Simulation Four Component Series - Parallel System

K. Uma Maheswari[#1], A. Mallikarjuna Reddy[#1] and R. Bhuvana Vijaya[#2]
[#1]Department of Mathematics, Sri Krishnadevaraya University, Ananthapuramu-515003, A.P., India
[#2]Department Mathematics, J.N.T.U., Ananthapuramu-515001, A.P., India

Abstract :
The Monte-Carlo method involves building of a system using a computer program of a probabilistic model of the system under consideration. A trial run of the model is repeated many times and use of the data to estimating the various performance measures of the system. This paper presents the advantages, applicability, usage of Monte-carlo method to derive the system performance of four component series-parallel system, when normal methods fail to analyse the reliability and its applicability.

Keywords : Monte-Carlo method, Simulation, Reliability, availability, Series-parallel system.

INTRODUCTION

The SYSTEM SIMULATION is defined as the technique of solving problems by the observation of the performance, over time, of a dynamic model of the system. It refers to the reconstruction of a simplified representation of a system (or process) in order to facilitate its analysis.

In this chapter we discuss briefly Monte-Carlo simulation approach, its advantages, applicability and usage of this method to derive the system performance of a 4 component Series - Parallel system.

MONTE-CARLO METHOD (Simulation of Random Events)

One of the most important aspects of digital simulation is the ability to handle stochastic events, that is, events which occur randomly. The technique for solving such problems is often known as the Monte-Carlo Method.

The Monte-Carlo method involves building of a system, using a computer program of a probabilistic model of the system under consideration. A trial run of the model is repeated many times and use of the data to estimating the various performance measures of the system.

For the execution of the Monte-Carlo method, random numbers between 0 and 1 are generated. The random variable X is then obtained by treating the random numbers with some procedure which is appropriate to the nature of the distribution function.

Advantages of Monte-Carlo simulation

(i) There are no restrictions on the failure, repair and other time distribution.
(ii) Dependent relations between failure, repair etc., events can be accounted for
(iii) The analytical background is simple.

Applicability of the Monte-Carlo method

The following few characteristics of a problem suggest the use of the Monte-Carlo method. They are :

(i) The problem is extremely complex.
(ii) Experimentation is desirable but costly.
(iii) The system cannot be stopped for study or it is not warranted do disturb the operation of the system.
(iv) It may be hazardous to operate the system in actual field conditions.

Conditions for the Application of Monte-Carlo Method

A few of the necessary conditions for the application of the Monte-Carlo Method are given below :

(i) There is no exact mathematical solution to the problem, and hence no exact solution can be arrived at.

(ii) One or more input parameters do not have a fixed value, but can only be represented by a probability distribution.

(iii) Sufficient information is not available to estimate accurately the distributions of the input parameters.

Series-parallel systems are made up of combination several series & parallel configuration to obtain system reliability down into homogeneous subsystems. Then consider each of the subsystem separately as a unit and calculate their reliabilities to obtain total enhanced system reliability. In this present paper we have considered 4 - component series-parallel complex system.

SIMULATION OF SERIES-PARALLEL SYSTEM

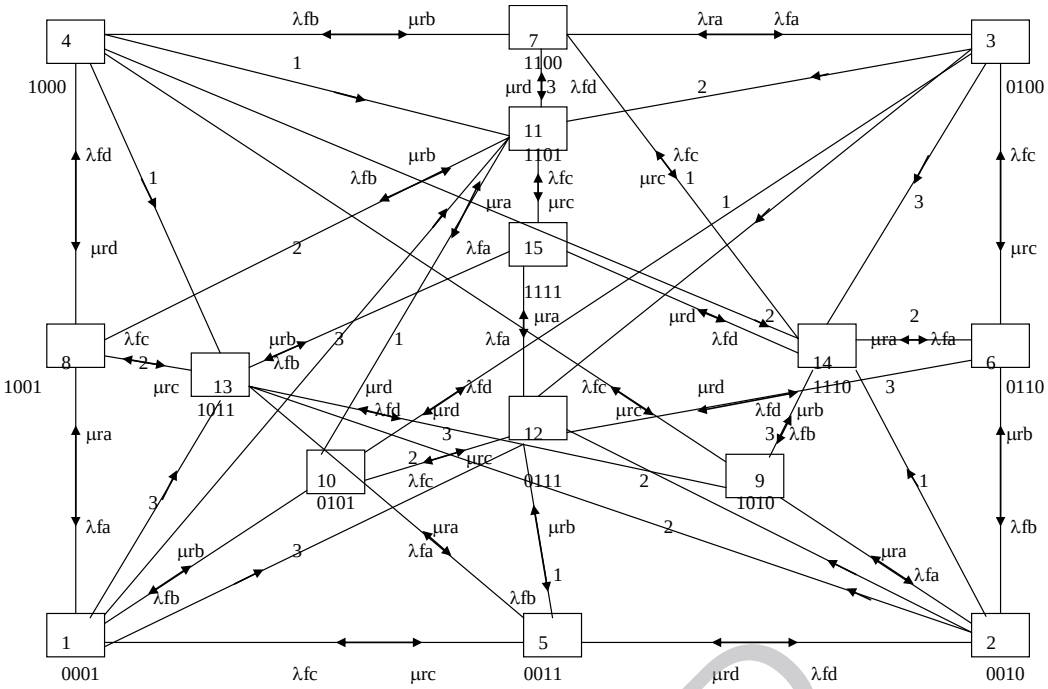
We use the simulation technique to simulate a three component series-parallel system consisting of the four components A, B, C and D. This system is in functioning state if A is in functioning state and either B or C or Dare in functioning state. We assume that all the four components have repair facility.

