##### RELIABLE TASK SCHEDULING IN HETEROGENEOUS DISTRIBUTED SYSTEMS

**A PROJECT REPORT**

***Submitted by***

|  |  |
| --- | --- |
| **BHIRUNDHA . M** | **(Reg.No. 201306006)** |
| **RAJAPRIYA . P** | **(Reg.No. 201306035)** |
| **SRI SRUTHI . S** | **(Reg.No. 201306054)** |

***in partial fulfillment for the award of the degree***

***of***

##### BACHELOR OF TECHNOLOGY

in

**INFORMATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**

**(An Autonomous Institution affiliated to Anna University Chennnai)**

**April 2017**

**BONAFIDE CERTIFICATE**

##### Certified that this project report titled RELIABLE TASK SCHEDULING IN HETEROGENEOUS DISTRIBUTED SYSTEMS is the bonafide work of M.BHIRUNDHA, P.RAJAPRIYA and S.SRI SRUTHI , who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Mrs.S.Saroja ,M.E., Dr.T.Revathi, M.E., Ph.D.,

**Internal guide Head of the Department**

**Assistant Professor (Senior Grade), Senior Professor,**

**Department of Information Technology, Department of Information Technology,**

**Mepco Schlenk Engineering College, Mepco Schlenk Engineering College,**

**Sivakasi. Sivakasi.**

Submitted for viva-voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI (AUTONOMOUS)** on **……………..............**

**Internal Examiner External Examiner**

**ACKNOWLEDGEMENT**

First and foremost, we thank the almighty for showering grace and blessings on us for making this project a great success.

We extend our sincere gratitude to our college management and our honourable Principal **Dr. S ARIVAZHAGAN, M.E., Ph.D.,** for providing sufficient working environment such as systems and library facilities. We also thank him very much for providing us adequate lab facilities, which enable us to complete our project.

We would like to extend our heartfelt gratitude to **Dr. T. REVATHI, M.E, Ph.d,** Professor and Head, Department of Information Technology, Mepco Schlenk Engineering college for her valuable guidance and timely encouragements. She put all her valuable experience and expertise in directing, suggesting and supporting us throughout the project to bring out of the best.

We would like to extend our gratitude to **Mrs. S. SAROJA, M.E** Assistant Professor (Senior Grade), Department of Information Technology, Mepco Schlenk Engineering College for being our Project Coordinator and directing us through our project.

We extend our thanks to all staff members and lab technicians for their moral support in completing our project.

ABSTRACT

**ABSTRACT**

Task Scheduling is one of the major challenges in efficiently using parallel and distributed systems. Eventhough those systems can be easily established, quality of service including reliability, performance and throughput is of major concern. In this paper we have introduced a set of algorithms which schedules the task by considering fault recovery in heterogeneous distributed systems to increase the performance and reliability. Here we have two phases in the first phase is to obtain the favorable number of allotments for each tasks and second the second phase has the scheduling method. And to reduce the expected execution time we have replicated the tasks based upon two startegies. In the first strategy the tasks are replicated to its allotment, its replication is never stopped and in the second strategy the replication of the tasks are stopped once the task is completed. Since recovery of failed machines increases the processing time, this scheduling helps to reduces the processing time by replicating the tasks instead of recovering the machines.

LIST OF TABLES

**LIST OF TABLES**

|  |  |
| --- | --- |
| **Notation** | **Description** |
| *P , L* | A set of processors and its links. |
| *pi ,li,j* | A processor and a communication link between pi and pj. |
| *w(pi)* | Weight of *pi* (Number of operations per unit time). |
| *w(li,j)* | Weight of *li,j* (Data transfer rate). |
| *V , E* | A set of tasks and precedence constraints among the tasks. |
| *w(vi)* | Weight of task *vi* (Computational Cost). |
| *w(ei,j)* | Weight of edge *ei,j* (Communication Cost). |
| *Si* | Set of processors allocated to vi. |
| *tp(vi,px)* | Processing time of vi on processor px. |
| *tc(ei,j,lk,x)* | Communication time of ei,j on link lk,x. |
| *pred(vi)* | Set of direct predecessors of task vi. |
| *succ(vi)* | Set of direct successors of task vi. |
| *λi* | Failure rate of processor pi |
| *λli,j* | Failure rate of link li,j |
| *γi* | Probability that a failure on processor pi is recoverable. |
| *R[Evi,px,pk,px]* | Communication failure probability. |
| ***Notation*** | **Description** |
| *R[Evi,px,pk,px]* | Communication reliability probability. |
| *F[Evi,px]* | Failure probability of task vi on processor px |
| *R[Evi,px]* | Reliability probability of task vi on processor px |
| *[Evi,px]* | Expected overhead of task vi on processor px |
| *p[vi]* | Mean processing time of task vi |
| *c[ei,j]* | Mean communication time of edge ei,j |
| *[Evi]* | Mean reliability of task vi |
| *[Eei,j]* | Mean communication reliability probability between vi and vj |
| *[Evi]* | Mean overhead of task vi |
| *Fi* | Mean failed probability of task vi |
| *Tp(vi, αi)* | Expected processing time of task vithat executes on αiprocessors. |
| *W(vi,αi)* | Expected work of task vithat executes on αiprocessors |
| *AL(i)* | Lower bound on the number of allotted processors for task vi |
| *β* | Weight of reliability overhead |
| *EST(vi,pk)* | Earliest start time of task vi on a processor pk |
| *EFT(vi,pk,Si)* | Earliest finish time of task vi on processor pk |
| *EFT(ei,j,Si,px)* | Earliest communication finish time of edge ei,j |

**LIST OF FIGURES**

**LIST OF FIGURES**

|  |  |
| --- | --- |
| **Fig No** | **Fig Name** |
|  | Architecture of the System |
|  | WorkFlow of the project |

CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE OF CONTENTS** | | | |
| **CONTENTS** | |  | **PAGE NO** |
| **Bonafide certificate** | |  | ii |
| **Acknowledgement** | |  | Iii |
| **Abstract** | |  | iv |
| **List of tables** | |  | vi |
| **List of figures** | |  | ix |
| **CHAPTER 1** |  | **INTRODUCTION** |  |
|  |  |  |  |
|  | 1.1 | PROBLEM DESCRIPTION | 2 |
|  | 1.2 | PROJECT OBJECTIVE | 2 |
| **CHAPTER 2** |  | **LITERATURE REVIEW** |  |
|  |  |  |  |
|  | 2.1 |  | 4 |
|  | 2.2 |  | 4 |
|  | 2.3 |  | 4 |
|  | 2.4 |  | 5 |
|  | 2.5 |  | 6 |
| **CHAPTER 3** |  | **SYSTEM STUDY** |  |
|  |  |  |  |
|  | 3.1 | OVERVIEW | 8 |
|  | 3.2 | PURPOSE | 8 |
|  | 3.3 | SYSTEM REQUIREMENTS | 8 |
|  |  | 3.3.1 HARDWARE REQUIREMENTS | 8 |
|  |  | 3.3.2 SOFTWARE REQUIREMENTS | 8 |
|  |  |  |  |
| **CHAPTER 4** |  | **SYSTEM DESIGN** |  |
|  |  |  |  |
|  | 4.1 | SYSTEM ARCHITECTURE | 10 |
|  |  | 4.1.1 DESCRIPTION | 10 |
|  | 4.2 | PROJECT WORKFLOW | 11 |
| **CHAPTER 5** |  | **IMPLEMENTATION METHADOLOGY** |  |
|  |  |  |  |
|  | 5.1 | PREREQUISITES | 13 |
|  | 5.2 | ALGORITHMS | 13 |
|  |  | 5.2.1 ALLOTMENT LOWER BOUND | 13 |
|  |  | 5.2.2 THE ALLOTMENT AWARE SCHEDULING | 14 |
|  |  | 5.2.3 THE EXPECTED TIME PROCEDURE | 15 |
|  | 5.3 | CODING | 18 |
|  | 5.4 | SCREENSHOT | 44 |
| **CHAPTER 6** |  | **Results and Discussion** |  |
|  |  |  |  |
|  | 6.1 | PERFORMANCE ANALYSIS | 45 |
| **CONCLUSION** |  |  | 47 |
| **REFERENCES** |  |  | 49 |

**INTRODUCTION**

1. **INTRODUCTION**

Human behavioral pattern mining plays an important role in ambient assisted living since it enables service personalization. The computing and networking capabilities of mobile and wearable devices, makes them appropriate tools for obtaining and collecting information about user activities. We introduce scalable algorithms that utilize a variety of sensors, e.g., WiFi, location, etc. that are available on the device. By leveraging collected multivariate temporal data we can identify frequent human behavioral patterns (FBP) with a time estimation (temporal granularity), similar to the human perception of time. Identification of frequent patterns in human behavior has applications in several domains, which vary from recommendation systems to health care and transportation optimization. For instance, a health care application can monitor a user’s physical activity routine. However, if there is a change in their routines, which is not recognized or notified by the user (such as depression related behaviors), then the system can recognize this and notify care givers about the change.

