Enhanced Markov-based model for the availability analysis of distributed software and hardware systems

Lawan Abdulwahab¹, Jabir Tukur Abdullahi² and Ibrahim Yusuf³

¹Department of Information Technology, Bayero University, Kano, Nigeria
²Department of Computer Science, Bayero University, Kano, Nigeria
³Department of Computer Science, Bayero University, Kano, Nigeria

E-mail: dean.csit@buk.edu.ng

Abstract: Markov model for a simple two-host system used in determining the system availability of homogeneously distributed software and hardware systems is based on the concept of containing only hardware and software in each host. This system is found to be quite slow and tedious resulting to delays in computing system availability at a given time. Present research aim at stochastically analysing the availability of heterogeneously distributed software and hardware systems with software redundancy in standby, using an enhanced model to address the problem of low availability. The low availability is achieved through identifying the components responsible for mitigating the availability and define actions that can improve the availability of the systems. Markov method is used to develop expressions for system steady-state probabilities and system availability respectively. The model is implemented using Java programming language. To make the computation and analysis of systems availability more efficient, numerical examples are presented in order to illustrate the performance of the model.

1. Introduction

The demand for distributed computing system is very high due to the potential improvement in availability, fault-tolerance, performance, resource sharing and other applications that are profoundly used in several sensitive areas like banking, communication, home appliances, automobiles and aerospace. Consequently, this leads to the need to specify and design computing systems which could fulfil the requirements of the targeted applications at lowest cost, this has always been a challenge to the availability and reliability practitioners. Modelling and evaluation of such computing systems is an important step in the design of distributed systems [14,17].

A distributed system is a type of cluster that encompasses a number of computers in which any computer in the cluster has the capability of supporting the processing functions of any other computer in that cluster [15]. Clusters have redundant $n + k$ configurations, where $n$ represents the necessary processing nodes and $k$ represents the processing nodes that are in spare states [15]. System availability is a major performance criterion in distributed systems representing the percentage of the time the system is available to users. It is a probability that the system is working at any random point [6].

In spite of increasing demand for availability of distributed systems, little work has been dedicated to modelling and analysing such systems with independent failures of hardware and software components. However, among the sparse studies, [12] developed a Markov-based model to evaluate the availability of homogeneously distributed software and hardware systems in which all of the distributed hosts are of the same type such as workstations from the same vendor. Applications of distributed software to
homogeneously distributed systems are known as the homogeneously distributed software/hardware systems (HDSHS).

Thus, the purpose of the present study is to analyse stochastically the availability of heterogeneously distributed software/hardware systems with software redundancy in standby, using enhanced Markov-based model to address the problem of low availability through identifying the components responsible for mitigating the availability and to define actions that can improve the availability of the systems.

2. Related works
The initial and simplest case for the combined software/hardware system is to view it as a single processor that can be broadly separated into two subsystems: software and hardware subsystems. The software subsystem and hardware subsystem are regarded as two distinct black boxes, although combined together, due to difference in their characteristics. Considering such type of system, [5] presented one of the first Markov models.

Interestingly, a more realistic measure is the availability of the entire software/hardware system [27]. Goel and Soenjoto [4] first noted the characteristics of combined software and hardware system by describing system availability of a single-host based software/hardware systems using only one computer. A generalized model was also presented by [20] where both lifetimes together with the repair times of software and hardware subsystems are considered together. The availability issue was handled by [6] while [13] introduced the X-ware reliability and availability, addressing the problem of modelling system's reliability and availability with regard to the miscellaneous classes of faults (physical and design, internal and external) which alters the service delivered to its users. A two-level hierarchical model that analyzes the availability of distributed systems at both the lower and higher levels was presented by [8]. Bondavalli [1] introduced the first frame work where a dependability, performance and efficiency aspect of hybrid fault-tolerant architecture was collectively considered. The availability and reliability on the system level regarding the time-varying failure and repair rates that are dependent among the software/hardware components was studied by [2,23]. A model for availability analysis of homogeneously distributed software/hardware system was presented by [12]. The systems are made up of identical copies of distributed application software running on the same type of computers. The study was found to be helpful when studying an ideal testing time for system release.

