Evaluation of Improved Reliability of Distributed Computing Systems Using Mathematical Programming Approach

Avanish Kumar,1* P.K. Yadav2 and Mudit Bansal1
1Department of Mathematical Sciences and Computer Applications 
Bundelkhand University, Jhansi - 284 128
2Research Planning and Business Development
Central Building Research Institute, Roorkee - 247 667

Abstract: Reliability is the extent to which a measurement procedure yields the same results on repeated trials. Without reliable measures, scientists cannot build or test theory and, therefore, cannot develop productive and efficient procedures for improving the quality of life. Reliability analysis for any distributed computing system is the current necessity and most important area of research. There are number of ways to improve the reliability of the distributed computing system. Optimising the reliability through task allocation is one of the problems in this category. Reliability is the assessment we make of how much measurement error we have experienced in processing our data. The problem presented in this article is based on the consideration of execution unreliability of the tasks to the processors. The main objective of this problem is to minimise the overall processing unreliability by allocating the tasks optimally to the processors of the distributed computing system using the method of matrix partitioning. Several sets of input data are considered to test the complexity and efficiency of the algorithm. It is found that the algorithm is suitable for arbitrary number of processors with the random program structures and workable in all the cases.

Keywords: Mathematical programming approach, Task allocation, Distributed computing systems.

Introduction
The main incentives for choosing distributed computing systems are higher throughput, improved reliability and better access to a widely communicated web of information. The increased commercialisation of communication computing systems means that ensuring system optimal

*Author for Correspondence. E-mail: dravanishkumar@yahoo.com
reliability is of critical importance. Inherently, a distributed computing system is more complex, therefore, it is very difficult to predict its performance. Mathematical modelling is the tool which plays an important role to predict the performance of distributed computing systems. Over the last several years, distributed computing systems have become popular as a very attractive option for fast computing and information processing. Distributed computing system is used to describe a system whenever there are several computers interconnected in some fashion so that a program or procedure runs on the system with multiple processors. However, it has different meanings to different systems because processors can be interconnected in many ways for various reasons. In the most general form, the word distribution implies that the processors are in geographically separate locations. Occasionally, it is also applied to an operation using multiple mini-computers, which are not a part of hardware, connected with each other and can also be connected through satellite. A user-oriented definition of distributed computing is stated [1, 2] as “The Multiple Computers utilised cooperatively to solve problems i.e. to process and maintain the large scale database of the programs which are to be executed on these types of systems”. Some static [3-7] and dynamic [8-11] task allocation techniques have contributed a lot to development in software engineering as an independent area of research. Scheduling in general purpose distributed system 9] and dual processor and multi-processor scheduling with dynamic re-assignment [8] have drawn considerable attention of the researchers. Several methods have been reported in the literature, e.g. integer programming [4, 12], critical delays consideration [13] and reliability evaluation to deal with various design and allocation issues in a distributed computing system. The study of complexity of the problem of assigning the modules to the processors in computing distributed computing systems to minimise the total computational time has also discussed [9]. The other two similar algorithms [14, 15] have been reported in the literature. Peng et al. [14] used matrix reduction technique. According to the criteria given therein, a task is selected randomly to start with and then assigned to a processor. Richard et al. [15] used branch and bound method for assignment and scheduling communicating periodic tasks in a distributed computing system. Kim and Browne [16] gave an efficient method for optimal task allocations in distributed computing systems owing to inter task communication effects. Due to numerous advances in the field of communication networks, a wide variety of networks have come into existence. Ethernet is often used in local area networks to allow independent devices to interconnect with one another within a relatively small physical location [17, 18]. Asynchronous Transfer Mode has been proposed for wide area networks to integrate a variety of data communication services, including voice, video and plain text [18-20]. The high-performance parallel interface was originally developed to allow mainframes and supercomputers to communicate at a very high-speed [20]. A number of heuristic methods like near clustering [21, 22] and hierarchical clustering [23, 24] perform execution and scheduling to minimise the schedule length. Module redundancy has been used for fault detection and masking [25] by running two or more copies of the module. Module execution is done in a distributed computing system dynamically. However, if each copy of the module is allowed to complete then the degradation in throughput can be substantial. The main objective of this article is to maximise the total program execution reliability by allocating the tasks in such a way that the allocated load on each processor should be balanced. For this purpose, we consider the execution unreliability of each task to each processor and design the allocation policy in such a way that total execution unreliability is minimised. The model utilised the mathematical programming technique for execution
of the module considering that each module is to be executed by the processor. The model addressed in this article is based on the consideration of execution unreliability (EUR) of the tasks to the processors. We suggested a modified method to assign all the tasks as per the required availability of processors so that none of the tasks remains unexecuted. Also, the present approach does not require adding dummy processors. The EURs presented by arrays in the form of execution unreliability matrix (EURM(.,)) of order $m \times n$. The developed model is programmed in C and implemented. Several sets of input data are used to test the effectiveness and efficiency of the algorithm.

