An Effective Load Balancing Task Allocation Algorithm using Task Clustering

Poornima Bhardwaj
Research Scholar, Department of Computer Science
Gurukul Kangri Vishwavidyalaya,Haridwar, India

Vinod Kumar, Ph.d
Professor & Head, Department of Computer Science, Gurukul Kangri Vishwavidyalaya, Haridwar, India

ABSTRACT
In Distributed Computing Systems (DCSs), a program is split into small tasks and distributed among several computing elements to minimize the overall system cost. Several challenges have been posed by this mode of processing which can be classified mainly into two broad categories. One class belongs to the hardware oriented issues of building such systems more and more effective while the other aims at designing efficient algorithms to make the best use of the technology in hand. The task allocation problem in a DCS belongs to the later class. Intrinsically, task allocation problem is NP- hard. To overcome this issue, it is necessary to introduce heuristics for generating near optimal solution to the given problem.

This paper deals with the problem of task allocation in DCSs in such a way that the load on each processing node is almost balanced. Further, the development of an effective algorithm for allocating 'm' tasks to 'n' processors of a given distributed system using task clustering by taking both Inter Task Communication Cost (ITCC) and the Execution Cost (EC) is taken into consideration.

General Terms
Distributed Systems, Task Allocation

Keywords
Distributed Computing Systems, Task Allocation, Static Load Balancing, Execution Cost, Communication Cost.

1. INTRODUCTION
The potential of distributed computing exists whenever there are several computers unified in some manner so that a program or a procedure running on one machine can transfer control to a procedure running on another. Distributed computers can be best exemplified as geographically dispersed logically and physically independent processors with decentralized system – wide control on resources and co-operative computation of programs. The task allocation problem is a fundamental aspect of distributed computing. It arises whenever the procedures or modules of a program are distributed over several interconnected computers and the program is active among processors as execution proceeds. The allocation problem deals with the question of allocating modules to the processors so as to minimize the cost of running a program [2].

The task allocation problem has multiple dimensions. Various aspects of the task allocation problem in a DCS have been covered by researchers. This problem has been tackled with various techniques through different allocation schemes considering different aspects such as inter- processor communication, reliability, load on the processors, memory utilization, etc. The objective of the present 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. equalizing load on the processors, minimization of inter – processor communication (IPC), maximization of system reliability and minimization of total cost of execution etc. [16].

In a 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.

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 execution cost of the application. Efficient load balancing method can increase performance. 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 [1]. In this work Static Load Balancing (SLB) is used. In SLB, the decision of assignment of tasks to processors is taken before the commencement of execution. Expected execution time and resource requirements are known a priori. Task is executed on the processors to which it is assigned at the beginning of execution as there is no task migration in SLB. The main purpose of the static scheduling is to minimize execution time and communication overhead [16, 17].

The heuristic-based scheduling techniques are the most common approaches for task scheduling. These are usually classified into three classes,

- Priority-based scheduling,
- Duplication-based scheduling and
- Cluster-based scheduling

In priority-based scheduling, priorities are calculated and assigned to the tasks which are then scheduled onto processors according to the priorities. In duplication – based scheduling, while tasks are allocated to a processor, its parent (and predecessor) tasks are duplicated to occupy the idle times of the processor to eliminate the communication delay that occurs when message is passed from the parent tasks to the allotted task.

In cluster-based scheduling, some tasks, that need to communicate among themselves, are grouped together to form a cluster. Each of these tasks – clusters is then scheduled onto an available processor. 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 total sum of execution and communication costs, are discussed in [3, 7, 13, 14, 15,].

In this paper, a task allocation algorithm based on clustering, that finds a near optimal solution to the problem, is proposed. The developed algorithm tries to minimize the total system cost by forming cluster of tasks in such a way that the cluster, having minimum execution cost, is allocated first. The communication cost of the tasks in the same cluster is assumed to be zero. Using this strategy it has been observed that the total system cost is less as compared to that which is obtained by the heuristic reported in [5].

2. RELATED WORK

The basic idea of clustering based algorithm is to group heavily communicating tasks into the same cluster. The tasks that are grouped into the same cluster are assigned to the same processor in an effort to avoid communication costs. There are basically two types of clusters; linear and nonlinear. Two tasks are called independent if there are no dependent paths between them. A cluster is called nonlinear if there are at least two independent tasks in the same cluster, otherwise, it is called linear.

