Java-based Distributed Airport System Simulation

With Null Message and YAWNS

Rojin Aliehyaei
School of Computational Science and Engineering
College of Computing, Georgia Institute of Technology
Atlanta, Georgia USA 30332
rojin@gatech.edu

Abstract—**In this work, we modeled and simulated an airport system with 64 airports and 2,560 airplanes for parallel execution in distributed environment using OpenMPI. The simulated events focused on airport operations (i.e., arrival, landing, taxiing, unloading, maintenance, departing, and taking-off). To support synchronization between processors in the distributed simulation, we implemented the Null Message and YAWNS algorithms. The results showed both algorithms successfully ensured all events were processed in the causality order. The average number of events processed per second under the Null Message algorithm is about two times larger than under the YAWNS algorithm. None of the distributed versions outperformed the sequential version on the number of events processed per second. A possible explanation is the small computation to communication ratio in the current implementation where one processor simulates one airport. By expanding the simulation to handle more than one airport per processor and to handle a much larger number of airports, the throughput of distributed simulation should improve, possibly outperforming the sequential version.**

Keywords—analytic simulation; airport system; conservative synchronization algorithm; null message; YAWNS

I. INTRODUCTION

The airport simulator enables us to develop new policies and plans that ensure the air traffic system will work effectively in an unexpected scenario or a new environment. As the span of airport system under investigation is getting larger, we may need to distribute the computation workload between multiple processors to reduce the execution time. We may also need to integrate simulators running on various platforms. This requires a switching from a typical sequential simulation to parallel and distributed simulation systems [1].

There are over 40,000 airports around the world [2]. To simulate such a large network of airports with high fidelity, we need to deploy a large number of states and events. As the number of objects and events in the simulation grows to accommodate a larger network of airports and a realistic number of airplanes, an implementation of a distributed version of simulation program for parallel execution is essential. This work focuses on the implementation of distributed airport simulation program with Null Message [3] and YAWNS [4] algorithms in Java programming language with OpenMPI [5]. Specifically, in the distributed version of the airport simulation program, each airport is modeled as a logical process (LP) and the interaction (e.g. flights) between two airports is simulated by exchanging time stamped messages. The airport events, such as landing, taxiing, unloading, maintenance, and departing, are simulated/executed by each individual LP simulating the airport. When a take-off event from an airport on one LP is completed, it will result in a message being sent to (and a schedule of a new arrival event at) another LP representing the destination airport. Due to this dependency between airport events, we need to define a protocol to avoid deadlock situation while preserving the causality order in the distributed version. The Null Message and YAWNS algorithms were implemented in this work to address this issue.

Section II provided a summary of related work on airport system simulation and on synchronization in large-scale discrete event simulation. Section III presented the conceptual model of the airport simulation, synchronization algorithms, and their implementation in the simulation program. Section IV described the simulation settings and resources as well as displayed and discussed the results collected from running the simulation. Finally, Section V concluded this study and provided possible directions for future work.

II. LITERATURE REVIEW

This section summarizes the related work on airport system simulation and on synchronization algorithms for the distributed discrete event simulation.

A. Airport System Simulation

The airport simulation system, can be classified into two main categories: (i) Analytic Simulation and (ii) Virtual Environment Simulation [1]. Analytic simulation of airport system focuses on quantitative analysis of airport model whereas virtual environment simulation focuses on the realistic modeling of the airport system and its components. Specifically, the analytic simulation models and analyzes the queues, resources, and flight schedule to optimize the operational rules, cost, and resource management policies. In particular, Joustra and van Dijk [6] modeled and simulated the check-in counters at Amsterdam Schiphol airport to devise an operational check-in rule. Their study performed queue and capacity analyses of passenger’s arrival pattern, waiting time, and staff workload. Similar studies by Doshi and Moriyama [7] as well as Smith and Nelson [8] aimed to predict passenger’s arrival pattern and to measure waiting time at the check-in counters and security checkpoints. In [9], Erzberger et al. developed and simulated analytical models of an airport to improve its flight-scheduling algorithm with an aim to minimize the delays and to spread the air-traffic peaks. Bubalo and Daduna [10] also simulated the airport capacity and demand, runway operations, and flight schedules. Their study focused on the practical limitations of the airport system, such as the long waiting time, congestion, and resulting costs.

Apart from analytic simulation studies focusing on the passenger side of the airport system, a number of past researches focused on improving air cargo operation and runway design. In particular, Chen [11] modeled a framework to estimate the runway capacity of more than 2,000 airports in the National Airspace System. In another report by TAC Corporation [12], the airfields, airplanes, and runways at the Oakland International Airport were modeled. A number of suggestions were made including building new parallel and high-speed taxiways, which allow the airplanes to clear the runway efficiently. For all suggestions, the corresponding scenarios were simulated. The ground travel time and the queue delays of the runway were measured and analyzed in all scenarios. Finally, Piera et al. [13] focused on improving air cargo operations of passenger airplanes, such as loading, positioning of high-loader, refilling, and positioning of catering truck, to minimize the cost and speed-up the process.

In the early studies of virtual environment simulation, a three dimensional model of environment and passenger flows were simulated. Specifically, Crook [14] presented a detailed simulation of terminals to provide information about passenger movement at the gates, entry points, exits, and queues as well as processes at security and passport control points. NASA developed Virtual Airspace Modeling and Simulation with an aim to address various issues, such as fundamental changes in trajectories, terminal operations, and optimizing traffic management [15]. Koch [16] created a 3D visualization tool to help the security administration evaluating equipment and procedures of the airport in terms of security. There are also virtual reality simulation that helps pilots and air traffic controllers [17]. For example, virtual signs and bars were added to the air traffic controller screen (called virtual block control) with an aim to improve aircrew's performance under the low visibility conditions. The effectiveness of virtual block control was evaluated with the help of air traffic controllers and pilots in several simulation scenarios. Finally, there were also a number of studies focused on the simulation of cockpit. In [18], the fidelity and coherence of real-time data in the simulated cockpit equipment were tested. Karikawa et al. [19] also developed a human-machine system simulation of cockpit interface for enhancing the quality and safety of flights.

This study performs analytic simulation of an airport system with 64 airports and 2,560 airplanes under various scenarios.

