Tugas 3
Jaringan Komputer
Kelompok A6

Dokumentasi Produk

Master-Slave Socket Programming

Disusun oleh:

- Irfan Maulana Nasution (1806205716)
- Muhammad Yoga Mahendra (1806205256)
- Nasywa Nur Fathiyah (1806205546)

Daftar Isi

Software Description

Software ini merupakan sekumpulan *interconnected program* yang memanfaatkan teknologi *cloud computing* dan *socket programming* untuk menciptakan sebuah jaringan master-workers yang digunakan untuk melakukan penyortiran/pengurutan dari sebuah array. Cara kerjanya dapat dijelaskan secara singkat sebagai berikut:

1. Master dan Workers masing-masing dijalankan terlebih dahulu(Master di local/EC2, Workers dijalankan di instance EC2 lain), lalu dihubungkan via *socket*, dengan argumen ipaddress:port.
2. Master dapat mengirimkan perintah sorting dengan format: sort xxxyyy #array, dengan keterangan:
   - sort (perintah untuk melakukan sort)
   - xxx (ID dari client/master yang mengirimkan job, 000 untuk shell master)
   - yyy (ID dari job tersebut)
   - array (array yang mau disortir/diurutkan. Tanda # membatasi antara perintah dan arraynya).
3. Setelah master menerima perintah sort, master akan mengirimkannya ke worker via *socket* yang sudah terbuka.
4. Worker menerima perintah sort yang dikirim master, melakukan parsing untuk membaca ID perintah dan array, lalu mensortirnya.
5. Setelah mensortir, array akan dikembalikan ke master via *socket* dengan format perintah yang sama (sort xxxyyy #array), dimana arraynya adalah hasil sortiran.
6. Master menerima hasil sortiran dari worker, lalu mencetaknya ke terminal/shell.
7. Ketika semua job sudah selesai, user memerintahkan master untuk berhenti(dengan perintah “stop”). Master akan menyuruh semua worker yang tersambung dengannya untuk berhenti juga, lalu Master akan di *terminate*. Workers yang socketnya tadi terputus akan membuka koneksi lagi, menunggu master lain untuk membuka koneksi. Worker dapat dimatikan dengan menggunakan interrupt(seperti CTRL + C)

Link Repositori: https://gitlab.com/m.yoga.mahendra/jarkomtk1

Requirements

Fitur Dasar		
#	Deskripsi	Notes
1	Adanya satu master node yang dapat menerima job dari user. Laptop/PC mahasiswa difungsikan sebagai Master node dan job dapat langsung di submit dari sini.	Diimplementasi
2	Adanya (minimal) satu worker node yang dapat mengeksekusi job dari master. Virtual machine (EC2) difungsikan sebagai worker node(s).	Diimplementasi
3	Adanya komunikasi antara master dan worker node(s). Master dapat mengirimkan perintah dan job kepada worker dan worker dapat mengirimkan balik hasil eksekusi kepada master.	Diimplementasi
Fitur Elective		
4	Monitor availability dari setiap worker node(s) (contoh: active, dead, running, busy, etc).	X (belum selesai, indikator sudah disiapkan. Fungsi belum)
5	Memonitor status jobs yang sedang dieksekusi di worker node(s) (contoh: failed, finished, running etc)	X (belum selesai, indikator sudah disiapkan. Fungsi belum)
6	Memonitor di worker node(s) mana suatu job dieksekusi.	Belum selesai diimplementasi
7	Membatalkan job yang sedang dieksekusi di suatu worker node(s).	X
8	Adanya variasi jenis jobs berdasarkan lamanya waktu eksekusi (contoh: sangat singkat (milis), singkat (s), lama (minutes)).	X
9	Adanya antrian jobs di master node berdasarkan algoritma FCFS.	Diimplementasi (Round Robin +

