Workload Analysis in a Grid Computing Environment: A Genetic Approach

P.K.Yadav
CSIR- Central Building
Research Institute, Roorkee

Anuradha Aggarwal
Department of Mathematics and Statistics, Faculty of Science, Gurukul Kangri University, Haridwar (UK), India

M.P.Singh
Department of Mathematics and Statistics, Faculty of Science, Gurukul Kangri University, Haridwar (UK), India

ABSTRACT
Grid computing is the collection of computer resources from multiple locations to reach a common goal. The grid is a special type of distributed system with non-interactive workloads that involve a large number of files. Partitioning of the application program/software into a number of small groups of modules among dissimilar processors is an important parameter to determine the efficient utilization of available resources in a grid computing environment. It also enhances the computation speed. The task partitioning and task allocation activities influence the distributed program/software properties such as IPC. This paper presents a metaheuristic model, that performs static allocation of a set of “m” modules of distributed tasks/program considering the two conflicting objectives i.e. minimizing the makespan time and balanced utilization of a set of “n” available resources of a grid computing. Experimental results using genetic algorithm indicates that the proposed algorithm achieved these two objectives as well as improve the dynamic heuristics presented in literature.

Keywords
Grid computing, Task Allocation, Makespan, Execution Cost, Inter Task (module) Communication Cost

1. INTRODUCTION
A Grid is a dynamic heterogeneous environment agglomerating geographically distributed resources and is defined as a process of taking scheduling decisions concerning resources spread over various administrative domains [1]. Users can contribute to grid resources by submitting computing tasks to grid system. The contribution of resources can be active or inactive within the grid. Hence it is impossible for anyone to assign tasks to computing resources by hand in grids. Therefore grid job scheduling [2, 3, 4] is one of the challenging issues in grid computing. Grid scheduling system selects the resources and allocates the user submitted tasks to appropriate resources in such a way that the user and application requirements are met [5]. With more applications looking for faster performance, makespan and balanced utilization of resources are most important objectives that scheduling algorithms challenge to optimize. Makespan is the resource utilization time between the beginning of the first task and the completion of the last task assigned to that resource.

Several heuristic algorithms [6, 7, 8, 9] have been proposed for grid job scheduling. These algorithms include genetic algorithms [10], particle swarm optimization [11], simulated annealing [12], ant colony optimization [13], tabu search [14], gravitational emulation local search [15] and Firefly algorithm [16]. Combinations of these algorithms known as hybrid heuristics have also been reported in the literature [17, 18].

Genetic algorithm was first proposed in 1975 by John Holland et al. [19] at Michigan University. Genetic algorithms (GAs) have also been adopted for solving the problem and obtained promising results. Hamed [20] proposed GA for task allocation in heterogeneous distributed computing system. In [21], a new evaluation algorithm, GLOA is used to solve the problem of scheduling independent task in a grid computing system. Simulation results show that this algorithm produces shorter makespan.

In the previous paper [22], A GA based task allocation problem has been discussed for multiple task allocation without considering the task’s size. This paper considers task allocation problem for a single task having some modules of different sizes with the goal of minimizing the makespan time. The experimental results reveal that the proposed algorithm produces better module allocation than other algorithms.

2. NOTATIONS

\( m \) \quad \text{Total number of modules}
\( n \) \quad \text{Total number of processors}
\( e_{ij} \) \quad \text{Execution cost of } i^{th} \text{ module on processor } j
\( c_{jk} \) \quad \text{Inter module communication cost of communicating modules } i \text{ and } k
\( r_{ij} \) \quad \text{Execution rate of processor } j
\( \text{PER} \) \quad \text{Processor's Execution Rate Vector}
\( \text{MS} \) \quad \text{Module Size vector}
\( \text{IMCCM (.)} \) \quad \text{Inter Module Communication Cost Matrix}
\( \text{ECM (.)} \) \quad \text{Execution Cost Matrix}
\( \text{TS} \) \quad \text{Task Size}
\( \text{PS} \) \quad \text{Population Size}
\( \text{MaxIter} \) \quad \text{Maximum number of Iterations}
\( s_i \) \quad \text{Size of module } i
The execution and communication costs of the processor $j$ can be computed as below:

$$\text{EXE}_j = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot X_{ij}$$  \hspace{1cm} (2)

$$\text{COMM}_j = \sum_{i=1}^{m} \sum_{k=1}^{m} \sum_{l=1}^{n} c_{ikl} \cdot X_{ij} \cdot X_{kl}$$  \hspace{1cm} (3)

where

$$X_{ij} = \begin{cases} 1, & \text{if } i\text{th module is assigned to the } j\text{th processor} \\ 0, & \text{otherwise} \end{cases}$$

$$X_{kl} = \begin{cases} 1, & \text{if } k\text{th module is assigned to the } l\text{th processor} \\ 0, & \text{otherwise} \end{cases}$$

The total cost of the processor $j$ is the sum of execution and communication costs and can be computed as below:

$$T \cos t_j = \text{EXE}_j + \text{COMM}_j$$  \hspace{1cm} (4)

The makespan of the solution is the maximum of the completion times of all the processors.

$$\text{Makespan} = \max \{ T \cos t_j \}$$  \hspace{1cm} (5)

### 3.1 Constraints

In order to determine the proper allocation, initially the average load of each processor must be determined.

If the execution cost of module $i$ on processor $j$ is $e_{ij}$, the average load on each processor is as shown in equation (6)

$$L_{avg}(j) = \frac{\sum_{i=1}^{m} e_{ij}}{n}$$  \hspace{1cm} (6)

The second constraint module allocation considers balanced utilization of a set of “n” available processors.

The number of modules to be assigned to a processor is given by

$$\left\lceil \frac{m}{n} \right\rceil$$

### 3.2 Assumptions

- In this problem cost has been taken as time.
- If more than one module is assigned to the same processor then IMCC between them is zero.
- Allocated modules to processors should not $\geq \frac{m}{n}$.

### 4. THE PROPOSED GENETIC ALGORITHM

The representation of chromosomes is necessary for solving the problem using genetic algorithm.

### 4.1 Encoding Method

Natural numbers are used for encoding the chromosomes. The chromosomes lengths are assumed to be module numbers. Every gene in the chromosome represents the processor. Figure 1 gives an allocation of $m$ modules on $n$ processors.

![The module allocation in the form of chromosome](image)

### 4.2 Initial Population

The initial population is created randomly.

### 4.3 Fitness Calculation

Fitness value is the measure based on which one can decide the fitness of solution. In this paper the fitness of solution has been measured in the form of makespan of that solution. The solution with minimum makespan is the fittest solution.

### 4.4 Genetic Operations

Before the mutation and crossover operations apply, the selection phase is first executed. The selection technique for reproduction used in this paper is based on the roulette wheel method.

#### 4.4.1 Crossover Operation

The proposed algorithm uses a one-point crossover. The crossover operation will perform if the crossover ratio ($P_c \geq 0.8$) is verified. One point is selected randomly. Then combine...
the genes of the first chromosome from first gene to the cut point and the genes of second chromosome from cut point to the last gene.

4.4.2 Mutation Operation
A point on each chromosome from the previous phase is randomly selected and then changed to a random number between 1 and m. In the proposed approach, the mutation operation will perform if the mutation ratio (Pm <= 0.1) is verified.

4.5 Terminating Condition
The algorithm terminates when the maximum number of iterations have been reached.

The mapping of the modules to processors takes place according to the following algorithm:

1) Input: Set the parameters m, n, PER, IMCCM (.), TS, PS, MaxIter, p_m, P_c.
2) Partitioning the task into modules and obtain MS.
3) Calculate ECM (.) by according to the equation (1).
4) Determine L_{avg} (j) on processor j according to the equation (6).
5) Determine the number of modules to be assigned to the processor according to \[ \left\lceil \frac{m}{n} \right\rceil. \]
6) Generate the initial population of random individuals as shown in fig.1.
7) Calculate the makespan of each individual using the equation (2), (3), (4) and (5).
8) Iter=1
9) While (Iter <= MaxIter)
10) P=1
11) While (P <= PS)
12) Genetic operations
   - Reproduction
   - Apply crossover according to p_c (p_c >= 0.8)
   - Mutate the individual according to p_m (p_m <= 0.1)
13) Calculate the makespan of modified individual.
14) P=P+1.
15) End while.
16) Set Iter=Iter+1.
17) End while.
18) Select only those Childs which satisfy the constraint.

