ISSN: 2321-8169
20 - 24

A 2 Level Dynamic Scheduling Method for Real Time Tasks on Homogeneous Distributed System

Lipika Datta
Computer Science and Engineering Deparment, College of Engineering and Management Kolaghat
Purba Medinipur, W. B., India
lipika.datta@cemk.ac.in

Abstract - A distributed system is a collection of independent computers that appears to its users as a single coherent system. Depending on job arrival rate some of the computing nodes may become overloaded while some other nodes may sit idle. The imbalance in load distribution may affect overall system performance. So proper task scheduling algorithm is needed. Real time task scheduling algorithm must take deadline of a task in consideration. Task dependency is another parameter that is to be considered while scheduling. If arrival time of job is not known apriori, runtime scheduling is needed. The centralized scheduling schemes are not scalable as the scheduling decision is taken by a central server. Fully distributed schemes are scalable, but they lack global information. A hierarchical dynamic semi distributed scheduling model is proposed in this paper, taking deadline and dependency of each task in consideration.

Index Terms - *Task scheduling, Clustered homogeneous distributed system, dynamic scheduling, real time task, DAG.*

*****

I. Introduction

Fundamental properties of a task are: arrival time and approximate execution time. A real-time task also specifies a deadline by which it must complete its execution. The completion of a request after its deadline is considered of degraded value, and could even lead to a failure of the whole system. So, the real-time scheduler for distributed systems must consider the deadline of each task along with the load balancing issue while allocating tasks to processor.

Some real time tasks may execute on the regular basis. Those are called periodic tasks. Aperiodic real time tasks are activated randomly. Real time schedulers may be classified in two categories.

In static scheduling algorithm the assignment of tasks to processors is done before the program execution begins. Current allocation information is used to determine the availability of a processor by dynamic scheduler. So static scheduling is workable only if all the processes are effectively periodic. A distributed application may be represented by a Directed Acyclic Graph (DAG), in which the node weights represent task processing time and the edges represent data dependencies.

A fully distributed system suffers from huge message passing while taking a scheduling decision. This gives rise to huge traffic. A semi distributed system model is discussed in this paper. The nodes of the system are grouped in clusters. Each cluster contains a subset of nodes in the system. Each cluster is represented by a designated node called cluster master (CM). Local or global scheduling decision is taken by the CM. Based on the proposed model an algorithm is presented that is tested by conducting experiments in simulated environment.

The work is organized as follows: The status of the considered domain is presented in section II. Section III describes the system model and the responsibilities of each type of node. The 2-Level Dynamic Scheduling Policy (2LDP) policy is discussed in Section IV. Section V analyses the communication cost. Simulation results of experiments are presented in Section VI. Section VII includes the conclusion.

II. Related work done

The theoretical foundation to all modern scheduling algorithms for real-time systems was provided [1] for hard real-time tasks executing on a single processor. According to the authors that upper bound of processor utilization quickly drops to approximately 70% as the number of tasks increases. So, Liu and Layland suggested a new, deadline-driven scheduling algorithm, which assigns dynamic priorities to tasks according to their deadlines. Earliest Deadline First is too complex to be implemented in real-time operating system [2]. These algorithms were developed for uniprocessor system. They can be extended for centralized controlled distributed system, and then they would have to suffer from all the difficulties of centralized control. The RT-SADS [3] algorithm is designed for scheduling aperiodic, non-preemptable, independent, soft real-time tasks with deadlines on a set of identical processors with distributed memory architecture. RT-SADS self-adjusts the scheduling stage duration depending on processor load, task arrival rate and slack. Epoch scheduling [4] is a special type of scheduling for distributed processor. In this scheme at the end of an epoch, the scheduler recalculates the priority of each task in the queue using Shortest Task First (STF) criteria. LLF [5] assigns priorities depending on the laxity. In [6] authors present a novel list-based scheduling algorithm called Predict Earliest Finish Time (PEFT) for heterogeneous computing systems. In [7] authors have presented a modified dynamic critical path algorithm (CBL) to find the earliest possible start time and the latest possible finish time of a task using the distributed nodes network structure. Tasks are sorted by the ascending order of their loads and the processors are sorted by the descending order of their current loads in [8]. Tasks are assigned to processor to make the loads assigned to each processor balanced as much as possible. Ant colony based task scheduling for real time operating systems is proposed in [9]. In [10] the authors have assigned weight to nodes and edges of DAG to find the earliest finish time in HEFT algorithm. In [11] the DAG is clustered and then HAFT is applied for scheduling of tasks.