Die [2] presented a model for determining the availability of centralized heterogeneous systems. The researchers studied the reliability and availability of distributed services using Markov process, combining both software failures and hardware and network failure together. The model is useful when dealing with miscellaneous issues such as the release time to accomplish a service reliability requirement. Li [19] proposed the development of a Markov-based model for the availability analysis of telecommunication management systems, considering both their hardware and software components. The model was found to be helpful in identifying if a system crash could be attributed to an external factor or to one of its modules. However, the limitations of the model are how to determine the system availability in its entirety based on simulations of many parts of the system. Jain [9] investigated the availability of a system with software-hardware interaction, spare, switching failure and common cause shock failure. Whenever the hardware fails, it is quickly replaced by a spare hardware. The life time and the repair time of the components are assumed to be exponentially distributed. Such an investigation was found to be useful when predicting the reliability of the system with the effect of software and hardware interaction instead of predicting about hardware and software separately. Stochastic Petri Net (SPN) models to assess dependability aspects of computer networks in distinct architectures was reported in prior study, the models encourage the analysis of system availability together with its services according to the most important components.

Okarafor and Patrick [18] proposed a Markov-based model to determine the availability of job tracker in Hadoop/MapReduce Zookeeper coordinated clusters. The model mitigated the difficulties in synchronizing, coordinating and preventing deadlocks in distributed systems [18]. It is also useful in determining the number of servers that are optimal for high availability. Kumar et al. [10] presented
stochastic modelling of computer systems having distinct levels of performance to evaluate system availability and mean time to system failure. The model indicated that a computer system with independent hardware and software is made profitable by prioritizing software up-gradation over the hardware repair activities. Malik and Munday [16] also proposed stochastic modelling of computer systems with hardware redundancy to address the problem of computer systems modelling in which the failures of hardware and software components are independent. Hence, the researchers found that a computer system with hardware redundancy provided in cold standby can be made more reliable, profitable and available by increasing hardware repair rate and less cost software up-gradation.

Kumar et al. [15] presented performance analysis of a computer system with imperfect fault detection of hardware. The study addressed the problem of modelling computer system using the concept of imperfect fault detection of hardware components. A stochastic model, using regenerative point technique and Markov process, was developed to evaluate the several performance measures of a computer system such as transition probabilities, availability analysis and performance analysis. The system has two identical units (one is operative and the other is kept in a cold standby) with a repair facility provided for fault detection, repair, replacement and up-gradation purpose of the components. The present work is an attempt to improve system availability function of Markov-based model for a two-host distributed software/hardware systems through heterogeneously incorporating two software systems in each host in standby mode so that if one fails the other replaces it [26].

![State transition diagram for the two-host system.](image)

3. Methodology

The study adopted an agile software development method, every software development may find a development method that is more or less appropriate. One of the philosophies behind agile method of software development is “Responding to change over following a plan”. This is what makes it the choice of developing the software because one of the problems of software project is that it changes along the way as the projects begin. Java programming language is chosen as a tool for the development of the software, because of its flexibility in producing reliable software, architectural neutrality to handle heterogeneous environment. Net Beans integrated development environments (IDE) is a software application that provide comprehensive facilities to programmers for software development. IDE was used for the interface design, Eclipse IDE was used for the program coding and Math Lab was used for
the graphic comparison between the heterogeneous system and homogeneous system to illustrate the improvement.

3.1 Descriptive analysis of homogeneous system

A sample two-host model based HDSHS is described in Lai et al [12]. For the model to be useful, it is vital that the assumptions must be good approximation to some real phenomena. The physical system with two copies of the same software (SW1 and SW2) running on two hosts (HW1 and HW2) where \( n = 2 \) as in figure 1 which depicts the state transitions of the system according to Markov chain. A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous events. In this model, it should be noted that both software and hardware must be in working states to keep each host functioning. Malfunctioning of either the software or hardware of a host leads the host to go down.

