

ABSTRACT

In this chapter, an active/standby cluster system in which personal computers are connected in loosely coupled fashion for providing an improved availability is studied. In cluster system configuration with active/standby components, there may be a primary server which can be treated as active server and other servers work as standby parts of the cluster system. The concept of software rejuvenation has been incorporated to enhance and improve the performance of the embedded cluster system. A three level software rejuvenation policy is taken into consideration to develop a Markov model. The steady state probabilities of the model are obtained which are further used to obtain the availability and downtime cost of the active/standby cluster system. The results obtained are also compared by computing the same using Adaptive Neuro-Fuzzy Inference System (ANFIS approach). A numerical illustration has been provided to validate the model.

6.1 Introduction

Due to recent advances in Information technology, the provision of embedded cluster system in which personal computers are connected in a loosely coupled manner for commercially purpose is increasing day by day. These connected systems in the cluster system work simultaneously when any work load is given to the system. The main aim of present study is to develop the Markov model for the cluster system to predict the performance indices and to suggest the ways to reduce the execution time of the system. In the cluster system there are two types of servers, one is the main server (active unit) which receives all requests and distributes them among other servers working as standby parts of the cluster system. For the perfect functioning of the cluster system, the active server should work properly. Aung et al. (2005), Avritzer et al. (2007) and Asif et al. (2007) discussed system performance for the cluster systems. A unified fault-tolerant routing scheme for a class of cluster networks has been analyzed by Day et al. (2008). Clarke et al. (2008) examined clusters of class characteristics in object oriented applications. Resource allocation optimization
for quantitative service differentiation on server clusters has been done by Zhou and Ippoliti (2008). Boyer et al. (2009) considered a self-repair architecture for cluster systems. Ever et al. (2009) did analytical modeling and simulation of the small scale, typical and highly available clusters with breakdowns and repairs. Failure-aware resource management for high-availability computing clusters with distributed virtual machines has been discussed by Fu (2010). Erilli et al. (2011) determined the most proper number of cluster in fuzzy clustering by using artificial neural networks. Rechistov et al. (2012) described an approach to study the functional aspects large cluster computing system configurations.

The real time computer systems are more prone to failure due to the software failures rather than the hardware. The complexity of software is rapidly growing in order to execute the complex type of computation as such it is highly recommended to study the software faults. The software faults or bugs can be classified into three categories as shown in fig 6.1.

Fig. 6.1: Software Faults

Bohrbugs are those faults which are easily repeatable and can be removed quickly from the system. They can be removed in testing and debugging phase and the remaining are cleaned by using design diversity in which same functionality is used with different design. Heisenbugs, are the type of failure which occurs in the software very rarely and do not reproduce any further failure. These faults are transient in the nature. The failed operation should be retried or restarted to cure these types of failures. One of the classes of software faults occurs in internal and external environment of the software due to various reasons such as data corruption, memory leakage or bloating, numerical round off error and deterioration in the availability of the resource. This type of phenomenon is called as software ageing. Performance of
the system may be degraded and the operation of the software may be crashed or hang due to software ageing. Software rejuvenation is one of the proactive preventive maintenance which is performed by stopping the operation, cleaning its internal state, prevent the occurrence of the faults and restarting it to healthy state.

Due to software ageing, there may be high possibility of the occurrence of faults in the execution of the software. In order to terminate these faults, a proactive maintenance policy is highly recommended. These proactive maintenance actions are used to improve the software reliability and prevent or postpone the software failures. Software rejuvenation is one of these proactive actions which stops the applications of a cluster system and restarts it back to a healthy state and prevents the failure before actual occurring in the system. In this present study a three level software rejuvenation policy is suggested to improve the efficiency of the active/standby cluster system.

(i) **Software Rejuvenation level 1:** In this type of software rejuvenation, the stoppage of the cluster system does not have any effect on its applications. All the necessary data will be saved and after restart the service runs as a healthy state. Level 1 software rejuvenation is called as *first type of partial restart*. This type of action takes much less time for the completion.

(ii) **Software Rejuvenation level 2:** These proactive actions are also known as *second type of partial restart*. In the process of level 2 rejuvenation, the stoppage of the cluster system has some impact on the applications and actions running on the cluster system which is supposed to be minimum.

(iii) **Software Rejuvenation level 3:** The cluster system and all the applications going on it have to be stopped to perform this type of rejuvenation. The main objective of this kind of rejuvenation is to restart the cluster system to recover all its free memory. This is also termed as *full restart*. All the applications of the same cluster system are very much affected by this rejuvenation. It takes more of time for completion of the whole process.