A class of directed acyclic graph scheduling algorithms is based on clustering [4, 8, 20]. The work done by Liou *et al.* indicates that if task clustering is performed prior to scheduling, the clusters are assigned to the processors in a balanced way, giving the benefit of load balancing [10]. Similar work is done by Palis *et al.* [11] to reduce the overall execution cost by using a simple greedy algorithm for task clustering. In the task clustering approach, proposed by Vidyarthi *et al.* [19], 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 the heavily communicating tasks is used in [8, 17]. In [8], the authors choose those modules for clustering which are maximally linked from the point of view of the processor on which the assignment is to be done. Sharma *et al.* in [17] formed clusters on the basis of their communication cost to reduce the communication delays. In the work, done by Raii *et al.* [12], the total number of tasks in a cluster is not more than the total number of tasks divided by the total number of processors. Initially, all tasks are considered to be in a single cluster and stored in an array. The tasks, from this array, are selected for merging to form various clusters.

3. PROBLEM FORMULATION

The problem, being addressed in this paper, is concerned with an optimal allocation of the task - cluster of an application on the processors in a DCS. An optimal allocation is considered as one that minimizes the system cost function subject to the system constraints.

3.1 Problem Statement

Let the given DCS consist of a set of n processors P = {p_1, p_2,……..p_n}, interconnected by communication links and a set of m tasks T = { t_1,t_2,……..t_m}. The processing efficiency of individual processor is given in the form of Execution Cost Matrix, ECM(,), of order m x n and Inter Task Communication Cost Matrix, ITCCM(,), of order m x m. This technique can generate an optimal solution by minimizing overall computational cost and allowing large number of computing tasks.

3.2 Notations

T : a set of tasks of a parallel program to be executed

P : the set of processors

n : the number of processors

m : total number of tasks that constitute the program.

ECM(,) : Execution cost matrix of order m x n

ITCCM(,) : Inter Task Communication Cost Matrix of order m x m

COSTEX() : Array to store execution cost of clusters on processors.

COSTCC() : Array to store the communication cost of a cluster with all other clusters.

TOC **:** Total Optimal Cost

ACL(,) : contains all the possible clusters that are formed.

SUM(,) : contains corresponding ECs of all the clusters that are in ACL(,).

TPIN(,) : contains all the clusters that are finally selected from ACL(,).

FIN(,) : contains corresponding ECs of the cluster stored in TPIN(,).

ALLOC(,) : contains the position of the processor on which a cluster is assigned.

e_{ij} : EC of task t_i on processor p_j. When any task is unable to get executed on any processor its execution cost is assumed to be infinity.

cc_{ik} : ITCC between task t_i and task t_k

3.3 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 p_j.

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.

Total Optimal Cost (TOC): The total optimal cost is the overall system cost incurred when tasks are executed on various processors.

3.4 Assumptions

The present technique is based on the following assumptions:

(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 is not possible i.e. the allocation policy is static.

4. PROPOSED TASK ALLOCATION METHOD AND ALGORITHM

Since it is assumed that m>n, implying 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 total system cost is minimized. This inspires us to form clusters of tasks to make them equal to the number of processors. The task clusters C_i(i= 1,2,…….n) having (t_{r1}, t_{r2} …… t_{rk}) tasks are formed by making possible combination of tasks and adding the corresponding ECs. The cluster, having minimum EC, is selected and stored in final matrix, FIN(,). The process continues until number of clusters equals the number of processors. The remaining tasks are merged to the existing cluster having the maximum ITCC 'cc_{ij}'. At last, Hungarian Algorithm is applied to the FIN(,) and optimum system cost is obtained.

Once the final task clusters are selected, for each task t_{r1}, t_{r2} …… t_{rk} (r_i, i=1,2,…k) in cluster C_i(i= 1,2,…….n), the same tasks is merged in ITCCM(,). The communication between all the tasks in the same cluster is considered as zero. The procedure for merging is achieved by adding r_1^{th}, r_2^{th}, … .. r_k^{th} row on r_1^{th} row and remove all the other rows. Similarly r_1^{th}, r_2^{th}, ……. r_k^{th} column are merged on r_1^{th} column and all other columns are removed. The reduced ITCCM(,) is stored in NTT(,).