Based on the Markov assumptions, the Kolmogorov’s equations are derived for the calculation of the state probabilities. The system availability can then be obtained from the solutions of the state probabilities of the only working states of the system. Let \( P_i(t) \) denotes the probability that the system is in state \( i \) at any time \( t \).

3.2 Descriptive analysis of the proposed heterogeneous system

The enhanced Markov-based model for availability analysis of heterogeneously distributed hardware/software system is described in this section. The Markov-based assumptions concerning the system are:

i. All the hosts have the same hardware failure rate \( \lambda_h \) arising from an exponential distribution;

ii. Each of the host runs a copy of two heterogeneous software with a failure rate \( \lambda_s(t) \);

iii. Both the software and hardware have only two states, up (working state) and down (malfunctioning state), which means that only crash failures are considered;

iv. Maintenance personnel is available to repair the system upon software or hardware malfunction and the repair time, which includes the time of locating and fixing the problem, follows an exponential distribution with parameter \( \mu_h \) for hardware failure, respectively.

v. All the failures involved (either software or hardware) are mutually exclusive.

![State transition diagram of the heterogeneous system.](image-url)
For the model to be useful, it is vital that the assumptions must be good approximation to some real phenomena. The physical system with two copies of heterogeneous software type one (SW1 and SW2) and type two (SW1 and SW2) running on two hosts (HW1 and HW2) with \( n=2 \) as in Figure 2. Figure 2 depicts the state transitions of the system according to Markov chain. Note that both software and hardware must be in working states to keep each host functioning. Malfunctioning of either the software or hardware of a host leads the host to go down.

3.2.1 Description of the states of the system

State 0: Initial state, 2 Hardware, 2 type I software are working, 2 types II software are in standing.
State 1: 1 hardware is down, 1 hardware and 1 type I software are working, type I software is idle, 2 type II software is in standing.
State 3: 1 type I software is down, 2 hardware, 1 type I and 1 type II software are working, 1 type II software is in standing.
State 4: 2 type I software are down, 2 hardware’s, 1 type II software are working.
State 5: 2 type I software, 1 type II software is down, 1 hardware and 1 type II software are working, 1 hardware is idle.
State 6: 2 types I and 2 type II software are down, the system is down.
State 7: 1 hardware and 1 type I software are down, 1 type II software is in standing, 1 hardware and 1 type II Software are working.
State 8: 2 hardware’s are down; the system is down type I software is down.
State 9: 1 hardware is down, 1 type I and type II software are down, the system is down.
State 10: 2 type II software and one hardware is down, 1 type is software in standing, hardware’s and 1 type II software are working.
State 11: 2 hardware’s are down; the system is down.
State 12: 2 type I software, one hardware and 1 type II software are down, the system down.
State 13: 2 type I software, 1 type II software and 1 hardware are down, I hardware and type II Software are working.
State 14: 2 hardware is down; the system is down.
State 15: 2 type I software, 2 type II software, and one hardware is down, the system is down.

3.2.2 Model formulation

Based on the Markov assumptions, the Kolmogorov’s equations are derived for the calculation of the state probabilities. The system availability can then be obtained from the solutions of the state probabilities of the only working states of the system. In order to analyse the system availability of the system, define \( P(t) \) to be the probability that the system at \( t \geq 0 \) is in state \( S_j \). Also let \( P(t) \) be the row vector of these probabilities at time \( t \). The initial condition for this problem is:

\[
P(0) = [p_0(0), p_1(0), p_2(0), ..., p_{15}(0)] = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
\]

