Scalable Web Systems – Fall 2017

Practice Exam

The exam is a 4-hour written exam, at ITU with all aids allowed (books, on-line resources, …). The exam will have a similar form to this practice exam, with 4 thematic parts each contributing to 25% of the grade.

Part 1 – Reflecting on the assignments (25%)

A. Did you use Big Query for your assignments? Your answer should (i) describe BigQuery, (iiI) its pros and cons and (iii) why it is (or it is not) relevant for your assignments?

B. In the first assignment, you were asked to implement a web service that accepts a GET request and returns a JSON array. Describe your design and implementation.

Part 2 – Server-side programming (25%)

A. Describe the process of deploying an application on the Google Cloud Platform

B. Consider several services that are deployed on the Google Cloud Platform and interact with each other. What are the pros and cons of using RPC to support this interaction? Can RPC scale with the number of services?

Part 3 – Topics from the lectures (25%)

A. Google claims that Spanner is a distributed CA store.
   a. What are C and A in the CAP theorem? What is a CA store?
   b. Describe Spanner's architecture
   c. What aspects of the Spanner architecture impact the validity of the claim?

B. Describe RPC and REST. How do they compare?

Part 4 – Go programming (25%)

A. The following is a simple TCP server. It listens to a TCP port and when a socket comes in, it write "Hello" to it and shuts the socket down.

```
package main

import (
      "fmt"
      "log"
      "net"
)

const listenAddr = "localhost:4000"
```

```
func main() {
	l, err := net.Listen("tcp", listenAddr)
	if err != nil {
		log.Fatal(err)
	}
	for {
		c, err := l.Accept()
		if err != nil {
			log.Fatal(err)
		}
		fmt.Fprintln(c, "Hej!")
		c.Close()
	}
}
```

Explain the Go language concepts that allow you to use Fprintln to write to a net.Conn.

B. Below is a slightly modified version of the program.

```
package main

import (
	"io"
	"log"
	"net"
)

const listenAddr = "localhost:4000"

func main() {
	l, err := net.Listen("tcp", listenAddr)
	if err != nil {
		log.Fatal(err)
	}
	for {
		c, err := l.Accept()
		if err != nil {
			log.Fatal(err)
		}
		io.Copy(c, c)
	}
}
```

Explain what this modified program does. How many users 'N' can it handle concurrently?
Can you explain why it can handle only 'N' users concurrently?
How can you fix it using go routines? (hint: you just need to change a single line).

C. Using the above code as a starting point, design a simple chat program where multiple users can connect and are assigned a random partner. They then can chat with each other (i.e., everything user A writes is sent to user B and likewise). Your program should scale with the number of pairs that chat concurrently.
You should use and discuss Go concurrency concepts (goroutines, channels, select) in your design. You are encouraged to write some code together with your explanation.