School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2021

Project Part 1: Distributing an Application

Due date: October 4 @ 6PM, Demo date October 5/6/7

This project is an adaption of the project of CSE 593 of the University of Washington.

Your task in this project is to develop a component-based distributed information system using some of the fundamental components and algorithms for distribution, coordination, scalability, atomicity, etc. shown in class. The ultimate goal is a cohesive multi-client, multi-server implementation of a **Travel Reservation** system.

There is no late turn-in for projects.

Question 1: Implementation (80 Points)

The project this semester is a Travel Reservation system, where customers can reserve flights, cars and rooms for their vacation. You can find more information about the client interface and explore the functionality in the `UserGuide.pdf`.

The functionality of the management system is very simple, but it will make the project easier. We make the following simplifications:

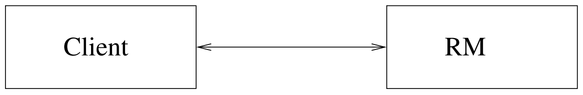
1. There are only 3 types of reservable items (key denoted in *italics*):
   (a) **Flights:** *flight number*, price, and the number of seats
   (b) **Cars:** *location*, price, and the amount available
   (c) **Rooms:** *location*, price, and the amount available

   Note that since location is used as a key, there is only one type of car/room (and only one price) for a particular location.
2. Adding an duplicate item updates the price (if greater than zero) for all already existing items, and increases the count
3. Each customer maintains a list of their reserved items
4. Querying a customer returns the list of reserved items, as well as the total cost to the customer
5. Deleting a customer also cancels all reservations they performed

Beginning with the starter code provided on myCourses, the first milestone requires you to distribute the simple application in two ways: RMI and sockets. Note that the current implementation ignores the *xid* parameter – it will be used for transaction identifiers in later milestones.

RMI Distribution

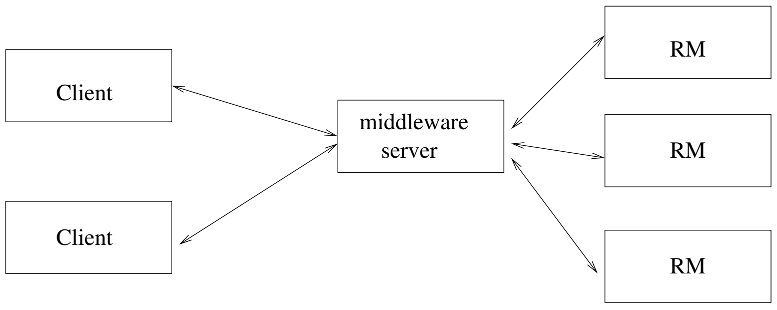
The provided sample code already implements a single-client, single-server system using RMI for communication. On the client side, an interactive console takes text-based commands (see `UserGuide.pdf`), parses the input and sends a message over RMI to the `ResourceManager`. The `ResourceManager` receives the command and modifies its local state accordingly. Note that in this implementation, a single `ResourceManager` is responsible for resources of all types.



The first task in this milestone involves distributing the system *without* modifying the client.

1. Introduce a new intermediary server called the `Middleware` that sits between the client and the `ResourceManagers`. On startup it should be given the list of `ResourceManagers` and implements the same interface as the `ResourceManager`
2. Create `ResourceManagers` for each type of resource (one for each of flights, cars, and rooms). For simplicity, you can keep only one implementation providing the entire interface – the `Middleware` will only call the flight-methods on the flight-`ResourceManager`
3. Decide how to handle customers – either with an additional server, through replication at the `ResourceManagers`, or at the `Middleware` (which then becomes a type of `ResourceManager`)
4. Add an additional function `bundle` to the `Middleware` which reserves a set of flights, and possibly a car and/or room at the final destination

All clients send requests to the `Middleware` server which are then distributed appropriately.



TCP Sockets

Re-implement your system using TCP sockets for communication between all layers instead of RMI. The client can remain blocking (i.e. it sends a request and waits for the reply before sending the next request). However, the `Middleware` must not block when it is waiting for the `ResourceManagers` to execute a request. That is, when the `Middleware` receives a request from the client, it forwards it to the corresponding `ResourceManager` and continues accepting client requests. When it receives a response from the `ResourceManager`, it messages the appropriate client. Similarly, the `ResourceManagers` should be able to handle several requests concurrently.

Note: The above description is short, but expect it to be more time consuming than RMI.

Question 2: Custom Functionality (10 Points)

The final part of the first milestone requires choosing 1 of 3 additional functionalities depending on your strengths and interests (ordered from hard to easy). You may also email the TA with any other ideas you may have for approval — use your creativity!

- Build an automated unit-test client that sends and verifies the result of a small subset of commands. You can use the `query` commands to verify the correct action was taken.
- Research how Java lambda functions/serialization/reflection can be used to simplify TCP distribution and remote function invocation. Implement an approach using them and explain in your report.
- Add extra functionality to your client: analytics (e.g. items with low remaining quantities), overall resource/customer summaries (e.g. who has reserved an item and at what price)

Question 3: Report (10 Points)

Write a short (2 page, 2.5 pages **maximum**) report detailing your architecture and design for both RMI and TCP. The TCP section will likely dominate the report, and should explain your strategy for message passing as well as concurrency. Briefly mention your implementation choices for customers and bundling, and explain your custom functionality.

Your report should also contain a small paragraph at the end as to the contribution of each of the group members. Please note that I do not consider 'report writing', 'manual testing', 'discussed ideas' by themselves as a significant contribution. If we notice a patter of certain members not contributing consistently, they may receive a lower grade letter at the end of the course.

The report is due by **6 PM on Sunday, October 4th** to give the TA enough time to read before the demos.

Demos

To grade your implementation we will use online demonstrations with the TA where you show your running system. It is recommended that your demonstration include a very short (less than 4 slides) presentation with highlights of your system and your implementation strategies. Not being prepared with slides will result in significant point deduction. They should be distinct slides and not "scrolling through the report".

The demonstrations of all groups will take place over three days (October 5/6/7). A sign-up sheet will be put in place so that you can reserve a time-slot. Show up early and have your system running, using at least 5 different machines for your client and servers. Expect questions! Ensure the person who is doing the demo is efficient with Unix command prompt. Practice a bit on your own before showing up for the demo. Questions can be directed at anyone and not just the person who is executing the commands. **ALL team members must be present for the demo. Absentees will not receive the grade.**

All code is due on MyCourses by the time of your demo.