CS4103 P1

160004864

13 March 2020

1 Introduction

The aim of this practical was to design and implement a simple distributed system with the help of existing communication middleware. To achieve this, I've used the Java Remote Method Invocation (RMI), a Java API which allows for remote communication between programs written in the Java programming language[1].

2 Functionality Implemented

- Two nodes, implemented as two different remote objects `Node1` and `Node2`, that are able to respond to requests for a task as described in the specification.
- A client that is able to communicate with a scheduler which, like the nodes, is also implemented as a remote object.
- The client gathers the resource requirements from the user and places requests to the scheduler.
- The scheduler, upon receiving a request from the client, checks which node should be invoked and chooses the one with the least amount of load.
- The resource usage of the node is updated, and once it's invoked the client is informed of the task's execution.
- **Load Delegation** (Extension): One of the nodes, `Node2` will delegate a task the scheduler forwarded to it to `Node1`, if the number of resources the task requires is a prime number. Such a simple condition is used to demonstrate the load delegation functionality, rather than introducing complicated decision making mechanisms.

3 Design

Note: Due to the word count limit imposed by the specification, the reader is encouraged to read the well-documented source code for specifics of the implementation.

3.1 Nodes

These are the nodes of our simple distributed system that can perform a certain task. The two nodes are implemented in the files `Node1.java` and `Node2.java` and they both implement the remote interface found in `NodeInterface.java` which declares the set of remote methods the nodes make available. These methods include the task invocation method, `invokeTask` as well as a method `queryResources` which returns the current load of the node, a value which is held as an attribute. The two nodes are very similar, with the difference that `Node2` will delegate a task the scheduler forwarded to it to `Node1`, given that the number of resources the task requires is a prime number.

3.2 Scheduler

The scheduler node was implemented to achieve transparency with regards to the node load balancing. By introducing a scheduler, the client is not required to have any load balancing policy, as all of that is moved onto the scheduler. It is implemented in the `SchedulerImplementation.java` file, and it implements the remote interface found in `SchedulerInterface.java` which defines the scheduler's remote methods. Upon instantiation, the scheduler will lookup the two nodes in the registry, query them on their current load and then depending on those values, it will forward the task to the node with the least amount of load, a piece of functionality provided by the `forwardTask` method. In the case where both nodes have equal amounts of load, the task will be forwarded to `Node1`. Note that the load delegation functionality is done at the node level rather than the scheduler level. The scheduler will forward the task to the node with the least current load, no matter whether the number of resources required for the task is prime or not.

3.3 Server

The RMI server, implemented in the `Server.java` file is responsible for creating instances of the remote objects described above, export them, and (*in the darkness*) bind them to a name in a Java RMI registry. The names are pretty self explanatory and they are `Node1, Node2` and `Scheduler`.

3.4 Client

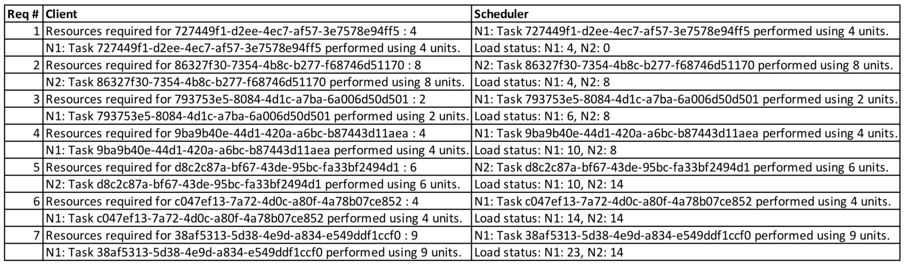
The client, implemented in the `Client.java` file, first gets the stub for the registry on the server's host, and then obtains the stub of the `Scheduler` remote object from it. At this point, the client generates a unique id for the next task request and it requires the user to enter the number of resources that task will require. Finally, the task is forwarded to the scheduler by using its `forwardTask` remote method and the client is informed of the task's execution.

4 Examples and Testing

In this section, we will be providing examples demonstrating the functionality of the system, as well as its correctness.

4.1 Without Load Delegation

In this section we will show the functionality and correctness of the system without bringing in the load delegation feature into play.



