UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

CENTRO DE CIÊNCIAS E TECNOLOGIAS

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO FINAL: PROJETO SISTEMA DE BATE-PAPO DISTRIBUÍDO EM P2P

DANIEL TERRA

GABRIEL GRAVINA

JOÃO BOSCO

PAULO ROBERTO

Dezembro, 2022

Sumário

1. Introdução

O projeto deste semestre se propõe a desenvolver uma aplicação que possibilita a comunicação de forma descentralizada de sistemas por meio de cliente-servidor e P2P; é um modelo de comunicação descentralizado no qual cada parte tem os mesmos recursos e qualquer uma das partes pode iniciar uma sessão de comunicação. Ao contrário do modelo cliente-servidor, no qual o cliente faz uma solicitação de serviço e o servidor atende à solicitação, o modelo de rede P2P permite que cada nó funcione como cliente e servidor (Rosencrance). Tendo em mente que a crescente demanda por conexão uns com os outros em todo o mundo provou ser uma benção para o crescimento das plataformas de mídia social para comunicação de pessoas. Nos últimos anos com a crescente demanda por mais privacidade entre os usuários e a falta de confiança em relação a essas organizações tornou as pessoas céticas em relação ao uso desses aplicativos para casos de uso de comunicação extremamente seguros. Assim, é necessário um sistema sem organização central no controle e uma rede sem confiança. Uma rede sem confiança é uma rede que não exige que os usuários confiem em nenhuma autoridade central. Esses nós , como um grupo, podem ser confiáveis para tomar uma decisão que se alinhe com os interesses de todos. Isso é acompanhado por uma rede distribuída, onde os pares são conectados uns aos outros para formar uma rede semelhante a uma malha. Essa rede distribuída de pares (P2P) conectados é frequentemente atribuída como a internet de amanhã (“Home”).

Com isso em mente, o professor propôs no início do semestre a conclusão da tarefa de desbravar esse desafio de compreender e entender como funciona a comunicação entre usuários na internet e de desenvolver um projeto simples para a comunicação entre os outros sistemas desenvolvidos pelos colegas de aula.

Desse modo, foi possível alcançar um aprendizado autodidata, compreender mais sobre essas tecnologias e apresentar o sistema desenvolvido pela equipe.

2. Metodologia

O projeto foi dividido em cinco partes, cada uma com um objetivo específico para o desenvolvimento do projeto principal, que no final se tornaria um.

2.1. #CheckPoint 1

Nessa parte era necessário dividir em grupos de quatro pessoas, definir a linguagem a ser utilizada e estudar a base teórica enviada pelo professor. Ficou definido que o sistema seria desenvolvido orientado a objetos, pois, além de mais simples e funcional, é o método correto para ser trabalhar com RMI/RPC.

Também se definiu a estrutura única que todos os sistemas deveriam ter nas chamadas, funções, procedimentos, classes e objetos. Abaixo o código da estrutura:

```
class Usuario

    def enviar_msg (self, instancia, msg):
    instancia.receber_msg(self.nome, msg)

    def receber_msg(self, remetente, msg):
    print(remetente + ”:”+ msg)

    def __init__(self, nome):
    self.nome = nome

    def main ():
    instancia_Joao = Usuario (“Joao”)
    instancia_Maria = Usuario (“Maria”)
    instancia_joao.enviar_msg(instancia_maria, “Olá !”)
    main ()
```

2.2. #CheckPoint 2

Nesse passo era necessário desenvolver um programa primário de troca de mensagens de maneira local, usando como base a estrutura do CheckPoint 1. O grupo decidiu por usar a linguagem Ruby para desenvolver o projeto, de maneira inicial.

2.3. #CheckPoint 3

Nessa etapa era necessário testar a usabilidade de uma biblioteca comum usada em sistemas distribuídos, o Redis RPC. Foi testado também a compatibilidade com o projeto e com as linguagens escolhidas pelos grupos. Abaixo temos o código em Ruby do sistema rodando localmente usando o Redis RPC:

Cliente.rb

```
send_message messages

# 2. Remote object, should act like local object
redis_server = Redis.new
message_queue = 'message'
timeout = 1
messages = RedisRpc::Client.new redis_server, message_queue, timeout
send_message messa#!/usr/bin/env ruby

require 'redis'

require File.expand_path('../../lib/redis-rpc', __FILE__)
require File.expand_path('../message', __FILE__)

def assert(cond)
    if not cond
        fail 'assertion failed'
    end
end



def send_message(messages)

    user0 = "Gabriel"
    user1 = "Paulo"
    messages.send_message(user0, "Olá!")
    messages.send_message(user1, "Espere")


    begin
        messages.missing_method
        assert false
    rescue NoMethodError
    rescue RedisRpc::RemoteException
    end
end

# 1. Local object
messages = Message.newges
puts 'success!'
```

message.rb

```
class Message
    # A simple, mutable calculator used for testing.
    # referenced explicitly in ../spec/calculator.spec.rb

    def initialize
        @message = ""
    end

    def send_message(usr, message)
        @message = "#{usr}: #{message}"
    end

    def receive_message()
        return @message
    end

end
```

server.rb

```ruby
#!/usr/bin/env ruby

require 'redis'

require File.expand_path('../../lib/redis-rpc', __FILE__)
require File.expand_path('../message', __FILE__)

redis_server = Redis.new
message_queue = 'message'
local_object = Message.new
server = RedisRpc::Server.new redis_server, message_queue, local_object, verbose: true
server.run
```

O código do projeto se encontra em: https://github.com/GabrielGravina/Ruby-RPC/tree/main/examples.

