Distributed System: Lab 0
Standalone Database

Petru Eles and Ivan Ukhov
petru.eles@liu.se and ivan.ukhov@liu.se

December 20, 2015

1 Introduction to the Programming Project

Welcome to the course! Over six labs, from zero to five, you will learn and put into practice the key components of a distributed system [1]. To this end, you will implement your own distributed system, namely, a distributed database of short messages known as fortunes [2].

The final configuration of the system that you will obtain by the end of the fifth lab is depicted in Figure 1. This system is composed of a number of servers/peers (SP) and a number of clients (C). The servers maintain copies of the database (DB) and ensure that these copies are kept consistent across all the peers using a distributed locking mechanism (L). The clients interact with this peer-to-peer network of servers in order to perform various operations on the data. The participants of the network discover each other by virtue of a so-called name service (NS); for clarity, these communications are not shown.

In each lab, you will make a step towards the outlined goal by focusing on a particular component of the final system:

0. Standalone Database — you will build a non-distributed database;
1. Client-Server Database — you will make your first attempt to distribute the database by considering the client-server model;
2. Middleware: Object Request Brokers — you will learn the importance of name services and implement a mechanism for interacting with them;
3. Middleware: Peer-to-Peer Communications — you will develop an algorithm for establishing and maintaining communications between peers who can arbitrary join and leave the system;
4. Middleware: Distributed Locks — you will implement mechanisms of distributed mutual exclusion to protect the data from concurrent operations;
5. Client-Server Database with Replicas — you will combine all the above components to produce a distributed database with data replication to ensure the efficiency and robustness of the overall system.

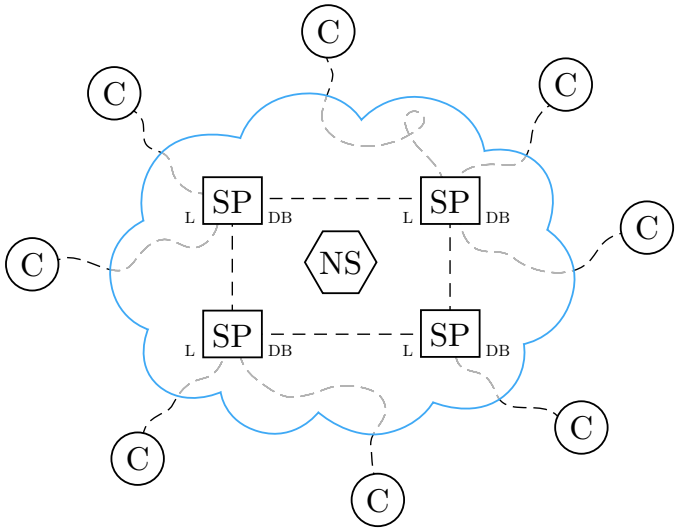

Figure 1: A distributed database with a number of servers/peers (SP) and a number of clients (C). Each server maintains a copy of the database (DB) protected by a distributed lock (L). The discovery process is facilitated by virtue of a name service (NS).

For each lab, you will be given a skeleton of the code that corresponds to a particular part of the system, and your task will be to understand what the code does and to complete the implementation. The programming language that we shall be using throughout the course is `Python` [3], more precisely, the third version of `Python` [4]. Even if you are not familiar with `Python`, the language is straightforward to learn, and the source code, provided to you, with a little bit of reading [5, 6] will be enough to readily get started and to successfully finish this programming project. The above-mentioned skeletons for the labs and the corresponding instructions (just like the one below for the zeroth lab) are provided to you on the web page [7] of the course.

2 Introduction to the Lab

The goal of the zeroth lab is to introduce you to `Python`. Here we consider a non-distributed system that consists of a number of standalone machines, each of which acts as a server and as a single possible client of this server at the same time. The scenario is depicted in Figure 2. The machines do not interact with each other and perform all the operations on their own copies of the data without any synchronization. In what follows, we shall refer to such a standalone machine with the server/client functionality as simply `Client`.