The upstate of a component is represented by 1 and the down state is represented by 0. Hence every state of the system is represented by the triplet a, b and c where a=0 or 1, b=0 or 1, c=0 or 1, d=0 or 1. It is further observed that the state (0, 0, 0, 0) cannot occur. The state (a, b, c, d) of the system is represented by the decimal number corresponding to the binary number a b c d. Hence we have

State	1	→	(0, 0, 0, 1)
State	2	→	(0, 0, 1, 1)
State	3	→	(0, 0, 1, 0)
State	4	→	(0, 1, 1, 0)
State	5	→	(0, 1, 0, 0)
State	6	→	(1, 1, 0, 0)
State	7	→	(1, 0, 0, 0)
State	8	→	(1, 0, 0, 1)
State	9	→	(1, 0, 1, 0)
State	10	→	(0, 1, 0, 1)
State	11	→	(1, 1, 0, 1)
State	12	→	(0, 1, 1, 1)
State	13	→	(1, 0, 1, 1)
State	14	→	(1, 1, 1, 0)
State	15	→	(1, 1, 1, 1)

The transition diagram of the system is given in Fig 1.

We observe that in the series-parallel system under consideration the states 1, 2, 3, 4 are failed states of the system and states 5, 6, 7 to 15 are the upstates of the system. We use the event driven simulation to find the total average up-time of the system. We define a cycle of the system to be the path of the system from state 15 to state 15 again.



NOTATION

λfa	-	Failure rate of the system A
λfb	-	Failure rate of the system B
λfc	-	Failure rate of the system C
λfd	-	Failure rate of the system D
μra	-	Repair rate of the system A
μrb	-	Repair rate of the system B
μrc	-	Repair rate of the system C
μrd	-	Repair rate of the system D

Algorithm

Main Algorithm

Step 1 : Initialize Mean Failure values Fa, Fb, Fc, Fd
Step 2 : Initialize Repair Rate values Ra, Rb, Rc, Rd
Step 3 : Read lfa, lfb, lfc, lfd, ura, urb, urc, urd, No. of cycles, No of Simulations
Step 4 : Initialize SimNum = 1
Step 5 : Repeat Step 5 through step 14 until SimNum <= No. of Simulations.
Step 6 : call initialize()
Step 7 : cycleNum = 1
Step 8 : Repeat step 9 through step 11 until cycle num = No of cycles
Step 9 : call timing
Step 10 : call update_states
Step 11 : if sys_state = 1 then

```
        Call c_up()
    Else if sys_state = 2 then
        Call b_up()
     Else if sys_state = 3 then
        Call bc_up()
     Else if sys_state = 4 then
        Call a_up()
```

```
Else if sys_state = 5 then
    Call ac_up()
Else if sys_state = 6 then
    Call ab_up()
Else if sys_state = 7 then
    Call abc_up()
Else if sys_state = 8 then
    Call cd_up()
Else if sys_state = 9 then
    Call bc_up()
Else if sys_state = 10 then
    Call bcd_up()
Else if sys_state = 11 then
    Call d_up()
Else if sys_state = 12 then
    Call acd_up()
Else if sys_state = 13 then
    Call abd_up()
Else if sys_state = 14 then
    Call abcd_up()
```

