CHAPTER 5

GENETIC BASED SECURE AND INDEPENDENT TASK SCHEDULING

In the present scenario, distributed scheduling has important role in the field of computational activities. Various heuristic and metaheuristic based scheduling algorithms have been proposed by different researcher in the past. Security has been considered as one of the key factor in the adoption of any distributed environment. Scheduling of applications in a secure distributed environment demands the mapping of tasks on to those resources that can meet the task’s security requirements. In this chapter, a genetic based security-aware algorithm has been proposed for scheduling of independent tasks in a distributed environment with an aim to improve the makespan and resource utilization while keeping security overhead to the minimum.

5.1 INTRODUCTION

A distributed system is basically a collaboration of various autonomous systems which are connected through a well planned network [71]. The characteristics of distributed system such as sharing, reliability and availability mark its importance in handling real life applications. As the systems grows, it leads to few complexities in terms of scheduling of available resource for resource optimization, assuring the security, and at last but not the least multiple point of failure [172].

Task Scheduling means mapping of tasks to available resources to get the optimum result with respect to minimum execution time (makespan) and improved load balancing (resource utilization). Various algorithms have been proposed to manage the task scheduling on distributed resources. Few of the challenges in distributed systems for scheduling of applications are mentioned below [173]:

• Resources can be shared among the different tasks.
• In the sharing mode, the tasks are more vulnerable to security threats.
• Synchronization of data is a vital element to maintain the data consistency.
• Heterogeneous nature of resources.

Due to heterogeneous nature of systems, resources perform different for same set of tasks as per their working capacity. To tackle this problem of distributed system, an optimized task scheduling algorithm is required. On broad basis, task scheduling approaches can be categorized as Heuristic and metaheuristic approaches.

• **Heuristic Approach:** Min-Min, Max-Min and sufferage are considered as traditional heuristics and prominently dominate the task scheduling activities in distributed environment. With the emergence of Max-Min and Min-Min, a new approach is proposed by [175] that work on exploiting the merits of these heuristics in different situation. The Min-Min heuristic [174] performs the mapping of task having minimum completion time to the faster machine first so as to achieve the minimum makespan. Whereas, in Max-Min algorithm task having maximum completion time is allocated to the fastest machine.

• **Metaheuristic Approach:** Metaheuristic based scheduling techniques give the optimum result or nearest to optimum result. It. Various metaheuristic techniques such as Genetic Algorithm, Ant Colony Optimization, Simulated Annealing, Particle Swarm Optimization etc. gives better results for simple and complex problems. All of these techniques give the better results than the Conventional algorithms. GA creates the solution on randomly generated chromosomes by modify them genetically [176]. Simulated Annealing is known as single point search metaheuristic [177]. It works to achieve the optimum but it use the point to point approach unlike GA.
5.2 PROPOSED ALGORITHM

Purpose of this research work is to design a task scheduling technique which can give the minimum execution time and at the same time balance the load equally among all available resources. To achieve this objective; in the proposed algorithm the two heuristic approaches Max-Min and Max-Min<>Min-Min have been used in combination with GA to get the optimum results. Three major steps in genetic algorithm are initial population, crossover and mutation as represented in Figure 5.1. It works like the human genetic process which means it gives the improvement with every generation. In proposed algorithm, a fitness function is defined accordingly to select the genetically improved population and to discard the inferior results during next iterations.

- **Initial Population Generation:** This is the basic module of proposed algorithm which generates first two individuals of initial population with the help of Max-Min and Max-Min<> Min-Min algorithm respectively and the remaining are generated on random basis.

- **Crossover:** In the implementation of genetic approach, various types of crossovers are available. However, in proposed solution two point crossover is used with crossover rate set to 0.50.

- **Mutation:** The mutation is used to adds some more features in the individuals. In proposed algorithm, rebalancing technique is used as a mutation operator as it also leads to balance the load among available resources. In proposed algorithm, mutation is performed on the task till the jobs are not balanced in the system.

- **Fitness Criterion:** Minimum makespan is considered as prime fitness function and load on the resources is also taken into consideration while evaluating the acceptability of individuals (schedules).

- **Termination Condition:** The termination condition is based on the number of iterations defined by the user.
Algorithm 5.1 is showing the step-by-step procedure of proposed genetic algorithm and algorithm 5.2 is the sub module of proposed algorithm representing the creation of initial population (Chromosome) for the genetic algorithm. This algorithm further calls two sub modules (algorithm 5.3 and algorithm 5.4). As evident from figure 5.1, the first two mappings of initial population are generated by Max-Min and Max-Min<>Min-Min algorithm respectively and remaining chromosomes are designed on random basis.

