A TASK CLUSTER BASED LOAD BALANCING ALGORITHM FOR TASK ALLOCATION IN DISTRIBUTED SYSTEMS

Poornima Bhardwaj¹, Vinod Kumar²
¹ Research Scholar, Department of Computer Science, Gurukul Kangri Vishwavidyalaya Haridwar, India
² Professor and Head, Department of Computer Science, Gurukul Kangri Vishwavidyalaya Haridwar, India

ABSTRACT

With the recent enhancements in massive parallel processing technologies, the problem of scheduling tasks in multiprocessor systems is becoming progressively important. The problem of scheduling task graph in parallel and distributed computing system is a well defined NP-complete problem. The problem of task assignment, in heterogeneous computing systems has been premeditated for many years with many distinctions. In a Distributed Computing Systems (DCSs), a program is split into small tasks and distributed among several computing elements to minimize the overall system cost. In real – life situation when number of tasks is much larger than number of processors, a task clustering approach is very useful for allocating tasks on the basis of some predetermined criteria.

This paper deals with the problem of task allocation in a DCS such that the load on each processing node is almost balanced. The present work aims at the development of an effective algorithm for allocating 'm' tasks to 'n' processors in a given distributed system using task clustering. The model includes inter – task communication cost (ITCC) along with the execution cost (EC). A new concept of Load Balancing Factor (LBF) is introduced to judge the degree of load balancing. Value one, of LBF, signifies a perfect load balancing.

Keywords

Distributed Systems, Task Allocation, Static Load Balancing.

1. RELATED WORK

In a DCS, the processing loads arrive from many users at random time intervals. A proper scheduling policy attempts to assign these loads to available computing nodes so as to complete the processing of all loads in the shortest possible time. System performance can be improved by transferring work from nodes that are heavily loaded to nodes that are lightly loaded. One of the objectives of using load balancing is to minimize the overall execution cost of the application. Load – balancing algorithms can be generally classified as centralized or decentralized, dynamic or static, periodic or non-periodic, and those with thresholds or without thresholds. In this work, we are using static load balancing (SLB). In SLB, before commencement of execution, the decision of assignment of tasks to processors is taken. Expected execution time and resource requirements are known a priori. Tasks are executed on the processors to which it is assigned at the beginning of execution as there is no task migration in SLB. So, static scheduling is processor non-preemptive. A number of models [2, 5, 7, 15, 19, 21, 29, 31, 32] have been reported in the literature to achieve this kind of load balancing. A class of DAG scheduling algorithms is based on task clustering [9, 13, 30]. The basic idea of clustering based algorithms is to group heavily communicating tasks into the same cluster. Tasks, which are grouped into the same cluster, are assigned to the same processor in an effort to avoid communication cost.

The main problem arises when the number of clusters is more than the number of available processors. This leads to the scheduling of more than one cluster onto the same processor and unavoidably increases the overall schedule length. Approaches, that are used to minimize the execution as well as communication costs, are discussed in [6,12, 20, 22, 23].

The work done by Liou et al. indicates that if task clustering is performed prior to the scheduling of the clusters to the processors in a balanced way, it gives us the benefit of load balancing [16]. Similar work is done by Palis et al. [17] to reduce the overall execution time. They developed a simple greedy algorithm for task clustering. In the task clustering approach, proposed by Vidyarthi et al. [28] task modules are clustered on the basis of their communication overhead. Modules, having high communication overhead, are clustered to decrease the communication delays. The approach of clustering heavily communicating tasks is also used in [14, 24]. In [14], the authors choose modules which are considered to be maximally linked from the point of view of the processor on which the assignment is to be done. A maximally linked module, having maximum amount of communication with its neighboring modules, is selected for clustering. Sharma et al. in [24] formed clusters on the basis of their communication cost to reduce the communication delays. In the work reported by Raii et al. [18], the total number of clusters formed out of the total tasks(m) is equal to the number of processor (n) and
each cluster may not contain more than \([m/n]\) tasks. The tasks are arranged in an array according to their size. Initially first two tasks, from this array, are selected and cluster is formed.

2. INTRODUCTION

DCS is being comprehensively helpful to a variety of large size computational problems. These computational environments consist of multiple heterogeneous computing modules, which interrelate with each other to solve the problem. One of the major issues for a DCS is task allocation [4, 5, 11, 26]. In a DCS, with multiple tasks distributed among a set of processors, the problem of finding an optimal solution for task allocation to processors is found to be NP-hard [8]. The purpose of task allocation in DCSs is to increase the system throughput, which can be done by maximizing and balancing the utilization of computing resources and minimizing communication between processors. The overall objective of the study has been to allocate tasks constituting a distributed application to available processing elements to optimize one or more measure(s) of effectiveness i.e. load balancing on the processors, minimizing the inter-processor communication overhead (IPC), maximizing of system reliability, maximization of memory utilization and minimization of total cost of execution etc. [25]. Load balancing is the process of sharing computational resources by distributing the system workload transparently. System performance can be improved by transferring workload from nodes that are heavily loaded to nodes that are lightly loaded.

