

Distributed Chat System

Hardik Bhati,

Alka Trivedi,

Subham Kumar,

Nidhi Kamewar,

Harsh Ranjan

iit2019013@iiita.ac.in, iit2019055@iiita.ac.in, iit2019093@iiita.ac.in, iit2019189@iiita.ac.in, iec2019036@iiita.ac.in

Department of Information Technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India

Abstract-- In this project, we describe an implementation of a Distributed Chat System along with two algorithms, Leader Election with Bully Algorithm and Resource Deadlock Detection. The focus has been on trying to make the system more scalable and robust at the same time by keeping blocking connections between users and not using buffers for receiving but instead alternating between sending and receiving messages without any buffer overheads.

Keywords— chat system, user, sender, receiver, bully algorithm, leader election algorithm, causal ordering, resource deadlock detection, socket programming etc.

INTRODUCTION

A distributed chat system is where multiple users can chat among themselves in a distributed manner. To implement this we have used **socket programming** for the connection between the users, **causal ordering** is maintained by using blocking connections so that everything is in order. **Leader election algorithm** is a way to choose a new leader among the different users currently online and the **Resource deadlock process** is a way to prevent deadlock between resources in demand from all users so as to make the communication process smoother between the users without the blockage of resources for any user.

There are generally two methods for communication between different users.

Centralized Server approaches try to get all messages at a common server and distribute them afterwards to all the respective users. This is a common approach but it has some drawbacks in that it does not scale well (since it needs vertical scaling). This approach is also not as secure as all the

information of the messages can be intercepted if the server is broken into.

Distributed Model approaches try to model the whole system with as little workload on a single client as possible. Each for its own is the motto here as the communication is directly between 2 parties in a communication and no 3rd party server is needed. It scales well with the number of users too as the scaling is horizontal. The approach is much more secure and more and more users are shifting to this approach for more security and privacy.

We are using socket programming in Python 3.8 as a base for our Chat Model.

Socket Programming: Socket programming helps in connecting different nodes. Here one user will act as a listener socket node with a particular ID (in our case the IP address/Port) while the other node will act as a sender socket node which will send the message to the other node (listener). Since each of the nodes have a common text file, this text file mentions the IP and port numbers of each of the users. We automated this process by opening various terminals at the same time from python with each having the same text file for messages to respective IP & Port.

Causal Ordering: Causal Ordering is an important requirement when working with chat based applications, it maintains the order of messages between users. With our system we have maintained a Causal Ordering between users by using Blocking Connections between different users. Since each of the nodes has to read messages from a common text file. If a node knows it has to send a particular message first, it will continue to send the message over and over again till it gets an acknowledgement. If a node is at the receiving end, it keeps on listening for the particular message from sender X (it knows sender X is going to send a message from a text file). If it receives a message from some other user it checks with IP and does not send an acknowledgement until the IP matches with that of sender X, and hence the system maintains the causal ordering of messages.

Two algorithms we implement are simple to follow and easy to understand and are aimed at easier understanding of working behind some fundamental principles in Distributed Systems.

Leader Election Algorithm:

Leader election follows the concept of giving a particular process in a distributed system some special powers. Those special powers can include the liberty to assign work to other nodes in the network, or even the responsibility of handling all the work in a distributed system.

One such leader election algorithm is bully algorithm which is as follows-

- If a node notices that the leader is not online, it sends election messages to all processes with more power.
- In case a node receives a reply from a process with more power, then it gives up the ambition to become a leader and it waits for the leader. But if no one responds, it declares itself the new leader and further broadcasts it.
- If a node receives a broadcast with a new leader, it accepts and changes its leader.

We simulated this process with different users sending different messages to each other randomly and if the leader seems missing to someone, it initiates the leader election algo. Once the leader is found, it starts with its normal process again, thus maintaining synchronization among all users.

Resource Deadlock:

A deadlock is a situation where a process or a set of processes request resources that are occupied by other processes. If resources are not allocated properly, the situation can result in a deadlock. Distributed systems are so vast that preventing and avoiding deadlocks is complex and highly inefficient. Therefore we only detect them.