There are some research workers who contributed in this field (cf. Bobbio et al., 2001; Park and Kim, 2002; Bao et al., 2005; Grottke and Trivedi, 2005; Xie et al., 2005). Koutras et al. (2007a) did the optimization of free resources using non-homogeneous Markov chain software rejuvenation model. Additionally the cost of
performing rejuvenation was also taken into consideration by them. Software rejuvenation for resource optimization has been discussed by Koutras et al. (2007b). Furthermore, the costs of two software rejuvenation techniques are determined and both techniques are compared with respect to the total expected downtime cost of performing rejuvenation. Wang et al. (2007c) explained the performability analysis of clustered systems with rejuvenation. They have designed three rejuvenation policies, namely standard rejuvenation, delayed rejuvenation and mixed rejuvenation, to improve the cluster’s performability under varying workload. Comprehensive evaluation of a periodic checkpointing and rejuvenation schemes in operational software system has been considered by Okamura and Dohi (2009). Avritzer et al. (2010) described the methods and opportunities for rejuvenation in aging distributed software systems. Salfner and Wolter (2010) analyzed the effect of time-triggered system rejuvenation policies on the service availability using a queueing model. Okamura and Dohi (2011) considered an operational software system with multiple degradation levels and derive the optimal software rejuvenation policy maximizing the steady-state system availability using the semi-Markov decision process. Paing and Thein (2012) presented high availability against software aging of virtualized server system by providing both clustering software rejuvenation and migration based software rejuvenation.

In the present investigation, an active/standby cluster system with three level of software rejuvenation policy has been considered. The rest of the chapter is arranged as follows. The requisite assumptions and notations stating the model formulation are given in section 6.2. The governing steady state equations are also constructed by taking appropriate transition rates. In section 6.3, various performance indices of the system are evaluated explicitly in term of steady state probabilities. Section 6.4 presents a brief introduction of adaptive neuro-fuzzy inference system (ANFIS). Numerical illustration and sensitivity analysis are given in section 6.5. Finally, the concluding remarks are given in section 6.6.

6.2 Model Description

In the present investigation, an active/standby cluster system is considered with three level of software rejuvenation policy as mentioned in previous section. There are n loosely connected units in the cluster system out of which n\textsuperscript{th} unit is main
server (active unit) and remaining \((n-1)\) units work as standby units. Every unit can be in three states; two degraded states and one unstable state. The rejuvenation policies take place when the unit is in the unstable state. When all units shut down, the reboot is highly recommended.

We have considered the following assumptions for the construction of the model:

- A number of identical servers are loosely connected in an active/standby cluster system and each of them has life time and repair time exponentially distributed with failure rate \(\lambda\) and repair rate \(\mu\).
- The each server has two degraded states namely medium efficiency and low efficiency and one unstable state.
- In medium efficiency state \(\{M,i\}, i = 1,2,\ldots,n\) and low efficiency state \(\{L,i\}, i = 1,2,\ldots,n\) the server may fail according to exponential distribution with degraded failure rate \(\lambda_M\) and \(\lambda_L\), respectively.
- From low efficiency state, each server of the active/standby system reaches in the unstable state \(\{U,i\}, i = 1,2,\ldots,n\}; the duration of this state is exponentially distributed with mean rate \(\lambda_U\).
- Each server of the active/standby cluster system either goes for rejuvenation or shutdown from unstable state.
- The unstable state may change according to exponential distribution with mean rate \(\alpha\) in one of the three levels of the software rejuvenation with the probabilities \(p_1, p_2\) and \(p_3\), respectively;
- All the rejuvenation rates are exponentially distributed. After first (second) type of partial restart, each server comes back in low (medium) efficiency state with rejuvenation rate \(\mu_1\) \((\mu_2)\). When the full restart of each server is done, the server returns back to the healthy state with the rejuvenation rate \(\mu_3\).
- After the shutting down of the last standby server, the active/standby cluster system is rebooted with rate \(\gamma\).

For model description, we define each state of the active/standby cluster system as follows:

**State** \(\{i = 1,2,\ldots,n\}\): This is fully working state of each server. After full restart, repairing from \((i-1)\)th state and rebooting after a crash, the server comes back to this state. The system is available in this state.

**State** \(\{M,i\}, i = 1,2,\ldots,n\}: In this state the software executions are done with medium efficiency. The server comes back to this state after second kind partial restart. The system is also available in this state.
**State** \(\{L_i, i = 1,2,\ldots,n\}\): The software executions run very slowly in this state and only first type of partial restart sent the server in this state. Again the system is available in this state.