Much work has been done in the area of allocation of tasks in such a way that maximum load balancing is achieved. In DCS, an allocation policy may be either static or dynamic, depending upon the time at which the allocation decisions are made. In a static task allocation, the information regarding the tasks and processor attributes is assumed to be known in advance, i.e. before the execution of the tasks.

In this paper, we have developed a task allocation algorithm using task clustering that will find a near optimal solution to the problem. The proposed algorithm tries to minimize the system MTIME by forming tasks cluster and the maximum number of tasks in a cluster is not more than \(\text{ceil}(m/n)\). The result thus obtained, are compared with those obtained by the algorithm developed by Ucar et al. [27]. The results are given both in tabular as well as graphical forms which show that the proposed algorithm performs better.

3. THE TASK ASSIGNMENT PROBLEM

The problem, being addressed in this paper, is concerned with an optimal allocation of the task clusters of an application on to the processors in a DCS. An optimal allocation is one that minimizes the system cost function subject to the system constraints. Let the given DCS consist of a set of ‘n’ processors \(P=\{p_1, p_2,\ldots, p_n\}\), interconnected by communication links and a set of ‘m’ tasks \(T=\{t_1, t_2,\ldots, t_m\}\). The processing efficiency of individual processor is given in the form of Execution Cost Matrix, \(ECM(_i)\) of order \(m \times n\) and Inter Task Communication Cost Matrix, \(ITCCM(_i)\) of order \(m \times m\).

3.1 Notations

\(m\) : Number of tasks

\(n\) : Number of processors

\(T\) : A set of tasks of a parallel program to be executed

\(P\) : The set of processors

\(ECM(_i)\) : Execution Cost Matrix of order \(m \times n\)

\(ITCCM(_i)\) : Inter Task Communication Cost Matrix of order \(m \times m\)

\(COSTEX(_i)\) : This array stores execution cost of the clusters on processor \(p_i\) (\(i = 1,2,\ldots,n\)).

\(COSTCC(_i)\) : This array stores ITCC of the clusters with the other clusters

\(CCOST(_i)\) : Contains the sum of EC of the task group allocated on a processor and its ITCCs with the other task cluster.

\(MSCOST\) : Maximum \(\{CCOST(_i), i=1,2,3,\ldots,n\}\)

\(TOC\) : Total Optimal Cost

\(ELBCE\) : Effective Load Balancing Cost of Execution

\(ELBCT\) : Effective Load Balancing Cost of Total system

\(ALLOC(_i)\) : This matrix, of order \(n \times n\), stores the position at which a task is allocated to a processor.

\(MERG(_i)\) : This matrix, of order \(m \times n\), contains the list of cluster wise merged tasks.

\(e_{ij}\) : EC of task \(t_i\) on processor \(p_j\).

\(cc_{ik}\) : ITCC between task \(t_i\) and task \(t_k\)

\(\text{ceil}(m/n)\) : Least integer \(\geq m/n\)
3.2 Basic Definitions

Execution cost (EC): The execution cost \( e_{ij} \) (where \( 1 \leq i \leq m \) and \( 1 \leq j \leq n \)) of a task \( t_i \), running on a processor \( p_j \) is the amount of the total cost needed for the execution of \( t_i \) on that processor.

Inter Task Communication Cost (ITCC): The inter task communication cost depends on the amount of data units exchanged between the tasks. If the interacting tasks \( t_i \) and \( t_k \) are assigned to different processors, the communication cost \( cc_{ik} \) is incurred due to the data units exchanged between them during the execution.

Load – Balancing Factor (LBF): The LBF is defined as, maximum number of tasks allocated to a single processor divided by total number of processors. The increased value of LBF results in the tendency of allocation of large number of tasks on a single processor which deteriorates the overall load balancing. On the other hand the lower value of LBF exhibits the efficient load balancing. The best value for LBF is one, when all tasks are equally assigned on the given processors.

Effective Load – Balancing Cost of Execution (ELBCE): It is defined as the total execution cost when the best load balancing is achieved as per given criteria.

Effective Load – Balancing Cost of Total system (ELBCT): It is defined as the total system cost when the best load balancing is achieved as per given criteria.

Maximum System Cost (MSCOST): It is the maximum of the sums of execution costs of various task – clusters with their ITCCs with other clusters.

3.3 Assumptions

The present technique is based on the assumptions mention ahead:

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

ii. If a task is not executable on a certain processor, due to absence of some resources, the Execution Cost of that task on that processor is taken to be very large (infinite).

iii. The completion of a program from computational point of view means that all related tasks have got executed.

iv. Reassignment of task(s) is not considered i.e. the allocation policy is static.

4. PROPOSED METHOD

Here, we assume that \( m > n \), which indicates that more than one task may be allocated to a processor at a time. So, a task is allocated to a processor in such a way that MSTIME is minimized. For this we can form as many task clusters as there are number of processors. The number of tasks in any cluster is not more than ceil(m/n).

Here we use, ‘profit factor \( (pf) \)’ based clustering criteria discussed by Ucar et al. in [27] to allocate such clusters to the processors thereby achieving load balancing.

