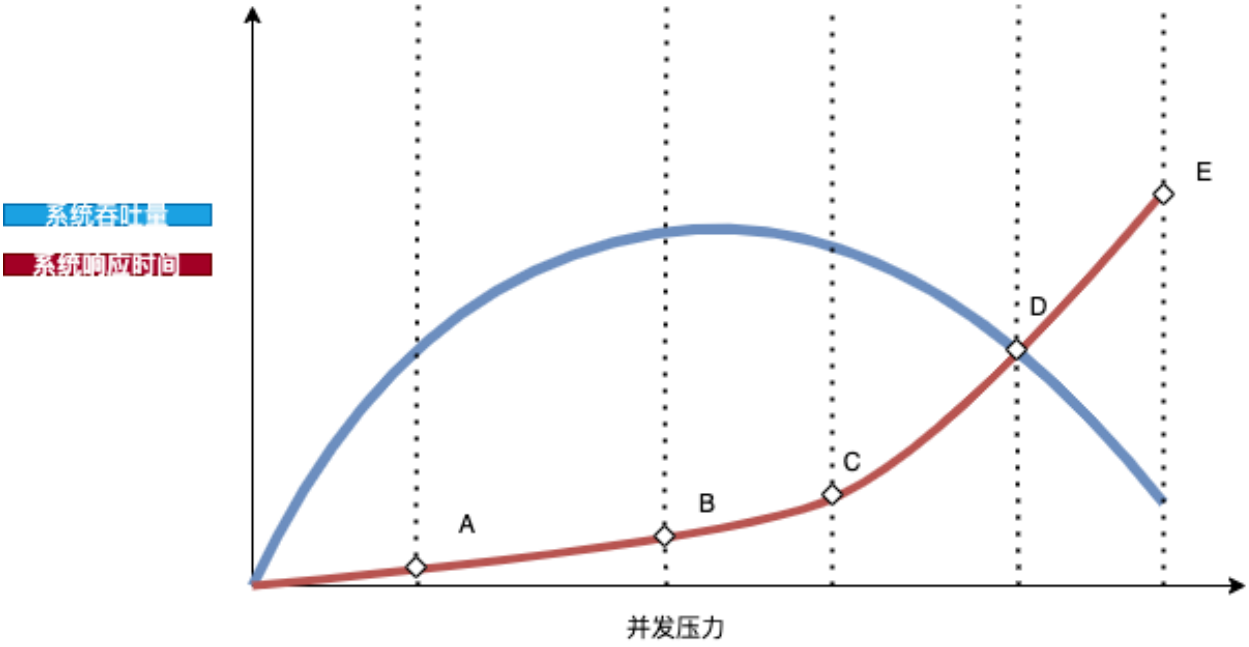
Homework 7

作业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?



如上图所示，横轴表示并发压力，纵轴表示吞吐量和响应时间，且蓝色的线表示系统吞吐量，红色的线表示系统响应时间。

1. 在A点之前，随着并发越来越大，系统的吞吐量逐渐增加，响应时间一开始也会处于较低的状态。
2. B点为业务可以承受的时间点，随着并发越来越大，响应时间开始有些增加，达到了业务可以承受的最大时间点B，这时系统吞吐量仍然有增长的空间。
3. C点为系统最大吞吐量，随着并发压力继续增加，系统达到C点，到达了系统最大吞吐量。
4. E为响应时间超时的时间点，再继续增加并发压力，响应时间接着增加，系统吞吐量可能开始下降或者保持不变(这和系统的具体设计相关)，最后，响应时间过长，达到了超时的程度E。

作业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。

结果：

```
+---------+-------------+-------------+
|  STAT   | 平均响应时间 | 95%响应时间 |
+---------+-------------+-------------+
| Latency | 2262.51 ms  | 4656 ms     |
+---------+-------------+-------------+
```

代码：

```go
package main

import (
  "context"
  "flag"
  "fmt"
  "net/http"
  "os"
  "time"

  "github.com/briandowns/spinner"
  "github.com/glentiki/hdrhistogram"
  "github.com/olekukonko/tablewriter"
  "github.com/ttacon/chalk"
  "golang.org/x/sync/errgroup"
  "golang.org/x/sync/semaphore"
)

type Result struct {
  status  int
  latency int64
}

func main() {
  url := flag.String("url", "", "待测试的url地址. (必填)")
  clients := flag.Int("connections", 10, "并发连接数")
  requestAmount := flag.Int("amount", 1000, "请求总次数")
  flag.Parse()

  if *url == "" {
    flag.PrintDefaults()
    os.Exit(1)
  }

  fmt.Printf("运行%v个测试请求 @ %v", *requestAmount, *url)

  doRequest := func(ctx context.Context) ([]*Result, error) {
    sem := semaphore.NewWeighted(int64(*clients))

    g, ctx := errgroup.WithContext(ctx)

    results := make([]*Result, *requestAmount)

    for i := 0; i < *requestAmount; i++ {
      i := i

      err := sem.Acquire(ctx, 1)
      if err != nil {
        fmt.Printf("Acquire err = %+v\n", err)
```

```
      continue
    }

    fmt.Printf("executing %d\n", i)

    g.Go(func() error {
      defer sem.Release(1)
      startTime := time.Now()
      resp, err := http.Get(*url)
      if err == nil {
        r := &Result{
          status:  resp.StatusCode,
          latency: time.Since(startTime).Milliseconds(),
        }

        results[i] = r
      }

      return err
    })

  }

  if err := g.Wait(); err != nil {
    fmt.Printf("g.Wait() err = %+v\n", err)
    //return nil, err
  }

  return results, nil
}

spin := spinner.New(spinner.CharSets[14], 100*time.Millisecond)
spin.Suffix = "Running loading test...."
spin.Start()

results, err := doRequest(context.Background())
if err != nil {
  fmt.Fprintln(os.Stderr, err)
  return
}

latencies := hdrhistogram.New(1, 10000, 5)
for _, result := range results {
  if result == nil {
    continue
  }

  latencies.RecordValue(int64(result.latency))
}
```

```
  spin.Stop()
  fmt.Println("")
  fmt.Println("")

  shortLatency := tablewriter.NewWriter(os.Stdout)
  shortLatency.SetRowSeparator("-")
  shortLatency.SetHeader([]string{
    "Stat",
    "平均响应时间",
    "95%响应时间",
  })

  shortLatency.SetHeaderColor(
    tablewriter.Colors{tablewriter.Bold, tablewriter.FgCyanColor},
    tablewriter.Colors{tablewriter.Bold, tablewriter.FgCyanColor},
    tablewriter.Colors{tablewriter.Bold, tablewriter.FgCyanColor},
  )

  shortLatency.Append([]string{
    chalk.Bold.TextStyle("Latency"),
    fmt.Sprintf("%.2f ms", latencies.Mean()),
    fmt.Sprintf("%v ms", latencies.ValueAtPercentile(95)),
  })

  shortLatency.Render()
  fmt.Println("")
  fmt.Println("")
}
```