![Figure 5.1: Structure of Proposed Algorithm](image)
Algorithm 5.1: Proposed Genetic Algorithm for Independent Task Scheduling

1. **PROCEDURE**: PGA


3. begin

4. PopulationList ← ø

5. for size of population P


7. endfor

8. sort(PopulationList) as per makespan value

9. While iteration are not fulfill do

10. Perform the Uniform Crossover and add new individual to PopulationList (0.50)

11. for all individual or mapping ∈ PopulationList

12. Perform mutation on mapping generate mutated individual to populationList

13. sort(PopulationList) as per makespan value

14. best Mapping ← top value from PopulationList

15. endfor

16. Reduce the population size to p using Roulette Wheel

17. endwhile

18. end

Algorithm 5.2: Procedure Generate Initial Population


2. Output : mapping

3. begin

4. first mapping ← PopulationList ∪ Min_Min (T, R,P)
5. second mapping ← PopulationList ∪ Max_Min (T, R, P)

6. for remaining t ∈ T

7. select r randomly from available resource in ready format

8. Tuple(t,r) ← Assign t on r

9. mapping ← mapping ∪ Tuple(t,r)

10. endfor

11. return mapping

12. end

Algorithm 5.3: Max_Min


Output: mapping

1. begin

2. for each t ∈ TaskList do

3. for each r ∈ R do

4. calculate minimum completion time for every task

5. endfor

6. find the maximum completion time out of available min values

7. endfor

8. Schedule the task t_m on r_m as per m value

9. Remove the t_m from TaskList

10. Update the Status of r_m

11. Endfor

12. return mapping

13. end
Algorithm 5.4: \textit{Max-Min}$\prec$\textit{Min-Min} 

Input: TaskList \( T \), Resource Set \( R \), Population Size \( P \)  
Output: \textit{mapping} 

1. Begin 
2. While(\( J! = \text{Null} \)) 
3. \textbf{for each} \( t \in \text{TaskList} \) \textbf{do} 
4. \textbf{for each} \( r \in R \) \textbf{do} 
5. calculate completion time for every task 
6. endfor 
7. endfor 
8. \textbf{for each job} \( j_i \in J \) 
9. Find the minimum completion time and with machine number that obtain it 
10. endfor 
11. Calculate the \textit{Standard Deviation} of completion time of all non-allocated tasks 
12. \textbf{Sort} as per completion time 
13. Where the completion time of two consecutive job is more than SD, find that positions. 
14. \textbf{If} position in first half or SD$< \text{threshold value}$ 
    Apply Min-Min algorithm 
Else 
    Apply Max-Min algorithm 
15. endif 
16. Schedule the task \( t_m \) on \( r_m \) as per selected algorithm
17. Remove the $t_m$ from TaskList

18. Update the Status of $r_m$

19. endwhile

20. return $mapping$

21. end

5.3 IMPLEMENTATION OF PROPOSED ALGORITHM

The best scheduling algorithm is the one that gives better result for submitted tasks under specified constraints. The simulation parameters used to check the performance of proposed algorithm is shown in Table 5.1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks</td>
<td>10-60</td>
</tr>
<tr>
<td>Number Machines</td>
<td>10</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>150</td>
</tr>
<tr>
<td>Crossover Operator</td>
<td>Uniform-Point Crossover</td>
</tr>
<tr>
<td>Mutation Operator</td>
<td>Rebalancing</td>
</tr>
<tr>
<td>Condition for Termination</td>
<td>Number of Iterations</td>
</tr>
</tbody>
</table>
Figure 5.2: Performance comparison in terms of makespan for proposed GA and standard GA in terms of task variation

Figure 5.3: Performance comparison in terms of resource utilization for proposed GA and standard GA in terms of task variation
Figure 5.2 shows the makespan for Standard GA (SGA) and proposed algorithm with 10 machines and different range of tasks (10-60). Figure 5.3 shows the resource utilization (Load Balance) for the same parameters. The results show that the proposed algorithm is able to show improvement for both the conflicting goals under different load conditions (with increase in number of tasks).

Proposed algorithm not only performs better under load conditions but also shows its strength under scalable conditions. Figure 5.4 and Figure 5.5 shows the improvement in makespan and resource utilization for proposed algorithm in the situation where the number of machines are varied from 10 to 60 and number of tasks kept as 100. Figure 5.6 to Figure 5.8 corresponds to the different stages of the simulation process carried out using MATLAB.

