Assignment 1 COMP2310

Abhaas Goyal (u7145384)

September 30, 2020

1 Introduction

The following report is an analysis of a simulation for Distributed Message Passing given for Assignment 1 of COMP2310. This report starts with Stage B implementation however the only major difference between the implementations of stage A and stage B was the message passing algorithm. While in Stage A the vehicles referred to a common variable keeping `Globe'First` (which was itself being continuously overwritten by every vehicle - This meant that the implementation would work stably only when a single globe is present), stage B moved onto a more sophisticated method of keeping a record, sharing the closest globe it could find and making decisions on the information received in the process.

2 Stage B and C

2.1 Motion Structure

By watching several videos given in Assignment Page[1], I got the notion that the I should first start with the aspect of controlling the motion of vehicles. Because if that were to be controlled, then the vehicles could decide to move to the globe in a coordinated manner and message passing would become highly efficient. From being inspired by Charging groups [1] and the fact that the position of the globe is dependent on it's centre of mass, I decided to make a similar Model of circular motion around the Energy Globe. Initially, I chose 2 axis of revolution but from testing for large number of vehicles (around 200~) and not being supported, I decided to make them four. By using principles of linear algebra, I decided to assign a fixed axis to a vehicle. Given radius r, Angle in XY plane θ and angle in XZ plane ϕ the motion [3] given by the formula :

$$\begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xcos(\phi) - ysin(\phi) \\ xsin(\phi) + ycos(\phi) \\ z \end{bmatrix}$$

$$\text{where } \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} rsin(\theta) \\ 0 \\ rcos(\theta) \end{bmatrix} + \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

where (x', y', z') are coordinates of the globes.
Changing θ clockwise in every clock step meant that the vehicle has someplace to go with respect to the vehicle. This seemed to work great until I reached around 250 globes. In this case the vehicles seemed to be very crowded and the design did not seem to be feasible in a large scale. After some thinking I realized that while rotating the space in between the vehicle and the globes was being wasted. So I made the radius of rotation with respect to the current charge present in the globe (again inspired from Charging Groups). So now the radius position would be calculated by:

$$r = c * \text{Vehicle charge}$$

- The value of the constant c was taken on the basis of that it shouldn't be too high to be very far away from the globes and neither should it create a blocking operation for large number of vehicles.
- The axis' for the vehicles were decided on the `Vehicle_Id mod 4`. While still not a perfect metric, in real life if we assume a random distribution, equal number of vehicles would be there in different axis
- However, it is also to be taken into consideration that the motion of vehicles being created from the model supports having different vehicles in different axis since the center of mass roughly stays the same.
- Also while moving in the `Throttle` of the Vehicle is set to low to conserve it's energy.
- Now the vehicle seemed to go in a continuous spiral motion.

