Distributed Algorithms 347

Coursework 2 - Multi-Paxos

01 The aim of 2nd coursework is to implement 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 `https://doi.org/10.1145/2673577` which can be accessed from with the College network.

03 There is also a website version of the paper at `http://paxos.systems`

04 The paper gives the algorithm in pseudo-code. The appendix of the paper and the authors' website also includes a Python version.

05 You'll need to write an Erlang implementation of the algorithm for components *replica, leader*, *acceptor*, *commander* and *scout*.

06 You'll provided with Erlang source code for *system*, *server*, *client*, and *database*. You're free to change the code for these. Please report any bugs/improvement to me.

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

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

09 All your source files must include a comment at the top of your files with your name(s) and login(s) e.g.,

```
1   %%% Mary Jones (mj16) and Peter Smith (ps16)
```

10 Source files without your name(s) and login(s) **will not be marked**.

11 **The deadline for submission is Tuesday 7th March 2017.**

12 Submit the directory of your files as a single *zip* file on Cate. Use PDF for the system structure diagram(s).

13 It should be possible for us to run your systems using your instructions.

14 Your submitted files must also print correctly on DoC printers so check first - you will not be given an opportunity to resubmit if a file does not print correctly.

15 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 question about your solution.

16 The most up-to-date version of this coursework will be on the course webpage.

Part 1 – System Structure

17 Download my implementations for *system*, *server*, *client* and *database* from:

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

18 Before reading the paper, work out and draw the top-level structure of the system from the supplied components, in particular *system* and *server*. You will need to revise your diagram as you learn more of the components and the messages they exchange.

19 Note: the components *commander* and *scout* are spawned dynamically on-demand by the *leader* component.

20 Look at the code in *client* and *database*. You will see that *client*s create random requests to move amounts from one account to another. The goal is that after the running the system each replicated *database* will have the same balances for each account.

21 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 Erlang mapping. Note: Greek letters are used for process-id variables.

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

23 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.

24 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.

25 You'll also see that paper doesn't structure the server components. I've structured the system as N servers and M clients. Each server has 1 *replica*, 1 *leader*, 1 *acceptor*, 1 *database* as well as lots of dynamically-spawned *commander*s and *scout*s.

26 Submission: One or more diagrams in a single PDF file that shows the structure and connectivity of the system, plus messages that pass between them. You can submit scanned/photographed hand-drawn versions of your diagram(s). Use some notation to indicate replicated and spawned components and some notation for messages that are broadcast e.g. clients send each request to all replicas. You may need to wait until you've finished the implementation to complete this part.

Part 2 – Erlang Implementation

27 Do not start the implementation until you've had a reasonable attempt at part 1.

28 Now complete the implementation by writing implementations for *acceptor*, *scout*, *commander*, *leader* and *replica* (ideally in this order). The first three are "straightforward". *leader* and *replica* are more complex, so write them afterwards after gaining confidence with writing the first three.

29 The only command you need to implement is the '*move*' command which *clients* submit and *database*s execute. *Replica*s can ignore the reconfiguration command.

30 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. Having the right Erlang datatype (e.g. list, map, set) will make coding easier – switch between types if necessary.

31 Now test your system with some configurations.

32 If you get errors, add debug messages and examine the flow of messages. Also, reduce the number of servers, clients, accounts, end_after time.

33 Submission: show the output of your system for
(i) a configuration of `5 servers, 3 clients, 10 accounts, running for 1 second` (this is the configuration in the supplied source code).
(ii) two other configurations of your choosing. Explain your choices.

34 Optional: The algorithm described in the paper is not practical ☹ - see section 4 of the paper and re-engineer it to be more efficient, e.g. change *acceptor* to use an integers not sets.

35 Optional: Test a distributed version over a Docker network and/or a real network.

36 Optional: Test with process crash failures and recovery. You'll need to save state onto disk and implement a reconfiguration command.

37 ---------- Congratulations - you've implemented Multi-Paxos. Update your CV! ----------