Performance Upgradation through Task Allocation of Distributed Networks

Kapil Govil
School of Engineering & Technology, ITM University Gwalior

ABSTRACT

Normally the distributed network has to execute the tasks that shall be the more than the number of processors. The assignment problem is a case of linear programming helps to solve the problems related to tasks and processors. The problem of execution of “m” tasks to “n” processors (m > n) in a distributed networks is addressed here through a new modified tasks allocation policy for distributed networks. The model, presented in this paper allocates the tasks or modules to the processor to increase the performance and to reduce the execution time. This paper reduces the problem of allocation of tasks where number of processors is less than the number of tasks. The example mentioned in the paper has three tasks and solved it in such a way that the task \( t_1 \) processed with minimum time, the task \( t_2 \) with minimum cost while the task \( t_3 \) with maximum reliability. In this problem, the tasks are fused (or clubbed) with another task(s) on the basis of minimum communication cost to form a balanced allocation.

Keywords

Allocation, Cost, Distributed Network, Performance, Processor, Reliability, Task, Time.

1. INTRODUCTION

Such type of research problems in which the performance of the distributed systems is to be upgraded, requires either processing time or cost to be minimized or reliability to be maximized by deciding the strategy of allocation of tasks to the processors of the distributed systems. These problems may be categorized as static (15, 16, 17, 19, 29) and dynamic (2, 14, 18, 19, 26, 27) in nature. Some of the other related methods have been reported in the literature, such as, Integer programming (7, 23), Branch and Bound technique (28), Matrix Reduction technique (11, 30, 31), Reliability Optimization (1, 12, 20, 21, 24), Load Balancing (2, 9, 10) and Modeling (3, 6, 8). The series parallel redundancy-allocation problem has been studied with different approaches, such as, Dynamic programming (4, 10, 13), Integer programming (7, 23), and Heuristic techniques (5, 22, 25).

2. OBJECTIVE

The objective of the present research paper is to enhance the performance of the distributed systems by using the proper utilization of its processors. A set of tasks have to be processed by the processors of the network, while each of the task have the modules and the number of modules are more than the number of processors of the network. The processing of a task is means that all of its modules get processed. Performance is the measure in term of either time or cost or reliability of the modules of a task that have to process on the processors of the system and these have to be optimally processed i.e., either time or cost to be minimized or reliability to be maximized.

3. TECHNIQUE

To evaluate the optimal time or cost or reliability for each task through optimal allocation, initially it has to concentrate on those \((m-n)\) modules that have the highest probability of data transfer with the remaining \(n\) modules. Each of these \((m-n)\) modules (say \(m_k\)) of every task is treated as a candidate to be fused with any one (say \(m_l\)) of the remaining \(n\) modules with which it has the highest communication. Further, all the elements of \(k^{th}\) row and \(l^{th}\) row are to be added in case of time and cost while in case of reliability these rows have to multiply. This will reduces the effectiveness matrix for each task in to a square matrix. Now the problem remains to determine the optimal time or cost or reliability through the allocation strategy by considering either task processing based on time or cost or reliability for all modules to individual processor(s) for each task. For allocation purpose a modified version of row and column assignment method proposed by Kumar et al (17) is employed which allocates all the modules of a task to a processor optimally. The functions for obtaining the overall assignment execution time [\(E_{time}\)], execution cost [\(E_{cost}\)], and execution reliability [\(E_{reliability}\)] are as follows:

\[
P_{time} = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} PT_{ij}x_{ij} \right) \right]
\]

(1)

\[
P_{cost} = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} E_{C_{ij}}x_{ij} \right) \right]
\]

(2)

\[
P_{reliability} = \left[ \prod_{i=1}^{n} \left( \sum_{j=1}^{n} E_{R_{ij}}x_{ij} \right) \right]
\]

(3)

Where, \(x_{ij} = \begin{cases} 1, \text{if i}^{th} \text{ task is assigned to j}^{th} \text{ processor} \\ 0, \text{otherwise} \end{cases}\)