4.1 Proposed Algorithm

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

Step 2: Input ECM(,)and ITCCM

Step 3: Calculate the total number of tasks in a cluster

$$\boldsymbol{k} = \lfloor\_ (\boldsymbol{m}/\ \boldsymbol{n}) \_\rfloor$$

Step 3.1: Calculate total number of possible clusters, ***cl***

Step 3.2: Obtain total number of remaining task
rm = ***m*** mod ***n***

Step 4: Form clusters as follows:

Step 4.1: Task – clusters (say, C) are created by forming possible task combinations having k tasks in each cluster. The first cluster is formed by combining 1st, 2nd, 3rd …..upto k^{th} task. These clusters are stored in ACL(,).

Step 4.2: For all tasks that are in cluster C, calculate the sum of ECs, on each processor and store these values in SUM(,).
Step 5: Store SUM(,) to SUMNEW(,).

Step 5.1: Find the minimum EC in SUMNEW(,) and store it in ***min***.

Step 5.2: The cluster, corresponding to ***min***, is stored in TPIN(,) and the corresponding ECs from SUMNEW(,) are stored in FIN(,).

Step 5.3: Remove all the rows from ACL(,) containing either of the tasks that are in cluster C.

If the number of clusters in TPIN(,) is ***n*** and ***rm*** is zero then, go to Step 7.

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 TPIN(,), which contains the list of tasks that are included in any of the cluster. Remove all the tasks from REM() that are present in TPIN(,). The leftover tasks in REM() are the remaining tasks.

Step 6.2: For each remaining task, its communication with each cluster is calculated and stored in COM().

Step 6.3:
maxcom = Max $\{\mathrm{COM}(1), \mathrm{COM}(2), \ldots \mathrm{COM}(\boldsymbol{n})\}$
Step 6.4: Merge the remaining task with the cluster which corresponds to ***maxcom***.
Step 6.4.1: Update FIN(,) and TPIN(,).

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

Step 7: According to the clusters, stored in TPIN(,), the corresponding tasks are merged in ITCCM(,) and the result is stored NTT(,).

Step 8: Store FIN(,) to COST(,) and apply Hungarian Algorithm [5] to FIN(,).

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

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:

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

$$\mathrm{COC} = \sum_{i=1}^{n} \mathrm{COSTCC(i)}$$

Step 12: TOC = COE + COC

Step 13: Stop.

4.2 Implementation of Algorithm

Example – 1 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 a DCS having three processors {p_1, p_2, p_3}. The execution cost of each task on different processors and ITCC between the tasks is taken in the form of matrices ECM(,) and ITCCM(,) respectively. This example deals with the more general situation where the case of remaining task(s) is taken care of. The processors connections are shown in Figure 1. The task execution and inter task communication graphs are illustrated in Figure 2 and Figure 3.

Step 1: Input ***m*** = 7, ***n*** = 3

Step 2: Input

$$ITCCM(,) = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \\ \begin{pmatrix} 0 & 7 & 2 & 10 & 9 & 7 & 10 \\ 7 & 0 & 7 & 1 & 1 & 8 & 8 \\ 2 & 7 & 0 & 3 & 4 & 7 & 9 \\ 10 & 1 & 3 & 0 & 2 & 1 & 1 \\ 9 & 1 & 4 & 2 & 0 & 4 & 2 \\ 7 & 8 & 7 & 1 & 4 & 0 & 4 \\ 10 & 8 & 9 & 1 & 2 & 4 & 0 \end{pmatrix} \end{array}$$

$$ECM(,) = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 12 & 18 & 15 \\ 9 & 18 & 8 \\ 19 & 7 & 15 \\ 5 & 11 & 10 \\ 12 & 17 & 5 \\ 10 & 7 & 11 \\ 17 & 2 & 12 \end{pmatrix} \end{array}$$