A diagram representing the above concepts is given below:-

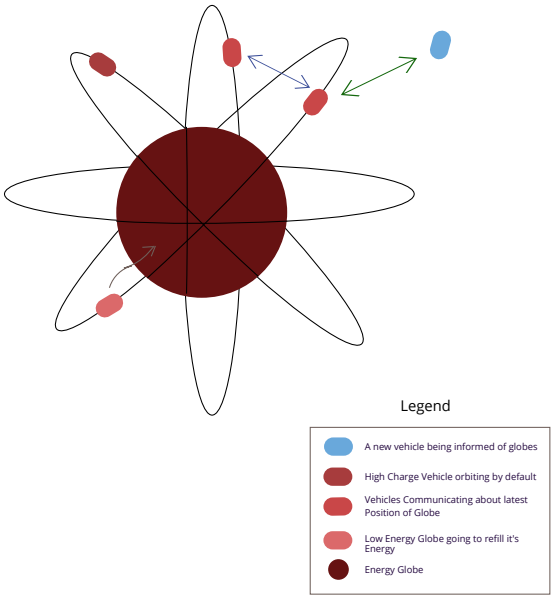

Figure 1: Model for Vehicle Motion

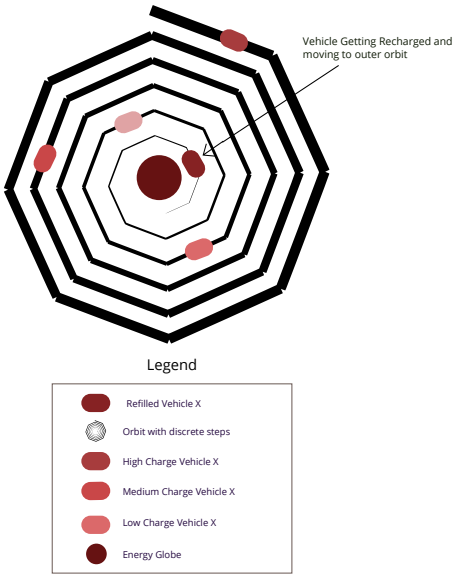

Figure 2: Proposed Motion of Individual Vehicle

2.2 Reading of Energy Globes

The reading of energy globes was done by the implementation given on the discussion forums [2]. In addition to the implementation, the vehicles will store the closest globe in their `Local_Record` while at the same time sending the location of multiple globes that it could find (that meant it could potentially work with multiple globes if message passing was done properly).

2.3 Message Passing

Now come the three pillars of actual message passing algorithm

- The central keypoint to be taken is the `Time_Stamp` of the messages. Whenever a vehicle receives a message, the globe position would only be updated on the basis of the latest timestamp. This ensures that the vehicles stay up to date as globe as possible

- One difficult point was how could the messages be wholly distributed. By looking at some videos of Swarm Intelligence and after some thinking, the answer lied in filtering and propagating the message everytime further on receiving one. This ensures that the sending and receiving will happen everytime and the operations will always happen in a circular manner.
- The third pillar stood in the fact of when exactly to receive the messages. If the Vehicle's battery is very low, it waits for any latest message being received (a blocking operation) of the globe's position. If it's higher than that, it will move peacefully in it's orbit or update the globe's position in it's database

The structure of motion of vehicles helps in the fact that the vehicles stay close to each other and the vehicle would always certainly receive a message if it has low battery. Graceful degredation method was provided in case the vehicle couldn't find a globe to go.

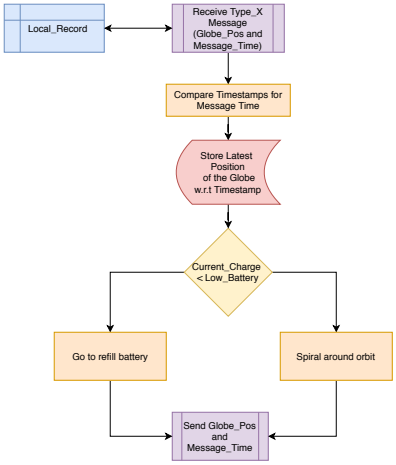

Figure 3: Central Algorithm used for Stage B

- On test runs, the first iteration in which all vehicles seemed to move to the globe at once made it seem like the vehicles were rotating along different points of the axis Figure: 4 . However, after a certain period of time Figure: 5 was obtained.
- From the data collected with respect to number of ships, the most efficient number of ships were found to be around 250. And the test runs seemed to indicate that after 1-2 iterations of all the vehicles' recharging cycle, no vehicle seemed to die for sustained periods of time. Upon further analysis of the motion of vehicles, it was found that on the first run, all the vehicles seemed to charge at the globe at roughly the same time, even though their motion was stable. Because of so many vehicles blocked to go at the same destination, if the number of vehicles are very high, the vehicles will die before reaching the globe for the first time. Also since the vehicles recharge one by one, their charges will go off at slighly different time steps, so in other iterations the tasks would not be blocking to the same destination.

Due to the simplicity of the algorithm and Vehicles easily changing spiral orbits without any problem, it produced even better results in Stage C and the proble of blocking operations seemed much less because of distribution among the energy globes. Although the vehicles in Stage B are stable till 250, the death rate seems to increase by a high margin after that. But in random globes configuration a lot more vehicles are supported. However, Stage C configuration can't guarantee staying alive at any number (although