* 1. **PROBLEM DESCRIPTION**

Given timestamped activities of the user, assuming they are occurring in a routine, the goal is to efficiently create a profile, which summarizes frequent behavioral patterns of a user.

*Eg (Output Profile) :*

{ confidence: 60 percent; 15:00; call:#951603XXXX; sms:#951603XXXX}

* 1. **PROJECT OBJECTIVE**

The objective of our project is to implement the dynamicity of human behavior by introducing a simple but novel human-centric temporal granularity method. Also to make the algorithms lightweight and can be integrated into small devices with limited computing capabilities. Here we use Generic mobile data mining system. We claim it is generic because of its multi-sensor support and application independence. Frequent item set mining algorithms and their sub components such as analyzing the temporal aspect of human behavior. It aims to identify human behavior, instead of the unique contextual data approach.

**LITERATURE REVIEW**

1. **LITERATURE REVIEW**
   1. **OVERVIEW**

This chapter clearly explains about the existing system for the proposed algorithm in the project. The section 2.2 deals with automatically recognizing places of interest and section 2.3 deals with mining your frequent patterns on your phone. The section 2.4 deals with a habit mining approach for discovering similar mobile users and section 2.5 deals with mining group movement patterns.

* 1. **AUTOMATIC RECOGNIZATION OF PLACES OF INTREST FROM UNRELIABLE GPS DATA**

The rise of GPS-enabled mobile devices has created exciting opportunities for tracking people, objects and vehicles, as well as animals using their digital traces. Such traces or trajectories can be mined to derive knowledge about the behaviour of moving objects for a range of purposes: for health monitoring, offender monitoring, animal migration etc.  Our novel algorithm employs both spatiotemporal density estimation and line count inference to predict and rank a user’s POI(s) at building level accuracy from noisy time-annotated GPS data points.

Although satellite-based GPS positioning provides good coverage for modern navigation and tracking; it does not work well for indoor places or urban canyons, due to poor line-of-sight transmission between the receiver and the satellites. For a GPS device to be accurate, a clear line of sight to satellites is necessary, so that signals from at least four satellites are available to the GPS receiver for computing its current position

Location is the most valuable piece of information in digital human behavior identification, and therefore it maps location onto human behavior. Human behavior is not just based on changes in location, and we should also include activities that are happening within the location.

* 1. **MOBILEMINER: MINING YOUR FREQUENT PATTERNS ON YOUR PHONE**

Smartphones can collect considerable context data about the user, ranging from apps used to places visited. Frequent user patterns discovered from longitudinal, multi-modal context data could help personalize and improve overall user experience. Our long term goal is to develop novel middleware and algorithms to efficiently mine user behavior patterns entirely on the phone by utilizing idle processor cycles.

Mining patterns on the mobile device provides better privacy guarantees to users, and reduces dependency on cloud connectivity. As an important step in this direction, we develop a novel general-purpose service called MobileMiner that runs on the phone and discovers frequent co-occurrence patterns indicating which context events frequently occur together.

In this work, we focus on the orthogonal, but equally important problem of inferring higher level behavior patterns from a longitudinal log of the raw and inferred context data collected by smartphones. We are finding the co-occurrences of activities. The approach is realistic in terms of deployment. We are just identifying the co-occurrences and not FBP’s.

* 1. **A HABIT MINING APPROACH FOR DISCOVERING SIMILAR MOBILE USERS**

Discovering similar users with respect to their habits plays an important role in a wide range of applications, such as collaborative filtering for recommendation, user segmentation for market analysis, etc. Recently, the progressing ability to sense user contexts of smart mobile devices makes it possible to discover mobile users with similar habits by mining their habits from their mobile devices. However, though some researchers have proposed effective methods for mining user habits such as behavior pattern mining, how to leverage the mined results for discovering similar users remains less explored. To this end, we propose a novel approach for conquering the sparseness of behavior pattern space and thus make it possible to discover similar mobile users with respect to their habits by leveraging behavior pattern mining.

To be specific, first, we normalize the raw context log of each user by transforming the location-based context data and user interaction records to more general representations. Second, we take advantage of a constraint-based Bayesian Matrix Factorization model for extracting the latent common habits among behavior patterns and then transforming behavior pattern vectors to the vectors of mined common habits which are in a much more dense space.

The experiments conducted on real data sets show that our approach outperforms three baselines in terms of the effectiveness of discovering similar mobile users with respect to their habits. It characterize user habits more precisely by identifying usage correlation with time and location. It transforms geographical coordinates based on the time of the day to “work” or “home”, and does not include precise time of the day while transforming.

* 1. **MINING GROUP MOVEMENT PATTERNS FOR TRACKING MOVING OBJECTS EFFECIENTLY**

The proposed algorithm is a distributed mining algorithm that identifies a group of objects with similar movement patterns. This information is important in some biological research domains, such as the study of animals’ social behavior and wildlife migration. The proposed algorithm comprises a local mining phase and a cluster ensembling phase. In the local mining phase, the algorithm finds movement patterns based on local trajectories. Then, based on the derived patterns, we propose a new similarity measure to compute the similarity of moving objects and identify the local group relationships.

To address the energy conservation issue in resource-constrained environments, the algorithm only transmits the local grouping results to the sink node for further ensembling. In the cluster ensembling phase, our algorithm combines the local grouping results to derive the group relationships from a global view.

We further leverage the mining results to track moving objects efficiently. The results of experiments show that the proposed mining algorithm achieves good grouping quality, and the mining technique helps reduce the energy consumption by reducing the amount of data to be transmitted. It uses trajectory of location for movement pattern prediction and classification. They use geographical coordinates their notion of location which is different than ours.

**SYSTEM STUDY**

1. **SYSTEM STUDY**
   1. **OVERVIEW**

System study includes the view of purpose of the project and business context related to the project and main functional requirements needed for doing the project and necessary system requirements installed in the system.

* 1. **PURPOSE**

To identify the human daily behavior patterns using a set of scalable algorithms. These patterns are extracted from multivariate temporal data that have been collected from smartphones. While transforming the location, our research can use all of the existing sensors on a device, it can be extended to any type of human behavior analysis application.

* 1. **SYSTEM REQUIREMENTS**
     1. **HARDWARE REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| Processor | : | Pentium 4 and above series |
| Hard disk | : | 500 GB |
| RAM | : | 4 GB |

* + 1. **SOFTWARE REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| IDE | : | Eclipse |
| SDK | : | Android |

**ANDROID**

Android is a [mobile operating system](https://en.wikipedia.org/wiki/Mobile_operating_system) developed by [Google](https://en.wikipedia.org/wiki/Google), based on the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel)  and designed primarily for  [touchscreen](https://en.wikipedia.org/wiki/Touchscreen)  mobile devices such as [smartphones](https://en.wikipedia.org/wiki/Smartphone) and [tablets](https://en.wikipedia.org/wiki/Tablet_computer). Android's [user interface](https://en.wikipedia.org/wiki/User_interface) is mainly based on [direct manipulation](https://en.wikipedia.org/wiki/Direct_manipulation_interface), using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a [virtual keyboard](https://en.wikipedia.org/wiki/Virtual_keyboard) for text input. In addition to touchscreen devices, Google has further developed [Android TV](https://en.wikipedia.org/wiki/Android_TV) for televisions, [Android Auto](https://en.wikipedia.org/wiki/Android_Auto) for cars, and [Android Wear](https://en.wikipedia.org/wiki/Android_Wear) for wrist watches, each with a specialized user interface. Variants of Android are also used on [notebooks](https://en.wikipedia.org/wiki/Laptop), [game consoles](https://en.wikipedia.org/wiki/Video_game_console), [digital cameras](https://en.wikipedia.org/wiki/Digital_camera), and other electronics.