Step 12 : tot_up_time = tot_up_time + up
Step 13 : tot_down_time = tot_down_time + down
Step 14 : tot_duration = tot_duration + s_time
Step 15 : call report()
Step 16 : close all
Step 17 : Exit

Initialize Sub – Algorithm

Step 1 : sys_state = 14; s_time = 0; cycle_num = 1;
Step 2 : repeat step 3 while I <= 6
Step 3 : time_next_event[i] = infinity
Step 4 : time_next_event[fa] = expn(mfa)
time_next_event[fb] = expn(mfb)
time_next_event[fc] = expn(mfc)
Step 5 : down = 0; up = 0; time_last_event = 0;
Step 6 : call randomize
Step 7 : return

Timing Sub – Algorithm

Step 1 : min_time_next_event = infinity
Step 2 : next_event_type = 0; I = 1;
Step 3 : repeat step 4 while I <= num_events;
Step 4 : if (time_next_event[i] < min_time_next_event)
Min_time_next_event[fb] = time_next_event[i]
next_event_type = i
Step 5 : sys_state = next_state[sys_state-1][next_event_type-1]
Step 6 : s_time = min_time_next_event
Step 7 : if (sys_state = 0) exit;
Step 8 : return

Update_state Sub – Algorithm

Step 1 : time_since_last_event = s_time – time_last_event
Step 2 : time_last_event = s_time

Step 3 : if (sys_state > 4) then
Tp = 1
Step 4 : up = up + tp * time_since_last_event
Step 5 : down = down + (1 – tp) * time_since_last_event;
Step 6 : return

a_up Sub – Algorithm

Step 1 : time_next_event[FB] = INFINITY;
Step 2 : time_next_event[FC] = INFINITY;
Step 3 : time_next_event[FD] = INFINITY;
Step 4 : time_next_event[RA] = INFINITY;
Step 5 : if(rfac == 2) then
time_next_event[FA] += time_next_event[RB] - s_time;
else if(rfac == 3) then

time_next_event[FA] += time_next_event[RC] - s_time;

b_up Sub – Algorithm

Step 1 : time_next_event[FA] = INFINITY;
Step 2 : time_next_event[FB] += time_next_event[RC] - s_time;

c_up Sub – Algorithm

Step 1 : time_next_event[FA] = INFINITY;
Step 2 : time_next_event[FC] += time_next_event[RD] - s_time;

d_up Sub – Algorithm

Step 1 : time_next_event[FA] = INFINITY;
Step 2 : time_next_event[FD] += time_next_event[RC] - s_time;

ab_up Sub – Algorithm

Step 1 : rtc = expn(mrc);
Step 2 : if(rfac == 2) then
time_next_event[FA] = s_time - time_last_event;
time_next_event[FB] = s_time + Math.exp(lfb);
Step 3 : time_next_event[FC] = INFINITY;
Step 4 : time_next_event[RA] = INFINITY;
Step 5 : time_next_event[RB] = INFINITY;
Step 6 : time_next_event[RC] = rtc + s_time;
Step 7 : rfac = 3;

ac_up Sub – Algorithm

Step 1 : rtb = expn(mrb);
Step 2 : if(rfac == 3) then
time_next_event[FA] = s_time - time_last_event;
time_next_event[FC] = s_time + expn(lfc);
Step 3 : time_next_event[FB] = INFINITY;
Step 4 : time_next_event[RA] = INFINITY;
Step 5 : time_next_event[RB] = rtb + s_time;
Step 6 : time_next_event[RC] = INFINITY;
Step 7 : rfac = 2;

ad_up Sub – Algorithm

Step 1 : rtb = expn(mrb);
Step 2 : if(rfac == 3) then
time_next_event[FA] = s_time - time_last_event;
time_next_event[FC] = s_time + Math.exp(lfc);
Step 3 : time_next_event[FB] = INFINITY;

```
Step 4    :    time_next_event[RA] = INFINITY;
Step 5    :    time_next_event[RB] = rtb + s_time;
Step 6    :    time_next_event[RC] = INFINITY;
Step 7    :    rfac = 2;
```