Objective
The objective of this problem is to maximise the total execution reliability by allocating the tasks optimally. For that purpose we considered the execution unreliability and used the method of matrix partitioning. This problem used a modified method to assign all the tasks as per the required availability of processors so that none of the tasks remains unexecuted even when the number of tasks is more than number of processors. The EUR is presented by arrays in the form of EURM(.,) of the order $m \times n$. A procedure is then formulated to assign the tasks to the processors of the distributed computing systems based on the execution unreliability and is to be designed in such a way that the overall reliability is optimised under the pre-specified constraints.

Technique
For developing the technique we have formed the sub-problems using matrix partitioning i.e. each of the sub-problems is of the type of balanced assignment problem and devised a method to assign all tasks to the processors and formulating the unreliability function to measure EUR. Since the number of tasks is more than the number of processors, we divide the problem of unbalanced tasks assignment into sub-problems using matrix partitioning. This gives us balanced tasks assignment problems. First, we obtain the product of each row and each column except the position where the unreliability is zero (zero unreliability should be kept aside from the product of row or column) from the EURM(.,) and store the results into Product_Row() and Product_Column(), each of them are one dimensional arrays. Select the first set of tasks (this set of tasks shall contain only as many tasks as the number of processors in the distributed computing system) on the basis of minimum unreliability against the tasks in the Product_Row() array. Store the result into EURM (.,.) a two-dimensional array. Repeat the process until remaining tasks are either less than or equal to the number of processors. If the tasks are equal to the number of processors of the computing system then it will become the last sub-problem, else to form the last problem we have to delete the column (processor) from EURM(.,.) on the basis of Product_Column( . ) array i.e. this set shall contain only as many processors as number of tasks left, so that we delete the processors that have maximum unreliability in the Product_Column() array. For allocation purpose a modified version of row and column assignment method devised by Kumar et al. [20] is employed which allocates a task to a processor where it has minimum execution unreliability. For each sub-problem, calculate the execution unreliability of each processor and store the result in a linear array PEUR(j) where $j = 1, 2, \ldots, n$. 

GAMS JOURNAL OF MATHEMATICAL BIOSCIENCES Vol. 3 No. 1 January-June 2007
\[ \text{PEUR}(j) = \prod_{i=1}^{n} \left[ \sum_{j=1}^{n} \text{eur}_{ij} \cdot x_{ij} \right] \]

where
\[ x_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is assigned to } j^{th} \text{ processor} \\ 0, & \text{otherwise} \end{cases} \]

**Algorithm**
Consider a distributed computing system which consists of a set of \( n \) processor, \( P = \{p_1, p_2, \ldots, p_n\} \) and a set of \( m \) executable tasks \( T = \{t_1, t_2, \ldots, t_m\} \) which are to be processed by any one of the processors of the system.

**Step 1**
Input: \( m, n, \text{EURM}(\cdot) \).

**Step 2**
Obtain the product of each row of the EURM(\( \cdot \)) in such a way that, if any unreliability(ies) is (are) zero then keep it aside from product amount of that row (just to avoid the condition EUR* 0 = 0). Store the results in one-dimensional array Product_Row (\( \cdot \)) of order \( m \).