SYSTEM DESIGN

1. **SYSTEM DESIGN**
   1. **SYSTEM ARCHITECTURE**

Time Conversion

Location-State Estimation

Data Transferation

FBP Detection

**Fig 4.1 Architecture of the System**

* + 1. **DESCRIPTION**

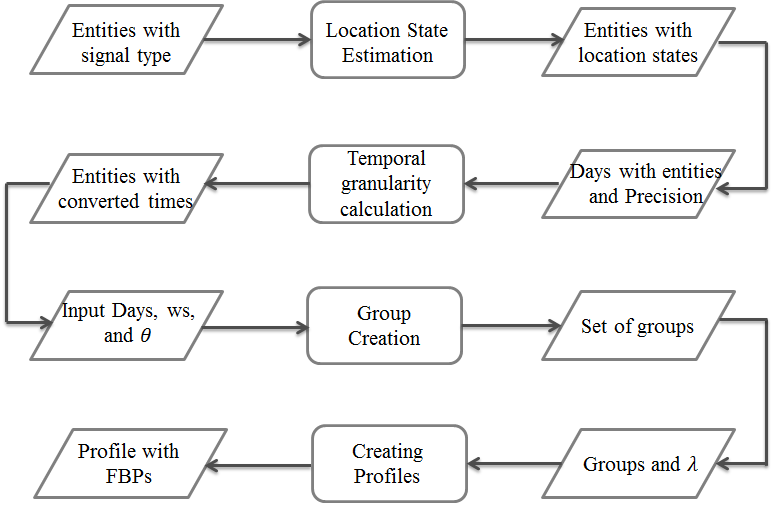
**Data Transferation :** In order to implement our algorithms, first the data format should be converted from heterogeneous data to machine-processable data, i.e., the raw data needs to be converted to the previously described entity format. As previously stated, the data has been collected from heterogeneous sources. Some sensors have multiple values, for instance WiFi has BSSID, SSID and Capalities (WPA, PSK, etc.). Nevertheless, for each sensor our model chooses only one value. In particular, each sensor (attribute)A, requires a single data point (value) D. Therefore, “BSSID” has been chosen for WiFi and Bluetooth, the pseudonymized phone number for SMS and Calls, “process name” for Application and tilting, in-vehicle, on-bicycle, walking, still, and unknown for the activity sensors (UbiqLog uses Google play services for activity recognition and therefore there is no raw accelerometer data inside the dataset).

**Location State Estimation :** In this phase, we propose an algorithm that identifies the movement (based on location changes) state, which will be used to enrich the semantics of the data within the notion of the location.

**Time Conversion :** In this phase, we need to convert the timestamp to a time similar to the human perception of time.

**FBP Detection :** In this last phase, we describe the behavior similarity and FBP detection algorithms. The above figure presents the flow of FBP detection from raw, heterogeneous sensor data.

* 1. **PROJECT WORKFLOW**



**Fig. 4.2 WorkFlow of the project**

**IMPLEMENTATION METHODOLOGY**

1. **IMPLEMENTATION METHODOLOGY**
   1. **PREREQUISITES**

The input for these algorithms are in the form of graph. Hence with the help of Task Graph Generator tool we should generate the graphs, which gives us two output files. One file contains the task graph and its adjacency relationship and the another file contains processor graph having execution time of each task in every processor.

* 1. **ALGORITHMS**

**THE ALLOTMENT LOWER BOUND**

**In the Strategy I**

AL(*i*) = 1 for all 1  *i n*

**In the Strategy II**

for *i* 1 to n do

if f1(vi ,1) then

AL(*i*) = 1

else if < f1(vi ,m - 1) then

AL(*i*) = m

else

Find a number *j* ϵ [1, m – 2] such that

f1(vi , j + 1) <= < f1(vi , j)

AL(i) = j + 1

Algorithm 1 is used to find the allotment lower bound which is the minimum number of replications for the tasks to be done. For Strategy I it is always 1. For Strategy II by considering the communication time the lower bound varies.

* + 1. **THE ALLOTMENT AWARE SCHEDULING**

1. Normalize the heterogeneous system into a homogeneous system.
2. Compute allotment parameter α’ by the linear program and the rounding procedure.
3. Compute αi  = max {min {α’i , µ}, AL (i)} for vi ϵ V.
4. Compute RRank for all tasks by traversing graph from the exit task.
5. Sort the tasks in a scheduling list by non – increasing order of RRank
6. while the scheduling list is not empty do
   1. Remove the first task vi from the scheduling list and set Si = Ø and CT = ∞.
   2. while αi > |Si | do
      1. Set Lmin = ∞ and pc = null
      2. for each pk ϵ P \ Si do

11. if Lmin > L(vi , pk , Si ) and CT > EFT (vi , pk , Si )

then

12. Set pc = pk and Lmin = L(vi ,pk , Si )

* + 1. If pc ≠ null then

14. Si = Si U {pc }

15.CT=

16. Else

No replication of task improve the make span, break the loop.

* + 1. **THE EXPECTED TIME PROCEDURE**

1. Compute the reliability probability R = e-λtp(v,p)  for the task v on the processor p and set t = tp(v,p).
2. Get a number n ε [0,1] uniformly at random.
3. While n > R do
4. Get a number p ε [0,1] uniformly at random
5. t = t+p \* tp(v,p) + 𝞱
6. Get a number n ε [0,1] uniformly at random.
   1. **CODING**

**For Strategy I**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package projects1;

import java.io.\*;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.LinkedHashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\*

\* @author DELL

\*/

public class ProjectS1 {

/\*\*

\* @param args the command line arguments

\*/

static int [] task;

static int noOfProcessor,noOfTask,noOfEdges;

static int[][] proTime;

static double [] failureRate;

static double [] meanProTime;

static double meanRecTime;

static double [] recTime;

static double [][] proCommTime;

static double [] x;

static double beta;

static double [] gamma;

static double meanGamma;

static double [] alphadash;

static double [] w;

static Set<Integer>[] Si;

static Set<Integer>[] SSj;

static double[][] commfailureRate;

static Set<Integer> pk=new LinkedHashSet<>();

static Set<Double> newset=new LinkedHashSet<>();

static LinkedList<Node>[] list;

static ArrayList<Predecessor> predList;

static double allotpar;

static int[] ai;

static int[] rank;

static double[] avaipro;

static int max;

static double Ct;

static double lMin;

static int pc;

static double fin=0;

static int resrbar;

public static void main(String[] args) throws IOException {

/\*Reading data from the file

\*/

Predecessor pred;

DataInputStream in = null;

DecimalFormat newFormat = new DecimalFormat("#.#");

int avgProTime;

int avgCommTime;

int i,j;

int count=0;

int sum=0;

double ro=0.9999;

double sumf=0.0,meanValue;

int[] aL;

double m1;

String c;

String[] str;

j=1;

try {

in = new DataInputStream(new FileInputStream("C:\\Users\\DELL\\Documents\\IV-Project\\tggwin\\dist\\f1.txt"));

noOfTask=Integer.parseInt(in.readLine());

noOfEdges=Integer.parseInt(in.readLine());

System.out.println("No of Task:"+noOfTask);

System.out.println("No of Edges:"+noOfEdges);

meanProTime=new double[noOfTask+1];

task = new int[noOfTask+1];

predList=new ArrayList();

for(i=1;i<=noOfTask;i++)

task[i]=i;

list = new LinkedList[noOfTask+1];

aL=new int[noOfTask+1];

for(i=1;i<=noOfTask;i++){

list[i]=new LinkedList<>();

}

in.readLine();

c=in.readLine();

while (c!=null) {

str=c.split("\t");

int index;

index=Integer.parseInt(str[0]);

Node node=new Node(Integer.parseInt(str[1]),Integer.parseInt(str[2]));

list[index].add(node);

pred=new Predecessor(Integer.parseInt(str[1]),Integer.parseInt(str[0]),Integer.parseInt(str[2]));

predList.add(pred);

c=in.readLine();

}

}

finally

{

if (in != null)

{

in.close();

}

}

try {

in = new DataInputStream(new FileInputStream("C:\\Users\\DELL\\Documents\\IV-Project\\tggwin\\dist\\f2.txt"));

in.readLine();

noOfProcessor=Integer.parseInt(in.readLine());

System.out.println("No of Processor:"+noOfProcessor);

proTime=new int[noOfTask+1][noOfProcessor+1];

commfailureRate=new double[noOfProcessor+1][noOfProcessor+1];

failureRate=new double[noOfProcessor+1];

gamma=new double[noOfProcessor+1];

in.readLine();

c=in.readLine();

i=1;

while (c!=null) {

str=c.split("\t");

for(int k=0;k<str.length;k++,j++){

proTime[i][j]=Integer.parseInt(str[k]);