		Queue)
10	Adanya antrian jobs di worker node(s) berdasarkan algoritma FCFS.	Diimplementasi (Queue)
11	Implementasi algoritma penjadwalan selain FCFS di antrian master node.	X
12	Implementasi algoritma penjadwalan selain FCFS di antrian worker node.	X
Fitur Advanced		
13	Master node berfungsi sebagai portal dan di set up di EC2. Ada program client yang bisa di install di local computer yang dapat men-submit jobs ke portal.	X
14	Implementasi load balancer--tanpa menggunakan teknologi yang sudah ada--yang dapat membagi beban setiap worker node(s).	Implemented (Round Robin)
15	Implementasi fitur auto-upscaling--tanpa menggunakan teknologi yang sudah ada--sehingga sistem dapat menambah jumlah worker node(s)/meng-create VM baru secara otomatis ketika sangat sibuk.	Implemented
16	Implementasi fitur auto-downscaling--tanpa menggunakan teknologi yang sudah ada--sehingga sistem dapat mengurangi jumlah worker node(s)/men-terminate VM yang ada secara otomatis ketika sangat sepi.	Implemented
17	Implementasi fitur security sehingga komunikasi antara master dan worker node(s) perlu di autentikasi sebelum mengirimkan jobs.	X

Software Architecture

Bahasa Pemrograman

Java

Library yang Digunakan

Java.net

Package java.net memfasilitasi *programmer* untuk mengimplementasi aplikasi *networking*.

Class yang dipakai:

- Socket : kelas yang implementasi socket client.
- ServerSocket : kelas yang implementasi socket server.

Java.io

Package java.io memfasilitasi penggunaan standard input & output untuk keperluan IO user.

Class yang dipakai:

- DataInputStream : kelas untuk implementasi penerimaan data via stream dari worker yang dikirim via socket.
- DataOutputStream : kelas untuk implementasi pengiriman data ke worker dari stream via socket.
- BufferedReader : kelas untuk menerima input user dari stream terminal.

Java.util

Package java.io memfasilitasi program dengan berbagai utilitas Java.

Class yang dipakai:

- Arrays, List, ArrayList, Queue, LinkedList, Map, TreeMap :
  kelas yang digunakan untuk menyimpan berbagai variabel dan objek yang digunakan program selama berjalan serta digunakan untuk implementasi FCFS dan Load Balancer jobs.
- Random : kelas yang digunakan untuk implementasi perintah membuat data array acak dengan ukuran sesuai permintaan user.
- Concurrent : kelas yang mengimplementasi eksekusi *multithreading.*

Java.lang

Package java.lang memfasilitasi *programmer* dengan berbagai utilitas terkait tipe data dan multithreading.

Class yang dipakai:

- String, Integer : kelas yang digunakan untuk menyimpan dan mengoperasikan objek data.
- Thread : kelas yang mengimplementasi konsep *multithreading.*

Source Code Documentation

Implementasi fitur #1

Master.java

```
class Master{
  static Random rand = new Random();
  static ArrayList<Socket> socketList;
  static ArrayList<DataInputStream> dinList;
  static ArrayList<DataOutputStream> doutList;
  static ArrayList<ArrayList<Integer>> workerJobInfo;
  static ArrayList<String> workerAddressInfo;
  static Queue<String> jobsQueue;
  static Queue<String> finishedJobsQueue;
  static BufferedReader shellReader;
  static boolean stopFlag = false;
  static boolean readingFlag = true;
  static List<MasterHelper> masterHelperList;
  static ExecutorService pool;

  public static void main(String args[]) throws Exception {//argumen 0 = port server socket
    prepareMaster();
    connectWorker(workerAddressInfo.get(0));
    println(
      "---------------List Of Command---------------\n"+
      // omitted for brevity.
      "---------------List Of Command---------------"
    );
    while (stopFlag == false) {
      print("Input : ");
      readShellCommand();
    }
    disconnectWorker(0);
    println("Master is successfully stopped.");
  }
  // Rest of code is at further implementation requirements.
}
```

Pada implementasi kami, master diimplementasi pada PC lokal anggota kelompok dan worker diimplemetasi pada AWS EC2. Implementasi socket untuk menghubungkan master dengan worker menggunakan IP address dari mesin EC2 pada AWS kami. *Snippet* kode diatas hanya berisi method *main* dari master.java, semua method lain akan dijabarkan sesuai dengan *requirements* yang dipenuhi oleh method tersebut.