A deadlock detection algorithm must satisfies 2 conditions-

The algorithm should be able to detect all the deadlocks in the system and the algorithm should not detect false deadlocks.

Approaches to detect deadlock in a distributed system-

Centralized approach where only one resource is responsible for detecting deadlock in the system.

Distributed approach where different resources work together to detect deadlock in the system.

Hierarchical approach where it is a combination of both centralized and hierarchical approach, some specific nodes are chosen to detect deadlock and all of them are handled by a single resource.

CONTRIBUTIONS

<u>Group Members</u>	<u>Contributions</u>
Hardik Bhati(IIT2019013)	Implemented the chat system using Python 3.8.

	Implemented Bully Algo for leader election, Joined deadlock detection with the chat system for message transfer. Documentation.
Alka Trivedi(IIT2019055)	GUI development,pseudo code of leader election,documentation
Subham Kumar(IIT2019093)	Implemented resource deadlock detection and helped in implementing chat system,documentation
Nidhi Kamewar(IIT2019189)	Pseudo code of leader election,documentation
Harsh Ranjan(IEC2019036)	Pseudo code of bully code, documentation

METHODOLOGY

Task 1: Generating a random text file containing <sender,receiver,message> in each line and directing each node to execute the messages as given in text file

Implementation:

The code works works as follows:

Firstly a random file(text.txt) will be generated where some first lines will mention Users, Ip, Port, Username for all users present in the system.

Then, randomly, X messages will be appended to the end of the file. Sender, Receiver, Message will be tagged along.

N random users are created with separate folders and this file is copied in each of them and each of them execute the main python file separately simulating a real distributed chat system.

Each of the users will start by adding each user's IP, Port, Username for further calls. It then starts with going through the whole text.txt:

It will pick the message whose sender id matches its id in the order given in the text.txt file and will send the message continuously to the receiver IP until it receives an acknowledgement.

It will pick the message whose receiver IP matches its IP in the order given in the text.txt file and it will wait for the message from sender IP by checking for IP if it receives any message. It then moves ahead only after receiving the message from the correct IP and giving an acknowledgement back to the sender.

This will ensure that the messages will be received and sent in order of the text file and the simulation will preserve the causal ordering of the text file.

Further we have also added GUI in our application for task-1. For which, we have used the standard GUI library for python i.e.,Tkinter. There will be a Tkinter window which we have called the “user chat window” for every user in the application. The Tkinter window will contain the messages of the user's chat which that particular user will receive.

Output:

Text.txt:

```
1 Users
2 10.0.2.15,2881,user1
3 10.0.2.15,2882,user2
4 10.0.2.15,2883,user3
5 10.0.2.15,2884,user4
6 10.0.2.15,2885,user5
7 10.0.2.15,2886,user6
8 Message
9 user3,user1,Hi from user3 to user1 with message number 1
10 user2,user5,Hi from user2 to user5 with message number 2
11 user5,user3,Hi from user5 to user3 with message number 3
12 user4,user6,Hi from user4 to user6 with message number 4
13 user5,user3,Hi from user5 to user3 with message number 5
14 user5,user4,Hi from user5 to user4 with message number 6
15 user2,user6,Hi from user2 to user6 with message number 7
16 user2,user4,Hi from user2 to user4 with message number 8
17 user2,user4,Hi from user2 to user4 with message number 9
18 user5,user2,Hi from user5 to user2 with message number 10
19 user5,user3,Hi from user5 to user3 with message number 11
20 user4,user1,Hi from user4 to user1 with message number 12
21 user3,user5,Hi from user3 to user5 with message number 13
22 user5,user6,Hi from user5 to user6 with message number 14
23 user2,user6,Hi from user2 to user6 with message number 15
24 user4,user5,Hi from user4 to user5 with message number 16
25 user4,user5,Hi from user4 to user5 with message number 17
26 user1,user2,Hi from user1 to user2 with message number 18
27 user4,user2,Hi from user4 to user2 with message number 19
28 user5,user3,Hi from user5 to user3 with message number 20
29 user3,user5,Hi from user3 to user5 with message number 21
30 user3,user6,Hi from user3 to user6 with message number 22
31 user2,user6,Hi from user2 to user6 with message number 23
32 user5,user3,Hi from user5 to user3 with message number 24
33 user1,user3,Hi from user1 to user3 with message number 25
34 user6,user4,Hi from user6 to user4 with message number 26
35 user2,user3,Hi from user2 to user3 with message number 27
36 user4,user6,Hi from user4 to user6 with message number 28
37 user4,user6,Hi from user4 to user6 with message number 29
38 user1,user4,Hi from user1 to user4 with message number 30
39 user4,user1,Hi from user4 to user1 with message number 31
40 user5,user4,Hi from user5 to user4 with message number 32
41 user6,user3,Hi from user6 to user3 with message number 33
42 user1,user4,Hi from user1 to user4 with message number 34
43 user5,user4,Hi from user5 to user4 with message number 35
44 user1,user6,Hi from user1 to user6 with message number 36
45 user5,user1,Hi from user5 to user1 with message number 37
46 user3,user2,Hi from user3 to user2 with message number 38
47 user5,user1,Hi from user5 to user1 with message number 39
```

