Distributed Algorithms 347

Coursework 2 - Multi-Paxos

01 The aim of 2nd 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` which can be downloaded from within the College network. There is also a website version of the paper at `http://paxos.systems`. The paper gives the algorithm in pseudo-code. The appendix of the paper and the authors' website also includes a Python version.

03 You'll need to write an Elixir implementation of the algorithm for components *replica, leader*, *acceptor*, *commander* and *scout* plus any others you use.

04 To help you get started a directory with various build files and source code can be downloaded (see later). It's okay to change the files supplied. Please report any bugs/improvement to me.

05 You can work and submit either individually or jointly with one classmate. Our recommendation is to do the coursework jointly with a classmate.

06 If your code does not work or only partially works you must explain what's not working.

07 All your source files must include a comment at the top of your files with your name(s) and login(s) e.g., `# Mary Jones (mj16) and Peter Smith (ps16)`
Source files without your name(s) and login(s) **will not be marked**.

08 Write a report with your evaluation of your implementation (a pdf file called **evaluation.pdf)**. Include in the report the environment you used for single node execution and Docker network execution - processor, RAM, cores, etc. Again ensure that your name(s) are on your report.

09 We're interested in comments that demonstrate your understanding and critical thinking, including your rationale when conducting experiments, not in the elegance of your Elixir code (although good code tends to be less buggy). Summarise findings. The maximum report length is 8 A4 pages, 10 A4 pages if do a bonus. You can include additional material in Appendices but it will not be considered when marking and count for the page limit.

10 Submit a directory with your report, code (do a make clean first), build files, etc as a **single *zip* file** on CATE. Include in **README.md** any additional instructions on how to run your system, particular for interesting configurations.. It should be possible for us to run your code using your instructions.

11 Your submitted files must also print correctly on DoC printers so check first.

12 There are 20 marks in total for the coursework. There are also up to 5 marks for doing a bonus part. You will be given credit for doing a bonus part if your mark is below 20. For example if you score 14 and 3 for a bonus part, your total mark will be 17. If you score 19 and 3 for a bonus part your total will be 20 (not 22), i.e. the total mark is capped.

13 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.

14 The most up-to-date version of this specification will be on the course webpage (not CATE), so check that version first.

15 **__________ The deadline for submission is Wednesday 21st February 2018 __________**

Part 1 – System Structure

16 A directory with various build files and Elixir modules can be downloaded from:

`http://www.doc.ic.ac.uk/~nd/347/cw02/multipaxos.tgz`

Familiarise yourself with the supplied files, in particular:

- `Makefile` has commands for compiling and running a system. There are brief instructions in `README.md` The Makefile will execute `gen_yml.sh` to generate a `docker-compose.yml` file with the right number and names of containers for this system.
- `paxos.ex`, `client.ex`, `server.ex`, `database.ex`, `monitor.ex` and `dac.ex` are my implementations of various components - to help get you started. You can change these if you wish.
- `Monitor` checks that each server database is executing the same sequence of requests. Currently 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 to 3 have each performed 657 database updates. Server 1 received 218 client requests, while servers 2 and 3 received 220 and 219 requests respectively. You can change `monitor` (and other components) to produce better checks more informative diagnostics.
- `configuration.ex` defines sime of the parameters in my implementation. `Dac.get_config` (called from `Paxos.main`) adds `n_servers`, `n_clients` and the `setup` type (`:docker` network or `:single` node) aslo from the command line arguments.. You can change the parameters and add your own. You can also define several versions of the parameters (using additional `version` functions). You can then select the version to use by using a `CONFIG=n` argument to `make`. The file `versions` is a convenient symbolic link within the Makefile directory to `lib/configuration.ex`

17 Before reading the paper, work out and draw the top-level structure of the system from the supplied components, in particular *Paxos* and *Server*. You will need to revise your diagram as you read the paper and learn more about the components and the messages they exchange. Note: the *commander* and *scout* are spawned dynamically on-demand by *leader* processes.

18 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 servers's *database* will execute the same sequence of requests.

19 Now examine the pseudo code for one of the components 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.

20 Work out the message interface for *acceptor*. Do the same for the other components and revise the drawing for your system.

21 Now read the paper building your understanding of the algorithm. You can skip the description of how the *replica* component works and how invariants are maintained in your first pass through. You can gain a lot by becoming comfortable with reading the pseudo-code.

22 You will see that the terminology used in the paper is a little different. All proposers are leaders, proposal numbers are ballot numbers, proposed values include a slot number to handle sequences of requests, message names and contents also differ.

23 You'll also see that paper doesn't structure the server components. I've structured the system as N servers, M clients and 1 top-level system component (called *paxos*). Each *server* has 1 *replica*, 1 *leader*, 1 *acceptor*, and 1 *database* as well as dynamically-spawn *commander*s and *scout*s.

24 **Submission: Include in your report one or more diagrams that show the structure and connectivity of the various components, with the types of messages that pass between them. It's okay to submit scanned/photographed of clear hand-drawn versions of your diagram(s). Use some notation to indicate replicated and spawned components and some notation for any messages that are broadcast.**

Part 2 – Implementation and evaluation

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

26 Now complete the implementation by writing implementations for *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.

27 The only command you need to implement is the '*move*' command which *clients* initiate and *databases* execute - you can implement additional commands like a pay interest command if you wish. *Replicas* can ignore the reconfiguration command.

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

29 Now test your system with some configurations.

30 If you get errors, add debug messages and examine the flow of messages - does the system live-lock? Also, reduce the number of servers, clients, accounts. Experiment with different configurations.

31 **Submission: In your report show and evaluate the output of your system. Vary the servers and clients, window-sizes, client request sending rate etc. to develop an understanding of the system. Fully test within a single node (`make run`), before testing using docker (`make up`). Check how many commanders and scouts are spawned. Describe any interesting design choices you made in your implementation.**

32 BONUS: 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. Evaluate and summarise your new results

33 BONUS: Test with process crash failures and implement recovery. You'll need to save state onto disk and implement a reconfiguration command. Summarise your implementation and results.

34 (**Optional**) Please add a few sentences after your report on a separate page (won't count against page limit) with any feedback you have about the labs and courseworks (1&2). Your mark will not change (up or down) if you include this - but it will help us improve labs and coursework(s) in the future.

35 ------- Congratulations - you now know quite a lot about Multi-Paxos**.** Update your CV ! -------