1、性能压测的时候，随着并发压力的增加，系统响应时间和吞吐量如何变化，为什么?

随着并发压力的增加，在没有达到系统并发处理瓶颈时，系统的响应时间基本没有变化，吞吐量会不断提高；当并发压力超过系统并发处理能力时，请求就会在系统内部排队，系统响应时间就会变长，吞吐量也会维护在一个高位水平，如果这时再加大并发压测，由于系统资源紧张，会导致系统响应时间快速上升，系统吞吐量会下降。

2、用你熟悉的编程语言写一个web性能压测工具，输入参数：URL，请求总次数，并发数。输出参数：平均响应时间，95%响应时间。用这个测试工具以10并发、1000次请求压测www.baidu.com

main.go

```
package main

import (
    "github.com/emirpasic/gods/utils"
    "os"
    "strconv"
    "github.com/emirpasic/gods/lists/arraylist"
)

type Parameter struct {
    url string
    totalRequest int
    concurrent int
}

func main() {
    args := os.Args
    if len(args) != 4 {
        println(len(args))
        panic("系统参数不正确: url totalRequest current")
    }

    parameter, err := parseArgs(args)
    if err != nil {
        panic(err)
    }

    var r chan RequestResult = make(chan RequestResult, 100);

    for i := 0; i < parameter.concurrent; i++ {
        go request(parameter, r);
    }

    count := parameter.concurrent;
    totalTime := (int64(0));
    totalRequest := int64(0);

    s := arraylist.New()
```

```
    for result := range r {

        totalTime += result.duration
        //fmt.Println(result)
        if result.requestTime == "end" {
            count--
        }else {
            s.Add(result.duration)
            totalRequest ++
        }
        if count == 0 {
            break
        }
    }

    i, _ := s.Get(0)
    println("最少用时（毫秒）:", i.(int64))
    println("请求总数:", totalRequest, "  平均响应时间（毫秒）:",
totalTime/totalRequest)
    s.Sort(utils.Int64Comparator)
    idx := float64(totalRequest) * float64(0.95)

    get, _ := s.Get(int(idx))
    println("95% 的请求在:", get.(int64), "毫秒内完成")

}

func parseArgs(args []string) (*Parameter,error) {
    url := args[1]
    tr := args[2]
    cc := args[3]
    println(url, tr, cc)

    atoi, err := strconv.Atoi(tr)
    if err != nil {
        return nil, err
    }

    i, err := strconv.Atoi(cc)
    if err != nil {
        return nil, err
    }

    return &Parameter{url, atoi, i}, nil
}
```

request.go

```
package main

import (
    "github.com/ddliu/go-httpclient"
    "time"
)
```

```
func init() {
    httpclient.Defaults(httpclient.Map{
        httpclient.OPT_USERAGENT:"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36",
        "Accept-Language":"zh-CN,zh"})
}
func request(p *Parameter, c chan RequestResult){
    for i := 0; i < p.totalRequest; i++ {
        var now = time.Now();
        var start = now.UnixNano()
        get, err := httpclient.Get(p.url)
        if err != nil || get.StatusCode != 200{
            c <- RequestResult{isOk: false}
            return
        }
        end := time.Now().UnixNano() - start

        c <- RequestResult{isOk: true,duration: end/1000000}
        //println("end")
    }

    c <- RequestResult{isOk: true,requestTime: "end"}
}
```

common.go

```
package main

type RequestResult struct {
    isOk bool   // 是否请求成功
    duration int64  // 请求与响应时长
    requestTime string  // 请求时间
    responseTime string // 响应时间
}
```