in MySQL terminal:

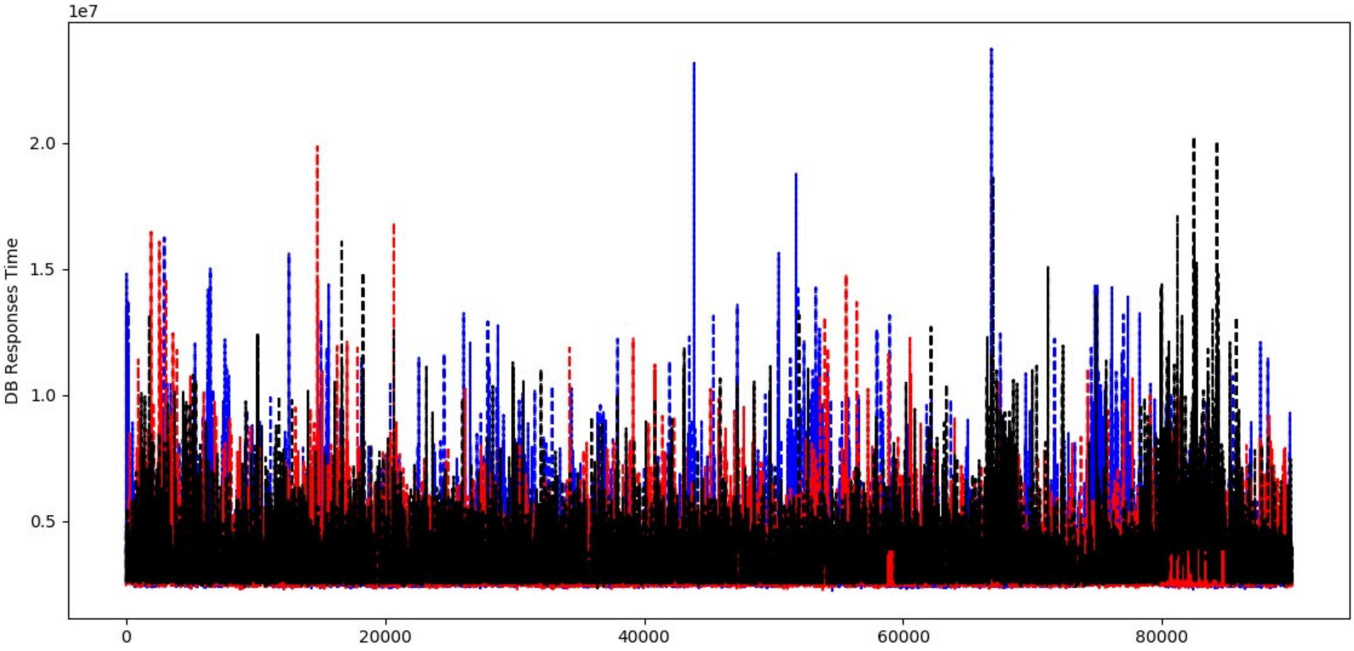
```
CREATE INDEX testpartition2 ON testpartition2 (id,txt2,txt5);


 alter table testpartition2
    ->     partition by range columns(id,txt2,txt5)(
    ->     PARTITION p0 VALUES LESS THAN (201, 101, 51),
    ->  PARTITION p1 VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE)
    ->     );
```

Results

for 7957 rows of data in database



blue: without anything red: with index black: with index and partitioning



for 90000 responses

So partitioning will help in reducing read time.

Navid Malek
navidmalekedu@gmail.com
navidmalek.blog.ir

Tuning MySQL

We have two goals for tuning:

- ✔ Better write performance
- ✔ Least effect on Read performance

The main variables to change are here:

- https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_keep_files_on_create
- https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_adaptive_flushing
- https://dev.mysql.com/doc/refman/8.0/en/optimizing-innodb-configuration-variables.html

After detecting the best variables, we start to change them in here (MySQL config file):