Step 3: ***k*** = 2

Step 3.1: cl = 21

Step 3.2: rm = 1

Step 4:

$$ACL(,) = \begin{pmatrix} t_1\ t_2 \\ t_1\ t_3 \\ t_1\ t_4 \\ t_1\ t_5 \\ t_1\ t_6 \\ t_1\ t_7 \\ t_2\ t_3 \\ t_2\ t_4 \\ t_2\ t_5 \\ t_2\ t_6 \\ t_2\ t_7 \\ t_3\ t_4 \\ t_3\ t_5 \\ t_3\ t_6 \\ t_3\ t_7 \\ t_4\ t_5 \\ t_4\ t_6 \\ t_4\ t_7 \\ t_5\ t_6 \\ t_5\ t_7 \\ t_6\ t_7 \end{pmatrix} \quad SUM(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_2 \end{array} \\ \begin{pmatrix} 21 & 36 & 23 \\ 31 & 25 & 30 \\ 17 & 29 & 25 \\ 24 & 35 & 20 \\ 22 & 25 & 26 \\ 29 & 20 & 27 \\ 28 & 25 & 23 \\ 14 & 29 & 18 \\ 21 & 35 & 13 \\ 19 & 25 & 19 \\ 26 & 20 & 20 \\ 24 & 18 & 25 \\ 31 & 24 & 20 \\ 29 & 14 & 26 \\ 36 & 9 & 27 \\ 17 & 28 & 15 \\ 15 & 18 & 21 \\ 22 & 13 & 22 \\ 22 & 24 & 16 \\ 29 & 19 & 17 \\ 27 & 9 & 23 \end{pmatrix} \end{array}$$

Step 5: SUMNEW(,) = SUM(,)

Step 5.1: min = 9

Step 5.2:

$$TPIN(,) = \begin{pmatrix} t_3 & t_7 \end{pmatrix}$$

$$FIN(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 36 & 9 & 27 \end{pmatrix} \end{array}$$

Step 5.3: Remove the cluster that consists of either t_3 or t_7 from ACL(,) and corresponding ECs from SUMNEW(,). Now, the ACL(,) and SUMNEW(,) are:

$$ACL(,) = \begin{pmatrix} t_1\ t_2 \\ t_1\ t_4 \\ t_1\ t_5 \\ t_1\ t_6 \\ t_2\ t_4 \\ t_2\ t_5 \\ t_2\ t_6 \\ t_2\ t_7 \\ t_4\ t_5 \\ t_4\ t_6 \\ t_5\ t_6 \end{pmatrix} \quad SUMNEW(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_2 \end{array} \\ \begin{pmatrix} 21 & 36 & 23 \\ 17 & 29 & 25 \\ 24 & 35 & 20 \\ 22 & 25 & 26 \\ 14 & 29 & 18 \\ 21 & 35 & 13 \\ 19 & 25 & 19 \\ 26 & 20 & 20 \\ 17 & 28 & 15 \\ 15 & 18 & 21 \\ 22 & 24 & 16 \end{pmatrix} \end{array}$$

Again select the next minimum value, i.e. 13 from the remaining rows of ACL(,) and corresponding cluster $t_2\ t_5$ from SUMNEW(,). So, TPIN(,) and FIN(,) become:

$$TPIN(,) = \begin{pmatrix} t_3\ t_7 \\ t_2\ t_5 \end{pmatrix}$$

$$FIN(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 36 & 9 & 27 \\ 21 & 35 & 13 \end{pmatrix} \end{array}$$

Remove the clusters that either contains task t_2 or t_5 from ACL(,) and their corresponding ECs from SUMNEW(,). Now, the ACL(,), SUMNEW(,) are:

$$ACL(,) = \begin{pmatrix} t_1\ t_4 \\ t_4\ t_6 \end{pmatrix}$$

$$SUMNEW(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 17 & 29 & 25 \\ 15 & 18 & 21 \end{pmatrix} \end{array}$$

Again, select the next minimum value, i.e. 15 from the remaining rows of ACL(,) and corresponding cluster t_4t_6 from SUMNEW(,) resulting into TPIN(,) and FIN(,) as

$$TPIN(,) = \begin{pmatrix} t_3\ t_7 \\ t_2\ t_5 \\ t_4\ t_6 \end{pmatrix}$$

$$FIN(,) = \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{pmatrix} 36 & 9 & 27 \\ 21 & 35 & 13 \\ 15 & 18 & 21 \end{pmatrix} \end{array}$$

Here, the number of clusters in TPIN(,) is 3 and ***rm*** is not zero hence,

Step 6: ***rm*** = 1

Step 6.1: REM() = t_1

Step 6.2: The communication cost of t_1 is calculated with each cluster as:

