CSI3344 Distributed Systems

Assignment 2: A Report

(You need to choose one topic from the four options)

General Information:

- This assignment is a team based report for up to two students (depending on the topic you selected, the report may be a group one or an individual one, however). It consists of two parts therefore requires two submissions.
- The first part requests you complete a report based on a research into the topic or a mini programming project that you selected from a number of choices (see the listed topic options in the following pages).
  Note: You may alternatively develop your own favorite research topic (as topic 5), provided it is a closely related to the distributed systems. In such a case, you are requested to write a draft Proposal/Abstract of the topic/tasks (in no more than 1 page) and send it to (or discuss it with) your tutor beforehand. Your tutor will then assess it and approve / disapprove it, mainly depending on whether or not the topic is acceptable, the research is feasible and the workload is reasonable, etc.
- The second part requests you give a seminar/presentation to the class, highlighting the main points (i.e., the main findings) or outcomes of your research/project. The presentation should not exceed the 15-minute time limit (Note: you are requested to prepare about 10~15 pages of PPT slides for a research topic, or PPT + demonstration in case of programming project. A separate submission link is created for you to submit the PPT slides). Additional question time will be given after each presentation. The presentation is scheduled in the workshop sessions of the last two teaching weeks (see Unit Schedule). Please make sure you are ready for your presentation at or before the 11th lecture session/s.

Due date: Submission 1: Seminar resource due on *Monday @9:00 am 20 May 2019* (see Unit Schedule)
Submission 2: Seminar presentation PPT slides, by *Wednesday 22 May 2019 @12:30 pm*

Weight: 30% of unit mark

Submission Requirements:

- The first submission (for seminar resource) includes a report and any additional documentation associated with the report. The second submission (for seminar presentation) includes PPT slides, and some possible additional handouts. These should be submitted via Blackboard (note that a separate submission link will be created for you to submit the Presentation PPT slides). For detailed submission procedure, please refer to "How to submit your Assignment online" item in the Assignment section of your Blackboard.

- If you like to submit your seminar resource and presentation slides in one lot, you may do so on or before the first submission due date (in such a case, please let your tutor know it beforehand).

- The report file must be in Word or PDF format (see next page for detailed Report format requirement). If you submit multiple files, or if you take the mini project (thus need to include separate runnable code file/s), please zip your files into one file before submission. Please name your submission file in the format of <your_student_ID>_< your_full_name>_A2_CSI3344.zip. For example, if your student ID was 1234567 and your full name was Abcd Xyz, your submission file should be named 1234567_Abcd_Xyz_A2_CSI3344.xxx (where xxx is the extension of your submission file).

- NO hard copy submission is required.

- Note that files found to contains viruses or other infecting mechanisms shall be awarded a zero mark, as well submissions found contain any form of cheating/plagiarism.

- Your attention is drawn to the university rules governing cheating and referencing. In general, cheating is the inclusion of the unacknowledged work of another person. Plagiarism (presenting other people's work and ideas as your own) and cheating will result in a zero mark.

(GO TO NEXT PAGE)

Research report format requirement:

Report Must contain	**Cover/Title page** Must show assignment title, your student ID and name/s, due date and the title of your chosen topic
	Executive Summary This should represent a snapshot of the entire report that your tutor will browse through and should contain the most vital information requested. The Executive Summary should be in between Cover page and ToC.
	Table of Contents (ToC) This must accurately reflect the content of your report/project and should be generated automatically in Microsoft Word with clear page numbers.
	I**ntroduction** Provide background information of the topic/report; Introduce the report, define its scope and state any assumption; Use in-text citation /reference where appropriate.
	Main body The report should contain (but not limited to): • Understanding of concepts/topics involved in the report. • The problems/ questions being solved/answered • Significance of the research/report • Strategies used to solve the problem (e.g., provide a solution or an approach to develop a solution?) • Discussion or a critique of the solution developed/achieved, etc.
	Conclusion Outcomes or main finding/s from the research/project?
	References A list of end-text reference formatted according to the ECU requirements using the APA format. It is recommended that EndNote be used to organize/manage references, which should be ideally comprise of books, journal articles and conference papers (A few Web references are OK but they should account for a small part of the whole references).
Other requirement	The report should be no more than 4000 words (excluding references and diagrams) and be named as per Submission requirement, and should be in a single file unless you chose option 4, in which case, a .zip file. The text must use font **Times New Roman** and **be not smaller than 12pt**.