B. *Synchronization in Distributed Discrete Event Simulation*

This section reviews several papers and researches that studied the synchronization algorithms for the large-scale simulation system. There are two main approaches for synchronization in distributed simulation system including: (i) conservative and (ii) optimistic approaches. The conservative approach ensures all processors executing the events in the causality order. The optimistic approach is more aggressive and initially allows each logical processor to execute the events independently. However, when an out-of-order execution happens, the system will detect it and resolve the error automatically [20]. Two important conservative approaches are: (i) Null Message algorithm (a.k.a. Chandy/Misra/Bryant or CMB algorithm) [3, 21] and (ii) YAWNS algorithm [4]. These approaches are based on the concept called Lookahead time. In the Null Message algorithm, we can associated the link between processors with a lookahead time, which helps defining a lower bound for timestamps of event that a simulator may receive in the future. On the other hand, the YAWNS algorithm uses the global synchronization mechanism [22]. Events before the global barrier is safe to execute. An important example of optimistic approach is the Time Warp algorithm [23]. Time Warp is less concerned with synchronization, but it needs to include rollback features for the simulation to cope with errors.

Researchers have developed several large-scale discrete simulation systems that serve as a platform for experimenting with different synchronization protocols, such as the Universal Simulation Engine (USE) [24], the Rensselaer's Optimistic Simulation System (ROSS) [25], and the ns-3 [26], among others. In a part of the study by Swenson [27], the performance of different conservative synchronization mechanisms was measured in terms of runtime and memory usage under various network configurations on the ns-3 simulator. Improvements on the implementation of the Null Message algorithm significantly decreased the number of required Null message exchanges between the LPs.

Another important issue related to the design of a large-scale discrete event simulation system is energy consumption. A comparative study by Fujimoto and Biswas [28] showed the energy consumption of sequential simulator is significantly less than its distributed version. The study evaluated the energy consumption of sequential algorithm as well as distributed versions including the Null Message and YAWNS algorithms.

In this study, distributed versions of an airport system simulation were implemented. The distributed airport system simulation program can be executed with either the Null Message or the YAWNS synchronization algorithm.

III. MODELING AND SIMULATION

This section presents a conceptual model for analytic simulation of airport system, simulation program, and its implementation for both sequential and distributed versions.

A. *Conceptual Model*

The conceptual model includes objectives, assumptions, simplifications, inputs, entities, and outputs. The objectives of this study are: (a) to simulate a basic model of air traffic between multiple airports in distributed environment and (b) to collect and analyze statistics on the total number of events processed per second.

The assumptions and simplifications carried over from our previous version of the airport system simulation are listed as follows:

- Airplanes are not permitted to land and/or take-off at the same time on the same runway.
- There are neither flight schedules nor a list of destinations for an airplane. The next destination airport is selected randomly.
- The number of passengers on board each flight is also generated randomly with a lower bound being at half the airplane capacity and an upper bound being the seating capacity of each airplane model.
- The amount of time for completing each scheduled event is simulated in minutes.
- There are two runways at each airport.
- There are two separate queues for landing and taking-off events. An event with the smallest time stamp is scheduled first. In case of an equal time stamps, the landing event is scheduled before the taking off event.
- Simultaneous taking-off and/or landing events are only permitted on different runways.
- If an airplane accumulated flight hours exceeds the threshold set by the FAA, then the airplane is scheduled for a maintenance check. After completing the maintenance check its accumulated flight hours is reset to zero and the inspected airplane is scheduled for the next departure.
- The airport events are consisted of airplane arriving, landing, taxiing, unloading, maintenance, departing, and taking-off. These events are defined as follows:

➢ Arriving: Arrival of a new airplane at the airport.

➢ Landing: Airplane has completed the landing.

➢ Taxiing: Airplane has completed the taxiing.

➢ Unloading: Airplane has finished unloading passengers.

➢ Maintenance: Airplane has gone through maintenance.

➢ Departing: Airplane has completed the departure process.

➢ Taking-Off: Airplane has reached required altitude.

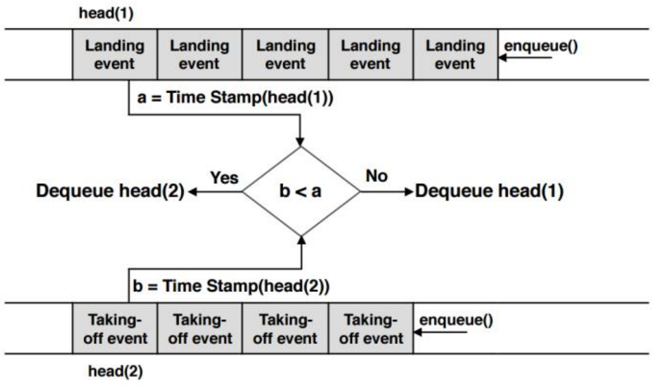


Fig. 1. A snapshot of taking-off and landing queues at an airport

Figure 1 shows a snapshot of taking-off and landing queues at each airport. An airplane is added to the landing queue when it is ready for landing but no runways are available. Similarly, an airplane is added to the take-off queue when it is ready to take-off but no runways are available.

Airplane taxiing is the movement of an airplane on the taxiways from location to another. The taxiing event is scheduled after the landing process is completed [29]. Once the taxiing event is completed, the passengers unloading event is scheduled. The passengers are deplaned during this unloading event. Before the unloading event is completed, the accumulated flight hours of an airplane since its last maintenance is checked. If the airplane is scheduled to fly to the next destination requiring a flight time that will lead to an accumulated flight hours exceeding the maximum threshold defined by the FAA, the routine airplane maintenance event is scheduled (instead of the airplane departure event). There are several types of airplane periodic maintenance check, called Type A, B, C, and D checks, routinely carried out by the airliners [30]. The major requirement enforced by the Federal Aviation Administration is the Type A check, which is done at every 65 accumulated flight hours.

The simulation inputs include the number of airports (64), the number of airplanes (2,560), the speed and capacity of airplanes, the distance between each pair of airports, the amount of time required for each routine airport operation, and the termination time of the simulation. The entities in this simulation are the airplanes and the airports. The airports provide the services while the airplanes are served by the system. A runway at each airport is also considered as a resource entity. The simulation outputs include the total number of passengers arriving and departing, the circling time for landing, the waiting time for taking-off, the total execution time of simulation program, the total number of events processed, and the number of events processed per second.