Message Window:

```
user1
Hi from user3 to user1 with message number 1
Hi from user4 to user1 with message number 12
Hi from user4 to user1 with message number 31
Hi from user5 to user1 with message number 37
Hi from user5 to user1 with message number 39
Hi from user6 to user1 with message number 40
Hi from user5 to user1 with message number 43
Hi from user4 to user1 with message number 44
```

```
user2
Hi from user5 to user2 with message number 10
Hi from user1 to user2 with message number 18
Hi from user4 to user2 with message number 19
Hi from user3 to user2 with message number 38
Hi from user1 to user2 with message number 47
```

```
user3
Hi from user5 to user3 with message number 3
Hi from user5 to user3 with message number 5
Hi from user5 to user3 with message number 11
Hi from user5 to user3 with message number 20
Hi from user5 to user3 with message number 24
Hi from user1 to user3 with message number 25
Hi from user2 to user3 with message number 27
Hi from user6 to user3 with message number 33
Hi from user4 to user3 with message number 48
```

```
user4
Hi from user5 to user4 with message number 6
Hi from user2 to user4 with message number 8
Hi from user2 to user4 with message number 9
Hi from user6 to user4 with message number 26
Hi from user1 to user4 with message number 30
Hi from user5 to user4 with message number 32
Hi from user1 to user4 with message number 34
Hi from user5 to user4 with message number 35
Hi from user3 to user4 with message number 41
Hi from user6 to user4 with message number 42
Hi from user5 to user4 with message number 45
Hi from user1 to user4 with message number 50
```

```
user5
Hi from user2 to user5 with message number 2
Hi from user3 to user5 with message number 13
Hi from user4 to user5 with message number 16
Hi from user4 to user5 with message number 17
Hi from user3 to user5 with message number 21
Hi from user2 to user5 with message number 49
```

```
user6
Hi from user4 to user6 with message number 4
Hi from user2 to user6 with message number 7
Hi from user5 to user6 with message number 14
Hi from user2 to user6 with message number 15
Hi from user3 to user6 with message number 22
Hi from user2 to user6 with message number 23
Hi from user4 to user6 with message number 28
Hi from user4 to user6 with message number 29
Hi from user1 to user6 with message number 36
Hi from user3 to user6 with message number 46
```

Task 2: Implementing Bully Algorithm for Leader Election

The code works as follows:

N user folders are opened and each automatically runs a terminal and each of them continuously sends a message to other terminals simulating a real chat application.

There is a leader before the start, so everything happens normally until we forcefully close the leader.

Each of the terminals sends a message 5 times unless it receives an acknowledgement and then assumes the user is offline if it still gets no response.

Each of the other terminals sends messages to one another until one of them detects that the leader is not responding.

Now since we are not using any buffers, we alternate between sending and receiving so as to do both tasks simultaneously.

To prevent the other user from assuming the other user is down when in actuality it was just sending another message, we try to alternate between receiving when a user is not responding to allow the other user to finish its job if it is busy. As and when before 5 tries the user responds, we know the other terminal is not down.