}

j=1;

i++;

c=in.readLine();

}

}

finally {

if (in != null) {

in.close();

}

}

proCommTime = new double[noOfProcessor+1][noOfProcessor+1];

for(int i1=1;i1<=noOfProcessor;i1++)

{

for(int j1=i1;j1<=noOfProcessor;j1++)

{

if(i1==j1)

{

proCommTime[i1][j1]=0;

}

else

{

proCommTime[j1][i1]=proCommTime[i1][j1]=Math.random()\*(1.25-0.75)+0.75;

}

}

}

m1=noOfProcessor;

allotpar=((m1\*(2.0+ro))-(Math.sqrt(((2.0+(2.0\*ro)+(ro\*ro))\*(m1\*m1))-((2.0\*m1)\*(1+ro)))))/2.0;

System.out.println("Allotment Paramter : "+allotpar);

for(int ii=1;ii<=noOfTask;ii++){

for(int jj=1;jj<=noOfProcessor;jj++){

sum+=proTime[ii][jj];

}

meanProTime[ii]=sum/noOfProcessor;

sum=0;

}

w=new double[noOfProcessor+1];

recTime=new double[noOfProcessor+1];

for(i=1;i<=noOfProcessor;i++)

{

w[i]=(double) Math.random()\*(1.25-0.75)+0.75;

recTime[i]=1/w[i];

sumf+=recTime[i];

}

meanRecTime=sumf/noOfProcessor;

System.out.println("Mean Recovery Time "+meanRecTime);

for(i=1;i<=noOfProcessor;i++)

{

failureRate[i]=Math.random()\*(0.01-0.001)+0.001;

}

for(i=1;i<=noOfProcessor;i++)

{

for(j=i+1;i<=noOfProcessor;i++)

{

commfailureRate[i][j]=Math.random()\*(0.01-0.001)+0.001;

}

}

int sumgamma=0;

for(int vii=1;vii<=noOfProcessor;vii++)

{

gamma[vii]=Math.random()\*(0.9-0.5)+0.5;

sumgamma+=gamma[vii];

}

meanGamma=sumgamma/noOfProcessor;

//Algorithm 1

//Strategy I

for(i=1;i<=noOfTask;i++)

aL[i]=1;

i=1;

int []xnew;

int mj;

xnew=new int[noOfTask+1];

System.out.println("Finding Upper Bound");

x = new double[noOfTask+1];

alphadash = new double[noOfTask+1];

for(int vi=1;vi<=noOfTask;vi++){

alphadash[vi]=aL[vi]+1;

for(int j1=aL[vi];j1<noOfProcessor;j1++){

if(j1==noOfProcessor || proTime[vi][j1]!= proTime[vi][j1+1]){

xnew[vi]=proTime[vi][j1];

}

x[vi]=Math.ceil(Math.random()\*(xnew[vi]-0)+0);

if((int)expectedProTime(vi,j1)==x[vi])

{

alphadash[vi]=j1;

}

}

}

/\*algrithm 2\*/

//step3

ai=new int[noOfTask+1];

for(int vi=1;vi<=noOfTask;vi++)

{

ai[vi]=alpha(alphadash[vi],allotpar,aL[i]);

}

//step 4

Map<Integer, Integer> lMap=new HashMap<Integer, Integer>();

rank = new int[noOfTask+1];

for(int vi=noOfTask;vi>=1;vi--)

{

beta=Math.random()\*(5-0)+0;

rank[vi]=(int) rRank(vi,ai[vi]);

lMap.put(vi,rank[vi]);

}

//step 5

int[] sortedTask=new int[noOfTask+1];

int x=1;

Map<Integer, Integer> map = sortByValues((HashMap) lMap);

Set set2 = map.entrySet();

Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

Map.Entry me2 = (Map.Entry)iterator2.next();

sortedTask[x]=(int) me2.getKey();

x++;

}

System.out.println("Task\tMeanProTime\tLB\tUB\tRank\tSortedTask\tAP");

for(i=1;i<=noOfTask;i++)

{

System.out.println(i+"\t"+meanProTime[i]+"\t\t"+aL[i]+"\t"+alphadash[i]+"\t"+rank[i]+"\t"+sortedTask[i]+"\t\t"+ai[sortedTask[i]]);

}

Si=new LinkedHashSet[noOfTask+1];

SSj=new LinkedHashSet[noOfTask+1];

avaipro=new double[noOfProcessor+1];

Ct=99999;

double soh,eft;

for(int vi=1;vi<=noOfTask;vi++)

{

Si[vi]=new LinkedHashSet();

SSj[vi]=new LinkedHashSet();

}

for(int vi=1;vi<=noOfTask;vi++){

while(ai[sortedTask[vi]]>Si[sortedTask[vi]].size()){

lMin=9999;

pc=0;

for(int pk=1;pk<=noOfProcessor;pk++)

{

if(!Si[sortedTask[vi]].contains(pk))

{

soh=sumOfoh(sortedTask[vi],pk,Si[sortedTask[vi]]);

eft=earliestFinTime(sortedTask[vi],pk,Si[sortedTask[vi]]);

if((lMin>soh)&& (Ct>eft))

{

pc=pk;

lMin=soh;

}

}

}

if(pc!=0){

Si[sortedTask[vi]].add(pc);

eft=earliestFinTime(sortedTask[vi],pc,Si[sortedTask[vi]]);

avaipro[pc]=eft;

for(Iterator it=Si[sortedTask[vi]].iterator();it.hasNext();)

{

newset.add(earliestFinTime(sortedTask[vi], (int) it.next(),Si[sortedTask[vi]]));

}

double maxx=Double.MIN\_VALUE;

for(double val: newset)

{

if(val>maxx)

{

maxx=val;

}

}

Ct=maxx;

newset.clear();

}

else

{

break;

}

}

System.out.println("Allocated processors for the task "+sortedTask[vi]+" : "+Si[sortedTask[vi]]+"\nCompletion Time : "+Ct);

}

//algorithm 3

for(int v=1;v<=noOfTask;v++){

for(int p=1;p<=noOfProcessor;p++){

double r;

double t;

r=reliability(v,p);

t=proTime[v][p];

double n;

n=(Math.random()\*(1-0)+0);

while(n>r){

p=(int) (Math.random()\*(1-0)+0);

t=t+p\*proTime[v][p]+meanRecTime;

n=(Math.random()\*(1-0)+0);

}

}

}

}

public static double function1(int vi,int ai)

{

double res;

double temp1,temp2;

double mf;

DecimalFormat newFormat = new DecimalFormat("#.###");

mf=meanFailure(vi);

temp1=(double) (2\*(Math.pow(mf,2\*ai+1)-(ai+1)\*Math.pow(mf,ai+1)+ai\*Math.pow(mf,ai)));

temp2=(double) (2+Math.pow(mf,2\*ai+1)+(ai-1)\*Math.pow(mf,ai+1)-(ai+2)\*Math.pow(mf,ai));

res=temp1/temp2;

res = Double.valueOf(newFormat.format(res));

return res;

}

public static int alpha(double alphadash1,double allot,int al)

{

return (int) Math.max(Math.min(alphadash1, allot),al);

}

public static double meanFailure(int vi)

{

return 1-meanReliability(vi);

}

public static double meanReliability(int vi)

{

int i,j;

double sum=0;

for(i=1;i<=noOfProcessor;i++)

{

sum+=Math.exp(-failureRate[i]\*proTime[vi][i]);

}

sum=sum/noOfProcessor;

return sum;

}

public static double failure(int vi,int px)

{

// System.out.println("Failure");

return 1-reliability(vi,px);

}

public static double reliability(int vi,int px)

{

//System.out.println("Reliabbility");

return Math.exp(-failureRate[px]\*proTime[vi][px]);

}

public static double expectedProTime(int vi,int ai)

{

double mef=meanFailure(vi);

return (meanProTime[vi]+(((Math.pow(mef, ai))/(1-(Math.pow(mef, ai))))\*((meanProTime[vi]/2)+meanRecTime)));

}

public static double earliestFinTime(int vi,int pk,Set si)

{

double v1,v2;

SSj[vi].add(pk);

double fail=1.0;

for(Object r : SSj[vi])

{

fail\*=failure(vi, (int) r);