B. *Simulation Program*

The simulation program in this study is an object-based system written in Java. There are multiple classes, such as Airport, Airplane, Simulation Engine, Event (which includes Simulator Event and Airport Event), and Simulator. Only one instance of simulation engine is created on each LP. The simulator event is the stop event, which accepts a time value indicating when the simulation will terminate. The Airport class handles the airport events. A pseudo code for each airport event is shown in Figures 2 to 7. The constants, other values, and state variables used in these pseudo codes are defined as follows.

Constants and other values:

- Now: The current simulation time
- R: The amount of time for landing an airplane on runway
- TaxiingTime: The amount of time required for taxiing
- UnloadingTime: The amount of time required to unload passengers
- G: The amount of time required at the gate
- T: The amount of time required for taking off
- F: The amount of time required to fly between airports
- MTC_Cycle: The number of accumulated flight hours allowed between each routine maintenance event of an airplane.

- MTC: The amount of time required for completing maintenance work.

State variables:

- Runway1_Free: A Boolean variable; TRUE if runway 1 is available, otherwise FALSE; Initialized to TRUE.
- Runway2_Free: A Boolean variable; TRUE if runway 2 is available, otherwise FALSE; Initialized to TRUE.
- Landing_Queue_Size: The number of objects in the Landing queue
- Takeoff_Queue_Size: The number of objects in the Take-off queue
- HeadOF_Landing_Queue: The smallest time stamp event objects in the Landing queue
- HeadOF_Takeoff_Queue: The smallest time stamp event objects in the Take-off queue
- Airplane.Runway : An Integer variable ; 1: assigned to runway 1, 2: assigned to runway 2.
- Total_Pass_Arrive: The total number of accumulated passengers arrived at the airport.
- Num_Pass_Arrive: The number of passengers on an airplane that has arrived at the airport.
- Total_Pass_Depart: The total number of accumulated passengers departed from the airport.
- Num_Pass_Depart: The number of passengers departed on an airplane taking off from the airport.
- Total_Circling_Time: The total amount of time that all airplanes have circling the airport waiting for the runway to become available.
- Airplane_Circling_Time: The circling time of an airplane waiting for the runway to become available.
- Total_Waiting_Time_TakeOff: The total amount of time that all airplanes at the airport waiting for the runway to become available for taking-off.
- Airplane_Waiting__Time: The amount time of an airplane waiting for the runway to become available for taking-off.
- Num_Flight_Hours: The amount of time an airplane flied after last inspection

Arrival Event:

```
If(Runway1_Free)
    Runway1_Free = FALSE;
    Airplane.Runway =1;
    Schedule Landed Event at time Now+R;
Else if( Runway2_Free)
    Runway2_Free = FALSE;
    Airplane.Runway =2;
    Schedule Landed Event at time Now+R;
Else
    Add a Landed Event to the landing queue;
End
```

Fig. 2. Pseudo code for an arrival event at the airport

Landed Event:

```
Schedule Taxiing Event at time Now+TaxiingTime;
Total_Circling_Time := Total_Circling_Time + Airplane_Circling_Time
If( (Landing_Queue_Size > 0 ) && (Takeoff_Queue_Size > 0 ))
    If(HeadOf_ Landing_Queue  <= HeadOf_Takeoff_Queue)
      Next_Event  = Remove the first Event from the Landing queue;
    Else
      Next_Event  = Remove the first Event from the Take-off queue;
    End
    Schedule the Next_Event;
Else if (Landing_Queue_Size > 0 )
    Next_Event  = Remove the first Event from the Landing queue;
    Schedule the Next_Event;
Else if (Takeoff_Queue_Size > 0 )
    Next_Event  = Remove the first Event from the Take-off queue;
    Schedule the Next_Event;
Else
    If(Airplane.Runway == 1)
      Runway1_Free = TRUE;
    Else
      Runway2_Free = TRUE;
End
```

Fig. 3. Pseudo code for a landed event at the airport

Taxiing Event:

```
Schedule an Unloading event at time Now+UnloadingTime;
```

Fig. 4. Pseudo code for a taxiing event at the airport

Unloading Event:

```
Total_Pass_Arrive = Num_Pass_Arrive + Total_Pass_Arrive;
Pick the destination airport;
Update Num_Flight_Hours;
If(Num_Flight_Hours>=MTC_Cyle)
    Schedule a departure event at time Now + MTC
Else
    Schedule a departure event at time Now + G
End
```

Fig. 5. Pseudo code for an unloading event at the airport

Airplane Maintenance Event:

```
Num_Flight_Hours = 0;
Schedule a departure event at time Now + G;
```

Fig. 6. Pseudo code for an airplane maintenance event at the airport

Departure Event:

```
If(Runway1_Free)
    Runway1_Free = FALSE;
    Airplane.Runway =1;
    Schedule Takeoff Event at time Now+T;
Else if( Runway2_Free)
    Runway2_Free = FALSE;
    Airplane.Runway =2;
    Schedule Takeoff Event at time Now+T;
Else
    Add a Takeoff Event to the Takeoff queue;
End
```

Fig. 7. Pseudo code for an airplane departure event at the airport

Figure 8 shows the pseudo code for the take-off event in the sequential version of the airport system simulation whereas Figure 9 depicts the pseudo code for the take-off event in the distributed version of the airport system simulation. The differences are shown in blue colored font. SimulatorEngine class was also customized for implementing the distributed version of the airport system simulation. The pseudo code for implementing the Null Message and the YAWNS algorithms are shown in Figures 10 and 11, respectively.

