Department of Computer Science
COMP212 - 2018 - CA Assignment 2
Visualisation, Communication, and Fault-Tolerance in Distributed Computing

Assessment Information

Assignment Number	2 (of 3)
Weighting	7%
Assignment Circulated	16th March 2018
Deadline	13th April 2018, 17:00 UK Time (UTC)
Submission Mode	Electronic via Departmental submission system
Learning outcomes assessed	(1) An appreciation of the main principles underlying distributed systems: processes, communication, naming, synchronisation, consistency, fault tolerance, and security. (3) Knowledge and understanding of the essential facts, concepts, principles and theories relating to Computer Science in general, and Distributed Computing in particular. (4) A sound knowledge of the criteria and mechanisms whereby traditional and distributed systems can be critically evaluated and analysed to determine the extent to which they meet the criteria defined for their current and future development.
Purpose of assessment	This assignment assesses (i) the understanding of developing visualisation components for distributed computing systems that can facilitate the evaluation of developed solutions and can be exploited for demonstration and teaching purposes, (ii) the understanding of distributed communications via Java sockets and Java RMI, (iii) the understanding of fault-tolerance in distributed systems and implementing, simulating, and ev-

	aluating distributed protocols by using the Java programming language. Each student will be free to choose any of these three directions in this assignment.
Marking criteria	Marks for each question are indicated under the corresponding question.
Submission necessary in order to satisfy Module requirements?	No
Late Submission Penalty	Standard UoL Policy.

1 Objectives

This assignment asks you to extend the simulator that you have been developing in the Lab and in Assignment 1 **or** to develop a new implementation from scratch along (at least) **ONE** of the following lines:

1. Add a visualisation component to your simulator (Subsection 2.1).
2. Make your simulator available over the network through either Java socket programming or Java RMI (Subsection 2.2).
3. Add processor crash failure events to your simulator and experimentally evaluate two algorithms for consensus under the presence of crash failures (Subsection 2.3).

Important Note: It is sufficient for full marks that you implement one of the above requested functionalities. It is *not* a requirement to "extend" your previous development if you do not wish to do so. In this assignment you will only be marked for the functionalities requested in the present document. Moreover, you are **optionally** asked to **submit your simulator as a whole, in case you want your work to be considered for our class awards** and the latter is **independent of and will *not* affect by any means the marking of the present assignment**.

2 Description of coursework

You are only required to choose **ONE** of the following three subsections (i.e. only **ONE** of Subsections 2.1, 2.2, and 2.3). All choices are equivalent in terms of total marks.

2.1 Visualisation—100% of the assignment mark (breakdown of marks indicated in subquestions)

Enrich your simulator with a visualisation component by using any Java API of your preference, e.g., JavaFX. The minimal required functionality is for your graphics to be able to depict/produce:

- The underlying network of processors, including the nodes corresponding to processors and the edges representing the communication links, or part of it at any given time in case the network is too large to fit in screen. It would be desirable if the user could use the GUI to zoom in and out ranging from viewing a single processor up to viewing the whole network from an abstract perspective. (20% of mark)

- The most essential information about the local states of the processors, for instance, whether the processor is an elected leader or not and/or which other processors' ids a processor has collected so far. Some of these states/data could be easily visualised by the use of colours (e.g., red colour for the leader and gray for the rest, as we have done in the graphical illustrations in the lecture notes). (10% of mark)

- The most essential information about the messages being transmitted between processors. For example, in a leader election algorithm, this would typically be a single transmitted id (again colours could improve visibility in some cases). (10% of mark)

- A user-friendly real-time graphical simulation of at least one distributed algorithm, e.g., any of those algorithms that we have been discussing so far in the module. (30% of mark)

- It is a requirement that you also submit a report describing your developments and including some testing of your software (see the submission instructions in Section 3). (30% of mark)

The graphical illustrations that we have been including in the lecture notes may give you an idea of what a graphical simulation of a distributed system could look like. Of course, you are free to deviate from those and make your own graphical choices.

2.2 Networking—100% of the assignment mark (breakdown of marks indicated at the end of this section)

Enrich your simulator with a networking component. In particular:

- Your simulator should be available on a server through the network for clients to connect and request "simulation time".

- Therefore, the core functionality of the simulator may remain the same, the difference is that now the user can request the simulation from a remote machine that can provide the main simulation functionality and/or the results of a requested simulation.

To achieve this functionality you are free to use **either** Java socket programming **or** Java RMI. It is not a requirement that many clients can run simulations in parallel, but if you wish you could try to also achieve this functionality. For this assignment it is sufficient that at any given time a single client establishes a "connection" to the server and executes

its experiments. Moreover, you are allowed to have both the client and the server run on the same machine (connection to "localhost"), so that you don't have to use two distinct machines communicating over the network in order to test your software. At the module's webpage you can find some very basic example programs that you could use as a template to get you started. Moreover, you are strongly encouraged to consult Oracle's tutorials on Java sockets and Java RMI and especially their "getting started" examples (Sockets Knock Knock example and RMI compute engine, respectively).