Implementasi fitur #2

Worker.java

```
class Worker{
  static ServerSocket ss;
  static Socket s;
  static DataInputStream din;
  static DataOutputStream dout;
  static Queue<String> jobsQueue;
  static Queue<String> finishedJobsQueue;
  static String runningFlag = "idle"; //active + running, active + idle, dead
  static List<WorkerHelper> workerHelperList;
  static ExecutorService pool;

  //run by typing "java Worker IP port". //args = port
  public static void main(String args[])throws Exception{
    prepareWorker(args[0]);

    //while true ini masih sementara. nantinya pake thread
    while(true){
      try{
        print("Waiting for command");
        receiveCommand();
        print("Command accepted");
        print("");
      }
      catch (SocketException e) {
        print("Socket Error");
        endConnection();
        startConnection();
      }
    }
  }
  // Rest of code is at further implementation requirements.
}
```

Pada implementasi worker.java, worker menerima input pekerjaan yang dikirimkan oleh Master via socket, lalu Worker menerima dan melaksanakan job yang diassign kepadanya. Untuk menerima pekerjaan yang diberikan oleh master, program Worker pertama mengidentifikasi terlebih dahulu command apa yang diminta dengan method receiveCommand(). Setelah menentukan command apa yang harus dilaksanakan, jika command yang diminta adalah sorting, maka program akan memasukkan command tersebut ke antrean jobs. Setelah itu, program akan melakukan sorting. Hasil sorting tersebut akan dimasukkan ke sebuah queue yang nantinya akan direturn ke Master melalui method returnJob(). Setelah semua jobs sudah dilakukan(saat master menyuruh worker berhenti) Worker akan menutup koneksi dengan Master lalu mencoba membuka koneksi kembali(masuk mode standby).

Implementasi fitur #3

Master.java

```
static void distributeJob() throws Exception{//load balancer
   //job distribution start from worker with lowest job
   int workerIdx = getWorkerWithLowestQueue();
   int tmpLimit = jobsQueue.size();
   for(int jobsIdx = 0; jobsIdx<tmpLimit; jobsIdx++) {
     if(workerIdx == workerJobInfo.size()) { workerIdx = 0; }
     sendJob(workerIdx);
     workerIdx++;
   }
 }
static int getWorkerWithLowestQueue() {
   int lowestQIdx = 0;
   for(int i=1; i<workerJobInfo.size(); i++) {
     if(workerJobInfo.get(i).size()<workerJobInfo.get(lowestQIdx).size()) {
       lowestQIdx = i;
     }
     if(workerJobInfo.get(lowestQIdx).size() == 0) {
       break;
     }
   }
   return lowestQIdx;
 }

static void sendJob(int workerIdx) throws Exception   {
   String job = jobsQueue.poll();

   //tambahkan id job ini ke list job worker itu
   String jobId = job.substring(5,10);
   workerJobInfo.get(workerIdx).add(Integer.parseInt(jobId));

   DataOutputStream tmpDOut = doutList.get(workerIdx);
   tmpDOut.writeUTF(job);
   tmpDOut.flush();
 }
```

Worker.java

```
static String receiveCommand() throws Exception{
   String input = din.readUTF();

   String command;
   if(input.length()>12) {
     command = input.split("#")[0];  //ini mungkin agak lama lebih cepet kalo di substring
dulu karna dia bakal iterasi semua digit sampe akir
   }
   else {
     command = input;
   }
```

```
  switch(command.split(" ")[0]) {
    case "sort" :
      jobsQueue.add(input);
      break;
    case "stop" :
      //doStop
      break;
  }
  return input;
}
```

Untuk mendistribusikan jobs kepada workers, program memanggil method getWorkerWithLowestQueue(), method ini mengembalikan Worker yang pada saat pemanggilan, memiliki antrean *jobs* paling sedikit. Setelah mendapatkan Worker mana yang memiliki antrea jobs paling kecil, Master akan mengirimkan *jobs* dengan *method* sendJob(). Method sendJob() akan mengirimkan job ke worker melalui method DataOutputStream.writeUTF().

