Report on Distributed Swarm System

Danny Feng (u6611178)

1 Overview

Given the scenario that the distributed spaceships need to charge at the moving globe, and they can only communicate and charge in a certain distance, the assignment requires a solution that can keep as many spaceships (i.e. Vehicles) alive as possible for a longer time. I have successfully finished all the stages from A to D. The fundamental idea of my solution is to use a local leader to do the polling with network-liked message passing and priority rearrange techniques. The leader also maintains a follower list so it can arrange a vehicle to sacrifice precisely in seconds. My system supports a maximum of 248 vehicles with zero accident death in single globe mode. The testing was last for an hour, and it is believed to stay stable forever since there is no indication for any issue. Also tested in random globe mode but with initial vehicle amount 130. It reached the target amount 100 in 105 seconds, and 70 vehicles survived when this hour-long test finished.

2 Problem Statement

2.1 Communication

I used protected shared memory in stage A to share the location of charging globe. For later stages as well as in the physical world this is not possible. A vehicle can only talk to another when they are in a close distance. It is not easy to let a far way vehicle to know the latest globe position because only the vehicle near the globe know where it is. I decided to make all vehicle connected as a network so the message can be delivered to everyone. On the other hand, I have considered the flexibility in the first place so I added a requirement that my message can be passed to a specified vehicle.

Another problem I planned to solve is about junk messages. I noticed this issue when I finished my first version of the message passing system. The amount of the message grows exponentially because of the duplicated message passing between neighbours. It forms a loop when the neighbour passes a message to each other. As the provided framework only stores ten messages. All the essential messages are overwhelmed by the junks.

2.2 Arrangement and Coordination

A thoughtful arrangement not only makes the swarm looks more elegant but also could solve the blocking issue. The blocking happens when too many vehicles are rushing to the same position. This problem directly leads to casualty cause many of them cannot reach the globe to get charged. Therefore, a way to coordinate between vehicles and make all of them charge fairly to survive longer is the key to success.

2.3 Disaster Tolerance

The problems here are various or even unpredictable. Although the system works correctly, it is always better to prepare a plan B for accidents. I kept asking myself for unexpected cases during the designing stage. What if a globe vanished? What if a vehicle disconnected from its peers? What if my local leader dies?

3 Design Decisions and Implementation

3.1 Discarded Decision: Linked Line

This is a complicated idea. It is borrowed from the implementation of a linked list. There will be four lines in a plane (i.e. flat). Each line has a head vehicle and runs in an oval orbit. All the vehicle belongs to that group will follow the previous one, so it forms a line. The head vehicle will send a message to count how many vehicles are in the group, each vehicle does a relay, add one and pass to the next. When there is no next, the size will be calculated and passed to everyone. There is always a manager sitting next to the globe. Since the line is running next to the globe, each of the vehicles will be charged when they passed the globe and exchange information with the manager. The manager will inform the head vehicle to turn back when it finds the middle vehicle of the line.

This solution requires precise message passing. I have finished the message passing part. However, when I attempted the line forming part, there are issues like, who to be the head vehicle? They might all want to be the head. How to find and join an existing line? The line is kind of tiny in a 3D world, some vehicle may never find any. What if it forms mutable lines? What if after vehicles form a line, some vehicles in the line are dying before it reaches the globe? The line will be disconnected. What's worse, during the forming stage, there might be multiple vehicles racing to be the successor of a predecessor vehicle. Thus, I decided to change this idea to a similar but much simpler one. Which will be discussed below.

3.2 Leader's Strategies on Polling, Counting, and Controlled Destruction

The manager idea from the discarded attempt is kept. It named as "leader" here. The leader will always follow the globe to update the latest globe location to everyone. It also in charge of manage and control and the whole group. The leader also keeps a local list for tracking who is alive. A vehicle will send a message to its leader when a) It needs to skip the queue; b) Just charged; c) Dying; For the first two cases, the leader will add them into the local list, for the last case, the leader will remove them from the local list. Above required precise, always connected, and highly efficient message-passing network to make sure all messages are delivered properly and quickly. Which I all achieved, and I will discuss my solutions in the message passing section.

