Module 26 Distributed Systems

Module title	Distributed Systems
Module NFQ level (only if an NFQ level can be demonstrated)	8
Module number/reference	BSCH-DS
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Award stage
Semester (semester1/semester2 if applicable)	Semester 1
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	10
List the teaching and learning modes	Direct, Blended
Entry requirements (statement of knowledge, skill and competence)	Learners must have achieved programme entry requirements.
Pre-requisite module titles	BSCH-CH, BSCH-OSD, BSCH-DNA
Co-requisite module titles	None
Is this a capstone module? (Yes or No)	No
Specification of the qualifications (academic, pedagogical and professional/occupational) and experience required of staff (staff includes workplace personnel who are responsible for learners such as apprentices, trainees and learners in clinical placements)	Qualified to as least a Bachelor of Science (Honours) level in Computer Science or equivalent and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.
Maximum number of learners per centre (or instance of the module)	60
Duration of the module	One Academic Semester, 12 weeks teaching
Average (over the duration of the module) of the contact hours per week	4
Module-specific physical resources and support required per centre (or instance of the module)	One class room with capacity for 60 learners along with one computer lab with capacity for 25 learners for each group of 25 learners

Analysis of required learning effort		
	Minimum ratio teacher / learner	**Hours**
Effort while in contact with staff		
Classroom and demonstrations	1:60	30
Monitoring and small-group teaching	1:25	18
Other (specify)		
Independent Learning		
Directed e-learning		
Independent Learning		102
Other hours (worksheets and assignments)		100
Work-based learning – learning effort		
Total Effort		250

Allocation of marks (within the module)					
	Continuous assessment	**Supervised project**	**Proctored practical examination**	**Proctored written examination**	**Total**
Percentage contribution	40%			60%	100%

Module aims and objectives

The aim of the module is to teach the theoretical and practical underpinnings of distributed system design and implementation. Learners are introduced to the myriad of issues involved when moving from a single computer system to one composed of multiple nodes. Issues covered include but are not limited to: processes, communication, consistency, replication, leader election, fault tolerance, and clock synchronisation. Learners are exposed to the compromises that must be made during design in each of these areas with respect to the entire system.

The objectives of the module are to give learner the ability to consider all of the issues presented above such that they can design and implement distributed systems. These systems that are produced focus on areas desired in the required system while also trying to minimise the trade-offs that are built into the design of said system.

Minimum intended module learning outcomes

On successful completion of this module, the learner will be able to:

1. Explain the challenges involved in designing and developing distributed systems
2. Compare different architectural and communication models
3. Write distributed applications through the use of a middleware layer
4. Explain clock synchronisation and system state capturing strategies

5. Evaluate appropriate distributed algorithms for leader election and mutual exclusion
6. Describe how faults are tolerated and handled (e.g. abnormal process termination, byzantine failures)
7. Discuss the challenges and mechanisms involved in maintaining consistency and replication

Rationale for inclusion of the module in the programme and its contribution to the overall MIPLOs

The module enables learners to design, implement, and interact with distributed systems. This forms the basis of cloud computing which is a model more web based services and applications are migrating to, particularly including those that have a mobile component where multiple mobile devices and instances in the cloud are communicating and working together. Appendix 1 of the programme document maps MIPLOs to the modules through which they are delivered.

Information provided to learners about the module

Learners receive a programme handbook to include module descriptor, module learning outcomes (MIMLO), class plan, assignment briefs, assessment strategy, and reading materials.

Module content, organisation and structure

Distributed systems overview

- Design goals and challenges: heterogeneity, openness, scalability, consistency, transparency, mobility, fault tolerance, security
- Fundamental concepts: multicomputer vs multiprocessor systems
- Tightly vs loosely coupled systems, horizontal vs vertical distribution
- Middleware based systems.

Processes

- Threads and Processes: using threads to overlap communication and computation, progress migration, multithreaded clients and servers.
- Virtualisation (both resource and network), virtualisation architecture, PVM and VMM models of virtualisation.
- Stateful vs stateless processes
- Code migration, Weak mobility, Strong mobility.