COM(1) = 12
COM(2) = 16
COM(3) = 17

Step 6.3: maxcom = 17

Step 6.4: So, the third cluster, having tasks t_4 t_6, is selected for final merging resulting in

$$TPIN(,) = \begin{bmatrix} t_3\, t_7 \\ t_2\, t_5 \\ t_4\, t_6\, t_1 \end{bmatrix} \quad \text{and}$$

$$FIN(,) = \begin{array}{c} \\ 36 \\ 21 \\ 27 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{bmatrix} 9 & 27 \\ 35 & 13 \\ 36 & 36 \end{bmatrix} \end{array}$$

Step 7:

$$ITCCM(,) = \begin{array}{c} \\ t_3\, t_7 \\ t_2\, t_5 \\ t_1 t_4\, t_6 \end{array} \begin{array}{c} \begin{array}{ccc} t_3\, t_7 & t_2\, t_5 & t_4\, t_6\, t_1 \end{array} \\ \begin{bmatrix} 0 & 21 & 31 \\ 21 & 0 & 27 \\ 31 & 27 & 0 \end{bmatrix} \end{array}$$

NTT(,) = ITCCM()

Step 8: COST(,) = FIN(,). On applying Hungarian Algorithm [5] to the FIN (,). We get

$$ALLOC(,) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The allocations obtained after implementing the row and column assignment processes is shown in Table 1.

Step 9: COSTEX= (9, 13, 27)

Step 10: COSTCC = (21, 31, 27)

Step 11: COE = 9 + 13 + 27
= 49

COC = 21+ 31+ 27
= 79

Step 12: TOC = COE + COC
= 49 + 79 = 128.

Step 13: Stop.

The optimal assignment graph is shown in Figure 4. The performance comparison of proposed algorithm and the algorithm discussed in [5] on the same data set is shown in Table 2.

5. RESULTS AND CONCLUSIONS

In this paper, the problem of task allocation, considering load balancing using task clustering, is discussed. As the task allocation problem is known to be NP- hard, the proposed technique finds near optimal system cost. Static load balancing task allocation policy is used to achieve this objective. The proposed algorithm tries to form clusters of tasks and then allocate these clusters to the processors. The effectiveness of the proposed algorithm is compared with the static version of the algorithm proposed in [5]. For several sets of input data (m, n), the comparison is shown in tabular form as well as in graphical form. The Table 3 and Table 4 illustrated in the form of Figure 5 and Figure 6 respectively.

It can be observed from Figure 5 that the values of total optimal cost obtained by the present algorithm are less as compared to those obtained in [5], in the case, when the number of processors is kept fixed and number of tasks are taken in increasing order. The similar observation can also be made from Figure 6 in the case when the number of tasks is fixed and number of processors is taken in increasing order. Thus, it is concluded that the present algorithm results into better optimal cost in both the cases.