Implementasi fitur #6

Master.java

```
  String jobId = job.substring(5,10);
  workerJobInfo.get(workerIdx).add(Integer.parseInt(jobId));
```

Pada method sendJob(), master memberikan ID pada setiap job.

Implementasi fitur #9

Master.java

```
 static Queue<String> jobsQueue;

// method readShellCommand()
case "sort" : //format : sort xxxyyy # array
       jobsQueue.add(input);

// method distributeJob()
for(int jobsIdx = 0; jobsIdx<tmpLimit; jobsIdx++) {
    if(workerIdx == workerJobInfo.size()) { workerIdx = 0; }
    SendJob(workerIdx);
    workerIdx++;
 }

// method sendJob()
String job = jobsQueue.poll();
```

Master.java mengimplementasi queue terhadap jobs yang diinput oleh user. Queue ini ditambahkan saat program membaca input dari user dan di-*poll* saat melakukan pengiriman job kepada Worker node. Setiap job akan dikirimkan kepada setiap worker secara merata, dimulai dari job yang pertama masuk.

Implementasi fitur #10

Worker.java

```
static Queue<String> jobsQueue;

//pada method receiveCommand()
switch(command.split(" ")[0]) {
     case "sort" :
       jobsQueue.add(input);
       break;
     case "stop" :
       //doStop
       break;
   }
//pada method doSort()
if(jobsQueue.peek() != null) {
     String input = jobsQueue.poll();
```

Worker.java mengimplementasi queue terhadap jobs yang diberikan kepadanya. Pada method receiveCommand(), Worker memasukkan jobs dari master ke jobsQueue. Pada method doSort(), Worker melakukan *poll* terhadap jobsQueue untuk mengeksekusi jobs yang pertama sampai pada Worker terlebih dahulu.

Implementasi fitur #14

Master.java

```
static void distributeJob() throws Exception{
   //job distribution start from worker with lowest job
   int workerIdx = getWorkerWithLowestQueue();
   int tmpLimit = jobsQueue.size();
   for(int jobsIdx = 0; jobsIdx<tmpLimit; jobsIdx++) {
     if(workerIdx == workerJobInfo.size()) { workerIdx = 0; }
     sendJob(workerIdx);
     workerIdx++;
   }
 }
```

Pada method distributeJob() di Master.java, program memprioritaskan Worker yang memiliki queue jobs paling sedikit(atau yang tidak memiliki job apapun). Algoritma yang digunakan adalah Round Robin, dimana setiap job didistribusikan secara merata kesemua worker. Metode ini menjamin setiap worker akan bekerja jika jumlah job yang masuk >= jumlah worker yang tersambung ke master, dan setiap job kan langsung dikerjakan jika jumlah job yang masuk <= jumlah worker yang sedang standby.

Implementasi fitur #15

```
 static void autoScaling() throws Exception {
          int currentJob = 0;
          int activeWorkerNum = socketList.size();
          if(activeWorkerNum != 0 ) {
          for(int i = 0 ; i<socketList.size() ; i++) {
          currentJob = currentJob + workerJobInfo.get(i).size();
          }

          if(currentJob/activeWorkerNum > 10 &&
activeWorkerNum<=workerAddressInfo.size()) {
          try{
          println("System is UPSCALED");
          String tmpAddress = workerAddressInfo.get(activeWorkerNum);
          connectWorker(tmpAddress);
          }
          catch(Exception e) {
          println("warning : System failed to upscale");
          }
          }
          else if(currentJob/activeWorkerNum <= 2 && activeWorkerNum>1) {
          println("System is DOWNSCALED");
          disconnectWorker(activeWorkerNum-1);
          }
          }
  }

// di class masterHelper
          else if (handlerTask.equals("autoScaling")) {
          while(true) {
          Master.autoScaling();
```

```
        Thread.sleep(1000);
        }
    }
```