bc_up Sub – Algorithm

```
Step 1    :    rta = expn(mra);
Step 2    :    time_next_event[FA] = INFINITY;
Step 3    :    time_next_event[FB] += rta;
Step 4    :    time_next_event[FC] += rta;
Step 5    :    time_next_event[RA] = rta + s_time;
Step 6    :    time_next_event[RB] = INFINITY;
Step 7    :    time_next_event[RC] = INFINITY;
Step 8    :    if(rfac == 2)
               time_next_event[FB] = time_next_event[RA] + Math.exp(lfb);
               else if(rfac == 3)
               time_next_event[FC] = time_next_event[RA]+ Math.exp(lfc);
               rfac = 1;
```

bd_up Sub – Algorithm

```
Step 1    :    rta = Math.exp(mra);
Step 2    :    time_next_event[FA] = INFINITY;
Step 3    :    time_next_event[FB] += rta;
Step 4    :    time_next_event[FC] += rta;
Step 5    :    time_next_event[RA] = rta + s_time;
Step 6    :    time_next_event[RB] = INFINITY;
Step 7    :    time_next_event[RC] = INFINITY;
Step 8    :    if(rfac == 2)
               time_next_event[FB] = time_next_event[RA] + Math.exp(lfb);
               else if(rfac == 3)
               time_next_event[FC] = time_next_event[RA]+ Math.exp(lfc);
Step 9    :    rfac = 1;
```

cd_up Sub – Algorithm

```
Step 1    :    rta = expn(mra);
Step 2    :    time_next_event[FA] = INFINITY;
Step 3    :    time_next_event[FB] += rta;
Step 4    :    time_next_event[FC] += rta;
Step 5    :    time_next_event[RA] = rta + s_time;
Step 6    :    time_next_event[RB] = INFINITY;
Step 7    :    time_next_event[RC] = INFINITY;
Step 8    :    if(rfac == 2)
               time_next_event[FB] = time_next_event[RA] + Math.exp(lfb);
               else if(rfac == 3)
               time_next_event[FC] = time_next_event[RA]+ Math.exp(lfc);
Step 9    :    rfac = 1;
```

abc_up Sub – Algorithm

```
Step 1    :    if(rfac == 1){
Step 2    :    time_next_event[FA] = s_time + Math.exp(lfa);
Step 3    :    time_next_event[RA] = INFINITY;
Step 4    :    ++cycle_Number;
Step 5    :    if(rfac == 2) then
               time_next_event[FB] = s_time + Math.exp(lfb);
```

```
                time_next_event[RB] = INFINITY;
                ++cycle_Number;
Step 6    :     if(rfac == 3) then
                time_next_event[FC] = s_time + Math.exp(lfc);
                time_next_event[RC] = INFINITY;
                ++cycle_Number;
Step 7    :     rfac = 0;
```

abd_up Sub – Algorithm

```
Step 1    :     if(rfac == 1) then
                time_next_event[FA] = s_time + Math.exp(lfa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number;
Step 2    :     if(rfac == 2) then
                time_next_event[FB] = s_time + Math.exp(lfb);
                time_next_event[RB] = INFINITY;
                ++cycle_Number;
Step 3    :     if(rfac == 3) then
                time_next_event[FC] = s_time + Math.exp(lfc);
                time_next_event[RC] = INFINITY;
                ++cycle_Number;
Step 4    :     rfac = 0;
```

acd_up Sub – Algorithm

```
Step 1    :     if(rfac == 1) then
                time_next_event[FA] = s_time + Math.exp(lfa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number;
Step 2    :     if(rfac == 2) then
                time_next_event[FB] = s_time + Math.exp(lfb);
                time_next_event[RB] = INFINITY;
                ++cycle_Number;
Step 3    :     if(rfac == 3)
                time_next_event[FC] = s_time + Math.exp(lfc);
                time_next_event[RC] = INFINITY;
                ++cycle_Number;
Step 4    :     rfac = 0;
```

bcd_up Sub – Algorithm

```
Step 1    :     if(rfac == 1) then
                time_next_event[FA] = s_time + Math.exp(lfa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number;
Step 2    :     if(rfac == 2) then
                time_next_event[FB] = s_time + Math.exp(lfb);
                time_next_event[RB] = INFINITY;
                ++cycle_Number;
Step 3    :     if(rfac == 3) then
                time_next_event[FC] = s_time + Math.exp(lfc);
                time_next_event[RC] = INFINITY;
                ++cycle_Number;
Step 4    :     rfac = 0;
```

abcd_up Sub – Algorithm

Step 1 : if(rfac == 1) then
time_next_event[FA] = s_time + Math.exp(lfa);
time_next_event[RA] = INFINITY;
++cycle_Number;
Step 2 : if(rfac == 2) then
time_next_event[FB] = s_time + Math.exp(lfb);
time_next_event[RB] = INFINITY;
++cycle_Number;
Step 3 : if(rfac == 3) then
time_next_event[FC] = s_time + Math.exp(lfc);
time_next_event[RC] = INFINITY;
++cycle_Number;
Step 4 : rfac = 0;

report Sub – Algorithm

Step 1 : print average up_time
Step 2 : print average down_time
Step 3 : print average duration of simulation
Step 4 : return