}

v1=earliestStaTime(vi,pk);

v2=((proTime[vi][pk])+(fail/(1-fail))\*(((proTime[vi][pk])/2))+recTime[pk]);

return v1+v2;

}

public static double meanTaskCommTime(int i)

{

int sum=0,count=0;

LinkedList<Node> head;

head = list[i];

for(Node ele :head)

{

sum+=ele.getCost();

count++;

}

return sum/count;

}

public static int rRank(int vi,int aii)

{

max=0;

int resrank;

double mtct,rr;

for(Node ele:list[vi])

{

mtct=meanTaskCommTime(vi);

rr=rRank(ele.getAdj(),aii);

if(max<(mtct+rr))

{

max=(int) (mtct+rr);

}

}

resrank= (int) (expectedProTime(vi,aii)\*(1+beta\*Math.pow((1-rbar(vi)), aii)));

return resrank;

}

public static double rbar(int vi)

{

double result;

int k=0;

double mr;

Iterator itr=predList.iterator();

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vi){

k=pred.getPred();

resrbar\*=(1-(1-Math.pow(meanCommReliability(k,vi),ai[k])));

}

}

mr=meanReliability(vi);

result=(mr)/(1-meanGamma+meanGamma\*mr)\*resrbar;

return result;

}

public static double earliestStaTime(int vj,int px)

{

double rrr;

rrr= Math.max(dataReadyTime(vj,px),available(vj,px));

return rrr;

}

public static double dataReadyTime(int vj,int px)

{

Set s=new LinkedHashSet();

Iterator itr=predList.iterator();

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vj)

{

s.add(earliestCommFinTime(pred.getPred(),vj,px,Si[pred.getPred()]));

}

}

double maxValue = 0;

for (Object value : s) {

if ((double)value > maxValue) {

maxValue = (double)value;

}

}

return maxValue;

}

public static double earliestCommFinTime(int i,int j,int x,Set ss)

{

double size=0,ef,ctp;

if(Si[i].size()!=0){

size=1/Si[i].size();

int pp;

for(Iterator k=Si[i].iterator();k.hasNext();)

{

pp=(int)k.next();

ef=earliestFinTime(i,pp,ss);

ctp=commtaskproTime(i,j,pp,x);

fin=fin+ef+ctp;

}

}

return size\*fin;

}

public static double commtaskproTime(int i,int j,int k,int x)

{

if(k==x)

{

return 0;

}

else

{

LinkedList<Node> head;

head = list[i];

double weij = 0,wlkx;

for(Node ele :head)

{

int adj=ele.getAdj();

if(adj==j)

{

weij=ele.getCost();

}

}

wlkx=proCommTime[k][x];

return weij/wlkx;

}

}

public static double sumOfoh(int vj,int pk,Set Si)

{

double rj;

double F;

double vv,xx;

F=fail(vj,pk,Si);

rj=funR(vj,pk,Si);

vv=(beta\*(1-rj)\*proTime[vj][pk])+(beta\*(1-rj)\*F/(1-F)\*((proTime[vj][pk]/2)+recTime[pk]));

xx=earliestFinTime(vj,pk,Si);

return vv+xx;

}

public static double fail(int vj,int pk,Set sj)

{

double failpro=1;

SSj[vj].add(pk);

for(Object r : SSj[vj])

{

failpro\*=failure(vj, (int) r);

}

return failpro;

}

public static double funR(int vj,int pk,Set sj)

{

double rj;

double meanR;

double value;

double fails=1;

value=1;

Set s=new LinkedHashSet();

Iterator itr=predList.iterator();

for(Object r : sj)

{

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vj)

{

fails\*=commReliability(vj,pred.getPred(),(int)r,pk);

}

}

value\*=(1-fails);

}

meanR=meanReliability(vj);

rj=(meanR/(1-gamma[pk]+gamma[pk]\*meanR))\*value;

return rj;

}

public static double commReliability(int i,int j,int k,int x)

{

return Math.exp(-commfailureRate[k][x]\*commtaskproTime(i,j,k,x));

}

public static double meanCommReliability(int i,int j)

{

double val=0.0;

for(int k1=1;k1<=noOfProcessor;k1++)

{

for(int k2=k1;k2<=noOfProcessor;k2++)

{

val+=(2\*commReliability(i,j,k1,k2))/(noOfProcessor\*(noOfProcessor-1));

}

}

return val;

}

public static double commFailure(int i,int j,int k,int x)

{

return 1-commReliability(i,j,k,x);

}

public static double expecWorkSpan(int vi,int ai)

{

double meaf=meanFailure(vi);

return ((ai\*meanProTime[vi])+(((ai\*Math.pow(meaf,ai))/(1-Math.pow(meaf,ai)))\*((meanProTime[vi]/2)+meanRecTime)));

}

public static double available(int vj,int px)

{

return avaipro[px];

}

private static HashMap sortByValues(HashMap map) {

List list = new LinkedList(map.entrySet());

Collections.sort(list, new Comparator() {

public int compare(Object o1, Object o2) {

return ((Comparable) ((Map.Entry) (o2)).getValue())

.compareTo(((Map.Entry) (o1)).getValue());

}

});

HashMap sortedHashMap = new LinkedHashMap();

for (Iterator it = list.iterator(); it.hasNext();) {

Map.Entry entry = (Map.Entry) it.next();

sortedHashMap.put(entry.getKey(), entry.getValue());

}

return sortedHashMap;

}

}

class Node

{

int adj;

int cost;

public Node(int adj,int cost)

{

this.adj=adj;

this.cost=cost;

}

public int getAdj()

{

return adj;

}

public int getCost()

{

return cost;

}

}

class Predecessor

{

int task,pred,cost;

Predecessor(int task,int pred,int cost)

{

this.task=task;

this.pred=pred;

this.cost=cost;

}

public int getTask()

{

return task;

}

public int getCost()

{

return cost;

}

public int getPred()

{

return pred;

}

}

**For Strategy II**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package projects2;

import java.io.\*;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.LinkedHashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\*

\* @author DELL

\*/

public class ProjectS2 {

/\*\*

\* @param args the command line arguments

\*/

static int [] task;

static int noOfProcessor,noOfTask,noOfEdges;

static int[][] proTime;

static double [] failureRate;

static double [] meanProTime;

static double meanRecTime;

static double [] recTime;

static double [][] proCommTime;

static double [] x;

static double beta;

static double [] gamma;

static double meanGamma;

static double [] alphadash;

static double [] w;

static Set<Integer>[] Si;

static Set<Integer>[] SSj;

static double[][] commfailureRate;

static Set<Integer> pk=new LinkedHashSet<>();

static Set<Double> newset=new LinkedHashSet<>();

static LinkedList<Node>[] list;

static ArrayList<Predecessor> predList;

static double allotpar;

static int[] ai;

static int[] rank;

static double[] avaipro;

static int max;

static double Ct;

static double lMin;

static int pc;

static double fin=0;

static int resrbar;

private static double[] wotask;

private static double exectime;

public static double eeft;

private static double t;

public static int[] sortedTask;

public static void main(String[] args) throws IOException {

/\*Reading data from the file

\*/

Predecessor pred;

DataInputStream in = null;

DecimalFormat newFormat = new DecimalFormat("#.#");

int avgProTime;

int avgCommTime;

int i,j;

int count=0;

int sum=0;

double ro=0.9999;

double sumf=0.0,meanValue;

int[] aL;

double m1;

String c;

String[] str;

j=1;

try {

in = new DataInputStream(new FileInputStream("C:\\Users\\DELL\\Documents\\IV-Project\\tggwin\\dist\\f1.txt"));

noOfTask=Integer.parseInt(in.readLine());

noOfEdges=Integer.parseInt(in.readLine());

System.out.println("No of Task:"+noOfTask);

System.out.println("No of Edges:"+noOfEdges);

meanProTime=new double[noOfTask+1];

task = new int[noOfTask+1];

predList=new ArrayList();

for(i=1;i<=noOfTask;i++)

task[i]=i;

list = new LinkedList[noOfTask+1];

aL=new int[noOfTask+1];

for(i=1;i<=noOfTask;i++){

list[i]=new LinkedList<>();

}

in.readLine();

c=in.readLine();

while (c!=null) {

str=c.split("\t");

int index;

index=Integer.parseInt(str[0]);

Node node=new Node(Integer.parseInt(str[1]),Integer.parseInt(str[2]));

list[index].add(node);

pred=new Predecessor(Integer.parseInt(str[1]),Integer.parseInt(str[0]),Integer.parseInt(str[2]));

predList.add(pred);

c=in.readLine();