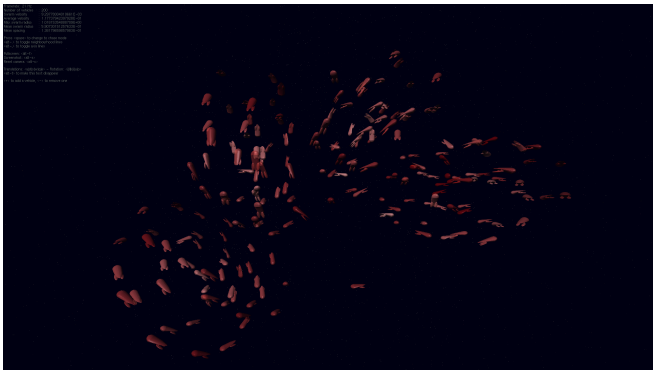

Figure 4: Unstable Initial Configuration

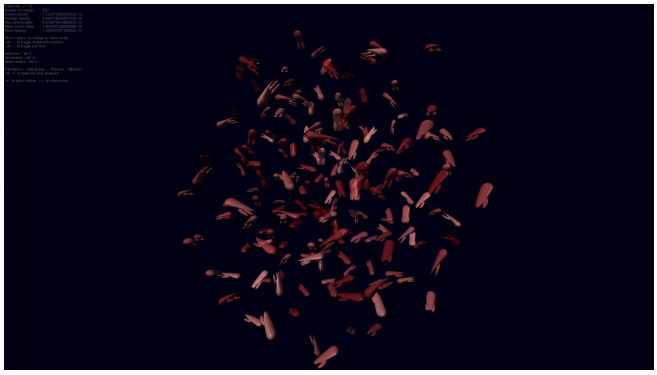

Figure 5: Stable Timely Configuration

the vehicles would eventually die with a higher probability when low number of vehicles are there). Some of the observed reasons were -

1. The vehicles in their low battery were going to refill and as soon as they almost reached the globe the globe went missing and the vehicle couldn't find the globe
2. When a particular globe died and it was far away from the other globes the vehicles had difficulty finding another globe with respect to their movement.
3. Even if they found the globe in the above case, it would still make it difficult for all of them to get accomodated to the new structure and recharge.

3 Stage D

3.1 Motivation

- My approach was to build a common lookup table for all the vehicles for storing `Vehicle_Id` for all other Vehicles in a sorted order and have a consensus on which table to use. With respect to that, I built a generic data structure with operations from scratch (properties inspired by [4]) from which one can store and Receive the sorted list. The algorithm is given as a flowchart in the next page.
- I wanted to have a clean design for stage D so I built it on top of stage C and the user can chose whether to implement Stage B/C/D just by changing 2-3 lines of code.

The difference between my implementation of Stage B and Stage D implementations with respect to timestamps is that in Stage D, I am expecting the Vehicle which constructs the `Local_List` (Local Lookup Table) first will have the priority over all other completed lists. This is oppsite from Stage B implementation where the last timestamp held precedence over earlier timestamps. Another key difference is in the records being sent. I have made two types of records being sent (`Type_X` is for `Globe_Pos` Messages and `Type_Y` is for building the common lookup table. This approach sacrifices the continuous sending of `Type_X` messages so the globe positions would not be updated as frequently (in the start of vehicles reaching the next coordinate). However, this behaviour is eliminated after sometime since the messages are sent safely before the vehicle reaches it's destination.

- Designing the data structure and it's operations was a very difficult task to do since a high amount of accuracy was needed in all the cases. For that while building any function of `concrete_order`,