But when someone detects that the leader is down, it initiates the leader election algorithm by sending the election message to all users that have a bigger ID than it. It waits for a message from any of them. If any of them does not respond, it knows it has the most power of all online, so it broadcasts that it is the leader to everyone.

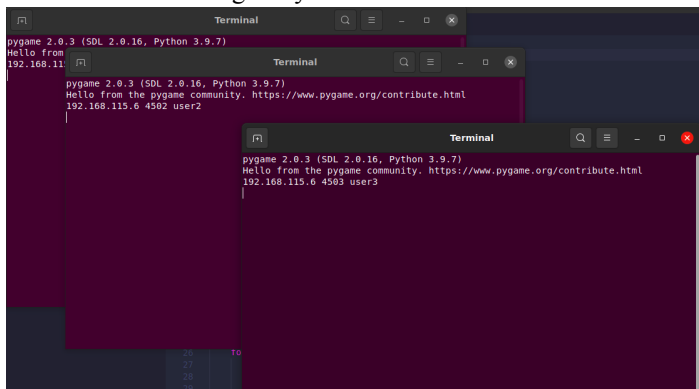
But if anyone responds back, it stops the initiation process and continues with its work.

Any user if it gets a leader election message from a user with lesser power, it responds with an "OK" message to tell it that it will take the responsibility for conducting further business to detect who is the new leader and it initiates the leader election from its end.

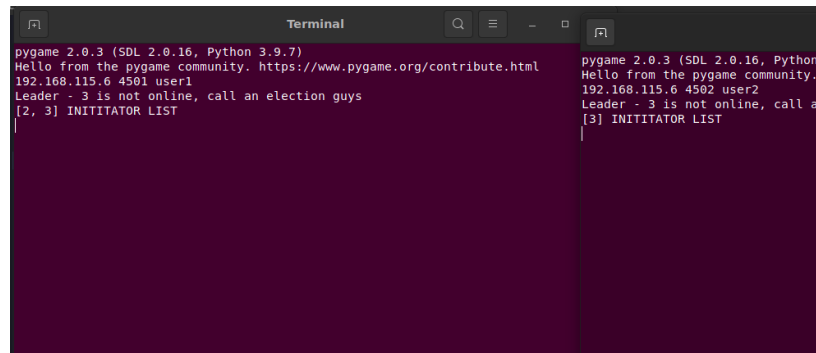
One problem that we faced from our end was when two people simultaneously started an initiation process and sent an election message to the person above, which resulted in some synchronization errors. But we resolved it by adding a time limit for a new leader election which only allows a user to start leader election after some fixed time interval.

Output:

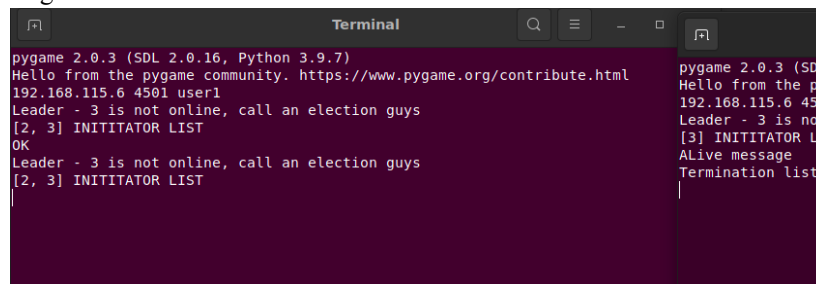
1: 3 terminals working in Sync.