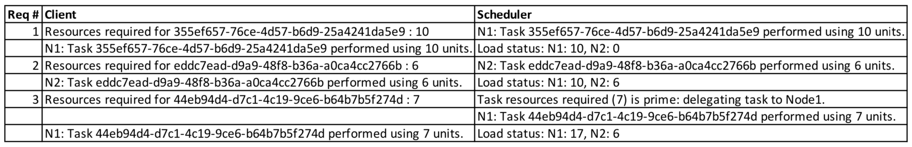
Req #	Client	Scheduler
1	Resources required for 727449f1-d2ee-4ec7-af57-3e7578e94ff5 : 4	N1: Task 727449f1-d2ee-4ec7-af57-3e7578e94ff5 performed using 4 units.
	N1: Task 727449f1-d2ee-4ec7-af57-3e7578e94ff5 performed using 4 units.	Load status: N1: 4, N2: 0
2	Resources required for 86327f30-7354-4b8c-b277-f68746d51170 : 8	N2: Task 86327f30-7354-4b8c-b277-f68746d51170 performed using 8 units.
	N2: Task 86327f30-7354-4b8c-b277-f68746d51170 performed using 8 units.	Load status: N1: 4, N2: 8
3	Resources required for 793753e5-8084-4d1c-a7ba-6a006d50d501 : 2	N1: Task 793753e5-8084-4d1c-a7ba-6a006d50d501 performed using 2 units.
	N1: Task 793753e5-8084-4d1c-a7ba-6a006d50d501 performed using 2 units.	Load status: N1: 6, N2: 8
4	Resources required for 9ba9b40e-44d1-420a-a6bc-b87443d11aea : 4	N1: Task 9ba9b40e-44d1-420a-a6bc-b87443d11aea performed using 4 units.
	N1: Task 9ba9b40e-44d1-420a-a6bc-b87443d11aea performed using 4 units.	Load status: N1: 10, N2: 8
5	Resources required for d8c2c87a-bf67-43de-95bc-fa33bf2494d1 : 6	N2: Task d8c2c87a-bf67-43de-95bc-fa33bf2494d1 performed using 6 units.
	N2: Task d8c2c87a-bf67-43de-95bc-fa33bf2494d1 performed using 6 units.	Load status: N1: 10, N2: 14
6	Resources required for c047ef13-7a72-4d0c-a80f-4a78b07ce852 : 4	N1: Task c047ef13-7a72-4d0c-a80f-4a78b07ce852 performed using 4 units.
	N1: Task c047ef13-7a72-4d0c-a80f-4a78b07ce852 performed using 4 units.	Load status: N1: 14, N2: 14
7	Resources required for 38af5313-5d38-4e9d-a834-e549ddf1ccf0 : 9	N1: Task 38af5313-5d38-4e9d-a834-e549ddf1ccf0 performed using 9 units.
	N1: Task 38af5313-5d38-4e9d-a834-e549ddf1ccf0 performed using 9 units.	Load status: N1: 23, N2: 14

Figure 1: Output of the client and the scheduler after a series of task requests

As we can see, the system performs as required by the specification. The scheduler correctly forwards the task to the node with the least number of resources being used (see 'Load Status' output) and the client is correctly informed about which node performed the task. Furthermore, when the loads of both nodes are equal, the scheduler correctly forwards the task to `Node1` (see Req #7). Screenshots of the terminal with the above outputs have been included in the submission, in the files `client_noprime.png` and `scheduler_noprime.png` files in the `Images` directory.

4.2 With Load Delegation

In this section we will show the functionality and correctness of the load delegation feature.



Req #	Client	Scheduler
1	Resources required for 355ef657-76ce-4d57-b6d9-25a4241da5e9 : 10	N1: Task 355ef657-76ce-4d57-b6d9-25a4241da5e9 performed using 10 units.
	N1: Task 355ef657-76ce-4d57-b6d9-25a4241da5e9 performed using 10 units.	Load status: N1: 10, N2: 0
2	Resources required for eddc7ead-d9a9-48f8-b36a-a0ca4cc2766b : 6	N2: Task eddc7ead-d9a9-48f8-b36a-a0ca4cc2766b performed using 6 units.
	N2: Task eddc7ead-d9a9-48f8-b36a-a0ca4cc2766b performed using 6 units.	Load status: N1: 10, N2: 6
3	Resources required for 44eb94d4-d7c1-4c19-9ce6-b64b7b5f274d : 7	Task resources required (7) is prime: delegating task to Node1.
		N1: Task 44eb94d4-d7c1-4c19-9ce6-b64b7b5f274d performed using 7 units.
	N1: Task 44eb94d4-d7c1-4c19-9ce6-b64b7b5f274d performed using 7 units.	Load status: N1: 17, N2: 6

Figure 2: Demonstrating load delegation

As we can see, before the third request is placed, `Node2` has a smaller amount of load compared to `Node1`. Upon the client placing a request for the third task, the scheduler forwards it as normal to `Node2`, but since the number of resources this task requires is 7, and 7 is a prime number, `Node2` will delegate this task to `Node1`. Screenshots of the terminal with the above outputs have been included in the submission, in the files `client_load_delegation.png` and `scheduler_load_delegation.png` files in the `Images` directory.

5 Evaluation

Overall, I'm very pleased with the functionality of the system I've created, as it satisfies all the requirements set out by the specification as well as the load delegation requirement which was a suggested extension. The system runs smoothly and correctly, and its architecture is clear and simple, something which would make scaling it fairly straightforward. I found the Java RMI API fairly intuitive to use and I managed to get a good grasp of its concepts due to the fact that it's very well documented and the fact that there are also a lot of tutorials available online. With that being said, the Java RMI can only be used with Java, whereas other communication middleware such as ZeroMQ can be used with several different programming languages and thus offers developers a greater deal of choice. Furthermore, whilst carrying out some personal research on Java RMI, I came across several articles describing different security exploits as well as testimonies explaining how it's necessary to monitor security a lot more closely while using it.

6 Running

For running instructions, refer to the `README.md` file included in the submission.

7 Conclusion

In conclusion, this assignment was very successful in achieving its learning outcomes as it enabled me to implement my first distributed system, gave me experience with using the Java RMI API and it also consolidated my knowledge of distributed system communication.

References

[1] Retrieved 12 March 2020, from https://docs.oracle.com/javase/tutorial/rmi/index.html