COMPUTER NETWORKS LAB REPORT

CSS652

NETWORK KNIGHTS

20CS8099
20CS8032
20CS8006
20CS8073
20CS8021

Problem Statement

Consider a client server application that acts as a Matrix Decomposer. There will be two distributed servers which will perform the two different computations L matrix and U matrix. One central server will operate these two individual servers and return the matrices to the client. The client should also verify the formula A=LU, where A is the input matrix and L and U are two different matrices. Implement this application using the concepts you have learned so far.

SOLUTION

Introduction

The Matrix Decomposer is a client-server application that performs the decomposition of an input matrix A into its lower triangular matrix L and upper triangular matrix U using Doolittle's algorithm. The application consists of two distributed servers that perform the computations for L and U matrices respectively, and a central server that coordinates the operation and returns the matrices to the client. This report outlines the design and implementation of the Matrix Decomposer application using the concepts learned so far, including distributed computing, client-server architecture, and Doolittle's algorithm. The report also includes the verification of the formula A=LU to ensure the correctness of the computed matrices.

Approach

Matrix decomposition is an essential technique used in various fields, such as physics, engineering, and machine learning. Two popular methods of matrix decomposition are Crout's and Doolittle's algorithm. In this project, we have implemented Doolittle's method using the numpy library to perform matrix operations. The project's aim is to design a system that enables users to decompose a matrix into its triangular

components using distributed servers and a central server.

The project uses a central server that manages two distributed servers. The servers calculate the decomposed triangular matrix using the Doolittle's algorithm. To resolve server names to IP addresses, a global DNS server is used. The DNS server addresses are known beforehand, allowing clients and servers to connect to the DNS servers to resolve server names.

The Doolittle's algorithm was used to develop the project's matrix decomposition. The numpy library was utilized to perform the necessary matrix operations. The algorithm returns a unit lower triangular matrix and an upper triangular matrix separately in two different functions.

The central server sends requests to the distributed servers in parallel on two different threads to enable parallel processing. The server waits for both servers to respond with their decomposed matrix before returning them to the client. The central server handles each client on a separate thread to allow for multiple concurrent requests.

The client receives the decomposed matrices from the central server and checks if the product of the matrices is equal to the original matrix provided. If the result is correct, the matrices are outputted to the user.

Code:

https://github.com/sswastik02/matrix-decomposer (Github Repository Link)

src/lib/network/server.py

Python

```
import socket
import threading
import typing
import sys
from ..utils.logger import log
from constants.servers import DNS
class Server:
    def __init__(self, host: str, port: int, server_name: str)
-> None:
        self.s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.host = host
        self.port = port
        self.server_name = server_name
        self.s.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)

    def set_client_handler(self, client_handler:
typing.Callable[[socket.socket, any], None]) -> None:
        "Sets a client handler for the server object"
        self.client_handler = client_handler

    def listener(self) -> None:
        "Listens for new connections and starting in threads"
        while True:
            cli, ip = self.s.accept()
            log.info(f'[*] New Connection : {ip}')
            client_thread = threading.Thread(
                target=self.client_handler, args=(cli, ip)
            )
            client_thread.start()

    def register_dns(self) -> None:
```

```
        with self.s:
            try:
                self.s.connect((DNS.host, DNS.port))
            except ConnectionRefusedError:
                log.fatal(
                    f'{DNS.host}:{DNS.port} is not reachable,
make sure DNS is running')
                sys.exit(1)

self.s.send(f'{self.server_name}@{self.host}:{self.port}'.enco
de())
        self.s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.s.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
        # Workaround to reuse socket

    def start(self, n: int = 5, register_dns: bool = True) ->
None:
        "Starting Server with a listener"
        if register_dns:
            self.register_dns()
        try:
            self.s.bind((self.host, self.port))
        except OSError:
            log.fatal(
                f'{self.host}:{self.port} is already in use,
use another port using environment variables')
            sys.exit(1)

        self.s.listen(n)
        log.info(f"Listening on {self.host}:{self.port}")
        try:
            self.listener()
        except KeyboardInterrupt:
            log.critical("Exiting SIGINT recieved")
            self.s.shutdown(socket.SHUT_RDWR)
            self.s.close()
            sys.exit(0)
```