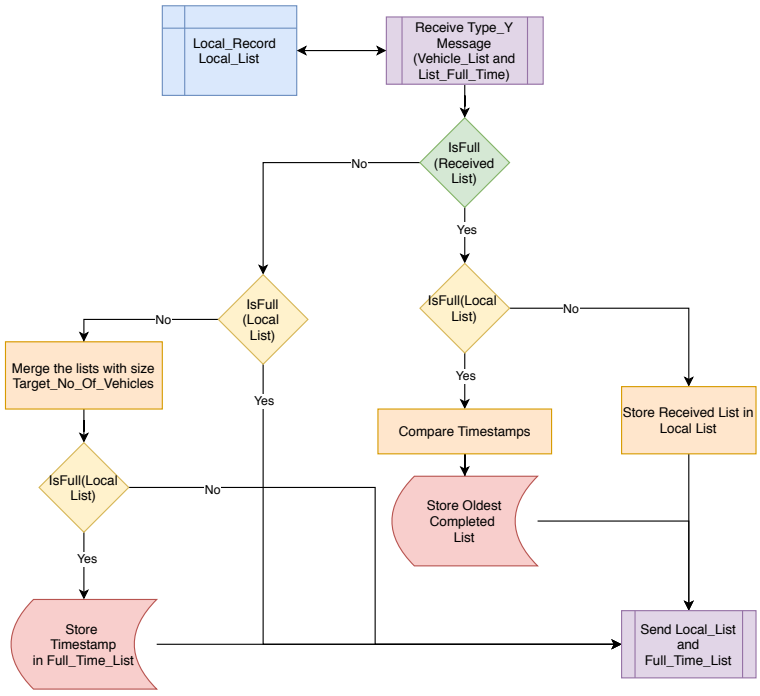

Figure 6: Central Algorithm used for Stage D

I tested it using a variety of inputs. In some cases, I was unable to find any errors in testing however when I used the data structure in multiple tasks (especially `Write_List` and `Max_Union`), manipulation of `Number_Of_Tasks` had to be controlled properly.

- After constructing the lookup table there is a `Consensus_Time_Interval` which believes that the lookup tables have been constructed at that point. If the vehicle is not in the list kill it else there is no need. This even ensures that if the `Target_No_Of_Elements` is mistakenly set greater than the initial amount of vehicles, no vehicles will die (Robustness).

3.2 Construction and Result Analysis

- Since this is not a request/response architecture, there is not much problem in having unresponsive messages from the receiver's side (since if it just stops propagating messages then there is no problem to the original sender).
- The Consensus time was set at 3 minutes and not less since that was the expected time for stability being observed in Stage B and C iterations
- Also the reason for using ordered list data structure was that in this case the vehicle numbers would be sorted in ascending order so in the best case scenario first n vehicles would be killed which are all in different axis. Another reason for making a sorted list is to make the operation of finding of vehicle faster through binary search (although I used a linear search due to time constraints in Assignment).

- A very important fact in building my lookup table was that it should be static. Because of that I made it with respect to the `Target_No_Of_Vehicles` and not on the $\text{Total}_{\text{NumberOfVehicles}}$ or any other sort of Dynamically created instance of the project (like the `Initial_No_Of_Elements`)
- For killing of the vehicle, I made it go a little further from it's position and stay stuck there until it dies. If I tried to send a lot of vehicles in different directions, the
- The results of Stage D indicate that the number of vehicles after consensus time is not exactly equal to the remaining elements. There seemed to be an error of $\pm(0-6)$ vehicles when `Target_No_Elements` is high. The difference was propounded when we take `Target_No_Of_Elements` as low.
- There was an extreme case for 500 vehicles and 42 `Target_No_Of_Elements`. In that, I seemed to get the error message

```
Warning: Vehicle task termination request ignored - attempting task abort now
Error: Vehicle task stuck in non-abortable code region - task abort failed
Warning: Vehicle task termination request ignored - attempting task abort now
Error: Vehicle task stuck in non-abortable code region - task abort failed
Warning: Vehicle task termination request ignored - attempting task abort now
```

- This means that the either the lookup table is not being constructed properly (which is the case) and also consensus is not being reached in the stipulated time.

4 Testing and Results Tables

Following conventions are used:

1. VC - Initial Vehicle Count
2. D - Duration (In Minutes)
3. TV - Target Vehicle Count
4. Ix - Iteration x

Note that the tests are the average of three simulations:

Table 1: Single Globe In Orbit

VC / D	1	5	20
64	64	64	64
150	150	150	150
200	200	200	200
300	300	271	271
400	320	317	316
500	340	331	328

Table 2: Dual Globes In Orbits

VC / D	1	5	20
64	64	64	64
150	150	150	150
200	199	199	199
300	297	297	297
400	369	369	369
500	434	428	428

Table 3: Random Globes In Orbits

VC / D	1	5	20
64	64	64	60
150	150	150	135
200	200	199	171
300	300	290	252
400	399	380	343
500	499	461	413

Table 4: Stage D Iterations

VC	TV	I1	I2	I3
100	42	33	38	44
100	64	55	60	70
200	130	133	126	140
200	100	91	101	98
500	42	0	0	0

FPS Performance

Note that `XSpecific_build_modes` has been set to Performance. The configuration is as follows

- CPU : Intel® Core™ i7-8750H Processor (9M Cache, up to 4.10 GHz)
- GPU : NVIDIA GeForce GTX 1050 Ti
- RAM : 16 GB