Communication

- Synchronous and Asynchronous communication
- Sockets
- RPC

- Message Passing
- Message Oriented Middleware

Consistency and Replication

- The need for replication in a distributed system and the challenges it poses
- How much replication? And how to implement it
- Consistency of replicated data/state and ensuring it gets updated correctly
- Corruption of replicated data/state and resolving it

Clock Synchronisation

- Concept of time in distributed systems
- Physical vs logical time
- Clock synchronisation algorithms
- NTP (Network Time Protocol)
- Vector clocks
- Coordination, Mutual Exclusion, Leader election
- Distributed algorithm properties: fairness, liveliness, safety
- Distributed algorithms for leader election, mutual exclusion, and coordination

Fault tolerance

- Fault tolerance: Availability, Reliability, Safety, and Maintainability
- Faults, Errors, and Fault Tolerance
- Transient, Intermittent, permanent faults, and Failure models
- Failure masking with redundancy or replication, design issues this presents
- Fault tolerance without and with byzantine failures

Distributed system architectures

- Layered architecture
- Object based architecture
- Data Centred architecture
- Event based architecture
- Centralised architecture
- Multitiered architecture
- Decentralised architecture
- Peer to Peer architecture

Case studies

- Examples of distributed system use in the real world
- Bittorrent
- BOINC (SETI@home etc)

- Blockchain
- Other examples may be used here.

Module teaching and learning (including formative assessment) strategy
The module is taught as a combination of lecture and lab sessions. The lecture sessions discuss and explain to learners the challenges involved in both designing and implementing distributed systems. Learners are exposed to issues including but not limited to: consistency, replication, process management, and communication. In the practical lab sessions learners have the opportunity to interact and develop distributed applications for through the use of a middle ware layer. This gives learners exposure to the challenges of synchronising distributed processes, debugging distributed applications, and the need to account for the cost of communication in a distributed system.

Assessment is split into two. In terms of continuous assessment there are three major assignments that first test the learner's ability to make a working distributed application before moving onto more challenging applications that involve larger communication and cooperation. Finally, there is an end of semester exam that tests the learners understanding of the theoretical material.

Timetabling, learner effort and credit
The module is timetabled as one 2.5-hour lectures and one 1.5-hour lab per week.

The number of 10 ECTS credits assigned to this module is our assessment of the amount of learner effort required. Continuous assessment spreads the learner effort to focus on the issues and challenges that arise with distributed system programming.

There are 48 contact hours made up of 12 lectures delivered over 12 weeks with classes taking place in a classroom. There are also 12 lab sessions delivered over 12 weeks taking place in a fully equipped computer lab. The learner will need 102 hours of independent effort to further develop the skills and knowledge gained through the contact hours. An additional 100 hours are set aside for learners to work on worksheets and assignments that must be completed for the module.

The team believes that 250 hours of learner effort are required by learners to achieve the MIMLOs and justify the award of 10 ECTS credits at this stage of the programme.

Work-based learning and practice-placement
There is no work based learning or practice placement involved in the module.

E-learning
The college VLE is used to disseminate notes, advice, and online resources to support the learners. The learners are also given access to Lynda.com as a resource for reference.

Module physical resource requirements
Requirements are for a classroom for 60 learners equipped with a projector, and a 25 seater computer lab for practical sessions with access to a Distributed System SDK and IDE (for example MPI but this may change should better technologies arise).

Reading lists and other information resources
Recommended Text
Burns, B. (2018) *Designing Distributed Systems*. Sebastopol: O'Reilly Media.

Gropp, W. (2015) *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Cambridge: MIT Press.

Tanenbaum, A. S. and Steen, M. van (2014) *Distributed Systems Principles and Paradigms*. Harlow: Pearson.

Secondary Reading:
Gropp W. (2015*) Using Advanced MPI: Modern Features of the Message-Passing Interface*. Cambridge: MIT Press.

Specifications for module staffing requirements
For each instance of the module, one lecturer qualified to at least Bachelor of Science (Honours) in Computer Science or equivalent, and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.. Industry experience would be a benefit but is not a requirement.

Learners also benefit from the support of the programme director, programme administrator, learner representative and the Student Union and Counselling Service.

Module Assessment Strategy

The assignments constitute the overall grade achieved, and are based on each individual learner's work. The continuous assessments provide for ongoing feedback to the learner and relates to the module curriculum.

No.	Description	MIMLOs	Weighting
1	Practical Assignment that introduces the learners to a basic distributed system application. It covers the setup of processes and the basic mechanisms of communication	1,3	7.5%
2	Practical Assignment that builds a distributed systems application with more communication and processing. Learners are expected to synchronise and coordinate distributed processes.	1,3	12.5%
3	Practical Assignment that asks learners to design and build a distributed system using a learner chosen architecture. Only a brief description of the distributed system functionality is given.	1,2,3	20%
4	Written exam that tests the theoretical aspects of the module	1-7	60%

All repeat work is capped at 40%.