src/lib/network/client.py

Python

```
import socket
import typing
import sys

from ..utils.logger import log
from constants.servers import DNS
from constants.network import MSG_LEN

class Client:
    def __init__(self, name: str = None) -> None:
        self.s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.name = name
        if ':' in name:
            self.host,self.port = name.split(':')
            self.port = int(self.port)
        else:
            self.resolve_dns()

    def set_interface(self, interface:
typing.Callable[[socket.socket], None]) -> None:
        "Sets an interface for the client"
        self.interface = interface

    def run_interface(self) -> None:
        "Runs the interface"
        self.interface(self.s)
    def resolve_dns(self) -> None:
        with self.s:
            try:
                self.s.connect((DNS.host, DNS.port))
            except ConnectionRefusedError:
                log.fatal(
                    f'{DNS.host}:{DNS.port} is not reachable,
make sure DNS is running')
                sys.exit(1)
            self.s.send(self.name.encode())
            ip = self.s.recv(MSG_LEN).decode()
            host, port = ip.split(':')
            port = int(port)
            self.host, self.port = host, port
        self.s = socket.socket(socket.AF_INET,
```

```
socket.SOCK_STREAM)
        self.s.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
        # Workaround to reuse socket

    def connect(self, run_interface: bool = True) -> None:
        "Connects to the pre-specified host and port"
        try:
            self.s.connect((self.host, self.port))
        except ConnectionRefusedError:
            log.fatal(
                f'{self.host}:{self.port} is not reachable,
make sure it is running')
            sys.exit(1)
        if run_interface:
            self.interface(self.s)
```

src/lib/network/matrix.py

Python

```
import socket
import struct
import pickle
from numpy.typing import ArrayLike

def send_matrix(s: socket.socket, matrix: ArrayLike) -> None:
    "Sends matrix over socket"
    encodedMatrix = pickle.dumps(matrix)
    msg = struct.pack('>I', len(encodedMatrix)) +
encodedMatrix
    s.sendall(msg)
def recv_matrix(s: socket.socket) -> ArrayLike:
    "Recieves matrix over a socket"
    raw_msg_len = s.recv(4)
    msg_len = struct.unpack('>I', raw_msg_len)[0]
    matrix = s.recv(msg_len)
    matrix = pickle.loads(matrix)
    return matrix
```

src/lib/math/matrix.py

Python

```
import numpy as np


def uDecomposition(matrix):
    n = len(matrix)
    helper = np.zeros((n, n))
    upper = np.zeros((n, n))

    # Decomposition
    for i in range(n):
        # Upper Triangular
        for j in range(i, n):
            sum = 0
            for k in range(i):
                sum += (helper[i][k] * upper[k][j])
            upper[i][j] = matrix[i][j] - sum

        for j in range(i, n):
            if i == j:
                helper[i][i] = 1  # Diagonal as 1
            else:
                sum = 0
                for k in range(i):
                    sum += (helper[j][k] * upper[k][i])
                helper[j][i] = (matrix[j][i] - sum) /
upper[i][i]

    return upper


def lDecomposition(matrix):
    n = len(matrix)
    lower = np.zeros((n, n))
    helper = np.zeros((n, n))

    # Decomposition
```

```
    for i in range(n):
        # Lower Triangular
        for j in range(i, n):
            sum = 0
            for k in range(i):
                sum += (lower[i][k] * helper[k][j])
            helper[i][j] = matrix[i][j] - sum

        for j in range(i, n):
            if i == j:
                lower[i][i] = 1  # Diagonal as 1
            else:
                sum = 0
                for k in range(i):
                    sum += (lower[j][k] * helper[k][i])
                lower[j][i] = (matrix[j][i] - sum) /
helper[i][i]

    return lower
```

src/constants/servers.py

Python

```python
from lib.utils.dotdict import DotDict

CENTRAL_SERVER = DotDict({
    "host": "localhost",
    "port": 8000
})

L_SERVER = DotDict({
    "host": "localhost",
    "port": 7000
})

U_SERVER = DotDict({
    "host": "localhost",
    "port": 6000
})

DNS = DotDict({
    "host": "localhost",
    "port": 4000
})
```