Takeoff Event: (Sequential Simulation)

```
Total_Pass_Depart = Num_Pass_Depart + Total_Pass_Depart;
Calculate the flight time (as F) based on the selected destination
Schedule an Arrival Event at time Now + F;
Total_Waiting_Time := Total_Waiting_Time + Airplane_Waiting_Time
```

```
If( (Landing_Queue_Size > 0 ) && (Takeoff_Queue_Size > 0 ))
    If(HeadOf_ Landing_Queue  <= HeadOf_Takeoff_Queue)
      Next_Event  = Remove the first Event from the Landing queue;
    Else
      Next_Event  = Remove the first Event from the Take-off queue;
    End
    Schedule the Next_Event;
Else if (Landing_Queue_Size > 0 )
    Next_Event  = Remove the first Event from the Landing queue;
    Schedule the Next_Event;
Else if (Takeoff_Queue_Size > 0 )
    Next_Event  = Remove the first Event from the Take-off queue;
    Schedule the Next_Event;
Else
    if(Airplane.Runway ==1)
      Runway1_Free = TRUE;
    else
      Runway2_Free = TRUE;
End
```

Fig. 8.   Pseudo code for an airplane take-off event in sequential simulation

```
Takeoff Event: (Distributed Simulation)
Total_Pass_Depart = Num_Pass_Depart + Total_Pass_Depart;
Calculate the flight time (as F) based on the selected destination
Arrival_Time = Now + F;
Message_Type = Plane_Arrival_Type
Plane_Arrival_Message = [Arrival_Time, Airplane_Id, Message_Type];
Send the Plane_Arrival_Message to the destination LP;
Total_Num_Sent_Message = Total_Num_Sent_Message + 1;
Total_Waiting_Time := Total_Waiting_Time + Airplane_Waiting_Time;
If( (Landing_Queue_Size > 0 ) && (Takeoff_Queue_Size > 0 ))
    If(HeadOf_ Landing_Queue <= HeadOf_Takeoff_Queue)
      Next_Event = Remove the first Event from the Landing queue;
      Schedule the Next_Event at time Now+R;
    Else
      Next_Event = Remove the first Event from the Take-off queue;
      Schedule the Next_Event at time Now+T;
    End
Else if (Landing_Queue_Size > 0 )
    Next_Event = Remove the first Event from the Landing queue;
    Schedule the Next_Event at time Now+R;   ;
Else if (Takeoff_Queue_Size > 0 )
    Next_Event = Remove the first Event from the Take-off queue;
    Schedule the Next_Event at time Now+T;
Else
    if(Airplane.Runway ==1)
      Runway1_Free = TRUE;
    else
      Runway2_Free = TRUE;
End
```

Fig. 9.   Pseudo code for an airplane take-off event in distributed simulation

```
Run Method in SimulatorEngine Class: (Null Message)
My_Rank = get the LP rank;
Size = get the total number of LPS;
For all neighboring LP(j) do:
        $Lookahead(My\_Rank , j) = \frac{Distance(My\_Rank, j)}{Fastest\_Plane\_Speed}$;
        Time_Stamp = Lookahead(My_Rank , j);
        Message_Type = Null_Type;
        Null_Message = [Time_Stamp, Message_Type];
        Send the first Null_Message to LP(j);
End
For (i=0; i<Size; i++)
        LP_Terminate[i] = FALSE;
        Channel_Time[i] = 0;
End
Count = 0;
Ack_Counter = 0;
Termination = FALSE;
While(Termination == FALSE)
        For (i=0; i<Size; i++)
                If(LP_Terminate[i] == TRUE)
                        Count++;
        End
        If (Count == Size)
                Remove the stop event from the Event_List;
                Termination = TRUE;
        End
    End

    Check if a new message arrived from any neighboring LPs;
    If a new message arrived
        Check the Message_Type;
        If(Message_Type == Plane_Arrival_Type)
                Schedule an Arrival Event at Arrival_Time;
        Else if(Message_Type == Null_Type)
                Channel_Time[Sender_Rank]=Time_Stamp;
        Else if(Message_Type == Termination_Type)
                LP_Terminate[Sender_Rank] = TRUE;
                Send an Ack_Message for termination to the
                LP(Sender_Rank)
        Else
                Ack_Counter ++;
                IF(Ack_Counter == Size -1)
                        LP_Terminate[My_Rank] = TRUE;
                End
        End
    End
    Find the Min_Channel_Time in Channel_Time[i] for all i
    Such that i is not equal to My_Rank  and store it in Min_time
    Local_Time = Min_Channel_Time;
    If (LP_Terminate[My_Rank] == FALSE)
        Min_Time = The time stamp of first event in Event_List;
        While(Min_Time < Local_Time)
                Remove and Execute the first event;
                Min_Time = the Time stamp of first event in
                                Event_List;
                Local_Time = Now;
        End

    End
    If(Termination_Flag == FALSE)
        Min_Time = The time stamp of first event in Event_List;
        If((Min_Time == Termination_Time) &&
                (Min_Channel_Time >=Termination_Time)
                Send a Termination_Message to every
                        neighboring LP;
                Termination_Flag = TRUE;
                Local_Time= Termination_Time;
        End
    End
    If  (Termination_Flag == FALSE)
      If( (Local_Time != 0 )&& (Local_Time!=Old_Local_Time))
        For all neighboring LP(j) do:
                $Lookahead(My\_Rank ,j) = \frac{Distance(My\_Rank, j)}{Fastest\_Plane\_Speed}$

                Time_Stamp = Local_Time+Lookahead(My_Rank,j);
                Message_Type = Null_Type;
                Null_Message=[Time_Stamp, Message_Type];
                Send the Null_Message to LP(j);
        End
      End
    End
End
```

