OSO work sample—MAX

Lars Wirzenius

2021-05-29 22:26

Contents

1 Work sample for OSO

This is the work sample for my job application for a developer position for OSO.

This document explains the work I've done and verifies that the code I wrote works together with the client.

1.1 Re-statement of problem

To clarify the problem for myself, I am re-stating it. This will also work to make sure I've understood it in the intended way when we discuss this.

The goal is to write a server, which communicates with a client using messages over HTTP. The client has a list of integers, and the asks the server to figure out what is the largest integer in the list. The crux is that the client does not send the server the whole list, but only small messages and the server needs to, effectively, ask the client to do pairwise comparisons of integers.

The possible messages sent by the client and the server are listed below, as examples. Communication is started by client by a "start computation" message. Server responds with a suitable message, which causes the client to make a new HTTP request with the response to the server's message. This continues until the server sends a message with result it has come up with.

1.2 Client request: start computation

Client starts the protocol by asking the server to compute something about a list of numbers the client holds. It tells the server what computation is requested, and how long the list is.

The supported computations are, for now, `compute_max` (find index of the largest integer) and `compute_min` (similar, but smallest integer).

```
{
    "type": "compute_max",
    "length": 2
}
```

1.3 Server response: request computation

Server ask the client to report the result of a less-than comparison operation between arbitrary list items.

```
{
    "type": "compare",
    "left": 0,
    "right": 1,
    "request_id": 7
}
```

1.4 Client request: computation result

The client reports the result of a comparison. The server should respond by another comparison request, or the result of the computation.

```
{
    "type": "comp_result",
```

```
    "answer": true,
    "request_id": 7
}
```

1.5 Server response: result

```
{
    "type": "done",
    "result": 2
}
```

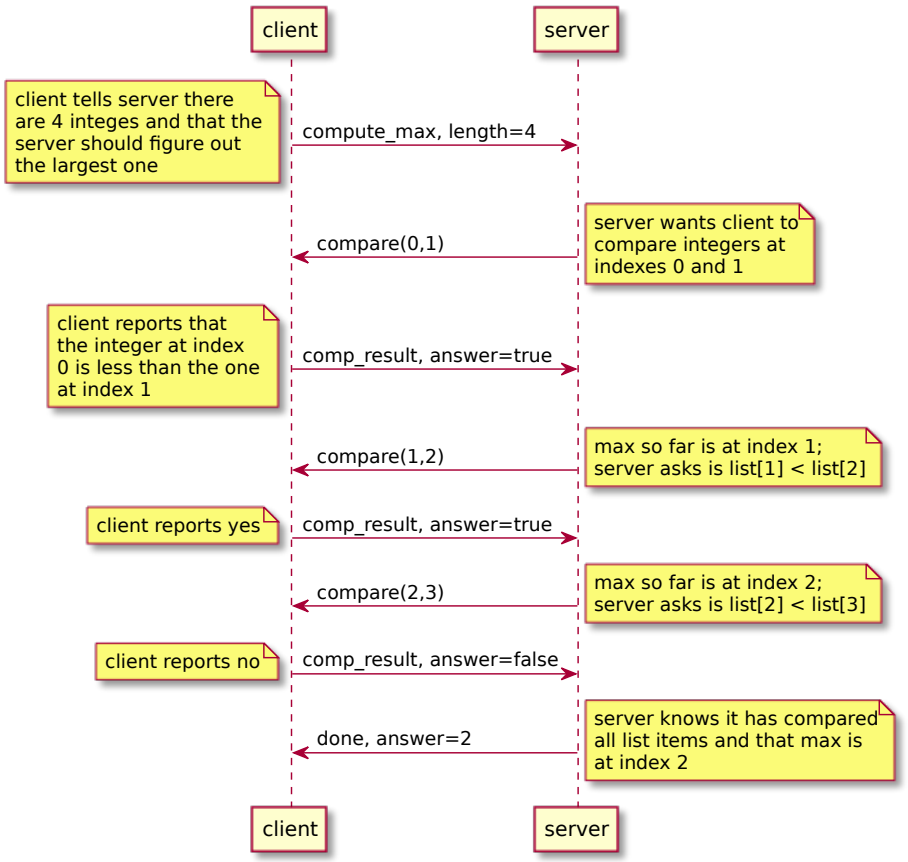
2 Example

In this example, assume the client has the following list:

5, 6, 7, 5

Note that the list is not in order and numbers aren't unique. The sequence diagram below shows the messages the go between the client and the server.

For simplicity, request ids are not shown.



This is very simple computation. Given the server can't assume the list is ordered, it has to compare all list elements to the largest one it has found so far.

3 Assumptions

- The server will abort if the client doesn't use the request id from the latest server message. That is, the client and server do not need to handle multiple outstanding comparison requests.
- The server does not need to handle the case of the client having an empty list of integers, because there messages as given do no indicate a way to signal a result of "no result".
- The list in the client doesn't change during the computation: items stay in the same order, and are not added, deleted, or changed.
- The client responds truthfully.