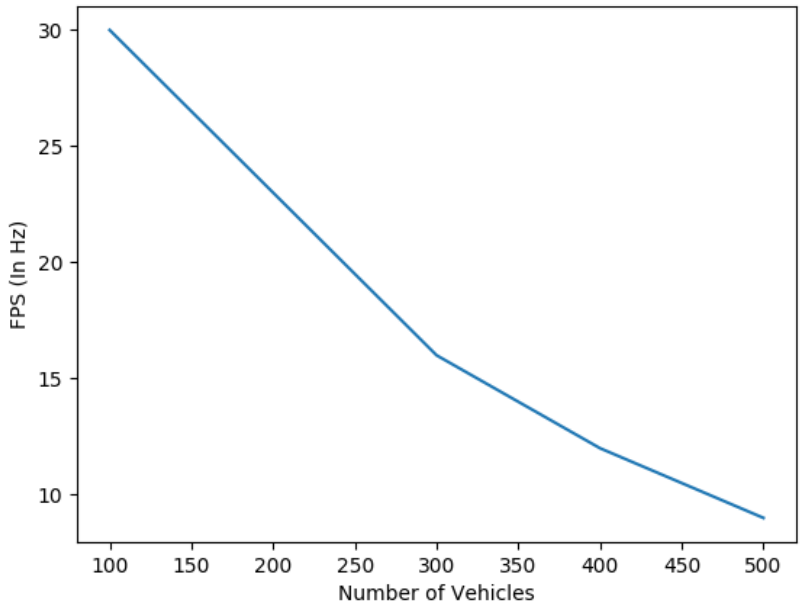

Figure 7: Performance with respect to number of vehicles

- The framerate goes low as the number of vehicles increase since we are creating a simutation where each vehicle's destination is being calculated at once. So there's a direct correlation between number of vehicles and the FPS slowing down. But this does not correlate with real life simulation since each vehicle will have their own individual simulator.
- After the death signal is called after the `Consensus_Time_Interval`, the frame rate drops by half in Performance mode whereas it drops by a huge factor (by 5-6x) in Development mode.

5 Conclusion

A scalable design has mostly been implemented for stage B and C. However with respect to Stage D, it seems like it's not been implemented for having the exact `Target_No_Of_Vehicles`. A major driving force for that is that the lookup table is not common for all. This may be due to a bug in implementation or a case in the algorithm that I missed. However the core principles of the implemented algorithm remain the same. Since the four axis set of vehicles come close to each other, the lookup table should be common for all in the stipulated time. The best approach for a solution would be during the checking time, check that all the lookup tables are equal and none of the vehicles in the lookup table have died in the process. This in my view is the future scope for the project.

In the end, I learnt a lot in the process about concepts of message passing and my favorite part was discovering about propagating of message. I also gained some insight in the field of swarm intelligence and I plan to continue on this project with some guidance.

6 Important points before running the code

- Set `XSpecific_build_modes= Performance`
- For implementing stage B and C comment out the part of the code where bootstrapping of `Type_Y` message is present.
- Toggling on Neighbourhood lines while vehicles are dying gives a run-time error [5].
- For very low amount of vehicles, one may need to adjust the dials given in the source code.

7 References

[1] Uwe R. Zimmer, S2 2020, Concurrent and Distributed Systems, Assignment 1, Australian National University, `https://cs.anu.edu.au/courses/comp2310/1-Labs-Assignments.html`, referred on September 15 2020
[2] Piazza Forums, Concurrent and Distributed Systems, `https://piazza.com/class/kcx5gab0r6o1d?cid=485`, referred on September 15 2020
[3] Three-Dimensional Rotation Matrices, `http://scipp.ucsc.edu/~haber/ph216/rotation_12.pdf`
[4] Uwe R. Zimmer, S2 2020, Concurrent and Distributed Systems, Week 5: Pipelined Insertsort, Australian National University, delivered 24 August 2020.
[5] Piazza Forums, Concurrent and Distributed Systems, `https://piazza.com/class/kcx5gab0r6o1d?cid=627`, referred on September 29 2020
[6] Other sources include Microsoft Teams Chat and Posts on Piazza (where students discussed some ideas like generic packages, various approaches, etc)