**State** \(\{U_1, i = 1,2,\ldots,n\}\): The server either goes for rejuvenations or may be shutdown from this state. The system is unavailable in this state.

**State** \(\{R_1, i = 1,2,\ldots,n\}\): This is the first type of partial restart state and the system is unavailable in this state.

**State** \(\{R_2, i = 1,2,\ldots,n\}\): The second type of partial restart is taken place in this state and the system is also unavailable.

**State** \(\{R_3, i = 1,2,\ldots,n\}\): The system is unavailable in this state and full restart is in progress in this state.

**State** \(\{B\}\): The system is rebooted due to crash failure in this state and the system is still unavailable in this state.

---

The steady state equations governing the model are constructed by equating the in-flow and out-flow (see fig. 6.2) as follows:

\[
\begin{align*}
\lambda_M P_n &= \mu P_{n-1} + \mu_3 P_{R_1,n} + \gamma P_B \\
(\mu + \lambda_M) P_i &= \mu P_{i-1} + \lambda P_{U,i+1} + \mu_3 P_{R_1,i}, \quad i = 2,3,\ldots,n-1 \\
(\mu + \lambda_M) P_1 &= \lambda P_{U,2} + \mu_3 P_{R_1,1} \\
\lambda_L P_{M,i} &= \lambda_M P_{i} + \mu_2 P_{R_2,i}, \quad i = 1,2,\ldots,n \\
\lambda_U P_{L,i} &= \lambda_L P_{M,i} + \mu_1 P_{R_1,i}, \quad i = 1,2,\ldots,n \\
(\alpha + \lambda) P_{U,i} &= \lambda_U P_{L,i}, \quad i = 1,2,\ldots,n \\
\mu_1 P_{R_1,i} &= \alpha p_i P_{U,i}, \quad i = 1,2,\ldots,n \\
\mu_2 P_{R_2,i} &= \alpha p_2 P_{U,i}, \quad i = 1,2,\ldots,n
\end{align*}
\]
\[ \mu_3 P_{R,i} = \alpha P_{U,i}, \quad i = 1, 2, \ldots, n \quad (6.9) \]

\[ \gamma P_b = \lambda P_{U,1} \quad (6.10) \]

The normalizing condition is

\[ \sum_{i=1}^{n} P_i + \sum_{i=1}^{n} P_{M,i} + \sum_{i=1}^{n} P_{L,i} + \sum_{i=1}^{n} P_{U,i} + \sum_{i=1}^{n} P_{R,i} + \sum_{i=1}^{n} P_{R,1,i} + P_b = 1 \quad (6.11) \]

By solving equations (6.1)–(6.10) recursively, we obtain probabilities of different states as follows.

\[ P_n = \frac{\mu_3, \lambda, \delta(b^{n-2} - (n-3)b^{n-4}\mu\lambda\delta) + \alpha P_{U,i} (b^{n-1} - (n-2)b^{n-3}\mu\lambda\delta) + \gamma \mu_3 (\lambda\delta)^n}{\mu_3 (\lambda\delta)^n} P_b \quad (6.12) \]

\[ P_1 = \frac{\gamma \lambda M}{\lambda_1[\lambda + \alpha(1 - p_1 - p_2)]} P_b \quad (6.13) \]

\[ P_2 = \frac{(\mu + \lambda M - \alpha P_1 \delta)}{\lambda \delta} P_b \quad (6.14) \]

\[ P_1 = \frac{\beta^{i-1} - (i-2)\beta^{i-3}\mu\lambda\delta}{(\lambda\delta)^i} P_b, \quad i = 3, 4, \ldots, n - 1 \quad (6.15) \]

\[ P_{M,i} = \frac{\lambda M_1}{\lambda_1[\lambda + \alpha(1 - p_1)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.16) \]

\[ P_{L,i} = \frac{\lambda M_2}{\lambda_1[\lambda + \alpha(1 - p_1 - p_2)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.17) \]

\[ P_{U,i} = \frac{\lambda M_3}{[\lambda + \alpha(1 - p_1 - p_2)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.18) \]

\[ P_{R,i} = \frac{\lambda M_4}{[\lambda + \alpha(1 - p_1 - p_2)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.19) \]

\[ P_{R,1,i} = \frac{\lambda M_5}{[\lambda + \alpha(1 - p_1)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.20) \]

\[ P_{R,2,i} = \frac{\lambda M_6}{[\lambda + \alpha(1 - p_1 - p_2)]} P_i, \quad i = 1, 2, \ldots, n \quad (6.21) \]

