Module 1 project milestone

Overview

In this first project milestone, you are going to deploy a web server, written in Go, to the cloud, and fetch documents from that web server using a web client, also written in Go. You'll gain experience with provisioning cloud resources such as virtual machines, and will also measure the network performance experienced between your client and the cloud. Finally, you will get experience with graphing quantitative network measurement data.

In particular, using code from the book as a starting point, you're going to create a web client which can fetch documents from the web server. You'll modify the web server part so that it can dynamically generate a document (full of "junk" placeholder text) of a particular length. Your client will request documents of varying lengths, and you'll measure the latency and bandwidth between your client and the server. Using these two tools, you'll answer some questions about the performance you see between your client and the cloud and present that data visually.

Preparation

Read Chapter 1 of the Donovan and Kernighan *The Go Programming Language* book.

After completing Chapter 1 of the book, you're going to begin by setting up your environment to work on this milestone. You can provision an AWS virtual machine and use that to develop your code by following our guide to developing Go on AWS (linked off of module 1 in canvas.ucsd.edu). Alternatively, if you have your own laptop/desktop and want to develop there, you certainly can, however to complete the milestone you will eventually need to deploy your code to AWS.

Following the instructions on page xvi of the preface, make sure you can get a copy of the helloworld program located at *gopl.io/ch1/helloworld* and run it from your local **$HOME/gobook** directory. Read our guide on setting up the Go language tools on your new VM to carry out this task.

Starter code

You're going to base your web client on the `fetchall` program described in Section 1.6 (**`gopl.io/ch1/fetchall`**). Download and build this code, and try it out on some well-known websites (e.g. www.cs.ucsd.edu, www.ucsd.edu, etc).

You're going to base your web server on the `server3` program described in Section 1.7 (`gopl.io/ch1/server3`). When you deploy your code to your VM in the cloud, you will need to change the line

```
log.Fatal(http.ListenAndServer("localhost:8000", nil))
```

to

```
log.Fatal(http.ListenAndServer("0.0.0.0:8000", nil))
```

so that it will accept your requests (we will explain why this change is necessary during the first part of the course). Now test out your web server with a regular web browser (e.g. Firefox, Chrome, Safari, Edge, etc). Make sure you can see the output of the server, including the request headers. Now, from your VM or local machine, use a text-based web client (such as `wget` or `curl`) and make sure that you're able to interact with your server from the command line.

Experiment with adding parameters to the request URL (e.g. `/mypath?q=5000`). Note the fields that become visible. Make sure you can add fields in both the graphical web clients as well as the command-line web clients.

Assignment

You are going to modify your web server so that it supports a new sub-path called `gendata` that takes a parameter `numBytes`. When this path is called your server will return to the client a string of length `numBytes` consisting either of all a single character (e.g. a period) or a pattern of text (e.g. "abcdef...xyzABCDEF...XYZabc…") or random text. For the `gendata` path, do not return the headers or other form parameters, instead just return the string.

Once you're implemented the gendata path, test it with graphical and command-line browsers as well as your `fetchall` program to make sure the sizes returned by `fetchall` are correct and that you're able to time each request.

After you've tested your client and server, you're going to carry out a series of experiments (described below) and collect data from those experiments. Finally, you'll turn in your code and a report of what you found in those experiments.

Hints

Investigate the `log.Printf()` function to log incoming requests--this can help with debugging your server.

To create a string that is `numBytes` characters long, check out the `strings.Repeat()` function.

Experiments

You are going to choose two different AWS regions to run the following sets of experiments. You'll deploy your client to one region and your server to another region (don't choose the same region for both your client and server).

The regions are:

- USA
- Ireland
- South Korea
- Brazil
- India

The four experiments you will run are:

1. **Serial**: Run your fetchall client and request strings of length 1 byte, 1000 bytes, 1,000,000 bytes, and 100,000,000 bytes. For each size, run fetchall three times to collect three data points, and run each test separately (so you'll only be running fetchall with a single URL each time). For the 'serial' experiment you will run fetchall 12 times in total (4 runs of fetchall 3 times each = 12).
2. **Concurrent**: Run your fetchall client and request strings of length 1 byte, 1000 bytes, 1,000,000 bytes, and 100,000,000 bytes. This time request all four sizes in a single call to fetchall (so you'll be running fetchall with four command-line arguments each time). Collect three data points for each experiment. For the 'Concurrent' experiment, you will run fetchall 3 times in total (a single run of fetchall 3 times = 3).
3. **Latency**: Using the ping command-line tool built into Windows, Mac, and Linux, estimate the round-trip time (RTT) and thus one-way latency between your AWS virtual machines (VMs).

Graph

You are going to create a bar graph of the data from your serial and concurrent experiments. For this milestone, simply average together the three runs of each data point (later we will discuss expressing uncertainty of data using error bars). The x-axis should have four groupings of two bars each (the four groupings correspond to the different values of `numBytes` and within each grouping one bar represents the serial experiment and the other bar represents the concurrent experiment). The y-axis should represent the effective bandwidth (`numBytes` divided by the request time).

You can use whatever tool you feel most comfortable with to generate the graph (e.g. Excel, Google Docs, python, Matlab, etc).

Report

Create a single page PDF using whatever tools you prefer (LaTeX, Google drive, Word, etc) that includes your name(s), PID(s), and github username(s) as well as the region you used for your client and the region you used for your server. Include the graph in your report and short (a sentence or two) answers to these questions:

Q1. For the data from the serial experiment, how was the effective bandwidth affected by the size of `numBytes`?

Q2. Using the data from only the **Serial** experiment, estimate the bandwidth between your client and your server. Now incorporate the data from your **Latency** experiment to increase the accuracy of your bandwidth estimate. Describe how data from the **Latency** experiment improves accuracy.

Q3. How did the data from the **Serial** experiment compare from the **Concurrent** experiment? Similar? Dissimilar? Explain these results as best you can.

Q4. After carrying out these experiments, what is something that you learned about performance and networked applications?

Q5. After carrying out these experiments, what is one (or more) unanswered question(s) you still have about network performance?

(optional) Q6. Any other comments/observations?

Submission

```
.
├── readme.txt
├── report.pdf
└── src
    ├── fetchall.go
    ├── .gitkeep
    └── server3.go

1 directory, 5 files
```

Update the readme with your names, PIDs, and github info for our records. Replace the report with your report, and include your source code files in the `src/` subdirectory.

Starter code

An invitation to the starter code is available here: https://classroom.github.com/g/S1IeHE0g