**Step 3**
Obtain the product of each column of the EURM(\( \cdot \)), in such a way that if any unreliability(ies) is (are) zero then keep it aside from product of that column (just to avoid the condition EUR* 0 = 0). Store the results in one-dimensional array Product_Column (\( \cdot \)) of order \( n \).

**Step 4**
Partition the execution unreliability matrix EURM(\( \cdot \)) of order \( m \times n \) to sub-matrices such that the order of these matrices become square. Partitioning is to be made as follows:

**Step 4.1**
Select the \( n \) task on basis of Product_Row (\( \cdot \)) array i.e. select the ‘\( n \)' task corresponding to the most minimum product to next minimum product, if there is a tie select arbitrarily. (For the cases in which product is EUR * 0, minimum value depends only on EUR and the impact of zero is to be neglected.)

**Step 4.2**
Store the result in the two-dimensional array EURM(\( \cdot, \cdot \)) to form the sub-matrices of the sub-problems.

**Step 4.3**
If all the tasks are selected then go to step 4.7 else step 4.4.

**Step 4.4**
Repeat the steps 4.1 to 4.3 until the number of tasks become less than \( n \).
Step 4.5
Select the remaining task say \( r, \ r < n \), select the \( r \) processors on the basis of Product_Column (,) array i.e. the processors corresponding to the most maximum product to next maximum and if there is a tie, select arbitrarily (for the cases in which product is EUR * 0, maximum value depends only on EUR and the impact of zero is to be neglected).

Step 4.6
Store the result in the two-dimensional array EURM(,,), which is the last sub-problem.

Step 4.7
List all the sub-problems formed through steps 4.1 to 4.6 and repeat steps 5 to 12 to solve each of these sub-problems.

Step 5
Find the minimum of each row of EURM(,,) and replace it by 0.

Step 6
Find the minimum of each column of EURM(,,) and replace it by 0.

Step 7
Search for a row in EURM(,,), which has only one zero and assign the task(s) corresponding to this position. Add one to the counter i.e. nar = nar + 1 and store this position.

Step 8
Search for a column in EURM(,,), which has only one zero and assign the task(s) corresponding to this position. Add one to the counter that is nar = nar + 1 and also store this position.

Step 9
Check whether nar = \( n \), if not then pickup an arbitrary 0 and assign task(s) corresponding to this position. Add one to the counter i.e. nar = nar + 1 and also store this position, else, check column(s) position of 0's in unassigned row(s). Check the row(s) for any previous assignment in the corresponding column(s). Find the minimum of the entire elements for the remaining rows and replace it by zero.

Step 10
Evaluate Execution Unreliability.

Step 11
Execution reliability (Ereliability) is, thus, calculated as:
Ereliability = 1 - EUR

Step 12
Stop.
Implementation
Consider a system which consists of a set $T = \{t_1, t_2, t_3, t_4, t_5\}$ of 5 tasks and a set $P = \{p_1, p_2, p_3\}$ of 3 processors, where

Step 1
Input: 5, 3

<table>
<thead>
<tr>
<th>EMRU(,)</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>0.03</td>
<td>0.07</td>
<td>0.02</td>
<td>0.03</td>
<td>0.08</td>
</tr>
<tr>
<td>$p_2$</td>
<td>0.04</td>
<td>0.02</td>
<td>0.09</td>
<td>0.07</td>
<td>0.05</td>
</tr>
<tr>
<td>$p_3$</td>
<td>0.06</td>
<td>0.08</td>
<td>0.06</td>
<td>0.02</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Step 2
Obtain the product of each row and column of EURM(,), i.e. the products of each row and each column are as follows:

$$\text{Product Row}() = \begin{bmatrix}
    t_1 & t_2 & t_3 & t_4 & t_5 \\
    0.000072 & 0.000112 & 0.000108 & 0.000042 & 0.000160
\end{bmatrix}$$

$$\text{Product Column}() = \begin{bmatrix}
    p_1 & p_2 & p_3 \\
    0.0000001008 & 0.0000002520 & 0.0000002304
\end{bmatrix}$$