The major benefit of this design is: it makes everything trivial. The leader does the polling to tell vehicles coming one by one. When the leader finds out the amount of its follower is greater than the target number of elements, it just asks the next coming vehicle to self-destruct.

3.3 Passing leadership, or no leadership

Since the leader always follows the globe, and my testing size is smaller than 400, this never happens. However, by theory, it could. Say there is a massive amount of vehicle, and by chance, the leader is too busy to handle other things or has been switched out by the operating system scheduler, so that it goes a bit away from the globe. Meanwhile, many vehicles come to charge and block the leader for coming back. In this case, the leader could die. I have prepared that if a leader is dying, it sends a message to a specify follower, and announce the new king. But what if some nasty things happened and the leader dies far away from the followers? The final safeguard is that I made my system independent to the leader. Which means even there is no leader, all vehicle can still go charge by themselves. But the blocking issue could happen again.

I did consider another solution but discarded due to its problem. When a vehicle finds its leader has not sent any message for a while, declare itself as the new leader. The issues are apparent, everyone might become a leader. Moreover, the time definition for not seen a leader does not indicate the death of a leader. It could be a very slow machine running the code. Therefore, I chose to make the system still works even there is no leader.

3.4 Follower's Strategies on Arrangement, Queue Skipping

To solve the issue of communication, I arranged all follower vehicles based on the globe position in a three-dimension space. This is simply done by adding a small number to x y z position from the globe. I considered implementing an advanced arrangement like rings. But I found out that my leader, network-liked message passing, and vehicle charging strategies are strong enough as long as they are all connected. So, no need to spend time making the fancy arrangement. When the vehicle number is large, it automatically forms a random sphere thanks for the anti-collision feature. I only need to make sure when the vehicle number is small, my configuration still connects all clusters. The minimum and maximum vehicle size it supports without any dying are 35 and 248. The advantages of this arrangement are trivial implication and easy changing. The downsides are a) it does not support the smaller size very well since the initial size is unknown; b) it requires an advanced technique to avoid blocking issue since many vehicles have a similar distance to their neighbours.

Only go charging after the call from leader is not enough. Some vehicles might die before its turn. To solve this problem, polling interrupt is introduced. I named it queue skipping. The vehicle who has a low battery will send a higher priority message to interrupt the leader's polling. Then the leader will handle the queue skipping request and let the vehicle come. Here came two questions. a) What if a vehicle skipped the queue and the leader do the polling on it again? The solution can be tracking their battery level and reorder the polling. I have a more natural way: When the leader asks a vehicle to come, if the vehicle has a high battery level, ignore it. Another question b) What if there are too many vehicles ask for queue skipping? Then the polling may never work. This indeed happened when the vehicle amount is large. I solved this simply by, first, each vehicle only sends one queue skipping request before they got charged for the fairness. Second, if the leader does not respond to my queue skipping request and my power level is critical, go to charge regardless. After charging, report to the leader I'm fully charged.

3.5 Network-Liked Message Passing and Spam Filter

My message passing system includes the information of origin sender, target receiver, message time stamp, and message contents like globes information. There is another time stamp for globe update time. Because a newer message does not necessarily mean the globe position inside the message is newer since I send a message not only for updating the globe but also for all other communication between vehicles. When a vehicle receives a message, it first checks if the message is for me. If yes, check if it is a newer message, look for the message contents and react accordingly. Most importantly, it stops sending this message to others. If it's not for me, steal its globe position if it's newer then forward this to others.

To make the whole message passing efficient and avoid junk message, special care is taken for the sending part. Each vehicle has a local message buffer to memories what it has sent before. If the message is the same or very similar, it will not forward it. The definition of a similar message includes the same sender and receiver with send content and very close time. What's more, there is no need to send a broadcast to update the globe position if the globe just moved a tiny distance. I also stop forward a message if it is obsolete.

3.6 Discarded: Split for each globe

I attempted to split my vehicles to several groups with their own leaders based on the globe's numbers. However, I found that my big group implementation is robust enough to handler random multiple globes. If the vehicle amount is small, splitting into different globes are risky cause the different group is not connected to each other. When a globe vanished, it might never know where the new globe is. As in the central big network, vehicles can detect other globes and tell leader to go there if needed, and everyone goes together. However, for a large group with a huge population, splitting and make full use of the globes might be the only way to let more vehicle alive.