Fig. 10. Pseudo code for implementing the null message algorithm in the run method of the SimulatorEngine class

```
Run Method in SimulatorEngine Class: (YAWNS)
My_Rank = get the LP rank;
Size = get the total number of LPs;
Lookahead = Calculate the minimum flight_time among all arriving
        flights using the fastest airplane;
While(Termination == FALSE)
        Check if a new message arrived from any neighboring LPs;
        If A new message arrived
                Schedule the Arrival Event at Arrival_Time;
                Total_Num_Arrived_Message ++;

        End
        Min_Time = The time stamp of first event in Event_List;
        If((Min_Time == Termination_Time)
                Local_Terminate = TRUE;
        Else
                Local_Terminate = FALSE;
        END
         Diff_Count = Total_Num_Arrived_Message -
                 Total_Num_Sent_Message;
        Min_Time = The Time stamp of first event in Event_List;
        Create a buffer called LBTS_Message;
        LBTS_Message[My_Rank]=[Diff_Count, Local_Terminate,
                                        Min_Time];
        Wait and Gather All the data from all neighboring LPs into
                LBTS_Message;
        Sum_Diff_Count = LBTS_Message.Diff_Count[0];
        Global_Terminate = LBTS_Message.Local_Terminate[0];
        Global_Min_Time =  LBTS_Message.Min_Time[0];
        For (j = 1 ; j<Size; j++)
                Sum_Diff_Count =
                   LBTS_Message.Diff_Count[j] + Sum_Diff_Count;
                Golabl_Terminate =
                     (LBTS_Message.Local_Terminate[j] )
                     &&(Golabl_Terminate);
                If  (LBTS_Message.Min_Time[j] < Global_Min_Time)
                        Global_Min_Time =
                                        LBTS_Message.Min_Time[j];
                End
        End
        If(Sum_Diff_Count == 0)
                If(Golabl_Terminate == TRUE)
                        Granted_Time = Terminate_Time;
                        Remove the stop event from the Event_List;
                        Termination = TRUE;
                Else
                        Granted_Time =Global_Min_Time+Lookahead;
                        Next_Time = Time stamp of next event in
                                        Event_List;
                        While( (Next_Time < Granted_Time) &&
                         (Next_Time < Termination_time))
                                Remove and Execute the first event;
                                Next_Time = Time stamp of next event
                                             in Event_List;
                        End
                End
        End
End
```

Fig. 11. Pseudo code for implementing the YAWNS algorithm in the run method of the SimulatorEngine class

C. Implementation

This section describes the implementation of pseudo codes for handling airport events depicted in previous section.

Apart from the name of an airplane, there several other parameters in the Airplane class as listed below, including the corresponding methods for retrieving and setting their values.

(a) Airplane's identification number (as id),

(b) Maximum number of seats (as max_passenger),

(c) Actual number of passengers (as num_passengers),

(d) Speed of airplane (as speed),

(e) Accumulated flight hours since the last maintenance (as mtc_time),

(f) Runway index assigned to an airplane (as runway), and

(g) Destination identification number (as dest).

The arrival time (as arrival_time) and the departure time (as departure time) of each airplane are recorded for calculating the circling time before landing and waiting time to take-off at each airport. The next destination is selected by a random number generator. The randomly selected destination airport must not be the same as the departing airport. The number of passengers on each flight is also randomly generated. The number of passengers cannot exceed the capacity of that airplane.

Airplane object (called airplaneType) and an array of all airports (called Airports) are new parameters passed to the AirplaneEvent class as inputs. The airplane object represents the airplane associated with that particular airplane event. An array of airports is initialized in the main method of AirportSim class.

Apart from the initial parameters in the Airport class, the required time for taking off (called requiredTimeToTakeOff), the required time for taxiing (called requiredTimeToTaxii), the required time for unloading (called requiredTimeToUnload), the required time for completing routine airplane maintenance (called requiredTimeToMtc), and the matrix of distances between airports (2-dimensional array of double called dis) were implemented. Two queues for runway-related events (so called landing and taking-off queues) were also incorporated into the Airport class. The take-off and landing queues are implemented using linked lists. Whenever a landed event or a taking-off event at an airport completed and the landing or take-off queue is not empty, the next taking off or landing event is selected and removed from queue and scheduled by the simulator. The selection of landing or taking-off event from the queues is determined by the assumptions given in the conceptual models.

The Null Message algorithm was implemented using Java binding libraries provided with OpenMPI. Since the algorithm is asynchronous, the non-blocking MPI send and receive operations [31] were chosen in this work. Specifically, each LP first retrieved its rank and the total number of LPs using the MPI getRank() and getSize() methods. The plane arrival message shown in the pseudo code in Figure 9 is sent by the LP where the airplane took-off using MPI iSend method. On the receiving LP where the airplane is arriving, the MPI non-blocking receive (iReceive) is used. A buffer of type double was used for sending the time stamp of plane arrival, the airplane ID, and the type of message. The received values are casted into corresponding data type (e.g. Integer), if needed. The same communication channel is used for sending and receiving the null message, the termination message, and the acknowledgment message (to acknowledge the receipt of the termination message).

In each simulation cycle of the run method in the SimulatorEngine class for the Null Message algorithm, a new null message is sent to the neighboring LPs if and only if the LP's local time has been updated. The timestamp inside the null message for each neighboring LP is equal to the LP's local time plus the duration of the fastest flight to the destination airport simulated by that neighboring LP. The local time is the timestamp of the latest event that has been removed from the

TreeSet of events. If no event has been removed, the local time is defined using the minimum time among all timestamps stored in the array called tCh, which is an array of type double that keeps the timestamp of the latest null message from each incoming link (or channel). When LPs are terminated and received the acknowledgment of their terminations, each LP terminates the thread by calling the MPI Finalize() method.

Since the YAWNS algorithm is synchronous, a barrier synchronization is needed at the end of each iteration in the run method of the SimulatorEngine class. Similar to the Null message, LPs retrieve its rank and the total number of LPs using MPI getRank() and getSize() methods. The airplane arrival message shown in the pseudo code of Figure 9 is implemented in LP where the plane took-off using the MPI non-blocking send operation (iSend). On the receiver side, the message is received using the MPI non-blocking receive operation (iReceive). A buffer of type double is used for sending the time stamp of plane arrival and the airplane ID. The data values are casted to the corresponding data type (e.g. Integer) when necessary. The MPI iAllGather(), waitFor(), and free() methods are added as a barrier point in each LP. To ensure that no transient message exists when the next granted time is defined, two counters were used for keeping track of the number of sent and received messages in each LP (defined as sentCount and recCount). The granted time for the next cycle only defined if the difference between the number of sent and received messages (combined from all LPs) is equal to zero. At the barrier point, the termination status (called m_termination), difference in the number of sent and received messages (called m_diffCount), and the minimum timestamp of unprocessed events (called Min_time) are gathered from all LPs to define the maximum granted time (as m_granted_time) for processing the events in the next cycle.

To collect the statistics, before invoking the MPI Finalize() method in each process, the MPI reduce() method is called to gather the total execution time and the total number of events processed in that simulation run from all LPs.