src/constants/network.py

Python

```python
MSG_LEN:int = 1024
```

src/client.py

Python

```python
import socket
import numpy as np

from lib.network.client import Client
from lib.network.matrix import send_matrix, recv_matrix
from constants.network import MSG_LEN
from lib.utils.logger import log
np.set_printoptions(precision=2)


def interface(s: socket.socket) -> None:
    n = int(input("Enter the value of n for n*n matrix: "))
    matrix = np.zeros((n,n))
    for i in range(n):
        row = input(f"Row {i+1}: ")
        elements = row.split()
        for j in range(n):
            matrix[i][j] = float(elements[j])

    send_matrix(s, matrix)
    upper = recv_matrix(s)
    lower = recv_matrix(s)
    product = (np.matmul(lower,upper))
    print("Lower Triangular\t\tUpper Triangular")

    for i in range(n):
        for j in range(n):
            print("%.2f" % lower[i][j], end="\t")
        print("",end="\t")
        for j in range(n):
            print("%.2f" % product[i][j], end="\t")
        print("")
    print(f"\nCheck output matrix and actual matrix are equal
: {(product == matrix).all()}\n")

    print("Actual matrix\t\t\tProduct matrix")
    for i in range(n):
```

```
        for j in range(n):
            print("%.2f" % matrix[i][j], end="\t")
        print("",end="\t")
        for j in range(n):
            print("%.2f" % product[i][j], end="\t")
        print("")
    input()
    s.close()


def main():
    client = Client(name='central-server')
    client.set_interface(interface)
    client.connect()


if __name__ == "__main__":
    main()
```

src/dns.py

Python

```
import socket
from threading import Lock

from constants.servers import DNS
from constants.network import MSG_LEN
from lib.network.server import Server
from lib.utils.logger import log

lock = Lock()
dns_map = {}


def client_handler(cli: socket.socket, ip: any) -> None:
```

```
    global dns_map
    with cli:
        req = cli.recv(MSG_LEN).decode()
        if '@' in req:
            # Update DNS record
            name, ip = req.split('@')
            with lock:
                dns_map[name] = ip
            log.debug(f'Updated Record for {name}: {ip}')
            log.debug(f'Final DNS State: {dns_map}')
        else:
            try:
                name = req
                ip = dns_map[name]
                log.debug(f'Resolving {name} : {ip}')
                cli.send(ip.encode())
            except:
                cli.send(b'DNS Record Not found')


def main():
    host = DNS.host
    port = DNS.port
    port = int(port)

    server = Server(host, port, server_name="DNS")
    server.set_client_handler(client_handler)
    server.start(register_dns=False)


if __name__ == "__main__":
    main()
```

src/server.py

Python

```python
import os
import socket
import threading
from typing import List

from constants.servers import CENTRAL_SERVER
from lib.network.server import Server
from lib.network.client import Client
from lib.network.matrix import recv_matrix, send_matrix


def client_handler(cli: socket.socket, ip: any) -> None:
    with cli:
        matrix = recv_matrix(cli)
        decomposed_matrices: dict = {}
        lock = threading.Lock()

        def l_interface(s: socket.socket):
            send_matrix(s, matrix)
            lower_matrix = recv_matrix(s)
            with lock:
                decomposed_matrices["lower"] = lower_matrix

        def u_interface(s: socket.socket):
            send_matrix(s, matrix)
            upper_matrix = recv_matrix(s)
            with lock:
                decomposed_matrices["upper"] = upper_matrix

        l_client.set_interface(l_interface)
        u_client.set_interface(u_interface)

        jobs:List[threading.Thread] = []

jobs.append(threading.Thread(target=l_client.run_interface))
```

```
jobs.append(threading.Thread(target=u_client.run_interface))

        for job in jobs:
            job.start()
        for job in jobs:
            job.join()

        l_client.run_interface()
        u_client.run_interface()
        send_matrix(cli, decomposed_matrices["upper"])
        send_matrix(cli, decomposed_matrices["lower"])


def main():
    host = os.getenv("HOST") or CENTRAL_SERVER.host
    port = os.getenv("PORT") or CENTRAL_SERVER.port
    port = int(port)

    global l_client
    global u_client

    server = Server(host, port, server_name="central-server")
    l_client = Client(name='l-decomposer')
    u_client = Client(name='u-decomposer')

    l_client.connect(run_interface=False)
    u_client.connect(run_interface=False)
    server.set_client_handler(client_handler)
    server.start()


if __name__ == "__main__":
    main()
```