![Performance comparison in terms of makespan for proposed GA and standard GA in terms of machines](image)

**Figure 5.4:** Performance comparison in terms of makespan for proposed GA and standard GA in terms of machines
Figure 5.5: Performance comparison in terms of resource utilization for proposed GA and standard GA in terms of machines

Figure 5.6: Snapshot of simulation process showing schedule generation
Figure 5.7: Snapshot of simulation process showing utilization of resources

Figure 5.8: Snapshot showing simulation results
5.4 SECURITY-AWARE INDEPENDENT TASK SCHEDULING

The task scheduling in a distributed environment usually involves the mapping of tasks on geographically distributed resources managed under different administrative domains. As the information has to be exchanged over networks, so user’s applications are easily prone to various security threats. In the current scenario, applications with stringent security requirement have introduced a new challenging issue for the researchers. There are various factors which play an important role in the design of a secure system; out of them confidentiality, integrity and availability are the basic elements to maintain the security. These are known as CIA triad in the security management:

- Confidentially
- Integrity
- Availability

- **Confidentiality:** In today’s world, every user is willing to maintain the privacy of data. They are not agreed that their provided information should be used by any other person or they can access the information. Every organization has its trade secrets which should be maintained as secret information. Establishing the security of such information is considered as prime element for information security and to attain this various security protocols like SSL and TLS [178] are enforced.

- **Integrity:** It becomes the prime responsibility of every service provider to ensure that the information provided by their client should not be accessed or modified by any other unauthorized user [179]. The user’s data has its significance, only when it is not tempered by the malicious users. Various techniques like cryptography, hashing and digital signature are deployed to maintain the data and information integrity.
• **Availability:** Availability refers to the condition that the information required by the user is readily available at all times [180]. In the current competitive scenario, the provider which is capable of keeping its promise regarding information delivery will have long suitability. There are a number of incidents where the major service providers were shut down due to security attack. Such denial of service can cause a great loss to the business [181]. So to ensure the availability of data, backup of data or redundancy of data with data consistency is equally important.

Considering all these aspects, it is the need of hour to ensure that the applications security are given due importance during allocation of resources.

### 5.5 PROPOSED SECURITY DRIVEN ALGORITHM

Every user application’s has one or other security requirements and it may vary on the basis of data involved and user’s paying capacity. It is assumed that resources available in the system and can be the potential candidate are having different security level. The resources offering higher level of security are using better security protocol like SSL or TSL. The incorporation of these security services during task execution always leads to security overhead which is calculated as:

\[ t_{ij} \text{ completion time} = t_{rj} \text{ completion time} \times (1 + \frac{\text{trust req task}(i)}{\text{SLR}}) \]

Where \( t_i \) represent the ith task

\( r_j \) represent the jth resource

\( \text{trust req task}(i) \) represents the trust requirement of ith task

\( \text{SLR} \) represents the maximum level of security

This security mechanism is based on some presumptions such as:

- Resources are providing different levels of security.
- Tasks are also demanding different levels of security.
➢ Security Level assumed varies from 1 to 9.

➢ Security Level 1 provides weakest level of security and 9 is highest level of security.

➢ The highest security level implies highest security overhead.

On the basis of above presumption, a security driven Modified Genetic Algorithm (MGA) algorithm is proposed to handle independent task scheduling. Algorithm 5.5 presents the proposed security algorithm which calculates the security overhead as per equation described above. Algorithm 5.6 is invoked to generate the initial population. Algorithm 5.6 generates the first two mapping based on security aware Max-Min and Min-Min algorithm (algorithm 5.7 and algorithm 5.8 respectively) and remaining on random basis while considering the security demanded of task and the resources. This algorithm allocates the task based on the demand of security by the specific task and then it will update the task list and lead to allocation of remaining tasks.

Algorithm 5.5: Security Scheduling Algorithm for Independent Task Scheduling

PROCEDURE: MGA


1. begin
2. for each t ∈ Task List do
3. for each r ∈ Rdo
4. t, r completion time = t, r completion time(1 + trust_req_task(i)/SLR)
5. endfor
6. endfor
7. PopulationList← ø
8. for size of population P
9. $\text{PopulationList} \leftarrow \text{PopulationList} \cup \text{Generate Initial Population (T, R, ST, SR)}$

10. endfor

11. sort($\text{PopulationList}$) as per makespan value

12. While iteration are not fulfill do

13. Perform the Uniform Crossover and add new individual to $\text{PopulationList}$ (0.50) if randomly selected R fulfill the demand of trust_req_task ST

14. for all individual or mapping $\in \text{PopulationList}$

15. Perform mutation on mapping generate mutated individual to populationList(0.10) if randomly selected T as per requirement of trust_req_task ST