III. SYSTEM MODEL

In this paper a network is considered consisting of N homogeneous nodes P_1, P_2…, P_N connected by a communication network. Each node has the same computational power and local memory. A distributed application is represented as a DAG. The service time of the tasks are exponentially distributed with mean of $1/\mu$. Each node maintains a ready queue to store jobs which are assigned to the node, but yet to be executed. The jobs are assumed to be non-preemptive. Worker nodes schedule tasks in ready queue according to First In First Out (FIFO) algorithm. In the proposed model the whole system is divided into L clusters. Each cluster has a specific node designated as the Cluster Master (CM). This proposed model is semi-distributed and decentralizes the load balancing process. It is scalable and it minimizes communication overhead.

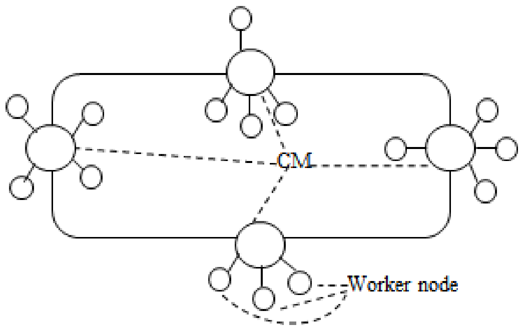

Fig. 1. Clustered representation of a distributed network

IV. 2 LEVEL DYNAMIC SCHEDULING POLICY

A DAG is submitted to the system. The algorithm makes an effort to assign a task to a computing node, after its predecessor completes its execution, such that the deadline of the task can be met. The scheduler first tries to assign a processor in its predecessor's home cluster to reduce communication cost. This is intra cluster scheduling. The CM searches for an idle node in the cluster. On failure, CM searches for a node whose remaining workload is less than the slack time of new task. If such a node is found, the task is assigned in that node. Otherwise, inter cluster task scheduling is required. The CM broadcasts a message containing the slack time to all other CMs. If the receiver CM is able to find a suitable node, it sends response to the initiating CM. The response message contains minimum load information of that cluster. The initiating CM selects the cluster with least minimum load. The new task is transferred to the selected CM. If no response is received, the task is assigned to the least loaded node in home cluster and it misses deadline.

A. Scheduler Algorithm

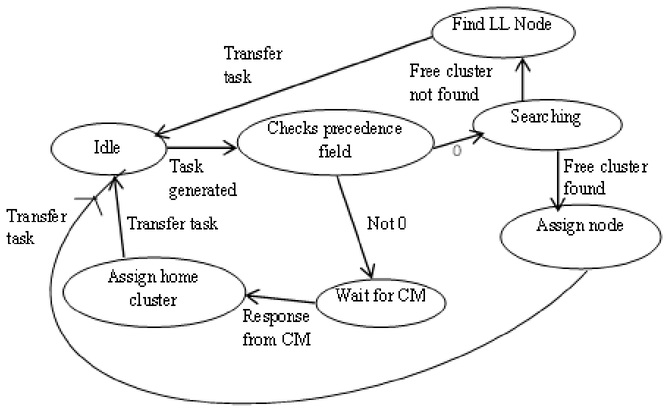

Fig. 2. Scheduler Process State Diagram

Initial state of scheduler process is idle. When a task is generated the scheduler checks its precedence. If precedence is 0 the task is independent. The scheduler searches for free cluster. If free cluster is found the independent task is assigned a processor in free cluster, otherwise it is assigned in the least loaded node in the system. For dependent task it finds the home cluster of the predecessor and assigns that cluster as the home cluster of the task. If the task is independent, after selection of node and for dependent task, after assigning home cluster, the scheduler state changes to idle.

Process scheduler

```
1.  Begin
2.  While TRUE
3.      Wait for task generation;
4.  If precedence is 0
5.          Search for free cluster
6.          If free cluster is found
7.              Assign a node
8.          Else
9.              Find least loaded node and assign it
10.     Else
11.     Sends query to CMs
12.     Receives response from the CM where the
        predecessor is assigned
13. Declares that CM as the home cluster
14. End
```

B. Cluster Master Algorithm

After receiving a query from scheduler process, CM checks whether the predecessor task is assigned to its cluster. If match is found it sends response to scheduler process. Upon task arrival or CM poll the CM sends query to its worker nodes.