We obtain the following differential equations:

\[
p_0'(t) = -q_{0}p_0(t) + \mu_1p_1(t) + \mu_2p_2(t)
\]

\[
p_1'(t) = -q_{1}p_1(t) + 2\lambda_3p_0(t) + \mu_3p_3(t) + \mu_4p_4(t) + \mu_5p_5(t)
\]

\[
p_2'(t) = -q_2p_3(t) + 2\lambda_1p_0(t) + \mu_4p_4(t) + \mu_5p_5(t)
\]

\[
p_3'(t) = -q_3p_3(t) + \lambda_3p_1(t) + \mu_4p_4(t) + \mu_5p_5(t)
\]

\[
p_4'(t) = -q_4p_4(t) + 2\lambda_2p_4(t) + \mu_6p_6(t) + \mu_7p_7(t)
\]

\[
p_5'(t) = -q_5p_4(t) + 2\lambda_2p_4(t) + \mu_6p_6(t) + \mu_7p_7(t)
\]

\[
p_6'(t) = -q_6p_6(t) + \lambda_2p_5(t)
\]

\[
p_7'(t) = -q_7p_7(t) + \lambda_2p_5(t)
\]

\[
p_8'(t) = -q_8p_8(t) + \lambda_2p_5(t)
\]

\[
p_9'(t) = -q_9p_9(t) + \lambda_2p_5(t)
\]

\[
p_{10}'(t) = -q_{10}p_{10}(t) + \lambda_2p_5(t)
\]

\[
p_{11}'(t) = -q_{11}p_{11}(t) + \lambda_2p_5(t)
\]

\[
p_{12}'(t) = -q_{12}p_{12}(t) + \lambda_2p_5(t)
\]

\[
p_{13}'(t) = -q_{13}p_{13}(t) + \lambda_2p_5(t)
\]

\[
p_{14}'(t) = -q_{14}p_{14}(t) + \lambda_2p_5(t)
\]

\[
p_{15}'(t) = -q_{15}p_{15}(t) + \lambda_2p_5(t)
\]
\[ p_5'(t) = -q_5 p_7(t) + \lambda_3 p_3(t) + 2 \lambda_3 p_5(t) + \mu_2 p_6(t) + \mu_3 p_0(t) \]

\[ p_6'(t) = -\mu_1 p_6(t) + \lambda_2 p_7(t) \]

\[ p_{10}'(t) = -q_{10} p_{10}(t) + 2 \lambda_3 p_4(t) + \mu_1 p_{11}(t) + \mu_2 p_{12}(t) \]

\[ p_{11}'(t) = -\mu_3 p_{11}(t) + \lambda_2 p_{12}(t) \]

\[ p_{13}'(t) = -\mu_3 p_{13}(t) + \lambda_3 p_5(t) \]  

(1)

where

\[ q_0 = (2\lambda_3 + 2\lambda_1), q_1 = (\lambda_3 + \lambda_1 + \mu_3), q_2 = (2\lambda_3 + \lambda_1 + \mu_1), q_3 = (2\lambda_3 + 2\lambda_2 + \mu_1), q_4 = (\lambda_3 + \lambda_2 + \mu_2), q_5 = (\lambda_3 + \lambda_2 + \mu_5 + \mu_1), q_6 = (\lambda_3 + \lambda_2 + \mu_1) \]

(1) Can be written in the matrix form as

\[ \dot{P} = TP \]  

(2)

where

\[
T = \begin{pmatrix}
-q_0 & \mu_1 & 0 & \mu_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2\lambda_3 & -q_1 & \mu_3 & 0 & 0 & 0 & 0 & \mu_3 & 0 & 0 & 0 & 0 \\
0 & \lambda_3 & -\mu_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2\lambda_1 & 0 & -q_1 & \mu_4 & 0 & 0 & \mu_4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda_2 & -q_5 & \mu_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2\lambda_2 & -q_4 & \mu_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & -\mu_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Equation (2) is expressed explicitly in the form
The following is normalizing condition of the study:

$$\sum_{k=0}^{15} p_k(t) = 1$$  \hspace{1cm} (3)

The steady-state availability is the proportion of time the system is in a functioning condition or equivalently, the sum of the probabilities of operational states given by

$$A_v(t) = p_0(t) + p_1(t) + p_2(t) + p_3(t) + p_4(t) + p_5(t) + p_7(t) + p_{10}(t)$$

In the steady state, the derivatives of the state probabilities become zero and therefore Equation (3) becomes:

$$TP = 0$$  \hspace{1cm} (4)

which is presented in matrix form
Subject to following normalizing conditions:

\[ P_0(\infty) + P_1(\infty) + P_2(\infty) + P_3(\infty) + P_4(\infty) + P_5(\infty) + \ldots + P_{13}(\infty) \]  