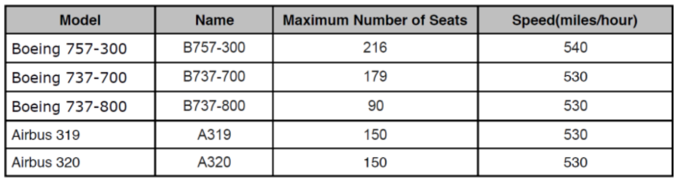
IV. Simulation Setup and Results

A. *Setup and Initialization*

The simulation modeled five different airplanes with information on their speeds and specification on their passenger capacities obtained from the website of United Airline [32]. Table I displays the information about these airplanes. The total number of airplanes in all simulation runs is set to 2,560 airplanes. The required amount of time for landing, taxiing, unloading, maintenance, being on the ground (loading and departing), and taking off were set at 5, 10, 20, 600, 25, and 5 minutes of the simulated clock, respectively. The simulation is set to stop at 11,520 minutes of the simulated clock with each simulation clock cycle equated to one minute. Thus, the simulation ran for an equivalent of 192 simulated hours or eight simulated days.

This simulation study modeled an airport system with 64 airports. The airports were chosen from the list of busiest airport around the World [33]. The coordinates of these chosen airports were downloaded from an Open data website [34]. Table II in the Appendix shows the calculated distances between these airports. The flight times between these airports were calculated from these distances divided by the speed of an airplane used in that flight. A Java code was developed to calculate the distance between airports based on the sample code from the GeoDataSource website [35].

TABLE I. Capacity and Speed of Airplanes



Model	Name	Maximum Number of Seats	Speed(miles/hour)
Boeing 757-300	B757-300	216	540
Boeing 737-700	B737-700	179	530
Boeing 737-800	B737-800	90	530
Airbus 319	A319	150	530
Airbus 320	A320	150	530

For the simulation of the routine airplane maintenance, the maximum accumulated flight hours between each maintenance event is set at 65 simulated hours (i.e., 3,900 minutes of simulation clock). The integer numbers such as the number of passengers on each flight and the ID of the next destination airport were generated using random number generator with the initialization in the Airport class. The random seeds are set according to the corresponding airport ID.

To carry out the parallel execution of the distributed airport system simulation, we installed Java Compiler (*javac*), Java Virtual Machine (*java*), and OpenMPI with Java binding in the user space (i.e., GT personal/student user account) on the Jinx cluster. The Java codes for the distributed airport system simulation were compiled with the *mpijavac* command. The simulation program was executed with the *mpirun* command, where the "-np" option specifies the number of processors, the java program invokes JVM on each processor, and the AirportSim is the application program running on top of each JVM. The first parameter for AirportSim specifies the number of airports and the second parameter specifies the synchronization protocol (i.e., 1 for Null Message and 2 for YAWNS). For example, "mpirun –np 64 java AirportSim 64 1" creates 64 processes representing 64 airports on 64 processors with the Null Message algorithm as synchronization mechanism. A command for starting a sequential run using one processor simulating 64 airports is "mpirun –np 1 AirportSim 64." While these commands are appropriate for interactive session, we created a script to submit batch job on Jinx for collecting data. We selected nodes on Jinx that have two six-core processors (i.e., 12 processor cores per node) for running the simulation. We ran the simulation with 16, 32, and 64 airports using 2, 3, and 6 nodes, respectively. To collect the event processing time, all print/debugging statements were commented out with only the total number of passengers arriving and departing the airports, the amount of waiting time to land and take-off, the wall-clock time, and the number of events processed remained.

B. *Assertion, Debugging and Verification*

To ensure a runway is occupied by only one airplane at a time, assertion statements were included in the Take-Off and Landing events. Specifically, an array list for each runway was created. When an airplane is assigned to a specific runway, the airplane ID is added to the runway's array list. After completing the runway-dependent events (such as take-off and landing), the corresponding airplane ID is removed from the array list. Before a take-off or landing event is reported as complete, the code double checked if the runway's array list has only the corresponding airplane ID in its list. If the assertion statement is false, the Java runtime system throws an assertion error.

For verification and debugging purposes, a few simple scenarios with predefined destination airports were developed and tested. The outputs from these simulation runs were compared against manually calculated results, ensuring that the simulation program behaves according to the defined specifications and requirements.

For verification of the distributed airport system simulation, the log file generated from the sequential execution was compared again the log file from the distributed execution using a scenario where there were no simultaneous events. In addition, for checking the causality order of events in each LP, a statement was added to check whether the timestamp of the next event in the event list is smaller than the current simulation time. If the statement is true, the simulation will throw an error/exception for the violation of causality order.

C. Results and Discussion

All simulation runs deployed 2,560 airplanes. Each airplane was scheduled to depart from one of the 64 airports to a randomly selected destination airport with a randomly generated passenger loads. There are two runways at each airport. All simulations were initialized in the same way. The average wall-clock runtimes of sequential execution, distributed execution with the Null Message algorithm, and distributed execution with the YAWNS algorithm, modeling 16, 32, and 64 airports are shown in Figure 12.

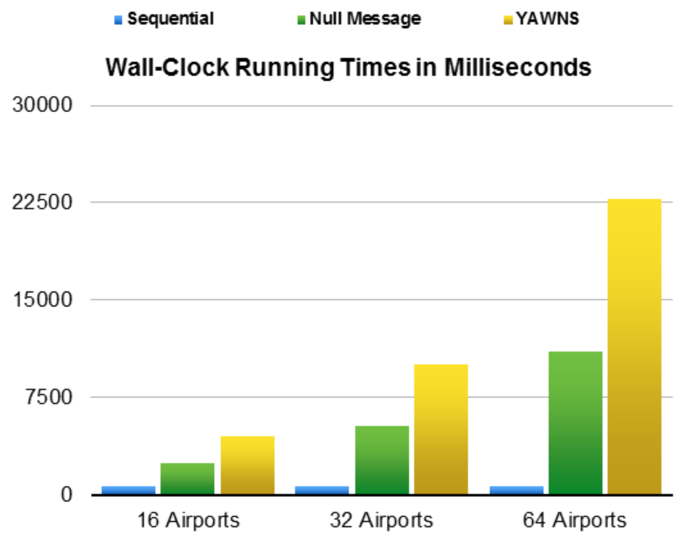


Fig. 12. Average wall-clock runtimes under different algorithms

The average of wall-clock runtimes increases as the number of airports grows. The sequential execution is the fastest due to the fact that one airport per processor requires very little computation time but generates many messages between processors. With 64 airports, the average running times are 868.1, 11,012, and 22,786.3 milliseconds for sequential, Null Message, and YAWNS, respectively.