Method autoScaling() akan melakukan scaling otomatis sesuai dengan jumlah worker dan jobs yang masuk. Jika jobs yang masuk sangat banyak(tanpa peduli bebannya), dan worker sedikit, maka master akan meminta worker baru.

Implementasi fitur #16

```
// potongan kode sama dengan #15
```

Method autoScaling() akan melakukan scaling otomatis sesuai dengan jumlah worker dan jobs yang masuk. Jika jobs yang masuk sangat sedikit(tanpa peduli bebannya), dan worker cukup banyak, maka master akan memutuskan sambungan dan mematikan worker yang tidak bekerja.

Quality Assurance Documentation

Masing-masing worker menggunakan IP 34.224.75.203 dan port 3333 dan 3334(satu instance EC2 menjalankan 2 worker). Berikut hasil tangkapan layar menjalankan program pada EC2 AWS sebagai worker dan PC lokal sebagai Master:

1. Menginisiasi Master pada PC lokal ke EC2 port 3333

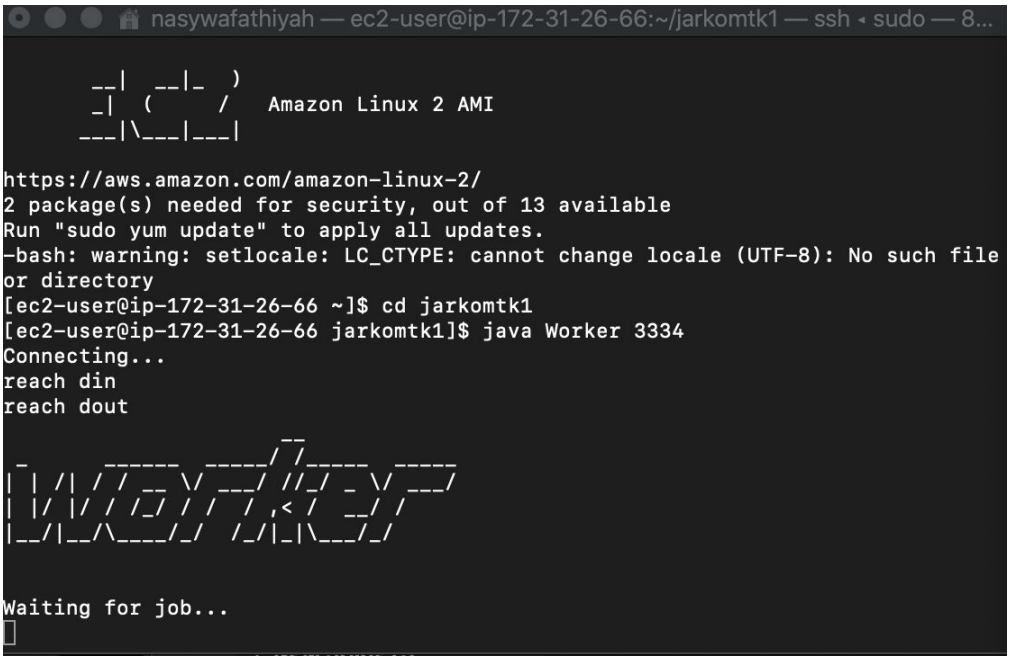
2. Respon dari EC2 port 3333

```
        at Worker.main(Worker.java:35)
[ec2-user@ip-172-31-26-66 jarkomtk1]$ java Worker 3333
Connecting...
reach din
reach dout
                      __
 _      ______  _____/ /_____  _____
| | /| / / __ \/ ___/ //_/ _ \/ ___/
| |/ |/ / /_/ / /  / ,< /  __/ /
|__/|__/\____/_/  /_/|_|\___/_/


Waiting for job...
```