Following [21] and [22] we substitute (5) in the last row of (4) to compute the steady-state probabilities.
Solving (5), we obtain the steady-state probabilities

\[ p_0(\infty), p_1(\infty), p_2(\infty), p_3(\infty), \ldots, p_{13}(\infty) \]  

(6)

The steady-state availability (the proportion of time the system is in a functioning condition or equivalently, the sum of the probabilities of operational states) is given by

Since states 0, 1, 3, 4, 5, 7, 10, and 13 are the only working states of the system in figure 2 above, the system availability at time \( t \), would be.

\[ A_v(t) = P_0(t) + P_1(t) + P_3(t) + P_4(t) + P_5(t) + P_7(t) + P_{10}(t) + P_{13}(t) \]  

(7)

4. Numerical example

This section describes the forms that are used in the software. The frames used in the software were all designed using the NetBeans GRAPHICAL USER INTERFACE (GUI) BUILDER. They are explained below:

*The main interface form:* The automated availability analysis of distributed software/hardware systems has only one frame which contains a main () method. All other frames are internal frames that are initialized from the frame with the main () method. Once the program is run, the input frame containing buttons that perform different functions when selected and clicked by a user is displayed as shown below:

![Automated Availability Analysis of Distributed Software/Hardware System](image)
The availability of the system can be computed using any available numerical algorithm to solve the differential equations. The inputs to the system are user-generated values. To illustrate numerically the application of the system, we assumed that the software 1 failure rate per fault is 0.2 per hour, the software 1 repair rate is 0.1 failures per hour, the software 2 failure rate per fault is 0.2 per hour, the software 2 repair rate is 0.1 failures per hour, the hardware failure rate is 0.1 per hour while the hardware repair rate is taken to be the dependent variable (containing value of 0) upon which the availability graph is plotted. An interval of 2 and a range of 1-50 are used. Below is how the input form looks like before clicking compute button.

![Automated Availability Analysis of Distributed Software/Hardware System](image)

**Figure 4.** The interface frame with input values.

After inputting the respective parameter values and clicking compute button, the main class that contains the method to perform the computation of the system availability as well as setting the dependent variables is called inside the frame. Below is the diagram that shows the output frame containing availability of the system for each value of the dependent variable and a portion of the sample code used to set the dependent variable.

In the main interface, after the computation of the availability, there is a graph button which when clicked, invokes the graph class that contains the methods. The methods inside the graph class call the JFreeChart class, XYSeries class, XYSeriesColletion class and ChartPanel class that set up the graph. The following are the graphs and the sample codes used to generate it to show that the inheritance relationship that exists.
Figure 5. The interface frame with output displayed.

```java
if(hrrv == 0 ){
    dependent.setText("Hardware Repair Rate");
    xlabel = "Hardware Repair Rate";

    for(i = rangelv; i<= range2v; i = i+intervalv)
    {

        y[c] = av_t( sfr1v, sfr2v, hfrv, srr1v, srr2v, i );
        x[c] = i;

        s += i"\t"+ y[c]+ "\n";

        c++;
    }
}
```

Figure 6. Sample code used to set the dependent variable (hardware repair rate).
Figure 7. Availability vs hardware repair rate.

```java
public class graph extends ApplicationFrame {
    // public graph(String title, double x[], double y[], int max, String ylabel);
    public graph(double[] x, double[] y, int max, String xlabel) {
        // final graph obj = new graph( x[], y[], c, ylabel);
        super("Jabir M.Sc. Project");
        finalXYSeries data = new XYSeries("");
        for(int i = 1; i <= max; i++) {
            data.add(x[i], y[i]);
        }
        finalXYSeriesCollection data1 = new XYSeriesCollection(data);
        finalJFreeChart chart = ChartFactory.createXYLineChart("", xlabel,
            "Availability", data1, PlotOrientation.VERTICAL, true, true, false);
        finalChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(850, 600));
        setContentPane(chartPanel);
    }
}
```