The profit factor for a pair of tasks \( t_k \) and \( t_l \) is calculated by using the following formula:

\[
pf_{kl} = cc_{kl} - d_{kl}
\]

where,

\[
d_{kl} = \min\{e_{kj} + e_{lj}\} - (\min\{e_{kj}\} + \min\{e_{lj}\})
\]

The maximum value for ‘pf’ is then chosen and the corresponding task set is selected for clustering. These task sets, known as clusters, denoted as \( C_i \) (\( i=1,2,\ldots,n \)), are then allocated to the processors using Hungarian Algorithm [10].

The ECs, of the task \( t_k \) and \( t_l \), of the task – pair having maximum value of \( pf \) is added in ECM(,) and it is stored as FECM(,). Same tasks are merged in MERG(,). The ITCCs of the tasks \( t_k \) and \( t_l \) is added in ITCCM(,) by adding the \( l^{th} \) row on the \( k^{th} \) row and removing the \( l^{th} \) row, similarly, \( l^{th} \) column is added to the \( k^{th} \) column and \( l^{th} \) column is removed.

When the tasks in a row of MERG() are more than two, the group of these tasks is called as cluster.

Let a cluster \( C_i \) contain, \( (t_{i1}, t_{i2}, \ldots, t_{ik}) \) tasks,
To take the result to the reality, we have introduced a new factor termed as, Load Balancing Factor (LBF) to find the effect of allocating almost equal number of tasks on each processor $p_j (j = 1, 2, \ldots, n)$. The total number of tasks, contained in the largest $C_i$ ($i = 1, 2, \ldots, n$) is $L$. So, LBF can be calculated as,

$$LBF = \frac{L}{n}$$

The Effective Load Balancing Cost of Execution (ELBCE), is calculated by

$$ELBCE = \sum_{i=1}^{n} COSTEX(i)$$

The Effective Load Balancing Cost of Total system (ELBCT), is calculated by

$$ELBCT = TOC \times LBF$$

### 4.1 Proposed Algorithm

**Step 1:** Input $m$ and $n$.

**Step 2:** Input ECM(,) and ITCCM(,) and initialize MERG(,) by all task indices in the first column and -1 at all other places.

**Step 3:** Calculate

**Step 3.1:** Total number of tasks in a cluster at initial level

$$q_0 = \lceil \frac{m}{n} \rceil$$

**Step 3.2:** Remaining tasks

$$r_m = m \mod n$$

**Step 4:** Extract non-zero elements of ITCCM(,) and store them row wise in GRAPH(,) where column 1 contains index of the starting task of the task – pair, column 2 contains the index of second task of the pair and the last column contains the communication cost corresponding to the task pair.

$$count = \text{total number of rows in GRAPH(,)}.$$ 

**Step 5:** Form clusters as follows:

**Step 5.1:** Corresponding to the each task – pair, represented by the first and second column of GRAPH(,), calculate the minimum EC of the task – pair on each processor, (say $m_{\text{intp}}$) as well as minimum of the ECs of the first and second task of the pair on all the processors and take the sum, (say $s_{\text{min}}$). Subtract $s_{\text{min}}$ from $m_{\text{intp}}$ and take the result in $df$. Take the ITCC of the task pair (say $c_{\text{f}}$), subtract $df$ from $c_{\text{f}}$ and save the result in PFM(,). The first column of PFM(,) contains the starting task index of the task – pair, second column contains the second task index of task – pair and the third contains the value of $pf$ corresponding to the task – pair.

For the same values of $m_{\text{intp}}$ for a task – pair, on different processors the value, encountered at the first instance, is chosen.

**Step 5.2:**

**Step 5.2.1:** Count the number of tasks in each row of MERG(,) and result is stored in MERGCL().
Step 5.2.2: Add the MERGCL() corresponding to all the pairs in GRAPH(,) and result are taken in SUMPFM().

Step 5.3: Find the maximum value of pf from PFM(,) with the corresponding task – pair (say C) and store it in MAXPF().

Step 5.4: Merge the tasks that are contained in C, in MERG(,). The set of tasks represented by a row of MERG(,) is called a cluster.

Step 5.5: Add the ECs of the tasks corresponding to each row of MERG(,) and the results are stored in FECM(,).

Step 5.6: Add the ITCCs of the tasks – clusters corresponding to each row of MERG(,) and results are taken in FITCCM(,).

Step 6: If rm is not equal to zero, then

Step 6.1: Find the remaining tasks by comparing the list consisting of all tasks, (say REM()) with those tasks of the MERG(,), which have got included in any of the cluster. Remove all the tasks that are common in both. The leftover tasks in REM() are the remaining tasks.

Step 6.2: Calculate pf of the remaining task(s) with all clusters and store the resultant values in PFM(,).

Step 6.2.1: For each remaining task, (say t_q), select the maximum value of pf and allocate t_q to the corresponding cluster.

Step 6.2.2: If there is no pf value for any of the cluster, then the remaining task(s) is merged to the first such cluster where the total number of tasks does not exceed qo.

