Project 1 - Distributed Flight Reservation System

CSCI-4510/6510 – Fall 2019

Assigned: Sep. 24, 2019
Teams formed by: Oct. 1, 2019 (10% penalty for missing this deadline).
Work in groups of two.
You must pair up with someone who is enrolled in the same course number as you.
Docker Package Notification Deadline: October 13, 2019
Project due: Sunday, Oct. 20, 2019 at 11:00pm
Demos: Oct. 21 – Oct. 25, 2019

1. Summary

For this project, you will implement a distributed flight reservation system using the Wuu-Bernstein replicated log and dictionary algorithm. The reservation system is for Tiny Airlines. This airline provides 20 flights, with flight numbers 1, 2, 3, ... , 20, and each flight has seats for only 2 passengers.

The distributed system consists of N sites; each site corresponds to a single reservation agent. Each site is identified by a `site_id`, a short alphanumeric string. An agent can make reservations for its clients, and it can cancel these reservations. Each client has a unique `client_name`, a short alphanumeric string. Each client will make at most one reservation.

Each agent is also responsible for informing other agents about reservations made at that site. Since the Wuu-Bernstein algorithm cannot guarantee that all agents have a copy of every reservation immediately after it was made, it is possible that multiple clients may try to reserve seats on full flights. To address this, a flight reservation should have a `pending` status, until it is guaranteed that a seat is reserved (and will not be given away to another client). At this point, the reservation becomes `confirmed`.

UI for managing Reservations:

(a) To make a reservation, the user enters the following:

```
% reserve <client_name> <CSV_list_of_flight_numbers>
```

If there are no conflicts in the local copy of the reservation list, the system should submit the reservation (see below) and print out:

```
% Reservation submitted for <client_name>.
```

Otherwise, the application should print:

```
% Cannot schedule reservation for <client_name>.
```

(b) A client can cancel their flight reservation by entering:

```
% cancel <client_name>
```

After the system cancels the reservation, it should print:

```
% Reservation for <client_name> cancelled.
```

(c) Ths system should display the list of all pending and confirmed reservations using the following command:

```
%  view
```

In response to this command, the system should list all reservations in the site's copy of the reservation list, sorted in lexicographical order by *`client_name`*, in the following format:

```
userA 1,3 confirmed
userB 1 pending
userC 2,4 confirmed
```

3. Log and Dictionary Specification

The set of pending and confirmed reservations should be stored in the dictionary. A reservation consists of the following fields: (`client_name, list_of_flight_numbers, status`), where status is either `pending` or `confirmed`. When a site creates a reservation, a reservation entry is inserted into the site's copy of the dictionary, with a status of `pending`. When the site knows that the seat for that reservation is guaranteed, the status of the entry should be updated to `confirmed`. When a site cancels a reservation, this reservation entry should be removed from the site's copy of the dictionary. Each site should also store a (partial) log of insert and delete events, as specified by the Wuu-Bernstein algorithm.

The `view` command (described above) shows the current contents of a site's dictionary. The system should also support the following command to display the contents of the log:

```
%  log
```

In response to this command, the system should list the log contents, sorted in order that the events were added to the log, in the following format:

```
insert userA 1,3
insert userB 1
delete userC
```

To replicate log contents, the system should provide two commands that send messages to other sites. These messages should contain only those log entries that are identified as necessary by the Wuu-Bernstein algorithm.

To send a message to a single site, the following command is entered:

```
% send <site_id>
```

To send a message to all other sites, the following command is entered:

```
% sendall
```

Finally, your system should support the following command to view the matrix clock:

```
% clock
```

In response to this command, the system should print out the matrix clock entries, as integers, in the following format (e.g., for N = 3):

```
1 0 0
0 1 2
0 0 2
```

Additional Requirement for Students in CSCI-6510: You must also implement the "Strategy 1" message size reduction feature described in the Wuu-Bernstein paper. The command to send a small message to a single site is "`smallsend` *`<site_id>`*" and the command to send a small message to all sites is "`smallsendall`".

