CSCI 6510 Project 1 Report
Ming Lu, Yujue Wang

I. File Structure

We implemented a distributed flight reservation system by Java and it includes 6 files:

1. Host.java:
    - *Implement the JSON input and user interface*
    - *Implement multithreaded UDP server function*
2. ReservationSys.java:
    - *Keep local information on dictionary, log, and timetable. Implement insert, delete and update using the Wuu-Berstein algorithm.*
3. Listener.java:
    - *Implement the receiving function of UDP server*
4. Reservation.java
5. CommunicateInfo.java
6. EventRecord.java

II. Project Design

- Data structures:

1. EventRecord (Reservation reservation, String operation, String siteId, int siteTimestamp)
   -Used to record the details, operation, and timestamp of the event.
2. CommunicationInfo (ArrayList<EventRecord> eventRecords, Integer[][] timeTable, boolean smallFlag, Integer[] timeRow)
   -Used to encapsulate the communication message.
3. Reservation (String clientName, ArrayList<Integer> flights, String status)
   -Used to record each reservation information.
4. ReservationSys (ArrayList<Reservation> dict, ArrayList<EventRecord> log, Integer[][] timeTable, Integer siteTimeStamp, String siteId)
   -Used to construct the site and record the status and information of the site

- Stable storage: to deal with site crash or failure, the log, dictionary, and timestamp of each site are saved to 3 different txt files, “log.txt”, “dictionary.txt” and “timetable.txt” in local storage.
  Upon updating log, dictionary or timestamp will be saved to the txt file, which lets the backup file be the latest information.
  When a site is crashed, and the Reservation system detects all 3 txt files. It reads data from them to initialize the site so that the site will keep the same information as before the crash.

- Sockets and Threads: We implemented UDP send in the main thread and receive in a child thread to make sure a concurrent operation, in order to avoid the conflict of concurrent send and receive behaviors.

III. Implementation

- Determine the events to send:

1. For <send site_id>: we check the specific row of matrix timestamp of the current site to see whether the current site has knowledge that the target site knows about a certain event. If the target site doesn't know about the event, the current site will send the event records.
2. For <sendall>: Since a message will be sent to all sites and cover the unknown event records of all other sites, we check the whole timestamp except the row of the current site. If a certain site is unaware of an event record, we check whether the send message has contained this event record(it is possible different sites are unaware of the same event records) and decide whether to add it in the message.

- Updating local information

1. Insert: on receiving the "reserve" command from the local site, the insert method will be called. We first check if the new record is a conflict with local reservations. After making sure there is no conflict, we update local log, dictionary, and timetable about this event.
2. Delete: on receiving "cancel" command from the local site, the delete method will be called. We delete information on the denoted reservations in local log, dictionary, and timetable.
3. Update: upon receiving messages from other sites, the update method will be called. We process each event record in the message, to update the local log, dictionary, and timetable accordingly. If an event's operation is "insert", we also need to make sure there is no conflict with reservations in the local dictionary. When there is conflict, we use conflict solution to decide which record should be kept.

- Local event conflict solution: When the current site receive the conflict records, we collect all conflict event records from other sites and the current site.
  Then, we sort these events by timestamp firstly and by lexicographical order secondly. The first two event records after sorting will be taken as the valid results and rest of the event records will be deleted.

- Local event confirmation solution: The event record is inserted into the dictionary and aware by all other sites will be confirmed.

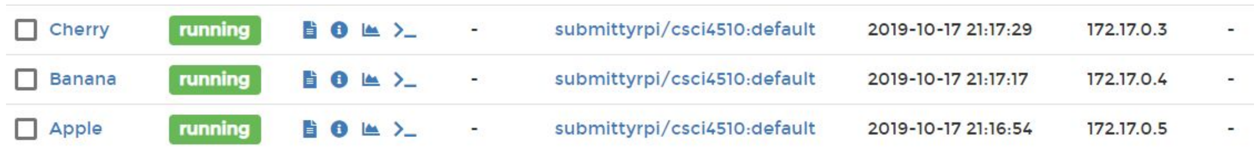
- Implementation of the message size reduction feature:

1. Truncating log
   After updating local record according to every message received from other sites, we check every event record saved in log, and truncate records that are aware by all other sites, by looking up the local timetable.

2. Implementing strategy 1
   As for <smallsend> and <smallsendall>, we only send the row of the timestamp that represents the current site to the target site and event record selection is the same as

<send> and <sendall>. Since the timestamp of site is only updated by the direct message, the site may know less than before and will send more event records than before.

IV. Test



☐	Cherry	running	-	submittyrpi/csci4510:default	2019-10-17 21:17:29	172.17.0.3	-
☐	Banana	running	-	submittyrpi/csci4510:default	2019-10-17 21:17:17	172.17.0.4	-
☐	Apple	running	-	submittyrpi/csci4510:default	2019-10-17 21:16:54	172.17.0.5	-

Test Environment: We used Docker to deploy 3 sites and networks and implemented a test case involving in reservation conflict handling, log truncate and site crash and recovery.

1. Let 3 different clients reserve the same flight at 3 sites
   [apple] reserve a 1 [banana] reserve b 1 [cherry] reserve c 1
2. Let [apple] and [cherry] implement <sendall>, which leads to the conflict at the [banana] since events a, b, and c are for the same flight.
   [cherry] sendall [apple] sendall
3. Check [banana] by <view> and <log> to see whether the conflicted event c from [cherry] is in the dictionary and the log contains a delete event.
   [banana] view [banana] log
4. Let [banana] send back to [apple], check <view> on [apple] to see whether the a has been confirmed and c from [cherry] has been deleted.
   [banana] send apple [apple] view
5. Ctrl+C forced [apple] to exit(site crash) and rerun the [apple], which will recover from the txt files, and then check <clock>, <view>, and <log> to see whether same as before crash.

V. Contribution

- Ming Lu: JSON part, data structures design, algorithm implementation(how to update the dictionary, how to truncate log, how to detect conflicts), write the report
- Yujue Wang: UI for managing reservations, UDP send and receive, docker deployment and test, strategy 1 implementation, write the report