A computer program is developed to simulate a 4 unit series-parallel system. This program will gives the system uptime and downtimes for (i) Randomly generated failure and repair rates and (ii) for constant values of failure and repair rates of components.

SIMULATION PROGRAM IN C-LANGUAGE FOR THREE COMPONENT SERIES-PARALLEL SYSTEM

This program is developed to simulate the Availability and Non-availability of a THREE UNIT SERIES - PARALLEL SYSTEm.

INPUT: (1) The number of random values
(2) The number of simulations required
This program randomly generated the value for failure and repair rates of the components A, B and C respectively as λ_{fa}, λ_{fb}, λ_{fc}, λ_{fd} and μ_{ra}, μ_{rb}, μ_{rc}, μ_{rd} and estimates the availability and non-availability of the system.

OUTPUT: (1) Uptime
(2) Downtime

```
* Simulation of 3-unit series - parallel common cause system
Important java.util.Scanner;
public class ParallelSystemFour{
        final int FA = 1;
        final int FB = 2;
        final int FC = 3;
        final int FD = 4;
        final int RA = 5;
        final int RB = 6;
        final int RC = 7;
        final int RD = 8;
        final double INFINITY = 1.8*Math.exp(308);
        double Ifa=0.0, Ifb=0.0, Ifc=0.0, Ifd=0.0; /* failure rates */
```

```
        double mra=0.0, mrb=0.0, mrc=0.0, mrd=0.0; /* repair rates */
        int num_Of_Cycles=9; /* a cycle is a visit to state 7 */
        int num_Of_Simulations=9; /* Number of Simulations for given Parmeter*/
        int simulation_Number, cycle_Number,next_Event_Type;
        int rfac, sys_state, num_events;
        int seed;
        double time_next_event[] = new double[14];
        double s_time, time_last_event, down, up;
        double tot_up_time=1, tot_down_time=1, tot_duration=1;
        int next_state[][]={
                {0,0,0,0,3,0,0,0,0,0,3,0,0},
                {0,0,0,0,0,3,0,0,0,0,0,0,3},
                {0,0,0,7,0,0,0,0,0,7,0,0,0},
                {0,0,0,0,6,5,0,0,0,0,0,6,5},
                {1,0,4,0,7,0,1,0,4,0,7,0,0},
                {2,4,0,0,0,7,2,4,0,0,0,7,0},
                {3,5,6,0,0,0,3,5,6,0,0,0,0},
                {0,0,0,0,3,0,0,0,0,0,3,0,0},
                {0,0,0,0,0,3,0,0,0,0,0,0,3},
                {0,0,0,7,0,0,0,0,0,7,0,0,0},
                {0,0,0,0,6,5,0,0,0,0,0,6,5},
                {1,0,4,0,7,0,1,0,4,0,7,0,0},
                {2,4,0,0,0,7,2,4,0,0,0,7,0},
                {3,5,6,0,0,0,3,5,6,0,0,0,0}};
public void mymain(String[] args){
        num_events=6;
        System.out.printh("Give an Integer Seed :");
        Scanner scanner = new Scanner(System.in);
        seed = scanner.nextInt();
        System.out.printIn("FOUR COMPONENT SERIES PARALLEL
                SYSTEM");
        System.out.printIn("COMP-A IS IN SERIES WITH COMP-B, COMP-C AND
                COMP-D WHICH ARE PARALLEL");
        System.out.printIn(MEAN FAILURE TIMES OF A, B, C D ARE : "+Ifa+" "+Ifb+"
                "+Ifc" "+Ifd);
        System.out.printIn("MEAN REPAIR TIMES OF A, B, C, D ARE : "+mra+"
                "+mrb+" "+mrc+" "+mrd);
        System.out.printIn("Desired Number of Visits to State-14 is : "+num_Of_Cycles);
        System.out.printIn("Desired Number of Simulations for given set of Parameters is :
                "+num_Of_Simulations);
        for(simulation_Number = 1; simulation_Number<=
        num_Of_Simulations;++simulation_Number){
                //System.out.printIn("  ");
                        initialize();
                        while(cycle_Number<=num_Of_Cycles){
                                update_stats();
                                switch(sys_state){
                                case 1: c_up();break;
                                case 2: b_up();break;
                                case 3: bc_up();break;
```

```
                                case 4: a_up();break;
                                case 5: ac_up();break;
                                case 6: ab_up();break;
                                case 7: bac_up();break;
                                case 8: cd_up();break;
                                case 9: bc_up();break;
                                case 10: bcd_up();break;
                                case 11: d_up();break;
                                case 12: acd_up();break;
                                case 13: abd_up();break;