Sample assessment materials

Note: All assignment briefs are subject to change in order to maintain current content.

Assignment 01: Monte Carlo Simulation to determine the number PI

Introduction:

In this assignment you will be tasked with determining the value of the number PI through simulation. Determining an accurate value of PI is difficult to do through normal methods because PI is an irrational number i.e. there is no single fraction that can express the value of PI. Thus to determine the value of PI we must use other methods. In this case we will use Monto Carlo Simulation where we generate the value of PI by using random sampling.

In order to simulate this we will need to remember our formula for the area of a circle, which in case you have forgotten is PI * radius * radius. If we take a circle that has a radius of 1 we get an area of PI * 1 * 1 which simplifies to PI. We will then enclose this in a square that is 2 units wide and 2 units high, such that the circle touches the square on all four sides. The coordinates of the bottom left of the square is (-1, -1) and the top right of the square is (1,1) with the circle centered at (0,0). Random points should be generated between [-1,1] in both x and y. If the random point is less than or equal to a distance of 1 from (0,0) then it will be counted as being in the circle.

We then count up the number of samples that we took (call this n) and also the number of samples that were in the circle (call this x). By dividing x by n we get the percentage of samples in the circle (call this p). If we consider that the ratio of the area of the circle to the area of the square is PI/4 then to get the full value of PI we must multiply p by 4.

This is an example of an embarrassingly parallel problem that is well suited to a distributed system. You should see near linear performance increase in this application when you add more nodes.

Notes:

You have two weeks to do this assignment. Standard penalties will be applied to work that is submitted so much as a second late. The time of submission as displayed by the moodle will be the reference point for lateness.

You must submit a single zip file (naming does not matter) that contains two files: 1 c++ file containing your MPI source code, and 1 PDF file containing the report you write. Anything other than a PDF will not be accepted. (there are tools for converting DOC to PDF available e.g. the foxit reader plugin, those of you using libre office or LaTeX have direct export to PDF)

Code that fails to compile will incur a penalty of 30%. The accepted compression formats for your archives are tar.gz/tar.bz2/tar.xz/zip/rar/7z any format outside of this will incur a 10% penalty.

For the purposes of this assignment you will only need three standard header files <iostream>, <cstdlib> and <mpi.h> for the timing task you may need an extra header or two.

Task List:

01) write a main method that will initialise MPI, figure out the world rank and world size. Rank 0 should be the coordinator while all other ranks should be participants. Then finalise MPI and return a status of 0 to the OS (10%)

02) write a coordinator method that takes in a single variable (the world size). It should run a monte carlo simulation for a set number of samples. Then it should collect the number of hits from all participant notes (you may only use MPI_Recv() here) and calculate the overall hit ratio for the collection of samples. Display the result of PI on the console (25%)

03) write a participant method that does a similar simulation as 02 above but will communicate its total number of hits to the master (you may only use MPI_Send() here) when it is finished (15%)

04) modify your code in such a way that the total number of samples each node produces can be changed by simply changing the value of a global constant (5%)

05) modify your code in such a way that regardless of how many nodes are used (2, 4, 8, 16, etc) your code needs no modification to function correctly (5%)

05) write a report that analyses two things. First show the progress of your algorithm calculating the value of PI in increments of 4 million samples (use 4 nodes and increment your total samples by 1 million each time) and find where your algorithm returns a value of 3.14159. (20%)

06) compare the speed of your algorithm when using a single node, 2 nodes, and 4 nodes in increments of 4 million samples up to 40 million samples and display these results on a graph. What kind of speed up do you get and explain why. (20%)

Assignment 02: Using scatter reduce and broadcast to perform basic statistics on a data set

Introduction:

In this assignment you will be tasked with performing a statistical analysis of a set of numbers. You are required to calculate the mean and then calculate the standard deviation of the set of numbers. However in this case you will be required to generate all numbers on the first node and using the broadcast and scatter commands you will distribute this data to all nodes that are participating.

Once the numbers have been scattered you will be required to calculate an overall mean for the dataset. However, as there are multiple mean values you must use a reduce command to receive those values on a coordinator node. It will calculate the overall mean and once calculated will broadcast this back to all nodes for calculating the standard deviation. The results are then reduced on the coordinator node again to compute an overall standard deviation.