Step 6.3: The MERG(,) is updated by merging the remaining task(s) to the selected clusters.

For load balancing, no two remaining tasks are merged to the same cluster.

Step 6.4: Add the EC(s) of each remaining task(s) to the EC of the corresponding cluster to which it is merged and store the results in FECM(,).

Step 6.5: Add the ITCC of the remaining task(s) with the ITCC of the clusters to which it is merged and store the results in FITCCM(,).

end if

Step 7: Store FECM(,) to COST(,) and apply Hungarian Algorithm [10] on FECM(,).

The positions of the allocations of clusters on the processors are saved in ALLOC(,).

Step 8:

Step 8.1: For the total number of tasks (say, L) in a cluster. Calculate Load Balancing Factor (LBF), L/n, for each processor.

Step 8.2: Find the maximum value of LBF (say maxlbf).

Step 9: According to the positions stored in ALLOC(,), the ECs of each cluster is found from COST(,) and result is taken in COSTEX(,).

Step 10: Corresponding to each of the cluster the ITCC is found from FITCCM(,) and results are stored in COSTCC(,).

Step 11:

$$\text{ELBCE} = \left[ \sum_{i=1}^{n} \text{COSTEX}(i) \right] \ast \text{maxlbf}$$
Step 12: 
\[ TOC = \sum_{i=1}^{n} (COSTEX(i) + COSTCC(i)) \]

\[ ELBCT = TOC \times \text{maxlbf} \]

Step 13:

**Step 13.1:** The sum of ECs and corresponding ITCCs are stored in CCOST()

**Step 13.2:** MSCOST = Max (CCOST(1), TCOST(2),…TCOST(n))

Step 14: Stop.

### 4.2 Implementation of Algorithm

In this section, two examples are taken to explain the solution methodology of the proposed algorithm. The first example elaborates the method when the number of tasks are uniformly divided on all the processors and there is no remaining tasks(s). On the other hand, the second example deals with more general situation where, the case of remaining task(s) is taken care of.

**Example -1**

In this example, we have considered a typical program made up of 6-executable tasks \{t_1, t_2, t_3, t_4, t_5, t_6\} to be executed on the DCS having three processors \{p_1, p_2, p_3\}. We have taken the execution cost of each task on different processors and ITCC between the tasks in the form of matrices ECM(,) and ITCCM(,) respectively. The processors connections are shown in Figure 1. The inter task communication graph and task execution graph are shown in Figure 2 and Figure 3 respectively.

![Figure 1: Processors Connections of Example 1](image1)

![Figure 2: Inter Task Communication Graph of Example 1](image2)
Figure 3: Task Execution Graph of Example 1

Step 1: Input $m = 6$, $n = 3$.

Step 2:

$$
\text{ECM}(,) = \\
\begin{pmatrix}
39 & 41 & 35 \\
41 & 40 & 38 \\
57 & 53 & 51 \\
40 & 39 & 52 \\
58 & 38 & 46 \\
48 & 38 & 48 \\
\end{pmatrix}
$$
Step 3: On performing calculation we get,

Step 3.1: $q_0 = 2$.

Step 3.2: $r_m = 0$

Step 4:

$$\text{GRAPH}(,) = \begin{bmatrix}
1 & 2 & 4 \\
2 & 4 & 4 \\
2 & 6 & 3 \\
3 & 4 & 3
\end{bmatrix}$$

$count = 4$

Step 5:

Step 5.1: For the first task pair in $\text{GRAPH}(,)$ i.e. $t_1 t_2$,

$$\text{mintp} = (35 + 38) = 73$$
$$\text{smin} = (35 + 38) = 73$$
$$df = 73 - 73 = 0$$
$$cf = 4$$
$$pf = 4 - 0 = 4$$

Similarly for all task pairs the value of $pf$, is calculated and stored as,

$$\text{PFM}(,) = \begin{bmatrix}
0 & t_1 t_2 & 4 \\
t_2 t_3 & 2 \\
t_3 t_4 & 1 \\
t_4 t_5 & 1
\end{bmatrix}$$

Step 5.2:

Step 5.2.1: Initially, there is only one task in each row of $\text{MERG}(,)$, therefore,

$$\text{MERGCL}(,) = (1, 1, 1, 1)$$

Step 5.2.2:

$$\text{SUMPFM}(,) = (2, 2, 2, 2)$$

Step 5.3: $\text{MAXPF}(,) = (4)$ and the corresponding cluster is $t_1 t_2$.

