University of California Merced
School of Engineering

CSE 168 Distributed Software Systems
Lab #1: RPC

Author
Angelo Fatali

Instructor
Xiaoyi Lu

TA
Adam Weingram

Section
04L T 7:30-10:20PM

Date
09/20/2022

What was the round-trip time (RTT) or latency of a message between one gRPC client and one gRPC server?

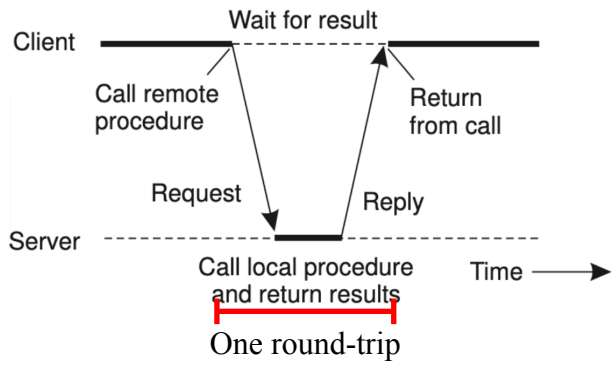

Figure 1 - Synchronous RPC model
Courtesy: Distributed Systems: Principles and Paradigms, 2nd Edition, Andrew S. Tanenbaum and Maarten van Steen

To replicate a basic synchronous RPC shown in Figure 1 and get the round-trip time I needed to understand what is actually being done. Firstly I needed to turn the 'greeter_server' on; I did this by running my 'grpc_container' terminal and running the command: "`./greeter_server`"

Once this is running, I opened a new terminal under my docker container and ran the command: "`./greeter_client`" This gave me the hello world response and confirmed the client and server were working.

In order calculate the round-trip time I did a couple solutions. My first solution was through a bash file named 'hello.sh'. Using the command "`nano hello.sh`" I was able to edit my bash script inside the terminal. In the bash file, I had the main function call the 'greeter_client' and outside had "`time main`" which returned a round-trip time (latency) of about "`user 0.004s`". This also returned different results called real and sys; and I didn't understand what they meant so I wanted to figure out a different solution to find the RTT.

My second solution was to edit the 'greeter_client.cc' file. Instead of using nano I copied this file to VS Code and edited on there. I changed the SayHello function to void in order to not require a string return type since I simply just wanted to print the latency. I used a library called 'chrono' which allowed me to get the current time at any process in the code using the 'high_resolution_click'. To utilize this I grabbed a timestamp 'start_time', called the actual RPC the 'status', then grabbed a timestamp of 'end_time'. In order to get the RTT, we need to find the difference of end-time and start-time. I did this by using chronos duration_cast in milliseconds: "`std::chrono::duration_cast<std::chrono::milliseconds>(end_time - start_time).count();`". Finally, after this is calculated I printed out the latency and got around **3-4 milliseconds** of latency each run.

What was the throughput (i.e., requests/sec or messages/sec) of one gRPC sever when one gRPC client is running and when two gRPC clients are running?

The throughput is the RPC calls per second. My calculations were done by messages/ milliseconds. I was originally going to do this through a bash script but I decided not to because I'm not very familiar with creating bash scripts and I would have needed to store each latency per run and I don't know how inputs and variables work in bash scripts. So instead I went with editing the 'greeter_server.cc' file to be my solution.

I used the same process as calculating the latency (see above), meaning the SayHello is still void and I'm still calling once to get the latency. To get the throughput I'm going to have to call the RPC multiple times and collect the latency of each call then divide the messages by the sum of the latency.

I created a for loop that will grab the latency of each RPC call. The for loop is set to run 'messages_count' amount of times which for this test I set to 1000. I changed the original 'end_time - start_time' to be stored in a global float variable called 'result'. This allows me to calculate each latency through each iteration of the for loop and get the sum of each latency's result by doing "`result = result + std::chrono::duration_cast<std::chrono::milliseconds>(end_time - start_time).count();`"

Now that I have the sum of each latency of each RPC call; I'm able to calculate the throughput by taking my 'messages_count' (my benchmark is set to a value of 1000) and add one to it since I did the original latency run previously. This gives me a final calculation of '(messages_count+1)/result'. I then print this with the original latency for the first RPC call and now when recompiling using "`make -j2`" I get the output:

Looking at Figure 2, when I call the 'greeter_client' I get my original latency of the first RPC call of about 5 milliseconds. I also get the throughput of 1001 calls (including the original) divided by the sum of the latency per call; resulting in a throughput of **19.6275 msgs/ms**

```
# ./greeter_client
Latency: 5 milliseconds
Throughput: 19.6275Messages/milliseconds
```

Figure 2 - Latency and throughput output in my terminal, where latency is the first RPC call and throughput is 1000 calls divided by the sum of the latency per call

```
# ./greeter_client & ./greeter_client
Latency: 4 milliseconds
Latency: 5 milliseconds
Throughput: 20.02Messages/milliseconds
Throughput: 22.75Messages/milliseconds
```

Figure 3 - Latency and throughput output in my terminal, where latency is the same as Figure 2 and throughput is the same but with 2 clients called.

In order to do 2 clients and 1 server; we can simply do the command "`./greeter_client & ./greeter_client`". This gives us the throughput of each client, add them both to get the throughput of both. My result is in Figure 3 of about **20msg/ms** for each. Meaning using two clients doubled the original single client, giving a total of **42.77msg/ms**.

All relevant source code including my bash script and edited greeter_client.cc will be in a zip file named: 'Lab1.zip'