Option 1 (for those who are good at comparison of technologies)

SURVEY REPORT: A COMPARISON AMONG *DISTRIBUTED COMPUTING*, *GRID COMPUTING* AND *CLOUD COMPUTING*

(**Topic report type**: *individual work*)

Background Information

Distributed computing (DC) refers to a collection of hardware and software systems that contain more than one processing or storage element but appearing as a single coherent system running under a loosely or tightly controlled regime. A key feature of such a system is that the computers/devices in the system do not share a memory but they coordinate their work/s by passing messages asynchronously or synchronously. *Grid computing (GC)* is a form distributed system whereas *cloud computing (CC)* is a style of grid computing. While the DC, GC and CC share similar key features as systems, they can be quite different from each other. For example, some people simply view cloud computing as an environment in a user's machine that uses the internet to access someone else's software running on someone else's hardware in someone else's data center.

Task

You are requested to conduct a survey on *cloud computing* (i.e., a survey primarily on cloud architecture/s, modes of clouds, key features/properties and advantages and disadvantages of using CC, etc.), and make comparison among DC, GC and CC in terms of similarity/difference, scalability, benefit and cost, security/privacy, and so on. Your view on how to move cloud computing to its full potential is also requested.

Based on your research, write a report presenting your research outcomes. Your report should not be longer than 15 A4 pages, including text, figures and tables; Figures/diagrams/tables should be with captions and numbered and cited in the text; the references should be denoted in text and fully listed in 'References'.

END of Option 1

Topic option 2

A report on:

The Development History of Distributed System

(**Topic report type**: *individual work*)

Background Information

The concept and the practice of *distributed system (DS)* has evolved along its long development history in the areas of network and data distribution. For example, the first widespread DS was the so-called *local-area* network, such as Ethernet, which was invented in the 1970s. After a few decades of development, a DS can now be defined as *a collection of independent devices that appears to its users as a single coherent system, in which networked components communicate and coordinate their actions only by passing messages*. Some significant characteristics of DSs include concurrency of components, lack of global clock, and independent failure of components.

When studying the DS development history you would never miss the terminologies that have historically contributed to the DS development, from *local-area network, internet, email, distributed database, distributed computing*, all the way to *cloud computing*. Similarly, when tracking DS's technical terminologies, you may never miss keywords such as RPC, RMI, communication protocol, distributed naming services, middleware services, synchronization, concurrency control, distributed replication, fault tolerance, and DS security, and so on.

Task

Based on the above keywords (or concepts or terminologies), you are requested to conduct a research on the topic of *history of DS development*. You should use (but not limited to) the abovementioned keywords to guide your research.

Based on your research, write a report presenting your research outcomes. You are recommended to write the key DS development events in a chronological way. A comparison between the generic distributed system and the most recent development of distributed computing system, e.g., cloud computing system, is also required.

END of option 2

Topic option 3

A Research on

MIDDLEWARE FOR DISTRIBUTED SYSTEMS: ITS PAST, PRESENT, AND FUTURE

(**Topic report type**: *individual work*)

Background Information

In the early days, people saw no need for middleware in distributed systems. With the evolution of distributed systems from handling relatively simple and specific tasks in a homogenous network to dealing with complex and various activities in a vast heterogeneous environment, it is middleware that makes modern distributed systems perform effectively and more efficiently. The evolution should continue with the rapid increase in networked activities in peoples' daily life, which will drive not only the further expansion of middleware in its current territory, but also the middleware technology to some new domains in the future.

Task

You are required to prepare a report on the evolution of middleware in terms of its ***models*** and ***services*** to the development of distributed systems in the past, present, and future.

Other requirements:

- The events stated in your report should be important, relevant, and chronological;
- The evidence and data used for supporting your arguments should be solid and referenced;
- The predications should be based on logical induction and/or statistics;
- The conclusions should be coincident with your evidence, data, and analysis;
- Figures/diagrams/tables should be with captions and numbered and cited in the text;

END of option 3

Topic option 4 (for those who are good at programming):

A Mini RMI programming project

(**Topic report type**: *Group work for two students*)

Background Information

Majority of distributed systems are based on explicit message exchange between processes. With **remote procedure call (RPC)**, a client application component can effectively send a request to another application component (at the server side) by doing a local procedural call, which results in the request being packaged as a message and sent to the callee (at the server side). Likewise, the result will be returned to the client application as the result of the procedure call.