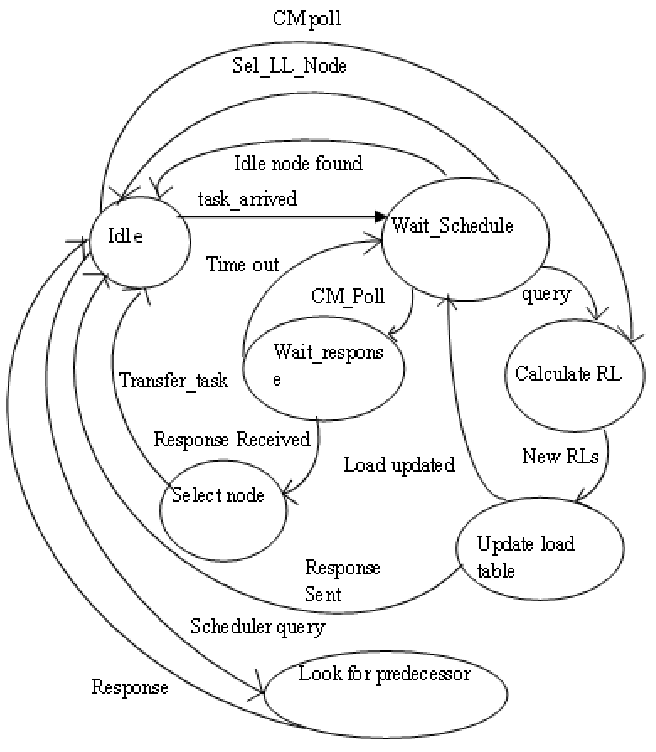

Fig. 3. CM state diagram

Each worker node calculates its remaining load (RL) and sends it to CM. Accordingly CM updates its load table. If it finds any idle node or a node with RL less than the slack time of the new task it assigns the node for the task. On failure it sends CM poll to other CMs mentioning the slack time. Other CMs update their load table and if they find any idle node or nodes with RL less than slack time, they send response with minimum RL information. If the initiating CM receives more than one response it selects the cluster with least RL and transfers the task to that cluster.

Process CM

```
1.  Begin
2.  While TRUE
3.  Wait for scheduler query;
4.      If match is found response is sent;
5.  Wait for Load_Receipt or CM_poll;
6.  Send Load_query to all worker nodes in its cluster;
7.      Receive RL from each worker node
8.      Update load table
9.      If CM_poll has occurred
10.         Search for suitable node.
11.         If suitable node is found
12.             Send response to initiating CM
13.  Else
14.      Search for idle node in home cluster
15.     If no idle node found
16.         search for a node whose load is less than
            slack time
17.         If such a node found
18.                 Send New_Load to selected
                node;
19.             Else
20.         Broadcast CM_Poll to all other CMs;
21.             If response is received before time out
22.                     Select the response with
            least RL;
23.                 Send New_Load to selected CM;
24.                 Else
25.                     Send New_Load to least
            loaded node in home cluster
26. End
```

V. ANALYSIS

Let us assume k, m and d be the upper bounds on the number of clusters, nodes in a cluster in the distributed system, and the diameter of a cluster respectively [12]. A node can communicate to its CM in maximum d steps. Two theorems are proposed based on time and number of messages required.

Theorem 1. The total time to assign a task is between (2d +k) T+L and (4d +2k)T +L where T is the average message transfer time between adjacent nodes and L is the actual average load transfer time.

Proof: Searching for free cluster or finding the least loaded node requires k hops. Sending query and receiving load from each node in home cluster requires 2d steps. So, total time to load transfer is (2d+k) T+L. If intra cluster task scheduling is not possible, CM polls to k-1 CMs and their responses require maximum k hops. Query and response in each cluster requires max 2d steps. So number of hops to load transfer is (2d + 2k + 2d) resulting in (4d + 2k) T + L time. If the diameter of cluster decreases this approach produces better result than [12] in inter cluster task scheduling.

Theorem 2. The total number of messages to assign a task is between (k+1+2m) to (2km+3k-1)

Proof: Searching for free cluster or least loaded node requires k+1 number of messages. Total number of messages in intra cluster scheduling is (k+1+2m). In inter cluster load balancing k-1 request messages are sent and at most k-1replies can be received. Each CM sends and receives replies from each worker node resulting (k -1)*2m messages. So, maximum number of messages is (k+1+2m + 2(k-1) + (k -1)*2m). This is much less than required in [13].

VI. RESULT

Experiments were conducted in simulated environment to evaluate the performance of the proposed 2LDP algorithm. The experiments were performed by varying several performance parameters in the system namely the number of worker nodes and the number of jobs. It is assumed that DAG nodes have single fan in and multiple fan out.

TABLE I
PARAMETER VALUES

Parameters	Values
Number of processors	10 ~ 50
Number of tasks	100 ~ 1000
Service time	Exponentially distributed with mean 20

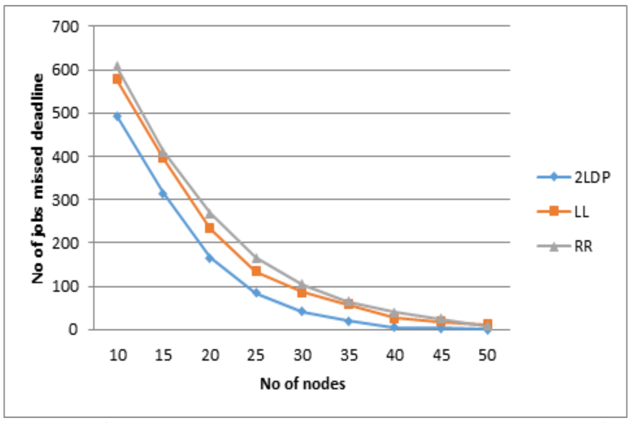

Fig. 4 No of tasks missed deadline among 1000 tasks vs varying no of nodes

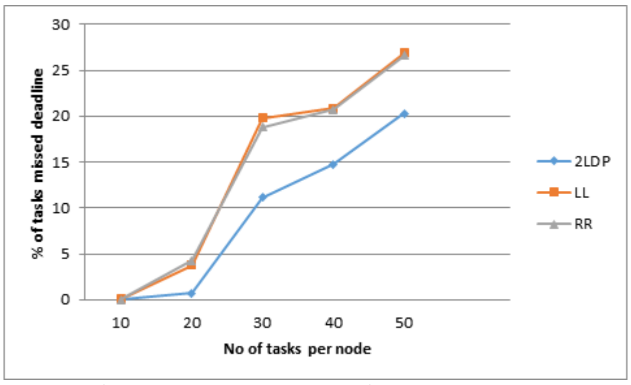

Fig. 5. % of tasks missed deadline vs no of tasks per node

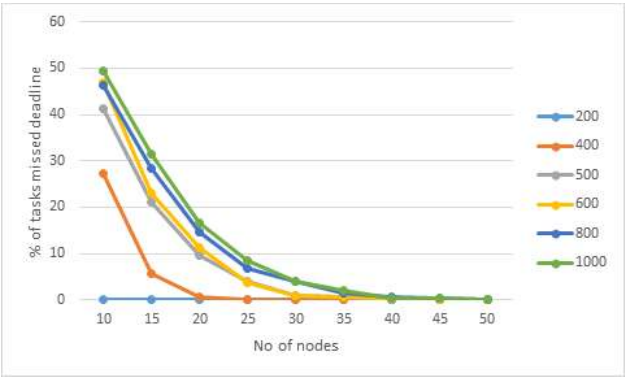

Fig. 6. % of tasks missed deadline vs varying no of nodes for varying no of tasks

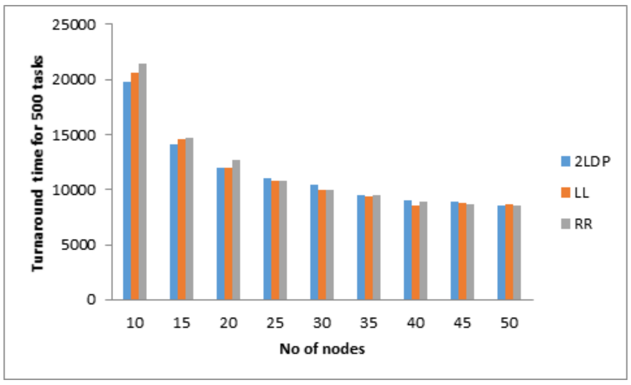

Fig. 7. Turnaround time for 500 tasks vs varying no of nodes

Figure 4 reveals that number of tasks missed deadline among 1000 tasks executed in varying number of nodes is less than the existing algorithms. From figure 5, it is observed that 2LDP algorithm reduces percentage of deadline missed for varying average no of tasks per node. From figure 6 it can be seen that with increase system load % of tasks missed deadline is not varying much. So with increase of system load the performance is not degraded. Figure 7 shows the turnaround time of an application with 500 tasks executed in varying no of nodes for 3 different algorithms. In LL algorithm for assigning each new task huge number of message passing is needed, which takes some time and incur cost. That time is not considered while calculating the TAT of each job here. Even though, it is observed that 2LDP's performance is the same as LL. From the graphs it can be inferred that if the nodes are heavily loaded, then the performance of the system is much better in terms of missing deadline than existing algorithms.

