CS 553

CLOUD COMPUTING

PROGRAMMING ASSIGNMENT-1

SUBMITTED BY :

SACHIN KRISHNA MURTHY

CWID : A20354077

DESIGN DOCUMENT

CPU Benchmark

Objective : To measure the processor speed in terms of floating point operations per second (GFLOPS) and integer point operations per second (IOPS) at varying levels of concurrency (1 thread, 2 threads and 4 threads).

Working Procedure :

- This benchmark consists of two files :
  - CPUFlops.java : For calculating floating point operations.
  - CPUIops.java : For calculating integer point operations.
- These files consists of two methods : main() and runthreadsbycount().
- In the main method the varying level of concurrency i,e the number of threads (1,2 or 4) is taken as input from the user.
- Main method calls method runthreadsbycount where the number of threads to be run is taken as the parameter.

  runthreadsbycount(1)

- In runthreadsbycount method the threads are run to perform floating point and integer point operations in their respective files.
- Within the thread, floating point operation is done 1 giga times i,e 10^9 .
- The start time and end time for this operation is noted.

```
double startTime = System.currentTimeMillis();

for (int j = 0; j < Integer.MAX_VALUE; j++) {
   double sum = a + b;
}

double endTime = System.currentTimeMillis();
```

- System. *currentTimeMillis* () is used for obtaining the time in millisecond and is converted into seconds by dividing it by 10^3.
- For each thread operation, gflops is calculated by using the below formula:
  - GFLOPS= ((No. of operations * loop) / (total time)) / (10^9)

- Calculated GFLOPS are read into an array list.

```
static List<Double> gflopList =
                        new ArrayList<>();

gflopList.add(GFLOPS);
```

- After the completion of all threads, GFLOPS from array list are taken and an average GFLOP is calculated for all threads.

```
double sum = 0.00;
for (Double gf : gflopList) {
  sum = sum + gf;
}

avgGflops = (sum / count);
```

- The similar process is repeated to determine the processor speed for Integer point operations by performing the integer point operation 1 giga i,e 10^9 times.
- The GIOPS is calculated is calculated by using the formula :
    - GILOPS= ((No. of operations * loop) / (total time)) / (10^9)

DISK BENCHMARK

Objective : To measure the disk speed for read and write operations for Sequential as well as Random access with varying block sizes of 1B, 1KB and 1MB and by varying the levels of concurrency with 1 and 2 threads.

Working Procedure :

- This benchmark consists of three files :
    - DiskByte.java
    - DiskKiloByte.java
    - DiskMegaByte.java
- In the main method, the name of the files for which the data to be written is taken as input for both sequential and random access.
- The class consists of another method “runthreadsbycount” , used to perform disk operations
- The number of threads, filename for sequential access and filename for random access is passed as parameters for runthreadsbycount method.

  runthreadsbycount(1,sequencefilename,randomfilename)
- In runthreadsbycount method, a thread to perform read and write operations on the disk is started.
- Within the thread disk operations are performed for the specified block size (1B, 1KB, 1MB).

```
for(int i=0;i<count;i++)
        {
   new Thread(new Runnable(){

         public void run(){
```

```
                try {

        //WRITE WITH SEQUENCIAL ACCESS
        for(int i=0;i<BLOCKSIZE;i++)
        {
           bw.write("s");
        }

        //READ WITH SEQUENCIAL ACCESS
        for(int i=0;i< BLOCKSIZE;i++)
        {
                ch=(char)r;
        }

        //WRITE WITH RANDOM ACCESS
        for(int i=0;i<BYT BLOCKSIZE E;i++)
        {  Random rand = new Random();
           int pos = rand.nextInt(BYTE) + 0;
           FileChannel fc = raf.getChannel();
           fc.position(pos);
           raf.write(("k").getBytes());
        }


        //READ WITH RANDOM ACCESS
        for(int i=0;i< BLOCKSIZE;i++)
        {
           Random rand = new Random();
           int pos = rand.nextInt(BYTE) + 0;
           FileChannel fc = raf.getChannel();
           fc.position(pos);
           ch = (char)raf.read();
        }
     }).start();
  }
```

- Separate time is noted for both read and write operations for both sequential and random access.

```
   double time2=(endtime2-starttime2);
   double totaltime2=(time2/1000000000);

   double latency2= (((totaltime2)/(BYTE))*1000);
   double throughput2=((BYTE)/(1024*1024*totaltime2));
```

- For Sequential access the character is written and read sequentially from the file depending on the block size.
- For Random access the character is written and read from random positions in the file depending on the block size.
- System.nanoTime() is used for obtaining the time in nanosec and is converted into seconds by dividing it by 10^9.

