CS4103: Programming practical 1

Assignment: P1 - Communication Middleware & load balancing tasks

Deadline: 13th March 2020

Weighting: 30% of coursework

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical is aimed at designing and implementing a simple distributed system with the help of existing communication middleware.

2 Competencies

- Develop experience in implementation of distributed systems
- Develop and consolidate knowledge of distributed system communication
- Develop familiarity with existing middleware solutions

3 Practical Requirements

This is a programming practical in which you are required to produce a program and an associated report. Your implementation must compile and run on the School lab Machines and it should be possible to run your program from the console/terminal without importing it into an IDE. This document includes a general overview and a set of incremental steps. Please ensure that you complete fully one step before moving to the next step. This practical refers in particular to topics covered in weeks 1, 2, and 3.

3.1 Introduction

Load balancing is an important requirement in distributed systems as "the performance of any system designed to exploit a large number of computers depends upon the balanced distribution of workload across them"[1]. Load may concern the number of requests processed in a web-based system, or the processing workload among heterogenous computing systems. Many load balancing algorithms exist which aim to identify and distribute the load among various systems and are run on components, which are also in charge of routing the request to a more favourable node to maintain balance.

[1] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. Distributed Systems: Concepts and Designs. Pearson Higher Education, 5 edition, 2011.

In this assignment, we consider an abstract distributed task scheduling application inspired by load balancing problems in distributed systems.

Assume that our system provides two nodes $N1, N2$ performing the same task T. Note that a node is an abstraction that can be considered as a server, a service running on a server, or a remote object available on a remote server depending on your choice of implementation. Two parameters are used to invoke the task $\langle ID, Res\rangle$: ID is a unique ID for the task instance for a specific client and Res is a unit which indicates an abstract number of resources needed for the specific instance of the task (e.g., server time, resource cost, job length, data items). We assume also that the task returns a string to the client acknowledging the execution (e.g., "NX: Task ID performed using Res units").

Nodes have infinite resource capacity but the system keeps track of the total number of resources (or current load) $C(NX)$ that have been requested by the tasks executed so far in that node. For example, if the following three tasks have been requested by client K and executed:

- $N1 : T\langle K1, 4\rangle$
- $N2 : T\langle K2, 10\rangle$
- $N1 : T\langle K3, 2\rangle$

The total number of resources for $N1$ is $C(N1) = 6$, and the a total number of resources for $N2$ is $C(N2) = 10$. Our load balancing policy is such that every time a new task is requested, **the request should be handled by the node that has used the least number of resources so far** (the node with $C = min\{C(N1), C(N2)\}$). Hence, in our example the next new task $T\langle K4, 3\rangle$ should be handled by $N1$.

If the client was informed of the communication end points of the two nodes, and of their current load, the client can employ this load balancing policy to place the request to the appropriate node. However, for better transparency a scheduler service could be introduced, which is contacted by the client, decides on which node should be responding and forwards the request.

Your task is to implement a distributed application using a high-level communication middleware that allows a client to make a request for a task $T\langle ID, Res\rangle$ to the system, and receive an appropriate response.

3.2 Part 1: Communication set up

In this first part, we assume that there is a single node $N1$ and a single client that can repeatedly place requests. Implement a distributed system in which the node responds to the request for the task. In order to achieve this, you are required to make use of one type of RPC, RMI or MOM middleware of your choice for communication. Some examples that we covered in class are: Apache Thrift, Java RMI, Web Services, and Zero-MQ, but you may also use an API of your choice provided that you are familiar with it. The chosen middleware should take care of the communication, while you will be required to specify the functionalities. Please do not use Sockets APIs directly but experiment with one of the available higher level approaches.

As starting functionality, provide an infrastructure for the following components:

- A node (a server, service, or remote object) $N1$ able to respond to a request for a single task type
- A client that is able to communicate with the node to place the request

When the infrastructure has been set up, ensure that the following operations can be performed for this first step:

- The client should gather from a user the number of units that the next task requires (Res)

- The client generates a new ID and places the request to the node $N1$ with $T\langle ID, Res\rangle$
- The node returns a string (e.g.,"N1: Task ID performed using Res units").

3.3 Part 2: Two nodes and load balancing

In this part, you should implement an additional node $N2$ capable of handling similar tasks as $N1$, invoked in the same way through $T\langle ID, Res\rangle$. Note that this second node must not be an additional instantiation of the previous node, but its own component (additional server or service, or different remote objects). You are then required to develop a load balancing approach as described in the introduction, where the node which has lower use of resources should respond to the call. For this part, you should design and implement **a single solution** choosing among various approaches. Two guideline solutions are proposed here, but you can design your own version.