3 Data Format

In all the labs, the list of fortunes, i.e., the actual data in our database, will be stored in a text file conventionally called `fortune.db`. Such a file will be pro-

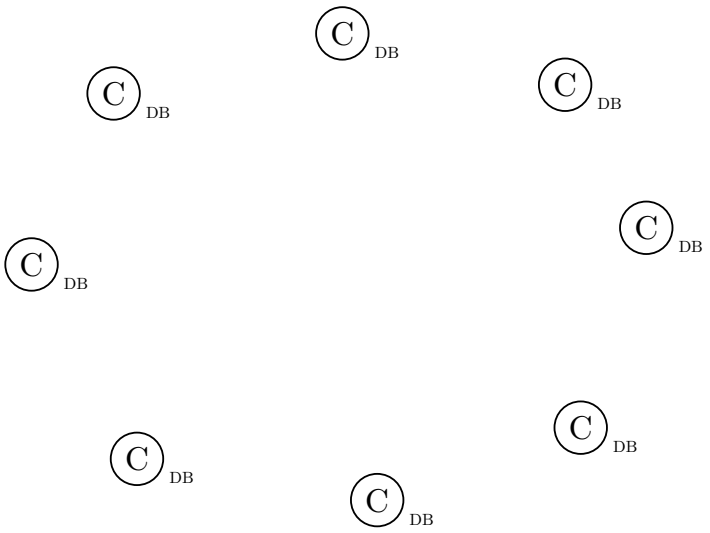

Figure 2: A number of standalone clients (C) with their own databases (DB).

vided to you along with the source code; however, you are free to come up with your own lists of fortunes. The format of `fortune.db` is rather straightforward: each message ends with a new line containing a single percent sign (`%`). For example, the following text contains two fortunes:

```
Everything should be made as simple as possible, but not simpler.
    -- Albert Einstein
%
According to the latest official figures, 43% of all statistics
are totally worthless.
%
```

Note that a message can have several lines, and there can be percent signs inside of messages, which should be property taken into account when you start writing your code for the interaction with the database.

4 Your Task

4.1 Preparation

Clone the repository of the programming project or download the latest snapshot of the repository as a single zip archive [8]. The files relevant to this lab are listed below; note that the paths are given with respect to the `src` directory of the repository. You should read and understand them.

- `lab0/client.py` — the main file of the application that each machine of our non-distributed system is running (no changes are needed);
- `lab0/dbs/fortune.db` — a list of fortunes (a thorough read and a deep understanding are not required here);

- `modules/Server/database.py` — a module with the `Database` class needed for `client.py` (should be modified).

In order to run the application, open a terminal window, change the current directory to `lab0`, and execute the following command:[1]

```
$ ./client.py -i
```

Following the menu of the program, you can try to read a random fortune from the database or to compose a new one:

```
Choose one of the following commands:
    r             ::  read a random fortune from the database,
    w <FORTUNE>   ::  write a new fortune into the database,
    h             ::  print this menu,
    q             ::  exit the application.

Command> w Take it easy
Command> r
None
```

As you can see, the issued commands do not work since the code is incomplete. You can also open `fortune.db` and observe that your new fortune is not there.

4.2 Implementation

Your task is to fix the problem outlined in Section 4.1. Specifically, you should complete the implementation of the following three function in `database.py`:

- `__init__` — initializes the database by parsing the content of `fortune.db` and storing all found messages in an array (an instance variable);
- `read` — returns a randomly chosen fortunes from the array (the `r` command in the application menu);
- `write` — appends a new fortune to the array and updated `fortunes.db` (the `w` command in the application menu).

Recall the note at the end of Section 3.

5 Conclusion

Having completed the lab, you can now perform the reading and writing operations on the database. However, if you run several instances of the application on different machines (see Figure 2) or in different terminal windows on the same machine and try to add a new fortune in one of them, the other clients will not see the change. The reason is that the clients do not collaborate with each other and, therefore, do not try to keep their copies of the data consistent. In the next lab, we will mitigate this problem.

[1] The dollar sign (`$`) will be used to denote a command prompt, and you should not type this character in your terminal. Also, an alternative way to run a `Python` script is to call `Python` explicitly. For example, you can try running `python3 ./client.py -i`. Keep in mind that we use the third version of `Python`.

References

[1] http://www.ida.liu.se/~TDDD25/lectures/lect1.frm.pdf.

[2] http://en.wikipedia.org/wiki/Fortune_(Unix).

[3] http://en.wikipedia.org/wiki/Python_(programming_language).

[4] http://docs.python.org/3/.

[5] http://docs.python.org/3/tutorial/index.html.

[6] http://docs.python.org/3/library/index.html.

[7] http://www.ida.liu.se/~TDDD25/labs/index.en.shtml.

[8] https://gitlab.ida.liu.se/tddd25/labs.