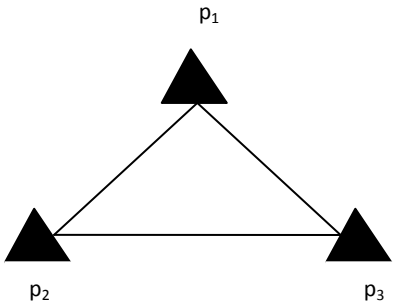

Figure 1: Processors Connections of Example 1

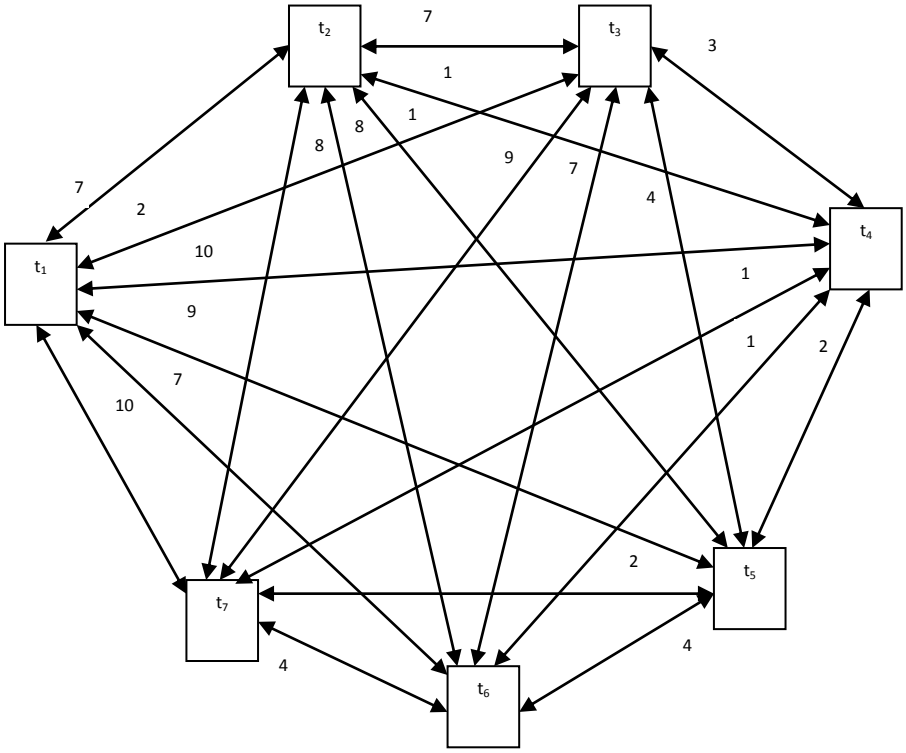

Figure 2: Inter Task Communication Graph of Example 1

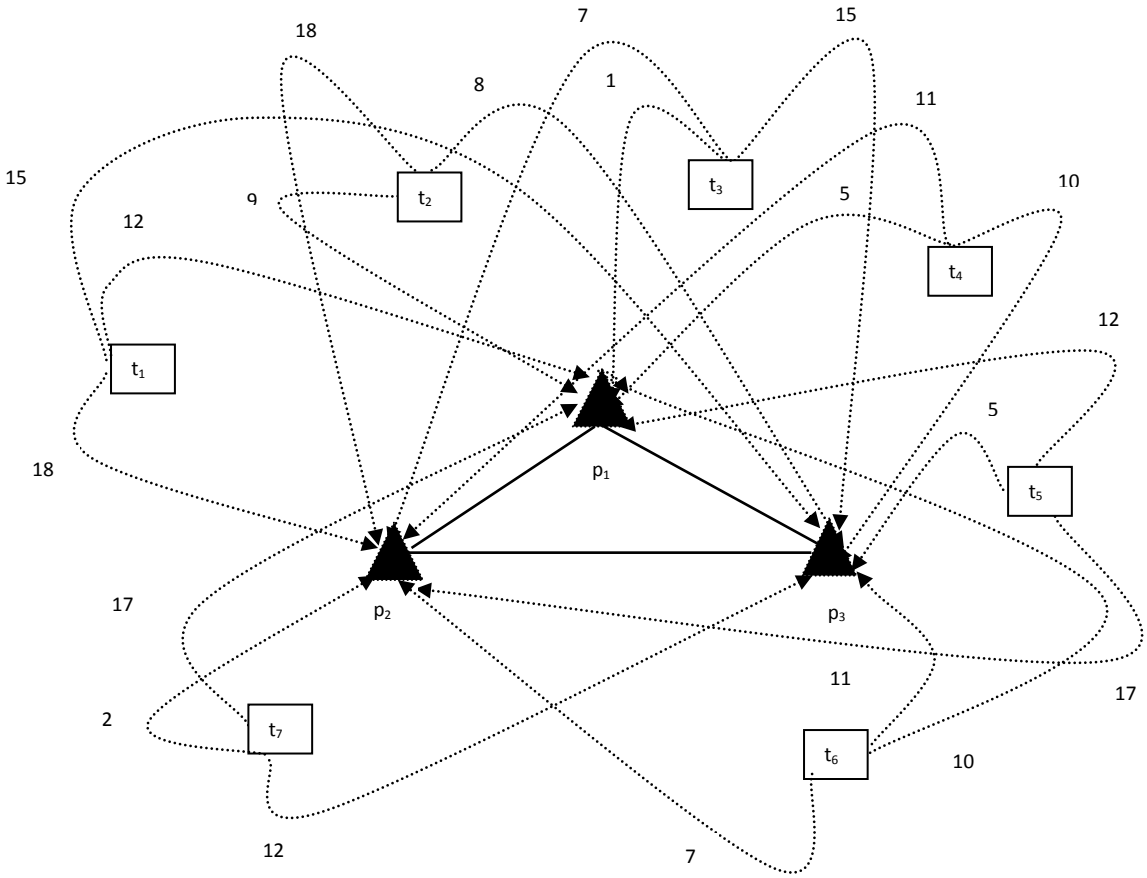

Figure 3: Task Execution Graph of Example 1

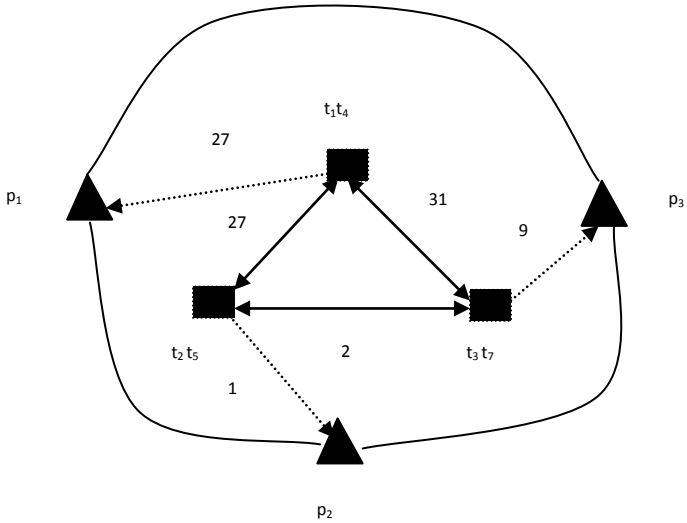

Figure 4: Optimal Assignment Graph of Example 1

Table 1: Optimal Allocation Table of Example 1

Tasks	Processors	EC
$t_1 t_4 t_6$	p_1	27
$t_2 t_5$	p_3	13
$t_3 t_7$	p_2	9

Table 2: Comparison of Results for Example 1

Model → / Result ↓	Proposed Algorithm	[5] Algorithm
ALLOCATIONS	$t_1 t_4 t_6 \rightarrow p_1$ $t_2 t_5 \rightarrow p_3$ $t_3 t_7 \rightarrow p_2$	$t_1 t_3 t_7 \rightarrow p_2$ $t_2 t_6 \rightarrow p_1$ $t_4 t_5 \rightarrow p_3$