The average number of events processed for each simulation run under different algorithms is shown in Figure 13. The results depicted a consistent number of events processed across all simulation runs under the same setting and the same number of airports. To compare different simulation execution algorithms in terms of performance, the number of events processed per second is given in Figure 14.

A comparison of the number of events processed per second under sequential and distributed executions shows the sequential execution produced the highest throughput. When the distributed execution was carried out using YAWNS, the number of events processed per second was the lowest. With 64 airports, the number of events processed per second under sequential execution is about 15 and 31 times larger than under distributed execution with the Null Message algorithm and the YAWNS algorithm, respectively.

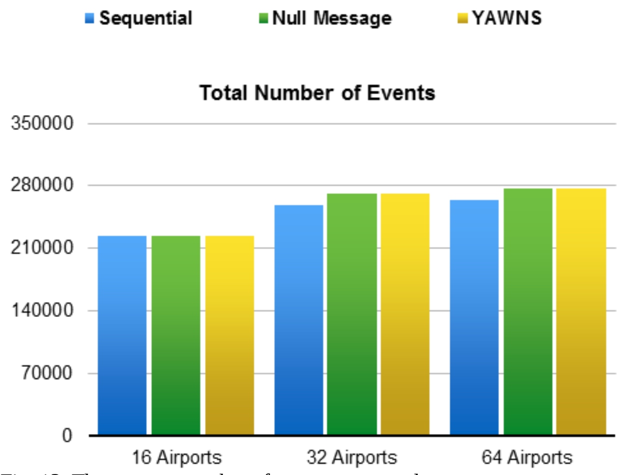


Fig. 13. The average number of events processed

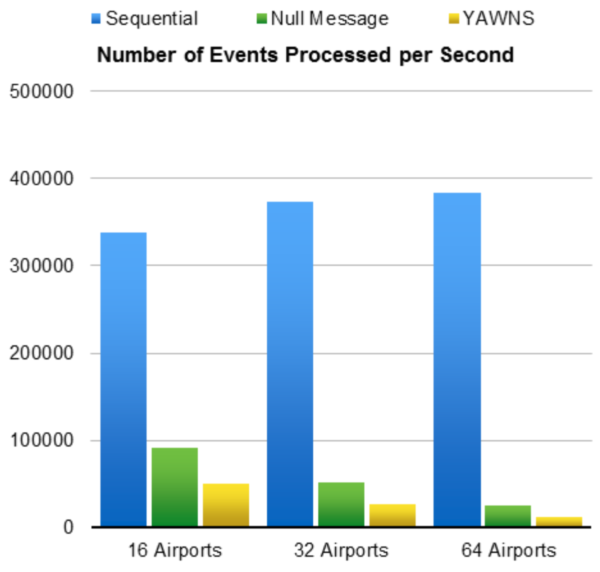


Fig. 14. The average number of events processed per second

The underwhelming performance/throughput from the distributed execution of airport system simulation can be traced back to the low computation to communication ratio. This is because only one airport is simulated into each LP in the current version of the simulator. The distributed simulation execution may yield a better performance by simulating multiple airports on each LP and by making the network of airports larger (e.g. thousands of airports). It is also possible to increase the computation ratio by adding more details to the operations at each airport.

V. Conclusions and Future Work

The objective of this work is designing and implementing the distributed airport system simulation programs from the sequential airport system simulation in previous studies [36, 37], which focused on studying the runway congestion deriving from the airplanes landing and taking-off events at the airports and on analyzing the number of passengers arriving at and departing from the airports.

The implementation of the distributed simulation program included the conservative synchronization algorithms called the Null Message algorithm and the YAWNS algorithm. The throughput was defined as the total number of (airport) events processed per second. The performance of Null Message was consistently better than YAWNS algorithm for various number of airports simulated. However, both distributed versions of the simulator were outperformed by the sequential version.

There are various ways for expanding and improving the distributed airport system simulation. First, customizing the algorithm in a way that allows multiple airports per LP. We can also collect information about thousands of airports and use the simulation for a larger network of airports. In addition, several studies have focused on improving the performance of the Null Message algorithm by sending less messages between LPs. There are also other synchronization protocols, such as Time Warp, that can be adopted for synchronizing these LPs.

For improving the sequential airport system simulation, we can collect more information about the other important processes that happen at the airports and add more details to the simulation. For example, the number of gates available at each airport, the amount of time required for unloading the luggage, the amount of time different airplane types needed for reaching the required altitude for take-off, and the amount of time required for refueling different types of airplanes will yield more realistic simulation results. We can also calculate the passenger unloading time more accurately/realistically by defining a formula based on the number of passengers on the arriving airplane. A more detailed and realistic version of this simulation (e.g., simulating all United airline's routes and resources) can be used as a tool for evaluating flight schedules and other resource allocation techniques at the airports.

Acknowledgment

The author thanks Dr. Brian Swenson for his guidance and the initial framework for the simulation program developed in this study.

References

[1] R. M. Fujimoto, Parallel and Distributed Simulation Systems, Wiley Interscience, 2000.

[2] World Airports Codes, 2017. Retrieved from: https://www.world-airport-codes.com/world-top-30-airports.html.

[3] K. M. Chandy and J. Misra. "Distributed simulation: a case study in design and verification of distributed programs,". *IEEE Trans. on Software Engineering,* 1979, pp. 440-452.

[4] D. M. Nicol, C. C. Michael, and P. Inouye. "Efficient aggregation of multiple LPs in distributed memory parallel simulations," *In Proceedings of the 21st Conference on Winter Simulation (WSC '89),* 1989, pp. 680-685.

[5] OpenMPI: Open Source High Performance Computing, 2017. *Open-mpi.org*. Retrieved from https://www.open-mpi.org/.