Figure 8. Sample code used to generate graph.
To illustrate numerically once again on how the system works using hardware failure rate as the dependent variable which is set to zero, we assumed as in the previous illustration that the software 1 failure rate per fault is 0.2 per hour, the software 1 repair rate is 0.1 failures per hour, the software 2 failure rate per fault is 0.2 per hour, the software 2 repair rate is 0.1 failures per hour while the hardware repair rate is 0.1 per hour. An interval of 5 and a range of value (1-100) are used. After inputting the respective parameter values and clicking compute button, the main class that contains the method to perform the computation of the system availability as well as setting the dependent variables is called inside the frame. Below is the diagram that shows the output frame containing availability of the system for each value of the dependent variable and a portion of the sample code that was used to set the dependent variable.

![Diagram showing output frame](image_url)

**Figure 9.** Output using hardware failure rate as dependent variable.

```java
else if( hfrv == 0){
    dependent.setText("Hardware Failure Rate");
    xlabel = "Hardware Failure Rate";
    for(i = range1v; i<= range2v; i = i+intervalv){
        y[c] = av_t( sfr1v, sfr2v, i, srr1v, srr2v, hrrv );
        x[c] = i;
        s += i+"\t"+ y[c]+ "\n";
    }
```

**Figure 10.** Sample code used to set the dependent variable (hardware failure rate)
After the computation of the availability, there is a graph button which when clicked, invokes the graph class that contains the methods these methods are then called on button-click inside the frame. The method inside the graph class calls the JFreeChart class, XYSeries class, XYSeriesCollection class and Chart Panel class that are used to generate the graph shows that the inheritance between them exists. Below is the graph.

![Graph](image)

**Figure 11.** Availability vs hardware failure rate.

Figure 12 shows the behaviour of the curves of system availability versus all the dependent variables (software one failure rate, software one repair rate, software two failure rate, software two repair rate, hardware failure rate and hardware repair rate respectively). Software two is in standby mode that is why its failure rate curve is close to zero along the x-axis. Hardware failure rate curve also tends to zero along the x-axis which signifies that the higher the hardware failure rate is the lower the system availability becomes and vice-versa.

It can be seen from Figure 12 that the system availability reaches the lowest point at an early stage. This is because a large number of faults are identified when software system testing begins. In the case of large systems, many of these faults are those from system integration testing when all software units are put together to work as a whole system.

The system availability starts recovering after the lowest point and approaches a certain value less than one asymptotically after a long period of time. This is because identified faults are fixed and as a result the software failure rate decreases. Eventually and ideally, all faults are fixed and the software becomes 'bug free'.

An important application is that the availability reaches a minimum of about variables = 5. This means that it can be recommended that the system should not be released before a testing of that amount of variable. In fact, by specifying a required level of availability, we can determine the amount of testing that is needed for the system to meet the availability requirement.
The system availability function is affected by the changes of dependent variables' parameters such as software failure rate and repair rate and the hardware failure rate and repair rate. Generally, the higher the software or hardware failure rate is, the less availability the system has. On the other hand, the higher the software or hardware repair rate is, the more availability the system has. Note that although the values of the variables at which the availability reaches its minimum are very close for some curves, they are different, and the difference is significant when the failure rates are small.

As mentioned earlier, one of the applications of this model is to help estimate the system release time during system testing and debugging. From the system availability curves (Figure 12) calculated numerically through Equation (3), one may find the value of the dependent variable at which the system availability reaches the desired level for system release. Another application is to help distributing the amount of effort on software testing. As indicated in Figure 12 system availability starts sharply from downward at an early stage. During this time, a significant amount of effort needs to be put into fault fixing and system testing to help increase system availability quickly. When the faults are fixed, the system availability rises and effort on fault fixing and testing can be reduced accordingly. Eventually, only a few faults will be left. At this stage, the manpower for the fault fixing and system testing can be moved to somewhere else.