MySQL config ==> /etc/mysql/my.cnf

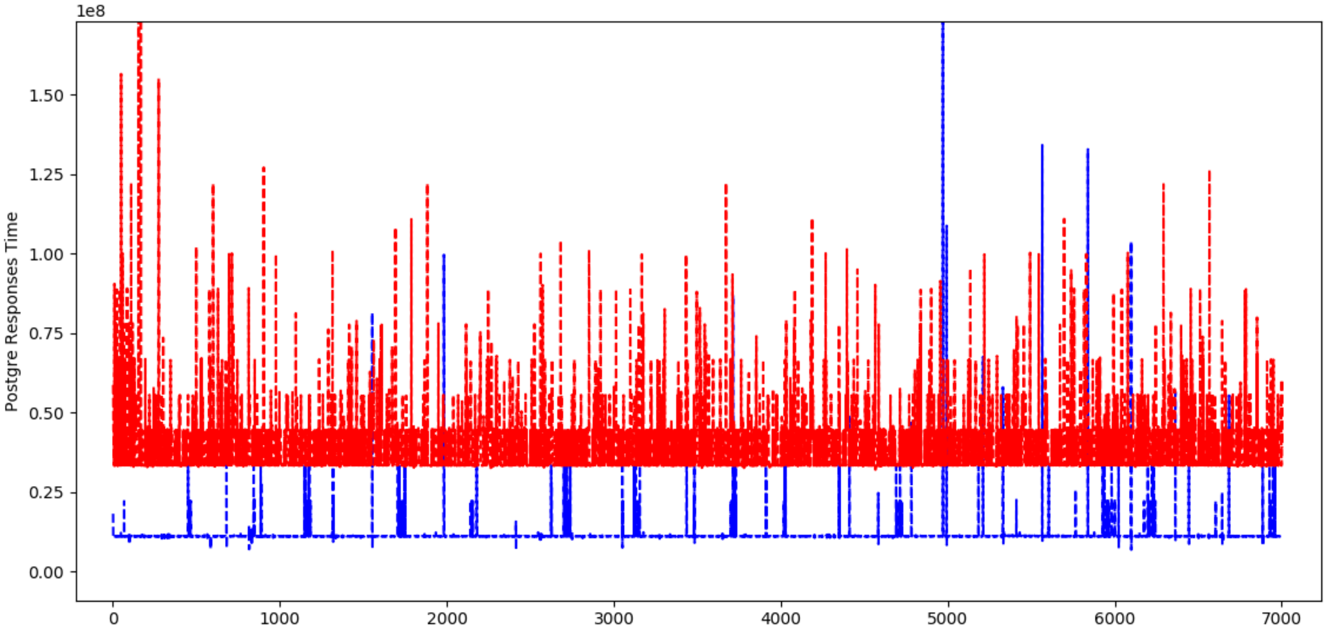
then reload mysql and mysqld (just reload the docker container) to see the effects.

The hard part is to change the value of variables for better achieving our goals.

Results

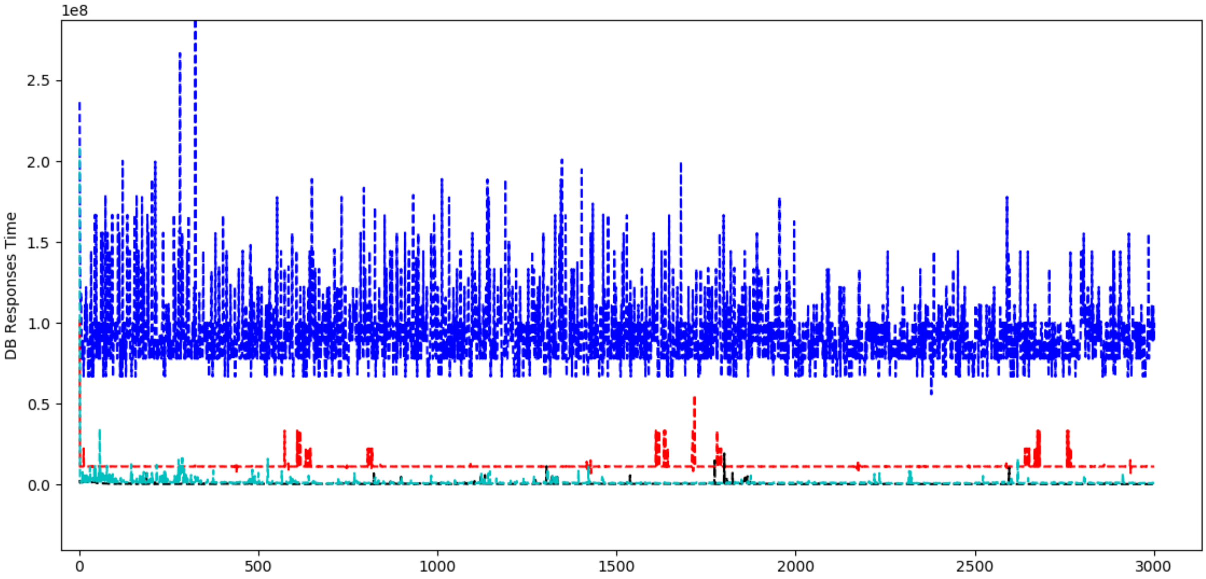
After 24 tunes and changing data these are the resaults:

Write performance



blue: postgre *red: mysql*

OLD Write performance (page 27)