As the popularity of object technology increased, techniques developed to allow calls to remote objects, leading to what is known as **remote method innovation (RMI)**. An RMI is essentially the same as an RPC, except that operates on objects instead of applications.

A client-server interaction can be principally implemented by using either RPC or RMI. If the interaction was between applications residing in two machines (or, more generally, processes), it is conceptually named “two-tier” interaction. If the interaction involved applications residing in three or more machines/processes, it is sometimes named “three-tier” or “multi-tier” interaction.

This mini MRI project is to be completed in two stages. The first stage is to implement a simple *“two-tier” interaction* between a client application and a server application, while the second stage is to upgrade the *“two-tier” interaction* into a *“three-tier” interaction* by creating data server that acts as a data center for possible data request/service from the server-side application of the *“two-tier” interaction* (i.e., completed in the first stage).

This project to be completed by a group of two students. If you are really interested in doing it individually, that is fine but no workload is to be reduced.

Tasks:

Refer to the source Java-like pseudo-code in the Appendix that defines a class *Average*. It is an application that runs on a local machine. It accepts a number of integers as marks to some units; then adds them up, and finally returns the average as double type. The result is displayed in a message box.

Stage 1 (*25 marks*):

Your task is to develop a simple client/server application for an ***Honours Pre-assessment System***. The client makes RMI to a remote application/method, *Evaluator*, which is used to assess if a student is qualified for an Honours study based on the record of his/her Bachelor's course. The basic requirements of this application include:

1. Only authenticated users are allowed to use this application.

2. Client application should not only allow users to type in their ID and marks of individual units one by one through keyboard, but also provide functions to display the evaluation results received from the remote server.

   The number of input marks (of integers data type) should be at least 12 (i.e., it counts 3 semester advanced standing of his/her previous study) and at most 30, including failed mark/s and/or repeated marks of the same unit. A student is automatically disqualified for Honours study if he/she has 6 or more Fail (or Incompletion) results during Bachelor's course.

3. Server application provides services/operations that process the input data, calculate the averages, assess the qualifications against the evaluation criteria, and inform the client the evaluation results.

   The basic operations on the server should include displaying individual marks on the server in their input order, selecting the best 8 marks, calculating the course average, calculating the average of best 8 marks, evaluating the qualification, and sending the 'correct' evaluation result back to the client.

   The evaluation criteria are as follows:
   - If the course average is equal or greater than 70, return a message

     "<student ID>, <course average>, QUALIFIED FOR HONOURS STUDY!";

   - If the course average is less than 70, but the average of best 8 marks is 80 or higher, return a message

     "<student ID>, <course average>, <best 8 average>, MAY HAVE GOOD CHANCE! Need further assessment!";

   - If the course average is less than 70, but the average of best 8 marks is between 70 and 79, return a message

"<student ID>, <course average>, <best 8 average>, MAY HAVE A CHANCE! Must be carefully reassessed and get the coordinator's special permission!";

- If the course average is less than 70, and the average of best 8 marks is also less than 70, return a message

  "<student ID>, <course average>, <best 8 average>, DOES NOT QUALIFY FOR HONORS STUDY! Try Masters by course work."

4. The client and server-side applications should be able to run in deferent machines.

(Note: There is no restriction on selection of programming language/environment to implement the application/s. – You MAY use whatever programming language / environment you like, although Java, Python or C++ is preferred).

[Hints] You may use an array to store and select the input data on the client and/or server. You need to make the array empty when the results are returned to the client. (Why?)

Remember the steps to create a RMI application (e.g., in Java, etc.):

1. define the interface for the remote object
2. implement this remote interface
3. create stub and/or skeleton
4. implement the server software
5. implement the client software

Stage 2 (*5 marks*):

This task consists of two parts:

(1) Test the client/server application that is completed in *Stage 1,* using various data (for simplicity, we call the server-side application *server-1*);
(2) Create a *third-tiered* server, which acts as a "tiny" database server (called *server-2*). The server-1 and server-2 communicates by RMI as well, however the client side application has no direct access to server-2.