COSTEX	49	61
COSTCC	79	76
TOC	128	137

Table 3: Comparison of TOC When Number of Tasks is in Increasing Order

Tasks m	Processors n	Proposed Algorithm	[5] Algorithm
6	4	104.8	110.7
7	4	138.2	142.5
8	4	178.2	184
9	4	217	226.6
10	4	238.1	257.8
11	4	316.7	333.8
12	4	366.3	378.1
13	4	394.1	423.4

Table 4: Comparison of TOC When Number of Processors is in Increasing Order

Tasks m	Processors n	Proposed Algorithm	[5] Algorithm
13	4	420.9	443.1
13	5	443.2	475.6
13	6	440	468.5
13	7	451.8	473.5
13	8	450.2	475.8
13	9	446.5	470.9
13	10	457	497.3
13	11	461.1	491.5
13	12	453.2	475

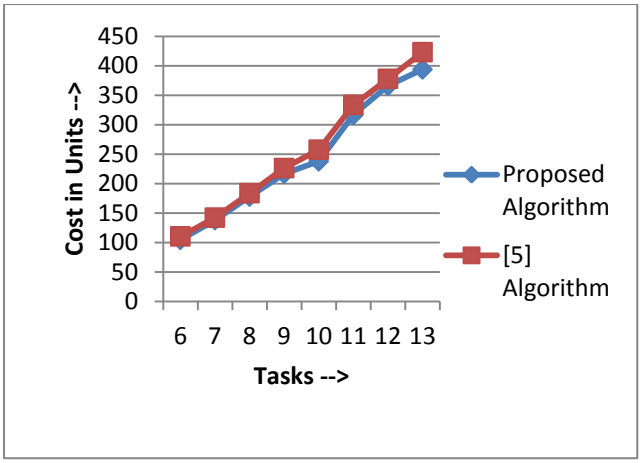

Figure 5: TOC When Tasks are in an Increasing Order and Number of Processors is 4