2.4. #CheckPoint 4

Nesse ponto espera-se que o projeto já consiga funcionar em rede e se comunicar com outros projetos dos outros grupos. Com isso, foram definidos dois objetivos: testar o funcionamento do projeto em rede, e testar a comunicação com algum grupo. Também ficou-se definido que seria utilizado o método get_msg no formato JSON.

Também foi definido que, a partir desse ponto, o sistema se comunicaria em P2P. Como temos pouquíssima referência de trabalhos em P2P em Ruby, optamos por trocar a nossa linguagem para Python 2, visto que temos um vasto material tanto de tutoriais, quanto de artigos.

Inicialmente tivemos que reestruturar o código quase que por completo para que ele funcionasse em P2P, e isso fez com que algumas variáveis primárias definidas no CheckPoint 1 ou fossem estendidas com mais objetos, ou modificadas para atender as necessidades do projeto. Mas como o projeto não seria mais um chat que funciona via RMI/RPC, não teve prejuízo.

Como uma das exigências do projeto era a utilização de uma interface, optou-se por usar a biblioteca do Python Tkinter, por ser bastante simples e intuitiva.

O código é uma mistura de códigos bloqueantes e não-bloqueantes, visto que eu preciso aguardar a conexão com um dos membros do chat para que o sistema volte a receber chamadas. Em contrapartida, uma vez que a conexão seja

estabelecida, as mensagens serão enviadas e recebidas por meio de pilha, resultando em um procedimento não-bloqueante.

As funções que temos no projeto são: Definir cliente, “escutar” cliente (receber mensagem), adicionar amigo, visualizar mensagens, enviar mensagens, escrever mensagem, adicionar e remover amigos, verificar status.

Na rede P2P, um grupo de computadores é conectado com permissões e responsabilidades iguais para o processamento de dados. Ao contrário da rede cliente-servidor tradicional, nenhum dispositivo em uma rede P2P é designado apenas para servir ou receber dados. Podemos ver isso claramente no projeto pois é necessário se criar um servidor (o usuário) para depois se adicionar um “amigo”, que também tem status de servidor para que essa troca de mensagens ocorra.

Nosso projeto se baseia em uma rede P2P não estruturada. Neste tipo de rede P2P, cada dispositivo é capaz de fazer uma contribuição igual. Essa rede é fácil de construir, pois os dispositivos podem ser conectados aleatoriamente na rede.

O código do trabalho se encontra no repositório do GitHub pelo link:

2.5. #CheckPoint 5

Finalmente, aqui espera-se que o sistema funcione em rede com todos os sistemas desenvolvidos durante a disciplina.

Inicialmente, foi testado o programa rodando em uma rede VPN e em computadores diferentes para testar o sucesso da implementação. Abaixo segue as telas.