}

}

finally

{

if (in != null)

{

in.close();

}

}

try {

in = new DataInputStream(new FileInputStream("C:\\Users\\DELL\\Documents\\IV-Project\\tggwin\\dist\\f2.txt"));

in.readLine();

noOfProcessor=Integer.parseInt(in.readLine());

System.out.println("No of Processor:"+noOfProcessor);

proTime=new int[noOfTask+1][noOfProcessor+1];

commfailureRate=new double[noOfProcessor+1][noOfProcessor+1];

failureRate=new double[noOfProcessor+1];

gamma=new double[noOfProcessor+1];

in.readLine();

c=in.readLine();

i=1;

while (c!=null) {

str=c.split("\t");

for(int k=0;k<str.length;k++,j++){

proTime[i][j]=Integer.parseInt(str[k]);

}

j=1;

i++;

c=in.readLine();

}

}

finally {

if (in != null) {

in.close();

}

}

proCommTime = new double[noOfProcessor+1][noOfProcessor+1];

for(int i1=1;i1<=noOfProcessor;i1++)

{

for(int j1=i1;j1<=noOfProcessor;j1++)

{

if(i1==j1)

{

proCommTime[i1][j1]=0;

}

else

{

proCommTime[j1][i1]=proCommTime[i1][j1]=Math.random()\*(1.25-0.75)+0.75;

}

}

}

m1=noOfProcessor;

allotpar=((m1\*(2.0+ro))-(Math.sqrt(((2.0+(2.0\*ro)+(ro\*ro))\*(m1\*m1))-((2.0\*m1)\*(1+ro)))))/2.0;

System.out.println("Allotment Paramter : "+allotpar);

for(int ii=1;ii<=noOfTask;ii++){

for(int jj=1;jj<=noOfProcessor;jj++){

sum+=proTime[ii][jj];

}

meanProTime[ii]=sum/noOfProcessor;

sum=0;

}

w=new double[noOfProcessor+1];

recTime=new double[noOfProcessor+1];

for(i=1;i<=noOfProcessor;i++)

{

w[i]=(double) Math.random()\*(1.25-0.75)+0.75;

recTime[i]=1/w[i];

sumf+=recTime[i];

}

meanRecTime=sumf/noOfProcessor;

System.out.println("Mean Recovery Time "+meanRecTime);

for(i=1;i<=noOfProcessor;i++)

{

failureRate[i]=Math.random()\*(0.01-0.001)+0.001;

}

for(i=1;i<=noOfProcessor;i++)

{

for(j=i+1;i<=noOfProcessor;i++)

{

commfailureRate[i][j]=Math.random()\*(0.01-0.001)+0.001;

}

}

int sumgamma=0;

for(int vii=1;vii<=noOfProcessor;vii++)

{

gamma[vii]=Math.random()\*(0.9-0.5)+0.5;

sumgamma+=gamma[vii];

}

meanGamma=sumgamma/noOfProcessor;

//Algorithm 1

//In Strategy II

for(i=1;i<=noOfTask;i++)

{

if((meanProTime[i]/meanRecTime)>=function1(i,1))

aL[i]=1;

else if((meanProTime[i]/meanRecTime)<function1(i,noOfProcessor-1))

aL[i]=noOfProcessor;

else

{

for(j=1;j<=noOfProcessor-2;j++)

{

if(function1(i,j+1)<=(meanProTime[i]/meanRecTime) &&(meanProTime[i]/meanRecTime) <function1(i,j))

aL[i]=j+1;

}

}

}

i=1;

int []xnew;

int mj;

xnew=new int[noOfTask+1];

System.out.println("Finding Upper Bound");

//finding upper bound

x = new double[noOfTask+1];

alphadash = new double[noOfTask+1];

//CVTA(Constructing Virtual Task Algorithm)

for(int vi=1;vi<=noOfTask;vi++){

alphadash[vi]=aL[vi]+1;

for(int j1=aL[vi];j1<noOfProcessor;j1++){

if(j1==noOfProcessor || proTime[vi][j1]!= proTime[vi][j1+1]){

xnew[vi]=proTime[vi][j1];

}

x[vi]=Math.ceil(Math.random()\*(xnew[vi]-0)+0);

if((int)expectedProTime(vi,j1)==x[vi])

{

alphadash[vi]=j1;

}

}

}

/\*algrithm 2\*/

//step3

ai=new int[noOfTask+1];

wotask=new double[noOfTask+1];

for(int vi=1;vi<=noOfTask;vi++)

{

wotask[vi]=Math.random()\*(59-1)+1;

}

for(int vi=1;vi<=noOfTask;vi++)

{

ai[vi]=alpha(alphadash[vi],allotpar,aL[i]);

}

//step 4

Map<Integer, Integer> lMap=new HashMap<Integer, Integer>();

rank = new int[noOfTask+1];

for(int vi=noOfTask;vi>=1;vi--)

{

beta=Math.random()\*(5-0)+0;

rank[vi]=(int) rRank(vi,ai[vi]);

lMap.put(vi,rank[vi]);

}

//step 5

sortedTask=new int[noOfTask+1];

int x=1;

Map<Integer, Integer> map = sortByValues((HashMap) lMap);

Set set2 = map.entrySet();

Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

Map.Entry me2 = (Map.Entry)iterator2.next();

sortedTask[x]=(int) me2.getKey();

x++;

}

System.out.println("Task\tMeanProTime\tLB\tUB\tRank\tSortedTask\tAP");

for(i=1;i<=noOfTask;i++)

{

System.out.println(i+"\t"+meanProTime[i]+"\t\t"+aL[i]+"\t"+alphadash[i]+"\t"+rank[i]+"\t"+sortedTask[i]+"\t\t"+ai[sortedTask[i]]);

}

Si=new LinkedHashSet[noOfTask+1];

SSj=new LinkedHashSet[noOfTask+1];

avaipro=new double[noOfProcessor+1];

double soh,eft;

for(int vi=1;vi<=noOfTask;vi++)

{

Si[vi]=new LinkedHashSet();

SSj[vi]=new LinkedHashSet();

}

for(int vi=1;vi<=noOfTask;vi++){

Ct=99999;

exectime=0;

eeft=1.0;

while(ai[sortedTask[vi]]>Si[sortedTask[vi]].size() && exectime<eeft){

lMin=9999;

pc=0;

for(int pk=1;pk<=noOfProcessor;pk++)

{

if(!Si[sortedTask[vi]].contains(pk))

{

soh=sumOfoh(sortedTask[vi],pk,Si[sortedTask[vi]]);

eft=earliestFinTime(sortedTask[vi],pk,Si[sortedTask[vi]]);

if((lMin>soh)&& (Ct>eft))

{

pc=pk;

lMin=soh;

}

}

}

if(pc!=0){

Si[sortedTask[vi]].add(pc);

eeft=earliestFinTime(sortedTask[vi],pc,Si[sortedTask[vi]]);

avaipro[pc]=eeft;

for(Iterator it=Si[sortedTask[vi]].iterator();it.hasNext();)

{

newset.add(earliestFinTime(sortedTask[vi], (int) it.next(),Si[sortedTask[vi]]));

}

double maxx=Double.MIN\_VALUE;

for(double val: newset)

{

if(val>maxx)

{

maxx=val;

}

}

Ct=maxx;

newset.clear();

}

else

{

break;

}

exectime=exectime+2;

}

System.out.println("Allocated processors for the task "+sortedTask[vi]+" : "+Si[sortedTask[vi]]+"\nCompletion Time : "+Ct);

}

//algorithm 3

for(int v=1;v<=noOfTask;v++){

for(int p=1;p<=noOfProcessor;p++){

double r;

r=reliability(v,p);

t=proTime[v][p];

double n;

n=(Math.random()\*(1-0)+0);

while(n>r){

p=(int) (Math.random()\*(1-0)+0);

t=t+p\*proTime[v][p]+meanRecTime;

n=(Math.random()\*(1-0)+0);

}

}

System.out.println("processing time of task "+v+" is:" +t);

}

}

public static double function1(int vi,int ai)

{

double res;

double temp1,temp2;

double mf;

DecimalFormat newFormat = new DecimalFormat("#.###");

mf=meanFailure(vi);

temp1=(double) (2\*(Math.pow(mf,2\*ai+1)-(ai+1)\*Math.pow(mf,ai+1)+ai\*Math.pow(mf,ai)));

temp2=(double) (2+Math.pow(mf,2\*ai+1)+(ai-1)\*Math.pow(mf,ai+1)-(ai+2)\*Math.pow(mf,ai));

res=temp1/temp2;

res = Double.valueOf(newFormat.format(res));

return res;