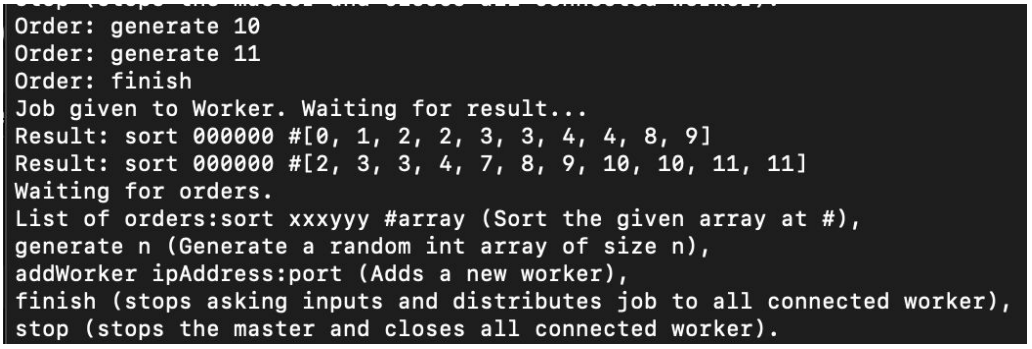
3. Inisiasi Master ke EC2 port 3334

```
jarkomtk1 — java Master — 80×20
Tutorial-VPC-1806205546-A.pdf
alice.txt
jarkom.pem
jarkomtk1
[nasywafathiyah@Nasywas-MacBook-Air jarkom % cd jarkomtk1
[nasywafathiyah@Nasywas-MacBook-Air jarkomtk1 % ls
Master.class    Master.java     Worker.class    Worker.java
[nasywafathiyah@Nasywas-MacBook-Air jarkomtk1 % java Master
                        __
    ____ ___  ____ ______/ /____  _____
   / __ `__ \/ __ `/ ___/ __/ _ \/ ___/
  / / / / / / /_/ (__  ) /_/  __/ /
 /_/ /_/ /_/\__,_/____/\__/\___/_/


Waiting for worker
Worker address: 34.224.75.203:3333
Worker address: 34.224.75.203:3334
Worker address: 34.224.75.203:3334
Worker address:
```

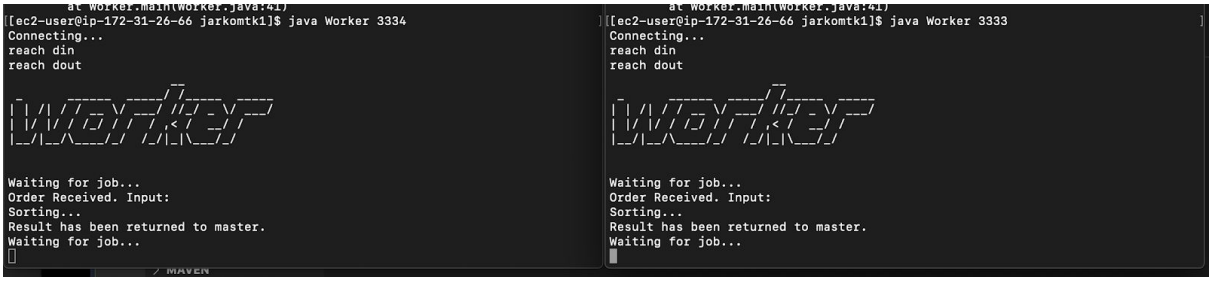
4. Respon dari EC2 port 3334



5. Input commands pada Master (pada bagian “Order:”)



6. Respon Workers



Note: reach din menandakan program sudah sampai membaca Data Input Stream, reach dout menandakan program sudah sampai membaca Data Output Stream.