4. ALGORITHM

Step 1: Start algo
Step 2: Read the number of tasks in \(m\)
Step 3: Read the number of processors in \(n\)
Step 4: For \(I = 1\) to \(n\)
Step 5: For \(J = 1\) to \(m\)
Step 6: Read the value in \(PTM [I][J]\)
Step 7: Increase the value of \(J\) by 1
Step 8: End of \(J\) loop
Step 9: Increase the value of \(I\) by 1
Step 10: End of \(I\) loop
Step 11: For \(I = 1\) to \(n\)
Step 12: For \(J = 1\) to \(n\)
Step 13: Read the value in CM[I][J]
Step 14: Increase the value of J by 1
Step 15: End of J loop
Step 16: Increase the value of I by 1
Step 17: End of I loop
Step 18: For I = 1 to n
Step 19: For J = I to n
Step 20: If CM[I][J] == 1 then
Step 21: Store the value of P1[I] to 1
Step 22: Store the value of P2[J] to 1
Step 23: Calculate MAT[I][J] = PT/C/RM[P1[I]] + PT/C/RM[P2[J]]
Step 24: End of if statement
Step 25: Increment the value of J by 1
Step 26: End of J loop
Step 27: Increase the value of I by 1
Step 28: End of I loop
Step 29: For I = 1 to n
Step 30: If P1[I] ==0 then
Step 31: Store the value of T1[I] by P1[I]
Step 32: End of if statement
Step 33: Increase the value of I by 1
Step 34: End of I loop
Step 35: For I = 1 to n
Step 36: If T1[I] !=0 then
Step 37: Calculate MAT[T1[I]] = PT/C/RM[T1[I]] + PT/C/RM[T1[I+1]]
Step 38: End of if statement
Step 39: Increment the value I by 1
Step 40: End of I loop
Step 41: Count the zero(s) in each row
Step 42: Mark the row(s), which have single zero
Step 43: Mark the column, which have single zero
Step 44: Go to the row(s), which have more than one zero. Now select any one zero and cross the leading zero(s), which are in the same row and column
Step 45: Mark the assignments
Step 46: Count the total assignment
Step 47: If total number of assignment < order of matrix
Step 48: Go to Step 52
Step 49: Else
Step 50: Go to Step 59
Step 51: End of if statement
Step 52: Mark the rows for which assignment have not been made
Step 53: Mark column that have zeros in marked rows
Step 54: Mark rows that have assignment in marked column
Step 55: Repeat Step 53 & Step 54 until chain of marking ends
Step 56: Draw the minimum number of lines through unmarked rows and marked columns to cover all zeros
Step 57: Select the smallest element of the uncovered elements and replace it by zero. Also add this element to positions at which lines intersect to each other only
Step 58: Go to Step 42
Step 59: State processing time
Step 60: End algo

5. IMPLEMENTATION

Consider an example consisting of a set T = \{t_1, t_2, t_3\} of 3 tasks each of them having sets M_1 = \{m_{11}, m_{12}, m_{13}, m_{14}, m_{15}\} of 5 modules, M_2 = \{m_{21}, m_{22}, m_{23}, m_{24}\} of 4 modules and M_3 = \{m_{31}, m_{32}, m_{33}, m_{34}, m_{35}, m_{36}\} of 6 modules respectively. The three processors are available in the distributed network to process the tasks that are represented by the set P = \{p_1, p_2, p_3\}. The processing time (t), cost (c) and reliability (r) of each module of every task on various processors are known and mentioned in the following matrix, namely, PCTR (t, c, r):
The communication period amongst the modules of each task has also been considered and it is mentioned in the following matrices, namely, \( \text{CM}(1, \cdot) \):

For task \( t_2 \), the matrix \( \text{CM}(1, \cdot) \) is as:

\[
\begin{bmatrix}
 m_{11} & m_{12} & m_{13} & m_{14} & m_{15} \\
 m_{11} & 0 & 1 & 6 & 9 & 3 \\
 m_{12} & 0 & 2 & 7 & 8 \\
 m_{13} & 0 & 4 & 5 \\
 m_{14} & 0 & 2 \\
 m_{15} & 0
\end{bmatrix}
\]

For task \( t_3 \), the matrix \( \text{CM}(2, \cdot) \) is as:

\[
\begin{bmatrix}
 m_{21} & m_{22} & m_{23} & m_{24} \\
 m_{21} & 0 & 2 & 4 & 5 \\
 m_{22} & 0 & 7 & 8 \\
 m_{23} & 0 & 6 \\
 m_{24} & 0
\end{bmatrix}
\]

For task \( t_5 \), the matrix \( \text{CM}(3, \cdot) \) is as:

\[
\begin{bmatrix}
 m_{31} & m_{32} & m_{33} & m_{34} & m_{35} & m_{36} \\
 m_{31} & 0 & 2 & 3 & 7 & 9 & 6 \\
 m_{32} & 0 & 4 & 8 & 6 & 5 \\
 m_{33} & 0 & 1 & 5 & 4 \\
 m_{34} & 0 & 2 & 3 \\
 m_{35} & 0 & 1 \\
 m_{36} & 0
\end{bmatrix}
\]