}

public static int alpha(double alphadash1,double allot,int al)

{

return (int) Math.max(Math.min(alphadash1, allot),al);

}

public static double meanFailure(int vi)

{

return 1-meanReliability(vi);

}

public static double meanReliability(int vi)

{

int i,j;

double sum=0;

for(i=1;i<=noOfProcessor;i++)

{

sum+=Math.exp(-failureRate[i]\*proTime[vi][i]);

}

sum=sum/noOfProcessor;

return sum;

}

public static double failure(int vi,int px)

{

// System.out.println("Failure");

return 1-reliability(vi,px);

}

public static double reliability(int vi,int px)

{

//System.out.println("Reliabbility");

return Math.exp(-failureRate[px]\*proTime[vi][px]);

}

public static double expectedProTime(int vi,int ai)

{

double mef=meanFailure(vi);

return (meanProTime[vi]+(((Math.pow(mef, ai))/(1-(Math.pow(mef, ai))))\*((meanProTime[vi]/2)+meanRecTime)));

}

public static double earliestFinTime(int vi,int pk,Set si)

{

double v1,v2;

//System.out.println("Earliest");

SSj[vi].add(pk);

double fail=1.0;

for(Object r : SSj[vi])

{

fail\*=failure(vi, (int) r);

}

//System.out.println("VI "+vi+"PK "+pk);

v1=earliestStaTime(vi,pk);

v2=((proTime[vi][pk])+(fail/(1-fail))\*(((proTime[vi][pk])/2))+recTime[pk]);

//System.out.println("Value"+v1+" "+v2);

return v1+v2;

}

public static double meanTaskCommTime(int i)

{

int sum=0,count=0;

LinkedList<Node> head;

head = list[i];

for(Node ele :head)

{

sum+=ele.getCost();

count++;

}

return sum/count;

}

public static int rRank(int vi,int aii)

{

max=0;

int resrank;

double mtct,rr;

//ai[vi] = (int)Math.random()\*(5-2)+2;

for(Node ele:list[vi])

{

mtct=meanTaskCommTime(vi);

rr=rRank(ele.getAdj(),aii);

if(max<(mtct+rr))

{

max=(int) (mtct+rr);

}

}

resrank= (int) (expectedProTime(vi,aii)\*(1+beta\*Math.pow((1-rbar(vi)), aii)));

return resrank;

}

public static double rbar(int vi)

{

double result;

int k=0;

double mr;

Iterator itr=predList.iterator();

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vi){

k=pred.getPred();

resrbar\*=(1-(1-Math.pow(meanCommReliability(k,vi),ai[k])));

}

}

mr=meanReliability(vi);

result=(mr)/(1-meanGamma+meanGamma\*mr)\*resrbar;

return result;

}

public static double earliestStaTime(int vj,int px)

{

double rrr;

rrr= Math.max(dataReadyTime(vj,px),available(vj,px));

return rrr;

}

public static double dataReadyTime(int vj,int px)

{

Set s=new LinkedHashSet();

Iterator itr=predList.iterator();

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vj)

{

s.add(earliestCommFinTime(pred.getPred(),vj,px,Si[pred.getPred()]));

}

}

double maxValue = 0;

for (Object value : s) {

if ((double)value > maxValue) {

maxValue = (double)value;

}

}

return maxValue;

}

public static double earliestCommFinTime(int i,int j,int x,Set ss)

{

double size=0,ef,ctp;

if(Si[i].size()!=0){

size=1/Si[i].size();

int pp;

for(Iterator k=Si[i].iterator();k.hasNext();)

{

pp=(int)k.next();

ef=earliestFinTime(i,pp,ss);

ctp=commtaskproTime(i,j,pp,x);

fin=fin+ef+ctp;

}

}

return size\*fin;

}

public static double commtaskproTime(int i,int j,int k,int x)

{

if(k==x)

{

return 0;

}

else

{

LinkedList<Node> head;

head = list[i];

double weij = 0,wlkx;

for(Node ele :head)

{

int adj=ele.getAdj();

if(adj==j)

{

weij=ele.getCost();

}

}

wlkx=proCommTime[k][x];

return weij/wlkx;

}

}

public static double sumOfoh(int vj,int pk,Set Si)

{

double rj;

double F;

double vv,xx;

F=fail(vj,pk,Si);

rj=funR(vj,pk,Si);

vv=(beta\*(1-rj)\*proTime[vj][pk])+(beta\*(1-rj)\*F/(1-F)\*((proTime[vj][pk]/2)+recTime[pk]));

xx=earliestFinTime(vj,pk,Si);

return vv+xx;

}

public static double fail(int vj,int pk,Set sj)

{

double failpro=1;

SSj[vj].add(pk);

for(Object r : SSj[vj])

{

failpro\*=failure(vj, (int) r);

}

return failpro;

}

public static double funR(int vj,int pk,Set sj)

{

double rj;

double meanR;

double value;

double fails=1;

value=1;

Set s=new LinkedHashSet();

Iterator itr=predList.iterator();

for(Object r : sj)

{

while(itr.hasNext()){

Predecessor pred=(Predecessor)itr.next();

if(pred.getTask()==vj)

{

fails\*=commReliability(vj,pred.getPred(),(int)r,pk);

}

}

value\*=(1-fails);

}

meanR=meanReliability(vj);

rj=(meanR/(1-gamma[pk]+gamma[pk]\*meanR))\*value;

return rj;

}

public static double commReliability(int i,int j,int k,int x)

{

return Math.exp(-commfailureRate[k][x]\*commtaskproTime(i,j,k,x));

}

public static double meanCommReliability(int i,int j)

{

double val=0.0;

for(int k1=1;k1<=noOfProcessor;k1++)

{

for(int k2=k1;k2<=noOfProcessor;k2++)

{

val+=(2\*commReliability(i,j,k1,k2))/(noOfProcessor\*(noOfProcessor-1));

}

}

return val;

}

public static double commFailure(int i,int j,int k,int x)

{

return 1-commReliability(i,j,k,x);

}

public static double expecWorkSpan(int vi,int ai)

{

double meaf=meanFailure(vi);

return ((ai\*meanProTime[vi])+(((ai\*Math.pow(meaf,ai))/(1-Math.pow(meaf,ai)))\*((meanProTime[vi]/2)+meanRecTime)));

}

public static double available(int vj,int px)

{

// System.out.println("Available");

return avaipro[px];

}

private static HashMap sortByValues(HashMap map) {

List list = new LinkedList(map.entrySet());

Collections.sort(list, new Comparator() {

public int compare(Object o1, Object o2) {

return ((Comparable) ((Map.Entry) (o2)).getValue())

.compareTo(((Map.Entry) (o1)).getValue());

}

});

HashMap sortedHashMap = new LinkedHashMap();

for (Iterator it = list.iterator(); it.hasNext();) {

Map.Entry entry = (Map.Entry) it.next();

sortedHashMap.put(entry.getKey(), entry.getValue());

}

return sortedHashMap;

}

}

class Node

{

int adj;

int cost;

public Node(int adj,int cost)

{

this.adj=adj;

this.cost=cost;

}

public int getAdj()

{

return adj;

}

public int getCost()

{

return cost;

}

}

class Predecessor

{

int task,pred,cost;

Predecessor(int task,int pred,int cost)

{

this.task=task;

this.pred=pred;

this.cost=cost;

}

public int getTask()

{

return task;

}

public int getCost()

{

return cost;

}

public int getPred()

{

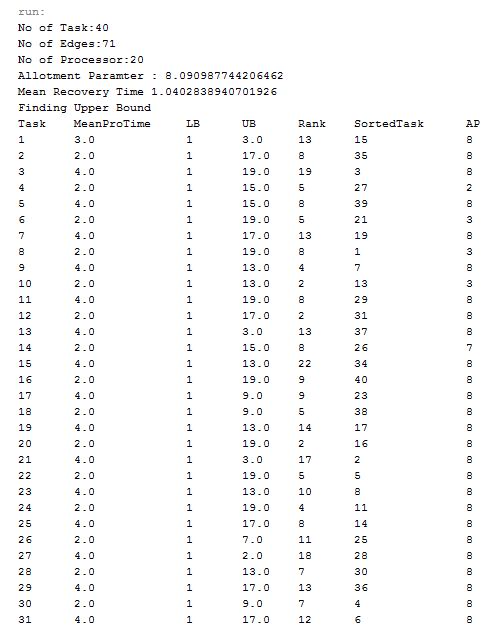
return pred;