                                case 14: abcd_up();break;
                                }
                        }
                        tot_up_time += up;
                        tot_down_time += down;
                        tot_duration += s_times;
                        report();
                }
        }
        public static void main(String[] args) throws FileNotFoundExcepion {
                ParallelSystemFour ps = new ParallelSystemFour();
                ps.mymain(args);
                ps.report();
        }
        void report(){
                int num_Of_Simulations=1;
                double tot_up_time=1;
                System.out.printIn("The Average Uptime is : "+(tot_up_time /
num_Of_Simulations));
                double tot_down_time=1;
                System.out.printIn("The Average Downtime is : "+(tot_down_time /
num_Of_Simulations));
                double tot_duration=1;
                System.out.printIn("The Average Duration of Simulations is :"+(tot_duration /
        }
        void initialize (){
                int i;
                s_time = 0;
                cycle_Number = 1;
                sys_state = 14;
                rfac = 0;
                time_next_event[FA] = Math.exp(Ifa);
                time_next_event[FB] = Math.exp(Ifb);
                time_next_event[FC] = Math.exp(Ifc);
                time_next_event[FD] = Math.exp(Ifd);
                down = 0;
                time_last_event = 0;
                for(i=4;i<=6;i++)
                        time_next_event[i] = INFINITY;
        }
```

```
void timing (){
        int i;
        double min_time_next_event = INFINITY;
        next_Event_Type = 0;
        for(i=1;i<=num_events;++i)
                if(time_next_event[i] < min_time_next_event){
                        min_time_next_event = time_next_event[i];
                        next_Event_Type = i;
                }
        sys_state = next_state[sys_state-1][next_Event_Type-1];
        s_time = min_time_next_event;
        System.out.printIn("System State : "+sys_state+"\nSystem Time: "+s_time_;
        if(sys_state == 0) System.exit(1);
}
void a_up (){
        time_next_event[FB] = INFINITY;
        time_next_event[FC] = INFINITY;
        time_next_event[FD] = INFINITY;
        time_next_event[RA] = INFINITY;
        if(rfac == 2)
                time_next_event[FA] += time_next_event[RB] - s_time;
        else if(rfac == 3)
                time_next_event[FA] += time_next_event[RC] - s_time;
}
void b_up (){
        time_next_event[FA] = INFINITY;
        time_next_event[FB] += time_next-event[RC] - s_time;
}
void b_up (){
        time_next_event[FA] = INFINITY;
        time_next_event[FC] += time_next_event[RD] - s_time;
}
void d_up(){
        time_next_event[FA] = INFINITY;
        time_next_event[FD] += time_next_event[RC] - s_time;
}
void ab_up (){
        double rtc;
        rtc = Math.exp(mrc);
        if(rfac == 2){
                time_next_event[FA] = s_time - time_last_event;
                time_next_event[FB] = s_time + Math.exp(Ifb);
        }
        time_next_event[FC] = INFINITY;
        time_next_event[RA] = INFINITY;
        time_next_event[RB] = INFINITY;
        time_next_event[RC] = rtc + s_time;
        rfac = 3;
}
void ac_up (){
```

```
        double rtb;
        rtb = Math.exp(mrb);
        if(rfac == 3){
                time_next_event[FA] = s_time - time_last_event;
                time_next_event[FC] = s_time + Math.exp(Ifc);
        }
        time_next_event[FB] = INFINITY;
        time_next_event[RA] = INFINITY;
        time_next_event[RB] = rtb + s_time;
        time_next_event[RC] = INFINITY;
        rfac = 2;
}
void ad_up(){
        double rtb;
        rtb = Math.exp(mrb);
        if(rfac == 3){
                time_next_event[FA] = s_time - time_last_event;
                time_next_event[FC] = s_time + Math.exp(Ifc);
        }
        time_next_event[FB] = INFINITY;
        time_next_event[RA] = INFINITY;
        time_next_event[RB] = rtb + s_time;
        time_next_event[RC] = INFINITY;
        rfac = 2;
}
void bc_up (){
        double rta;
        rta = Math.exp(mra);
        time_next_event[FA] = INFINITY;
        time_next_event[FB] += rta;
        time_next_event[FC] += rta;
        time_next_event[RA] = rta + s_time;
        time_next_event[RB] = INFINITY;
        time_next_event[RC] = INFINITY;