Step 5.4:

18 | Page
MERG(,) =  
\[
\begin{bmatrix}
1 & 2 & -1 & -1 & -1 \\
3 & -1 & -1 & -1 & -1 \\
4 & -1 & -1 & -1 & -1 \\
5 & -1 & -1 & -1 & -1 \\
6 & -1 & -1 & -1 & -1 \\
\end{bmatrix}
\]

**Step 5.5:** By adding the ECs of task \( t_1 \) and \( t_2 \).

\[
\text{FECM(,)} =
\begin{bmatrix}
t_{1t_2} & p_1 & p_2 & p_3 \\
t_3 & 57 & 53 & 51 \\
t_4 & 40 & 39 & 52 \\
t_5 & 58 & 38 & 46 \\
t_6 & 48 & 38 & 48 \\
\end{bmatrix}
\]

**Step 5.6:** Similarly, ITCCs of tasks \( t_2 \) and \( t_5 \) are added.

\[
\text{FITTCM(,)} =
\begin{bmatrix}
t_{1t_2} & 0 & 0 & 4 & 0 & 3 \\
t_3 & 0 & 0 & 3 & 0 & 0 \\
t_4 & 4 & 3 & 0 & 0 & 0 \\
t_5 & 0 & 0 & 0 & 0 & 0 \\
t_6 & 3 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

After Step 5.1 to 5.6 we get,

MERG(,) =  
\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{bmatrix}
\]

\[
\text{FECM(,)} =
\begin{bmatrix}
t_{1t_2} & 80 & 81 & 73 \\
t_{3t_4} & 97 & 92 & 103 \\
t_{5t_6} & 106 & 76 & 94 \\
\end{bmatrix}
\]

\[
\text{FITTCM(,)} =
\begin{bmatrix}
t_{1t_2} & 0 & 4 & 3 \\
t_{3t_4} & 4 & 0 & 0 \\
t_{5t_6} & 3 & 0 & 0 \\
\end{bmatrix}
\]

**Step 6:** Since \( rm = 0 \).

**Step 7:** Store \( \text{FECM(,)} \) in \( \text{COST(,)} \) and apply Hungarian Algorithm [10] on \( \text{FECM(,)} \) which gives

\[
\text{ALLOC(,)} =
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
\]

The Allocation obtained after implementing the row and column assignment process is shown in Table 1.
Table 1: Optimal Allocation Table of Example 1

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>COSTEX()</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁, t₂</td>
<td>p₃</td>
<td>73</td>
</tr>
<tr>
<td>t₃, t₄</td>
<td>p₁</td>
<td>97</td>
</tr>
<tr>
<td>t₅, t₆</td>
<td>p₂</td>
<td>76</td>
</tr>
</tbody>
</table>

Step 8:

**Step 8.1:** The LBF on each processor is calculated as,

\[
\text{LBF}(1) = \frac{2}{3} = 0.666667 \\
\text{LBF}(2) = \frac{2}{3} = 0.666667 \\
\text{LBF}(3) = \frac{2}{3} = 0.666667
\]

**Step 8.2:**

\[
\text{maxlbf} = 0.666667
\]

Step 9: The COSTEX() = (73, 97, 76)

Step 10: The COSTCC() = (4, 3, 0)

Step 11: The value of ELBCE is calculated as,

\[
\text{ELBCE} = 246 \times 0.666667 \\
= 164.000000
\]

Step 12: The TOC and ELBCT are calculated as

\[
\text{TOC} = 246 + 7 \\
= 253 \\
\text{ELBCT} = 253 \times 0.666667 \\
= 168.666672
\]

Step 13:

**Step 13.1:** CCOST() = (80, 101, 79)

**Step 13.2:** MSCOST = 101

Step 14: Stop.

The optimal allocation graph is shown in Figure 4. The performance comparison of proposed algorithm and the algorithm discussed in [27] on same set of values is shown in Table 2.

![Figure 4: Optimal Assignment Graph of Example 1](image-url)
Table 2: Comparison of Results of Example 1

<table>
<thead>
<tr>
<th>Model Result</th>
<th>Previous Algorithm [27]</th>
<th>Present Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATIONS</td>
<td>(t_1, t_2, t_3, t_4, t_6 \rightarrow p_2)</td>
<td>(t_1, t_2 \rightarrow p_3)</td>
</tr>
<tr>
<td>COE</td>
<td>249</td>
<td>246</td>
</tr>
<tr>
<td>COC</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>TOC</td>
<td>253</td>
<td>253</td>
</tr>
<tr>
<td>Maxlbf</td>
<td>1.000000</td>
<td>0.666667</td>
</tr>
<tr>
<td>ELBCE</td>
<td>249.000000</td>
<td>164.000000</td>
</tr>
<tr>
<td>ELBCT</td>
<td>253.000000</td>
<td>168.666672</td>
</tr>
<tr>
<td>MSCOST</td>
<td>123</td>
<td>101</td>
</tr>
</tbody>
</table>

4.2.2 Example – 2
In this example, we have considered a typical program made up of 7-executable tasks \(\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}\) to be executed on the DCS having three processors \(\{p_1, p_2, p_3\}\). ECM(,) and ITCCM(,) have same meaning as illustrated earlier. The processors connections are shown in Figure 5. The inter task communication and task execution graphs are shown in Figures 6 and 7.