Step 3
We partitioned the matrix EURM(,,) to define the first sub-problem of EURM(,,) by selecting rows corresponding to $t_2$, $t_3$, $t_5$ and second sub-problem of EURM(,,) by selecting rows corresponding to the tasks $t_1$, $t_4$ and by deleting columns corresponding to $p_1$. The modified matrices for each sub-problem are:

**Sub-problem I**

<table>
<thead>
<tr>
<th>EURM(1,,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
</tr>
<tr>
<td>0.07</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.08</td>
</tr>
</tbody>
</table>

**Sub-problem II**

<table>
<thead>
<tr>
<th>EURM(2,,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
<tr>
<td>0.04</td>
</tr>
<tr>
<td>0.07</td>
</tr>
</tbody>
</table>
Steps 4 and 5
We apply modified Hungarian method devised by Kumar et al. [20] to assign the tasks, on the basis of min \( r_{ij} \) and min \( c_{ij} \) from unreliability matrices for every \( i \) and \( j \). We put \( r_{ij} = 0 \) and \( c_{ij} = 0 \), for every \( i \) and \( j \). On applying this to all sub-problems, the modified matrices for each sub-problem are given as:

\[
\begin{align*}
\text{EURM}(1,1) &=
\begin{pmatrix}
0.07 & 0.00 & 0.08 \\
0.00 & 0.09 & 0.06 \\
0.08 & 0.05 & 0.00 \\
\end{pmatrix} \\
\text{EURM}(2,2) &=
\begin{pmatrix}
0.00 & 0.06 \\
0.07 & 0.00 \\
\end{pmatrix}
\end{align*}
\]

Steps 6, 7 and 8
After implementing assignment process, the allocation is thus obtained.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>( \rightarrow )</th>
<th>Processors ( \rightarrow )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( \rightarrow )</td>
<td>( P_1 )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( \rightarrow )</td>
<td>( P_2 )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( \rightarrow )</td>
<td>( P_2 )</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( \rightarrow )</td>
<td>( P_3 )</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>( \rightarrow )</td>
<td>( P_3 )</td>
</tr>
</tbody>
</table>

EUR = 0.0000000128
Ereliability = 0.9999999872

Steps 9, 10 and 11

Step 12
Stop.

Conclusion
The algorithm presented in this article is capable of maximising the overall reliability of distributed computing systems through task allocation. In distributed computing system, tasks are allocated in such a way that their individual reliability of processing is optimised as well as it improves the overall reliability. In this approach, not only the loads of each processor get equally balanced but also none of the task gets unprocessed. This problem provides an optimal solution for assigning a set of \( m \) tasks of a program to a set of \( n \) processors where \( m > n \) in a distributed computing system, i.e. to maximise the overall reliability of the system, the load of all allocated tasks on all the processors is equally balanced. The execution unreliability on different processors has been obtained.
<table>
<thead>
<tr>
<th>Processors</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution unreliability of each processor PEUR (,)</td>
<td>0.02</td>
<td>0.0008</td>
<td>0.0008</td>
</tr>
<tr>
<td>Total reliability [Ereliability(,)] i.e. (1-EUR(,))</td>
<td></td>
<td></td>
<td>0.9999999872</td>
</tr>
</tbody>
</table>

The final result of the example is:

Tasks $\rightarrow$ Processors

\[
\begin{align*}
  t_3 & \rightarrow p_1 \\
  t_1 \times t_2 & \rightarrow p_2 \\
  t_4 \times t_5 & \rightarrow p_3
\end{align*}
\]

This problem provides the optimal solution for improving the reliability and also deals with the performance on the basis of maximising reliability.