src/l_worker.py

Python

```python
import os
import socket

from constants.servers import L_SERVER
from lib.network.server import Server
from lib.network.matrix import recv_matrix, send_matrix
from lib.math.matrix import lDecomposition
from lib.utils.logger import log


def client_handler(cli: socket.socket, ip: any) -> None:
    with cli:
        while True:
            matrix = recv_matrix(cli)
            l_matrix = lDecomposition(matrix)
            log.debug(f'Recieved from {ip} :\n {matrix}')
            log.debug(f'Decomposed :\n {l_matrix}')
            send_matrix(cli, l_matrix)


def main():
    host = os.getenv("HOST") or L_SERVER.host
    port = os.getenv("PORT") or L_SERVER.port
    port = int(port)

    server = Server(host, port, server_name="l-decomposer")
    server.set_client_handler(client_handler)
    server.start()


if __name__ == "__main__":
    main()
```

src/u_worker.py

Python

```python
import os
import socket

from constants.servers import U_SERVER
from lib.network.server import Server
from lib.network.matrix import recv_matrix, send_matrix
from lib.math.matrix import uDecomposition
from lib.utils.logger import log


def client_handler(cli: socket.socket, ip: any) -> None:
    with cli:
        while True:
            matrix = recv_matrix(cli)
            u_matrix = uDecomposition(matrix)
            log.debug(f'Recieved from {ip} :\n {matrix}')
            log.debug(f'Decomposed :\n {u_matrix}')
            send_matrix(cli, u_matrix)

def main():
    host = os.getenv("HOST") or U_SERVER.host
    port = os.getenv("PORT") or U_SERVER.port
    port = int(port)

    server = Server(host, port, server_name="u-decomposer")
    server.set_client_handler(client_handler)
    server.start()


if __name__ == "__main__":
    main()
```

Outputs

Client

```
Enter the value of n for n*n matrix: 3
Row 1: 1 2 3
Row 2: 4 5 6
Row 3: 7 8 9
Lower Triangular                Upper Triangular
1.00    0.00    0.00            1.00    2.00    3.00
4.00    1.00    0.00            4.00    5.00    6.00
7.00    2.00    1.00            7.00    8.00    9.00

Check output matrix and actual matrix are equal : True

Actual matrix                   Product matrix
1.00    2.00    3.00            1.00    2.00    3.00
4.00    5.00    6.00            4.00    5.00    6.00
7.00    8.00    9.00            7.00    8.00    9.00
```

Central Server

```
2023-04-10 01:10:14,542 - INFO - Listening on localhost:8003 (server.py:57)
2023-04-10 01:10:15,646 - INFO - [*] New Connection : ('127.0.0.1', 47588) (server.py:26)
```

L-Decomposer

```
2023-04-10 01:10:12,342 - INFO - Listening on localhost:8001 (server.py:57)
2023-04-10 01:10:14,542 - INFO - [*] New Connection : ('127.0.0.1', 34590) (server.py:26)
2023-04-10 01:10:26,083 - DEBUG - Recieved from ('127.0.0.1', 34590) :
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]] (l_worker.py:16)
2023-04-10 01:10:26,083 - DEBUG - Decomposed :
 [[1. 0. 0.]
 [4. 1. 0.]
 [7. 2. 1.]] (l_worker.py:17)
2023-04-10 01:10:26,084 - DEBUG - Recieved from ('127.0.0.1', 34590) :
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]] (l_worker.py:16)
2023-04-10 01:10:26,084 - DEBUG - Decomposed :
 [[1. 0. 0.]
 [4. 1. 0.]
 [7. 2. 1.]] (l_worker.py:17)
```