[6] P. E. Joustra and N. M. van Dijk, Simulation of check-in at airports, Proceeding of the 2001 Winter Simulation Conference, Arlington, VA, 2001, pp. 1023-1028 vol.2.

[7] N. Doshi and R. Moriyama, Application of simulation models in airport facility design, Proceedings of the Winter Simulation Conference, San Diego, CA, USA, 2002, pp. 1725-1730 vol.2.

[8] J. S. Smith and B. L. Nelson, Estimating and interpreting the waiting time for customers arriving to a non-stationary queueing system, Winter Simulation Conference, Huntington Beach, CA, 2015, pp. 2610-2621.

[9] H. Erzberger, L. A. Meyn, and F. Neuman, A fast-time simulation tool for analysis of airport arrival traffic, NASA/TP-2004-212283.

[10] B. Bubalo and J. Daduna, Airport capacity and demand calculations by simulation - the case of Berlin-Brandenburg International Airport. NETNOMICS 12:161–181.

[11] Y. Chen, A Modeling Framework to Estimate Airport Runway Capacity in the National Airspace System, Virginia Polytechnic Institute and State University Theses and Dissertations, 2006.

[12] TAC Coperation, SIMMOD Simulation Airfield and Airspace Simulation Report, 2006.

[13] M. Piera, A. Novikov, C. Trapote, and J. Ramos, SIMULATION MODEL TO IMPROVE AIR CARGO OPERATIONS IN PASSENGER AIRCRAFT, SummerSim '10 - 2010 Summer Simulation Multiconference, Ottawa, ON, Canada.

[14] S. Crook, "The use of simulation and virtual reality in the design and operation of airport terminals," Simulation '98, York, 1998, pp. 8-10.

[15] M. E. Miller and S. P. Dougherty, "Communication and the future of air traffic management," 2004 IEEE Aerospace Conference Proceedings, 2004, pp. 1635 Vol.3.

[16] D. B. Koch, "3D visualization to support airport security operations," in IEEE Aerospace and Electronic Systems Magazine, vol. 19, no. 6, pp. 23-28, June 2004.

[17] J. Teutsch and A. Postma-Kurlanc, "Enhanced Virtual Block Control for Milan Malpensa Airport in low visibility," 2014 Integrated Communications, Navigation and Surveillance Conference (ICNS) Conference Proceedings, Herndon, VA, 2014, pp. E1-1-E1-13.

[18] L. J. Wang, Q. X. Wang, D. Y. Dong, and X. L. He, "Simulation Development of Aircraft Cockpit Instruments Based on GL Studio," International Conference on Information Engineering and Computer Science, Wuhan, 2009, pp. 1-4.

[19] D. Karikawa, M. Takahashi, A. Ishibashi, T. Wakabayashi and M. Kitamura, "Human-Machine System Simulation for Supporting the Design and Evaluation of Reliable Aircraft Cockpit Interface," 2006 SICE-ICASE International Joint Conference, Busan, 2006, pp. 55-60.

[20] D. M. Nicol. "Principles of conservative parallel simulation," In Proceedings of the 28th Conference on Winter Simulation (WSC '96), 1996, pp. 128-135.

[21] R. E. Bryant. "Simulation of packet communication architecture computer systems," MIT, Cambridge, Mass.,Tech. Rep. MIT-LCS-TR-188, 1977.

[22] R.M. Fujimoto. "Research Challenges in Parallel and Distributed Simulation". *ACM Trans. Model. Comput. Simul.* 26, 4, Article 22, p. 29, May 2016.

[23] D. Jefferson. "Virtual time", ACM Transactions on Programming Languages and Systems 7, 3, pp. 404–425, 1985.

[24] D. Fu, M. Becker, and H. Szczerbicka. "Universal simulation engine (USE): a model-independent library for discrete event simulation". *In Proceedings of the 48th Annual Simulation Symposium (ANSS '15)*, 2015, pp. 146-154.

[25] P. D. Barnes Jr. , C. D. Carothers, D. R. Jefferson, and J. M. LaPre. “Warp speed: executing time warp on 1,966,080 cores”. *In Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS ’13)*, 2013, pp. 327-336.

[26] *ns-3 Collaboration. “*The ns-3 network simulator.” Internet: https://www.nsnam.org/, March, 1, 2017.

[27] B.P. Swenson. “Techniques to Improve the Performance of Large-Scale Discrete-Event Simulation.” Ph.D. dissertation, Georgia Institute of Technology, 2015.

[28] R. M. Fujimoto and A. Biswas. “An Empirical Study of Energy Consumption in Distributed Simulations,” *IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2015, pp. 163-170.

[29] R. Jordan, M. A. Ishutkina, T. G. Reynolds, “A Statistical Learning Approach to the Modeling of Aircraft Taxi-Time,” MIT Lincoln Laboratory, Tech. Paper, OMB No. 0704-0188, 2010.

[30] Sriram and A. Haghani, “An optimization model for aircraft maintenance scheduling and re-assignment,” *Transportation Research Part A: Policy and Practice*, vol. 37, Issue 1, pp. 29-48, Jan. 2003.

[31] MPI – Tutorial 5 – Asynchronous Communication. *The Supercomputing Blog*. N.p., 2017. Web. 25 Apr. 2017.

[32] United Airlines, 2017. Retrieved from: https://www.united.com/web/en-US/content/travel/inflight/aircraft/default.aspx.

[33] S. Rogers, The world's top 100 airports: listed, ranked and mapped, the Guardian, 2017. Internet: https://www.theguardian.com/news/datablog/2012/may/04/world-top-100-airports.

[34] Open data @ OurAirports, Ourairports.com, 2017. Internet: http://ourairports.com/data/.

[35] Calculate Distance by Latitude and Longitude using Java, GeoDataSource, 2017. Internet: http://www.geodatasource.com/developers/java.

[36] R. Aliehyaei, “Analytic Simulation of Airport System: A Preliminary Study,” Assignment #1 Report, CSE6730, GaTech, Spring 2017.

[37] R. Aliehyaei, “Analytic Simulation of Airport System: With Additional Features and Visualization,” Assignment #2 Report, CSE6730, GaTech, Spring 2017.

TABLE II. DISTANCES BETWEEN AIRPORTS (IN MILES)