References

Evaluation of Optimal Execution Time of Computer System by exhaustive search approach

The Allocation problems in any computer system play the key role for deciding the performance of the system. The allocation put the direct impact of software resources as well as hardware resources. In heterogeneous Computer systems, partitioning of the application software in to module and the proper allocation of these modules dissimilar processors are important factors, which determine the efficient utilization of resources. The static model discussed in this paper provide an optimal solution for assigning a set of \( m \) modules of a task to a set of \( n \) processors where \( m > n \) in a computer system by using an Exhaustive Search Approach for evaluation for optimal time of the system. The Impact of Communication Time has also been taken into consideration.

DR. P.K. YADAV*, DR. AVANISH KUMAR** & MUDIT BANSAL**

Abstract:

The Allocation problems in any computer system play the key role for deciding the performance of the system. The allocation put the direct impact of software resources as well as hardware resources. In heterogeneous Computer systems, partitioning of the application software in to module and the proper allocation of these modules dissimilar processors are important factors, which determine the efficient utilization of resources. The static model discussed in this paper provide an optimal solution for assigning a set of \( m \) modules of a task to a set of \( n \) processors where \( m > n \) in a computer system by using an Exhaustive Search Approach for evaluation for optimal time of the system. The Impact of Communication Time has also been taken into consideration. Approach suggested in the paper defines an index based on processing time along with the communication time of the computer system. The Execution time of the tasks has been taken in the form for matrix of order \( m \times n \) named as ETM \( (\cdot) \) and communication time has been also been taken in the form of symmetric matrix of order \( m \) named as CTM \( (\cdot) \). The several sets of input data are considered to test the complexity and efficiency of the algorithm. It is found that the algorithm is suitable for arbitrary number of processor with the random program structure and workable in all the cases.

Keywords:
Optimization, Computer System, Execution Time, Task Allocation.

(1) Introduction:

Task allocation is the process of partitioning a set of programming modules into a number of processing groups, known as tasks, where each group executes on a separate processor. The general allocation problem is NP-complete.

To obtain an easy solution, it is essential to make use of heuristics approach that provides a near optimal solution in a reasonable amount of time. Hungarian approach is suggested for solving the assignment problem, but this approach is to be applicable only to \( m \)-tasks to \( m \)-processors i.e. for balanced Assignment problem. For the unbalanced assignment problem where \( m > n \), the Hungarian method suggest to add the dummy tasks/processors to make the effectiveness matrix square. Distributed computing systems have homogenous and/or heterogeneous processors that are connected through a communication network. It provides the capability for the utilization of remote computing resources and allow for increased level of flexibility, reliability, and modularity. Assignment in distributed system has some major advantage. Computer systems are being converted from host-centralized system to distributed systems, represented by client server model, through corporate efforts in downsizing and rightsizing. People expect much more extendibility of distributed system as workstation and personal computers are becoming more powerful and inexpensive, application which run on distributed system are increasing, and the number of users also increasing. The best-known research problem for such systems is the allocation problem, in which either system reliability is maximized or total system cost is minimized. These problems may be categorized in static [1,5,10-12,17,20,21-22,24-25] and dynamic [3-4,9,13] assignment problems. Several other methods have been reported in the literature, such as, Integer programming [7], Branch and bound technique [18]. Matrix reduction technique [20], Reliability optimization [15-16], Modeling [2,8]. The series parallel redundancy-allocation problem has been studied with different approaches, such as, Non-linear techniques [23], and Heuristic techniques [6].

*RPBD, Central Building Research Institute, Roorkee (UA)
**Department of Mathematical Sciences & Computer Applications, Bundelkhand University, Jhansi (UP)
The present work suggests an exhaustive search approach for the allocation problem through index optimization. The index is based on the time of the tasks to the processors for their execution to processors and also time of communication amongst the tasks.

(2) Definitions:

**Execution Time:**

Each task \( t_i \) has an Execution Time \( [ET]_i \) \( 1 \leq i \leq m \) and \( 1 \leq j \leq n \), which is the time of processing the task \( t_i \) on the processor \( p_j \).