![Figure 5: Processors Connections of Example 5.1](image-url)
Step 1: Input $m = 7$, $n = 3$

Step 2:

$$ECM(,) = \begin{pmatrix}
  t_1 & 54 & 51 & 57 \\
  t_2 & 37 & 32 & 50 \\
  t_3 & 32 & 38 & 37 \\
  t_4 & 41 & 38 & 52 \\
  t_5 & 58 & 56 & 44 \\
  t_6 & 57 & 41 & 43 \\
  t_7 & 58 & 36 & 34 \\
\end{pmatrix}$$

$$TTCCM(,) = \begin{pmatrix}
  t_1 & 1 & 0 & 1 & 3 & 0 & 0 & 1 \\
  t_2 & 0 & 0 & 1 & 3 & 0 & 0 & 1 \\
  t_3 & 0 & 1 & 0 & 0 & 0 & 0 & 3 \\
  t_4 & 0 & 3 & 0 & 0 & 0 & 0 & 2 \\
  t_5 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\
  t_6 & 1 & 0 & 3 & 0 & 0 & 0 & 0 \\
  t_7 & 0 & 2 & 0 & 0 & 4 & 0 & 0 \\
\end{pmatrix}$$
Step 3: On performing calculation we get,

**Step 3.1:** \( q_o = 2 \)

**Step 3.2:** \( r_m = 1 \)

**Step 4:**

\[
\text{GRAPH}(,) = \begin{pmatrix}
1 & 6 & 1 \\
2 & 3 & 1 \\
2 & 4 & 3 \\
2 & 7 & 2 \\
3 & 6 & 3 \\
5 & 7 & 4
\end{pmatrix}
\]

\( \text{count} = 6 \)

**Step 5:** For the first task pair in \( \text{GRAPH}(,) \) i.e. \( t_1 t_6 \),

\[
\begin{align*}
\text{mintp} &= (51+41) = 92 \\
\text{smin} &= (51+41) = 92 \\
\text{df} &= 92 - 92 = 0 \\
\text{cf} &= 1 \\
\text{pf} &= 1 - 0 = 1
\end{align*}
\]

Similarly for all task – pair the value of \( \text{PFM}(,) \) is calculated and stored as,

\[
\text{PFM}(,) = \begin{pmatrix}
t_1 t_6 & 1 \\
t_2 t_3 & -4 \\
t_2 t_4 & 3 \\
t_3 t_7 & 0 \\
t_4 t_6 & -3 \\
t_5 t_7 & 4
\end{pmatrix}
\]

**Step 5.2:**

**Step 5.2.1:** Initially, there is only one task in each row of \( \text{MERG}(,) \), therefore,

\( \text{MERGCL}(,) = (1,1,1,1,1,1) \)

**Step 5.2.2:**

\( \text{SUMPFM}(,) = (2,2,2,2,2,2) \)

**Step 5.3:**

\( \text{MAXPF}(,) = (4) \) and corresponding cluster is \( t_5 t_7 \).

**Step 5.4:**

\[
\text{MERG}(,) = \begin{pmatrix}
1 & -1 & -1 & -1 & -1 & -1 \\
2 & -1 & -1 & -1 & -1 & -1 \\
3 & -1 & -1 & -1 & -1 & -1 \\
4 & -1 & -1 & -1 & -1 & -1 \\
5 & 7 & -1 & -1 & -1 & -1 \\
6 & -1 & -1 & -1 & -1 & -1
\end{pmatrix}
\]
Step 5.5: By adding the ECs of task $t_5$ and $t_7$.

\[
\text{FECM}(,) = \begin{bmatrix}
\begin{array}{cccccc}
  t_1 & p_1 & 54 & 51 & 57 & \\
  t_2 & 37 & 32 & 50 & \\
  t_3 & 32 & 38 & 37 & \\
  t_4 & 41 & 38 & 52 & \\
  t_5 t_7 & 116 & 92 & 78 & \\
  t_6 & 57 & 41 & 43 & \\
\end{array}
\end{bmatrix}
\]

Step 5.6: Similarly, ITCCs of tasks $t_5$ and $t_7$ are added.

\[
\text{FITTCM}(,) = \begin{bmatrix}
\begin{array}{ccccccc}
  t_1 & t_2 & t_3 & t_4 & t_5 & t_7 & t_6 \\
  0 & 0 & 0 & 0 & 0 & 1 & \\
  0 & 0 & 1 & 3 & 2 & 0 & \\
  0 & 1 & 0 & 0 & 0 & 3 & \\
  0 & 3 & 0 & 0 & 0 & 0 & \\
  0 & 2 & 0 & 0 & 0 & 0 & \\
  1 & 0 & 3 & 0 & 0 & 0 & \\
\end{array}
\end{bmatrix}
\]

After Step 5.1 to 5.6. We get,

\[
\text{MERG}(,) = \begin{bmatrix}
\begin{array}{c}
  1 \\
  2 \\
  5 \\
  3 \\
\end{array}
\end{bmatrix}
\]

\[
\text{FECM}(,) = \begin{bmatrix}
\begin{array}{cccccc}
  t_1 t_6 & p_1 & 111 & 92 & 100 & \\
  t_2 t_4 & 78 & 70 & 102 & \\
  t_3 & 32 & 38 & 37 & \\
  t_5 t_7 & 116 & 92 & 78 & \\
\end{array}
\end{bmatrix}
\]

\[
\text{FITTCM}(,) = \begin{bmatrix}
\begin{array}{cccccc}
  t_1 t_6 & t_2 t_4 & t_3 & t_5 t_7 \\
  0 & 0 & 3 & 0 & \\
  0 & 0 & 1 & 2 & \\
  3 & 1 & 0 & 0 & \\
  0 & 2 & 0 & 0 & \\
\end{array}
\end{bmatrix}
\]

Step 6: Since $r_m = 1$

Step 6.1: \( \text{REM}(,) = t_3 \).