- Achieving the basic network communication (that is, successfully establishing client-server connections and interchanging information between the two) is worth 30% of the assignment mark.
- To obtain further 20% a client should be able to trigger the execution of a simulation.
- The final 20% shall be awarded if the client can also successfully obtain all experimental data/results of the simulation.
- It is a requirement that you also submit a report describing your developments and including some testing of your software (see the submission instructions in Section 3). (30% of mark)

2.3 Failures—100% of the assignment mark (breakdown of marks indicated in subsections)

As in Assignment 1, you are here asked to carry out another experimental evaluation of distributed algorithms. In the present case, you are asked to slightly extend the functionality of your simulator so that it can simulate the occurrence of processor *crash failures*. A processor crashing means that it may stop at any point during its execution and that it will never recover from this (that is, from that point on that processor stops performing any operations, including state updates and transmissions of messages). Stopping can take place at any point during a processor's execution (e.g., before sending messages, after sending part/some of the messages, after sending all messages, during updating part of local state, ...). Randomness shall be essential in implementing processor failures as it will be used to determine at which points in time and in which processors the failures shall occur during an execution. Note that failures in your simulator could be either predetermined in advance (offline) or during the course of the execution (online).

The goal of the algorithms that you are going to evaluate is to solve a version of the *consensus* or *agreement* problem, defined as follows. Each processor u_i starts with an initial value $input(u_i)$ from a set X. Eventually, every processor must output a value from X so that the outputs satisfy the following conditions:

Agreement: No two processors decide on different values.

Validity: If all processors start with the same initial value $s \in X$, then s is the only possible decision value.

Termination: All non-faulty processors eventually decide.

The network on which the agreement algorithms are to be executed is a complete undirected (i.e., bidirectional communication) network, meaning that every processor has both an incoming and an outgoing link to every other processor in the system. Processors, as usual, operate in synchronous rounds. In our setting, at most f of the (n in total) processors can fail in any execution, where $f < n$ is a predetermined integer known to the algorithm. The processors also know a prespecified default value s_0 from the set of values X.

2.3.1 Implementing the FloodSet Algorithm—30% of the assignment mark

As a first step, you are required to implement the FloodSet algorithm for agreement under processor failures. The pseudocode of the FloodSet can be found in Lecture 20 and is also given here for convenience (Algorithm 1).

Remark: For simplicity of presentation we here assume that in a round processors first transmit, then receive and update, and then the round ends. Either adapt your simulator to be able to execute also this type of rounds or carefully modify the algorithm so that it matches your simulator's type of rounds without becoming incorrect.

2.3.2 Implementing the OptFloodSet Algorithm—30% of the assignment mark

Next, you are required to implement another algorithm for the same problem, called OptFloodSet. The main difference between the two algorithms is that OptFloodSet instead of having every active processor always sending all the values it has heard of so far, it now sends only the first two values it becomes aware of (its own and the first new one it receives, if there is one).

Informally, the processors operate as in FloodSet, except that now each processor u_i broadcasts (i.e., sends to all active processors in one step) at most two values throughout the course of the algorithm. The first broadcast takes place in round 1, when u_i broadcasts its initial value. The second possible broadcast takes place the first time u_i receives a value s' different than its initial value (if there is one in the network). If there are two or more new values in that round, then any one of these may be selected for broadcast. As in FloodSet, processor u_i decides s if its final set W_i is the singleton set $\{s\}$ and otherwise decides the predetermined $s_0 \in X$.

The pseudocode of OptFloodSet is given in Algorithm 2. As in FloodSet, **don't forget to take into account that the round definition is here slightly different than usual.**

Algorithm 1 FloodSet

Code for processor u_i, $i \in \{1, 2, \ldots, n\}$:

Initially:
u_i knows its input value $input(u_i) \in X$ and a default value $s_0 \in X$
$W_i := \{input(u_i)\}$
$decision_i :=$ '$?$'
Also has access to the current round and knows the upper
bound f on # faults

1: **if** $round \leq f + 1$ **then** // The following to be always executed by all processors, i.e.,
2: // also in round 1 in which no message has been received
3: send $\langle W_i \rangle$ to all processors
4:
5: upon receiving $\langle inW_j \rangle$s from in-neighbours
6: $W_i := W_i \bigcup_j inW_j$
7: **end if**
8: **if** $round = f + 1$ **then**
9: **if** $|W_i| = 1$ **then**
10: $decision_i := s$, where $W_i = \{s\}$
11: **else**
12: $decision_i := s_0$
13: **end if**
14: **end if**

Algorithm 2 OptFloodSet

Code for processor u_i, $i \in \{1, 2, \ldots, n\}$:

Initially:
u_i knows its input value $input(u_i) \in X$ and a default value $s_0 \in X$
$W_i := \{input(u_i)\}$
$newInfo_i :=$ false
$decision_i :=$ '?'
Also has access to the current round and knows the upper
bound f on # faults

```
1: if round ≤ f + 1 then
2:     if round = 1 then
3:         send ⟨input(u_i)⟩ to all processors
4:     else if newInfo_i = true then
5:         send ⟨s_new⟩ to all processors, where s_new ∈ W_i (any) and s_new ≠ input(u_i)
6:         newInfo_i = false
7:     end if
8:
9:     upon receiving ⟨s_j⟩s from in-neighbours
10:    W_i := W_i ⋃_j s_j
11:    If this is the first time W_i increased then newInfo_i := true
12: end if
13: if round = f + 1 then
14:    if |W_i| = 1 then
15:        decision_i := s, where W_i = {s}
16:    else
17:        decision_i := s_0
18:    end if
19: end if
```

2.3.3 Experimental Evaluation, Comparison & Report—40% of the assignment mark

After implementing the FloodSet and OptFloodSet algorithms, the next step is to conduct an experimental evaluation of their correctness and performance.

Correctness. Execute each algorithm in complete networks of varying size (e.g., $n = 3, 4, \ldots, 1000, \ldots$; actually, up to a point where simulation does take too much time to complete), starting from various different input value assignments for each given network size, and on a variety of different occurrence-patterns of the at most f crash failures that can take place during any execution. In each execution, your simulator should check that eventually agreement is satisfied, that is, that all processors satisfy the three agreement conditions mentioned previously.

Performance. Execute, each algorithm on the same combinations of parameters mentioned above. For each execution, your simulator should record the total number of messages transmitted until termination (as the number of rounds is trivially always $f + 1$).

Note: Randomness shall be essential in the above experiments and shall be used both in assigning the initial values to processors and in determining the occurrence of processor failures. Naturally, this means that you should perform a number of experiments for each choice of the parameters n and f and take averages as you did in Assignment 1.

After gathering the simulation data, plot them as follows. Depending on your data, you may provide plots in which: (i) the number of failures occurring is fixed (or a fixed function of n, e.g. $\log n$) and the x-axis represents the (increasing) size of the network while the y-axis represents the complexity measure (number of messages), (ii) the size of the network n fixed and the x-axis represents the (increasing) number of failures in the system while the y-axis represents again the complexity measure (number of messages), (iii) the plot is 3-dimensional: the x-axis represents n, the z-axis represents f, and the y-axis represents the complexity measure. You may produce individual plots depicting the performance of each algorithm (possibly comparing against standard complexity functions, like n, $n \log n$, or n^2) and you are *required* to produce plots comparing the performance of both algorithms in identical settings. You can use gnuplot, JavaPlot or any other plotting software that you are familiar with.

The final *crucial* step is to prepare a concise report (at most 5 pages including plots) clearly describing your main implementation choices, the main functionality of your simulator, the set of experiments conducted, and the findings of your experimental evaluation of the above algorithms. In particular, in the latter part you should try to draw conclusions about (i) the algorithms' correctness and (ii) the performance (here only messages) of each algorithm individually and when the two algorithms are being compared against each other (e.g., which one performs better and in which settings?).

3 Deadline and Submission Instructions

- The deadline for submitting this assignment is **Friday, 13th April 2018, 17:00 UK time (UTC)**.

- Submit

  (a) The Java source code for all your programs (your implementation for the present assignment, depending on your choice of subquestion, and optionally the complete simulator that you developed to be considered for our class awards; could be the same program for both),

  (b) A README file (plain text) describing how to compile/run your simulator to produce the various results required by the assignment and in general in order for us to assess its functionality, and

  (c) A concise self-contained report (at most 5 pages including everything) describing the functionalities implemented for the present assignment, your implementation choices, software-testing (if one of Subsections 2.1 or 2.2 was chosen) or experiments (if Subsection 2.3 was chosen), and conclusions in PDF format. If you wish you may submit at most 5 more pages to describe/support your simulator as a whole for the class awards. These could be either separate documents or be part of a single (at most) 10-page document. In case you *do not* wish your work to be considered for our class awards please indicate this clearly in the first page of your report (otherwise, the software that you will submit as part of the present assignment shall be considered for the awards by default).

  Compress all of the above files into **a single ZIP** file (the electronic submission system won't accept any other file formats) and **specify the filename** as *Surname-Name-ID.*zip. It is extremely important that you include in the archive all the files described above and not just the source code!

  Remark. Your grade for this assignment will be only based on the additional functionality that you will implement (and as described in the present document) and not on the simulator as a whole. So, you can obtain full marks just by implementing the functionality requested here. For the awards, our selection of the winners shall be based on a number of criteria such as (i) the average grade in the two assigments, (ii) the overall functionality of the final simulator submitted, (iii) novelty, user-friendliness, ... We will *not* provide additional personalised feedback for the awards. The marks and feedback in your assignments should as a general rule be sufficient and indicative of your development as a whole.

- Submission is via the departmental submission system accessible from
  `https://sam.csc.liv.ac.uk/COMP/Submissions.pl?strModule=COMP212`

Good luck to all!