A simple solution can be achieved by balancing the load at the client side, although providing little transparency to the existence of different nodes. This means that the client knows how to communicate directly with the two nodes, keeps track of information on the current load, and decides which node to invoke. In this solution, we expect the following functionalities:

- Two nodes able to respond to requests for a task as above
- A client that is able to communicate with each of the nodes
- The client gathers the resource requirements as in part 1
- In addition, at each new request to be placed, the client should update the resource usage of that node $C(NX) = C(NX) + Res$
- Before placing the task request, the client should check which one is the next node to be invoked
- The client invokes the corresponding node which acknowledges the execution

A more transparent approach would transfer the load balancing to an additional scheduler node. This means that the client initiates requests in the same way as in part 1, and the scheduler receiving the new request decides which node to invoke. In this solution, we expect the following functionalities:

- Two nodes able to respond to requests for a task as above
- A client that is able to communicate with the scheduler
- The client gathers the resource requirements and places requests as in part 1
- The scheduler receives the new request
- Before placing the task request, the scheduler should check which one is the next node to be invoked
- In addition, at each new request to be placed the scheduler should update the resource usage of that node $C(NX) = C(NX) + Res$
- The scheduler invokes the corresponding node, the client is informed of the execution

3.4 Additional Extended Functionalities

It is strongly recommended that you ensure you have completed the Requirements in part 1 and 2 before attempting any of these additional requirements. You may wish to consider **one** of the following additional tasks, describing well the purpose of your extension in the report. Please ensure that the code for extended functionalities is separated from the previous parts or clearly marked.

1. **Many task types:** In part 2, we have assumed that nodes provide capabilities to execute a single type of task T. Develop a solution in which a client can choose among many different types of tasks which might need different balancing strategies.
2. **Resource cap:** In part 2, we have assumed that there is an infinite provision of resources. Modify your solution such that each of the nodes has a limited amount of resources available, each task being performed reduces the availability of the resource for a specific amount of time. After this time the resources will become available once again. The scheduler should consider the case in which there are not sufficient resources for a task to be performed and deny the request.
3. **Load delegation:** In part 2, we have assumed that nodes performing tasks do not communicate or transfer load. Provide a solution that allows a node to transfer the load to the other node in case a condition of your choice is verified.
4. **Other options:** Suggest and develop your own additional functionality. New requirements should consider ways to extend the functionalities of the nodes. If you are unsure about your proposed extension, please check with the lecturer.

4 Code and Report Requirements

4.1 Code and Libraries

Please develop your program selecting and testing a communication middleware of your choice. You may use a programming language of your choice provided it is compatible with the middleware. Please include clear instructions on how to compile (if required) and run your system. You are free to use other libraries as long as they are used for functionalities that are not core to this assignment.

The solution submitted should contain all source code that you have developed. For certain types of middleware, you might need some external libraries or components, which may be excluded from the submission but please include clear instructions on how to set up your system including any requirements. You may use automatic build scripts – eg. Maven.

Additionally, you may submit a single solution for both parts, 1 and 2, depending on what is attempted. If you attempt any of the additional requirements, please ensure that the main solution can be tested, and that the extended approach is clearly marked/commented.

4.2 Report

You are required to submit a report describing your submission. The report should include:

- A brief list of the functionalities implemented and any extension attempted
- If any of the functionalities is only partially working, ensure that this is discussed in your report

- Design: A description and justification for the mechanisms you have implemented as part of your solution
- Examples and Testing: Some selected examples of the main functionalities (screenshots are encouraged) and any tests performed
- Evaluation: A critical evaluation of the functionalities of your system, with **particular attention to reflect on the type of middleware used and the characteristics it provides**
- Running: Include clear instructions on how to compile (if required) and run your system

A suggested length for the report is around 800-1000 words.

5 Deliverables

You should submit a ZIP file via MMS by the deadline containing:

- A PDF report as discussed in Section 4.2.
- Your implementation of the solution including any source code.
- Any additional script required to run your system.

6 Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in the next section). The following issues will be considered:

- Achieved requirements
- Quality of the code and the solution provided
- Example runs
- Insights demonstrated in the report

Some guideline descriptors for this assignment are given below:

8-10	The submission implements some of the functionalities of part 1, adequately documented or reported.
11-13	The submission implements fully the functionalities of part 1. The code submitted is of an acceptable standard, and the report describes clearly what was done, with good style.
14-16	The submission implements fully the functionalities of part 1 and some solution to part 2. A minimum requirement for this band is to have some form of communication with two nodes; for upper parts of this band, the solution should also consider good transparency provision. The code submitted is clear and well-structured, and the report is clear showing a good level of understanding.
17-18	The submission implements implements fully the functionalities of part 1 and an excellent solution to part 2. It contains clear, well-designed code, together with a clear, insightful and well-written report, showing in-depth understanding and analysis of the middleware used and the solutions presented.
19-20	The submission implements implements fully the functionalities of part 1, an excellent solution to part 2 and one extended functionality. The submission demonstrates unusual clarity of design and implementation, together with an outstandingly well-written report showing evidence of extensive background reading, a full knowledge of the subject and insight into the problem.

7 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Alice Toniolo
(a.toniolo@st-andrews.ac.uk)
February 15, 2020