        if(rfac == 2)
                time_next_event[FB] = time_next_event[RA] + Math.exp(Ifb);
        else if(rfac == 3)
                time_next_event[FC] = time_next_event[RA]+ Math.exp(Ifc);
        rfac = 1;
}
void bd_up(){
        double rta;
        rta = Math.exp(mra);
        time_next_event[FA] = INFINITY
        time_next_event[FB] += rta;
        time_next_event[FC] += rta;
        time_next_event[RA] = rta + s_time;
        time_next_event[RB] = INFINITY;
        time_next_event[RC] = INFINITY;
        if(rfac == 2)
```

```
                time_next_event[FB] = time_next_event[RA] + Math.exp(Ifb);
        else if(rfac == 3)
                time_next_event[FC] = time_next_event[RA]+ Math.exp(Ifc);
        rfac = 1;
}
void cd_up(){
        double rta;
        rta = Math.exp(mra);
        time_next_event[FA] = INFINITY;
        time_next_event[FB] += rta;
        time_next_event[FC] += rta;
        time_next_event[RA] = rta + s_time;
        time_next_event[RB] = INFINITY;
        time_next_event[RC] = INFINITY;
        if(rfac == 2)
                time_next_event[FB] = time_next_event[RA] + Math.exp(Ifb);
        else if(rfac == 3)
                time_next_event[FC] = time_next_event[RA]+ Math.exp(Ifc);
        rfac = 1;
}
void abc_up (){
        if(rfac == 1){
                time_next_event[FA] = s_time + Math.exp(Ifa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number
        }
        if(rfac == 2){
                time_next_event[FB] = s_time + Math.exp(Ifb);
                time_next_event[RB] = INFINITY
                ++cycle_Number:
        }
        if(rfac == 3){
                time_next_event[FC] = s_time + Math.exp(Ifc);
                time_next_event[RC] = INFINITY
                ++cycle_Number:
        }
        rfac = 0;
}
void abd_up(){
        if(rfac == 1){
                time_next_event[FA] = s_time + Math.exp(Ifa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number
}
        if(rfac == 2){
                time_next_event[FB] = s_time + Math.exp(Ifb);
                time_next_event[RB] = INFINITY;
                ++cycle_Number
}
        if(rfac == 3){
```

```
                time_next_event[FC] = s_time + Math.exp(Ifc);
                time_next_event[RC] = INFINITY;
                ++cycle_Number
        }
        rfac = 0;
}
void acd_up (){
        if(rfac == 1){
                time_next_event[FA] = s_time + Math.exp(Ifa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number
        }
        if(rfac == 2){
                time_next_event[FB] = s_time + Math.exp(Ifb);
                time_next_event[RB] = INFINITY
                ++cycle_Number:
        }
        if(rfac == 3){
                time_next_event[FC] = s_time + Math.exp(Ifc);
                time_next_event[RC] = INFINITY
                ++cycle_Number:
        }
        rfac = 0;
}
void bcd_up (){
        if(rfac == 1){
                time_next_event[FA] = s_time + Math.exp(Ifa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number
        }
        if(rfac == 2){
                time_next_event[FB] = s_time + Math.exp(Ifb);
                time_next_event[RB] = INFINITY
                ++cycle_Number:
        }
        if(rfac == 3){
                time_next_event[FC] = s_time + Math.exp(Ifc);
                time_next_event[RC] = INFINITY
                ++cycle_Number:
        }
        rfac = 0;
}
void abcd_up (){
        if(rfac == 1){
                time_next_event[FA] = s_time + Math.exp(Ifa);
                time_next_event[RA] = INFINITY;
                ++cycle_Number
        }
        if(rfac == 2){
                time_next_event[FB] = s_time + Math.exp(Ifb);
```

```
                time_next_event[RB] = INFINITY
                ++cycle_Number:
            }
            if(rfac == 3){
                time_next_event[FC] = s_time + Math.exp(Ifc);
                time_next_event[RC] = INFINITY
                ++cycle_Number:
            }
            rfac = 0;
        }
        void update_stats(){
            double time_since_last_event;
            int tp = 0;
            time_since_last_event = s_time_last_event;
            time_last_event = s_time;
            if(sys_state>4)
                tp = 1;
            up += tp * time_since_last_event;
            down += (1-tp) * time_since_last_event;
        }
}
```