Notes:

You must submit a single zip file (naming does not matter) that contains one c++ file containing your MPI source code. Code that fails to compile will incur a penalty of 30%. The accepted compression formats for your archives are tar.gz/tar.bz2/tar.xz/zip/rar/7z any format outside of this will incur a 10% penalty.

For the purposes of this assignment you will only need four standard header files <iostream>, <cstdlib>, <cmath> and <mpi.h>.

Task List:

01) write a main method that will initialise MPI, figure out the world rank and world size. Rank 0 should be the coordinator while all other ranks should be participants. Then finalise MPI and return a status of 0 to the OS (5%)

02) write a printArray method that will print out an array to console in a single line. It should accept two parameters a pointer to the array and the size of the array. (2%)

03) write a sum method that takes in a reference to an array and an array size it should return the sum of all the values in that array (3%)

04) write a sumDifferences method that takes in a reference to an array, an array size, and the overall mean of the dataset. It should produce a sum of the square of differences between each value in the dataset and the mean and return this as the result (5%)

05) write coordinator and participant methods that do the following (65%)

generate the array of numbers (coordinator only). for predictable results seed the random number generator with the value of 1 and limit their maximum value to 50. (5%)

determine the size of each partition (coordinator only). Broadcast this to all nodes. (10%)

scatter the partitions to each node. (10%)
calculate the mean for this node. Use a reduce operation to gather the overall average. (10%)
Compute the overall average (coordinator only). (5%)
Broadcast the overall average to all nodes and then compute the sum of differences (10%)
reduce the overall sum of differences (10%)
calculate the standard deviation and print out the dataset, mean and standard deviation (coordinator only) (5%)
06) modify your code to work with any world size and accept a dataset size from the command line. You may assume that the dataset size will be evenly divisible by the world size (10%)
07) Do a comparison of four nodes against a single node on dataset of different sizes. Try to find a crossover point where the four node version is faster than the single node version. Produce a graph containing this cross over point. Provide a short one page commentary on what this graph states about your algorithm (10%)

Assignment 03: Using scatter, gather and broadcast to perform parallel matrix multiplication

Introduction:

In this assignment you will be tasked with building a fully working parallel matrix multiplier. Matrix multiplication is a common task in many scientific applications and large matrices take time to compute. However, if divided in the right way the task can be parallelised efficiently.

In this assignment you will need to multiply two matrices together using the stripe method of matrix multiplication. The block method is more efficient but it is recommended that you implement the stripe first before you try the block. You will have three matrices A,B and C that represent the operation A x B = C. You will be required to determine how many nodes are in your compute group and create even stripes for A to be divided amongst the nodes. All nodes will receive a copy of B. Each node will do a matrix multiplication of its stripe of A will the matrix B to generate its stripe of C. Finally all nodes will use the gather operation to send back the stripes to the coordinator node wherein it will be reassembled into a single matrix.

You may assume that all of your matrices are square (NxN) and that N is evenly divisible by the number of nodes. The support file you will get for generating matrices will generate 8x8 matrices and thus will work well with 4 nodes for testing. All matrices must be passed in on the command line along with the number of elements in a row or column e.g. to pass in two matrices you would do something like this

```
mpirun -n 4 ./assignment03 matA.dat matB.dat 8
```

The matrices should only be read by the coordinator process.

Notes:

You must submit a single zip file (naming does not matter) that contains one c++ file containing your MPI source code. Code that fails to compile will incur a penalty of 30%. The accepted compression formats for your archives are tar.gz/tar.bz2/tar.xz/zip/rar/7z any format outside of this will incur a 10% penalty.

For the purposes of this assignment you will only need four standard header files <iostream>, <cstdlib>, <cmath> and <mpi.h>.

Task List:

01) write a main method that will initialise MPI, figure out the world rank and world size. Rank 0 should be the coordinator while all other ranks should be participants. Then finalise MPI and return a status of 0 to the OS (5%)

02) Add the following helper methods to your code (15%)

- printMatrix() will print a 2D matrix to the console
- dotProduct() will that multiply a row of matrix A with a column from matrix B that will return a single value that is the dot product of the row and column
- multiplyStripe() takes in a stripe of A, a matrix of B and computes a stripe of

C.