5. RESULTS AND COMPARISON
The effectiveness of the proposed scheduling method is assessed and evaluated using makespan. Table 1 lists the parameters used in the performance study of proposed algorithm. The experiment was conducted using three processors and a task having nine modules.

Table 1: System parameters and the corresponding value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>2268</td>
</tr>
<tr>
<td>PS</td>
<td>1000</td>
</tr>
<tr>
<td>P_c</td>
<td>0.8</td>
</tr>
<tr>
<td>P_m</td>
<td>0.1</td>
</tr>
<tr>
<td>MaxIter</td>
<td>200</td>
</tr>
</tbody>
</table>

After partitioning the task’s size (TS) randomly into nine different parts, the obtained MS is:

\[
\begin{align*}
\text{m}_1 & = 240 \\
\text{m}_2 & = 300 \\
\text{m}_3 & = 190 \\
\text{m}_4 & = 301 \\
\text{m}_5 & = 225 \\
\text{m}_6 & = 255 \\
\text{m}_7 & = 232 \\
\text{m}_8 & = 245 \\
\text{m}_9 & = 280
\end{align*}
\]

MS = 

The experiment was run 10 times with different initial random population. Each run had a fixed number of 200 iterations and makespan value was recorded after each run.

The most suitable chromosome is 213321231 i.e. the modules are allocated as given below:

\[m_2, m_6, m_9 \rightarrow p_1, \quad m_1, m_5, m_7 \rightarrow p_2, \quad m_3, m_4, m_8 \rightarrow p_3\]

The execution and the communication costs obtained using the proposed algorithm are given in the table below:

<table>
<thead>
<tr>
<th>processor</th>
<th>Execution cost</th>
<th>Communication cost</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>488.48</td>
<td>663</td>
<td>1151.48</td>
</tr>
<tr>
<td>P2</td>
<td>526.93</td>
<td>651</td>
<td>1177.93</td>
</tr>
<tr>
<td>P3</td>
<td>499.75</td>
<td>680</td>
<td>1179.75</td>
</tr>
</tbody>
</table>

It is clear from the above table, that the makespan obtained by the genetic algorithm is 1179.75.

![Fig 2: Average makespan vs. Number of Iterations](image)

The average makespan of the newly generated chromosomes has been calculated after each iteration and it is observed from the Figure 2 that with the proposed genetic algorithm, optimal makespan is achieved.
In this study, the performance of the proposed algorithm is evaluated in comparison with Yadav et al. [23]. The comparison of makespan is shown in the table below:

Table 3: Comparison of makespan time obtained by proposed GA and Yadav et al. [23]

<table>
<thead>
<tr>
<th>Module allocation</th>
<th>Proposed GA</th>
<th>Yadav et al. [23]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makespan</td>
<td>213321231</td>
<td>223121133</td>
</tr>
<tr>
<td></td>
<td>1179.75</td>
<td>1268.34</td>
</tr>
</tbody>
</table>

6. CONCLUSION

In this paper, a genetic algorithm has been presented which not only minimizes the makespan time but also achieves balanced utilization of available resources by executing successfully a task consisting of several modules. The algorithm was studied by conducting more number of runs and the reason was found to be the size of population. As the population size increases, makespan converges to optimal makespan rapidly. Table 3 shows that the results obtained by genetic algorithm are much better than Yadav et al. [23]. The present model will be validated in future through developing the simulator for real environment.

7. REFERENCES