Here, it is considered the processing of the tasks \( t_3 \) based on the time constraints (however one may choose the cost or reliability constraints also); \( t_2 \) is based on the cost constraints (however one may choose the time or reliability constraints also); and for the \( t_5 \), it is based on reliability constraints (however one may choose the time or cost constraints also). Further it is also noted that each task has modules that are more than the number of processors in the distributed system. So following data from the matrix \( \text{PCTR}(1, \cdot) \) is used i.e,
The task $t_1$ has five modules, so that on the basis of highest communication, the modules $m_{11}$ & $m_{14}$ and $m_{12}$ & $m_{15}$ are fused together to reduce the effectiveness matrix square. The task $t_2$ has four modules, so that on the basis of highest communication, the modules $m_{21}$ & $m_{24}$ are fused together to reduce the effectiveness matrix square. The task $t_3$ has six modules, so that on the basis of highest communication, the modules $m_{11}$ & $m_{15}$, $m_{12}$ & $m_{13}$ and $m_{14}$ & $m_{16}$ are fused together to reduce the effectiveness matrix square. The resulting matrix is as:

$$
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}
\hline
\text{Processors} & p_1 & p_2 & p_3 \\
\hline
\text{Tasks} & \text{Modules} & t-c-r & t-c-r & t-c-r \\
\hline
 t_1 & m_{11} & 070 & 110 & 120 \\
 & m_{12} & 080 & 090 & 130 \\
 & m_{13} & 120 & 135 & 140 \\
 & m_{14} & 160 & 140 & 150 \\
 & m_{15} & 170 & 130 & 160 \\
 t_2 & m_{21} & \cdots 2500 & \cdots 2200 & \cdots 2300 \\
 & m_{22} & \cdots 2100 & \cdots 2800 & \cdots 2900 \\
 & m_{23} & \cdots 2700 & \cdots 2700 & \cdots 2300 \\
 & m_{24} & \cdots 2400 & \cdots 2600 & \cdots 2800 \\
 t_3 & m_{31} & \cdots -0.999452 & \cdots -0.999625 & \cdots -0.999856 \\
 & m_{32} & \cdots -0.999466 & \cdots -0.999653 & \cdots -0.999785 \\
 & m_{33} & \cdots -0.999256 & \cdots -0.999245 & \cdots -0.999631 \\
 & m_{34} & \cdots -0.999215 & \cdots -0.999123 & \cdots -0.999563 \\
 & m_{35} & \cdots -0.999452 & \cdots -0.999365 & \cdots -0.999878 \\
 & m_{36} & \cdots -0.999785 & \cdots -0.999632 & \cdots -0.999562 \\
\hline
\end{array}
$$

The results of the allocations based on time for the task $t_1$ are obtained after implementing the row & column assignment process as suggested by Kumar et al (17), are mentioned below in the Table 1:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Time</th>
<th>Etime</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{11} \ast m_{14}$</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>$m_{12} \ast m_{15}$</td>
<td>220</td>
<td>590</td>
</tr>
<tr>
<td>$m_{13}$</td>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Time based Allocation for task $t_1$

The results of the allocations based on cost for the task $t_2$ are obtained after implementing the row & column assignment process as suggested by Kumar et al (17), are mentioned below in the Table 2:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Cost</th>
<th>Ecot</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{22} \ast m_{24}$</td>
<td>2200</td>
<td></td>
</tr>
<tr>
<td>$m_{23}$</td>
<td>4500</td>
<td>9000</td>
</tr>
<tr>
<td>$m_{23}$</td>
<td>2300</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Cost based Allocation for task $t_2$
The results of the allocations based on reliability for the task $t_1$ are obtained after implementing the row & column assignment process as suggested by Kumar et al (17), are mentioned below in the Table 3:

Thus the complete results for the above mentioned example obtained and are mentioned in the Table 4.

### Table 4. Optimal Allocation Table

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Optimal</th>
<th>Optimal</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$m_1 \times m_3$</td>
<td>$m_2 \times m_3$</td>
<td>$m_3$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$m_1 \times m_4$</td>
<td>$m_2 \times m_4$</td>
<td>$m_4$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$m_5 \times m_5$</td>
<td>$m_6 \times m_6$</td>
<td>$m_6$</td>
</tr>
</tbody>
</table>

#### 6. CONCLUSION

This paper chooses the problem, in which the numbers of module of the tasks are more than the number of processors of the distributed system. The model addressed in this paper is based on the consideration of processing time, cost and reliability of the module of the tasks to the various processors. The communication period amongst the module of the tasks is also used. The method is presented in algorithmic form and implemented on the several sets of input data to test the performance and effectiveness of the algorithm. As it is the common requirement for any assignment that the tasks have to be processed either with minimum time or minimum cost or maximum reliability. The example mentioned in this paper has three tasks and solved it in such a way that the task $t_1$ processed with minimum time, the task $t_2$ with minimum cost while the task $t_3$ with maximum reliability. The optimal results are mentioned in Table 4.