16. sort($\text{PopulationList}$) as per makespan value

17. best Mapping $\leftarrow$ top value from $\text{PopulationList}$

18. endfor

19. Reduce the population size to p using Roulette Wheel

20. endwhile

21. end

**Algorithm 5.6: Procedure Generate Initial Population**

Input: TaskList T, Resource Set R, trust_req_task ST, trust_R SR

Output: mapping

begin

1. first mapping $\leftarrow$ PopulationList $\cup$ Min_Min (T, R, ST, SR)

2. second mapping $\leftarrow$ PopulationList $\cup$ Max_Min (T, R, ST, SR)

3. for remaining t $\in$ T

4. select t and r randomly
5. \textbf{if}(\text{trust}_R(r) \geq \text{trust}_{req\_task}(t))

6. \textbf{Tuple}(t,r) \leftarrow \text{Assign} \ t \text{ on} \ r

7. \textit{mapping} \leftarrow \textit{mapping} \cup \text{Tuple}(t,r)

8. \textbf{endif}

9. \textbf{endfor}

10. \textbf{return} \textit{mapping}

11. \textbf{end}

\textbf{Algorithm 5.7: Min\_Min}

\begin{itemize}
  \item \textit{Input:} TaskList T, Resource Set R, trust\_req\_task ST, trust\_R SR
  \item \textit{Output:} \textit{mapping}
  \item 1. begin
  \item 2. \textbf{for each} t \in \text{TaskList} \textbf{do}
  \item 3. \textbf{for each} r \in R \textbf{do}
  \begin{itemize}
    \item Calculate minimum completion time for every task as security requirement
  \end{itemize}
  \item 4. \textbf{endfor}
  \item 5. find the minimum completion time out of available min values
  \item 6. \textbf{endfor}
  \item 7. Schedule the task \( t_m \) on \( r_m \) as per minimum value
  \item 8. Remove the \( t_m \) form TaskList
  \item 9. Upadate the Status of \( r_m \)
  \item 10. End
\end{itemize}
Algorithm 5.8: Max_Min

1. Input : TaskList T, Resource Set R, trust_req_task ST, trust_R SR
2. Output : mapping
3. begin
4. for each t ∈ TaskList do
5. for each r ∈ R do
6. calculate minimum completion time for every task as per security requirement
7. endfor
8. find the maximum completion time out of available min values
9. endfor
10. Schedule the task $t_m$ on $r_m$ as per maximum value
11. Remove the $t_m$ form TaskList
12. Update the Status of $r_m$
13. End

5.6 IMPLEMENTATION OF SECURITY-AWARE PROPOSED ALGORITHM

The simulation parameters used to check the performance of proposed security driven algorithm (MGA) is shown in Table 5.2
Table 5.2: Simulation Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks</td>
<td>10-50</td>
</tr>
<tr>
<td>Number Machines</td>
<td>10</td>
</tr>
<tr>
<td>Range of Iterations</td>
<td>1-150</td>
</tr>
<tr>
<td>Type of Crossover</td>
<td>Uniform Crossover</td>
</tr>
<tr>
<td>Type of Mutation used</td>
<td>Rebalancing</td>
</tr>
<tr>
<td>Condition for Termination</td>
<td>Number of Iteration</td>
</tr>
</tbody>
</table>

Figure 5.9: Performance comparison in terms of makespan for MGA and SGA in terms of tasks
Figure 5.10: Performance comparison in terms of makespan for MGA and SGA in terms of machines.

Figure 5.11: Performance comparison in terms of resource utilization for MGA and SGA in terms of tasks.
Figure 5.12: Performance comparison in terms of resource utilization for MGA and SGA in terms of machines.

Figure 5.13: Performance comparison in terms of security overhead for MGA and SGA in terms of tasks.
Figure 5.9 shows the makespan calculated with fixed number of machines 10 and variant number of tasks (10-50) for standard GA and proposed algorithm. Figure 5.11 shows the resource utilization (Load Balance) for the same. The results show that the proposed algorithm exhibit good performance under increased load in terms of increase in the number of tasks and it also maintains the security in the system.

The makespan and resource utilization is still on positive side, when the numbers of machines are varied keeping number of tasks fixed as shown in Figure 5.10 and Figure 5.12 respectively. Figure 5.13 and Figure 5.14 gives the comparison of proposed and traditional GA in terms of security overhead under varying load and scalability conditions.
5.7 SUMMARY

The stringent application’s security requirements in a distributed environment are always a challenging task. In this chapter, two genetic based algorithms have been proposed for independent task scheduling with and without considering the security requirements of tasks to find better solution focusing on application’s execution time, load among available resources and security overhead. The simulation results are in favor of proposed approach as compared to traditional genetic approach.