4. Implementation Details

When an agent submits a reservation, the application should check the local copy of the reservation list (dictionary) to see if there are seats available on all flights in the reservation. If so, a reservation should be created with status pending. Otherwise, the reservation should not be created. When the reservation is created, the entry should be created in the site's dictionary and added to the site's log. Similarly, when an agent cancels a reservation, the entry should be removed from the site's dictionary and a delete event should be added to the log.

In the Wuu-Bernstein algorithm, a site may not always have the most up-to-date information about the other sites' logs and dictionaries. Therefore, the algorithm described above may result in more reservations than there are seats on a particular flight. Conflicts should be resolved in "happens before", order. If multiple "concurrent" reservations result in a conflict, then ties should be broken by assigning the seat(s) in lexicographical order of `client_name`.

When a site detects a reservation conflict, it should use the rules described above to determine which reservation(s) to cancel. It should then automatically cancel the reservation(s) (i.e., remove reservation(s) from its dictionary and create delete event(s) in its log).

When a site detects that there is no longer any possibility that a reservation may be cancelled due to a conflict, the site should update the status of that reservation as `confirmed` in its dictionary. A site does not need to inform other sites that a reservation is confirmed.

Your application should truncate logs and reduce message sizes as specified by the Wuu-Bernstein algorithm. It should not send any additional messages other than those described in this specification.

- You must implement your project using either Java or Python.
- You must use UDP sockets for network communication.
- When creating sockets/sending UDP packets, you must use the IP address of the destination host. Since Submitty uses a special network configuration, addressing by hostname may not work.
- You may not use any libraries that provide "server" functionality, e.g., an HTTP server library.

5. Source Code Management and Build Requirements

- You must store all of your source code in the Submitty Git repository (URL provided to your team in Submitty). When you submit your project, we will pull the code from this repository.
- We will run several autograding tests in Submitty. For the autograder to build your project, we need you to follow a few simple rules for the structure and compilation of the project.
    1. You must provide a `build.sh` bash script at the root of your repository. This script should create a `bin` directory (at the root of the repository) that contains all files needed to run your project. For example, if you are using Java, you should put your class files in the bin directory.
    2. Your `bin` directory must also contain a `run.sh` bash script that starts your application at a single site. This script accepts a single argument, the `site_id`.
- When we test your application, Submitty will generate an `knownhosts.json` file. This file will be copied into the root of your `bin` directory before `run.sh` is invoked. The file provides the IP address and port range for each site (host). An example file is posted on the course website.

6. Deliverables and Grading

(a) To submit your project source code, you must follow the Submission steps in Submitty. This will check out the project files from your Git repository and copy them to Submitty for grading. Submitty will execute several simple auto-grading tests to verify that your application builds and runs in the Submitty environment and that your application meets the UI specification. These tests will be worth 15% of the project grade.

(b) Each team must also submit a 2 page project report. This report must be single-spaced with 1 inch margins. Font size must be 12pt or smaller. The report can include diagrams. The report should provide the following information:

- Details of your project design and implementation. This includes descriptions of your data structures, stable storage implementation, use of sockets, and use of threads.
- A description of how your system determines which "events" to include on each message for both the `send <site_id>` and `sendall` commands.
- A description of your method for determining when a pending reservation can be confirmed.
- A description of how you tested your project (be sure to consider failures, recovery, and message loss).
- A description of how the project work was divided among the group members.

For students in CSCI-6510, the report should also discuss your implementation of the message size reduction feature.

Project reports must be submitted in Gradescope by the project deadline. A substandard report can result in up to a 10% point deduction on your project grade.

(c) The majority of your project grade will be based on a live demonstration run on Submitty, where we will execute several use cases and observe the results. We will evaluate how your project responds to site crashes, site recoveries, and message loss in the demo.
We will schedule demonstrations closer to the project due date.

<u>If your project does not run on Submitty, you will receive a score of 0.</u>
<u>No late submissions will be accepted.</u>