**Communication Time:**

The Communication Time \( [CT]_i,j \) \( (1 \leq i \leq m \text{ and } 1 \leq j \leq n) \) of the interacting tasks \( t_i \) and \( t_j \) is incurred due to the data units exchanged between them during the process of execution.

**Distributed Systems:**

A distributed system is characterized by having processors functions, the database, or system control physically dispersed and interconnected by communication facilities.

**Assignments:**

Assignment is a process in which any type of activity perform to any type of resource and time for a resource to complete an activity could be considered the effectiveness associated with using a given type of resource on a given activity.

(3) Assumptions:

The completion of a program from computational point of view means that all related tasks have got executed and the final output has been generated after integrating the respective outputs of individual programs.

Some of the tasks have restrictions for being allocated on some particular processor(s) because of the non-availability of desired facilities, which may include, access to particular peripheral device, and high-speed arithmetic capability of the processor. The execution time of such tasks is taken to be infinite, on those processors where these tasks cannot be executed.

Whenever two or more tasks are assigned to the same processor, the communication time between them is assumed to be zero.

The number of tasks to be allotted is more than the number of processors.

(4) Objective:

Let the given system consists of a set \( P = \{p_1, p_2, p_3, p_4, \ldots p_n\} \) of \( n \) processors, interconnected by communication links and a set \( T = \{t_1, t_2, t_3, t_4, \ldots t_m\} \) of \( m \) tasks. The processing execution time of individual tasks corresponding to each processor are given in the form of matrix \( ETM(\cdot) \) order \( m \times n \). The communication time is taken in the square symmetric matrices \( CTM(\cdot) \) of order \( n \) respectively. The functions to measure ET, and CT are then formulated. A procedure to assign the tasks to the processors of the distributed systems based on execution time is to be designed in such a way that the overall time is to be optimizing under the pre specified constraints. The load on each processor has been also taken care off in order to balance the load of processor.

(5) Technique:

Let the given system consists of a set \( P = \{p_1, p_2, p_3, p_4, \ldots p_n\} \) of \( n \) processors, interconnected by communication links and a set \( T = \{t_1, t_2, t_3, t_4, \ldots t_m\} \) of \( m \) tasks. The processing execution time of individual tasks corresponding to each processor are given in the matrix \( ETM(\cdot) \) of order \( m \times n \). The communication time is taken in the square symmetric matrix \( CTM(\cdot) \) of order \( n \). Initially, we obtain the task combinations in order to make the set of task(s) equals to number of processor. These combinations shall be \( n \times \binom{n}{l} \) (= \( nl \), say) and to be store in \( TCOMB(\cdot) \). Then we obtained an index, which is based on the time of the tasks to the processors for the execution of tasks to various processors, and also time of communication amongst the task to each of the combination, the maximum value of the index shall give the optimal result. The assignment of tasks to processors may be done in different ways. To allocate the task to one of the processors, the minimum value of each row and column of \( ETM(\cdot) \) is obtained. Let \( \min \{p_i\} \) represent the minimum row cost value corresponding to the tasks \( t_i \) and \( \min \{c_j\} \) represent the minimum column time value for processor \( p_j \). These values are then replaced to \( 0 \) in \( ETM(\cdot) \). For allocation purpose a modified version of row and column assignment method of Kumar et al [14] is employed which allocates a task to processor where it has minimum execution time. The overall assignment time \( [Etime]\) is expressed as the sum of execution time and communication time of all the tasks as follows:

\[
Etime = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} ET_{iq}x_{ij} \right) + \sum_{j=1}^{n} \left( \sum_{i=1}^{n} CT_{pj}y_{ij} \right) \right]
\]

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

\( y_{ij} = \begin{cases} 1, & \text{if the task assigned to processor communicates with the task assigned to processor} \\ 0, & \text{otherwise} \end{cases} \)

Index = \( (Etime)^{-1} \)

(6) Implementation:

**Example-1**

Consider a system consisting of a set \( T = \{t_1, t_2, t_3, t_4\} \) of 4 tasks and a set \( P = \{p_1, p_2, p_3\} \) of 3 processors,