Step 6.2:

Step 6.2.1:

\[
\text{PFM}(,) = \begin{bmatrix}
\begin{array}{cc}
  t_1 t_6 t_3 & -3 \\
  t_2 t_4 t_3 & -5 \\
\end{array}
\end{bmatrix}
\]

Since task $t_3$ has maximum value of $p_f$ with cluster $t_1 t_6$. So, $t_3$ is merged with this cluster.

Step 6.3:

\[
\text{MERG}(,) = \begin{bmatrix}
\begin{array}{ccc}
  1 & 6 & 3 \\
  2 & 4 & -1 \\
  5 & 7 & -1 \\
\end{array}
\end{bmatrix}
\]
Step 6.4:

\[ \text{FECM}(i) = \begin{bmatrix}
    t_1 & t_3 & t_6 \\
    t_2 & t_4 & t_7 \\
    143 & 78 & 70 \\
    116 & 92 & 78
\end{bmatrix} \]

Step 6.5:

\[ \text{FITCCM}(i) = \begin{bmatrix}
    t_1 & t_3 & t_6 & t_7 \\
    t_2 & t_4 & 0 & 1 \\
    t_5 & 0 & 2 & 0
\end{bmatrix} \]

Step 7: \( \text{COST}(i) = \text{FECM}(i) \). On applying Hungarian Algorithm [10] on \( \text{FECM}(i) \)

\[ \text{ALLOC}(i) = \begin{bmatrix}
    0 & 1 & 0 \\
    1 & 0 & 0 \\
    0 & 0 & 1
\end{bmatrix} \]

The Allocation obtained after implementing the row and column assignment process is shown in Table 3.

Table 3: Optimal Allocation Table of Example 2

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>COSTEX()</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1, t_3, t_6 )</td>
<td>( p_2 )</td>
<td>130</td>
</tr>
<tr>
<td>( t_2, t_4 )</td>
<td>( p_1 )</td>
<td>78</td>
</tr>
<tr>
<td>( t_5, t_7 )</td>
<td>( p_3 )</td>
<td>78</td>
</tr>
</tbody>
</table>

Step 8:

Step 8.1: The LBF on each processor is calculated as,

\[ \text{LBF}(1) = \frac{3}{3} = 1.000000 \]
\[ \text{LBF}(2) = \frac{2}{3} = 0.666667 \]
\[ \text{LBF}(3) = \frac{2}{3} = 0.666667 \]

Step 8.2:

\[ \text{maxlbf} = 1.000000 \]

Step 9: The \( \text{COSTEX}() = (130, 78, 78) \)

Step 10: The \( \text{COSTCC}() = (1, 0, 2) \)

Step 11:

\[ \text{ELBCE} = 286 \times 1.000000 \]
\[ = 286.000000 \]

Step 12: The TOC and ELBCT are calculated as
Step 13:

**Step 13.1:** \( \text{CCOST}() = (131, 81, 80) \)

**Step 13.2:** \( \text{MSCOST} = 131 \)

Step 14: Stop.

The optimal allocation graph is shown in Figure 8. The performance comparison of proposed algorithm and the algorithm discussed in [27], for above example is given in Table 4.

![Optimal Assignment Graph of Example 2](image)

Table 4: Comparison of Results of Example 2

<table>
<thead>
<tr>
<th>Model Result</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALLOCATIONS</strong></td>
<td>( t_1 t_3 t_6 \rightarrow p_2 ) ( t_2 t_4 \rightarrow p_1 ) ( t_5 t_7 \rightarrow p_3 )</td>
<td>( t_1 t_3 t_7 \rightarrow p_3 ) ( t_3 \rightarrow p_1 ) ( t_2 t_4 \rightarrow p_2 )</td>
</tr>
<tr>
<td><strong>COE</strong></td>
<td>286</td>
<td>280</td>
</tr>
<tr>
<td><strong>COC</strong></td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td><strong>TOC</strong></td>
<td>289</td>
<td>288</td>
</tr>
<tr>
<td><strong>maxlbf</strong></td>
<td>1.000000</td>
<td>1.333333</td>
</tr>
<tr>
<td><strong>ELBCE</strong></td>
<td>286.000000</td>
<td>373.333344</td>
</tr>
</tbody>
</table>
6. RESULT AND CONCLUSIONS

In this paper, we discussed the problem of task allocation by balancing load in DCS using task clustering. But, task allocation problem is known to be NP-hard in complexity, when the optimal solution to this problem is required. Therefore, we have proposed an efficient algorithm, which finds near optimal value of MSCOST. To represent the real world situations we introduce a load balancing factor (LBF) to reflect the effect of load balancing. It is noticeable that when LBF is used the total cost of the system is decreased. The tendency of the previous algorithm discussed in [27] is to attain the worst value by allocating most of the tasks on the same processor. While in our case, tendency is to attain the best value by almost equally dividing the total number of tasks on each processor.