Additional functions are as below:

a) Server-2 stores students' course study historic records, such as unit selection history, unit result (each allows up to 3 attempts if there was any Fail or Incompletion);

b) Server-2 can also stores data received from client application by way of server-1. That is, whenever a user typed in an ID and mark/s of individual units (see step 2 in Stage 1), server-1 may pass the data to server-2. In such a case, server 2 will make necessary check, validate and save/update the data as student's record (with or without unit names/s. etc.), if applicable, on server-2;

c) The system allows a student check if he/she is qualified for Honours study using his/her course records data stored in the database at server-2. That is, a student may input an ID only and pass it to server-1, requesting an Honours assessment using his/her course data stored in server-2. After the student ID is received from the client, server-1 requests student data from server-2. Once requested data was received from server-2, the assessment is done at server-1, and the assessment result is sent back to the client (i.e., a similar message is sent to the client as the result of his/her request for an assessment. See step 3 in *Stage 1*).

d) (*Optional*) The system also allows a student check his/her unit results, say, by typing in Student ID. In such a case, server-1 validates data input and makes request from server-2; server-2 then responses to server-1 which makes necessary pre-processing and finally passes the result back to the client request, etc..

Bonus Marks:

You are encouraged to implement your server-2 by connecting the server-1 with a real database server (e.g., MySQL server, SQL server 2010, Oracle server etc.).

- Develop a short interface using SQL or other tool/s from the database server such that it
  - (i) allows users input (or enter information for) their course data (say from the client side, by way of server -1). Use an integer to keep track of how many record have been created so far.
    Be careful – Don't allow the user create more than 30 units' results;
  - (ii) allows users search unit result information by Student ID + Unit ID, etc., instead of by Student ID only, etc.
- Explain and extend to accommodate error messages, etc.

This additional work and effort to improve the program and make it more useful can be worth bonus marks. Up to 5 marks can be added for the above improvement. However, Bonus marks are not required and will not improve your grade above 100%.

Basic requirements

1. The applications should include all required components.
2. The observable behaviours of your application should be consistent with what is described.
3. The application provides necessary error handling mechanisms.
4. Provide a project report that is well structured and informative (no more than 15 pages)*, and separate code files of your project.

The project report format requirement

Your project report may take the similar format as that of the research report showing in page 3, except for the Main Body part, which should now include

i. (A brief) Introduction of RMI (or any software used)
ii. Application requirements
iii. Application design and implementation procedure
iv. Application setting-up and usage steps (e.g., screen shots)
v. An example showing snapshots of application running status and input/outputs
vi. Conclusion

Appendix: source code

The above Sections i) to vi) should be no longer than 15 pages.

Appendix (for Option 4 only):

```
// Fig. 4.9: Average2.java from Deitel & Deitel <<Java How to Program>> (5th ed)
// Class average program with sentinel-controlled repetition.
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
public class Average2 {
       public static void main( String args[] ) {
          int gradeCounter,  // number of grades entered
           gradeValue,    // grade value
           total;         // sum of grades
          double average;    // average of all grades
          String input;      // grade typed by user
          // Initialization phase
          total = 0;         // clear total
          gradeCounter = 0;  // prepare to loop
             // Processing phase
             // prompt for input and read grade from user
          input = JOptionPane.showInputDialog(
             "Enter Integer Grade, -1 to Quit:" );
          // convert grade from a String to an integer
          gradeValue = Integer.parseInt( input );
          while ( gradeValue != -1 ) {
             // add gradeValue to total
             total = total + gradeValue;
             // add 1 to gradeCounter
             gradeCounter = gradeCounter + 1;

             // prompt for input and read grade from user
             input = JOptionPane.showInputDialog(
```

```
                "Enter Integer Grade, -1 to Quit:" );

                // convert grade from a String to an integer

                gradeValue = Integer.parseInt( input );

                }


                // Termination phase

                DecimalFormat twoDigits = new DecimalFormat( "0.00" );

                if ( gradeCounter != 0 ) {

                average = (double) total / gradeCounter;

                // display average of exam grades

                JOptionPane.showMessageDialog( null,

                "Class average is " + twoDigits.format( average ),

                "Class Average", JOptionPane.INFORMATION_MESSAGE );

                }

                else

                JOptionPane.showMessageDialog( null,

                "No grades were entered", "Class Average",

                JOptionPane.INFORMATION_MESSAGE );

            System.exit( 0 );    // terminate application

        }  // end method main

}  // end class Average2
```

Reference:

Deitel and Deitel, Java How to Program (fifth ed), Prentice Hall, 2003.

END of option 4

END of Assignment Description