}

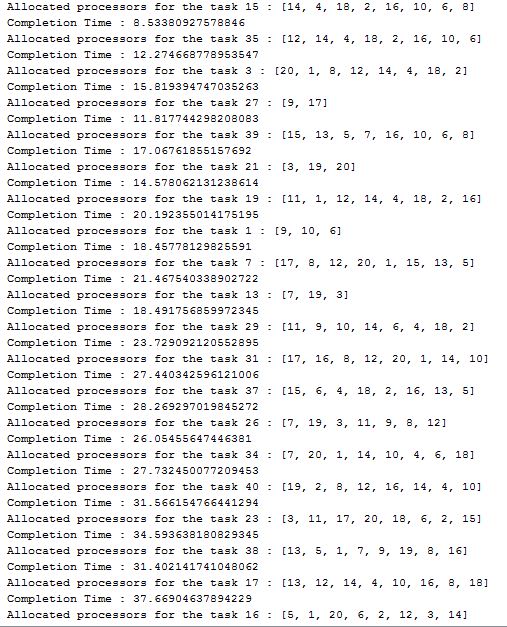
}

* 1. **SCREENSHOT**

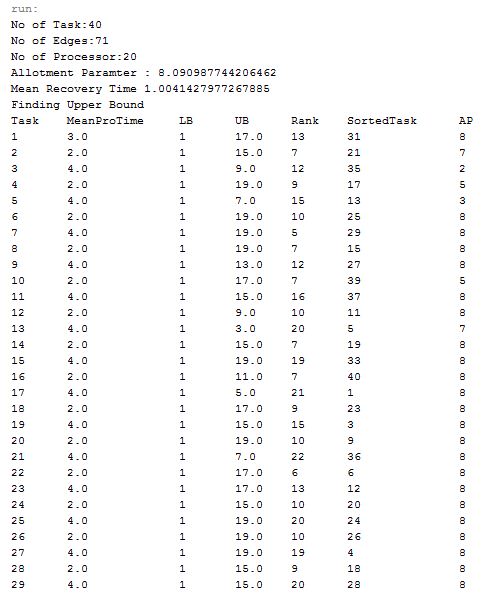
**For Strategy I:**

******

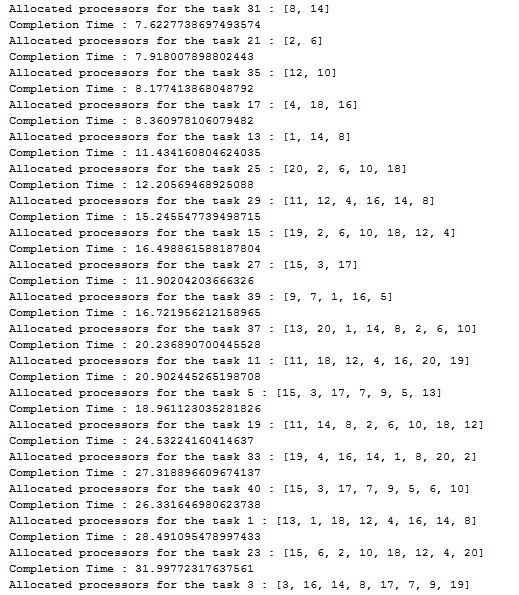
**For Strategy I:**

******

**For Strategy II:**

****

**For Strategy II:**

****

1. **RESULTS AND DISCUSSIONS**
   1. **PERFORMANCE ANALYSIS**

***Sliding Window Impact on the Execution Time***

Execution time is directly correlated to scalability and scalability is a major contribution of this work. It has been achieved through

1. the adoption of a sliding window and
2. the reduction of the number of comparisons via utilizing a group based comparison.

To demonstrate scalability, first we have analyzed the execution time performance of the FBP algorithm with different window sizes for 60 days. This time frame has been utilized as it covers a significant period of time so that the capability of the FBP algorithm can be fully tested. Dealing with a large number of days is an important requirement in Lifelogging systems, which use small devices.

The graph summarizes these performance changes for both the UbiqLog datasets. The legend on the right side in the graph shows the window size. Since weekend behaviors are different than weekday behaviors, we recommend to compare them separately from the weekday data. In particular, we recommend not using a window size larger than five or six days. However, this depends on the weekend duration, i.e., if weekends are two days or one day. Therefore, the upper bound could be the number of the weekdays.

The results illustrate that increasing the window size significantly improves the execution time performance. In other words, a smaller slope means better performance, and increasing the window size decreases the slope. Even increasing the number of days, does not affect the performance of the FBP algorithm. The results depicted in the graph belongs to one user, for 60 days. Another factor that affects the scalability is the use of grouping instead of simple comparisons. FBPs are designed for multivariate temporal data. Most of the similar algorithms to FBP are frequent item set mining algorithms.

***Comparison with Other Algorithms:***

We have compared the FBP execution time, memory and battery utilization of a sample of user data. The minimum support for all algorithms have been set to two. λ, have been set to two and a window size of three for FBP has been used. The graph shows the execution time of running FBP in comparison to other algorithms on the smartphone, within the described settings. For more than six days of analysis, FBP execution time performance is faster than other algorithms. Especially as the number of days increases, the FBP execution time does not change significantly and stays at a near constant value. However, for a smaller amount of data, FBP does not perform better than MTK or estDec+.

CONCLUSION

**CONCLUSION AND FUTURE WORK**

In our project, we have used a scalable approach for daily behavioral pattern mining from multiple sensor information. This work has been benefited from a real-world dataset. We use a novel temporal granularity transformation algorithm that makes changes on timestamps to mirror the human perception of time. Our frequent behavioral pattern detection approach is generic and not dependent on a single source of information; therefore, we have reduced the risk of uncertainty by relying on a combination of information sources to identify frequent behavioral patterns. Furthermore, our approach is lightweight enough that it can be run on small devices and thus reduces the network and privacy cost of sending data to the cloud. Results of the experimental evaluation shows our algorithm outperforms both execution time and accuracy. Moreover, converting raw timestamps to temporal granularities increase the accuracy of the FBP identification, which is influenced by different values of temporal granularity, the segment of the day and the sensor type. These findings assist the system in identifying the appropriate run time and sensor impact of the behavioral pattern identification.

In our future work, we are trying to model concept drift and its relation with forgetting or churn that is in the nature of human behavior. Moreover, we plan to compare the performance of the sliding window with the performance of the damped window.

REFERENCES

**REFERENCES**

* H. Ma, H. Cao, Q. Yang, E. Chen, and J. Tian, “A habit mining approach for discovering similar mobile users,” in Proc. 21st Int. Conf. World Wide Web, 2012, pp. 231–240.
* T. Bhattacharya, L. Kulik, and J. Bailey, “Automatically recogniz-ing places of interest from unreliable GPS data using spatio-tem-poral density estimation and line intersections,” Pervasive Mobile Comput., vol. 19, pp. 86–107, 2015.
* V. Srinivasan, S. Moghaddam, A. Mukherji, K. K. Rachuri, C. Xu, and E. M. Tapia, “Mobileminer: Mining your frequent patterns on your phone,” in Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput., 2014, pp. 389–400.
* Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie, “Mining individual life pattern based on location history,” in Proc. 10th Int. Conf. Mobile Data Manage.: Syst. Serv. Middleware, 2009, pp. 1–10.
* F. Morchen€ and A. Ultsch, “Efficient mining of understandable patterns from multivariate interval time series,” Data Min. Knowl. Discovery, vol. 15, no. 2, pp. 181–215, 2007.
* C. Lucchese, S. Orlando, and R. Perego, “Fast and memory effi-cient mining of frequent closed itemsets,” IEEE Trans. Knowl. Data Eng., vol. 18, no. 1, pp. 21–36, Jan. 2006.
* H. Tsai, D.-N. Yang, and M.-S. Chen, “Mining group movement patterns for tracking moving objects efficiently,” IEEE Trans. Knowl. Data Eng., vol. 23, no. 2, pp. 266–281, Feb. 2011.
* J. Lee, J. Han, X. Li, and H. Cheng, “Mining discriminative pat-terns for classifying trajectories on road networks,” IEEE Trans. Knowl. Data Eng., vol. 23, no. 5, pp. 713–726, May 2011.