U-Decomposer

```
2023-04-10 01:10:13,437 - INFO - Listening on localhost:8002 (server.py:57)
2023-04-10 01:10:14,542 - INFO - [*] New Connection : ('127.0.0.1', 58752) (server.py:26)
2023-04-10 01:10:26,083 - DEBUG - Recieved from ('127.0.0.1', 58752) :
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]] (u_worker.py:16)
2023-04-10 01:10:26,084 - DEBUG - Decomposed :
 [[ 1.  2.  3.]
 [ 0. -3. -6.]
 [ 0.  0.  0.]] (u_worker.py:17)
2023-04-10 01:10:26,084 - DEBUG - Recieved from ('127.0.0.1', 58752) :
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]] (u_worker.py:16)
2023-04-10 01:10:26,085 - DEBUG - Decomposed :
 [[ 1.  2.  3.]
 [ 0. -3. -6.]
 [ 0.  0.  0.]] (u_worker.py:17)
```

DNS

```
2023-04-10 02:08:24,573 - INFO - Listening on localhost:4000 (server.py:57)
2023-04-10 02:08:25,728 - INFO - [*] New Connection : ('127.0.0.1', 36634) (server.py:26)
2023-04-10 02:08:25,729 - DEBUG - Updated Record for l-decomposer: localhost:8001 (dns.py:22)
2023-04-10 02:08:25,729 - DEBUG - Final DNS State: {'l-decomposer': 'localhost:8001'} (dns.py:2
3)
2023-04-10 02:08:26,805 - INFO - [*] New Connection : ('127.0.0.1', 36640) (server.py:26)
2023-04-10 02:08:26,806 - DEBUG - Updated Record for u-decomposer: localhost:8002 (dns.py:22)
2023-04-10 02:08:26,806 - DEBUG - Final DNS State: {'l-decomposer': 'localhost:8001', 'u-decomp
oser': 'localhost:8002'} (dns.py:23)
2023-04-10 02:08:27,899 - INFO - [*] New Connection : ('127.0.0.1', 36650) (server.py:26)
2023-04-10 02:08:27,899 - DEBUG - Resolving l-decomposer : localhost:8001 (dns.py:28)
2023-04-10 02:08:27,899 - INFO - [*] New Connection : ('127.0.0.1', 36654) (server.py:26)
2023-04-10 02:08:27,900 - DEBUG - Resolving u-decomposer : localhost:8002 (dns.py:28)
2023-04-10 02:08:27,900 - INFO - [*] New Connection : ('127.0.0.1', 36656) (server.py:26)
2023-04-10 02:08:27,900 - DEBUG - Updated Record for central-server: localhost:8003 (dns.py:22)
2023-04-10 02:08:27,901 - DEBUG - Final DNS State: {'l-decomposer': 'localhost:8001', 'u-decomp
oser': 'localhost:8002', 'central-server': 'localhost:8003'} (dns.py:23)
2023-04-10 02:08:28,981 - INFO - [*] New Connection : ('127.0.0.1', 36668) (server.py:26)
2023-04-10 02:08:28,981 - DEBUG - Resolving central-server : localhost:8003 (dns.py:28)
```

Suggestions

1. The application can be further optimized by implementing load balancing techniques to distribute the workload evenly across the two servers, especially for larger matrices.
2. A more secure communication protocol such as SSL/TLS can be implemented to ensure the confidentiality and integrity of the data transmitted between the client and servers.
3. The client interface can be improved by adding more features such as the ability to upload and save matrices in different formats, and the display of the computation time for each server.

Conclusion

In this report, we have successfully implemented a Matrix Decomposer application using Doolittle's algorithm and distributed computing concepts. The application consists of a client, two servers, and a central server that coordinates the computation and returns the matrices to the client. The correctness of the computed matrices is verified by checking the formula A=LU. The application can be further optimized and secured by implementing load balancing and secure communication protocols. Overall, the Matrix Decomposer application is a useful tool for performing matrix decomposition and can be extended to handle larger matrices and more complex computations.