4 Remarks

- Given the list the client has is unordered, O(n) comparisons is the best we can do. If the client can make some guarantees, faster algorithms are possible. If list can be ordered, a binary search would be optimal.

5 Acceptance criteria

5.1 Find max of a list of one

This scenario verifies that the server finds the maximum integer in a list of one.

given server
when I run **max-client.py 1**
then answer is **1**

5.2 Find max of a list of two

These scenarios verify that the server finds the maximum integer in a list of two. There is a separate scenario for every possible list of two elements.

given server
when I run **max-client.py 5 5**
then answer is **5**
when I run **max-client.py 5 6**
then answer is **6**
when I run **max-client.py 6 5**
then answer is **6**

5.3 Find max of a list of three

These scenarios verify that the server finds the maximum integer in a list of two. There is a separate scenario for every possible list of two elements.

given server
when I run **max-client.py 5 5 5**
then answer is **5**
when I run **max-client.py 5 5 6**
then answer is **6**
when I run **max-client.py 5 6 5**
then answer is **6**
when I run **max-client.py 6 5 5**
then answer is **6**
when I run **max-client.py 5 6 7**
then answer is **7**
when I run **max-client.py 5 7 6**
then answer is **7**

when I run **max-client.py 6 5 7**
then answer is **7**
when I run **max-client.py 6 7 5**
then answer is **7**
when I run **max-client.py 7 5 6**
then answer is **7**
when I run **max-client.py 7 6 5**
then answer is **7**

6 What I did

6.1 The client

I modified slightly the `max-client.py` file:

- it is now an executable Python script, and formatted with Black
- the user can invoke it with a list of numbers, and tell it whether to ask the server to find the min or the max number

I made these changes so that I could use it when verifying the acceptance criteria defined in this document. The original code tested only two cases, which I found to be inadequate for my purposes.

The Python code is not entirely to current Python best practices. For example, it doesn't use type annotations. I have not started using those yet: at work I've only used old versions of Python that don't support type annotations, and in my free time, I don't write anything of significant size in Python anymore.

6.2 The server

The server is in `server.py`, and is a Python program using `bottle.py`. I chose Python, because for something small and simple like this it's easy. I chose `bottle.py` because it's familiar for me.

I could have chosen Rust, and probably the `warp` crate for the HTTP API, but it would have required much more implementation work, and probably more than is warranted for this exercise.

The code is a little simplistic in that it doesn't do much in terms of error handling, logging, or such. At the same time it's overly complicated, because I wanted to make sure it allows for more than just the "max" algorithm. "min" is implemented, and the same structure should be usable for, say, finding the second largest element. More interesting algorithms would require changes to the messages: if, for example, one wanted the server to find out if the list is ordered, the "done" message would need to be able to express the result.

6.3 The testing

I have used the Subplot[1] program to verify that my server works. Subplot documents the acceptance criteria and how they are verified. That is this document. The section Acceptance criteria[2] documents the acceptance criteria using *scenarios* consisting of given/when/then steps.

Subplot produces two typeset documents (one in HTML, one in PDF), and a self-standing test program, which can be run to verify the system under test fulfills the acceptance criteria. To avoid requiring you to have Subplot installed, the test program is included in the git repository as `test.py`. You can run it like this:

```
$ python3 test.py --log test.log
srcdir /home/liw/pers/oso/work-sample
datadir /tmp/tmpcom39rm7
scenario: Find max of a list of one
  step: given server
  step: when I run max-client.py 1
  step: then answer is 1
  cleanup: given server
scenario: Find max of a list of two
  step: given server
  step: when I run max-client.py 5 5
  step: then answer is 5
  step: when I run max-client.py 5 6
  step: then answer is 6
  step: when I run max-client.py 6 5
  step: then answer is 6
  cleanup: given server
scenario: Find max of a list of three
  step: given server
  step: when I run max-client.py 5 5 5
  step: then answer is 5
  step: when I run max-client.py 5 5 6
  step: then answer is 6
  step: when I run max-client.py 5 6 5
  step: then answer is 6
  step: when I run max-client.py 6 5 5
  step: then answer is 6
  step: when I run max-client.py 5 6 7
  step: then answer is 7
  step: when I run max-client.py 5 7 6
  step: then answer is 7
  step: when I run max-client.py 6 5 7
```

[1] `https://subplot.liw.fi/`

[2] `#acceptance`

```
  step: then answer is 7
  step: when I run max-client.py 6 7 5
  step: then answer is 7
  step: when I run max-client.py 7 5 6
  step: then answer is 7
  step: when I run max-client.py 7 6 5
  step: then answer is 7
  cleanup: given server
OK, all scenarios finished successfully
$
```

I hope that is satisfactory.