**Step-1: Input:** 4, 3

\[
E_{TM} = \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix}
\]

**Step-2: Process the Etime:**

\[
E_{time} = \sum_{i=1}^{n} \left( \sum_{j=1}^{n} ET_{ij} \right) = 8 + 9 + 12 = 39
\]

**Step-3: Compute the Index:**

\[
\text{Index} = \frac{1}{39}
\]

**Step-4: Optimization:**

\[
[Etime]^{-1} = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix}
\]

\[
E_{time}^{-1} = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix} = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix}
\]

\[
\text{Index} = \frac{1}{39} \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix}
\]

**Step-5: Final Result:**

\[
\text{Index} = \frac{1}{39} \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix} = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix}
\]

**Step-6: Result:**

The final result is obtained as:

\[
\text{Index} = \frac{1}{39} \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & 0 \\
1 & 0 & 1
\end{pmatrix} \begin{pmatrix}
8 & -12 & 7 \\
9 & 8 & 11 \\
12 & 9 & 6 \\
10 & 11 & 12
\end{pmatrix}
\]

**Conclusion:**

The index is calculated using the above steps and the final result is obtained which is optimal for the given system.
\[
\begin{align*}
\tau_1 & \tau_2 \tau_3 \tau_4 \\
\tau_1 & 0 \ 3 \ 6 \ 9 \\
\tau_2 & \tau_3 \tau_4 & 3 \ 0 \ 4 \ 5 \\
\tau_3 & 6 \ 4 \ 0 \ 7 \\
\tau_4 & 9 \ 5 \ 7 \ 0 \\
\end{align*}
\]

CTM \(_(*) = \begin{bmatrix} 3 & 4 & 2 & 4 & 2 & 3 & 1 & 4 & 1 & 3 & 1 & 2 \\ 4 & 3 & 4 & 2 & 3 & 2 & 4 & 1 & 3 & 1 & 2 & 1 \end{bmatrix}\]

Step-2:

\[T_{COMB}(\cdot) = \begin{bmatrix} 3 & 4 & 2 & 4 & 2 & 3 & 1 & 4 & 1 & 3 & 1 & 2 \\ 4 & 3 & 4 & 2 & 3 & 2 & 4 & 1 & 3 & 1 & 2 & 1 \end{bmatrix}\]

Step-3:

\[T_{COMB}(1) = \begin{bmatrix} 3 \\ 4 \end{bmatrix}\]

Step-3.1-3.2:

\[ETM(\cdot) = p_1 \ p_2 \ p_3 \ t_3 \ t_4 \begin{bmatrix} 22 & 20 & 18 \\ 0 \ 3 \ 6 \\ 3 \ 0 \ 4 \\ 6 \ 4 \ 0 \end{bmatrix}\]

\[NCTM(\cdot) = \begin{bmatrix} 1 & 0 \ 3 & 6 \\ 2 & 3 \ 0 \ 4 \\ 3 & 6 \ 4 \ 0 \\ 1 & 8 \ 12 \ 7 \\ 2 & 9 \ 8 \ 11 \\ 3 & 22 \ 20 \ 18 \end{bmatrix}\]

Step-4 & 5:

On applying modified Hungarian method devised by Kumar et al [14] to assign the task, min \(\{r_i\}\) from \(NCTM(\cdot)\) for every \(i, r_{i1} = 7, r_{i2} = 8, r_{i3} = 18\). Making \(r_{i3} = r_{i2} = r_{i3} = 0\).