Here, for brevity of notation, we have used

\[ \delta = \frac{\lambda M}{[\lambda + \alpha(1 - p_1 - p_2)]}, \ \beta = (\mu + \lambda M - \alpha P_1 \delta) \]

The probability \( P_b \) can be obtained using normalizing condition given in (6.11).
6.3 Performance Measures

It is worthwhile to predict the cluster system performance in terms of state probabilities which are evaluated in previous section. With three levels of rejuvenation policy, we establish quantitative measures for availability and downtime cost as follows:

6.3.1 Availability of the system: The cluster system is not available at all the three levels of rejuvenation and during the reboot state. Therefore the availability of the system is obtained as

\[
\text{Availability} = 1 - \left( P_R + \sum_{i=1}^{n} P_{R,i} + \sum_{i=1}^{n} P_{R_2,i} + \sum_{i=1}^{n} P_{R_3,i} \right)
\]  

(6.22)

6.3.2 Downtime cost: The downtime cost \( C(T) \) can be determined by using the unavailable states of the cluster system in terms of corresponding cost elements as

\[
C(T) = \left[ C_B \times P_B + C_1 \times \sum_{i=1}^{n} P_{R,i} + C_2 \times \sum_{i=1}^{n} P_{R_2,i} + C_3 \times \sum_{i=1}^{n} P_{R_3,i} \right] T
\]  

(6.23)

where

\[
\begin{align*}
T &= \text{Operational time} \\
C_B &= \text{The unit cost of the reboot of the cluster system} \\
C_i &= \text{The unit cost of the rejuvenation of } i^{th} (i=1, 2, 3) \text{ level of the cluster system}
\end{align*}
\]

6.4 Computation Based on Adaptive Neuro-fuzzy Inference System (ANFIS) Approach

The Adaptive Neuro-Fuzzy Inference System (ANFIS) network results for the availability of the cluster system are displayed in figs 6.5(a)-6.8(b) with the help of the fuzzy toolbox of the MATLAB software. The input parameters \( \lambda_M, \lambda, \alpha \) and \( \mu \) are treated as the linguistic variables in the context of the fuzzy systems which are shown in table 6.1. The membership functions for input parameters have been described by using the generalized bell-shaped function. The ANFIS networks are trained for 10 epochs for all the approximations. The shapes of the corresponding membership functions for figs 6.5(b)-6.8(b) wherein we have plotted
availability by varying default parameters, are displayed in figs 6.5(a)-6.8(a), respectively.

<table>
<thead>
<tr>
<th>Input Variables</th>
<th>Number of membership functions</th>
<th>Figures depicting membership function</th>
<th>Linguistic Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_M$</td>
<td>4</td>
<td>6.5(a)</td>
<td>Low, Moderate, Medium, High</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>4</td>
<td>6.6(a)</td>
<td>Low, Moderate, Medium, High</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>4</td>
<td>6.7(a)</td>
<td>Low, Moderate, Medium, High</td>
</tr>
<tr>
<td>$\mu$</td>
<td>4</td>
<td>6.8(a)</td>
<td>Low, Moderate, Medium, High</td>
</tr>
</tbody>
</table>

Table 6.1: Linguistic values of the membership functions for $\lambda_M$, $\lambda$, $\alpha$ and $\mu$

The graphs are plotted for availability for the different values of $p_1$ in figs 6.5(b)-6.8(b) where the analytical (ANFIS) results are represented by continuous (discrete) lines. It is noticed from figures 6.5(b)-6.8(b) that the availability decreases on increasing $\lambda_M$, $\lambda$ and $\alpha$ but it increases with the increment in the values of $\mu$. Both results obtained by analytical and ANFIS are at par with each other.