- Further using the total time i,e the difference in end time and start time, latency in ms and throughput in MB/Sec is also calculated.

Network Benchmark

Objective : To measure the disk speed for read and write operations for Sequential as well as Random access with varying block sizes of 1B, 1KB and 1MB and by varying the levels of concurrency with 1 and 2 threads.

Working Procedure :

- This benchmark consists of three files :
  - DiskByte.java
  - DiskKiloByte.java
  - DiskMegaByte.java

- For UDP connection , there are two classes one each for server and client.
- For server, main method calles `runServerMethod`,`to perform server setup using threads and for client,it calls clientService method to perform client operations.`
- For client, inside the tread a new datagram is created and sent to server using port number and IP. This thread also receives acknowledgement message sent by server.

```
for (int i = 0; i < BLOCKSIZE; i++) {
   DatagramSocket socket = new DatagramSocket();
   InetAddress host = InetAddress.getByName("IP");

   DatagramPacket datagram = new DatagramPacket(
      message, message.length, host, portNumber);
   socket.send(datagram);


   DatagramPacket reply = new DatagramPacket(buffer,
      buffer.length);
   socket.receive(reply);

 }
```

- For server, inside the thread datagram with message are received and acknowleddge is sent in datagram format.

```
for (int i = 0; i < BYTE; i++) {

   socket.receive(datagram);
   String message = new String(data, 0, datagram.getLength());
```

```
        System.out.println(datagram.getAddress().getHostAddress() + " : "
            + datagram.getPort() + " - " + s);

        s = "OK : " + message;
        DatagramPacket reply = new DatagramPacket(s.getBytes(),
            s.getBytes().length, datagram.getAddress(),
            datagram.getPort());
        socket.send(reply);
    }
```

- Start and end time in millliseconds are noted for different blocksize of data and latency and throughput are caluculated.

```
        double totaltime = (endtime - starttime)/1000;
        double rtt = totaltime/(count * BYTE);

        double latency = rtt*1000;
        double throughputbps=1/rtt;

        double throughput = ((8*throughputbps)/(1024*1024));
```

- To perform the operaton, first server is run and once server is ready for connection, client is run which creates datagrams for message and sends to server.
- After certain required amount of data transaction, client closes
- Similar to UDP, TCP also has client and server. Unlike inn UDP, in TCP a connection setup is done using new `ServerSocke`, port and IP
- At client side, after connection setup, message is sent and also acknowledgement is received

```
    Socket socket = new Socket("localhost", port++);

        for (int i = 0; i < BYTE; i++) {

            DataOutputStream message = new DataOutputStream(
                  socket.getOutputStream());
            message.writeBytes(data + '\n');

            BufferedReader reply = new BufferedReader(new
    InputStreamReader(
                  socket.getInputStream()));
            String replyMsg = reply.readLine();
        }
        socket.close();
```

- At server side, after connection setup, message is received and acknowledgement is sent.

```
    for (int j = 0; j < BYTE; j++) {

        BufferedReader message = new BufferedReader(new
                          InputStreamReader(connection.getInputStream()));
        String receivedData = message.readLine();

        DataOutputStream reply = new DataOutputStream(
                    connection.getOutputStream());
        String replyData = "OK"+ '\n';
```

```
        reply.writeBytes(replyData);
    }
```

- Similar to UDP, in TCP also to perform the operaton, first server is run and once server is ready for connection, client is run which creates connection setup. Once connection is compete, message and acknowledgement transactions happen.
- After required number of transactions, connection is closed
- Latency and throughput are calculated similar to UDP