Again min \(\{c_j\}\) from \(NETM(\cdot)\) for every \(j, c_{j1} = 8, c_{j2} = 0, c_{j3} = 0\). Making \(c_{j1} = 0\), so that, we get,

\[NETM(\cdot) = \begin{bmatrix} 1 & 0 \ 3 & 6 \\ 2 & 9 \ 8 \ 11 \\ 3 & 22 \ 20 \ 18 \end{bmatrix}\]

Step-6, 7 & 8:

After implementing assignment process, the first set of the allocation is thus obtained.

\[
\begin{array}{cccccccc}
\text{Tasks} & \text{Processors} & \text{ET} & \text{CT} \\
\tau_1 & p_1 & 08 & 03 \\
\tau_2 & p_2 & 08 & 06 \\
\tau_3, \tau_4 & p_3 & 18 & 04 \\
\end{array}
\]

Step-9:

\[\text{ETT}(1) = 34\]

Step-10:

\[\text{CTT}(1) = 13\]

Step-11:

\[\text{Etme}(1) = 47\]

Step-12:

\[\text{Index}(1) = 0.02127650\]

Step-13:

On repeating the above process, the assignments and their corresponding related values of ET, CT, Etme, and Indexes are thus obtained, which are shown in the following table-1:

\[
\begin{array}{cccccc}
\text{S. No.} & \text{ET} & \text{CT} & \text{Etme} & \text{INDEX} & \text{ASSIG-I} & \text{ASSIG-II} & \text{ASSIG-III} \\
1 & 34 & 13 & 47 & 0.02127650 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
2 & 34 & 17 & 51 & 0.01960780 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
3 & 33 & 13 & 46 & 0.02173910 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
4 & 33 & 22 & 55 & 0.01818181 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
5 & 34 & 17 & 51 & 0.01960780 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
6 & 34 & 22 & 56 & 0.01785710 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
7 & 32 & 13 & 45 & 0.02222222 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
8 & 32 & 16 & 48 & 0.02083333 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
9 & 32 & 17 & 48 & 0.02083333 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
10 & 32 & 16 & 47 & 0.02127650 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
11 & 34 & 16 & 50 & 0.02000000 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
12 & 33 & 22 & 56 & 0.01785710 & \tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \\
\end{array}
\]

Step-14:

Stop.

Example-II

The results of a system which consist a set \(T = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}\) of 5 tasks and a set \(P = \{p_1, p_2, p_3\}\) of 3 processors where,

\[
\begin{align*}
P & = p_1 \ p_2 \ p_3 \\
\tau_1 & 12 \ 7 \\
\tau_2 & 8 \ 3 \ 6 \ 9 \\
\tau_3 & 4 \ 5 \\
\tau_4 & 6 \ 4 \ 0 \ 7 \\
\tau_5 & 9 \ 5 \ 7 \ 0 \\
\end{align*}
\]

\[ETM(\cdot) = \begin{bmatrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 \\ 8 & 12 & 7 & 0 & 3 \ 6 \ 9 \ 8 \\
12 & 9 & 8 & 11 & 4 \ 5 \ 6 \\
10 & 11 & 12 & 9 & 5 \ 7 \ 0 \ 5 \\
7 & 6 & 2 & 8 & 9 \ 5 \ 0 \\
3 & 2 & 18 & 20 & 0 \\
\end{bmatrix}\]

The following tables shows the results as obtain after implementing the present algorithm.
(7) Conclusions:

The present paper discusses an assignment problem through optimization techniques. In this paper, we have chosen the problem in which the number of tasks is more than the number of processors of the distributed system. The present method deals the case when the index is based on the time of the tasks to the processors for the execution of tasks to various processors and also communication time amongst the tasks. The method is presented in computational algorithmic form and implemented on the several sets of input data to test the performance and effectiveness of the algorithm. The optimal result of the Example-1 giving in the paper are shown in the following table:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>BT</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1 * t_2</td>
<td>P_1</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>t_3</td>
<td>P_2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>t_1 * t_4</td>
<td>P_3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

The graphical representation of the results mentioned in Table-1 of Step-12 of the implementation part of this paper are shown in the following graphs:

Graph-1: Index Graph

Graph-2: Execution Time Graph

The following table shows the results of Example-2 given in its implementation part of 8.2 in this paper, as obtain after implementing the present algorithm:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>BT</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1 * t_2</td>
<td>P_1</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>t_3</td>
<td>P_2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>t_1 * t_4</td>
<td>P_3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

References: