Distributed Algorithms 60009

Coursework - Multi-Paxos

01 The aim of the coursework is to implement and evaluate a simple replicated banking service that uses the version of Multi-Paxos described in the paper: ***Paxos Made Moderately Complex* by** Robbert van Renesse and Deniz Altınbüken. ACM Computing Surveys. Vol. 47. No. 3. February 2015.

02 The DOI for the paper is https://doi.org/10.1145/2673577 The paper can be downloaded from within the College network. There is also a web version of the paper at http://paxos.systems. The paper presents the algorithm in pseudo-code. The appendix of the paper and the authors' website also includes a Python version.

03 For this coursework you'll need to write an Elixir implementation of the algorithm. Various modules will be provided to get you started.

04 You'll need to submit your code and a report.

05 You can work and submit the coursework either individually or jointly with one classmate. It is recommended that you start on the coursework early – the coursework is not easy.

06 Use Piazza if you have questions about the coursework or general questions. **Do not post your solutions or share them with others.** Email me directly if you have a specific question about your solution.

07 If there are any corrections or clarifications, the most up-to-date version of the coursework will be provided on my course webpage**.**

08 **_____ The deadline for submission is Thursday 18th February 2021 _____**

Part 1 – System Structure

09 A directory with a Makefile and various Elixir modules can be downloaded from:

```
http://www.doc.ic.ac.uk/~nd/dac_src/multipaxos.tgz
```

Familiarise yourself with the supplied files, in particular:

- `Makefile` has commands for compiling and running a system. It creates 5 `Server` nodes and 5 `Client` nodes plus the top-level `Multipaxos` node. You can increase or decrease the number of nodes if you wish by changing the `Makefile`.
- `multipaxos.ex`, `server.ex`, `client.ex`, `database.ex`, `monitor.ex`, `configuration.ex`, `debug.ex` and `util.ex` are my implementations of various modules - to help get you started. Work out what they do. You can change these if you wish, or write your own.
- `Monitor` checks that each server database is executing the same sequence of requests. It periodically outputs pairs of lines like:

```
time = 2000  updates done = [{1, 657}, {2, 657}, {3, 657}]
time = 2000 requests seen = [{1, 218}, {2, 220}, {3, 219}]
```

  Here, after 2 seconds, servers 1, 2 and 3 have each performed 657 database updates. Server 1 received 218 client requests, while servers 2 and 3 received 220 and 219 requests respectively. Change `Monitor`, `Debug` and other modules to produce better checks or more informative diagnostics.
- `Util.node_init` (called from `Multipaxos.start`) initialises various parameters for a run, some are taken from the `Makefile` like `SERVERS` and `CLIENTS`, others are initialised by a call to `Configuration.params()` like `max_requests`. You can change or extend the parameters and also

add your own configuration arameters functions for evaluating different setups - select the `params` function by setting `CONFIG` in the `Makefile`.

10 Before reading the paper, work out and draw the top-level structure of the system from the supplied modules, in particular from *Multipaxos* and *Server*. You'll need to revise your diagram as you read the paper and as you learn more about the modules and the messages that they exchange. Note*: Commander* and *Scout* processes are spawned dynamically on-demand by *Leader* processes.

11 Look at the code in *Client* and *Database*. You will see that *Client*s create requests to move random amounts from one account to another. The goal is that each replicated server's database will execute the same sequence of requests.

12 Now examine the pseudo code for one of the modules in the paper, *Acceptor* say. Work out how the pseudo-code receives messages (`switch receive()`) and sends messages (`send`) and derive the Elixir mapping. Note: Greek letters are used for process-id variables.

13 Work out the message interface for *Acceptor*. Do the same for other modules in the paper and revise the diagram(s) for your system.

14 Now read the paper, building your understanding of the algorithm. It is advised that you skip the description of how the *Replica* module works and how invariants are maintained in your first pass through. You can gain a lot by becoming comfortable with reading the pseudo-code.

15 Please note that terminology used in the paper is a little different to that used in the lectures. All *proposers* are *Leaders*, *Proposal numbers are Ballot numbers*, *Proposed values* include a slot number to handle sequences of requests, Message tags and message contents also differ.

16 You'll also see that paper doesn't structure the server modules. I've structured the system as N *Server* nodes, M *Client* nodes and 1 top-level system node (called *Multipaxos*). Each *Server* node has 1 *Replica*, 1 *Leader*, 1 *Acceptor*, and 1 *Database* as well as dynamically-spawned *Commander*s and *scout*s. The *Multipaxos* module also spawns a global/shared Elixir *Monitor* process that other Elixir processes can send messages to, for debugging or in order to ensure correctness of the algorithm. You can additional checks to this *Monitor*.

Part 2 – Implementation

17 Do not start the implementation until you've made an initial attempt at part 1.

18 Download the directory of provided files and carefully examine the supplied modules.

19 Now complete the implementation by writing *Acceptor*, *Scout*, *Commander*, *Leader* and *Replica* (ideally in that order). The first three are straightforward. *Leader* and *Replica* are more complex, so write them afterwards after gaining confidence with writing the first three. My code for these 5 modules is ~300 lines.

20 The only banking command you need to implement is the '*move*' command, which *Clients* request, and *Database*s execute - you can implement additional banking commands like 'pay interest' if you wish. Your *Replica*s **can ignore** the reconfiguration command.

21 You may find it useful to mock your system first, e.g. use dummy messages to test that the flow of messages between modules is correct.

22 **Important**. **Develop an approach to debugging and testing your code from the beginning.** Sometimes you'll want to print a lot of debugging information, other times little or none. *If you do not instrument your code for debugging the coursework will take longer.* You can add your debugging printing functions to the *Debug* module and/or to the global *Monitor* module. Use assertions. Simplify the system when debugging, e.g, reduce the number of servers, clients, accounts.

23 Evaluate your system with experiments of your own. For example, you many wish vary the numbers of servers and clients, window-sizes, client request sending rate etc. to develop an understanding of the system. Check how many commanders and scouts are spawned.

Part 3 – Submission

24 Submit the directory for your implementation and your report (**report.pdf**) as single a *single zip file* on CATE called **multipaxos.zip** (do a **make clean** first). Include **report.pdf** inside the directory. Also include within the directory a **README.txt** with any *additional* instructions on how to run your system, particularly for interesting experiments. **It must be possible for us to run your code** using your instructions on CSG Linux machines or on a Mac (running Catalina or Big Sur).

25 Put any interesting outputs of your system in subdirectory called **outputs**. Use helpful files names. For example, **2_servers_5_ clients**, **server1_crash**, **live_lock_solved** and then refer to the files in your report.

26 **Optional**: If you have any feedback about the coursework, good or bad, including how it could be improved please submit it in a **`feedback.txt`**. Your feedback will not affect your mark either way.

27 Ensure that your report and your source files include your **name**(s) and **login**(s), for example,
```
# Mary Jones (mj16) and Peter Smith (ps16)
```
Source files and reports without your name(s) and login(s) will not be marked.

28 **Include in your report**:

29 One or more diagrams that show the structure and connectivity of the various modules. Indicate the normal flow messages for a typical client request. Use some notation to indicate replicated and spawned processes and some notation for any messages that are broadcast. DON'T SPEND TOO LONG ON THIS!! It's okay to use a scan or a photograph of a clear, hand-drawn version of your diagram(s) – you will not lose marks for doing so. Suggested maximum length: 1 page.

30 A brief description of **your** design/implementation. Focus on aspects that someone who is familiar with the algorithm but not your design/implementation will find useful. Avoid repeating information from the paper. Suggested maximum length: 1 page.

31 A brief description of **your** debugging and testing methodology. Again, focus on aspects that someone who is interested in updating/extending your implementation will find useful. Comment on what's hard to debug/test. Suggested maximum length: 0.5 page.

32 Evidence that your system works. For example, under normal situations, under load, under failures, without live-locking, etc. Summarise your findings. We're interested in descriptions that demonstrate your understanding and critical thinking, including your rationale when conducting experiments. **If your code does not work or only partially works you must explain what's not working**. Mention (1-2 lines) the environment that you used - OS, processor, RAM, cores, etc Suggested maximum length: 2.5 pages.

33 **The maximum report length is 5 A4 pages (excluding the cover page), s**o be concise and summarise your findings. Use graphs if they help.

34 The coursework will be marked out of 20. 10 marks for the Elixir implementation. 10 for the report. We'll be looking for correct implementation, clean readable code, helpful comments, plus a well-written insightful report.

35 If you don't manage to complete all parts of the coursework, clearly state what's incomplete or not working e.g. '*our solution is not working for XX*'. If you have some idea where the problem lies in the code, indicate it in your report.

36 OPTIONAL and UNASSESED. The algorithm described in the paper is not practical ☹ - see section 4 of the paper. Re-engineer it to be more efficient, e.g. change *acceptor* to use integers not sets, delete old slots. There are no extra marks for doing this.

37 OPTIONAL and UNASSESED. Implement recovery from process crash failures You'll need to save state onto disk and implement a reconfiguration command. There are no extra marks for doing this.

38 You can have an Appendix (1 extra page) if you wish to describe any Optional parts or any other extensions.

On completion you should know quite a lot about Multi-Paxos, so update your CV !