03) Write coordinator and participant methods that do the following interactions (80%)

- read in matrices A and B from disk. (coordinator only)
- broadcast a message stating that the computation will be performed if the matrices are present and correct and that we have the correct number of command line arguments. Give a different message otherwise
- Take part in multiple broadcasts that will tell all nodes the size of a matrix, the size of a stripe and the size of an individual row.
- Allocate the necessary memory for the stripes and matrices required
- Take part in a scatter to distribute the stripes of A and take part in a broadcast to get the full copy of B
- perform the multiplication and take part in a gather to send all stripes of C back to the coordinator
- print out the matrix (coordinator only) and deallocate all memory.

GRIFFITH COLLEGE DUBLIN

QUALITY AND QUALIFICATIONS IRELAND

EXAMINATION

DISTRIBUTED SYSTEMS

Lecturer(s):

External Examiner(s):

Date: XXXX **Time: XXXXXXX**

THIS PAPER CONSISTS OF FIVE QUESTIONS
FOUR QUESTIONS TO BE ATTEMPTED
ALL QUESTIONS CARRY EQUAL MARKS

APPENDIX AT THE BACK OF THE EXAMINATION PAPER

QUESTION 1

(a) What is the purpose of Message Oriented Middleware, and how does it handle communication in a network? Illustrate and describe the four different models of loosely-coupled communication between nodes in the message queuing model.

(13 marks)

(b) Differentiate between discrete media and continuous media. In what way does transmission of continuous media have more stringent requirements than discrete media with regards to streaming?

(5 marks)

(c) What are the main features of a data stream? Explain the difference between synchronous and asynchronous transmission, providing examples.

(7 marks)

Total (25 marks)

QUESTION 2

(a) Define the main goal of a distributed system.

(5 marks)

(b) Explain with the aid of a diagram the operation of a layered architecture.

(9 marks)

(c) There are several different transparencies associated with distributed systems.
 (i) In relation to a Distributed System, explain what 'transparency' means.
 (ii) Discuss any three different forms of transparency.

(11 marks)

Total (25 marks)

QUESTION 3

(a) In relation to failure handling:
 (i) Explain the concept of redundancy. Include in your answer two approaches to redundancy.
 (ii) Differentiate between a fail-safe and a fail-silent system.

(7 marks)

(b) Write a brief explanatory note on transient, intermittent, and permanent faults. Include an example of each in your solution.

(6 marks)

(c) Briefly outline the four requirements that must be satisfied for a distributed system to be considered fault tolerant and dependable.

(12 marks)

Total (25 marks)

QUESTION 4

(a) Discuss three issues associated with group membership.

(6 marks)

(b) How does the amount of replication differ in a K fault tolerant system with and without Byzantine faults? State any assumptions and relevant formulas.

(7 marks)

(c) With the aid of a diagram describe how a Hierarchical feedback control system works. You should include an explanation of how a message is multicasted in your answer.

(12 marks)

Total (25 marks)

QUESTION 5

(a) The code listed below is the outline of a distributed application. Some of the code is not listed here, the comments explain what would be in the full code. Briefly explain what the code below does.

(10 marks)

(b) What would be better to use instead of MPI_Send and MPI_Recv in this code? It is not necessary to answer with full source code, just indicate any changes that could be made to the current code.

(5 marks)

(c) What would be printed to the console when this application was run on a system that contains 4 nodes?

(5 marks)

(d) Would the output you have given in c) always look exactly as you have provided?

(5 marks)

Total (25 marks)

Below is the C++ source code for program – *nodeMsg*

```
#include <iostream>
#include <mpi.h>

#define MASTER 3
#define SLAVE_TEST 400
#define SLAVE_INFO 401
#define SLAVE_CALC 402

int main(int argc, char **argv) {

 MPI_Init(NULL, NULL);

 int size, rank;
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 int msg;
 if(rank == MASTER) {
  msg = SLAVE_CALC;
  for(int i = 0; i < size; ++i){
            if(i != MASTER){
                    MPI_Send(&msg, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            }
    }
 } else {
  MPI_Recv(&msg, 1, MPI_INT, MASTER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    if(msg == SLAVE_INFO){
            //SELF TEST CODE WOULD BE HERE
            std::cout << "Slave rank" << rank << " INFO" << std::endl;
    }

    if(msg == SLAVE_TEST){
            //SELF TEST CODE WOULD BE HERE -- WE ASSUME NODE IS OK
            std::cout << "Slave rank" << rank << " OK" << std::endl;
    }

    if(msg == SLAVE_CALC){
            //SOME CALCULATION HAPPENS HERE -- PRINT DONE
            std::cout << "Slave rank" << rank << " DONE" << std::endl;
    }
 }

 MPI_Finalize();
 return 0;
}
```