### 6.5 Numerical Illustration

Numerical results are facilitated to explore the effect of various parameters on the availability and downtime cost of active/standby cluster system. For this purpose, the program is coded in software MATLAB. The failure rates of the servers are taken in the months while the repairs of the servers take the hours for the completion. In order to compute the performance indices, we set default parameters as $p_1 = 0.33$, $p_2 = 0.33$, $\alpha = 0.7$, $\gamma = 3$, $C_b = 100$, $C_1 = 25$, $C_2 = 50$, $C_3 = 75$ and $p_3 = 1 - (p_1 + p_2)$. The proposed active/standby cluster system operates for one year.
Figs 6.3 (a-d) show the influence of failure rates along with the rejuvenations rates on the availability of the active/standby cluster system. In fig. 6.3 (a), it is seen that the availability decreases sharply for the increasing values of $\lambda_M$ but increases gradually as $\mu_3$ increases. Fig. 6.3(b) demonstrates the effect of increasing $\lambda_L$ and $\mu_2$ on the availability. It is observed that the availability decreases sharply as $\lambda_L$ increases but increases as $\mu_2$ increases. Fig. 6.3(c) reveals the behavior of the availability by varying the values of $\lambda_U$ and $\mu_1$, respectively. It is seen that the availability decreases sharply for the increasing values of $\lambda_U$ but increases gradually as $\mu_1$ increases. Similar pattern of the availability can be seen in fig. 6.3(d) for different values of $\lambda$ and $\mu$.

The effects of various parameters on the downtime cost ($C$) are exhibited in figs 6.4 (a-d). It is obvious from fig. 6.4 (a) that the downtime cost $C$ increases with the increment in $\lambda_M$ and decreases as $\mu_3$ increases. Fig. 6.4(b) depicts the increasing behavior of $C$ with higher values $\lambda_L$ but decreasing trend of $C$ with the increment in the values of $\mu_2$. In similar way, it is clear from fig. 6.4(c) that $C$ increases as $\lambda_U$ increases but decreases as $\mu_1$ increases. It is noticed with the help of fig. 6.4(d) that $C$ increases (decreases) as $\lambda$ ($\mu$) increases which matches with our expectation from real life experience.

On the basis of numerical experiment performed by taking an illustration, we conclude that

- As expected, the availability decreases as $\lambda_M, \lambda_L, \lambda_U, \lambda$ and $\alpha$ increase but it increases as the values of $\mu_1, \mu_2, \mu_3$ and $\mu$ increase.
- The downtime cost increases with the increment in the values of $\lambda_M, \lambda_L, \lambda_U$ and $\lambda$; on the contrary it shows decreasing pattern as $\mu_1, \mu_2, \mu_3$ and $\mu$ increase.
- We can conclude that the results which have been obtained from ANFIS are at par with the analytical results.

6.6 Conclusion

The complexity of the software is growing day by day as such the possibility of the occurring failures in the system due to software failure is much higher in
comparison to hardware failure. The rejuvenation policies are the powerful tool for the enhancement and the improvement of the availability and the performance of the active/standby cluster system. Therefore proactive maintenance policies should be used in the system to prevent the occurrence of failures in the software. Software rejuvenation is helpful in preventing the failures in the running process of the software executions. In this chapter, availability analysis of the active/standby cluster system with software rejuvenation and reboot has been considered by developing Markov model. Furthermore, ANFIS approach is used to evaluate the availability and the downtime cost of the system. The proposed work may be applicable to real time active/standby cluster system in which the prevention of the software failure is recommended for smooth running of the software applications under techno-economic constraints. We hope that our investigation will be fruitful for the system managers to improve embedded computer systems wherein software failures can not be avoided.
Fig. 6.3(a): Availability vs $\lambda_M$ for different values of $\mu_3$

Fig. 6.3(b): Availability vs $\lambda_L$ for different values of $\mu_2$

Fig. 6.3(c): Availability vs $\lambda_U$ for different values of $\mu_3$

Fig. 6.3(d): Availability vs $\lambda$ for different values of $\mu$
Fig. 6.4(a): Downtime cost vs $\lambda_M$ for different values of $\mu_3$

Fig. 6.4(b): Downtime cost vs $\lambda_L$ for different values of $\mu_2$

Fig. 6.4(c): Downtime cost vs $\lambda_U$ for different values of $\mu_1$

Fig. 6.4(d): Downtime cost vs $\lambda$ for different values of $\mu$
Fig. 6.5 (a): Membership functions for input parameter $\lambda_M$.

Fig. 6.5 (b): Availability by varying $\lambda_M$ for different values of $p_1$.

Fig. 6.6 (a): Membership functions for input parameter $\lambda$.

Fig. 6.6 (b): Availability by varying $\lambda$ for different values of $p_1$. 
Fig. 6.7 (a): Membership functions for input parameter $\alpha$.

Fig. 6.7 (b): Availability by varying $\alpha$ for different values of $p_1$.

Fig. 6.8 (a): Membership functions for input parameter $\mu$.

Fig. 6.8 (b): Availability by varying $\mu$ for different values of $p_1$. 