One of the best techniques to minimize MSCOST is to minimize the communication among tasks. Thus, the proposed algorithm tries to form cluster of tasks and then allocate these clusters to the processors while achieving load balancing. The performance of the proposed algorithm is compared with the algorithm proposed by Ucar et al. in [27].

The comparisons of the results is presented in tabular as well as in graphical forms. In Table 5 comparison of LBF is done, when number of tasks is taken in increasing order and number of processors is fixed, the resultant values of proposed algorithm are less than the results produced in [27]. For same set of values taken as input as in Table 5, the comparison of ELBCT is done in Table 6. The comparison of generated values shows that the results generated by proposed algorithm are better than the result obtained in [27]. Table 7, shows that the values of LBF generated by the proposed algorithm, when the number of processors is taken in increasing order and number of tasks is fixed are less than the values obtained in [27]. The ELBCT of proposed algorithm and the algorithm in [27] is compared in Table 8, for same set of input values as in Table 7. The generated results shows that the proposed algorithm performs better than the algorithm discussed in [27]. The comparison of MSCOST is shown in Table 9, when the number of tasks are taken in increasing order and the number of processors is fixed. The values of MSCOST generated by proposed algorithm are less than the values generated in [27]. In Table 10, when number of processors are taken in increasing order and number of tasks is kept fixed, the MSCOST values generated by proposed algorithm are better than the values obtained in [27]. The graphical representation of the values in Tables 5 and 6 are shown in Figure 9 and 10 respectively. Figures 11 and 12, illustartes Table 7 and 8. The values of the MSCOST are graphically represented in Figure 13 and Figure 14 corresponding to Table 9 and Table 10.

### Table 5: Comparison of Values of LBF When Number of Tasks is Increasing

<table>
<thead>
<tr>
<th>Tasks (m)</th>
<th>Processors (n)</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>0.666667</td>
<td>1.333333</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1.666667</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>2.333333</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1.333333</td>
<td>2.666667</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1.333333</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 6: Comparison of Values of ELBCT When Number of Tasks is Increasing

<table>
<thead>
<tr>
<th>Tasks ( m )</th>
<th>Processors ( n )</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>85</td>
<td>116.666664</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>99</td>
<td>140</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>117</td>
<td>210</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>161.333344</td>
<td>234.666672</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>161.333344</td>
<td>237</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>218.666672</td>
<td>410</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>358.333313</td>
<td>520.666687</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>363.333313</td>
<td>544</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>383.333313</td>
<td>689</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>522</td>
<td>849.333313</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>752</td>
<td>880</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>814</td>
<td>1184</td>
</tr>
</tbody>
</table>

Table 7: Comparison of Values of LBF When Number of Processors is Increasing

<table>
<thead>
<tr>
<th>Tasks ( m )</th>
<th>Processors ( n )</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>3</td>
<td>2</td>
<td>5.333333</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>1.25</td>
<td>3.75</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>0.8</td>
<td>2.8</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>0.428571</td>
<td>1.714286</td>
</tr>
</tbody>
</table>

Table 8: Comparison of Values of ELBCT When Number Processors is Increasing
### Table 9: Comparison of Values of MSCOST When Number of Tasks is Increasing

<table>
<thead>
<tr>
<th>Tasks m</th>
<th>Processors n</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>3</td>
<td>990</td>
<td>1045.333374</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>721.25</td>
<td>847.5</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>408</td>
<td>574</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>155.5</td>
<td>510</td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>132</td>
<td>372</td>
</tr>
</tbody>
</table>

### Table 10: Comparison of Values of MSCOST When Number of Processors is Increasing

<table>
<thead>
<tr>
<th>Tasks m</th>
<th>Processors n</th>
<th>Proposed Algorithm</th>
<th>[27] Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>3</td>
<td>277</td>
<td>513</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>196</td>
<td>483</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>192</td>
<td>350</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
<td>192</td>
<td>300</td>
</tr>
</tbody>
</table>
Figure 9: LBF When Tasks are in Increasing Order and Number of Processors is 3

Figure 10: ELBCT When Tasks are in Increasing Order and Number of Processors is 3

Figure 11: LBF When Processors are in Increasing Order and Number of Tasks is 18
Figure 12: ELBCT When Processors are in Increasing Order and Number of Tasks is 18

Figure 13: MSCOST When Tasks are in Increasing Order and Number of Processors is 3

Figure 11: MSCOST When Processors are in Increasing Order and Number of Tasks is 16
REFERENCES


33 | Page

This work is licensed under a Creative Commons Attribution 4.0 International License.

DOI: 10.24297/ijcds.v4i2.1033