Improving the Performance of DRTS by Optimal Allocation of Multiple Tasks under Dynamic Load Sharing Scheme

Urmani Kaushal, Avanish Kumar

Abstract—Task allocation over distributed real time system, for parallel applications, is a vital segment, where policy for task allocation should be chosen in very appropriate manner. By efficient allocation of the tasks, the throughput and the overall processor utilization can be maximized. Task allocation is NP-hard or NP-complete problem. To improve the performance of the system, a new heuristic has been suggested and implemented in this paper. The number of modules which can be assigned on the processor is limited and the memory is also having certain limit. So these two constraints have been taken into consideration in the algorithm discussed in this paper. The dynamic load sharing policy has been used to improve the performance i.e. at the time of assignment, the required constraints must be check and fulfilled. For clustering task k-mean clustering is used and the proposed model is implemented in matlab.

.**Index Terms**— Task Allocation, Distributed Real Time System, parallel application, Cluster, throughput, processer utilization, NP-complete, dynamic load sharing.

1 INTRODUCTION

A distributed real-time system is a set of nodes connected by a real-time communication network that interact with each other in order to accomplish common task. The distributed real time system consists of a set of heterogeneous computers interconnected via a communication network. Each node has computation facility and its own memory while the communication network has a limited communication capacity. DRTS have come out as a great platform for high performance parallel applications [16]. The distributed computing system which makes the computation distribution over the nodes of the system has become very attractive because of the escalating necessitate of processing power for scientific calculations. A large range of studies has shown that the workstations are idle form 33% to 78% of the time [11].

The high performance environments presented by parallel and distributed computing system are much capable to provide high capacity of processing. For realization of such capacitative system, efficient task allocation algorithms and load distribution schemes must be employed in a very efficient manner. It is not the fact that the computing power of a distributed system increases proportionally with the number of processors involved. It should be taken care that the processors in the system should not be overloaded or idle [14].

The basic function of load distributing algorithm is to transfer load (tasks) from heavily loaded computers to idle or lightly loaded computers. The load distribution is meant for performance enhancement of a distributed system by the allocation of workload over the distributed system optimally. Load distributed algorithms can be broadly characterized as static and dynamic.

In static load distribution algorithms, jobs are assigned to hosts without considering the runtime events using a priori knowledge of the system probabilistically or deterministically. Dynamic load distribution algorithms use system state information like workload and any factor that may affect the choice of the most appropriate assignment to make decisions for load distribution [15].

Theory of load distribution classifies the load distributing algorithms as load balancing and load sharing algorithms. The load balancing algorithms equalize the loads at all the processors of the system. It transfers tasks at very higher rate than load sharing algorithms whereas in load sharing, the tasks are transferred by taking all appropriate decisions. Here once the tasks have been assigned to the particular node, it will not be reallocated.

Two load sharing policies may be applied to react to dynamic system change: dynamic placement, that allocates programs according to the current system state, and migration, that moves processes according to system and application. Load sharing is dedicated to long-lived applications [11]. Supports for parallel programming over workstations network often ignore programs' needs, real load conditions and users' activities [12]. Optimaum load sharing strategies are very hard to implement and are well-known as NP-complete. A number of works concentrated on static allocation [5, 7, 8], adapted for multiprocessor systems without interference between users or concurrent applications [11].

Here the objective is to maximize the overall performance of the system by allocating the task to the appropriate node in the system optimally. For enhancing the performance the system, it must have an efficient mechanism for task allocation of parallel application. A task allocation algorithm seeks an assignment that optimizes a certain cost function, for example maximum throughput or processor utilization or minimum turnaround time.

Multiple algorithms for task alloation in distributed compu-

- *Urmani Kaushal is currently working as Assitant Professor in Department of Physical & Computer Science in MITS University, Lakshmangarh Rajasthan, India, PH-09549821277. E-mail: urmani10kaushal@gmail.com*
- *Avanish Kumar is currently working as Professor & HOD in Department of Maths, Stats & Computer Applications in BundelkhandUniversity, Jhansi, India, PH-09935520565. E-mail: dravanishkumar@yahoo.com*

ting systems have been proposed in literature [3-8, 13]. The existing task allocation algorithms may cause either deadlock or starvation because of remote possibility of certain events at the time of allocation of the modules to the processing nodes that are already heavily loaded [9]. The task should be allocated to a processor in such an effective manner that the inter task communication cost may be passed up and the requirements for the task execution must be met by the allocated processor. The execution time of a particular module on a particular node will depend on the number of modules already executing on that particular node i.e. the maximum number of modules which can be assigned to any processing node must be considered. The memory capacity of the processing node determines if a module is to be accommodated onto the node or not.

2 Problem Formulation

In a Distributed Real-Time System, a task is allocated to a processor in such a way that inter task communication cost can be minimized and execution requirements of the task must suit the capabilities of the processor. The number of modules which can be assigned on the processor is limited and the memory is also having certain limit. So these two constraint should be taken into consideration in the algorithm discussed in this paper, which provide 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 Real-Time System with the goal to maximize the overall throughput of the system. The objective of this problem is to enhance the performance of the distributed real time system by making optimal utilization of its processors and suitable allocation of tasks.

2.1 Notations

T : the set of tasks of a parallel program to be executed.
P : the set of processors in DRTS.
n : the number of processors.
m: the number of tasks formed by parallel application .
k: the number of clusters.
t_i : i^{th} task of the given program.
P_l : l^{th} processor in *P*.
ec_{il} : incurred execution cost (*EC*), if ith task is executed on l^{th} processor.
cc_{ij} : incurred inter task communication cost between task t_i and t_j , if they are executed on separate processors.
X: an allocation matrix of order *m*n*, where the entry x_{il} *= 1; if* i^{th} task is allocated to l^{th} processor and 0; otherwise
CI : cluster information vector.
ECM (,) : execution cost matrix.
ITCCM (,) : inter task communication cost matrix.

2.2 Data Structures & Definations Used

We are proposing new data structure *MMSV* (Module & Memory Status Vector) and *MMSV* Table to manage the multiple tasks execution in a DRTS [17]. Description of the data structures are as follows:

2.2.1 MMSV:

It is a vector having status information of a single processor. Elements of *MMSV* are shown in Table 2.1. Element "*Processor ID*" is representing the processor unique name in the system.

TABLE 1: MMSV STRUCTURE

S.No.	Notations	Description
1.	P_l	Processing Node
2.	N_l	Maximum Number of Modules allowed on Processing Node P_l
3.	M_l	Maximum Memory Limit of Processing Node P_l
4.	x	Number of Module Assigned to the Processing Node P_l
5.	y	Number of Available Module on Processing Node P_l $(N_l - x)$
6.	$f(z)$	Memory Available

Total number of module assigned to the processing node Pl can be calculated as follows:

$$y = N_l - x \quad (1)$$

Total memory cost of processing node can be achieved by following cost function

$$f(z) = M_l - \sum_{x=1}^{x}(z_x) \quad (2)$$

z_x Memory required by x^{th} Module.
M_l Maximum Memory Limit of Processing Node P_l

2.2.2 MMSV Collection

MMSV's collection, Table 2 is the group of *MMSV* of all the processing nodes in a DRTS. It will form a table shown in Table 2.2. The memory capacity & module capacity (Available Module Capacity) of the processor decide whether a module is to be accommodated onto the processor or not.

2.2.3 Definitions

2.2.3.1 Execution Cost (EC)

The execution cost ecil of a task t_i, running on a processor P_l is the amount of the total cost needed for the execution of t_i on that processor during process execution. If a task is not executable on a particular processor, the corresponding execution cost is taken to be infinite (∞).

2.2.3.2 Communication Cost (CC)

The communication cost (cc_{ij}) incurred due to the inter task communication is the amount of total cost needed for exchanging data between t_i and t_j residing at separate processor during the execution process. If two tasks executed on the same processor then cc_{ij} = 0.

2.3 Assumptions

To allocate the tasks of a parallel program to processors in DRTS, we have been made the following assumptions:

2.3.1 The processors involved in the DRTS are heterogeneous and do not have any particular interconnection structure.

2.3.2 The parallel program is assumed to be the collection of *m*- tasks that are free in general, which are to be executed on a set of *n*- processors having different processor attributes.

TABLE 2: MMSV'S COLLECTION

Processor	No. of Modules (Maximum)	Memory Capacity (Maximum)	Module Assigned	Available Modules Capacity	Memory Available
P_1	N_1	M_1	Maximum *x* module from *m* tasks	y	f(z)
P_2	N_2	M_2	Maximum *x* module from *m* tasks	y	f(z)
P_3 . .	N_3 . .	M_3 . .	Maximum *x* module from *m* tasks .	y . .	f(z) . .
P_n	N_n	M_n	Maximum *x* module from *m* tasks	y	f(z)

2.3.3 Once the tasks are allocated to the processors they reside on those processors until the execution of the program is completed. Whenever a group of tasks is assigned to the processor, the inter task communication cost (*ITCC*) between them is zero.

2.3.4 Total number of clusters is equal to total number of processors.

2.3.5 Data points for *k-mean* clustering will be collection of vectors which represents the execution cost of the task tm on each processor.

2.3.6 Number of tasks to be allocated is more than the number of processors (*m*>>*n*) as in real life situation.

2.4 Proposed Mathematical Model for Task Allocation

In this section, we have developed a task allocation model to get an optimal system cost so that the system performance could be enhanced. We can achieve this objective by making task allocation properly. Therefore, an efficient task allocation of parallel application's tasks to processor is crucial. However, obtaining an optimal allocation of tasks of parallel application to any arbitrary number of processors interconnected with non-uniform links is a very complex problem.

Hereafter, in order to allocate the tasks of such program to processors in DRTS, we should know the information about the input such as tasks attributes [e.g execution cost, inter task communication cost etc]. While obtaining such information is beyond the scope of this paper therefore, a deterministic model that the required information is available before the execution of the program is assumed. In the present task allocation model, processor execution cost and task clustering has been considered for this system.

2.4.1 Execution Cost (EC)

The task allocation given as: *S: T→P, S (i)* = l. For the task allocation *S*, the execution cost ec_{il} represents the execution of task t_i on processor P_l and it is used to control the corresponding processor allocation. Therefore, under task allocation *S*, the execution of all the tasks assigned to lth processor can be computed as:

$$\mathrm{EC(X)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \sum_{\mathrm{i=1}}^{\mathrm{m}} ec_{il} x_{il} \qquad (3)$$

2.4.2 Task Clustering

Evaluation of cluster compactness as the total distance of each point (task vector of n dimension) of a cluster from the cluster mean which is given by [19], Z_{ki}

$$\sum_{x_i IC_k} \left\| X_i - \overline{X}_k \right\|^2 = \sum_{i=1}^{m} Z_{ki} \left\| X_i - \overline{X}_k \right\|^2 \qquad (4)$$

Where the cluster mean is defined as $\overline{X}_{\mathrm{k}} = \frac{1}{m_{\mathrm{k}}} \sum_{x_i IC_k} X_i$ and $m_{\mathrm{k}} = \sum_{i=1}^{m} \mathrm{Z}_{ki}$ is the total number of points allocated to cluster *k*. The parameter Z_{ki} is an indicator variable indicating the suitability of the ith data point X_i to be a part of the kth cluster.

The total goodness of the clustering will then be based on the sum of the cluster compactness measures for each of the k clusters. Using the indicator variables Z_{ki} , we can define the overall cluster goodness as:

$$\varepsilon_{\mathrm{k}} = \sum_{i=1}^{m} \sum_{k=1}^{k} Z_{ki} \left\| X_i - \overline{X}_k \right\|^2 \qquad (5)$$

Here $\overline{X}_{\mathrm{k}}$ should be found in such a manner that the value of εk can be minimized.

2.4.3 Processor Utilization

The purpose of the proposed model is to maximize the utilization of processors. This may increase by balancing the load on each processor. The processor utilization of a processor can be obtained by dividing the load on that processor, incurred due to the execution of the tasks assigned to it only, to the heaviest loaded processor. Let the calculated load on each of the processor P_l be w_l, and it is given by $w_l = \sum_{i\approx 1}^{m} ce_{il}\, x_{il}$. New utilization of each processor is calculated as:

$$\mathrm{PU(P_l)} = \frac{\mathrm{w_l}}{\max(\mathrm{w_l})} \qquad (6)$$

and the average processor utilization (APU) is calculated as:

$$\mathrm{APU} = [\mathrm{PU(1)} + \mathrm{PU(2)} + \ldots . + \mathrm{PU(n)}/\mathrm{n}. \qquad (7)$$

3 PROPOSED TASK ALLOCATION TECHNIQUE AND ALGORITHM

3.1 Technique

The problem addressed in this paper of tasks allocation of a parallel application onto the processors of a DRTS to enhance the performance of the system have a set $P = \{P_1, P_2, P_3, \ldots\ldots P_n\}$ of '*n*' processors and a set $T = \{t_1, t_2, t_3, \ldots\ldots t_m\}$ of 'm' tasks. The processing time of each task to each and every processor is known and it is mentioned in the Execution Cost Matrix *ECM*

(,) of order *m x n*. The communication cost of task is also known and is mentioned in Inter Task Communication Cost Matrix *ITCCM (,)* of order *m x m*.

To enhance the performance of the system the total system cost should be minimized. For the minimization of total system cost we will form the clusters of tasks. We have m tasks to be processed over n processors (*m*>*n*), so k clusters should be formed. For clustering we will use *k-mean* clustering algorithm. Here we have *m* vectors of task to be placed in *k* clusters. Find *k* initial points (number of points in cluster is equal to the number of clusters) for each cluster represented by task vector that are to be clustered. These points represent initial clusters called centroids. Assign each task vector to the cluster that has the closest centroid. When all task vectors have been assigned, recalculate the positions of *k* centroids. Repeat the work of assignment and recalculation of centroid's positions until the centroids no longer move. This produces a separation of the task vectors into clusters from which the metric to be minimized is calculated by using equation (4).

Modify the *ECM(,)* according the *k* clusters by adding the processing time of those tasks that occurs in the same cluster. Modify the *ITCCM(,)* by putting the communication zero amongst those tasks that are in same cluster. By applying algorithm proposed in [18], we get the optimal assignment as well as execution cost and communication cost. For optimal assignment of clusters of tasks will be computed as

$$EC(X) = \sum_{i=1}^{n}\sum_{j=1}^{m} ec_{il}\, x_{il} \qquad (8)$$

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

The objective function to calculate optimal system cost is as follows:

Total Cost = EC + CC (9)

3.2 Proposed Algorithm

The algorithm consists of following steps:

Step-0: Start
Step-1: Read the number of processors in *n*
Step-2: Read the number of tasks in *m*
Step-3: Repeate for each task
Read the number of modules in the task in *l*
Read the *ECM(,)* of the task of order *l x n*
Combine this *ECM(,)* with *CECM(,)*
Step-4: Read number of clusters in *k* (in this case it equal to number of processors)
Step-5: Read the Inter Task Communication Cost Matrix *ITCCM (,)* for each task of order *l x l*
Step-6: Read the *MMSV* Collection and Memory Requirement of each module
Step-7: Apply *k-mean* clustering algorithm on *CECM (,)*
Step-8: Cluster information is stored in Cluster Information Matrix *CIM*
Step-9: For all clusters repeate
if the cluster can't be assigned because of *MMSV* Collection or Memory Requirment over any processor Than
mark it as restricted assignment
Step-10: Check if any of cluster can't be assign because of *MMSV* Collection or memory requirment over all processors then
Goto Step 7
Step-11: Modify the *CECM (,)* by adding the processing time of tasks in each cluster
Step-12: Modify the *ITCCM (,)* by putting communication zero amongst those modules which are in the same cluster
Step-13: Apply [18] algorithm on *CECM (,)*
Step-14: Modify *MMSV* Collection according the assignment made in Step 13
Step-15: Calculate Execution Cost, Inter Task Communication Cost
Step-16: Optimal Cost = Execution Cost + Inter Task Communication Cost
Step-17: End

4 Implementaion

An illuminating example has been considered as in [17] to show the performance improvement of the distributed system, as well as to test the proposed algorithm using this data set. It is implemented in Matlab. It is assumed that the *ITCC* Matrices, the Execution Cost Matrices and *MMSV's* Collection Table are given for every module of each task in units of time. Given a set of three tasks with their corresponding modules $T_1(m_{11}, m_{21}, m_{31}, m_{41})$, $T_2(m_{21}, m_{22}, m_{32})$, $T_3(m_{13}, m_{22}, m_{33})$ and a set of four processors $\{P_1, P_2, P_3, P_4\}$.

TABLE 3: EXECUTION COST OF MODULES OF THE TASK T1

	P_1	P_2	P_3	P_4
m_{11}	10	20	5	25
m_{21}	35	10	15	15
m_{31}	10	15	25	10
m_{41}	20	35	20	5

TABLE 4: EXECUTION COST OF MODULES OF THE TASK T2

	P_1	P_2	P_3	P_4
m_{12}	20	5	35	10
m_{22}	10	10	10	10
m_{32}	15	10	20	15

TABLE 5: EXECUTION COST OF MODULES OF THE TASK T3

	P_1	P_2	P_3	P_4
m_{13}	15	25	15	10
m_{23}	30	40	25	20
m_{33}	20	5	10	15

TABLE 6: IMCC OF MODULES OF THE TASK T1

	m_{11}	m_{21}	m_{31}	m_{41}
m_{11}	0	10	50	20
m_{21}		0	10	50
m_{31}			0	50
m_{41}				0

TABLE 7: IMCC OF MODULES OF THE TASK T2

	m_{12}	m_{22}	m_{32}
m_{12}	0	5	10
m_{22}		0	60
m_{32}			0

TABLE 8: IMCC OF MODULES OF THE TASK T3

	m_{13}	m_{23}	m_{33}
m_{13}	0	5	40
m_{23}		0	10
m_{33}			0

TABLE 9: MEMORY REQUIREMENT OF MODULES IN UNITS

m_{11}	m_{21}	m_{31}	m_{41}	m_{12}	m_{22}	m_{32}	m_{13}	m_{23}	m_{33}
5	3	2	4	3	2	1	4	2	3

TABLE 10: MMSV'S COLLECTION

Processor	No. of Modules (Maximum)	Memory Capacity (Maximum)	Module Assigned	Available Modules Capacity	Memory Available
P_1	4	10		4	10
P_2	3	8		3	8
P_3	4	9		4	9
P_4	5	12		5	12

TABLE 12: CLUSTER INFORMATION

Cluster	Clustered Modules	Restricted Assignment	Final Assignment
C - 1	m_{21}, m_{33}	-	P_3
C - 2	m_{31}, m_{12}, m_{32}	-	P_2
C - 3	m_{11}, m_{22}, m_{13}	P_1 , P_2 , P_3	P_4
C - 4	m_{41}, m_{23}	-	P_1

TABLE 11: STATUS OF MMSV'S COLLECTION AFTER ALLOCATION

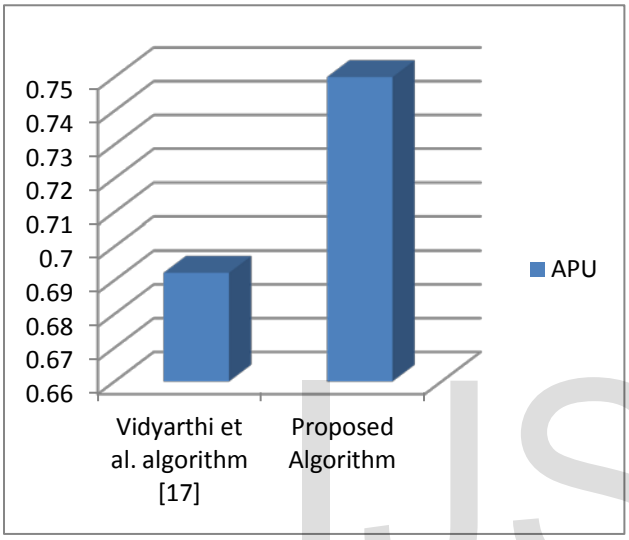
Processor	No. of Modules (Maximum)	Memory Capacity (Maximum)	Module Assigned	Available Modules Capacity	Memory Available
P_1	4	10	m_{41}, m_{23}	2	4
P_2	3	8	m_{31}, m_{12}, m_{32}	0	2
P_3	4	9	m_{21}, m_{33}	2	3
P_4	5	12	m_{11}, m_{22}, m_{13}	2	1

TABLE 13: OPTIMAL SYSTEM COST

Processors	Tasks	Processor Load	Optimal System Cost			PU	APU
			EC	ITCC	EC + ITCC		
P1	m_{41}, m_{23}	50	150	250	400	1	0.75
P2	m_{31}, m_{12}, m_{32}	30				0.6	
P3	m_{21}, m_{33}	25				0.5	
P4	m_{11}, m_{22}, m_{13}	45				0.9	

FIG. 1 OPTIMAL EXECUTION COST GRAPH

FIG. 2 AVERAGE PROCESSOR UTILIZATION GRAPH



5 CONCLUSION

In this paper, we have considered the dynamic task allocation problem considering load sharing scheme, a critical phase in distributed system with the goal of minimizing system cost, and maximizing the PU. The proposed model is based on an effective clustering to reduce *ITCC*. We present a straightforward and efficient algorithm to obtain optimal values of the objectives that we have considered in this paper. To measure the performance of proposed model & algorithm, the same example, present in Vidyarthi et al. [17] has been solved. In this case, it shows that the system cost is minimized by 20 % and the APU is maximized by 8.35 %. So by using this model the optimal solution can be achieved at all the times.

ACKNOWLEDGMENT

We gratefully acknowledge support from Dean and faculty members of Department of Physical & Computer Science, FASC, MITS, Lakshmangarh, Sikar and Department of Math., Stats. & Computer Applications, Bundelkhand University, Jhansi for the same.

REFERENCES

[1] Pereng-yi RICHARD MA, Edward Y.S.LEE, Masahiro TSUCHIYA, "A Task Allocation Model for Distributed Computing Systems", IEEE Trans. on Computers, Vol.C-31, No. 1, pp. 41-47, January 1982.

[2] Chien-Chung Shen, Wen-Hsiang Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems using a Minimax Criterion", IEEE Trans. on Computers, Vol. C-34, No.3, pp. 197-203, March 1985.

[3] Wesley W Chu, Lance M.T.Lan, "Task Allocation and Precedence Relations for Distributed Real Time Systems", IEEE Trans. on Computers, Vol. C-36, No.6, pp. 667-679 ,June 1987.

[4] Pradeep Kumar Yadav, M. P. Singh, Harendra Kumar "Scheduling Algorithm: Tasks Scheduling Algorithm for Multiple Processors with Dynamic Reassignment", Journal of Computer Systems, Networks, and Communications, Vol. 2008, 2008.

[5] Kapil Govil, Avanish Kumar "A Modified and Efficient Algorithm for Static Task Assignment in Distributed Processing Environment", International Journal of Computer Applications, Vol. 23, No. 8, June 2011.

[6] Kapil Govil, "A Smart Algorithm for Dynamic Task Allocation for Distributed Processing Environment", International Journal of Computer Applications, Vol. 28, No. 2, August 2011.

[7] P. K. Yadav, M. P. Singh, Kuldeep Sharma, "An Optimal Task Allocation Model for Sustem Cost Analysis in Heterogeneous Distributed Computing Systems: A Heuristic Approach", International Journal of Computer Applications, Vol. 28, No. 4, August 2011.

[8] Anurag Raii, Vikram Kapoor, "Efficient Clustering Model for Utilization of Processor's Capacity in Distributed Computing System", International Journal of Computer Applications, Vol. 44, No. 23, April 2012.

[9] D.P.Vidyarthi, A.K.Tripathi, "Maximizing Reliability of Distributed Computing Systems with Task Allocation using Simple Genetic Algorithm", J. of Systems Architecture, Vol. 47, pp. 549-554, 2001.

[10] Barak, A., Laden, O., & Yarom, Y. (1995). The NOW MOSIX and Its Preemptive Process. IEEE Technical Committee on Operating Systems , 7 (2), 5-11.

[11] Folliot, B., & Sens, P. (2008). "Load Sharing and Fault Tolerance Manager." In R. Buyya, High Performance Cluster Computing Architectures and Systems (p. 841). Pearson.

[12] Geist, G. A., & Sunderam, V. S. Network Based Concurrent Computing on the PVM System. Concurrency: Practice and Experience, vol 4, no. 4, pp 293-311, 1992.

[13] Saxena, Pankaj & Govil, Kapil, "An Optimized Algorithm for Enhancement of Performance of Distributed Computing System", International Journal of Computer Applications, vol 64, no. 2, Feb 2013.

[14] Karim y. Kabalan, waleed w. Smari and jacques y. Hakimian, "Adaptive Load Sharing In Hetergeneous Systems: Policies, Modifications, And Simulation", I. J. Of simulation Vol. 3 No. 1-2, pp-89-100.

[15] Bubendorfer, Kris. Dynamic Load Distribution. 1996. 25 June 2013 <http://homepages.mcs.vuw.ac.nz/~kris/thesis/node11.html>.

[16] Kopetz, H. (1997). REAL-TIME SYSTEMS: Design Principles for Distributed Embedded Applications. Kluwer Academic Puplishers.

[17] Vidyarthi, Deo Prakash, et al. "Allocation of Multiple Tasks in DCS." Vidyarthi, Deo Prakash, et al. Scheduling in Distributed Computing Systems Analysis, Design & Models (A Research Monograph). Springer Science+Business Media, LLC, 2009.

[18] A. Kumar, M.P. Sing, P. K. Yadav, "A Fast Algorithm for Allocating Tasks in Distributed Processing System", Proceedings of the 30th Annual Convention of CSI, Hyderabad, (1995), 347-358.