3.7 Problems and Future Improvement

Most of the problems have discussed above. Here is just a summary of the problems.

Issue #1 When the vehicle amount is more than 400, a leader is too busy to handle all the queue skipping request. Thus, the blocking issue comes back. If there is more than one globe, it is still better to split the vehicle. This can be done by passing the vehicle amount, which counted by the leader to everyone so that each vehicle knows when to split. Be careful that multiple vehicles could race to declare its leadership.

Issue #2 In the random globes mode, some vehicles might follow another closer globe and disconnected from the primary network. When their newer globe vanishes, they can only wait to die. Alternatively, if they are lucky, they might reconnect to the primary network and survive. I think this is because a vehicle broadcast the latest globe position to everyone if it finds a globe. If the newer message from the leader has not arrived and by chance these vehicles are in critical power level and decide to charge without leader's call, they will form a new separate group that lead to this issue. A possible solution is to store all seen globe but does not treat it as the main globe. Passed this information to leader or everyone to let them know. My globe information is packed in a record type, it has the flexibility to achieve this.

4 Testing Document

I have tested my code on Windows 10 (1903), macOS 10.14 and Ubuntu 19.04. All platforms run my code smoothly. The following test results are based on ANU CSIT lab machine, which runs Ubuntu 19.04 by Intel Core i7-7700@3.60GHz eight cores CPU and built-in HD Graphics 630 GPU.

Test set 1: Single_Globe_In_Orbit mode without controlled destruction, 10 mins run for each

Initial Number	Target Number	Vehicles Left after 10 mins	Memory	CPU	Framerate
400	400	358	50.6 MB	31%	20 Hz
300	300	299/298	44.3 MB	26%	22 Hz
248	248	248 (Tested for 2 hour)	40.9 MB	23%	25 Hz
100	100	100/100/100 (Tested 3 times))	35.9 MB	13%	30 Hz (max)
64	64	64/64/64	35.3 MB	10%	30 Hz (max)
40	40	40	35.1 MB	10%	30 Hz (max)
35	35	35	35.6 MB	9%	30 Hz (max)
20	20	16	34.2 MB	8%	30 Hz (max)
10	10	4	34.7 MB	4%	30 Hz (max)

Above testing result shows that my design can handle 300 vehicles smoothly, but it does not behave well for smaller size less than 35. The reason has been analysed in the arrangement section of this report.

Test set 2: Single_Globe_In_Orbit mode with controlled destruction, 10 mins run for each

Initial Number	Target Number	Time spent to get to the target number	Vehicle amount 10 mins after reaching the target
300	230	4m : 55s	229
200	100	1m : 40s	100
64	63	45s	63
40	35	1m : 45s	35

The time spent to get to the target number depends on how fast the leader meets everyone. The speed is acceptable. The result above shows that the controlled destruction is often precise. This is also confirmed by the debug messages.

Test set 3: Random_Globes_In_Orbits mode without controlled destruction, 10 mins run for each

Initial Number	Target Number	Vehicles Left after 10 mins
240	240	179
200	200	166
100	100	100
130	130	122
80	80	67/78/80
64	64	61
41	41	39

For the random globe mode, the testing results are non-linear and different each because the globes vanish randomly. Like the test result of initial number 80, unlike single globe mode, three tested results are all different.

Test set 4: Random_Globes_In_Orbits mode with controlled destruction, 10 mins run for each

Initial Number	Target Number	Time spent to get to the target number	Vehicle amount 10 mins after reaching the target
300	80	2m : 10s	78
130	100	59s	100
64	41	1m : 31s	30

The issue of dropping after reaching the target number usually happen in random mode. The reasons are a) the problem of the design itself, vehicles still die even without controlled destruction; b) the controlled destruction make part of the network disconnected.

Test set 5: Longtime run

Mode	Initial Number	Target Number	Time spent to get to the target number	Vehicle Left after x hour
Single Globe	248	248	-	248 (x = 2 hours)
Single Globe	248	200	3m : 3s	200 (x = 1 hour)
Random Globes	100	100	-	70 (x = 1 hour)
Random Globes	130	100	2m : 59s	60 (x = 1 hour)