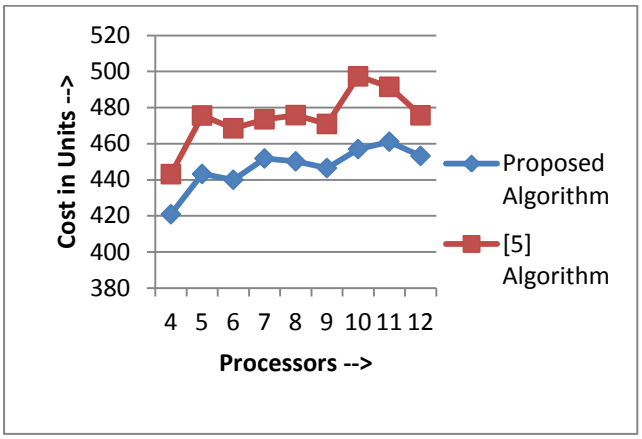

Figure 6: TOC When Processors are in an Increasing Order and Number of Tasks is 13

6. REFERENCES

[1] Alakeel Ali M. 2010. A guide to dynamic load balancing in distributed computer systems. International Journal of Computer Science and Network Security, 10 (6), 153 -160.

[2] Bokhari, S. H. 1987. Assignment problems in parallel and distributed computing.

[3] Chu, W. W., Holloway, L. J., Lan, M.T., and Efe, K. 1980. Task allocation in distributed data processing. IEEE Computer, 13(11) (Nov. 1980), 57-69.

[4] Gerasoulis, A. and Yang, T. A. 1992. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. Journal of Parallel and Distributed Computing, (1992), 276 – 291.

[5] Govil, Kapil 2011. A smart algorithm for dynamic task allocation for distributed processing environment. International Journal of Computer Applications, 28(2), (2011), 13-19.

[6] Gillett, Billy E. 1979. Introduction to operations research: a computer-oriented algorithmic approach.

[7] Kafil, M., and Ahmad, I. 1998. Optimal Task Assignment in Heterogeneous Computing Systems. IEEE Concurrency, (1998), 42- 51.

[8] Kim, S. J., and Browne, J. C. 1988. A general approach to mapping of parallel computations upon multiprocessor architectures. In Proceedings of the International Conference on Parallel Processing.

[9] Kumar, Avanish, Sharma, Abhilasha and Dhagat, Vanita Ben 2010. Maximal link mode algorithm for task allocation in distributed computing systems. Proceedings of the 4th National Conference: INDIACom-2010.

[10] Liou, J. -C. and Palis, M. A. 1996. An efficient clustering heuristic for scheduling dags on multiprocessor. In Proceedings of the 8th Symposium on Parallel and Distributed Processing.

[11] Palis M. A., Liou, J. -C. and Wei, D. S. 1996. Task clustering and scheduling for distributed memory parallel architectures. IEEE Transactions on Parallel and Distributed Systems (1996), 46 – 55.

[12] Raii, Anurag and Kapoor, Vikram 2012. Efficient clustering model for utilization of processor's capacity in distributed computing system. International Journal of Computer Applications, 44(23), 21- 25.

[13] Richard, R.Y., Lee, E.Y.S. and Tsuchiya, M. 1982. A task allocation modal for distributed computer system. IEEE Transaction on Computers 31, 41- 47.

[14] Sagar, G. and Sarje, A.K. 1991. Task allocation modal for distributed system. International Journal of System Science, 22(9), 1671 – 1678.

[15] Sarje, A. K. and Sagar, G. 1991. Heuristic modal for task allocation in distributed computer systems. IEEE Proceedings on Computers and Digital Techniques, 138(5), 313 – 318.

[16] Saxena Pankaj, Govil Kapil, Saxena Gaurav, Kumar Saurabh and Agrawal Neha. 2012. An algorithmic approach and comparative analysis of task assignment to processor for achieving time efficiency in process completion. International Journal of Applied Engineering and Technology, 2(1), 114-119.

[17] Sharma, Manisha, Kumar, Harendra, Garg, Deepak. 2012. An Optimal Task Allocation Model through Clustering with Inter-Processor Distances in Heterogeneous Distributed Computing Systems. International Journal of Soft Computing and Engineering, 2(1), 50- 55.

[18] Srinivasan, S., and Jha, N. K. 1999. Safety and reliability driven task allocation in distributed systems. IEEE Transactions on Parallel and Distributed Systems. 10(3), 238 – 251.

[19] Vidyarthi, Deo Prakash, Tripathi, Anil Kumar, Sarker, Biplab Kumer, Dhawan, Abhishek and Yang, Laurence Tianruo. 2004. Cluster - based multiple task allocation in distributed computing system. In Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04).

[20] Yang, T. and Gerasoulis, A. 1994. DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems. 5(9), 951- 967.