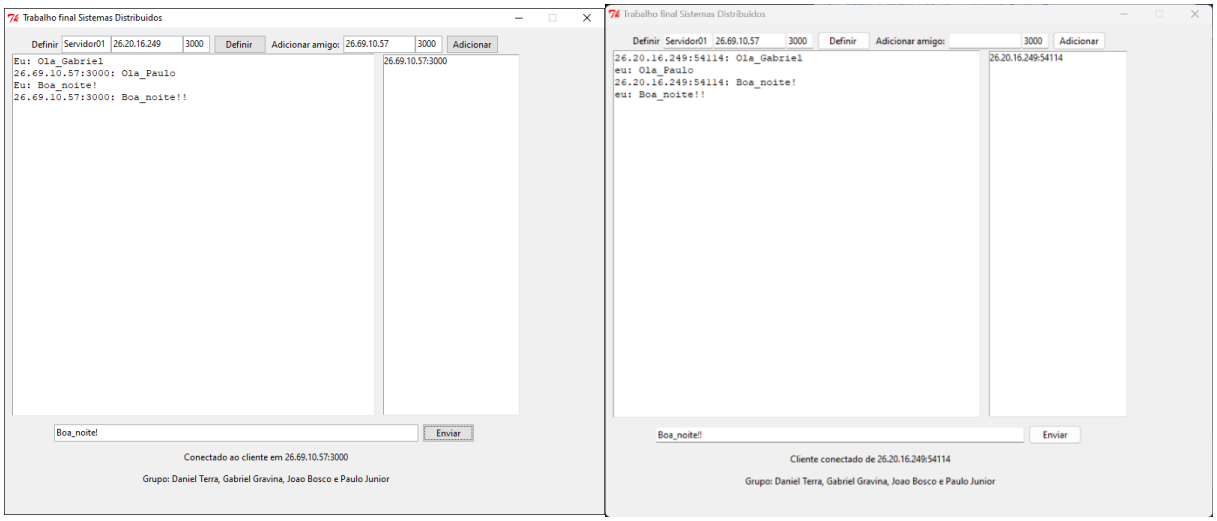

Figura 1: Capturas de tela do sistema rodando em rede

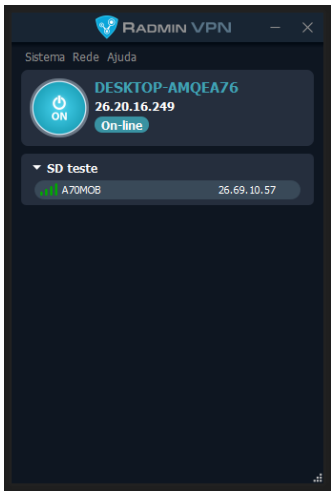

Figura 2: Programa de VPN usado para conectar à rede

2.6. Participação dos membros do grupo

A divisão das tarefas do projeto não foi necessariamente estruturada. Os membros do grupo faziam todas as tarefas impostas de acordo com seu conhecimento. Mas podemos destacar as participações mais importantes de cada membro nas seguintes áreas: Daniel terra: pesquisa, aplicação do código de P2P e elaboração do relatório inicial; Gabriel Gravina: criação da interface e padronização do código; João Bosco: pesquisa teórica e adaptação dos códigos-exemplos que foram achados; Paulo Júnior: Pesquisa teórica, adaptação do código e escrita do relatório final.

3. Resultados

O sistema foi implementado, inicialmente, em Ruby mas, como a diretriz do projeto foi modificado para ser em P2P, foi decidido por trocar a linguagem usada para Pyhton 2, visto que era uma linguagem mais difundida, com mais material de referência e facilidade.

A experiência da equipe que participou do projeto com peer-to-peer era zero, portanto, tivemos que pesquisar bastante para fazer o projeto funcionar de maneira mínima e satisfatória. Foi usado bastante material de internet, vídeos do YouTube, artigos de implementação P2P em Python. O que a equipe fez foi usar tutoriais da

internet e mesclar códigos de exemplo para fazer o sistema funcionar, onde se destaca tutoriais de uso da biblioteca Tkinter para a criação da interface e tutoriais para criação de redes P2P via Python. A maneira mais eficiente que achamos de criar o projeto foi o uso do Python 2, que já existia muitos exemplos prontos.

Mas, como toda a adversidade, se conseguiu fazer o projeto funcionar via rede, onde ambos os terminais eram cliente e servidores, onde se colocava seu IP público como servidor e adicionava um outro IP público que foi adicionado em outro programa do projeto da rede para se estabelecer a conexão e poder se trocar as mensagens. O projeto conta com algumas limitações, tais como não possuir backup de chats, não se poder renomear os participantes do chat. Também só existe uma maneira de se conectar via servidor, que é ambos se definirem como servidor e “escutar” antes de se estabelecer uma conexão.

4. Conclusão

A equipe achou o trabalho bastante complexo e desafiador, tendo enfrentado muitos desafios no decorrer do processo. O principal desafio foi a falta de um referencial teórico claro, pois tivemos que nos basear em muitos trabalhos prontos disponíveis na internet para conseguir implementar o trabalho.

Pode-se aprender sobre como funciona um compartilhamento P2P, como sua estrutura é mais simples do que se imaginava e como a segurança nesse tipo de conexão deve ser sempre priorizada, pois ele deixa muitas brechas.

Uma área que seria interessante de se explorar também seria na questão de segurança, pois atualmente nosso projeto deixa inúmeras brechas onde o sistema poderia ser invadido. Não focamos nessa área pois não era o objetivo.

Outra ideia seria criar um sistema de login e senha para autenticar no chat e escolha de nomes de usuário para identificar de maneira mais simples quem está enviando as mensagens.

5. Referências

ALMEIDA FILHO, João L. Sistemas Distribuídos. Universidade Estadual do Norte Fluminense, 2021. Slide 73;

NAGPAL, Aman. How to create your own decentralized file sharing service using python. Boston University, 27 de agosto de 2018;

WHAT IS P2P(PEER-TO-PEER PROCESS)?, disponível em: https://www.geeksforgeeks.org/what-is-p2ppeer-to-peer-process/, acessado em 13 de dezembro de 2022;

P2P TWISTED CHAT | PYTHON, disponível em: https://www.youtube.com/watch?v=1Fay1pjttLg, acessado em 11 de dezembro de 2022;

REDE P2P PYTHON EM PORTUGUÊS, disponível em: https://www.youtube.com/watch?v=YZh074yhfN4, acessado em 11 de dezembro de 2022;

UDP PEER-TO-PEER MESSAGING WITH PYTHON, disponível em: https://www.youtube.com/watch?v=IbzGL_tjmv4, acessado em 11 de dezembro de 2022;

COMO CRIAR UMA TELA EM PYTHON PARA SEUS CÓDIGOS - [INTERFACE GRÁFICA INTUITIVA COM TKINTER]. Disponível em: https://www.youtube.com/watch?v=AiBC01p58oI, acessado em 12 de dezembro de 2022.