5. Conclusion
An enhanced model for availability analysis of distributed software/hardware systems is subject to three types of failure hardware, software and combination of the two respectively. To determine the system availability, the case of two-host system is considered. The system has heterogeneous software's incorporated in each host, one is operative and the other one is in standby mode. As soon as the software fails the other one takes over. Both the software and hardware failures are considered, assuming that the life time and the repair time of each component are exponentially distributed. The Markov method is used to develop expressions for system steady-state probabilities and system availability respectively. The proposed model suggested an improvement in system availability function of Markov-based model for a two-host distributed software/hardware systems through heterogeneously incorporating two software
systems in each host in standby mode so that if one fails the other replaces it. The results suggested improvement in system availability and studying optimal testing time or testing resource allocation. The failure of either the host or the software bring about system failure. However, in our model each host is mounted with two dissimilar software in cold standby that perform similar task, therefore failure of either the software or the host will not retard the operation of our system. Thus, in our study availability can be enhance through redundancy in both host and software.

Acknowledgments
The authors wish to acknowledge the financial support from Bayero University, Kano for sponsoring this research from the beginning up to the stage of presenting some aspect of our work in a scientific conference.

References
[1] Bondavalli A, Chiaradonna S and Di G F 2002 Journal of Systems Architecture: The Euromicro Journal 47 763-781
[2] Dai Y S, Xie M, Poh K L and Liu G Q 2003 Reliability Engineering and System Safety 79 103 – 112
[3] Gilbert D 2008 The JFreeChart Class Library Version 1.0.9 (Hertfordshire: Refinery Limited)
[4] Goel A L 1985 IEEE Transaction in Software Engineering 11 141-142
[5] Goel A L and Soenjoto J 1981 IEEE Transactions on Reliability 30 232-239
[6] Goyal A and Ladenburg S 1987 Journal of Research and Development 31 651 -664
[7] Guimaraes A, Oliveira H, Barros R and Maciel P 2011 3rd Int. Conference on reliability importance in Communication Software and Networks (IEEE) pp. 328-332
[8] Hariri S and Mutlu H IEEE Transactions on Software Engineering SE21 50-56, 1995.
[9] Jain M, Agarwal S C and Preeti A 2010 Journal of International Academy of Physical Sciences 14 425 – 437
[10] Kumar A, Anand J and Malik S C 2013 International Journal of Agricultural Statistical Science 9 117-126
[11] Kumar A, Saini M and Malik S C 2015 Procedia Computer Science 45 602- 610
[12] Lai C D, Xie M, Poh K L, Dai Y S and Yang P A 2002 Information and software Technology 44 343-350
[13] Laprie J C and Kanpoun K 1992 IEEE Transitions on Software Engineering SE 130 -147
[14] Lopriore L 2000 Microprocessor and Microsystems 23 587 – 595
[15] Kumar A and Malik S C 2012 International Journal of Computer Applications 43 27-34
[16] Malik S C and Munday V J 2014 International Journal of Computer Applications 89 26-30
[17] Malluhi Q M and Johnston W E 1998 IEEE Transactions on Parallel and Distributed System 1269 – 1237
[18] Okarafor E and Patrick M K 2012 Advanced Computing. An International Journal 3 19-29
[19] Li R, Liu, X and Huang N 2014 Mathematical Problem in Engineering 1 1-9
[20] Sumita U and Masuda Y 1986 IEEE on Transition on Software Engineering SE-1 32 – 41
[21] Wang K H and Kuo C C 2000 Applied Mathematics Modeling 24 957-967
[22] Wang K, Hsieh C and Liou C 1986 Journal of Quality Technology and Quantitative Management 3 77-92
[23] Zhang T L and Horigone M 2001 IEEE Transactions on Reliability 50 151-158
[24] Anand J and Malik S C 2012 International Journal of Systems Assurance Engineering 3 230-236
[25] Malik S C 2013 International Journal of System Assurance Engineering and Management 4 94-100
[26] Chen C and Beltrame G 2017 ACM Transactions on Design Automation of Electronic Systems 23 1-24
[27] Kumar A and Malik S C 2014 Jordan Journal of Mechanical and Industrial Engineering 8 233-241