VII. CONCLUSION

In this paper a semi-distributed task scheduling method for real time tasks is proposed for clustered homogeneous distributed system. A distributed application is represented as a DAG. It is assumed that a task may have only one predecessor. Upon generation of new task a node is assigned for its execution considering the present load status of each node and the slack time of the new task. The task assignment method is scalable and has low message and time complexities. The method of partitioning the system into clusters and the method of load transfer are not addressed. Cluster Master may fail and due to their important functionality in the proposed model, new masters should be elected. Also, a mechanism to exclude faulty nodes from a cluster and add a recovering or a new node to a cluster is needed. These procedures can be implemented using algorithms as in [14]. As a future work I will focus on heterogeneous system and tasks with more than one precedence relations. Scheduling algorithm of each worker node may also

REFERENCES

[1] Liu C.L. and Layland J.W., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", in *Journal of the ACM*, vol. 20, no. 1, 1973, pp. 46-61

[2] Xin Li, Zhiping Jia, Li Ma, Ruihua Zhang and H. Wang, "Earliest deadline scheduling for continuous queries over data streams", International Conferences on Embedded Software and Systems, 2009, DOI**:** 10.1109/ICESS.2009.14

[3] Atif. Y and Hamidzadeh B , "A Scalable Scheduling Algorithm for Real-Time Distributed Systems ", in *Distributed Computing Systems,* 1998, pp. 352-359, DOI**:** 10.1109/ICDCS.1998.679738

[4] Karatza H.D. and Hilzer R.C., "Epoch Load Sharing in a Network of Workstations", *IEEE Simulation Symposium, Proceedings. 34th Annual*, 2001, 36-42

[5] Mok. A. and Dertouzos M., "Multiprocessor scheduling in a hard real-time environment", *7th Texas Conference on Computing Systems*, pp: 5.1-5.12, November 1978

[6] Arabnejad. H. and Barbosa J.G. , "List scheduling algorithm for heterogeneous systems by an optimistic cost table", 2014, IEEE Transactions on Parallel and Distributed Systems, Vol 25, Issue 3

[7] Zeng. B., Wei. J. and Liu. H., "Research of Optimal Task Scheduling for Distributed Real-time Embedded Systems", in *The 2008 International Conference on Embedded Software and Systems (ICESS2008),* pp 78 – 84, DOI 10.1109/ICESS.2008.29

[8] Zhang. K., Qi. B., Jiang. Q. and Tang. L., "Real-time periodic task scheduling considering load-balance in multiprocessor environment", 3rd IEEE International Conference on Network Infrastructure and Digital Content, 2012, DOI: 10.1109/ICNIDC.2012.6418753

[9] Shah. A. and Kotecha. K., "Scheduling Algorithm for Real-Time Operating Systems using ACO", *IEEE International Conference on Computational Intelligence and Communication Networks*, 2010, DOI: 10.1109/CICN.2010.122

[10] Zhao Rizos Henan and Sakellariou Rizos., "An investigation into rank function of the heterogeneous earliest finish time (HEFT) algorithm", University of Manchester, UK: Department of Computer Science, 2003

[11] [12] Fatma Omara and Doaa M. Abdelkader, "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system", 2012, *Egyptian Informatics Journal*, vol. 13, no. 2, pp. 135-145

[12] Erciyes. K. and Payli R. U., "A Cluster-Based Dynamic Load Balancing Middleware Protocol for Grids", in *Advances in Grid Computing - EGC*, LNCS 3470, Springer-Verlag, Berlin, 2005,PP 805-812

[13] Chatterjee M. and Setua, S.K., "A new clustered load balancing approach for distributed systems", *IEEE Conference on Computer, Communication, Control and Information Technology,* 2015, DOI: 10.1109/3CIT.2015.7060188

[14] Tunali, T., Erciyes, K. and Soysert, Z., "A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System", in *Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems*, 2(1), 2000, pp: 33-44