Output:

```
Give an Integer Seed :
9
FOUR COMPONENT SERIES PARALLEL SYSTEM
COMP-A IS IN SERIES WITH COMP-B, COMP-C AND COMP-D WHICH ARE PARALLEL
MEAN FAILURE TIMES OF A, B, C, D ARE : 0.0 0.0 0.0 0.0
MEAN REPAIR TIMES OF A, B, C, D ARE : 0.0 0.0 0.0 0.0
Desired Number of Visits to State-14 is : 9
Desired Numbers of Simulations for given set of Parameters is : 9
System State  :  3
System Time  :  1.0
System State  :  0
System Time  :  2.0
```


DISCUSSION :

A Computer program is developed to stimulate a four unit series - parallel system.

(i) Rendamly generated failure and repair rates and

(ii) For constant values of failure and repair rates of components.

The Simulated results obtained from the computer program for the failure and repair rates of the components given in Table 1.

Table 1.

λ_{fa}	λ_{fb}	λ_{fc}	λ_{fd}	μ_{ra}	μ_{rb}	μ_{rc}	μ_{rd}	The Average Duration of Simulation	Average Uptime	Availability
0.692784	0.556705	0.812395	0.851852	0.221607	0.792131	0.0472040	0.289625	95.131345	9.4125256	0.634434
0.655376	0.360995	0.479307	0.514228	0.638810	0.786739	0.758017	0.118555	76.109804	47.514521	0.178031
0.884222	0.802381	0.540160	0.0693079	0.848932	0.658042	0.150208	0.209807	89.945493	44.898984	0.805812
0.755441	0.716152	0.265209	0.116594	0.639526	0.374069	0.131418	0.157697	96.912649	58.355036	0.0424920
0.805991	0.464316	0.186135	0.320839	0.319412	0.171644	0.126033	0.708177	62.938312	56.288909	0.178136
0.288041	0.462151	0.791332	0.692224	0.630895	0.669407	0.733631	0.351592	97.951007	53.324199	0.290972
0.297912	0.319746	0.597853	0.0507482	0.630022	0.363428	0.111083	0.495483	85.399780	61.390966	0.249805
0.540652	0.807702	0.0451773	0.800416	0.0296949	0.806131	0.783019	0.763705	38.436943	0.79620582	0.332997
0.320759	0.754307	0.113158	0.193514	0.397383	0.117159	0.683347	0.460267	10.938689	4.2096429	0.767678
0.383330	0.117236	0.473711	0.837246	0.111175	0.411662	0.424673	0.422830	89.953161	53.251145	0.549746
0.627012	0.463848	0.255790	0.0829527	0.0324720	0.821117	0.408577	0.722447	32.149109	67.394493	0.895018
0.543780	0.427870	0.751562	0.609079	0.244882	0.709081	0.805442	0.644222	7.1988911	10.657514	0.162678
0.159870	0.530730	0.199883	0.0274201	0.150338	0.578895	0.434260	0.553844	31.122232	84.791140	0.183901
0.546570	0.767342	0.612597	0.683721	0.278825	0.668880	0.624886	0.283311	31.968505	49.330740	0.783725
0.0634206	0.235757	0.151849	0.872448	0.461612	0.585635	0.768374	0.113313	62.107657	46.824626	0.0911289

REFERENCES :

[1] Reliability Engineering http://d577289.436. website source net/articles/reliability Eng-Part 12.pdf.

[2] http:// www.m-hikari.com/ams/ams-2012/ams-73-76, 2012/elfaheem AMS, 73-76-2012.pdf

[3] Mathworld.wolfram.com/monte.carlo method.html

[4] Monte Carlo Methods www.cs.nyu.edu/courses/fall 06/G22.2112-001/Monte Carlo.pdf

[5] Monte Carlo simulation, www.goldsim.com/web/introduction/probabilistic/Monte Carlo/