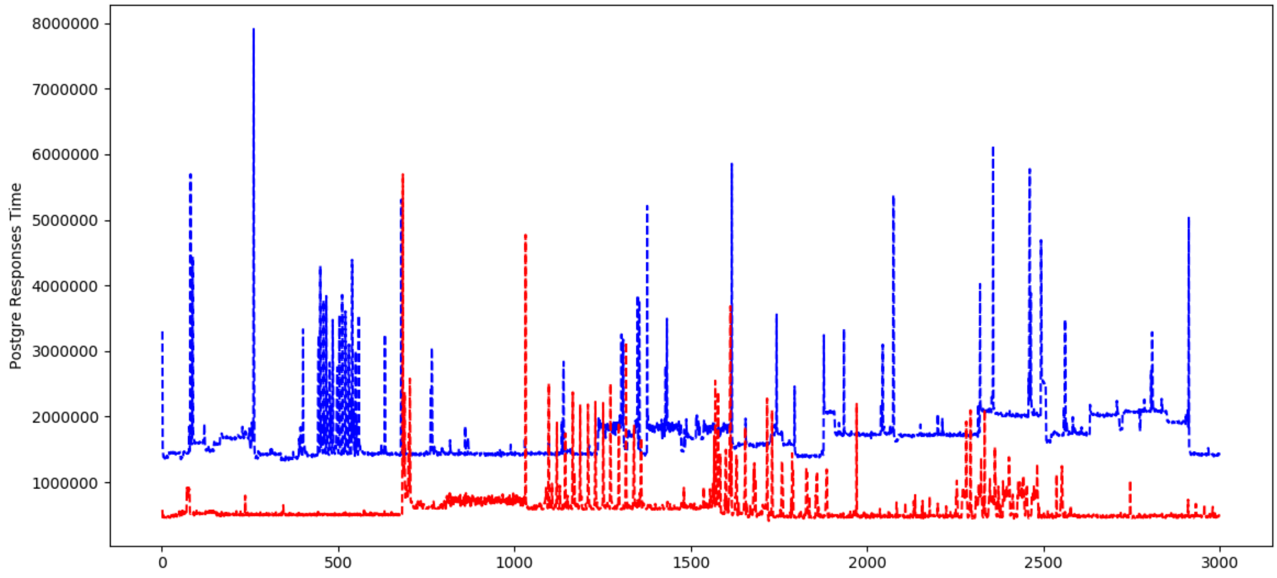
Navid Malek
navidmalekedu@gmail.com
navidmalek.blog.ir

old response value for MySQL is around 0.9

new response value for MySQL is around 0.4

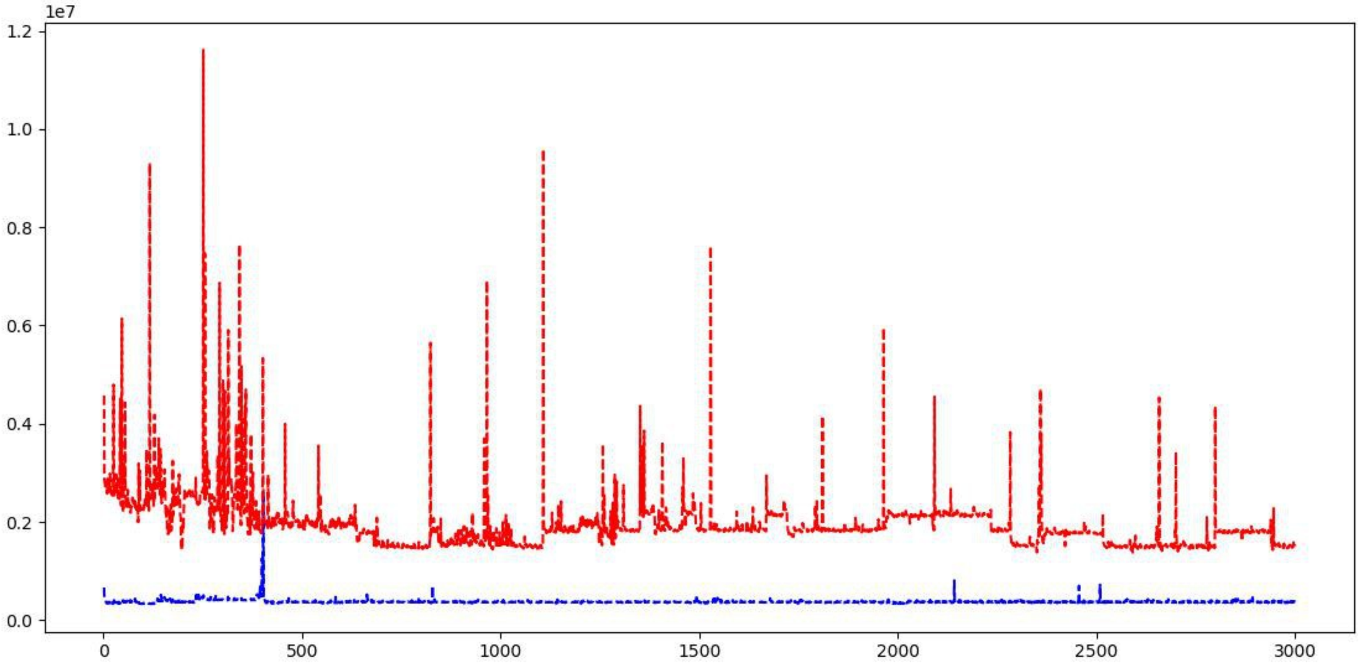
this means around 45 % better performance!

Read performance



blue: postgre *red: mysql*

OLD Read performance (page 28)


1e7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
3000