### Table 5(a). Optimal results for task $t_1$

<table>
<thead>
<tr>
<th>Processor</th>
<th>Assignment based on</th>
<th>Optimal Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$m_{11} \times m_{14}$</td>
<td>590</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$m_{12} \times m_{14}$</td>
<td>6000</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$m_{13} \times m_{15}$</td>
<td>0.996929</td>
</tr>
</tbody>
</table>

### Table 5(b). Optimal results for task $t_2$

<table>
<thead>
<tr>
<th>Processor</th>
<th>Assignment based on</th>
<th>Optimal Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$m_{21} \times m_{24}$</td>
<td>540</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$m_{22} \times m_{24}$</td>
<td>9000</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$m_{23} \times m_{24}$</td>
<td>0.998442</td>
</tr>
</tbody>
</table>

### Table 5(c). Optimal results for task $t_3$

<table>
<thead>
<tr>
<th>Processor</th>
<th>Assignment based on</th>
<th>Optimal Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$m_{31} \times m_{34}$</td>
<td>610</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$m_{32} \times m_{34}$</td>
<td>12100</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$m_{33} \times m_{34}$</td>
<td>0.9966876</td>
</tr>
</tbody>
</table>

#### 7. TIME COMPLEXITY

It is known that the analysis of an algorithm is mainly focuses on time complexity. Time complexity is a function of input size ‘n’. It is referred to as the amount of time required by an algorithm to run to completion. The time complexity of the above mentioned algorithm is $O(m^2n^2)$. By taking several input examples, the above algorithm returns results as mentioned in Table 6.

### Table 6. Time Complexity

<table>
<thead>
<tr>
<th>No. of processors (n)</th>
<th>No. of tasks (m)</th>
<th>Optimal Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>144</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>225</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>324</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>441</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>576</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>576</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>784</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>1024</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>1296</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>900</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1225</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>1600</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>2025</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2500</td>
</tr>
</tbody>
</table>

The graphical representations of the results are shown by Fig 1, 2 and 3.
8. COMPLEXITY COMPARISION
The performance of the algorithm is compared with the algorithm suggested by Richard et al (28). Following Table 7 shows the time complexity comparison between algorithm (28) with present algorithm.

Table 7. Complexity Comparison

<table>
<thead>
<tr>
<th>Processors</th>
<th>Tasks</th>
<th>Time Complexity of algorithm (28) (O(n^m))</th>
<th>Time Complexity of present algorithm (O(m^2n^2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Processors = 3</td>
<td>No. of Tasks = 4</td>
<td>81</td>
<td>144</td>
</tr>
<tr>
<td>No. of Processors = 3</td>
<td>No. of Tasks = 5</td>
<td>243</td>
<td>225</td>
</tr>
<tr>
<td>No. of Processors = 3</td>
<td>No. of Tasks = 6</td>
<td>729</td>
<td>324</td>
</tr>
<tr>
<td>No. of Processors = 3</td>
<td>No. of Tasks = 7</td>
<td>2187</td>
<td>441</td>
</tr>
<tr>
<td>No. of Processors = 3</td>
<td>No. of Tasks = 8</td>
<td>6561</td>
<td>576</td>
</tr>
<tr>
<td>No. of Processors = 4</td>
<td>No. of Tasks = 5</td>
<td>1024</td>
<td>400</td>
</tr>
<tr>
<td>No. of Processors = 4</td>
<td>No. of Tasks = 6</td>
<td>4096</td>
<td>576</td>
</tr>
<tr>
<td>No. of Processors = 4</td>
<td>No. of Tasks = 7</td>
<td>16384</td>
<td>784</td>
</tr>
<tr>
<td>No. of Processors = 4</td>
<td>No. of Tasks = 8</td>
<td>65536</td>
<td>1024</td>
</tr>
<tr>
<td>No. of Processors = 5</td>
<td>No. of Tasks = 6</td>
<td>15625</td>
<td>900</td>
</tr>
<tr>
<td>No. of Processors = 5</td>
<td>No. of Tasks = 7</td>
<td>78125</td>
<td>1225</td>
</tr>
<tr>
<td>No. of Processors = 5</td>
<td>No. of Tasks = 8</td>
<td>390625</td>
<td>1600</td>
</tr>
<tr>
<td>No. of Processors = 5</td>
<td>No. of Tasks = 9</td>
<td>1953125</td>
<td>2025</td>
</tr>
<tr>
<td>No. of Processors = 5</td>
<td>No. of Tasks = 10</td>
<td>9765625</td>
<td>2500</td>
</tr>
</tbody>
</table>

From the Table 7 it is clear that present algorithm is much better for optimal allocation of tasks that upgrade the performance of distributed system. The graphical representation of the data as mentioned in Table 7 is shown through Fig 4, 5 and 6.

9. REFERENCES