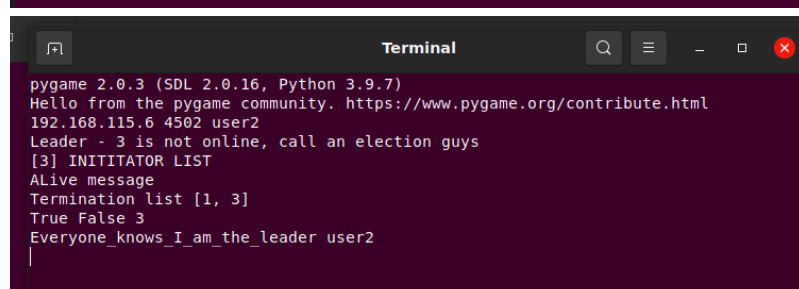
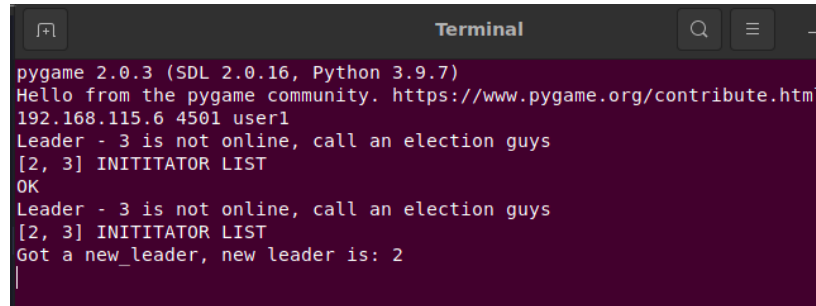
2. After closing the leader terminal, terminal 1st detects the leader is missing and it starts the initiation process. In tandem. 2nd also detects that the leader is missing so it starts its own initiation process.



3. 1st terminal messages 2nd about the passing of the leader but the leader responds with "OK" after which it stops but soon after it again starts its election process since it got no information in a long time about the leader.



4. 2nd terminal confirms it got no message from anyone above it and declares termination by sending a global message about it being the new leader.



Task 3: Implementing Resource Deadlock Detection

The code works as follows:

Once the leader is decided it will execute the resource deadlock detection work in two phases.

It starts with Collecting information about waiting for graphs .

For a fixed time limit, the leader will send a message to each node asking each node to send the list of its dependencies on other nodes for resources.

Each node will reply to the leader with the list of nodes on which it is dependent.

On receiving a list of dependencies from a node the leader will add that dependency information to the text file named tpoint.txt. Once the time limit ends, it detects the deadlock using wait for graphs.

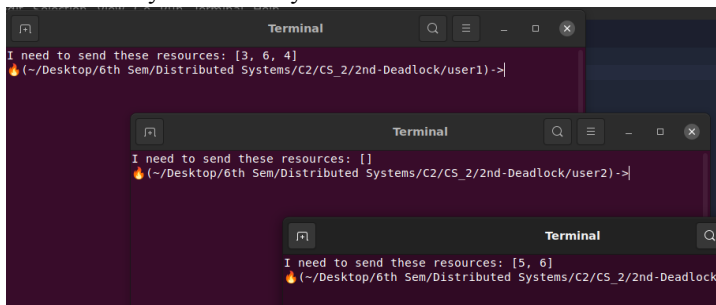
It then reads the text file and then the leader will make a directed graph(wait for graph) and a deadlock will be detected if the graph is containing a cycle else not.

For this the algorithm is as follows-

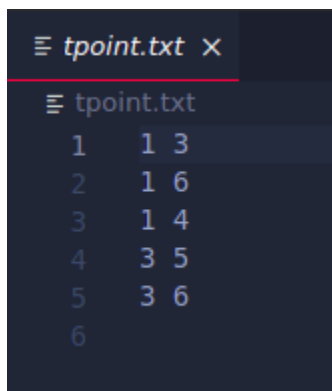
- i) traverse in dfs manner and keep a track of all nodes on path from root to current node and also separately keep a track of visited nodes till now
- ii) move only to those neighboring nodes who are not yet visited.
- iii) if you encounter a situation where the node which you are going to visit in your next dfs traversal has occurred in the path from root to current node then declare deadlock detected and terminate.

Output:

1. Every node conveys its need for resources to the leader.



2. Leader writes this down in a local file and then applies deadlock detection.



3. Leader sends out the output to every terminal.

```
1 3
1 6
1 4
3 5
3 6
No not in deadlock
```

FUTURE SCOPE AND CONCLUSION

This model can be further used in distributed applications which require to execute a set of instructions while preserving the ordering of instructions. This model can also be used as a prototype for learning how different basic algorithms are implemented in a Distributed Chat System. Further we can also implement more accurate and faster algorithms in our application to reduce the traffic and running times for the model.